ADD-ON BOARDS

This article describes the relationship between motherboards and add-on boards (daughterboards) utilizing the most popular computer buses such as the industry standard architecture (ISA), Versa Module Europe bus (VMEbus), and peripheral component interconnect (PCI) and will detail the practical aspects of interfacing one board to the other. In particular, topics such as physical dimensions, printed circuit board (PCB) construction, basic interface design, and software drivers, including plug and play, will be covered for personal computers based on the Intel 80*x*86 and VME-based systems families of microprocessors. There are other commercial computer systems, such as Apple Macintosh/PowerPC systems, Sun Microsystems, and Silicon Graphics systems, which incorporate add-on boards but these are not covered explicitly in this article.

All computer systems require a motherboard to contain the primary electronic circuits; and, depending on the system, electronic subsystems or add-on boards may be physically attached to it through suitable interfaces. Add-on boards are generally used to provide one or more of the following features:

- Additional functionality not contained on the motherboard
- A means of upgrading the system
- Modularity and flexibility
- Proprietary interfaces for peripheral equipment
- Prototype development

For computer systems of greater complexity than a single-board computer (e.g., personal computers), add-on boards are essential. As well as providing end users with the choice of specifying their own configuration and tailoring a system to suit their needs, they also allow computer manufacturers the freedom to develop motherboard designs without being constrained by peripheral circuits more commonly found on add-on boards. Due to recent advancements in PCB manufacturing processes and higher levels of IC integration, circuits that used to be supplied as add-on boards for key peripheral subsystems such as monitors, hard disks, and main memory have now migrated to, and have become integral parts of, modern motherboards.

BASIC DAUGHTERBOARD CHARACTERISTICS

Daughterboards are connected to motherboards using either integral edge connectors or board-mounted connectors such as Deutsche Industrie Norm (DIN) and multipin. Edge connectors have found favor with manufacturers of personal computers because they provide a cost-effective and easy method of connection. These can be found in IBM-compatible PC-ATs and IBM PS/2 series personal computers employing ISA, PCI, or microchannel architecture (MCA) buses. However, due to the relatively poor electrical and mechanical reliability, repeated insertions are not recommended because a poor contact on a vital bus signal may render the add-on board or computer unusable. One effective solution to this problem is to install an add-on board to expand the bus. This allows work to be done externally and hence does not compromise the useful life of a motherboard's bus connectors. Figure 1 shows a typical PC-AT add-on board being employed as a means of extending an ISA bus.

Industrial and specialized applications favour the use of add-on boards with board-mounted connectors in standard 19-inch rack enclosures. These are generally more expensive than their personal computer equivalent, but provide greater reliability and flexibility for a developer. A back plane containing the system buses and bus connectors is usually housed at the rear of an enclosure and fixed vertically. Add-on boards (one of which has to be a motherboard or master) are then mounted into this using guide slots located at the top and bottom. Figure 2 provides an example of a 68000-

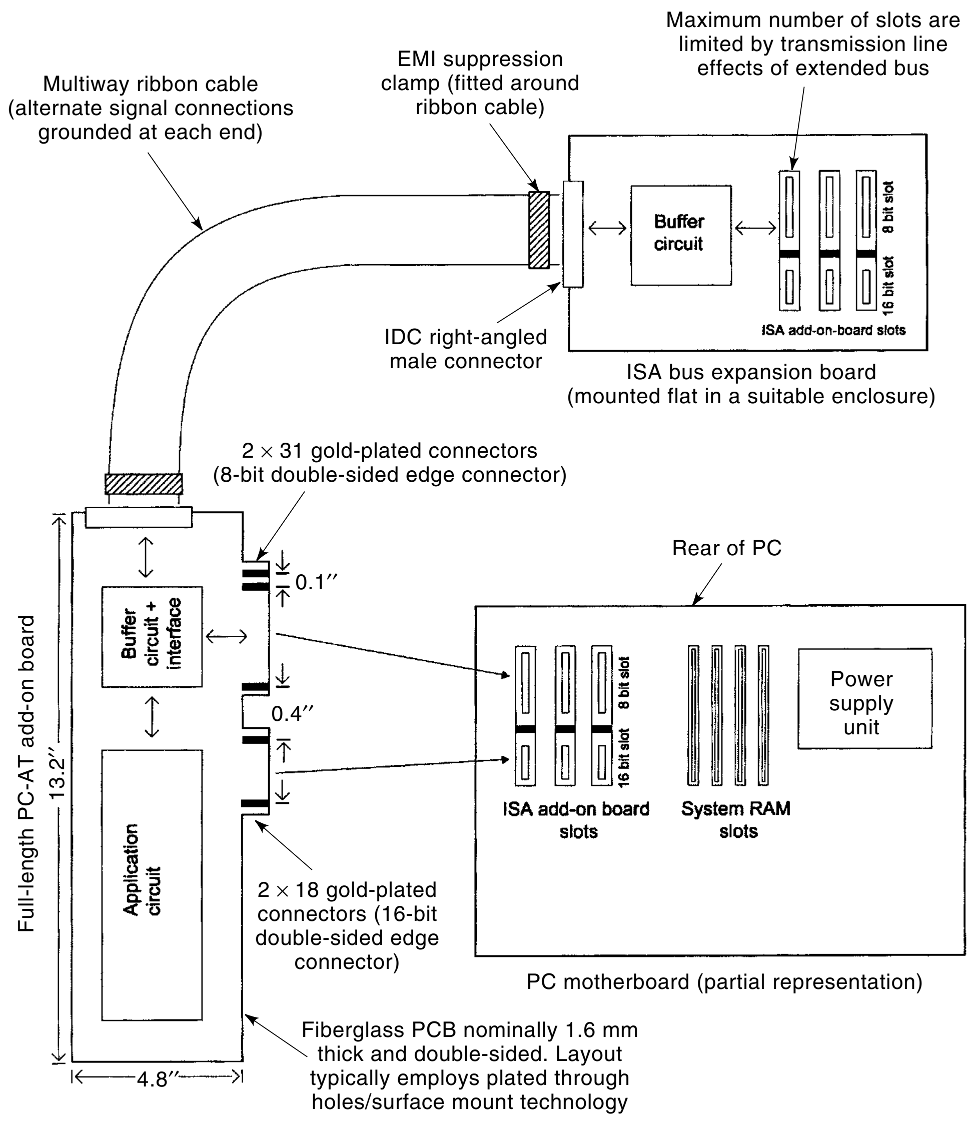

Maximum number of slots are limited by transmission line effects of extended bus
EMI suppression clamp (fitted around ribbon cable)
Multiway ribbon cable (alternate signal connections grounded at each end)
Buffer circuit
8 bit slot
16 bit slot
ISA add-on-board slots
IDC right-angled male connector
ISA bus expansion board (mounted flat in a suitable enclosure)
2 × 31 gold-plated connectors (8-bit double-sided edge connector)
Rear of PC
0.1″
0.4″
Buffer circuit + interface
Application circuit
Full-length PC-AT add-on board
13.2″
4.8″
Power supply unit
ISA add-on board slots
System RAM slots
2 × 18 gold-plated connectors (16-bit double-sided edge connector)
PC motherboard (partial representation)
Fiberglass PCB nominally 1.6 mm thick and double-sided. Layout typically employs plated through holes/surface mount technology

Figure 1. Extending an ISA bus as shown using a PC-AT add-on board increases the maximum number of add-on board slots beyond the number set by the motherboard and avoids the problem of repeatedly inserting boards into the motherboard slots, which reduces reliability.

based system employing the VMEbus. Another example is provided by industrial PCs which use a passive backplane and add-on processor boards.

Add-on boards are available in a variety of sizes usually conforming to a set of standard dimensions. This ensures that good electrical contact and mechanical alignment can be achieved during installation. Boards can, however, be made to custom dimensions, provided that:

- Bus connectors are mated to present the required bus signals
- The thickness of a board does not exceed the width of guide slots or impede movement
- A board fits into an enclosure without fouling other components
- Air circulation around other boards is not disrupted or restricted

Several types of PCB construction are employed for off-the-shelf boards ranging from plated through-hole (PTH) boards suitable for prototyping to complete systems based on a combination of surface mount and multilayered technology. Factors which determine and may limit the choice of PCB construction techniques can generally be listed as follows:

- Board space requirements for interface logic, application circuitry, and connectors
- Component package styles
- Complexity of interface and application circuits
- Mechanical strength
- Enclosure airflow
- Manufacturing and assembly costs

During the initial development of an add-on board, the use of readily available prototype boards may be considered if a fast route to evaluating a system at low cost is required. For example, using a half-length board supplied with a PTH matrix for application circuits, a basic 8-bit I/O mapped interface for the PC-AT bus can be constructed in a single day, em-

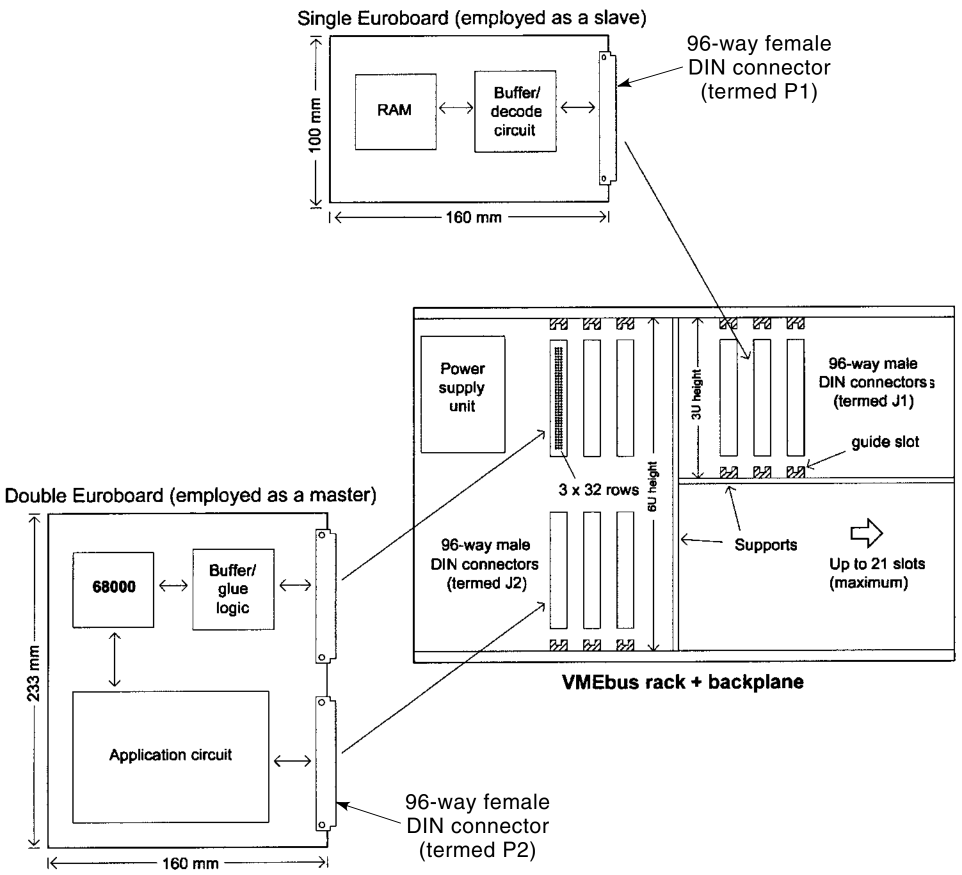

Single Euroboard (employed as a slave)
RAM
Buffer/ decode circuit
100 mm
160 mm
96-way female DIN connector (termed P1)
Power supply unit
3 x 32 rows
96-way male DIN connectors (termed J2)
3U height
6U height
96-way male DIN connectors (termed J1)
guide slot
Supports
Up to 21 slots (maximum)
VMEbus rack + backplane
Double Euroboard (employed as a master)
68000
Buffer/ glue logic
Application circuit
233 mm
160 mm
96-way female DIN connector (termed P2)

Figure 2. Schematic diagram illustrating master and slave board connections to a VME bus backplane. In this example, the system is extended by the addition of a board containing additional RAM.

ploying either standard wire-wrapping, speedwire, or soldered connections to interconnect components. If the development of an add-on board needs to progress beyond the prototype stage without incurring the costs normally associated with PCB manufacture, then direct computer-aided design (CAD) to PCB production using a computer-controlled tool may be considered. This allows a PCB output file to drive a milling/cutting machine and work a copper-clad board to produce tracks, pads, and other desired features without the need for any etching and the use of chemicals.

Communication between a computer and an add-on board is achieved by implementing an interface circuit close to the bus connectors of the add-on board. This is necessary for several important reasons:

1. To protect the integrity of the host's bus by buffering all bus signal lines utilized.
2. To allow individual boards to be selected for data transfers.
3. To enable data transfers by decoding bus control and address signals.
4. To generate bus control signals for the purpose of (a) providing the means for an application circuit to communicate with a host computer (e.g., interrupts) or with other boards by becoming a bus master and (b) enabling the features of a bus which may be necessary for the proper operation of an application—for example, extending bus cycles to allow data transfers between fast host memory and slow local memory.
5. To meet the timing requirements of a bus protocol.
6. To provide advanced features such as auto configuration defined by the plug and play standard.

A prerequisite for all add-on boards is to comply with point 1; and although requirements vary between computer buses, they can usually be met by employing off-the-shelf buffer ICs such as the 74LS244 and the 74LS245. Incorporating buffers will ensure not only that bus signals are protected by diodes but also that they drive just a single load on the add-on board. Bus signals generated by an application circuit will also be buffered from the bus and sourced with high drive for commoned bus signals.

The simplest and most commonly used method of achieving point 2 is known as address decoding. This is typically implemented using random logic to generate select signals from predefined address bus signals. These are then used as the primary control for points 3 and 4.

For most add-on board applications the use of discrete medium scale integration (MSI) chips in the interface circuit will be sufficient to satisfy point 5. However, interface circuits designed for high-specification buses such as Futurebus+ and PCI must be implemented using either special-purpose off-the-shelf ICs or user-programmable components such as complex programmable logic devices (CPLD) and field programmable gate arrays (FPGA) that exhibit very short pin-to-pin delays. Point 6 will be covered in a later section of this article.

DAUGHTERBOARD INTERFACES

Currently a multitude of proprietary and platform-independent computer buses are available for add-on boards, ranging from those designed to cater for applications requiring single byte input/output (I/O) operations at slow data transfer rates to those demanding the greatest performance such as 3-D image tracking. Figure 3 provides a comprehensive, although not exhaustive, list of popular buses together with some of their important features.

To implement an add-on board successfully, key design criteria need to be considered to ensure that the most appropriate bus is selected for an application. This, together with a good working knowledge of buses and how to interface hardware to them, will have a direct bearing on the cost, complexity, reliability, and performance of an add-on board. The essential design criteria to consider are:

- Data width (8, 16, 32, 64 bits or scaleable)
- Platform-independent or proprietary bus
- Data transfer; I/O mapped, memory mapped, or direct memory access (DMA)
- Board address: Dual in-line (DIL) switch selectable or auto configured
- Power requirements
- Interface, application, and connector area requirements
- Data throughput
- PCB mechanical strength and bus connector reliability
- Development tools
- Design effort and the consequent time to market delay
- System cost

By way of example, the three computer buses enjoying the most widespread use—namely ISA, VMEbus, and PCI—will be discussed in the following sections to a level of detail that will provide a basic understanding of simple interface design, from which practical add-on board applications may be constructed.

ISA Boards

In 1981 IBM introduced the first personal computer (PC) which allowed add-on cards to be connected via an 8-bit version of the proprietary ISA bus. This bus in an enlarged form is still available on the majority of PCs and provides the most readily accessible and straightforward means for a beginner or professional engineer to construct an add-on board for a PC with little or no prior knowledge of interfacing. The remainder of this section will be devoted to outlining the essential features of the ISA bus and some of the practicalities involved in interfacing an add-on board to it. A considerably fuller description of these topics can be found in Ref. 1.

An ISA bus slot found in PC-AT machines comprises up to 98 pins: 24 are for addresses, 16 are for data, 12 are for control, and the remainder are for interrupts, DMA, system, and power. The ISA bus employs synchronous bus cycles with a bus clock period of nominally 120 ns. Data are transferred on the data bus between a host and an add-on board using a port address which is uniquely associated with that add-on board to avoid conflicts with other add-on boards permanently connected to the same bus. For IBM compatible PCs the upper 512-byte region of the first 1 kbyte of system memory is allocated for standard add-on board (I/O) port addresses. This corresponds to addresses with hexadecimal values within the range 200h to 400h. However, some of this port address space will usually have been allocated by the system to essential peripheral hardware such as floppy diskette controllers, and in practice, end users are expected to choose a port address for their prototype add-on boards within the more restricted range of 300h to 3FFh. This method of enabling data to be transferred between an add-on card and the host computer is referred to as I/O mapping since the software accesses the card via an address in the system's I/O space rather than in the system's memory space.

By way of explaining the process in more detail, the design of a simple 8-bit I/O mapped interface capable of reading data on the bus and writing data back to it will be described in a step-by-step manner. The implementation of this example board will require four essential items:

- A port address to access the board
- Suitable driver software
- Data buffer logic
- Address decode logic

Choosing an appropriate port address may be affected by the presence of other add-on boards in the system, but it will be assumed that 300h is a suitable value in what follows. Creating the software needed to access the board via this I/O port and transfer data via the data bus is quite straightforward as

Bus Type	Full Name	Maximum Data Transfer Rate (Mbytes/s)	Data Widths (Bits)	Connector Type	IEEE Standard
ISA	Industry Standard Architecture	8.33	8,16	Edge	—
EISA	Extended ISA	33	8,16,32	Edge	—
MultibusII	—	48	8,16,24,32	DIN	1296
VMEbus	Versa Module Europe bus	57	8,16,32	DIN	1014
64-bit VMEbus	—	80	8,16,32, 64	DIN	1014
MCA	Micro Channel Architecture	160	8,16,32,(64)	Edge	—
PCI	Peripheral Component Interconnect	528	8,16,32,64	Edge	—
Futurebus+	—	>528	32,64,128,256	Multipin	896

Figure 3. A short list of important buses comparing their major characteristics.

illustrated by the following block of BASIC source code which would enable the host to execute a write command followed by a read command.

```
/*QBASIC listing*/
10 baseadd%=&H300'300 hex assigned to baseadd
for port address'
20 INPUT "8-bit data for Output="; dataout%
'user inputs dataout as data byte to be
written'
30 OUT baseadd%, dataout% 'data byte written to
port address'
40 datain%=INP(baseadd%) 'data byte to be read
assigned as datain'
50 PRINT "Input Data=";datain%   'view data
byte read from port address'
```

The same set of commands transposed into C source code would take the form

```
/*Borland TurboC v2.0 listing*/
#include ⟨dos.h⟩
#include ⟨conio.h⟩
#include ⟨stdio.h⟩
#define PORTID 0×300 /* 300 hex assigned to
PORTID for port address*/
void main(void)

{

int inbyte, outbyte; /*Define input and output
bytes*/
scanf("%x", &outbyte);      /*Enter output byte
in hex*/
outportb (PORTID, outbyte); /*Send byte to port
address 300 hex*/
inbyte=inportb(PORTID)      /*Read byte from
port address 300 hex*/
printf("%X hex",inbyte);   /*Print byte read
from port in hex*/}
```

Once a software driver is capable of single-byte I/O, routines can easily be developed to provide block transfers. These are particularly useful for video and file transfer applications as in frame grabber cards and local area network (LAN) cards, respectively. Device drivers intended for use with operating systems, such as Windows NT and UNIX, will be much more complicated than the examples just given.

To protect system bus logic, especially if a local bus is employed, buffer ICs are required between the system bus and an application circuit. The 74LS244 (an octal buffer with Schmitt triggers) and the 74LS245 (a bidirectional buffer with tristate capabilities) are commonly used to achieve this necessary level of isolation. Buffers are also vital to provide sufficient drive capability because bus pins should never be connected to more than two input pins on an add-on board. Buffers are selected by the board's address decode logic, sometimes supplemented by additional glue logic, to ensure that data transfers only occur when a valid port address is specified.

In order to effect a complete address decode, the decoder logic needs to examine the address lines A[9..2] of the ISA bus and check to see whether these values match the board address previously set on the card by the user to be the required I/O address (300h in this example). If a match is found and the address enable (AEN) bus signal is correctly set for I/O transfers as opposed to DMA transfers, a select signal is generated to indicate a valid I/O cycle. This process can be further understood by examining the system address bits A[9..0] and the most important control signals:

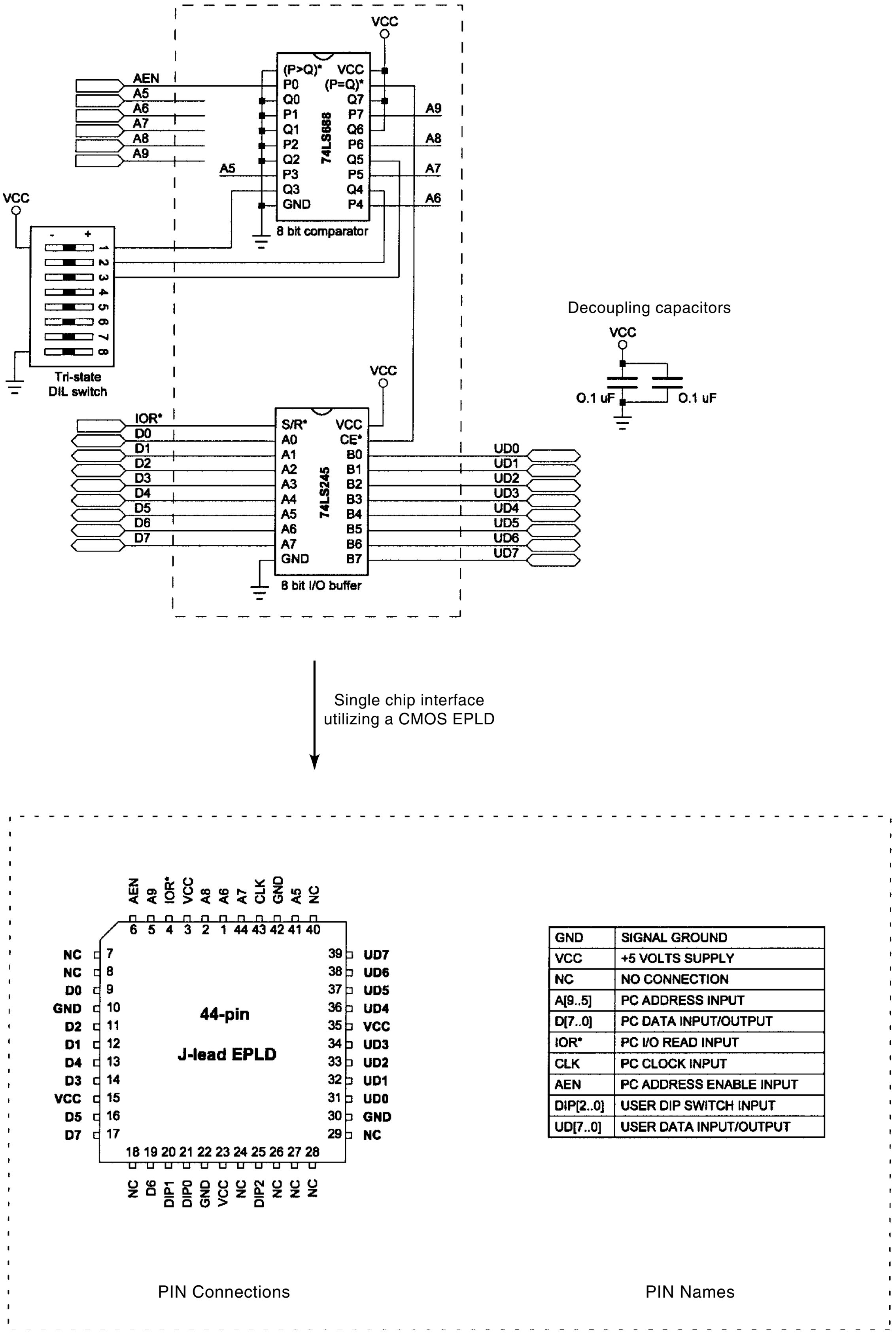
A9	Set to logic level '1' if the I/O address space is accessed
A8	Set to logic level '1' for prototype boards
A[7..1]	Used for offset addresses above the base address of 300h
A0	Generally not employed by basic decode circuits
ALE*	Active low address latch enable signal indicating start of an I/O cycle (not used in this example)
AEN	Active high signal indicating start of a DMA bus cycle
IOR*	Active low I/O read signal
IOW*	Active low I/O write signal

End-MultiList

For the sake of consistency, an * placed after a signal name will be used throughout this article to denote active low assertion. In the case of the current example, 300h involves setting A9 = A8 = 1, A[7..1] = 0 and A0 = 0. The values of these 10 address bits, together with AEN, are then compared using a suitable comparator (such as the 74LS688), with the board address bits, which are usually set by the user via a DIL switch. If a valid port address is decoded, an active low select signal is generated by the 688 and this is then used in combination with either the IOR* or IOW* signal and suitable glue logic to select a data buffer. In practice, completely decoding all 10 address bits is not necessary and less complicated logic circuits can be designed by ignoring some of the least significant address bits. An example of an efficient decode logic circuit comprising just two ICs for a basic 8-bit I/O mapped interface is illustrated in the top half of Fig. 4. This address decoder has treated bits A[4..0] as don't care signals which means that it would generate a select output for any address within the range 300h to 31Fh if A7, A6, and A5 were all set to 0 on the DIL switch. In the event of another add-on board having its port address set within this range, the DIL switch configuration would need to be changed appropriately. The bottom half of Fig. 4 indicates how this decoder could be realized as a single EPLD solution.

To understand fully the logical order and timing requirements of ISA bus transfers, reference is necessary to timing diagrams relevant to the type of data transfers of interest. Figure 5 shows a typical slightly simplified I/O write bus cycle which must encompass 6 clock periods. Reading data from an ISA bus is identical except that the IOR* bus signal would be employed rather than IOW*. For information on more complex transfers such as memory mapped I/O or DMA, as well as complete timing details, reference should be made to the official ISA bus specification which can be obtained from BCPR Services Inc.

It is essential for add-on board designers to ensure that their boards have no critical electrical faults which would damage the interface components on the motherboard before connecting the two. Although it is easy to test passively for short circuits, it is more difficult to be certain that there are



GND	SIGNAL GROUND
VCC	+5 VOLTS SUPPLY
NC	NO CONNECTION
A[9..5]	PC ADDRESS INPUT
D[7..0]	PC DATA INPUT/OUTPUT
IOR*	PC I/O READ INPUT
CLK	PC CLOCK INPUT
AEN	PC ADDRESS ENABLE INPUT
DIP[2..0]	USER DIP SWITCH INPUT
UD[7..0]	USER DATA INPUT/OUTPUT

Figure 4. Two separate implementations of a basic 8-bit I/O mapped ISA interface; the upper uses discrete 74-series components whereas the lower uses a single EPLD.

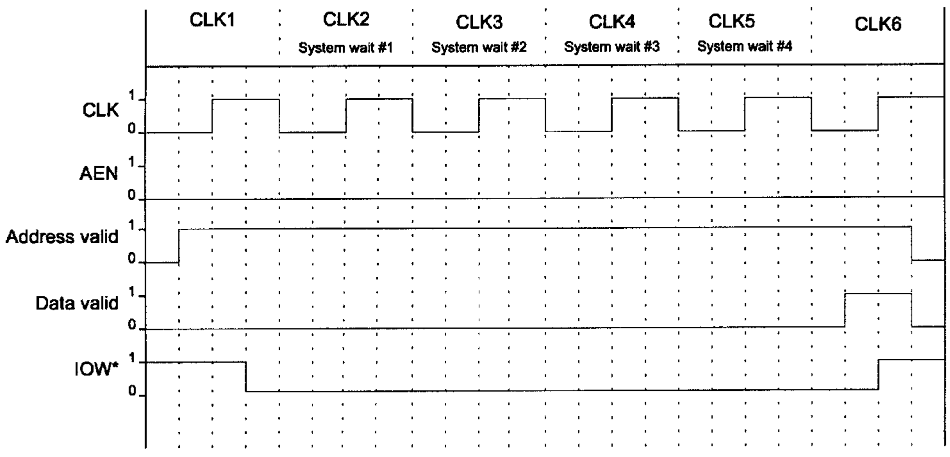


Figure 5. A schematic timing diagram illustrating an I/O write transfer across an ISA bus on a PC-AT computer. Only the relevant signals have been included and the number of wait states shown is appropriate to the minimum time required to effect the transfer.

no significant faults when power is applied to the board. One way around this problem is to connect the add-on board into a dynamic tester such as the one developed by the authors and described in Ref. 2. An alternative approach is to purchase from one of several manufacturers a relatively inexpensive ISA bus interface protection card with onboard buffers which will guarantee that the computer's motherboard is always adequately isolated and may provide in some cases useful diagnostic capability for detecting the origins of problems experienced with a malfunctioning add-on board.

VME Boards

The VMEbus specification was first released in 1982 by the VMEbus International Trade Association as a platform-independent bus, although it was originally based on the Motorola 68000 microprocessor series. It is commonly available in the form of a single or double bus backplane, typically housed in a standard 19 inch rack enclosure and optionally supplied with an integral power supply unit. The backplane usually consists of a specially fabricated multilayered PCB and comprises a set of partially terminated bus tracks, onto which either single or dual rows of male DIN connectors are mounted. Since the VMEbus by itself only provides the means to transfer data between boards plugged into it, electronic circuitry is required on each VME board to support the bus protocol. This is a distinct disadvantage when compared with the ISA bus, which provides bus controllers integrated on a motherboard. However, the VMEbus can support higher performance applications than ISA including full 32-bit multiprocessing, and for the most demanding applications a 64-bit version of the VMEbus is available.

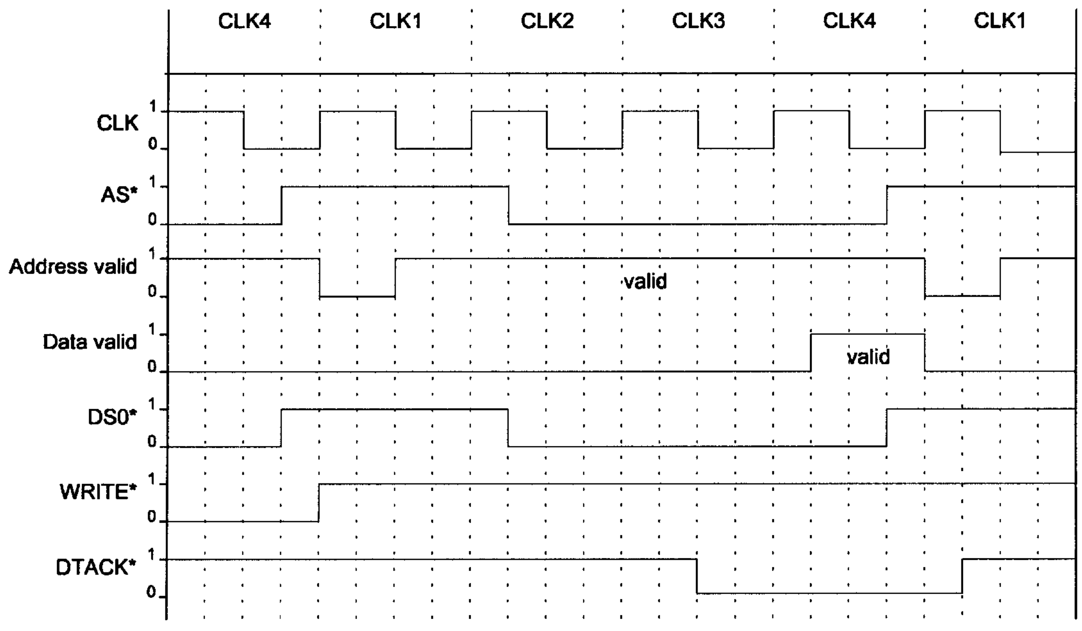
VME add-on boards are available either in standard single Euroboard (100 × 160 mm) or standard double Euroboard (233 × 160 mm), which allows room for one or two female DIN bus connectors, respectively. Boards are plugged into a VMEbus rack by sliding them along guide slots located at the top and bottom of the 19 inch rack, until they are mated properly with a backplane.

The VMEbus allows data transfers up to 32 bits wide, utilizing the data transfer bus (DTB), and supports memory mapped data transfer, DMA, and interrupts. Each add-on board or module is referred to as being either a master or a slave. A master is capable of initiating data transfers with any other type of module, whereas a slave can only respond to data transfer requests. Unlike the ISA bus, the VME bus does not have an I/O space, with the result that all peripheral devices must be mapped into the universal memory space. Another major difference is that the VME bus is asynchronous, which means that, after commencing a bus cycle, a bus master will not complete it until it receives an acknowledgment signal from the targeted slave device. To use the VME bus, the minimum requirement is that at least one master and one slave are present, such as a microprocessor-based module and a memory module, respectively. Other configurations are also possible—for instance, a single master and multiple slaves or multiple masters and slaves. Although the VMEbus is based on the 68000, no restriction is placed on the type of processor employed by a master module. The DTB bus is used to transfer data between installed modules and contains all the data, address, and control lines needed for this. Only the following subset of these signals is required for 8-bit data transfer.

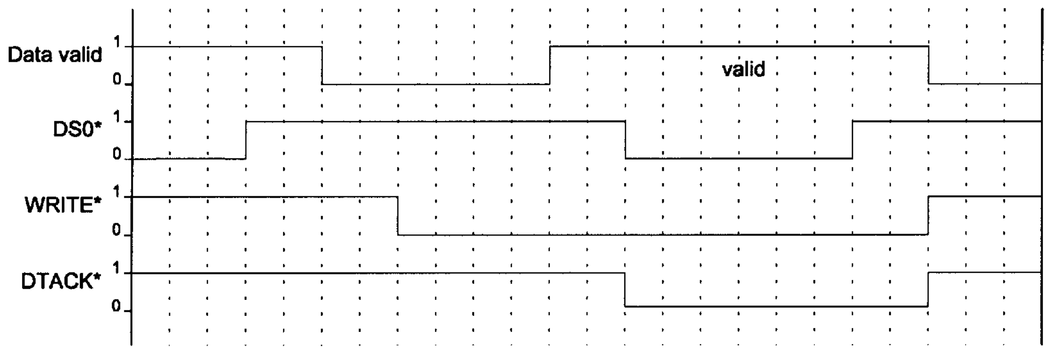
A[23..0]	Address bus
D[7..0]	Data bus
AS*	Active low address strobe
DS0*	Active low data strobe 0 (equivalent to LDS* for 68000 systems)
WRITE*	Read/write with logic levels '1' and '0' respectively (equivalent to R/W* for 68000 systems)
DTACK*	Active low data acknowledge

End-MultiList

The timing of these signals is shown in Fig. 6 for a single 8-bit read and a single 8-bit write. A bus cycle starts when the master places an address on the address bus and indicates when this is valid by asserting the AS* signal. The master also asserts one of the data strobes (DS0* in Fig. 6) to indicate which byte to transfer. In the case of a read cycle, the target has to recognize this address, place the corresponding data onto the data bus, and then assert the DTACK* signal. After recognizing the DTACK* signal, the master latches the data and completes the bus cycle by deasserting the address and data strobes. For detailed timing behavior, the



Read cycle for 8-bit I/O using DS0*



Write cycle for 8-bit I/O using DS0*
(only signal timings that differ from read cycle shown)

Figure 6. Schematic timing diagrams illustrating the reading and writing of 8-bit data across a VME bus. Only the relevant signals have been included and the timing of DTACK shown avoids the generation of extra wait states.

VMEbus specification should be consulted. This is available from the VMEbus International Trade Association.

By way of example the procedure for designing a general-purpose single Euroboard slave module will be covered for 8-bit bi-directional data transfers to a single master, such as a 68000-based module. The first consideration when designing a slave module is to decode the available address space, which for a 68000-based system is 16 Mbytes (24 bits). In a complete VMEbus system this would normally be partitioned to produce a memory map of all the available resources such as ROM, RAM, and input/output ports. In this particular example, however, the objective will be restricted to providing two (scaleable to eight) 8-bit data ports mapped to memory locations between 100000h and 100070h. Figure 7 shows how the VMEbus address bits would need to be assigned in order to achieve this. The values of the 17 most significant bits A[23..7] map the slave module's address, whereas the bits A[6..4] select up to eight single-byte ports. Bit A3 is ignored in this example and although bits A[2..0] are also not used specifically, they are available for additional features, such as the generation of general-purpose enable signals.

By combining the required address bits with AS*, an enable signal can be produced to form part of a general select signal for a data buffer, such as the 8-bit 74LS245 bi-directional buffer. Several methods are available for implementing an address decoder, but since a large number of address lines are commonly required for memory mapped I/O, a programmable device is highly recommended. In the case of the chosen example it would be necessary for the user simply to program this device to act as a relatively large comparator capable of generating two select signals for the ports at 100000h and 100010h. Figure 8 shows a circuit diagram for a suitable I/O decoder and buffer design employing five discrete ICs and one EPLD. The components used have been chosen

Address bits	A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12
Decimal equivalent	8M	4M	2M	1M	512K	256K	128K	64K	32K	16K	8K	4K
Binary value	0	0	0	1	0	0	0	0	0	0	0	0
Hex value	1				0				0			

Address bits	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Decimal equivalent	2K	1K	512	256	128	64	32	16	8	4	2	1
Binary value	0	0	0	0	0	#	#	#	0	#	#	#
Hex value	0				Y				Z			

Address bits	A6	A5	A4	Byte port Number	I/O address (hex)
Binary value	0	0	0	1	100000
	0	0	1	2	100010
	0	1	0	3	100020
	0	1	1	4	100030
	1	0	0	5	100040
	1	0	1	6	100050
	1	1	0	7	100060
	1	1	1	8	100070

Address bits	A2	A1	A0	Address offset (hex)	I/O address (hex)
Binary value	0	0	0	0	1000Y0
	0	0	1	1	1000Y1
	0	1	0	2	1000Y2
	0	1	1	3	1000Y3
	1	0	0	4	1000Y4
	1	0	1	5	1000Y5
	1	1	0	6	1000Y6
	1	1	1	7	1000Y7

Y = port number

Offset Z enables optional feature for selected port

Figure 7. An example of VMEbus address decoding for mapping single byte ports into memory. This maps the ports into the address range from 100000h to 100070h due to the values used in bits [A23 . . A7]. The values of the bits labeled Y and Z allow 64 distinct port locations within this 70h address range, as indicated in the lower two components of the figure.

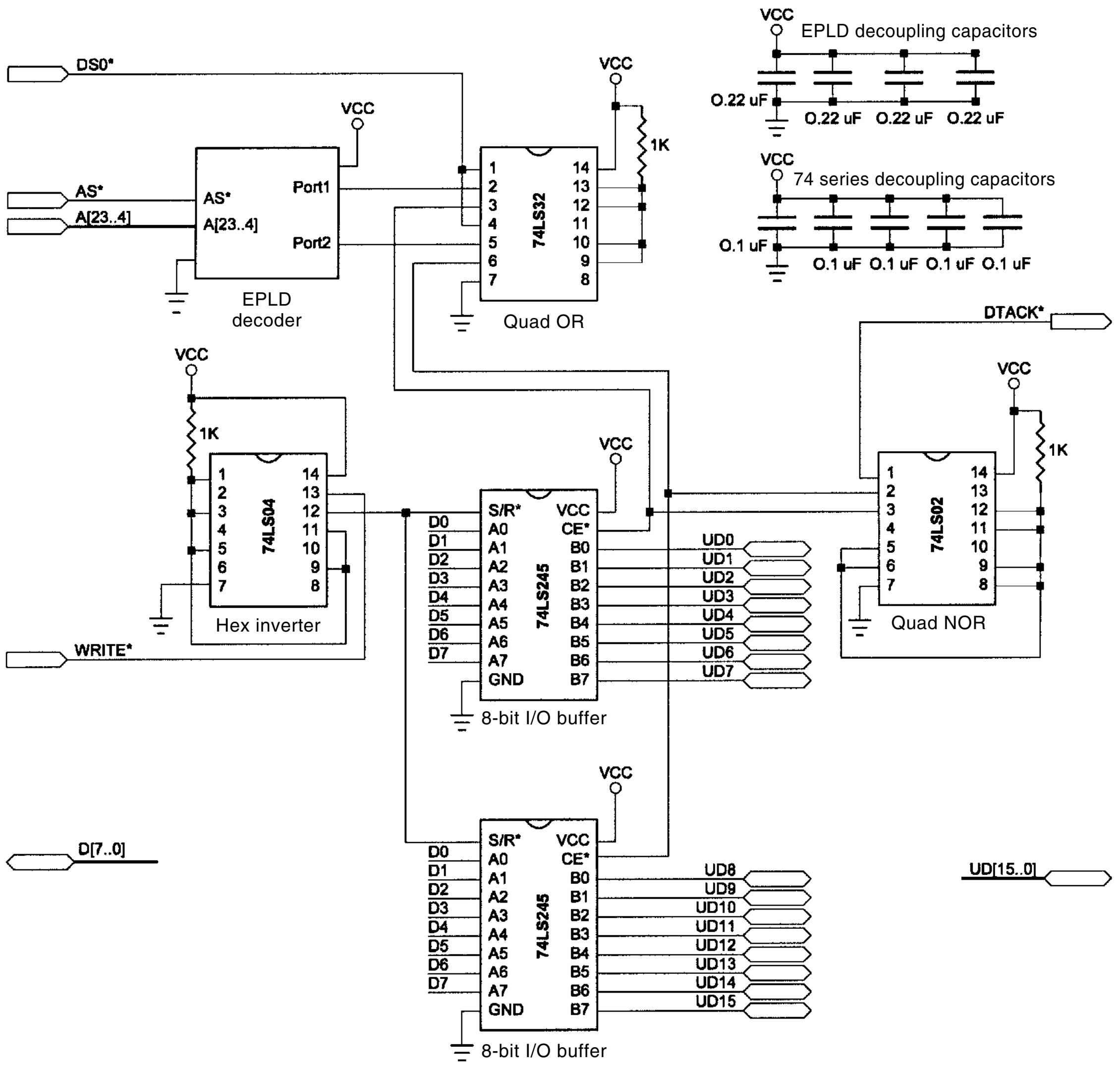


Figure 8. A basic 8-bit memory mapped circuit interfacing two byte-port addresses to the VMEbus. In this example the address decoding is achieved using an EPLD whereas DTACK generation and data buffering are implemented using standard 74-series components.

for their obvious function rather than to minimize the number of devices. In this circuit the active low-output signals from the EPLD are combined with the active low data strobe signal DS0*, to enable individually the two bi-directional data buffers. The direction of data transfer is determined by inverting the WRITE* bus signal and applying it to the S/R* pins of the buffers. The DTACK* signal is generated in this circuit as soon as the buffers are enabled, so no wait states are generated. If it were necessary to access a slow device, such as ROM, additional logic would be needed to extend the bus cycles by delaying DTACK* and thus ensure that the required setup and hold times were met. As with ISA bus interfacing an effective interface can be achieved using a single programmable component such as an EPLD or FPGA rather than several discrete components.

Readers who are interested in interfacing cards to VMEbus or Motorola 68000-based systems will find Refs. 3 and 4 to be useful sources of information. Reference 3 is a VMEbus user's handbook, while Ref. 4 contains explanatory material and many practical examples including circuit diagrams for interfacing memory and peripheral devices and for implementing interrupts and DMA operations.

PCI Boards

Add-on boards at the high end of the performance range use a local bus operating at speeds close to that of the native processor. In 1992 Intel Corporation created the peripheral component interconnect (PCI) specification, partly to prevent a proliferation of local bus designs and partly to address the longer-term needs of the computer market by defining a high-performance bus which is non-processor-specific. This is achieved on the motherboard using a bridge between the processor's local bus and memory bus and a PCI bus through which peripheral equipment such as hard disk drives, floppy diskette drives, local area networks, displays and add-on boards in general can gain access to the computer's resources. The latest revision of the PCI specification can be obtained from the PCI Special Interest Group. It should be appreciated that although the PCI bus is widely available on modern PCs, it is also found on other computers such as DEC workstations.

The most important features of the PCI bus affecting the design of add-on cards are as follows:

- The PCI bus allows a variety of bus masters and target boards to communicate with each other, and the host using synchronous-burst data transfers of a length which is negotiated between the initiator and target devices.
- Bus clock speeds from zero (lines held low) to 66 MHz and data widths up to 64 bits provide a maximum data transfer rate of 528 Mbytes per second.
- The PCI specification supports both 5 V and 3.3 V expansion cards using card edge connectors with keyed cutouts to prevent users plugging the wrong voltage card into one of the motherboard slots.
- Three standard card sizes—namely long, short, and variable-height short—are supported.
- Cards adhering to the PCI standard must contain a prescribed set of registers to hold information which facilitates automatic configuration at power up.

In order to minimize the number of physical lines required, PCI employs time-multiplexing of the address and data signals to create a 32-bit or 64-bit AD signal set. A maximum of 101 signals might need to be considered in the design of a 64-bit card, but many of these signals are optional. The minimum number of required signals for a 32-bit target card is 47, divided into three distinct groups, namely 37 address/data/command signals, 6 interface control signals, and 4 system signals. A 32-bit card containing bus mastering capability would need two additional system signals to handle bus arbitration issues. The signals which are optional divide into four groups, namely 39 address/data/control signals associated with the 64-bit extension, 5 JTAG boundary scan signals for in-circuit testing of the card, 4 interrupt request signals, and 4 miscellaneous signals. The latter include a lock signal for exclusive accesses during two or more data transactions, a clock control signal (intended for mobile rather than add-on cards), and two bus snooping signals. In addition to containing pins for all these signals, the PCI connector includes several power and ground pins together with two pins PRSNT1 and PRSNT2. One or both of these two pins must be connected to ground via a 10 nF high-speed capacitor on an add-on card in order to encode the card's maximum power requirement as 7.5 W, 15 W, or 25 W. A fuller description of the functional signal groups can be found in Ref. 5.

Since the add-on card designer needs to understand how data may be transferred across the PCI bus, the operation of burst, single, and configuration read/write transactions will now be briefly described. Figure 9 shows a schematic timing diagram for an optimized burst transaction involving four read transfers of data from a target card onto the bus with no wait states. All signals need to be stable on rising clock edges which are used to identify the start of numbered clock cycles. During the first clock cycle the initiator takes control of the bus by asserting the FRAME* signal and puts valid values on the address bus and the code for the required transaction type on the C/BE[3..0] control lines. During the second cycle, the

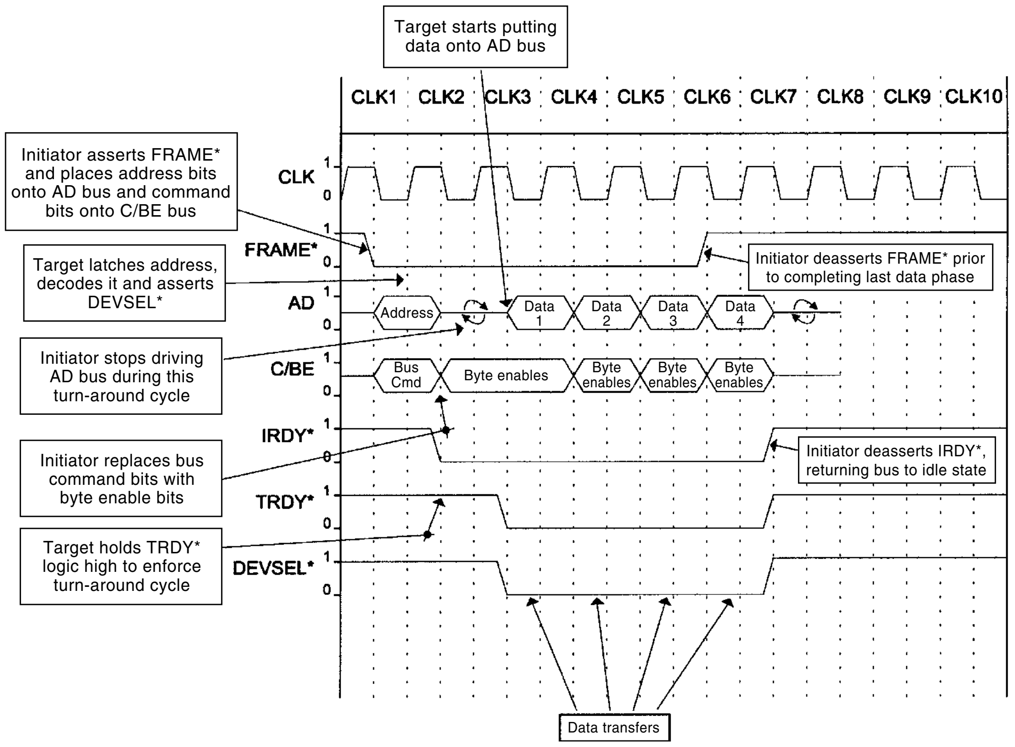

Target starts putting data onto AD bus
CLK1
CLK2
CLK3
CLK4
CLK5
CLK6
CLK7
CLK8
CLK9
CLK10
Initiator asserts FRAME* and places address bits onto AD bus and command bits onto C/BE bus
Target latches address, decodes it and asserts DEVSEL*
Initiator stops driving AD bus during this turn-around cycle
Initiator replaces bus command bits with byte enable bits
Target holds TRDY* logic high to enforce turn-around cycle
CLK
FRAME*
AD
C/BE
IRDY*
TRDY*
DEVSEL*
1
0
Address
Data 1
Data 2
Data 3
Data 4
Bus Cmd
Byte enables
Initiator deasserts FRAME* prior to completing last data phase
Initiator deasserts IRDY*, returning bus to idle state
Data transfers

Figure 9. A PCI bus timing diagram for an optimized burst of four read transfers. The turn-around cycle is required for read operations to provide time for the slave to replace address bits on the multiplexed AD bus. This is not necessary in write transfers because the master has control of both operations.

initiator indicates on the C/BE[3..0] control lines which individual bytes to transfer in the current double-word transaction and asserts the initiator ready IRDY* signal. During the third clock period, which is called a turn-around cycle, the addressed target device first asserts DEVSEL* to indicate that it has recognized its address, then asserts the target ready TRDY* signal (hence claiming the transaction) and, by the end of the cycle, places the first requested data item on the multiplexed address/data bus. The initiator responds by latching data on successive rising edges of the clock and de-asserts the FRAME* signal in the cycle before the transfer of the last data item. In the final clock cycle of importance the initiator de-asserts the remaining signals after which another bus master has an opportunity to gain control of the bus. During data transfer, a slow target card may introduce wait states by de-asserting the TRDY* signal until it is ready for the next data item. Write transfers are similar to read transfers except that the bus command type is different, the turn around cycle is not needed, and TRDY* and DEVSEL* are asserted simultaneously in the second clock period rather than the third.

Single transfers across the PCI bus are achieved in a similar manner to burst transfers except that every data item to be read is preceded by an address item and a turn-around cycle. Clearly, the rate of data transfer is drastically reduced in this mode and it is mainly used during configuration type transactions. A full description of all the various types of bus transfers may be found in Ref. 5.

All PCI add-on boards must contain registers holding configuration information. This so-called configuration space must be accessible at all times, but its principal use is during system initialization to configure the card for proper operation within the system. The first 64 bytes of configuration space is called the configuration header, and Fig. 10 shows how this is partitioned into regions containing fixed data—for example, vendor ID, device ID and class code, regions containing command and status registers, a region containing base address registers to indicate the memory, I/O and ROM space requirements of the card, and a region for interrupt requirements. A configuration access is achieved by asserting the IDSEL line during the address cycle. This acts like a device enable with the address bits A[10..8] selecting the device function while address bits A[7..2] select one of the 64 double-word registers of the complete configuration space to be read from or written to. In the case of an add-on card for a PC, each configuration transaction is a two-step process. A double-word specifying the nature of the transaction and the transaction address is first written by the PC to I/O space locations 0CF8h through 0CFBh, and the data are then transferred using I/O space locations 0CFCh through 0CFFh.

Although a PCI add-on card must be capable of operating at a minimum 33 MHz clock frequency, it is desirable that it should also operate correctly at slower speeds down to 0 Hz for debugging and power saving purposes. For conventional 33 MHz operation, the control signals have very stringent timing constraints—in particular a minimum 7 ns setup time on input bussed signals. The PCB traces on a PCI card are also critical. The PCI specification strongly recommends that the layout of the important shared signal pins on the PCI interface device should correspond closely with the PCB edge connector layout. One manufacturer of add-on board components recommends that trace lengths for data/address signals should be no more than 1.5 inches and 2 inches for other signals while the clock signal trace must be 2.5 inches ± 0.1 inches. For these reasons the design of a PCI card is intrinsically more difficult than the design of ISA or VME cards, and the use of MSI chips is not a viable solution. Some large companies manufacturing PCI add-on cards make their own full-custom PCI interface chips to designs tailored for their own specific applications. Other manufacturers utilize third-party PCI interface devices such as AMCC's S5933 PCI Matchmaker (6) and PLX Technology's PCI 9050 series of components (7). Programmable devices also enjoy wide usage for PCI interfacing because the card designer can program them to include only the application features necessary and may in addition be able to include some of the back-end application logic if there is space remaining on the device. Suppliers of suitable PCI compliant programmable devices include Altera

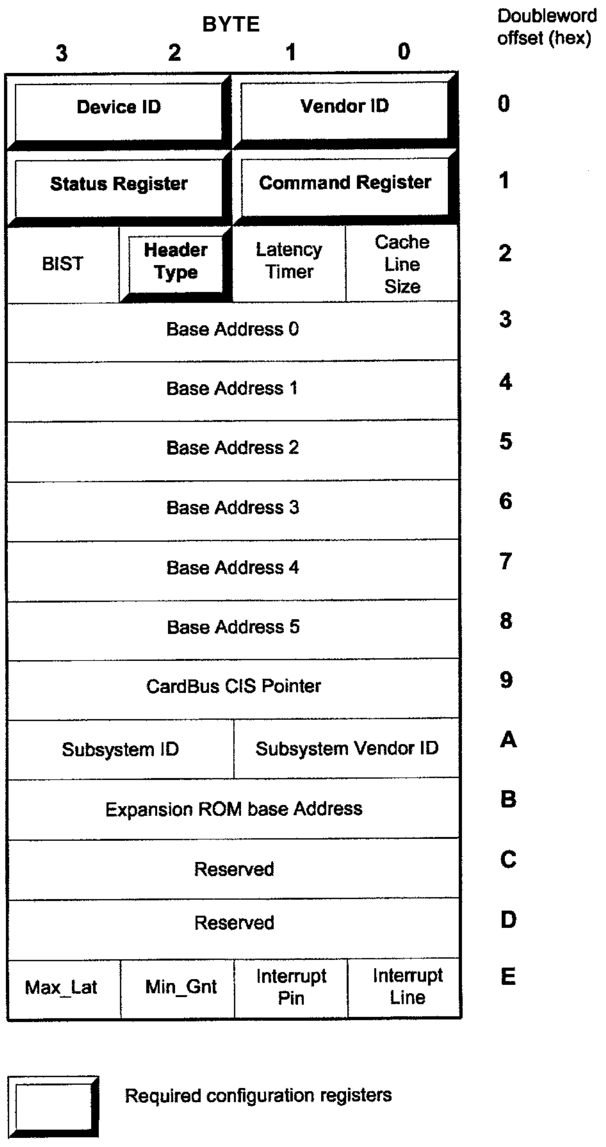


Figure 10. An illustration containing the configuration header registers for PCI add-on cards. Sufficient information can be stored to facilitate the design of autoconfigurable add-on-boards.

Corporation (8) and Xilinx Inc. (9), both of whom also supply useful design and application notes on request.

PCI Interfacing Examples. The production of a PCI compliant add-on card from scratch is a time-consuming task which can be speeded up in appropriate cases by the use of a PCI prototyping card. AMCC supply an evaluation board comprising a PCI Matchmaker interface chip, a nonvolatile RAM for holding configuration space information, a programmable memory, an ISA connector, and wire wrap areas to contain the user's back-end application. The ISA connector is a useful feature since it enables the designer to conveniently configure the PCI interface before the evaluation card is able to communicate via the PCI bus.

An example of an add-on card using programmable chips for interfacing is provided by the PCI "generic card" designed by Hauradou et al. (10). In this example a PCI interface is currently implemented on a fast Altera EPLD which communicates with a SRAM-based Altera CPLD containing the user's application. These authors suggest that their system could be advantageously applied to prototyping ASIC-oriented designs before going to large-scale manufacture. They are also proposing to develop PCI-compliant add-on boards for image filtering and cryptography, both of which would eventually combine the PCI interface function and back-end application in a single CPLD.

MULTICARD COMPATIBILITY

A typical computer system normally contains several add-on cards all requiring similar system resources such as I/O and memory space, interrupts, and DMA channels. The traditional means of ensuring mutual compatibility between the add-on components has been to provide each card with jumpers and DIL switch selectable storage requirements, interrupt lines, and DMA channels. The problem with this is that every time a new card is added to an existing system, the user is forced to manually configure it to avoid hardware resource conflicts with other cards. Clearly this approach is inconvenient, is prone to error, and requires users to possess detailed knowledge of the hardware components in their systems. In a perfect world, an ideal add-on-card would be expected on power-up to be configured automatically by software running on the system and to be compatible with all other components in the system. It was recognized that in order to achieve this ideal a standard was required, and this has come to be known as "plug-and-play" in the case of PC-based cards running under the Windows 95 operating system. PCMCIA cards are examples of cards which are automatically reconfigurable while power is applied to the system rather than just at switch-on. A Microsoft white paper describing the relationship between Windows 95 and PCMCIA can be downloaded from Compuserve's "Plug-Play" forum.

Generic Plug-and-Play Requirements

The plug-and-play card designer must implement the following features:

- A card detection mechanism to enable the system software to detect the presence of a card in a particular slot on the motherboard and to activate and deactivate individual cards at will
- A means for device vendor and device identification data and a list of resources needed by the card to be read by the system
- A set of configuration registers at standard locations which can be read from and written to by the system
- A device driver which can be loaded into or removed from memory if the associated card is added or removed while the system is running

The system designer must (a) provide system memory to store resource allocation information and configuration software to detect all cards, (b) read their resource requirements before allocating resources, and (c) write the appropriate configuration settings into each card's registers. After completing automatic configuration, the operating system software must identify and load only the device drivers associated with the currently installed cards and pass the configuration information of each card to its associated device driver. It is obvious that all these features must be implemented according to an agreed specification.

A few bus standards, notably PCI, provide all the necessary hardware features so that cards complying with these standards can be automatically configured by suitably designed motherboards and operating system software. Other bus standards do not incorporate all the necessary features, and these have to be added by the card designer in order to achieve full plug-and-play compatibility. In the case of PC architecture, ISA, EISA, and MCA buses all fall into this latter category. PC add-on cards designed after the advent of the Windows 95 operating system tend to have plug-and-play compatibility, whereas earlier ISA cards which cannot be automatically configured are consequently often called "legacy ISA cards." The EISA and MCA PC buses have most of the features required for plug-and-play compatibility but neither are able to implement a resource requirement list directly. Since this information is held on disk and must be loaded by the user, EISA and Microchannel are regarded as providing semiautomatic reconfiguration. Plug-and-play standards for ISA, EISA, and MCA add-on card designers all exist now, and the former will be described in the following subsection to illustrate how to implement the concept because all the required hardware features have to be explicitly added.

Design of ISA-Based Plug-and-Play Cards

Since the ISA bus provides no mechanism for isolating an individual card for configuration, the ISA plug-and-play specification demands that a particular sequence of steps must be carried out. A highly condensed overview of this procedure now follows. A special 32-byte sequence, generated by a linear feedback shift register and called the *initiation key,* is first sent to all cards to put them into a listening mode. This is a security feature to prevent any accidental access to a wrong location altering the configuration of a card. A special sequence of read commands sent to all cards, called a *wakeup call,* then causes them to arbitrate among themselves to choose one card to go into a state of isolation. The system configuration software then assigns a card select number (CSN) to this isolated card, reads its resource requirement list, and then causes it to enter the sleep mode. This process is repeated until all cards present have been isolated, processed, and put to sleep. The configuration software then uses the card selection number to wake up each card individually, assign nonconflicting resources to it, and activate it for normal operation.

In order to perform the necessary communications, plug-and-play ISA cards have three special 8-bit I/O ports. Two of these, namely the configuration address port and the configuration data port, are write-only and are implemented at the fixed addresses 0279h and 0A79h, respectively. The third, called the configuration read data port, is read-only and is implemented at an address ending in 11b located somewhere in the range 0203h through 03FFh (e.g., 0207h). The configuration software has to find an address in this range which does not select a legacy card and then tell all the plug-and-play cards the actual address of this port. The bottom 6 bits of a byte written to the address port access one of 64 eight-bit registers whose contents can be read via the read data port or overwritten via the write data port. These registers form the bottom quarter of the 256-byte configuration register space shown in Fig. 11.

The card wakeup call mentioned earlier in the overview consists of writing 03h to the address port and 00h to the write data port, following which all cards with nonassigned CSNs wakeup. Each card must contain in on-board memory a unique factory-generated 64-bit identification number plus an 8-bit checksum of these 64 bits which are used during the card arbitration process as follows. The cards all simultaneously examine the first bit of their own unique ID number. Cards having the bit value one put the number 55h in their isolation register at offset 01h followed on the next bus cycle by the number AAh. The numbers 55h and AAh are read by the configuration software and hence appear on the bus. Cards having zero in this bit position passively read the bus and go to sleep if the sequence of values 55h and AAh is observed. This process continues through the 72 bits leaving just one card awake. The isolated card then receives its unique CSN which is written into the register at offset 06h. A CSN of 00h is then written to the register with an address offset of 03h which causes the previously isolated card to go to sleep and all the others to wake up. The card isolation process is repeated until all cards have been allocated CSNs. During this isolation process the 8 bits of the checksum component provide a mechanism for the operating system to detect the presence of legacy cards through the bus contention they cause and to respond by trying a different read port address. When the operating system finds a suitable read port address, it writes this value to the register at 00h and uses it for all subsequent data reads. The checksum is also used to detect when there are no more plug-and-play cards to isolate.

Having assigned a unique CSN to every card, the operating system wakes up each card in turn and reads its resource requirement list. The resource data are read one byte at a time from the resource data register at offset 04h shown in Fig. 11. Each byte is made available from slow access nonvolatile memory and must not be read until it becomes valid as indicated by the status register at offset 05h. Once it has a complete picture of the total requirements of the add-on cards, the operating system analyzes the information to find a nonconflicting allocation which it then writes into every card's configuration register set located within the offset range 40h to FFh. Plug-and-play sets limits on each type of resource which an individual card may implement. These include up to four memory configuration registers numbered 0 through 3, up to eight I/O configuration registers numbered 0 through 7, up to two interrupt request lines (each with up to two associated configuration registers numbered 0 and 1), and up to two DMA channels (each with an associated configuration register numbered 0 and 1). Of course the card designer does not need to implement any of these resources if they are not required by the card's application.

Readers who need more information on the implementation of plug-and-play compatible cards should consult Ref. 11. This book contains a much fuller description of the material covered above as well as providing detailed information on plug-and-play BIOS and operating systems extensions.

Offset	Register name
00 h	Set read port address
01 h	Serial isolation
02 h	Configuration control
03 h	Wake command
04 h	Resource data
05 h	Status
06 h	Card select number (CSN)
07 h	Logical device number
08 h 1F h	Reserved card-level registers
20 h 2F h	Vendor-defined card-level registers
30 h	Activate
31 h	I/O range check
32 h 3F h	Reserved for logical device control
40 h 5F h	ISA memory configuration registers 0–3
60 h 6F h	I/O configuration registers 0–7
70 h 73 h	Interrupt configuration registers 0–1
74 h 75 h	DMA configuration registers 0–1
76 h A8 h	32-bit memory configuration registers 0–3
A9 h FF h	Reserved for logical device configuration

Figure 11. Plug-and-play add-on-boards based on the ISA bus must provide a sufficient subset of the registers shown to support the specific resources required by the board's application.

SUMMARY

In this article the authors have provided an overview of the relationship between computer systems and add-on boards. Since bus systems provide the lines of communication between an add-on board and a host processor, a considerable part of the article has been devoted to buses in common use. It is clear from the examples provided that there are many options for the add-on card designer, ranging from simple input/output interfaces to complex and high-speed video processors. Cards with undemanding requirements can be designed to interface to the simplest bus available and can be built and tested by anyone possessing quite limited design tools and hardware facilities. On the other hand, cards with very demanding requirements will need to take advantage of the capabilities afforded by one of the high-speed buses; for example, PCI and their design and construction will usually require rather sophisticated design tools and manufacturing facilities. Alternatively, when very high-speed data communication between two add-on boards is required, a direct board-to-board interface can be utilized. An example is provided by the feature connector found on many PC-based display cards and video-based application boards. Examples of proprietary board-to-board interfaces are DT connect from Data Translation and DSPlink from Loughborough Sound and Images.

The trend in add-on board design is toward cards which are programmable to a greater or lesser degree. This is evident in the emergence of cards which are automatically configurable and in cards whose functionality can be programmed to match different tasks.

BIBLIOGRAPHY

1. L. C. Eggebrecht, *Interfacing to the IBM Personal Computer,* 2nd ed., Carmel: SAMS, 1992.
2. S. S. Ipson, N. O. Van Haght, and W. Booth, A versatile stand-alone tester of PC expansion cards designed for undergraduate projects, *Int. J. Electr. Eng. Educ.,* **33**: 99–107, 1996.
3. S. Heath, *VMEbus User's Handbook,* Oxford: Heinemann Newnes, 1989.
4. A. Clements, *Microprocessor Systems Design,* 2nd ed., Boston: PWS-KENT, 1992.
5. T. Shanley and D. Anderson, *PCI System Architecture,* 3rd ed., New York: Addison-Wesley, 1995.
6. AMCC product information on the internet at http://www.amcc.com
7. PLX Technology product information on the internet at http://www.plxtech.com
8. Altera Corporation product information on the internet at http://www.altera.com
9. Xilinx Inc. product information on the internet at http://www.xilinx.com
10. S. Hauradou, T. Lejealle, S. Haezebrouck, O. Meullemeestre, and A. Galisson, The "PCI generic card": hardware reconfiguration using a FPGA-based PCI add-on board, http://www-elec.enst.fr/fiches/stages/hauradu/
11. T. Shanley, *Plug and Play System Architecture,* New York: Addison-Wesley, 1995.

S. S. Ipson
N. O. Van Haght
W. Booth
University of Bradford

ADDRESS INTERPRETATION FOR POSTAL SERVICES. See Postal services.

ADJUSTABLE FILTERS. See Programmable filters.

ADJUSTABLE SPEED DRIVES. See Induction motor drives.

ADVANCED INTELLIGENT NETWORKS. See Intelligent networks.

ADVANCED PROCESS CONTROL. See Semiconductor factory control and optimization.