COMPUTER RESOURCE CONFIGURATION METHOD AND APPARATUS

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to the field of automatic resource configuration systems for computers.

2. Prior Art

Computers, and especially micro-computers, in existence today often allow the user of the computer to configure the resources available on the computer to meet his or her particular needs. For example, a user requiring additional memory may purchase an add-on card. The memory is, by example, plugged into a slot on a motherboard such as in certain Apple II series computers. One computer may be configured with cards to run a color graphics monitor while another has cards to run a monochrome monitor. In computers that are specifically built to accept these add-on cards, it is necessary that the central processor is aware of what type of card is in each available slot.

Several methods are utilized to allow computer systems to determine their installed resource configuration at the time the computer system is powered on. In some computer systems, a system administrator must manually define to the computer system the available resources by keying into the system resource definitions and addresses. Another method involves setting switches on the add-on cards to identify the resources on the card to the computer.

The most pertinent prior art known to the applicant is Texas Instrument's NuBus system. The NuBus system involves a bus with 32-bit addressing capability (4 gigabytes). By system

convention, the upper 256 megabytes of this address space is reserved for control space for cards. The control space is then divided into sixteen 16 megabyte address partitions, each assigned to one card installed on the system. Thus, the NuBus addressing scheme supports up to sixteen cards in its dynamic configuration methodology. Each of the sixteen 16 megabyte address partitions is further divided into a 128 byte configuration ROM address space and the remainder of the 16 megabytes is available as defined by the individual card.

The 128 byte configuration ROM address space has fixed addresses assigned for various information about the card such as a serial number, vendor identifier, card type, part number, and address offsets for such items as the configuration register, device driver, diagnostic tools, etc. This convention of dividing address space and utilizing address offsets allows the device drivers, diagnostic tools, and other information to be placed in the remainder of the 16 megabytes of memory and allows this information to occupy varying amounts of memory depending on the specific card type. In other words, a card requiring a relatively large amount of device driver code and a relatively small amount of diagnostic code could utilize memory for each of the types of code in the amount required. Another card requiring a relatively small amount of device driver code and a relatively large amount of diagnostic code could utilize memory in the proportions required for it. A fixed amount of memory is not required for device driver code, diagnostic code, etc.

The configuration information resides on each card in the system in a read-only-memory (ROM) and is addressable as

described above. At the time the system is started, automatic self-configuration, system self-test, and processor booting occurs based on the information stored in these ROMS.

As will be seen, the present invention departs from the above convention by allowing multiple resources to be defined, instead of a fixed set in addition to other departures.

According to the present invention, in a computer having at least one slot for receiving circuits for coupling to said computer, there is provided a method of determining the configuration of resources of said circuits, comprising:

(a) for each said slot, examining a set of predetermined addresses in said computer's address space searching for the beginning of a format header;

(b) for each said slot, computing an address in said computer's address space where a resource directory is located from an offset value in a directory offset field located in said format header;

(c) sequentially reading entries in said resource directory and determining an address of a corresponding resource list from an offset value in said entry;

(d) for each said resource list, determining information about a resource on said circuit by sequentially reading data contained in said resource list and data located at addresses computed from offsets contained in said resource list;

(e) recording said resource information in a table accessible to said computer;

whereby, the configuration of resources on said circuit in said computer is determined.

A configuration read-only-memory (ROM) is placed on each add-on card placed in the computer system, and the configuration ROM is assigned a set of addresses within the addressing space of the computer system. Within this set of addresses, a format header is located at one of a set of definable addresses. The format header contains, among other information, an offset to a directory of system resource identities (Ids). The directory of system resource Ids contains offsets to resource lists which describe the various resources on the board. This method offers an advantage over the prior art in allowing multiple resources to be defined to the system, instead of a fixed set.

The format header contains information such as which byte lane(s) the card will be using, a test pattern for validating the format header, a revision level indicator, a checksum value for validating and determining whether errors exist on the configuration ROM, and a length field for specifying the lowest address used by the declaration ROM, in addition to the directory offset.

The bus for communication between the various cards and central processor is divided into a number of byte lanes, each lane capable of carrying data between the components of the system. The format header contains data describing which of the byte lanes will be used by the card. The card may use any combination of the available byte lanes. At the time the system

is powered on the address of the format header on each board is determined and then the configuration and resources of the card are read by the operating system.

The length field specifies the lowest address in the address_space assigned to the card which is used by the configuration ROM. This leaves, for other needs, the remainder of the available memory.

It is one object of this invention to provide a method of allowing a computer to determine its resource configuration at the time the computer is powered on.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a diagram illustrating a prior art convention of assigning address space in a computer system.

Figure 2 is a diagram illustrating a convention of assigning address space in the present invention.

Figure 3 is a block diagram illustrating a method of using offset pointers to allow n resource lists to be defined for a card in a computer in accordance with the present invention.

Figure 4 is a diagram illustrating a format header and its contents within the teachings of the present invention.

Figure 5 is a diagram illustrating a resource directory and its contents as taught by the present invention.

Figure 6 is a diagram illustrating a resource list and its contents as taught by the present invention.

Figure 7 is a flow diagram showing a method of determining a configuration when a computer is started in accordance with the present invention.

Figure 8 is a flow diagram showing a method of determining whether the start of a format header has been found in a configuration memory as taught by the present invention.

Figure 9 is a flow diagram showing a method of reading resource directories and resource lists in a configuration memory as disclosed in the present invention.

Figure 10 illustrates byte lanes combinations and corresponding byte lane values and format header starting addresses used in the preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE PRESENT INVENTION

A method and apparatus for automatically determining the resource configuration of a computer is disclosed. In the following description, numerous specific details are set forth to provide a thorough understanding of the present invention. It will be obvious to one skilled in the art that the invention may be employed without the specific details. In other instances, well-known methods and structures have not been set forth in order not to unnecessarily obscure the present invention.

Figure 1 illustrates Texas Instrument's NuBus, the most pertinent prior art known to the Applicant. The NuBus system involves a bus with 32-bit addressing capability (4 gigabytes) 1. The upper 256 megabytes of this address space is reserved for control space for boards 3. The remaining 3840 megabytes is used for global memory space 2. The control space 3 is further divided into sixteen 16 megabyte partitions 5. Each of the 16 megabyte portions 6 are then further divided into a 128 byte configuration read-only-memory (ROM) 9 and an area for other data 8.

The 128 byte configuration ROM 9 has fixed addresses for various information about the card such as a serial number, vendor identifier, card type, part number, and address offsets for such items as a configuration register, device driver, diagnostic tools, etc. The method allows for offsets to information regarding only a fixed set of resources.

Referring to Figure 2, the convention for assigning address space in the preferred embodiment is disclosed. The preferred

embodiment is capable of addressing 4 gigabytes of memory, block 20. The lower 3840 megabytes of address space 22 (addresses 0000 0000 through EFFF FFFF hexadecimal) is used for random-access-memory (RAM), for read-only-memory (ROM), for super slot space, or is reserved for future use. The upper 256 megabytes 21 (addresses F000 0000 through FFFF FFFF) is used for accessing cards and for their configuration memories. The preferred embodiment uses a read-only-memory (ROM) circuit as its memory means, although it will be obvious to one skilled in the art, alternate means may be available.

The upper 256 megabytes 21 is further divided as shown in Figure 2, block 24. The preferred embodiment currently has defined slot space for six cards, labeled 9 through E, hexadecimal. Each card is assigned sixteen megabytes of address space 26. Referring the Figure 2, block 27, the sixteen megabytes of address space 26 assigned to each card is further divided by convention into an area for a format header 29 and an area for other code and data 28 such as a resource directory, resource lists, and resource code and data. This information, collectively, makes up the configuration ROM 27. The remainder of the sixteen megabyte space 30 is used for other system needs. The first byte of the format header 29 must reside in one of the upper four bytes of the slotspace reserved for that card. That is, the format header must have its first byte at address FnFFF FFFF hexadecimal to FnFF FFFC hexadecimal (where n is the slot number).

Figure 4 describes the format header 29 of the preferred embodiment. The format header 29 consists of the byte lanes field 42, a reserved field 43, a test pattern field 44, a format

field 45, a revision field 46, a cyclic redundancy check (CRC) field 47, a length field 48, and a directory offset field 49.

The byte lanes field 42 tells the computer which of the byte lanes on the bus to use when communicating with the card's configuration ROM 27. The preferred embodiment uses a bus with four byte lanes numbered 0 through 3. This technique allows the card designer to place the configuration ROM 27 in any combination of the four byte lanes, thus allowing him or her greater flexibility in his design than prior art method which require the configuration ROM's fields to be at fixed addresses.

Figure 10 lists the valid byte lane values 103, and byte lane combinations 101 which may be used, and corresponding format header starting addresses 105. The value of the byte lanes field, Figure 4, block 42 may be computed by setting a bit in the low nibble of the byte lane field 42 for each byte lane used and then setting the high nibble of the byte lanes field 42 to the low nibble's complement. For example, if a card's configuration ROM 27 used byte lanes 0, 1 and 3 when communicating with the computer, the low nibble of its byte lanes field 42 would be set to 1011 binary. The high nibble would then be set to the low nibble's complement or 0100 binary, yielding a value for the field of 0100 1011 binary or 4B hexadecimal. The format header 29 would then start at address FnFF FFFF hexadecimal, where n is the slot number for the card.

Referring still to Figure 4, the second byte of the format header in the preferred embodiment is a reserved field 43 and must be set to hexadecimal 00. The next four bytes of the format header are a test pattern 44 used to insure that a valid byte lane value was found. The preferred embodiment, by

convention, requires this field to be set to 5A932BC7 hexadecimal. These values are used by the current preferred embodiment, however, it will be obvious to one skilled in the art the values may differ in another embodiment without departing from the spirit of the invention.

The format field 45 identifies the configuration ROM 27 format. The current preferred embodiment allows only one value, 01 hexadecimal, in this field. It will, however, be obvious to one skilled in the art that more values could be assigned as more format types are defined. The revision level field 46 identifies the current ROM revision. Currently, the preferred embodiment will accept values in the range of 1 to 9.

The next four bytes are the cyclic redundancy check (CRC) field 47. This field contains a checksum to allow the configuration ROM 27 to be validated. It is computed by applying a 32-bit rotate-left-and-add function to the number of bytes specified by the length field 48. Only bytes in byte lanes specified by the byte lanes field 42 are used to calculate the CRC field 47. In making the CRC computation, the value of the CRC field 47 itself is treated as 0.

The length field 48 contains a value specifying the number of bytes from the configuration ROM's 27 starting address to the lowest-address byte used by the configuration ROM 27. The remainder of the address space reserved for the card is then available for other uses.

The directory offset value 49 specifies the offset from its address to the address of the resource directory, Figure 5, block 50. It counts only bytes in the byte lanes being used.

Referring now to Figure 3, the directory offset value is used to determine the address of the resource directory 50 for the card. The resource directory 50 will have an address within the slot space for the card (addresses Fn00 0000 through FnFF FFFF hexadecimal, where n is the slot number for the card, except for the space allocated to the format header).

The contents of the resource directory 50 are shown in Figure 5. The resource directory 50 contains an entry for each resource list, listed by resource Id 51, in the card's firmware and provides an offset 52 to each resource list, Figure 6, block 60. The preferred embodiment allows resource Ids 51 in the range of 0 to 254. The range of 0 to 127 are reserved for resource lists 60 required by the computer, while 128 to 254 may be assigned by the individual card designer. The last entry 55 in the resource list 50 must have an identification number of 255 and an offset value of zero and indicates the end of list. This method of allowing between 1 and 255 resource lists for a card offers the card designer flexibility not available in Texas Instrument's NuBus System. The Texas Instrument's NuBus system allows a fixed number of predefined resource types to be defined and has offset pointers in the format header. The present invention provides a well defined mechanism for independent expansion by the individual card designer.

Again referring to Figure 3, the offset values 52 in the resource directory 50 point to resource lists 60. The contents of the resource lists 60 are further detailed in Figure 6. Each resource list 60 contains a set of references to information about a single resource. This information must include the type 61 and name of the resource 64. It may also include references

to information regarding the resources icon 66, drivers, and other parameters 67. The resource list 60 includes an identification number 63 for each entry and an offset 62 to the information. The identification numbers 63, in the preferred embodiment, must be in the range of 0 to 254. The numbers 0 to 127 are reserved for assigned entry types and the numbers in the range of 128 to 254 are available to the individual card designer as needed. The last value 69 in every list must have a value of 255 indicating the end of the list. Associated with each identification number is an offset value giving the address of the information.

Figure 3 shows use of the offsets 62 in the resource lists 60 to point to the code or data 31 associated with the entry.

The preferred embodiment requires one special resource list for all cards which will communicate with the computer called a board resource list. This resource list provides the computer with the cards indentification number, vendor information, board flags, and initialization code.

At the time the computer is started a two stage process takes place. First, each slot is checked for the presence of a card and if one is found the format header is read and pertinent information from the format header is stored in a table. Then information about each resource is read and stored in a second table and initialization code for each slot is executed. Then, the second stage reads pertinent information about the driver and stores the information in the second table.

Referring to Figure 7, the computer starts the automatic resource configuration process by determining a first slot to examine, block 70. A first byte lane to examine is selected,

block 71. The present invention inspects byte lane 3 first because the greatest number of byte lane combinations 101 will have a format header starting address 105 in this byte lane. A check is made to see if the beginning of the format header 29 for the card in the first slot is in the first byte lane, block 72. If the beginning of the Format header was not found, branch 73, a check is made to determine if there are more byte lanes to search for the current card slot, block 76. If there are, the byte lane value is changed to the next byte lane to check, block 75, and that byte lane is checked for the beginning of a format header, block 72. If there were no more byte lanes to search for the current card slot, an error is recorded in a table, block 77. If there are more slots to be checked, block 74, the slot number being checked is incremented, block 79 and the process is repeated.

Referring to Figure 8, a method used by the preferred embodiment for checking if the beginning of the format header has been found is disclosed. First, the byte lanes code field is examined, block 80, to determine whether it contains one of the valid byte lanes code as listed in Figure 10. If the byte lanes code is invalid, branch 81, the beginning of the format header was not found in this byte lane 89. If the byte lanes code is valid, branch 82, the test pattern field is examined, block 84, to determine whether it contains the valid test pattern. If not, block 85, the beginning of header was not found in this byte lane. Otherwise, if the valid test pattern was found branch 86, it indicates the start of header was found, block 87.

After each slot is searched for a format header 29 as described in Figures 7 and 8, slots with a valid format header 29 are examined. The resource directory 50, resource lists 60, and other information is read and data such as the reference type and hardware device identifiers are stored in a table. Referring to Figure 9, the process for reading resource directories 50 and resource lists 60 is disclosed. First, several edits are performed. The reserved field is examined to insure it is set to zero, block 90. The format field is examined to insure it contains a valid format code, block 91, the revision field is examined to verify it contains a valid revision level, block 92. The CRC value is examined to insure it contains the correct value, block 94. If any of the edits fail an error condition is recorded, block 99.

The address of the resource directory is determined from examining the offset value in the directory offset field of the format header, block 95. Then, for each entry in the resource directory, the resource list referenced by the offset is read and pertinent information stored in a table, block 97.

After the pertinent resource information from each resource list on each card has been read and stored, the computer has completed the automatic resource configuration process. Each card installed on the system has been identified by its identification number, vendor information, and card type and information on initialization code, device drivers, etc. has been determined.

The present invention offers a number of advantages over the prior art. First, only the 20 bytes of the format header 29 need be located at a predetermined location in the address

space. The prior art addressing scheme requires locating 128 bytes of header information 9 in a fixed location. Second, the configuration ROM 27, including the format header 29, may be located in any combination of the available byte lanes. The card designer is not restricted to placing the configuration ROM 27 in a fixed location as in the prior art. Third, any number of resources may be defined. The card designer is not limited to a set number of resources as in the prior art. Fourth, the prior art method determined resource type by setting bits in a resource type field. Each bit indicated a different type of resource. For example, setting bit 0 indicates a memory resource, setting bit 1 indicates a boot source, setting bit 2 indicates a LAN resource, setting bit 3 indicates a monitor resource. Using this method, as more resource types are added additional bits must be added to this field. The present invention utilizes an eight byte field containing a code identifying the resource type. This field is broken into sub-fields, including a flag indicating the field type, a category indicator describing the general category of the resource, a subclass indicator further dividing the general category of the resource and driver interface information. This list of advantages is not meant to be exhaustive of the features of the present invention, but rather shows by way of example some of the important improvements of the present invention over the prior art.

Thus, an improved method of and means for automatically configuring resources on a computer system is disclosed.

Utilizing a configuration ROM and a convention for defining use of address space, a method of determining resource types and other information at the time the computer system is started is disclosed.

CLAIMS

1. In a computer having at least one slot for receiving circuits for coupling to said computer, a method of determining the configuration of resources of said circuits, comprising:

(a) for each said slot, examining a set of predetermined addresses in said computer's address space searching for the beginning of a format header;

(b) for each said slot, computing an address in said computer's address space where a resource directory is located from an offset value in a directory offset field located in said format header;

(c) sequentially reading entries in said resource directory and determining an address of a corresponding resource list from an offset value in said entry;

(d) for each said resource list, determining information about a resource on said circuit by sequentially reading data contained in said resource list and data located at addresses computed from offsets contained in said resource list;

(e) recording said resource information in a table accessible to said computer;

whereby, the configuration of resources on said circuit in said computer is determined.

2. A method of determining the configuration of resources on a circuit in a computer, as recited in Claim 1, further comprising recording error status in a table if said

format header was not found at one of said predetermined addresses.

3. A method of determining the configuration of resources on a circuit in a computer, as recited in Claim 2, wherein the end of each said resource directory is determined by the presence of an end of list indicator as an entry in the resource directory.

4. A method of determining the configuration of resources on a circuit in a computer, as recited in Claim 3, wherein the end of each said resource list is determined by an end of list indicator as an entry in the resource list.

5. A method of determining the configuration of resources on a circuit in a computer, as recited by Claim 4, wherein said set of predetermined addresses at which the beginning of said format header may exist consists of the highest address in each byte lane in a partition of address space assigned to each said slot.

6. A method of determining the configuration of resources on a circuit in a computer, as recited by Claim 5, wherein said information about a resource consists of a resource type, resource name, initialization routine, and, optionally, resource icon, and device driver.

7. A method of determining the configuration of resources on a circuit in a computer, as recited in Claim 6,

wherein said determination is made automatically at the time the computer is started.

8. In a computer having at least one slot for receiving circuits for coupling to said computer, a method of determining the configuration of resources of said circuits, comprising:

(a) for each said slot, examining a configuration memory, located at one of a set of addresses corresponding to the highest byte in each byte lane in an address partition assigned to said card slot, for the presence of a valid byte lane code;

(b) for each said slot in which said valid byte lane code has been found, examining a test pattern field at a predetermined offset from said byte lane code field for the presence of a valid test pattern;

(c) for each said slot in which a valid byte lane code and test pattern has been found, editing fields at predetermined offsets from said byte lane code to detect the presence of valid values;

(d) for each said slot in which a valid byte lanes code, valid test pattern, and in which valid values exist in said edited fields, determining an address in said computer's address space where a resource directory is located by calculating the sum of the address of a directory offset field and of the contents of said directory offset field, said directory offset field being a predetermined offset from the address of said byte lane code field;

(e) sequentially reading entries in said resource directory and determining an address of a corresponding resource list for each said entry from an offset value in said entry;

(f) for each said resource list, determining information about a resource on said circuit by sequentially reading data contained in said resource list and data located at addresses computed from offsets contained in said resource list;

(g) recording said information describing the resources on each said circuit in a data table accessible to said computer;

whereby, the configuration of resources on said circuit in said computer is determined.

9. A method of determining the configuration of resources on a circuit in a computer, as recited in Claim 8, wherein said editing of fields comprises examining a valid format identity field for valid values, examining a revision level field for valid values, and performing a cyclic redundancy check (CRC) to insure a CRC field contains a valid value.

10. A method of determining the configuration of resources on circuits in a computer, as recited in Claim 9, further comprising the step of recording errors in a data table accessible to said computer when valid values are not found for said byte lane code, said test pattern, and said edited fields.

11. A method of determining the configuration of resources on circuits in a computer, as recited in Claim 10, wherein said valid byte lane values are computed by setting a bit in the low nibble of the byte lane field for each byte lane used by said configuration memory and by setting the high nibble of said byte lane field to said low nibbles complement.