(12) UK Patent Application (19) GB (11) 2 362 482 (13) A

(43) Date of A Publication 21.11.2001

(21) Application No 0011599.8

(22) Date of Filing 15.05.2000

(71) Applicant(s)
Ridgeway Systems & Software Ltd
(Incorporated in the United Kingdom)
66 Suttons Business Park, READING, Berkshire, RG6 1AZ, United Kingdom

(72) Inventor(s)
Nevil Morley Hunt
Malcolm Philip Ross

(74) Agent and/or Address for Service
Dummett Copp
25 The Square, Martlesham Heath, IPSWICH, Suffolk, IP5 3SL, United Kingdom

(51) INT CL[7]
G06F 12/10 9/35 12/02

(52) UK CL (Edition S)
G4A AFGDC AND

(56) Documents Cited
GB 2307569 A

(58) Field of Search
UK CL (Edition R) **G4A AFGDC AND**
INT CL[7] **G06F 9/35 12/00 12/02 12/04 12/06 12/08 12/10 13/00 13/10 13/12 13/14 13/16 13/28 13/36**
Online: EPODOC, WPI, JAPIO

(54) Abstract Title
Direct slave addressing to indirect slave addressing

(57) A computer bus system comprises: a direct address bus; at least one bus master device and at least one bus slave device, the bus master device and bus slave device being connected to the bus so that the bus master device may communicate with the bus slave device over the bus. The bus has an address space assigned to different devices connected to the bus and is a multiplexed address/data bus for transferring blocks of data (63,76) in a direct address transaction (60) between the devices. Each direct address transaction (60) comprises a burst transaction (61) having an address phase with a bus space address value (62) followed by a data phase (61). The bus slave device includes an indirect address device addressable in an indirect address transaction (70) that has an address register load transaction (71) followed by a data register load transaction (72). The indirect address device has a memory with memory locations identified by address values loaded into the address register of the indirect address device. The slave device includes a transaction translation device between the bus and the indirect address device that translates the direct address transaction (61) to an indirect address transaction (71,72) including mapping (64) the bus space address value (62) to the destination address value (74). Therefore, a direct address transaction (60) received by the slave device for communicated blocks of data is presented to the indirect address device as an indirect address transaction (71,72).

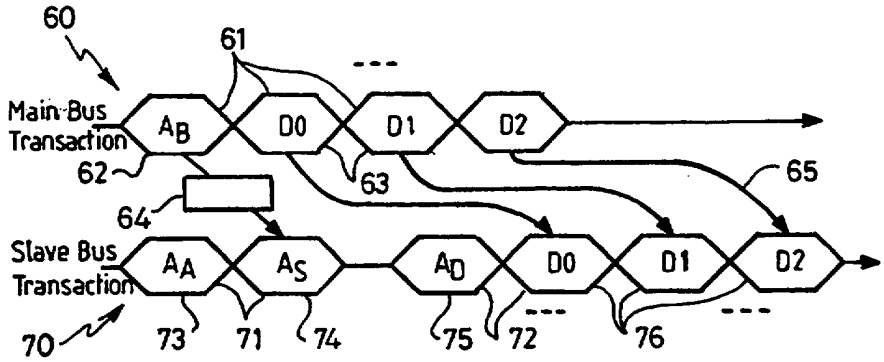

Fig. 6

At least one drawing originally filed was informal and the print reproduced here is taken from a later filed formal copy.

This print takes account of replacement documents submitted after the date of filing to enable the application to comply with the formal requirements of the Patents Rules 1995

GB 2 362 482 A

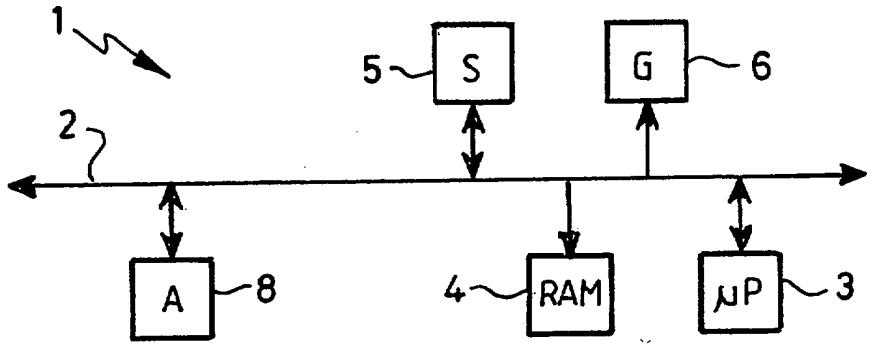

Fig. 1

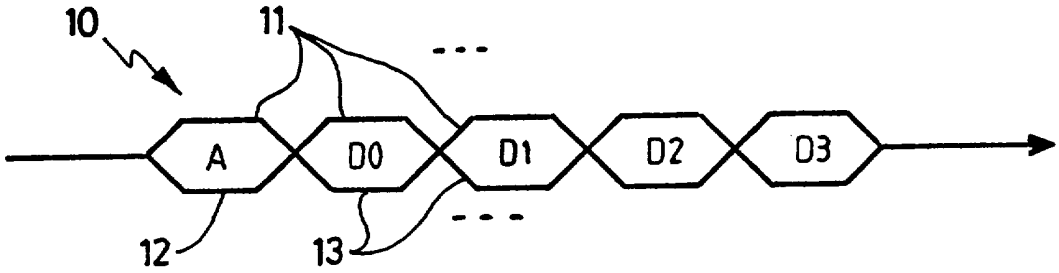

Fig . 2

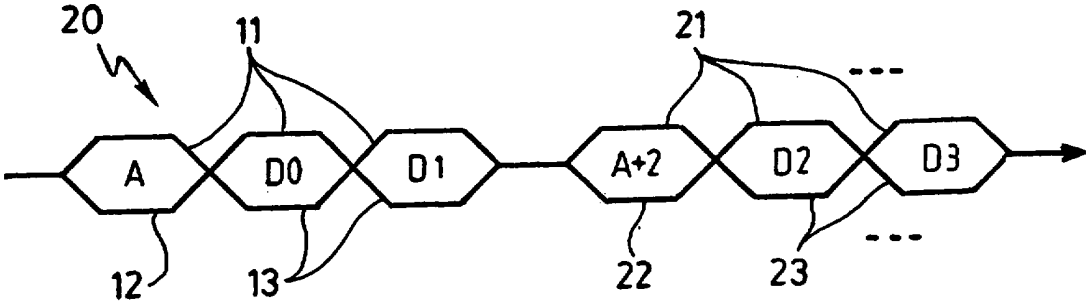

Fig. 3

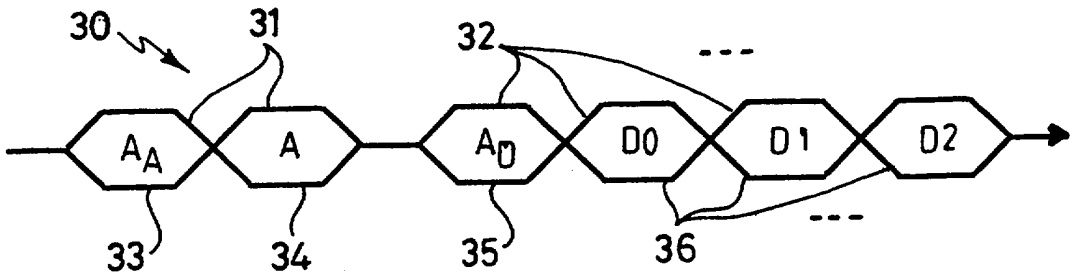

Fig. 4

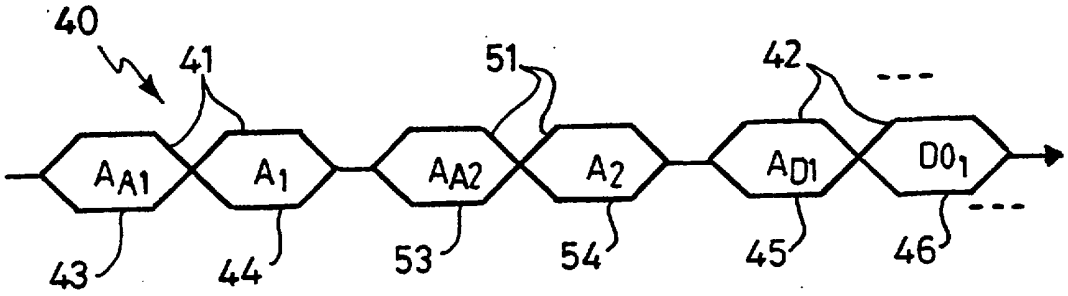

Fig . 5

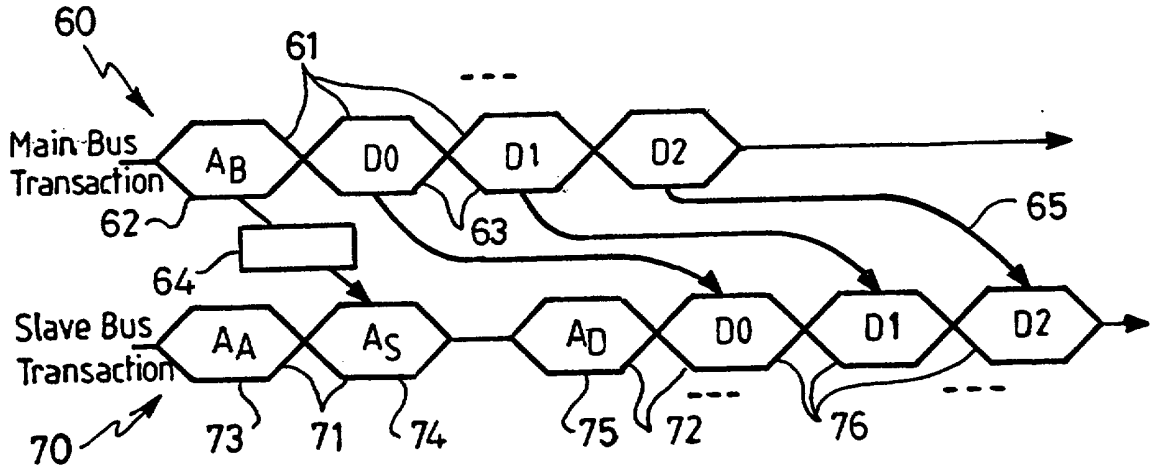

Fig. 6

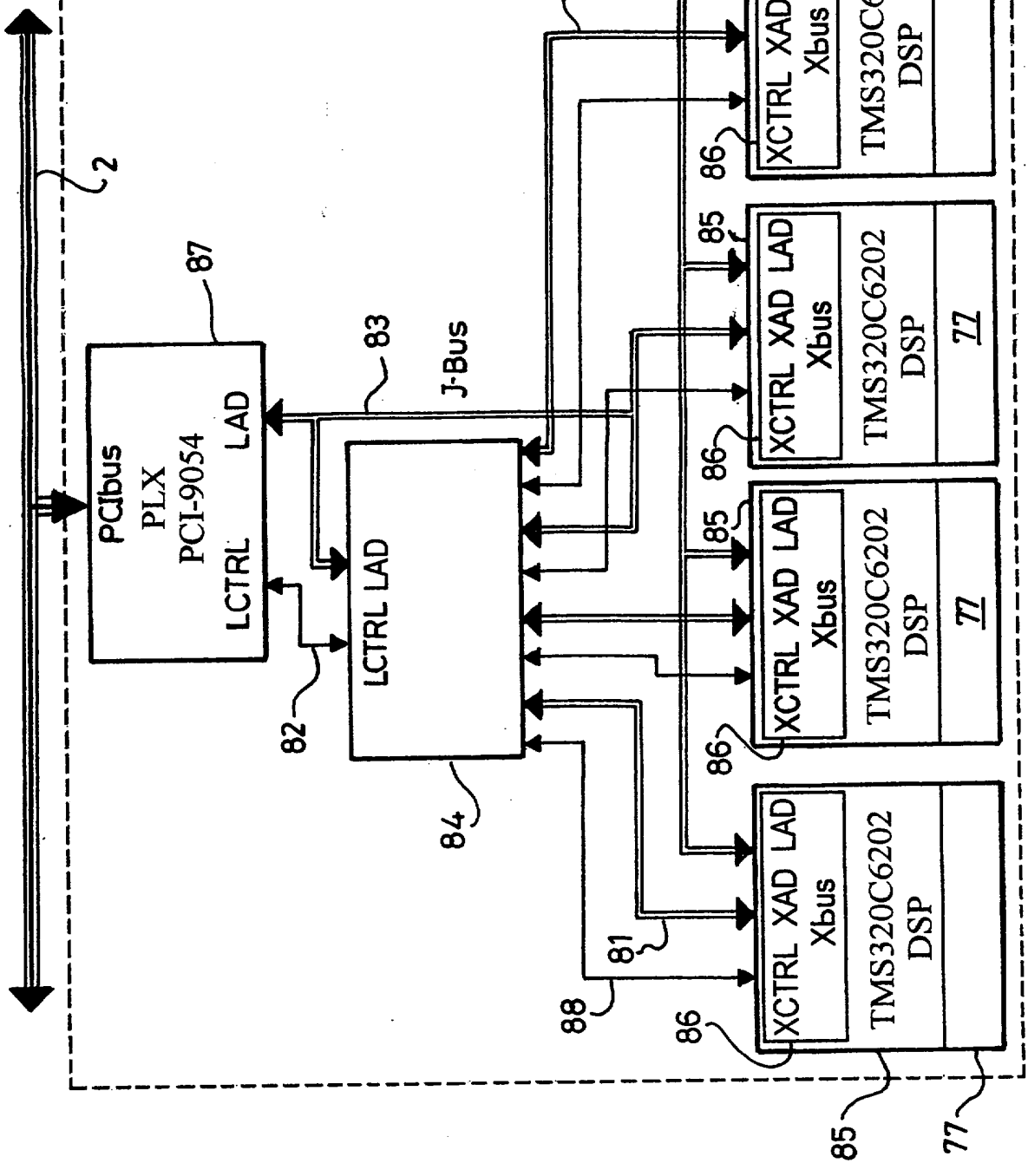
PCIbus
PLX
PCI-9054
LCTRL LAD
LCTRL LAD
J-Bus
XCTRL XAD LAD
Xbus
TMS320C6202
DSP
2
77
80
81
82
83
84
85
86
87
88

Fig. 7

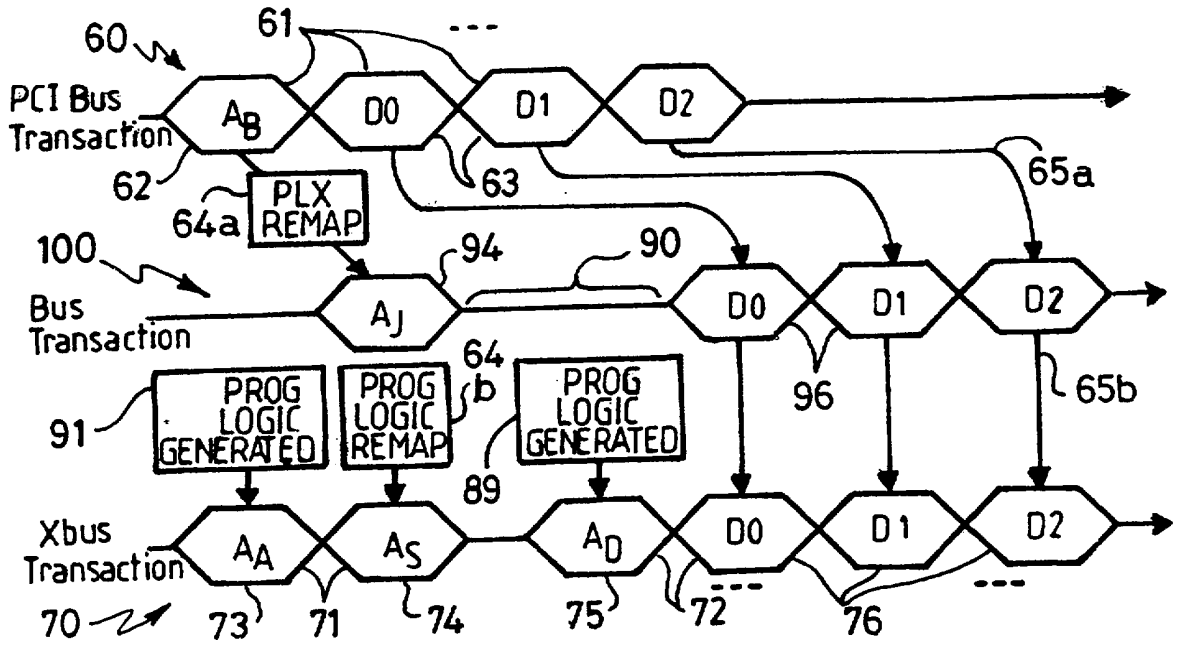
60
61
PCI Bus Transaction
A_B
D0
D1
D2
62
63
65a
64a
PLX REMAP
100
94
90
Bus Transaction
A_J
D0
D1
D2
64b
PROG LOGIC GENERATED
PROG LOGIC REMAP
PROG LOGIC GENERATED
91
89
96
65b
Xbus Transaction
A_A
A_S
A_D
D0
D1
D2
70
73
71
74
75
72
76

Fig . 8

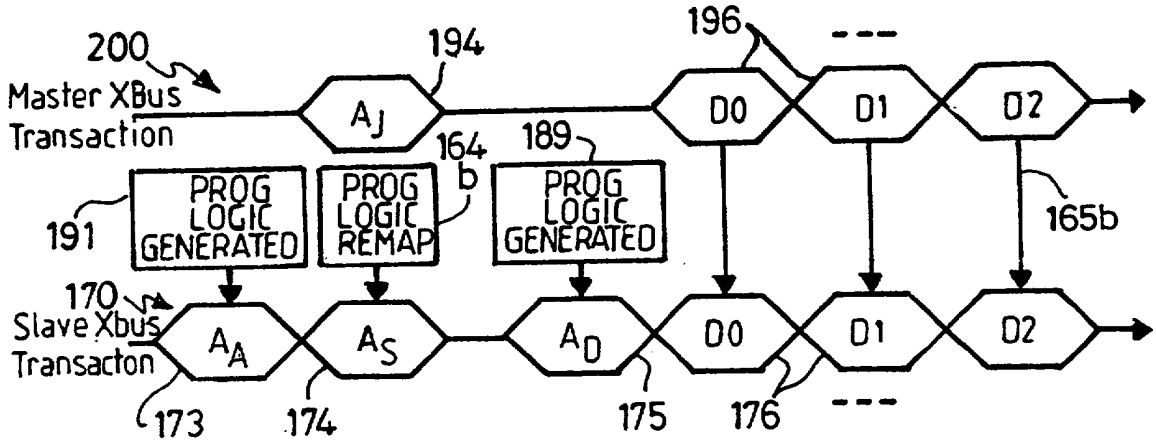
200
194
196
Master XBus Transaction
A_J
D0
D1
D2
164b
189
PROG LOGIC GENERATED
PROG LOGIC REMAP
PROG LOGIC GENERATED
191
165b
170
Slave Xbus Transacton
A_A
A_S
A_D
D0
D1
D2
173
174
175
176

Fig. 10

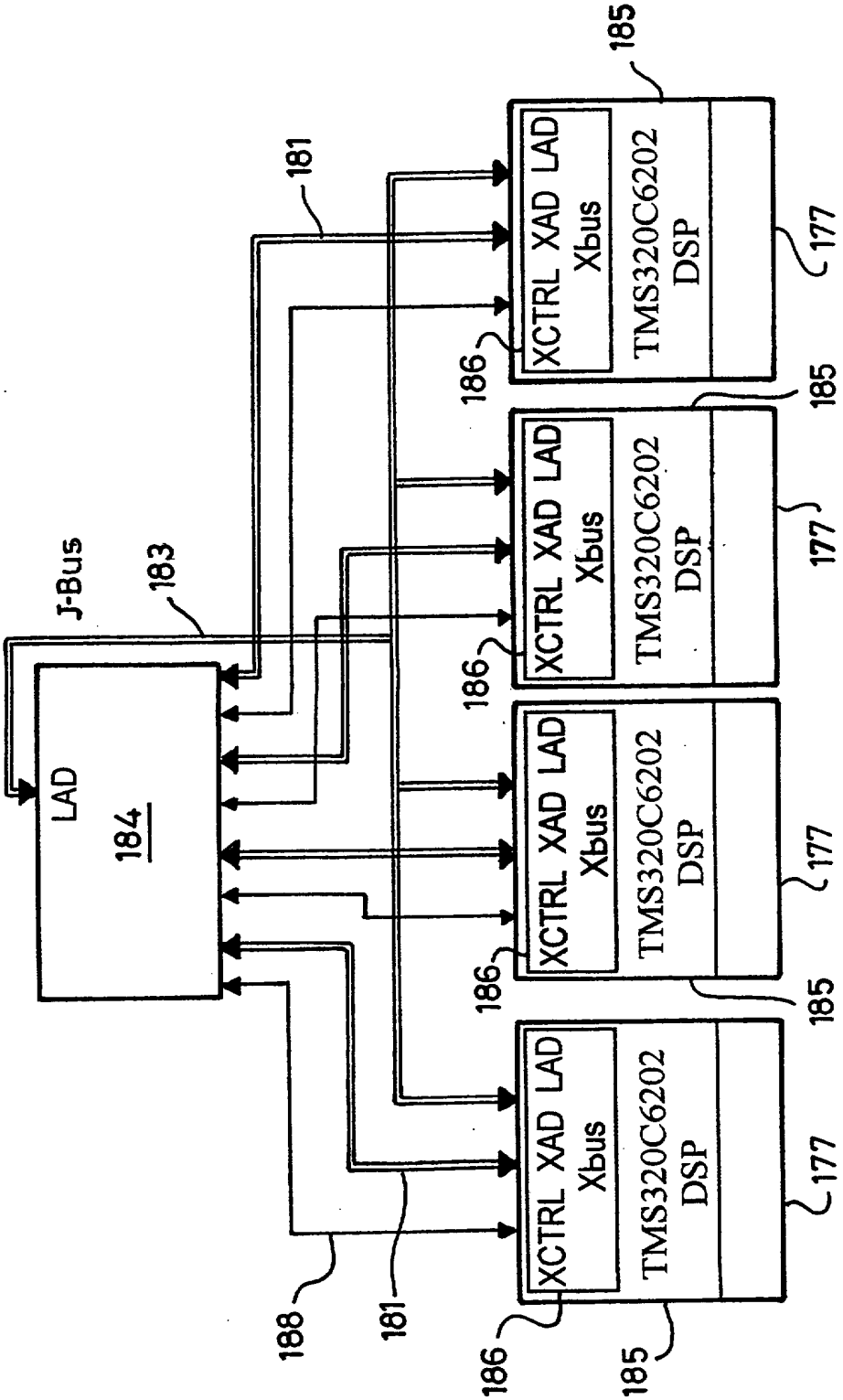
J-Bus
183
LAD
184
181
188
181
186
185
XCTRL XAD LAD
Xbus
TMS320C6202
DSP
177

Fig. 9

Direct Slave Addressing to Indirect Slave Addressing

The present invention relates to a computer bus system which uses direct addressing, where devices may be accessed using either direct addressing or indirect addressing.

Computer busses such as the PCI bus and J-Bus, multiplex the address and data onto a common bus. The address is presented for one clock cycle followed by the data on the next clock cycle. This has the advantage of halving the number of connections needed when compared with a non-multiplexed bus. For example, the PCI bus is a 32-bit bus with 32 data lines for communicating 32-bit words between devices connected to the bus. If the PCI bus did not multiplex address and data, there would need to be 32 lines for both data and address, making a total of 64 lines. Such a non-multiplexed bus would be relatively costly to implement, and would require a more complex system board.

To achieve fast data transfer on a multiplexed bus, 'burst' transfers are used. This involves a single address word 'A' followed by one or more data words where the first data word 'D0' is the data for address 'A', the second data word 'D1' is the data for address 'A+1', 'D2' is the data for address 'A+2', and so on. Each transaction therefore consists of an address phase followed by a data phase. Such a protocol for transferring data is known as "direct addressing", and devices that use this are called "direct address" devices. Direct address devices automatically increment the address for each sequential data word during the transfer of a block of data.

If a direct address burst is terminated before the data transfer is complete, when the burst resumes there must be

a new address cycle giving the address of the next data word as shown below.

Certain devices which may be connected to a bus using address/data multiplexing, use a different protocol for communicating data to or from the bus. These devices are known as "indirect address devices" and use a protocol called "indirect addressing", in which an address value is loaded in a first transaction into an "address register" of the device. The data is then loaded in a second transaction into a data register which may be a burst transaction of data into the data register. Indirect address devices automatically increment the address for each sequential data word during the transfer of a block a data. Thus this whole process involves two separate transactions across the bus to the memory of the indirect address device.

An advantage of indirect addressing is that it permits a much larger address space to be accessed within an indirect address device, whilst occupying a much smaller area of address space on the bus.

It is possible to access an indirect address device across a direct address bus by communicating sequentially two direct address transactions. During the address phase of the first transaction, the address of the address register is presented onto the bus, then during the data phase the address value is loaded into the address register. In the second transaction, the address of the data register is presented onto the bus during the address phase, followed one or more sequential data words loaded into the data register during the data phase loads.

Although it is possible to use an indirect address device

on a direct address bus, there are two main problems that make it difficult in practice to integrate an indirect address device with a direct address bus. The first of these is that if there are other devices connected to the bus which may communicate with the indirect address device, then these other devices may attempt to access the indirect address device in between the two sequential direct address transactions needed to complete the indirect address transaction. The second problem is that some direct address devices do not have the capability to perform two sequential direct address transactions to communicate with an indirect address device. Although it is in principle possible to devise a bus control system to avoid such problems, existing hardware and software for many applications will have to be completely redesigned to ensure that direct address devices can reliably communicate with indirect address devices using two sequential direct address transactions.

It is an object of the present invention to provide a computer bus system that addresses these problems.

Accordingly, the invention provides a computer bus system, comprising: a bus; at least one bus master device and at least one bus slave device, the bus master device and bus slave device being connected to the bus so that the bus master device may communicate with the bus slave device over the bus; wherein:

i) the bus has an address space with parts of the bus address space being assigned to different devices connected to the bus;

ii) the bus is a multiplexed address/data bus for transferring in a direct address transaction between said

devices, blocks of data, each of said direct address transactions comprising one or more burst transactions consisting of an address phase followed by a data phase, the address phase including a bus space address value;

iii) the bus slave device includes an indirect address device, addressable in an indirect address transaction, said transaction comprising an address register load transaction followed by a data register load transaction;

iv) the indirect address device has a memory with memory locations identified by address values;

v) the address register load transaction comprises a destination address value for blocks of data communicated to/from the memory of the indirect address device;

characterised in that the slave device includes a transaction translation device between the bus and the indirect address device, the transaction translation device being adapted to translate a direct address transaction on the bus to an indirect address transaction including a mapping of the bus space address value to the destination address value.

The bus may be a system bus, for example for a personal computer. Alternatively, the bus may be a local bus, such as a J-bus, for example a dedicated bus connecting a number of devices together separate from any other general bus such as one linking the devices to a system microprocessor.

Therefore, a direct address transaction received by the slave device is presented to the indirect address device as an indirect address transaction. In other words, one

direct address transaction can be translated to construct two sequential transactions to access the indirect address device.

The data transferred in the indirect address transaction may be data that is either written to the slave device from the master device, or data that is read from the slave device to the master device.

In a preferred embodiment of the invention, the address values for the indirect address device are used to identify both an address register and a data register in the indirect address device. The address register load transaction for blocks of data communicated to/from the memory of the indirect address device may then comprise two address values: an address register value and the destination address value. The data register load transaction then includes a data register address value. The transaction translation device, as part of the translation of the direct address transaction to the indirect address transaction, can then be arranged to generate both the address register address value and the data register address value.

In particular, the transaction translation device may translate the direct address transaction to the indirect address device as follows. First, the transaction translation device generates the address register address value for the indirect address device. It then translates the address value from the bus and loads it into the address register of the indirect address device. Then, it generates the data register address value for the indirect address device. Finally, it either passes the data word or words transparently through from the bus and loads them into the data register of the indirect address device, if

it is a data write transaction, or it passes the data word or words transparently through from the indirect address device data register to the bus, if it is a data read transaction.

The invention is therefore applicable to the case of a computer bus system conforming to the direct address protocol of busses such as the PCI bus, as used in personal computers, or the J-Bus as used by the Intel i960 family of microprocessors, and bus slave devices such as those based on the Expansion Bus (Xbus) standard, as implemented in the TMS320C6 series of digital signal processing (DSP) chips manufactured by Texas Instruments, Inc.

A block of data may comprise one or more data words, for example 32-bit data words. The data register load transaction may then comprise the data register address value followed by one or more data words.

The address of the address register and the address of the data register could be communicated by the master device to the slave device, therefore, either or both of the address register address value and/or the data register address value may be alterable and stored in the transaction translation device.

This, however, is information that does not normally need to be altered, therefore in some cases it is preferred if the address register address value and the data register address value are both fixed and generated internally by the transaction translation device.

Also according to the invention, there is provided a method of communicating blocks of data over a computer bus

system, the system comprising: a bus, the bus having an address space and being a multiplexed address/data bus for transferring in a direct address transaction blocks of data; at least one bus master device and at least one bus slave device, the bus master device and bus slave device being connected to the bus so that the bus master device may communicate with the bus slave device over the bus, the bus slave device including an indirect address device; the indirect address device has a memory with memory locations identified by address values; wherein the method comprises the steps of:

a) assigning parts of the bus address space to different devices connected to the bus;

b) communicating a block of data to/from a bus master device from/to a bus slave device in the form of a direct address transaction over the bus comprising one or more burst transactions consisting of an address phase followed by a data phase, the address phase including a bus space address value;

c) storing in the memory of the indirect address device a block of data communicated to the bus slave device , or retrieving from the memory of the indirect address device a block of data to be communicated to the bus master device, in the form of an indirect address transaction, the indirect address transaction comprising an address register load transaction followed by a data register load transaction, the address register load transaction comprising a destination address value for the received block of data;

characterised in that the method comprises the steps of:

d) prior to step c), translating the direct address transaction to the indirect address transaction including mapping the bus space address value to the destination address value.

The invention will now be described by way of example, with reference to the accompanying drawings, in which:

Figure 1 shows schematically a system bus for a computer to which a number of devices are connected;

Figure 2 shows schematically a burst transaction on a direct address bus with address/data multiplexing;

Figure 3 shows schematically multiple burst transactions on a direct address bus with address/data multiplexing;

Figure 4 shows schematically an indirect address transaction for an indirect address device;

Figure 5 shows schematically how an address register of an indirect address device can be corrupted if a direct address device interrupts two sequential direct address transactions to the indirect address device;

Figure 6 shows schematically how, according to a preferred embodiment of the invention, a single direct address transaction can be translated to an indirect address transaction;

Figure 7 shows a block circuit diagram of direct slave address interface logic used to translate a

direct address transaction to an indirect address transaction; and

Figure 8 shows schematically the translation by the circuit in Figure 7 of a single direct address transaction to an indirect address transaction.

Figure 1 shows a block schematic diagram of a conventional computer system 1, for example for a personal computer, having a system bus 2 to which a number of devices 3-6 are connected. The system bus 2 is a 32-bit PCIbus using address/data multiplexing with a direct address protocol for transferring data over the bus between devices 3-6.

The devices include a system central processor unit (µP) 3, a random access memory (RAM) 4, a sound card (S) 5, and a graphics card (G) 6. For clarity, not shown are the usual other inputs to and outputs from each of the devices 3-6, such as a keyboard connection to the microprocessor, a speaker output from the sound card 5, or a video monitor output from the graphics card 6.

A bus arbiter device (A) 8 is also connected to the system bus 2. In additional to 32 address/data lines, the bus 2 includes a variety of control lines. One purpose of these control lines is to control access to the bus 2. There may be only one bus master device at any one time, and the bus arbiter 8 controls which one of the devices 3-6 has access to the bus 2 as a master device so that it may communicate with one of the other devices 3-6 as a slave device.

Either the microprocessor 3 or the sound card may be a master device, while the random access memory 4 and graphics card 6 may only be slave devices.

As shown in Figure 2, all the devices 3-6 communicate with each other in a direct address protocol 10 in which one or more burst transactions 11, as shown in Figure 2, are communicated over the bus from a master device 3,5 to a slave device 3-6. The burst transaction consists of an address phase 12 consisting of a 32-bit address word followed by a data phase 13 consisting of one or more data words. Each device 3-6 has assigned to it an address space on the system bus 2 . The address word 12 therefore points to one of the devices as a slave device, and is interpreted by the slave device to be an address location associated with that device 3-6. The slave device receives the burst transaction 10 with each data word 13 being stored in a location in memory that is automatically incremented starting at the address value 12.

Figure 3 shows another example of the direct address protocol 20. The burst transaction 12 is interrupted and the master device 3,5, must request use of the bus again. Once it is allowed access to the system bus 2 by the bus arbiter 8, it communicates a further burst transaction 21 with an address value 22 incremented by an appropriate amount so that the remainder 23 of the data is transferred as a block to the correct address range associated with the slave device 3-6. The burst transaction(s) 20 make up a direct address transaction.

Indirect addressing uses an indirect address protocol 30, comprising two sequential transactions, as shown in Figure 4. The first of these transactions is an address register load transaction 31, which is followed by a data register load transaction 32.

As will be explained in greater detail below with

reference to Figure 7, the present example concerns the indirect addressing mechanism as used on a Texas Instruments (trade mark) TMS320C6 family of digital signal processors (DSP) 85 and its Expansion Bus (Xbus) 86. The indirect address transaction 30, involves firstly loading an address into the address register in the Xbus 86 before loading, or retrieving, a burst of one or more data words to, or from, the data register in the Xbus 86. This involves two separate accesses as shown in Figure 4, where:

- A_A is the address 33 of the address register.
- A is the destination address 34 for the data in a DSP memory 77.
- A_D is the address 35 of the data register.
- D0, D1, D2, etc are the data words 36 to be loaded into addresses A, A+1, A+2, etc.

Note that in a burst access to the data register, the address register auto-increments so that it always contains the address of the next data word in the data burst. Note also that these addresses 12,22,34 are word addresses. If the bus is 32-bit, as is the case with the Xbus 86 then the byte address equals the word address times four.

The advantage of the indirect addressing mechanism is that it can give access to a large address area via just two registers. If we take for example the PCIbus 2 connected to a TMS320C6 Xbus 86, the TMS320C6 Xbus 86 has a local address space of 4 GB, all of which can be accessed via two Xbus registers which can occupy just two PCIbus address locations.

Referring again to Figures 1-3, In a system such as PCIbus 2 which allows multiple bus master devices 3,5 the various bus masters request use of the bus 2 when they have data to transfer across the bus 2 to the target device 3-6. The bus arbiter 8 is then responsible for granting access to the bus 2. Once granted access to the bus 2, the bus master 3,5 presents the address cycle 12 followed by one or more data cycles 13. If this is a burst access then the data cycles 13 will continue until:

a) the bus master 3,5 completes its transfer.

b) the target device 3-6 tells the bus master 3,5 to terminate the burst.

c) the bus master 3,5 terminates the burst if for instance its FIFO buffers become full (read) or empty (write).

d) another bus master 3,5 has requested use of the bus and the bus arbiter 8 tells the current bus master 3,5 to relinquish control of the bus 2.

If the burst terminated before the bus master device 3,5 has completed its data transfer (i.e. cases b, c and d above) it must re-arbitrate for use of the bus in order for it to complete its transfer.

If multiple bus masters 3,5 are accessing a device 85 using indirect addressing then there is a danger that in between the time that the bus master 3,5 writes to the address register and the data register, another bus master 3,5 may have written another value to the address register. This possibility 40 is illustrated in Figure 5, where:

- A_{A1} is the address 43 of the address register driven by a first bus master.
- A_1 is a first destination address 44, loaded by the first bus master.
- A_{A2} is the address 53 of the address register driven by a second bus master.
- A_2 is a second destination address 54 loaded by the second bus master, thus corrupting the first value 44, previously loaded by the first bus master.
- A_{D1} is an address 45 of the data register loaded by the first bus master.
- $D0_1$ is the first data word 46 loaded by the first bus master, intended to be loaded into the first destination address A_1 44, which will in fact get loaded into the second destination address A_2 54.

Note that the 'corruption' of the address register by second bus master could occur between the first bus master's address register load transaction 41 and data register load transaction 42 (as shown in Figure 5), or it could occur when a burst transfer 42 to the data register gets broken up into a series of bursts due to reasons b), c) or d) above.

Possible solutions for the problem of multiple bus master devices are either to allow only a single bus master device 3,5 to access the indirect address device 85, or to disable all other bus master devices 3,5 until an active bus master device has completed all phases of its transfer. Both of these possibilities adversely affect the system's flexibility, add additional complexity in controlling the multiple bus master devices 3,5 and reduce the effective data transfer rate.

As mentioned above, bus master devices 3 driven by a microprocessor can be programmed to carry out the two stage process involved in accessing a device, which uses indirect addressing. Other non-intelligent bus master devices, such as the sound card 5, may only have the ability to read or write blocks of data 13,23 to a pre-programmed address. These devices 5 cannot be programmed to pre-load the address register of an indirect address device 85 prior to reading or writing its block of data. As a result such non-intelligent bus master devices 5 cannot directly read and write data to a device 85, which uses indirect addressing.

An example of a non-intelligent bus master device is the widely used Intel (trade mark) 82557 Ethernet LAN Controller. This is a PCIbus device for interfacing to 10BASE-T and 100BASE-T Ethernet. Buffer Descriptors give the device the PCIbus address to which it should write its receive data and from which it should read its transmit data. With the Xbus on the TMS320C6 family of DSPs using an indirect addressing mechanism this means that the Intel 82557 cannot read or write its data buffers directly from the TMS320C6 DSPs.

In the above example, to get data from the Intel 82557 to the TMS320C6 DSP, the data to/from the Intel 82557 must go via a buffer memory on the PCIbus where a microprocessor could then send, or retrieve, the data to, or from, the TMS320C6 DSP. This means that the data must be sent twice over the PCIbus, thus consuming additional bus bandwidth and adding latency to the transfer.

As described below with reference to Figures 6, 7 and 8, a preferred embodiment of the invention therefore provides

"direct slave address interface logic" 84,87 that allows bus master devices 3,5 on the address/data multiplexed direct address bus 2, for example the PCIbus, to directly address memory locations on a slave device 85 which also uses an address/data multiplexed bus but which uses an indirect addressing mechanism, for example the Expansion Bus Xbus 86 on the Texas Instruments TMS320C62 family of DSPs. The net result of this interface logic 84,87 is that any bus master 3,5 can read and write to the memory 77 on the slave device 85 as if the memory 77 were directly memory-mapped onto the bus 2.

The essence of the invention is summarised in Figure 6. The interface logic 84,87 takes a direct address bus transaction 60, consisting of one or more burst transactions 61, each of which has an address phase 62 followed by a data phase, consisting of one or more data cycles 63, and presents it to the indirect address device 85 as a slave bus transaction 70 consisting of two transactions 71,72. The first 71 of these two transactions takes the address cycle 62 of the original transaction 61, and remaps 64 the address 62 to a remapped address value 74 and writes this into the address register of the slave device 85. In general this remapping 64 will change the original address value 74, but the remapping may in some cases leave the address value 74 unchanged. For a data write transaction it then takes the one or more data cycles 63 and transfers 65 these to data values 76 of the data register load transaction 72 so that these are written without change into the data register of the slave device 85. This is shown in Figure 6, where:

- A_B is the address 62 of the slave device's memory 77 in the main bus's (e.g. the PCIbus 2) address space.

- A_s is the remapped address value 74 generated from A_B but is remapped 64 by the interface logic 84,87 to give the correct address in the slave device's address space, and is then loaded into the address register of the slave device's Xbus 86.
- A_A is the address 73 of the address register generated locally by the interface logic 84,87.
- A_D is the address 75 of the data register generated locally by the interface logic 84,87.
- D0, D1, D2 etc are the data words 76 to be loaded into addresses A_s, A_s+1, A_s+2, etc of the slave device's memory 77.

The direct slave address interface logic 84,87 shown in Figure 7 has been implemented as part of a slave device 80 connected to the bus 2, to interface between a 32-bit 33 MHz PCIbus 2 and the Expansion Bus (Xbus) 86 of a Texas Instruments TMS320C6202 DSP 85. The interface logic consists of a PLX PCI-9054 PCIbus interface IC 87 and a Xilinx XC95288XL programmable logic IC 84, and supports the connection of four TMS320C6202 DSPs 85 onto the PCI bus 2. Each of the DSPs 85 can be a PCIbus master or a PCIbus slave. The direct slave address interface logic 84,87 is also used to support local Xbus-to-Xbus data transfers, which would otherwise get corrupted, as described later. The various control lines of the Xbus 86 indicated in Figure 7 are:

- LAD - J-bus Address/Data Bus 83
- XAD - Xbus Address/Data bits remapped by Programmable Logic 81
- LCTRL - J-bus Control Signals 82 (LHOLD, LHOLDA, LADS, LW/R, LBLAST, LREADY, LWAIT, LBE[3:0])
- XCTRL - Xbus Control Signals 88 (XHOLD, XHOLDA, XCS,

XCNTL, XADS, XW/R, XBLAST, XREADY, XWAIT, XBOFF)

Figure 7 will now be described in more detail, with reference also to Figure 8. The PLX PCI-9054 chip 87 interfaces the PCIbus 2 through onto the J-Bus address/data bus 83, remapping 64a the PCIbus address (A_B) 62 to a required J-bus address (A_J) 94. The interface logic includes the programmable logic array 84 that remaps 64b the J-bus address (A_J) 94 to the required TMS320C6202 DSP memory address (A_S) 74.

Note that by careful design of the J-bus memory map it is possible to minimise the number of J-bus Address/Data lines, which need to be remapped. Thus only those Address/Data lines which must be remapped need to connect via the programmable logic 84.

The programmable logic 84 also connects to all of the Xbus control signals 88. "State machines" in the programmable logic 84 generate 91 the Xbus 86 address register address cycle (A_A) 73, and also generate 89 the Xbus 86 data register address cycle (A_D) 75. During the data cycles (D0, D1, D2 etc) 76 the J-bus Address/Data signals 96 are passed 65a,65b transparently through the programmable logic 84.

As shown in Figure 8, this permits the translation of a single direct address transaction 60 to be translated into an indirect address transaction 70 via a J-bus transaction 100 consisting of the J-bus address (A_J) 94, followed by a short time delay 90 until the programmable logic has generated 89 the data register address cycle (A_D) 75. To summarise, Figure 8 shows this for the case of a PCIbus 2 to Xbus 86 direct slave address translation, where:

- A_B is the address 62 of the DSP's memory 77 in PCIbus address space.
- A_J is a J-bus address 94 taken from A_B 62 but remapped 64a by the PLX PCI-9054 chip 87 to give the address 94 of the DSP's memory 77 in J-bus address space.
- A_s is the required DSP memory address 74 taken from A_J 94 but remapped 64b by the programmable logic 84 to give the address 74 of the DSPs' memory 77 in DSP address space, and is loaded into the Xbus 86 address register.
- A_A is the address 73 of the address register generated 91 locally by the programmable logic 84.
- A_D is the address 75 of the data register generated 89 locally by the programmable logic 84.
- D0, D1, D2 etc are the data words 76 to be loaded into DSP addresses A_s, A_s+1, A_s+2, etc.

A second preferred embodiment of the invention is shown in Figures 9 and 10. For convenience, those parts of the drawings corresponding with Figures 6 and 7 are indicated by reference numerals incremented by 100. The second preferred embodiment provides "direct slave address interface logic" 184 that allows one TMS320C6 Xbus 185 as bus master to directly address memory locations on a slave TMS320C6 Xbus device 185, which is connected to the same J-Bus 183. The net result of this interface logic 184 is that any Xbus as J-Bus master 185 can read and write to the memory 177 of the Xbus slave device 185 as if the memory 177 were memory-mapped directly onto the bus 183. More importantly, the "direct slave address interface logic", when used in conjunction with Xbus "back-off" logic, allows master Xbus to slave Xbus read or write data transfer across the J-Bus 183 without data corruption.

The direct slave address interface logic 184 shown in Figure 9 has been implemented to interface between the Xbuses 186 of four Texas Instruments TMS320C6202 DSPs 185. In this implementation, any one of the four Xbuses is capable of mastering the J-Bus 183 and reading or writing to the memory of any of the other three indirect address slave Xbus devices on the J-Bus 183 via the transaction translation device interface logic. The various control lines of the Xbus 186 indicated in Figure 9 are:

- LAD - J-Bus Address/Data Bus 183
- XAD - Xbus Address/Data bits remapped by programmable logic 181
- XCTRL - Xbus Control Signals 188 (XHOLD, XHOLDA, XCS, XCNTL, XADS, XW/R, XBLAST, XREADY, XWAIT, XBOFF)

Figure 9 will now be described in more detail, with reference also to Figure 10. The interface logic consists of the programmable logic array 184 that remaps 164b the J-Bus address (A_J) 194 generated by the master Xbus to the required slave TMS320C6202 DSP memory address (A_S) 174.

The programmable logic 184 also connects to all of the Xbus control signals 188. "State machines" in the programmable logic 184 generate 191 the slave Xbus 186 address register address cycle (A_A) 173, and also generates 189 the slave Xbus 186 data register address cycle (A_D) 175. During the data cycles (D0, D1, D2 etc) 176 the J-Bus Address/Data signals 196 are passed 165b transparently through the programmable logic 184.

As shown in Figure 10, this permits the translation of a single direct address transaction 200 to be translated into an indirect address transaction 170. To summarise,

Figure 10 shows this for the case of a Xbus-to-Xbus 186 direct slave address translation, where:

- A_J is the address 194 of the slave DSP's memory 177 in J-Bus address space.
- A_s is the required slave DSP memory address 174 taken from A_J 194 but remapped 164b by the programmable logic 184 to give the address 174 of the slave DSP's memory 177 in DSP address space, and is loaded into the slave Xbus 186 address register.
- A_A is the address 173 of the slave Xbus address register generated 191 locally by the programmable logic 184.
- A_D is the address 175 of the slave Xbus data register generated 189 locally by the programmable logic 184.
- D0, D1, D2 etc are the data words 176 to be loaded into DSP addresses A_s, A_s+1, A_s+2, etc.

To avoid data corruption during master Xbus to slave Xbus data transfers via the J-Bus, the burst transfer must be terminated by issuing a "Back off" to the master Xbus and a "Burst Last" to the slave Xbus, if during the transfer the slave Xbus is ready to send or receive the next data word when the master Xbus is not ready. The logic issues a "Back off" by asserting XBOFF; it issues a "Burst Last" by asserting XBLAST; the slave Xbus is ready when it asserts XREADY and the master Xbus is not ready when it asserts XWAIT.

Adding the direct slave address interface logic 84,87;184 to devices 85,185 that operate with an indirect addressing interface enables the following:

1) Multiple bus master devices 3,5,185 can perform

unrestricted concurrent accesses to the indirect address slave device 85,185.

2) Non-intelligent bus master devices 5 can read and write data directly to the slave device 85.

3) When using Texas Instruments TMS320C62 family of DSPs which are directly connected Xbus-to-Xbus, data can be transferred without corruption so long as the "Back off" logic is also implemented.

The advantages provided by the invention in each of these three cases will now be considered in turn.

Unrestricted concurrent access to a slave device has a number of performance advantages. Firstly, data can be transferred directly from source to destination across the bus without the need for all data to go via a single system master device. This halves the bus bandwidth used for the transfer and more than halves the latency of the transfer. Secondly, all bus master devices can be permanently enabled without the danger that one bus master device might corrupt the indirect address register setting of the indirect address device that has been set by another bus master device. Without this, other bus master devices would have to be disabled for the duration of every transfer to the Slave. This would result in inefficient bus usage, with data transfers being delayed. Finally, bus control is simplified since all bus master devices can be left enabled.

Direct access to a slave device from a non-intelligent bus master device provides the advantage that data can be transferred directly from source to destination across the bus without the need for all data to go via a single

system master device. This halves the bus bandwidth used for the transfer and more than halves the latency of the transfer.

The TMS320C6 family of DSPs can transfer data directly to or from other TMS320C6 DSPs, from Xbus-to-Xbus, without data corruption, so long as the Xbus-to-Xbus interface includes the "Transaction Translation" logic and the "Back-off" logic. Without the "Transaction Translation" logic and the "Back-off" logic, direct Xbus-to-Xbus data transfers cannot be guaranteed to be free from data corruption. This has the advantage that data can be transmitted directly from one TMS320C6 DSP to another.

In conclusion, the direct slave address interface logic allows bus masters on an address/data multiplexed bus (e.g. PCIbus or J-Bus) to directly address memory locations on a slave device, which also uses an address/data multiplexed bus but which uses an indirect addressing mechanism, for example, the Expansion Bus (Xbus) on the Texas Instruments TMS320C62 family of DSPs.

The direct slave address interface logic thus allows multiple bus masters to perform unrestricted concurrent accesses using byte, word or burst data read or write transfers to a slave device that uses an indirect addressing mechanism. It also allows non-intelligent bus master devices to read and write data directly to an indirectly addressable slave device.

In particular the direct slave address interface logic allows the Expansion Bus on the Texas Instruments TMS320C62 family of DSPs to connect to a PCIbus such that its memory is directly memory-mapped into the PCIbus's address space

The direct slave address interface logic also allows two Texas Instruments TMS320C62 family DSP devices to connect to each other via their Expansion Bus (XBus) ports such that they can transfer bursts of data between each other without data corruption.

The invention therefore provides a convenient and economical solution to the problems associated with integrating an indirect address device with a direct address bus.

Claims

1. A computer bus system, comprising: a bus; at least one bus master device and at least one bus slave device, the bus master device and bus slave device being connected to the bus so that the bus master device may communicate with the bus slave device over the bus; wherein:

i) the bus has an address space with parts of the bus address space being assigned to different devices connected to the bus;

ii) the bus is a multiplexed address/data bus for transferring in a direct address transaction between said devices, blocks of data, each of said direct address transactions comprising one or more burst transactions consisting of an address phase followed by a data phase, the address phase including a bus space address value;

iii) the bus slave device includes an indirect address device, addressable in an indirect address transaction, said transaction comprising an address register load transaction followed by a data register load transaction;

iv) the indirect address device has a memory with memory locations identified by address values;

v) the address register load transaction comprises a destination address value for blocks of data communicated to/from the memory of the indirect address device;

characterised in that the slave device includes a transaction translation device between the bus and the indirect address device, the transaction translation device being adapted to translate a direct address

transaction on the bus to an indirect address transaction including a mapping of the bus space address value to the destination address value.

2. A computer bus system as claimed in Claim 1, in which:

i) the address values for the indirect address device may be used to identify both an address register and a data register in the indirect address device;

ii) the address register load transaction for blocks of data communicated to/from the memory of the indirect address device comprises two address values: an address register address value and the destination address value;

iii) the data register load transaction includes a data register address value;

characterised in that the transaction translation device, as part of the translation of the direct address transaction to the indirect address transaction, generates both the address register address value and the data register address value.

3. A computer bus system as claimed in Claim 1 or Claim 2, in which a block of data comprises one or more data words, the data register load transaction comprising the data register address value followed by one or more of the data words.

4. A computer bus system as claimed in any preceding claim, characterised in that the address of the address register and the address of the data register are both fixed and generated internally by the transaction

translation device.

5. A computer bus system as claimed in any of Claims 1 to 3, in which the address register address value and/or the data register address value is/are alterable and stored in the transaction translation device.

6. A method of communicating blocks of data over a computer bus system, the system comprising: a bus, the bus having an address space and being a multiplexed address/data bus for transferring in a direct address transaction blocks of data; at least one bus master device and at least one bus slave device, the bus master device and bus slave device being connected to the bus so that the bus master device may communicate with the bus slave device over the bus, the bus slave device including an indirect address device; the indirect address device has a memory with memory locations identified by address values; wherein the method comprises the steps of:

a) assigning parts of the bus address space to different devices connected to the bus;

b) communicating a block of data to/from a bus master device from/to a bus slave device in the form of a direct address transaction over the bus comprising one or more burst transactions consisting of an address phase followed by a data phase, the address phase including a bus space address value;

c) storing in the memory of the indirect address device a block of data communicated to the bus slave device , or retrieving from the memory of the indirect address device a block of data to be communicated to the bus master device, in the form of an indirect address transaction,

the indirect address transaction comprising an address register load transaction followed by a data register load transaction, the address register load transaction comprising a destination address value for the received block of data;

characterised in that the method comprises the steps of:

d) prior to step c), translating the direct address transaction to the indirect address transaction including mapping the bus space address value to the destination address value.

7. A computer bus system, substantially as herein described, with reference to or as shown in the accompanying drawings.

8. A method of communicating blocks of data over a computer bus system, substantially as herein described, with reference to or as shown in the accompanying drawings.

28

Application No: GB 0011599.8
Claims searched: 1-8

Examiner: Brian Ede
Date of search: 14 December 2000

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.R): G4A(AFGDC, AND)

Int Cl (Ed.7): G06F 9/35 12/00 12/02 12/04 12/06 12/08 12/10 13/00 13/10 13/12 13/14 13/16 13/28 13/36

Other: Online: EPODOC, JAPIO, WPI

Documents considered to be relevant:

Category	Identity of document and relevant passage	Relevant to claims
A	GB 2307569 A (HOLTEK MICROELECTRONICS INC) see Fig 4	

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.