REPLACED SHEETS SG
☑ Description : paged 2 to paged 3
☑ Claim(s) : paged 16 to paged 18
☑ Drawing(s) : paged 1 to paged 7
☑ Abstract : paged 19 to paged

A COMMUNICATIONS ADAPTER

Field of the Invention.

The present invention relates generally to digital computer systems, and more specifically to adapters for interfacing host computer systems with serial communication lines.

Background of the Invention

Communications with the outside world are extremely important to modern computer systems. Some systems have several serial communications links which operate simultaneously. Controlling multiple links operating simultaneously can place demands on a system processor which degrade overall system performance.

One solution to this performance problem is to use "smart" communications adapters. These adapters handle all of the low level details of a communications session. The adapters communicate with their host system to transfer data in relatively large blocks. Both received data and data to be transmitted are transferred between the adapter and the host system using block transfers. Direct memory access (DMA) can be used for such transfers to further reduce the processing burden placed on the host central processor.

On order to provide multiple communications ports in a system having a limited number of adapters, several ports can be placed on a single adapter. However, this approach can lead to significant problems. Data and command handling for several independent ports can become quite complex, especially at high communications rates. When different speeds and protocols are used on the different communications ports, the problem becomes greatly exacerbated. It becomes difficult for the adapter to ensure that all communications ports are handled in a timely manner.

Disclosure of the Invention

Accordingly the invention provides a communications adapter for interfacing a plurality of communications ports with a host system, comprising: an interface to the host system; an intermediate memory; a processor for controlling operation of the adapter; a plurality of communications ports; a bus connecting said host interface, said intermediate memory, said processor and said communications ports; and means for scheduling execution of a plurality of communications tasks to be executed by said processor.

A single adapter can support multiple communications ports where each port operates independently of the others. The communications ports are handled in such a manner as to ensure that no data is lost on any port. In such a system, a scheduler executing on the adapter can schedule tasks at different priority levels, so that time-critical tasks are performed quickly enough to prevent data loss. Data to be transmitted or received through a communications port can be stored in buffers on the adapter, and data and command communications between the adapter and the host system are preferably performed over a DMA channel.

The invention also provides a method for controlling data transfer between a host system and a plurality of communications ports, comprising the steps of: defining a plurality of tasks for performing data transfer (or portions thereof) between the ports and an intermediate memory, and between the intermediate memory and the host system; assigning priorities to the defined tasks; scheduling tasks to be performed in response to events which occur at the ports and within the host system; and when a task is completed, selecting to be performed next a task having the highest priority of all scheduled tasks.

Preferably there is a table having an entry for each task, and the tasks are scheduled by writing information to the corresponding table entry. At least one entry in the table has a plurality of subentries each corresponding to a respective port, and the task is scheduled to be

This Sheet Has Been Replaced



performed for a port by writing information to the corresponding subentry. Preferably, at least one entry in the table represents a count of the number of times the corresponding task has been scheduled, and a task is scheduled by incrementing the value in the entry.

It is also preferred that the events which cause tasks to be scheduled comprises hardware interrupts raised by the host system or by the ports, or also by the execution of scheduled tasks. Each task is assigned a priority level, and the scheduled tasks having a higher priority level are performed before the scheduled tasks having a lower priority level.

Brief Description of the Drawings

Figure 1 is a block diagram of a host computer system;

Figure 2 is a block diagram of a preferred serial communications adapter for use with the system of Figure 1;

Figure 3 illustrates a preferred scheduler work table;

Figure 4 is a set of flow charts illustrating execution of tasks and interrupts on an adapter system;

Figure 5 illustrates contents of a memory included on an adapter; and

Figures 6 through 12 are flow charts illustrating operation of selected functions which can be scheduled in accordance with the method of the present invention.

Detailed Description

Referring to Figure 1, a computer system, referred to generally as a host system, is indicated by reference number 10. The system 10

This Sheet Has Been Replaced



includes a central processor 12 and a main memory 14 connected to a system bus 16. One or more user interface input/output devices 18 are also connected to the bus 16. These user interface devices 18 typically comprise a display and keyboard, and often include a pointing device such as a mouse.

A mass storage input/output device 20 is also preferably connected to bus 16, and interfaces the system 10 with one or more mass storage devices. These devices typically include magnetic or optical disk drives. Some systems do not include such mass storage devices 20, instead relying upon remote mass storage devices connected to a network. An adapter for serial input/output devices 22 is preferably connected to system bus 16. In a preferred embodiment, such adapter 22 provides communications between the system 10 and multiple serial communications ports. Various other devices may be included in the system 10 as known in the art.

The system of Figure 1 shows all of the input/output devices as being connected to the main system bus 16. Many high performance systems provide a high speed bus between the central processor 12 and main memory 14. In such systems, an input/output controller (not shown) is connected to the high speed bus, and in turn connects to the various input/output devices 18, 20, 22 through a separate input/output bus (not shown). In such systems, data transferred between main memory 14 and the various input/output devices is routed through, and controlled by, the input/output controller. The multiple-port serial adapter described below can function equally well in a system of either type. It is only necessary to provide an appropriate interface between the adapter and the bus to allow it to be used in many different types of computer systems.

Referring to Figure 2, a preferred serial input/output adapter 22 is shown. Adapter 22 is connected to the system bus 16 through a system bus interface 24. Interface 24 is connected to an adapter bus 26, which is a high speed bus. Also connected to adapter bus 26 are a central

processor 28 and an adapter memory 30. Central processor 28 can be a commercially available microprocessor, such as an INTEL 80186 processor. Adapter memory 30 is a computer system memory as known in the art, and is made available in an amount sufficient to support the functions described below. In typical applications, 512K bytes of adapter memory 30 are sufficient for four communications ports. Central processor 28 executes the functions of the adapter 22 using programs stored in the memory 30.

Also connected to adapter bus 26 are four serial port interfaces 32, 34, 36, and 38. Although four ports are shown in Figure 2, more can be included on a single adapter if desired. The number of ports which can be included on a single adapter card 22 is limited only by the processing power which is available to serve the ports, and the number of physical port connections which can be made to the adapter card 22.

A serial port DMA controller 40 is connected to the adapter bus 26, and is used to control DMA transfers between the ports 32, 34, 36, 38 and the adapter memory 30. Each port has two signal lines connected directly to the DMA controller 40 to indicate when data is available in a receive buffer and when a transmit buffer is available to accept data. The DMA controller 40 uses such signals to determine when to initiate a DMA transfer between a communications port and adapter memory 30.

The DMA controller 40 and the system bus interface 24 are both capable of acting as bus masters of the adapter bus 26 in order to perform DMA transfers. Bus system interface 24 and DMA controller 40 use DMA to transfer data to and from adapter memory 30. In addition, interface 24 can also become a bus master for system bus 16, and transfer data to and from main memory 14 using a DMA channel. In the embodiment described herein, a single DMA channel is used by system bus interface 24 to transfer data for all of the communications ports 32, 34, 36, and 38.

In order to control operation of the adapter 22, central processor 28 executes a number of different routines which will be described below. Of notable importance is a task scheduler routine which determines which task is to be executed next. Various routines for setting up DMA transfer to and from the system bus 16 through the system bus interface 24, setting up transmit and receive operations between the adapter memory 30 and the communications ports, and for handling various error conditions, are executed by the central processor 28.

Figure 3 shows a schedule work table 50 preferably used for scheduling tasks on the adapter. Different tasks operate at different priority levels, with priority levels 0-6 being shown in Figure 3. The number of priority levels actually used can be varied as needed to suit any particular implementation. Priority level 0 is the highest priority, with priority level 6 being the lowest priority.

Each priority level has an entry which is indicative of those tasks waiting to be performed. Priority levels 1-6 each consist of a bitmap having one entry corresponding to each communications port on the adapter. If any particular bit has a value of one, the corresponding port has the corresponding task scheduled.

The bits in the scheduler work table 50 are set by various interrupt handlers and by executing tasks as described in more detail below. When a hardware interrupt occurs, the appropriate handler determines which routine should be invoked to process it, and schedules such routine by setting a bit in the scheduler work table 50. When a routine executes, it can schedule other routines by setting bits in the table 50, and will generally unschedule itself by resetting its own bit.

The highest priority level, level 0, is used as a counter to indicate the number of currently scheduled tasks for that priority level. This difference in operation will be described in further detail below. In some implementations, it may be desirable to use other priority levels, or none at all, in this manner.

The highest priority level is the task which performs the system bus DMA transfers. Whenever a task needs to perform system bus DMA, it increments the level 0 counter and places necessary identifying information into a list in memory 30. Since only a single system bus DMA channel is available, only one system DMA task can operate at any given time. Any remaining tasks must wait until the currently executing system bus DMA task completes before they can be executed. Other tasks can be executed on the adapter while a system bus DMA task is executed as will be described below.

The level 1 task is called when a frame has been received over a communications port, and various steps must be taken to check the validity of the received frame. The receive data task causes a received frame to be examined, and any errors flagged. The receive data task also schedules a system bus DMA transfer of the received frame to the host.

The next priority task, level 2, is the transmit data task. This task initiates the transfer of data from the adapter memory 30 to the appropriate communications port. A transmit data task is scheduled when a frame of data for communications over a port has been assembled, or when a previous frame completes transmission.

The level 3 task is an error/status task which is invoked whenever an error appears on a communications port. The types of errors handled by this task preferably include line errors, such as unexpected changes in control signals, as well as data transmission errors.

Under circumstances which will be described below, various events which are generated by a port which is transmitting or receiving data must be queued to await later processing. Level 4 is a task which processes the entries in a port response queue. In a similar manner, level 5 is a task which processes commands which are sent to a port which have been queued to await later processing. Use of the port response queue and port command queue allows various types of processing

to be performed without waiting for a communications port to finish its current operation. For example, commands can be sent to a port when it is convenient for the host system, and they will reside in the queue until they can be applied to the port. Incoming events are stored in the port response queue until they are can be handled by the central processor 28 on the adapter 22.

The lowest priority level, level 6, is used to wake up tasks which have been put to sleep. Sometimes it is desirable to suspend operation of a task until a predefined event occurs. This task can be put to sleep as will be described below, and will remain suspended. When the predetermined event occurs, an interrupt handler associated with the event schedules the port response queue task, which in turn causes the appropriate bit to be set for level 6 of the scheduler work table 50. When the wakeup task is selected to be executed for that port, the suspended task resumes execution.

Figures 4(a), 4(b) and 4(c) illustrate the manner in which tasks are scheduled and executed. Referring to figure 4(a), the scheduler is seen to execute in an endless loop. A currently selected task, or process, is executed to completion 60. When execution of the task is complete, the scheduler selects the highest priority level which has any task scheduled 62. Within that priority level, the scheduler then selects a task to be executed. If more than one task is scheduled to be executed at the selected priority level, a round-robin strategy is used to determine which task is to be executed next. This ensures that all ports will be equally treated, and that no port will have a task executed at any given priority level until all other ports having a task scheduled at the same priority level have been serviced. After the task has been selected, control returns to step 60 and the selected task is executed.

Steps 62 and 64 operate very quickly, and preferably are made uninterruptable. This is because the various interrupts which may occur often schedule tasks by updating the scheduler work table 50, and this

activity would interfere with the selection process of steps 62 and 64. Process execution in step 60 is interruptible, and various hardware interrupts generated by the system and communications ports are used to schedule further tasks within the scheduler work table 50.

Figure 4(b) shows the general method by which an interrupt is handled. When a hardware interrupt occurs on the adapter 22, control of the central processor 28 is vectored to a general interrupt routine. This routine identifies the nature of the interrupt which occurred 66, and causes the appropriate interrupt handler to be executed 68. The interrupt handler which executes in step 68 is preferably very short, and is made non-interruptible. It typically clears the cause of the interrupt, copies one or two data items to an appropriate area where they can be found by one of the regularly scheduled tasks, and updates the scheduler work table 70 to cause the appropriate task to be scheduled. Thus, while a task is executing in step 60, additional tasks are scheduled in work table 50 by the various interrupt handlers which are executed.

The method of Figure 4(b) is used when the various tasks are made non-preemptable. In other words, an executing task is completed before another task is initialized by the scheduler. Since most tasks are fairly short, this does not present a problem in most systems. However, if desired, the various tasks can be made preemptable. If this is done, scheduling of a higher priority task by an interrupt causes the execution of the current task to be suspended and the higher priority task to be executed immediately. A flowchart of interrupt handling in such a situation is shown in Figure 4(c).

When a hardware interrupt occurs, the nature of the interrupt is identified 72 and an appropriate interrupt handler executed 74. As before, executing the interrupt handler causes the scheduler work table 50 to be updated 76. A check is then made 78 to see whether work having a higher priority than the currently executing task has been scheduled. If a higher priority task has not been scheduled 80, execution of the

current task resumes. If a higher priority task was scheduled during execution of the interrupt handler 74, the currently executing process is preempted 82 and the scheduler invoked again to initiate execution of the higher priority process.

Referring to Figure 5, some of the important data structures stored within the adapter memory 30 are shown. The major portion of memory 30 is preferably taken up by transmit/receive buffers 90. These buffers 90 are dynamically allocated from free space available within the memory 30. A list of free space available for the transmit/receive buffers 90 is maintained as known in the art.

A vector table 91 is allocated for use by the various tasks and interrupts. Stacks 92 are allocated for temporary data storage for each port independently. Thus, four stacks are allocated within memory 30 when four communications ports are provided on the adapter. A separate stack 93 is used by the scheduler. Each port also has a port control block 94 which contains pointers to various data structures which are accessed by the corresponding port. Port command queue 96 provides a FIFO queue for each port which contains commands sent to the port. Each command sent to a port is placed into a standard sized data structure referred to as a port command element, and the commands for each queue are placed into a circular buffer.

When a port command is executed which requires data to be transmitted to or from the host, further commands may not be executed for such port until the data transfer process is completed. In some systems, it is possible to use ping-pong buffers to transfer data through a port. In such cases, if one channel of a port is active, the other channel may be set up for the next transfer. Port commands waiting in the port command queue 96 may be executed in order to prepare the other buffer.

Each port also has a corresponding port response queue 98, which is comprised of a circular buffer in the same matter as the port command

queue 96. The port response queues 98 contain entries generated by the corresponding ports which must be processed by the system. If a system bus DMA transfer is being performed for a port, all such responses being generated by the port must be routed to the port response queue 98. Also, if a port is sleeping, to be awakened by the occurrence of a predetermined event, all responses generated by the port are routed into the appropriate response queue 98.

Also found within the adapter memory 30 is the executable code 100 for the scheduler, the various scheduled tasks, and the interrupt handlers. Various other locations, used for temporary variable storage as known in the art, are located in adapter memory 30, but are not specifically shown in Figure 5. Scheduler work table 50 is also contained in memory 30.

Referring to Figure 6, operation of the system bus DMA task is shown. As described above, the system bus DMA task uses the scheduler work table 50 differently from the other levels, in that the scheduler work table entry indicates the number of tasks which are scheduled rather than identifying the ports which have a task scheduled. When a system bus DMA task is scheduled, an indication of the requested task is stored in a separate data structure (not shown), and one of the scheduled system bus DMA tasks is selected according to any desired prioritizing scheme.

Once a system bus DMA task has been is initiated, other adapter processing may be performed while the transfer is taking place. Since the scheduler will attempt to schedule a level 0 task if possible, it is necessary to set the level 0 entry to 0 for the duration of the system bus DMA transfer. Thus, referring to Figure 6, the first step performed by a selected system bus DMA task is to save the current DMA count 108 as reflected in level 0 of the scheduler work table 50. A 0 entry is then made for level 0 of the scheduler work table 110, and the system bus DMA task is initiated 112. Once DMA is initiated 112, control can be returned to the scheduler to select another task for execution.

At some future time, the system bus DMA transfer will be completed 114. At this time, an additional task must be scheduled 116, and the scheduler work table level 0 entry is restored 118. The value placed back into level 0 of the table 50 is preferably decremented before it is restored.

If a transmit DMA has occurred, which moves data from the host system onto the adapter 22, it is necessary to remove any locks (described below) which have been placed on the port command queue. In addition, a port command queue task is scheduled. When a receive DMA, which transfers data from the adapter 22 to the host system, completes, a port response queue task is scheduled. In addition, a global flag which causes routing of response events for that port into the port response queue is cleared to allow port responses to be handled normally. All of these actions occur in step 116.

Figure 7 illustrates operation of the transmit data task. This task is scheduled when an end of frame interrupt indicates completion of a transmit operation through a port. The task first signals completion 120 by placing the relevant information into the port control block 94. The transmit buffer is then cleared and freed 122. The port command queue task is scheduled 124 by setting the appropriate bit in the scheduler work table 50. If another frame is ready for transmission 126, the transmission is initiated 128 and the task completes. If no frame is ready for transmission 126, the task simply returns.

Figure 8 shows the steps performed by the receive data task. First, a procedure which has been coded specifically to handle the communications protocol being used on the communications port is called 130. The protocol used on a communications port is indicated by appropriate values contained within the corresponding port control block 94. The steps shown in block 132 are actually performed by the protocol specific procedure. The received frame is processed 134 and any appropriate entries are added to the response queue 136. If errors

occur within the received data frame, an error/status task can be scheduled.

Figure 9 shows the steps performed by the error/status task. An error identifier is copied by the interrupt handler to the port control block 140 for the port in order to indicate to later tasks which access that port that an error has occurred. The error identifier indicates the type of error which has occurred and the corresponding possible error sources are checked 142. Once the source of the error has been identified, appropriate entries are added to the response queue 144 for that port. When these entries are eventually processed, the port errors will be indicated to the system 10.

Operation of the port response queue task is illustrated in Figure 10. A check is first made to see whether the queue is empty 150. If so, the flag which is used to automatically reroute port responses to the queue is cleared 152, causing port responses to be handled in the normal manner. The task then unschedules itself 154 from the scheduler work table 50 and completes. If there are entries in the queue 150, the next one is selected 156. A check is made 158 to see if the next entry is a response which requires a DMA transfer to be made to the host system 10. If so, the bus master DMA task is scheduled 160 and the current task is unscheduled 154. If the next entry is not a DMA transfer, a check is made to see whether a task is sleeping for the port 162. If not, the response is sent to the host 164. If a task is sleeping, the sleep/wakeup task is scheduled 166 and the current task is unscheduled 154.

Figure 11 illustrates the steps performed by the port command queue task. In general, this task selects the next entry in the port command queue 96 and causes it to be executed. The occurrence of various events may prevent execution of the next port command entry from occurring. The first step is to select the next entry 170 from the port command queue. If the next entry is not locked 172, a check is made to see whether a system bus DMA operation is required 174. If not, and the

port is otherwise available 176, transmission of the next data block is set up 177 and the transfer data task is scheduled 178. As described above, up to two transmits can be preprocessed if two ping-pong buffers are provided for each port.

In many cases, it is necessary to perform a system bus DMA transfer to transfer data onto the adapter 22 before the data can be transmitted. If the test of step 174 results in a YES response, it is necessary to schedule an appropriate system DMA activity 180 and lock the currently selected entry of the port command queue 182. The level 5 task for this port is then unscheduled 184. As described above in connection with Figure 6, completion of the system bus DMA task causes the lock set in step 182 to be removed, and the port command queue task for this port to be rescheduled. The next time through the process of Figure 11, the system DMA test of step 174 will result in a NO response. If the next entry is locked in step 172, or the port is not available in step 176, control transfers to step 184 and the level 5 entry for this port in the scheduler work table 50 is unscheduled 184.

Figure 12 illustrates the steps which are performed when a task puts itself to sleep and is subsequently awakened. A SLEEP_ENABLE variable is first set for the port 190. The task goes to sleep 191 and the current context is saved to the stack 192 for the port. Setting the SLEEP_ENABLE variable in step 190 causes any responses generated by the port to be routed to the port response queue 98. The task which is going to sleep is then exited and other processing occurs 194. While other processing occurs, the context which was saved in step 192 remains untouched on the port stack 92.

When new events are added to the port response queue, the port response queue task wakes up the sleeping task and the context is restored from the stack 196. A check is made 198 to determine whether the wakeup event was a terminal condition for which the task is waiting 198. If not, the task simply goes back to sleep 191. If the event actually causes the task to wake up, execution is resumed 200. When the

task completes, the SLEEP_ENABLE variable is reset. The task can put itself back to sleep at a later stage in its execution, which causes the described sequence of events to repeat.

Since waking a sleeping task is the lowest level task which can be scheduled, the system is guaranteed to have the context which was saved to the stack in step 192 as the top frame on the stack when the wakeup is received. A wakeup is scheduled by another task or interrupt, which sets the appropriate bit on level 6 on the scheduler work table 50.

The system described above provides for priority scheduling of various communications subtasks on the adapter. The number of priority levels and the number of communications ports which can be supported are easily expanded without changing the underlying scheduler. The adapter of the preferred embodiment supports two separate levels of DMA transfer. One level is the DMA transfer performed over the adapter bus 26, and the second level is the DMA transfer over the system bus 16 performed by the system bus interface 24.

Tasks have been prioritized by the preferred scheduler work table 50 so that the most time critical tasks have the highest priority. The transmit data and the receive data tasks, which must be performed in a timely manner to avoid data loss through the communications port, have a higher priority than handling of the various commands and responses placed in the queues 96 and 98. Tasks which put themselves to sleep, implying that time is not a critical factor, have the lowest priority of all. System bus DMA transfer tends to be the bottleneck because one resource is shared among all ports, so the system bus DMA task is given the highest priority. A mechanism is provided whereby other tasks are performed on the adapter 22 while a system bus DMA transfer is taking place. Since a round robin strategy is used for each priority level, no port becomes starved.

* C L M S *

CLAIMS

1. A method for controlling data transfer between a host system (10) and a plurality of communications ports (32,34), comprising the steps of:

defining a plurality of tasks for performing data transfer between the ports and an intermediate memory (30), and between the intermediate memory and the host system;

assigning priorities to the defined tasks;

scheduling tasks (70) to be performed in response to events which occur at the ports and within the host system; and

when a task is completed, selecting (62) to be performed next a task having the highest priority of all scheduled tasks.

2. The method of Claim 1, wherein a task is performed for all ports for which it is scheduled using a round robin strategy.

3. The method of Claim 1 or 2, wherein said scheduling step includes the step of:

providing a table (50) having an entry for each defined task, wherein tasks are scheduled by writing information to the corresponding table entry.

4. The method of Claim 3, wherein at least one entry in the table has a plurality of subentries each corresponding to a respective port, and wherein the task is scheduled to be performed for a port by writing information to the corresponding subentry.

5. The method of Claim 4, wherein each subentry comprises one data bit.

This Sheet Has Been Replaced

6. The method of Claim 3, 4 or 5, wherein at least one entry in the table represents a count of the number of times the corresponding task has been scheduled, and wherein a task is scheduled by incrementing the value in the entry.

7. The method of any preceding Claim, wherein the events which cause tasks to be scheduled comprise hardware interrupts raised by the host system or by the ports.

8. The method of Claim 7, wherein the events which cause tasks to be scheduled further comprise the execution of scheduled tasks.

9. A communications adapter (22) for interfacing a plurality of communications ports (32,34) with a host system (10), comprising:

an interface to the host system (24);

an intermediate memory (30);

a processor (28) for controlling operation of the adapter;

a plurality of communications ports (32,34);

a bus (26) connecting said host interface, said intermediate memory, said processor and said communications ports; and

means (28,30) for scheduling execution of a plurality of communications tasks to be executed by said processor.

10. The adapter of Claim 9, further comprising means for assigning each communications task a priority level, and wherein scheduled tasks having a higher priority level are performed before scheduled tasks having a lower priority level.

This Sheet Has Been Replaced

11. The adapter of Claim 9 or 10, wherein said scheduling means includes a table (50) having entries corresponding to said tasks, and wherein tasks are scheduled by writing information to the corresponding table entries.

12. The adapter of Claim 11, further comprising means for writing information to the table entries in response to interrupts generated by the host system and said communications ports.

13. The adapter of Claim 11 or 12, wherein executing tasks write information to the table entries.

14. The adapter of Claim 11, 12 or 13, wherein at least one table entry contains a plurality of bits each corresponding to a respective communications port, and wherein a task is scheduled to be performed for one of said ports by writing to its corresponding bit.

15. A computer system having a host system (10) including a central processor (12) and main memory (14); and an adapter (22) as claimed in any of claims 9 to 14 interfaced therwith.

This Sheet Has Been Replaced