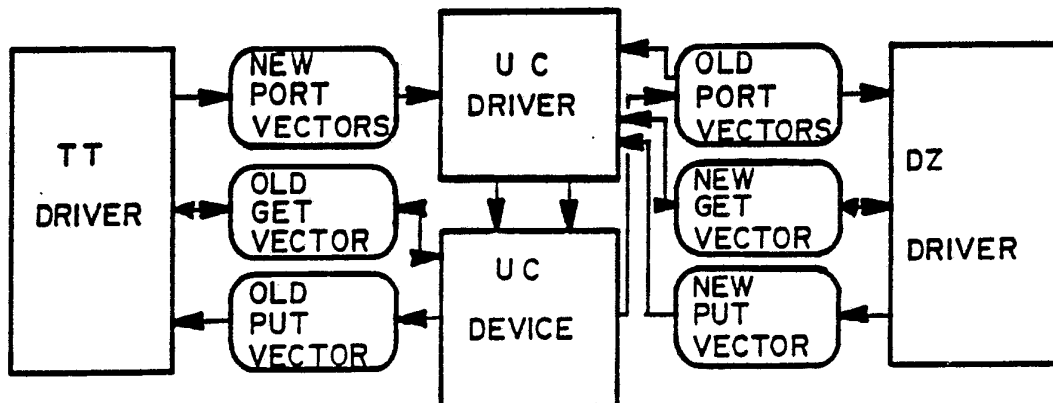




## INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

<p>(51) International Patent Classification <sup>4</sup> : <b>G06F 09/00 // G06F 11/00</b></p>	<p><b>A1</b></p>	<p>(11) International Publication Number:     <b>WO 89/ 03556</b> (43) International Publication Date:     20 April 1989 (20.04.89)</p>
<p>(21) International Application Number:    PCT/US87/02653 (22) International Filing Date:        8 October 1987 (08.10.87)  (71) Applicant: CLYDE, INC. [US/US]; 371 East 800 South, Orem, UT 84058 (US). (72) Inventor: CLYDE, Robert, A. ; 216 East 600 North, Orem, UT 84058 (US). (74) Agents: NYDEGGER, Rick, D. et al.; Workman, Ny- degger &amp; Jensen, American Plaza II, Third Floor, 57 West 200 South, Salt Lake City, UT 84101 (US).  (81) Designated States: AT (European patent), AU, BE (Eu- ropean patent), CH (European patent), DE (European patent), FR (European patent), GB (European pa- tent), IT (European patent), JP, LU (European pa- tent), NL (European patent), SE (European patent).</p>		<p><b>Published</b> <i>With international search report.</i> <i>With amended claims and statement.</i></p>

(54) Title: SYSTEM FOR EFFECTIVELY PARALLELING COMPUTER TERMINAL DEVICES



## (57) Abstract

In a digital computing system, in which terminal devices are connected to the system through terminal dependent device drivers coupled to terminal independent device drivers so that the operating system of the computer system need not be modified each time a terminal device is added or subtracted, a system is used for effectively paralleling an auxiliary terminal with a selected terminal of the system so that the selected terminal can be monitored by creating a user controlling driver and a user controlling device coupled thereto, and coupling the combination thereof between the terminal device drivers so that the output of the terminal independent device driver intended for the terminal dependent device driver passes through the user controlling driver and user controlling device before arriving at the terminal independent device driver and making the information passing through the user controlling driver available to the auxiliary terminal device.

***FOR THE PURPOSES OF INFORMATION ONLY***

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	FR	France	ML	Mali
AU	Australia	GA	Gabon	MR	Mauritania
BB	Barbados	GB	United Kingdom	MW	Malawi
BE	Belgium	HU	Hungary	NL	Netherlands
BG	Bulgaria	IT	Italy	NO	Norway
BJ	Benin	JP	Japan	RO	Romania
BR	Brazil	KP	Democratic People's Republic of Korea	SD	Sudan
CF	Central African Republic	KR	Republic of Korea	SE	Sweden
CG	Congo	LI	Liechtenstein	SN	Senegal
CH	Switzerland	LK	Sri Lanka	SU	Soviet Union
CM	Cameroon	LU	Luxembourg	TD	Chad
DE	Germany, Federal Republic of	MC	Monaco	TG	Togo
DK	Denmark	MG	Madagascar	US	United States of America
FI	Finland				

## SYSTEM FOR EFFECTIVELY PARALLELING COMPUTER TERMINAL DEVICES

1

BACKGROUND OF THE INVENTION1. Field of the Invention

5       The invention is in the field of methods and devices  
to enable the linking of a computer terminal device to  
one or more additional terminal devices so that each of  
the linked devices effectively operates in parallel to  
10 receive information from the computer or to input  
information into the computer.

2. State of the Art

15       Some computer systems are configured to allow  
several terminal devices to be connected effectively in  
parallel so that each of the parallel terminal devices  
20 receives identical information from the computer or each  
can supply information to the computer and to each  
other. For example, if two CRT input/output terminals  
are connected in parallel in such a system, any  
25 information from the computer will be displayed on both  
paralleled terminals. Also, in most instances either may  
be used to enter data or instructions into the computer  
and the entered data or instructions on one terminal will  
30 show on the other paralleled terminal.

Most computer systems today are designed to be  
connected to many terminal devices, such as many CRT  
35 displays, with each terminal device operating  
independently of the other terminal devices. In fact,  
most systems are designed specifically to prevent

1 terminal devices from being paralleled so that a person  
on one terminal cannot spy on another terminal and cannot  
interfere with the operation of such other terminal.  
5 With such systems, it is possible to electrically connect  
two or more terminals together in parallel, but once such  
electrical connection is made, the terminals are always  
connected in parallel. Further, unless an elaborate  
10 mechanical switching arrangement is used, a particular  
terminal cannot be easily paralleled to a desired one of  
a plurality of terminals. Such a switching system is  
completely impractical where terminals are located at  
15 physically separated locations such as in different parts  
of a city or country.

In many instances it may be desirable to effectively  
20 parallel two terminal devices on a selectable basis. For  
example, if an operator is having a problem with a  
particular program or particular operations in a program,  
rather than having a person more knowledgeable in  
25 operation of that program or a special program  
troubleshooter or instructor physically travel to the  
operator's terminal, or orally describe, such as by  
telephone, the problem and solution, that person could  
30 merely connect his terminal in parallel to the operator's  
terminal, find the problem, and show the operator the  
correct procedure. Another example where selectable  
parallel connection might be desirable is where a  
35 supervisor or other official may wish to observe for  
security reasons what a particular operator is doing.

1 With a parallel connection, the actions of the operator  
could be monitored unknown to the operator. In such  
instance, it is not necessary that the effective parallel  
5 connection provide for input from the auxiliary terminal,  
but merely for the providing of the same information flow  
between the CPU and monitored terminal to the auxiliary  
terminal. Further, rather than actually observing the  
10 operator, it may be desired to actually record and make a  
record or audit trail of an operator's actions to check  
at a later time, if necessary.

15 One computer system in wide use today is the VAX  
Computer system manufactured by Digital Equipment  
Corporation (DEC). This system uses what DEC calls the  
VMS operating system. Such system is different than most  
20 systems in use today in that rather than having the  
drivers for each terminal device a fixed portion of the  
operating system of the computer so that such operating  
system must be modified each time a terminal device is  
25 added or subtracted from the system, the VMS operating  
system provides a generalized or terminal independent  
device driver which is connected through specific or  
terminal dependent device drivers to specific terminal  
30 devices. Thus, it is not necessary to modify the  
operating system when terminal devices are added or  
subtracted, but to merely connect the terminal device to  
an appropriate specific terminal dependent device driver.  
35

1           There is currently no known way to effectively and  
easily provide selected parallel operation of two or more  
terminal devices with such an operating system.

5

SUMMARY OF THE INVENTION

10           According to the invention, a system of effectively  
connecting an auxiliary terminal device in parallel with  
a selected terminal device comprises a user controlling  
driver (UC Driver) and a user controlling device (UC  
15 Device) coupled together and coupled between the terminal  
independent device driver and the terminal dependent  
device driver associated with the terminal device to be  
paralleled so that the output of the terminal independent  
20 device driver intended for the terminal dependent device  
driver and a specific terminal device passes through the  
UC Driver and UC Device before arriving at the terminal  
dependent device driver and the output of the terminal  
25 dependent device driver from a specific terminal device  
and intended for the terminal device independent driver  
passes through the UC Driver and UC Device before  
arriving at the terminal independent device driver. In  
30 this way, all information passing to or from a particular  
terminal device passes through the UC Driver which stores  
such information and makes it available to the auxiliary  
35 terminal device.

1           The auxiliary terminal device can merely monitor the  
information either storing it, displaying it, or printing  
it, or can interact with the paralleled terminal so that  
5 information can be inputed to the computer at either  
terminal device. Thus, the auxiliary terminal device  
could be a CRT, hard copy device, or secondary storage  
device such as for example, magnetic disc, magnetic tape,  
10 or laser disc.

#### THE DRAWINGS

15           In the accompanying drawings, which illustrate an  
embodiment of the invention constituting the best mode  
presently contemplated for carrying out the invention in  
20 actual practice:

Fig. 1 is a block diagram showing the terminal  
device driving system in the existing Digital Equipment  
Corporation VMS operating system and represents the prior  
25 art environment for the invention;

Fig. 2, a block diagram showing a portion of prior  
art environment of Fig. 1, with additional blocks as  
created by the invention, but not connected into the  
30 blocks shown from Fig. 1;

Fig. 3, a block diagram showing the combination of  
the additional blocks of the invention interconnected  
with the blocks of the existing environment;  
35

1           Fig. 4, a block diagram showing additional blocks  
necessary to make use of the monitored terminal  
information and shows the program and information links  
5 for getting monitored data from the UC Device and UC  
Driver of Fig. 3 into a temporary buffer;

          Fig. 5, a block diagram showing additional blocks  
necessary to transfer the information from the temporary  
10 buffer as obtained in the system of Fig. 4 to any  
auxiliary terminal device, Figs. 4 and 5 together showing  
transfer of the monitored data to the auxiliary terminal;

          Fig. 6, a block diagram showing an alternate  
15 arrangement to perform the same function as performed by  
combined Figs. 4 and 5, to get the monitored data to an  
auxiliary terminal device;

          Fig. 7, a block diagram similar to Fig. 5, showing  
20 the program and information links required to transfer  
information from an auxiliary terminal device to a  
temporary buffer; and

          Fig. 8, a block diagram similar to Fig. 4, showing  
25 the program and information links required to transfer  
information from the temporary buffer to the UC Driver  
and UC Device, Figs. 7 and 8 together showing forcing of  
30 data from an auxiliary terminal to the CPU and monitored  
terminal.

35



1     DETAILED DESCRIPTION OF THE ILLUSTRATED EMBODIMENT

Although the invention may be used with any computer  
5 system using an operating system having independent  
device drivers and dependent device drivers for driving  
terminal devices, the invention will be described with  
particular reference to Digital Equipment Corporation's  
10 VMS operating system. Device drivers are explained in  
detail in the VAX/VMS Guide to Writing a Device Driver  
published by Digital Equipment Corporation.

15 As shown by Fig. 1, the standard Digital Equipment  
Corporation's (DEC's) VMS operating system is arranged  
with at least one terminal independent device driver, one  
such driver called TT Driver by DEC, which receives  
20 information from the central processing unit (CPU) of the  
system intended for a specific terminal device and  
receives information from specific terminal devices  
intended for the CPU. The terminal independent device  
25 driver is designed to be substantially terminal  
independent meaning that the information is directed to  
or from a particular terminal regardless of the  
particular physical, electrical, and software  
30 requirements of the terminal. The terminal independent  
driver contains all of the basic routines which all  
programs call through VMS for doing terminal input and  
35 output. The present VMS systems utilize a single  
independent device driver to drive all terminal dependent  
device drivers and all terminal devices.

1           Since the terminal independent device driver is  
substantially terminal device independent, in order to  
provide input or accept output from a particular  
5 terminal, such driver must work through a terminal  
dependent device driver. This is a driver which takes  
the information from the terminal independent device  
driver and puts it in a form necessary to operate the  
10 hardware terminal controller which actually operates the  
terminal itself. The terminal dependent device drivers  
are of various types dependent upon the particular  
hardware terminal devices to be operated by the driver.  
15 In DEC terminology such dependent device drivers may be,  
for example DZ Drivers, YC Drivers, or Console Drivers.  
Thus, if the hardware terminal controller to be operated  
20 is a DEC DZ-11 or DZ-32, the terminal dependent device  
driver is the DZ Driver. If the controller is a DMF-32,  
the terminal dependent device driver is the YC Driver.  
Other controllers will require other dependent device  
25 drivers. Here again, in DEC systems, one controller can  
control a number of actual terminals, and if all  
controllers used are of a single type, a single terminal  
dependent device driver may be used to drive all terminal  
30 controllers and all terminal devices. The terminal  
dependent device driver contains the routines necessary  
for actually placing output in a hardware register,  
getting input from a hardware register, handling  
35 interrupts, and handling other hardware dependent tasks.

1       As indicated above, a computer system may include  
many hardware terminal devices, such as CRT input/output  
terminals, printers, readers, or data storage devices,  
5 yet have a single hardware terminal controller, a single  
terminal dependent device driver and a single terminal  
independent device driver. In such instance, each actual  
terminal device has its own identifying data and any  
10 information from the CPU directed to a particular  
terminal device will also contain the identifying data  
for the terminal device to which it is directed. With  
the DEC VMS system, when the information directed to a  
15 particular terminal device reaches the terminal  
independent device driver, such driver looks up the  
identifying data by which the terminal dependent device  
20 driver identifies the terminal which has been identified  
by the CPU. The identifying data for the terminal  
dependent device driver is known in VMS as the Port  
Vectors and terminal Unit Control Block (UCB). These  
25 Port Vectors identify to the terminal independent device  
driver the corresponding terminal dependent device  
driver. The terminal UCB identifies the exact hardware  
terminal device to which the information from the CPU is  
30 to be directed. Thus, Figs. 1 and 2 show the terminal  
independent device driver connected through the Port  
Vector to the terminal dependent device driver indicating  
that information for a particular terminal device, passes  
35 by reason of the Port Vectors from the terminal  
independent device driver to the terminal dependent

1 device driver. The information then passes directly to  
the hardware terminal controller and the terminal. The  
address of the terminal UCB is always kept in a known  
5 general register so that it is available to any terminal  
device driver. Since the address of the UCB is always  
known and thus the UCB is always available to any of the  
terminal device drivers, the UCB is not shown in the  
10 drawings.

Information passing from the CPU to the terminal in  
the VMS system can be passed in one of two ways. The  
terminal independent device driver includes several  
15 registers which store information for direct transfer to  
the terminal dependent device driver. If the information  
to be passed is a single character, it can be placed in  
20 the general registers and passed directly from the  
terminal independent device driver to the terminal  
dependent device driver by use of the Port Vectors. If  
the information is more than a single character, such  
25 information to be transferred is stored in physical main  
memory and directions or a pointer to the location of the  
information in memory is placed in the general registers  
in the CPU and passed from the terminal independent  
30 device driver to the terminal dependent device driver  
through the Port Vectors. At the same time, control of  
the CPU is turned over to the terminal dependent device  
driver which then, through another communication  
35 direction referred to as the Get Vector, transfers the  
information from the CPU's memory directly to memory in

1 the hardware terminal controller; this transfer may be  
done by either the terminal dependent device driver or  
the hardware terminal controller. Whether one character  
5 or many characters are output, the Port Vectors and  
terminal UCB serve as a directing means to identify the  
proper terminal device and direct information to the  
terminal dependent device driver so that it reaches the  
10 proper terminal device.

In reverse fashion, when a terminal sends  
information to the CPU, it has to let the CPU know from  
which terminal the information is coming. When  
15 information from a terminal device reaches the terminal  
dependent device driver, such driver looks up the  
identifying information by which the terminal independent  
device driver identifies the terminal from which the  
20 information comes and through such identifying  
information, known as the terminal UCB, looks up the Put  
Vector in said UCB and then transfers the information to  
25 the terminal independent device driver through the Put  
Vector. As mentioned earlier, the address of the  
terminal device's UCB is always stored in a specific  
general register. Again, such information transfer can  
30 be accomplished in two ways. When the information is a  
single character in length, it is stored in general  
registers in the terminal dependent device driver and can  
be transferred directly from the terminal dependent  
35 device driver to the terminal independent device  
driver. If the information is longer than a single

1 character, then such information is stored in memory by  
either the terminal dependent device driver or the  
hardware terminal controller and directions or a pointer  
5 to the information is passed via the Put Vector through  
the general registers in the CPU to the terminal  
independent device driver which then, transfers the  
information to a temporary buffer controlled by the  
10 terminal independent device driver. In either case, the  
Put Vector and terminal UCB serve as directing means to  
identify the proper terminal device and direct  
information to the terminal independent device driver so  
15 that it is properly identified as to the terminal from  
which it came.

The Get Vector which in many instances facilitates  
20 two-way communication and information transfer between  
the two drivers is considered as part of each of the  
directing means which couple the two drivers together for  
information transfer.

25 The drivers in VMS, both device independent and  
device dependent, are separate and exist as separate  
executable images stored in separate disk files in the  
system. These drivers are created and exist as soft-  
30 ware. The hardware terminal controller and the terminals  
themselves represent actual hardware. The controller and  
the terminals are connected by a hardware communication  
link. When the system starts up, VMS automatically loads  
35 the drivers into various sections of system physical  
memory. New drivers can be added and loaded at

1 anytime. In the drawings, the drivers, which represent  
software, and the hardware are shown as rectangular  
boxes. Data structures which represent stored data, such  
5 as the Port Vectors, are represented with rounded  
corners.

While operation of the VMS system has been described  
in a general sense above, a more detailed description as  
10 to program structure and steps follows. Since the  
drivers are not bound into the operating system by a link  
editor, VMS requires all drivers to use position-  
15 independent code and to follow a rigid protocol in  
handling user requests and servicing device interrupts.  
Part of this protocol requires a description of various  
data structures for handling user requests and  
20 interrupts. Some of these structures are:

Channel Request Block (CRB) - Used to handle the  
allocation and transfer of data to the device hardware  
controller.

25 Unit Control block (UCB) - There is a UCB for each  
unit of a particular device type. The drivers use this  
block to store information about each unit. Such  
information is generally device dependent, but usually  
30 includes at least the following items: pointer to the  
CRB, device unit number, owner process identification,  
device characteristics, device status (on-line, busy,  
waiting for interrupt and/or timeout), device state  
35 (reading, writing, output stopped, etc.), information  
about I/O currently being handled, pointer to IRP

1 (described below) currently being processed, and saved  
registers and program counter if driver execution is  
suspended.

5 I/O Request Packet (IRP) - A driver can actually  
complete the processing of an I/O request while allowing  
the calling user process to continue executing. When  
this occurs, the I/O request is placed in an IRP and is  
10 then delivered by the I/O scheduler to a STARTIO routine  
in the device driver code. When the I/O is completed,  
the driver causes the operating system to set an event  
flag. The user process can then check this flag to  
15 verify that the I/O has in fact completed. The IRP  
includes the following information: requesting process  
identification, address of requested device's UCB, type  
20 of request (read, write, set characteristics, etc.),  
address of user buffer, length of user buffer, and event  
flat to set when I/O request is completed.

As explained above, the individual terminal inde-  
25 pendent device drivers and the terminal dependent device  
drivers are not connected directly together but communi-  
cate via a VAX jump subroutine through special Vector  
tables (i.e., Port Vectors, Get Vector, and Put Vector)  
30 which are included at a known position in each driver.  
The address of the Vector tables for a particular device  
is always placed in the UCB. The Port Vector table  
contains the Port Vectors which point to port driver  
35



1 routines in the terminal device dependent driver. The  
special port driver routines which are of particular  
interest for this invention are:

5 Start Terminal Output - This routine is used to  
initiate the terminal controller to ensure that it will  
cause an output interrupt to occur. It also is used to  
make sure that the data passed in the general registers  
10 is ready to output as soon as the interrupt occurs.  
Consequently, this routine has the following input from  
the general registers:

15 Input: Register 5 contains address of terminal's  
UCB. Register 3 contains either the character to output  
or the address of the data in memory to output and  
register 2 contains the number of characters to output.  
20 A condition code is set to indicate which case applies.

Start Terminal DMA Output - This routine is func-  
tionally the same as the Start Terminal Output  
routines. However, it is able to set up a direct memory  
25 access (DMA) transfer to be performed by the hardware  
terminal controller. Although the input is specified  
through different registers, it remains the same, except  
more than one character is always output if this routine  
30 is called.

In addition, the terminal dependent device driver  
can call routines in the terminal independent device  
driver in order to retrieve more characters to output and  
35 to give it input characters from the terminal. The

1 addresses for these routines are stored in the Get and  
Put Vectors in the terminal device's UCB. These routines  
are described below:

5 Get Output Data - This routine is called whenever an  
output interrupt occurs and there is no more data to  
output to the terminal. Whenever an output interrupt  
occurs, the output interrupt routine in a driver like  
10 DZDRIVER immediately outputs any data that was passed  
earlier to the Start Terminal Output routine. Once this  
data has been outputted and another output hardware inter-  
15 rupt occurs, the interrupt routine calls the get output  
data routine via the Get Vector to get more data to  
output. If there is no more data to output, then the  
interrupt routine does not ask the controller to generate  
20 another output interrupt, thereby terminating the output  
sequence. The Get Output Data routine has the following  
input and output passed through the general registers:

25 Input: Register 5 contains the address of the  
terminal's UCB.

Output: Register 3 contains either the character to  
output or the address of the data in memory to output and  
register 2 contains the number of characters to output.  
30 A condition code is set to indicate which case applies.

Put Input Data - Whenever input is available from a  
terminal, the hardware controller generates a terminal  
35 interrupt. This in turn causes VMS to call the terminal  
input interrupt routine in the terminal dependent device  
driver. This routine then reads the character from the

1 controller and then calls the Put Input Data routine via  
the Put Vector to let the terminal independent device  
driver process, echo and buffer the character.  
5 Eventually, the terminal independent device driver passes  
a stream of terminal input to the user process requesting  
it. The Put Input Data routine has the following input  
and output:

10 Input: Register 3 contains the input character from  
terminal. Register 5 contains the address of the  
terminal's UCB.

15 Output: The Put Input Data routine calls the Get  
Output Data routine in order to output any echo for the  
typed characters. Thus, the output from the Put Input  
Data routine is the same as the output from the Get  
20 Output Data routine.

Thus, as shown in Fig. 1, the output information  
from VMS to a terminal passes through a terminal indepen-  
dent device driver through the Port Vectors or Get Vector  
25 which direct the information to the appropriate terminal  
dependent device driver which, in turn, directs the  
information to the appropriate hardware terminal  
controller which actually causes the information to  
30 appear at the appropriate output terminal. With  
information from the terminal, it passes through the  
hardware terminal controller to the terminal device  
dependent driver which directs the information through  
35

1 the appropriate Put Vector to the terminal independent  
device driver which then directs the information on to  
the CPU.

5 When it is desired to monitor a particular terminal  
device using another terminal device, herein referred to  
as an auxiliary terminal device, it is necessary to  
effectively connect the auxiliary terminal device in  
10 parallel with the device to be monitored. As used in  
this application, the term "effectively parallel" a  
terminal device does not mean a true electrical parallel  
connection of the auxiliary terminal device and the  
15 terminal device being monitored, but merely a hook-up or  
linking of the two terminal devices so that information  
from the CPU to the monitored terminal device appears at  
20 both terminal devices. In some instances, the linking  
will also provide for information entered at the  
auxiliary terminal device to be directed to the CPU and  
appear on both terminal devices.

25 In order to create the parallel linking of the  
desired terminal devices, it is necessary to create  
another driver device, herein arbitrarily referred to as  
a user controlling driver or UC Driver, and to create  
30 another device for each terminal to be monitored, herein  
arbitrarily referred to as a user controlling device or  
UC Device. Figure 2 shows a UC Driver and UC Device. In  
order to direct information from the terminal independent  
35 device driver to the UC Driver, it is necessary to insert  
a set of New Port Vectors and a New Get Vector for the

1 output of the terminal independent device driver.  
Similarly, in order to direct information from the  
terminal dependent device drivers to the UC Driver, it is  
5 necessary to insert a New Put Vector at the output of the  
terminal dependent device drivers. These are shown as  
blocks labeled "New Port Vectors" and "New Put Vector."  
The UC Driver is a software driver not inherent or  
10 existing in the VMS software and is created by a  
programmer in accordance with the invention which may be  
loaded using the normal driver loading procedure provided  
15 by VMS. Once the driver is loaded, it remains in main  
memory until the system is taken down. When needed, the  
UC Driver creates a UC Device and the necessary New Port  
Vectors, New Put Vector and New Get Vector.

20 UC Driver allows a system manager to create as many  
UC Devices as he desires. Each of these software devices  
is capable of monitoring any terminal on the system.  
However, each device can only monitor one terminal at a  
25 time. Therefore, if ten terminals need to be monitored  
simultaneously then the system manager should create 10  
UC Devices. This can be done through normal DEC utili-  
ties provided with VMS. The UC Driver is basically a  
30 program device and is shown as a rectangular box in the  
drawing while the individual UC Devices are data struc-  
tures and shown with rounded corners.

35 In general terms, the UC Driver is set up to take  
the place of and act as the terminal dependent device  
driver when looking from the terminal independent device

1 driver and to take the place of and act as the terminal  
independent device driver when looking from the terminal  
dependent device driver. Initially, the UC Driver  
5 creates a UC Device and the New Port Vectors, New Put  
Vector and New Get Vector. These are set up and exist  
separately from the system. This is shown in Figure 2  
where the system of Figure 1 exists as in Figure 1 and  
10 the UC Driver, UC Device, New Port Vectors, New Put  
Vector, and New Get Vector exist separately as shown.

In order to monitor a desired terminal, the links as  
15 shown in Figures 1 and 2 are broken so that the UC  
Driver, UC Device, New Port Vectors, New Put Vector and  
New Get Vector are inserted and linked as shown in Figure  
3. With this arrangement, all information from the CPU's  
20 operating system to be directed to the terminal device  
being monitored is directed from the terminal independent  
device driver through the New Port Vectors, and in some  
cases, as described above, using the Old Get Vector, to  
25 UC Driver. It is then directed through the UC Device and  
the Old Port Vector, sometimes also using the New Get  
Vector, to the terminal dependent device driver. Infor-  
mation from the terminal is directed by the New Put  
30 Vector, and in some cases, as described above, using the  
New Get Vector, to UC Driver. It is then directed  
through the UC Device and the Old Put Vector, again  
sometimes also using the Old Get Vector, to the terminal  
35 independent device driver. As the information passing in  
either direction between the CPU and the terminal passes

1 through the UC Driver, the UC Driver sets up the informa-  
tion so that it can be made available to an auxiliary  
terminal device. This usually will take the form of at  
5 least one memory buffer into which the information is  
placed and a program in the system that accesses the  
memory buffer and transfers the information to the  
auxiliary terminal. Where it is desired to use the  
10 auxiliary terminal not only to monitor a terminal but  
also to provide input into the system, the program will  
also provide for taking input from the auxiliary terminal  
and providing it to the UC Driver for further transmis-  
15 sion to the CPU.

We now turn to a more detailed discussion of the UC  
Driver, UC Device, and their operation and linking into  
20 the system. Each UC Device has a UCB associated with it  
as mentioned above. However, no hardware is associated  
with the device. The UCB does contain the following  
additional information:

25 Address of monitored terminal's UCB  
Monitored terminal's Old Get Vector  
Monitored terminal's Old Put Vector  
Address of Monitored terminal's Old Port Vector Table  
Address of buffer to hold terminal output  
Size of buffer to hold terminal output  
Current position pointer within output buffer  
30 Address of buffer to hold terminal input  
Size of buffer to hold terminal input  
Current position pointer within output buffer

35

1           In addition, the UC Driver maintains a list of the  
UCB for each monitored terminal and the UCB for each  
corresponding UC Device. This list is used at times to  
5 find out which UC Device UCB belongs with a given  
terminal UCB.

          In order to use a UC Device, a user program just  
calls and performs a normal VMS ASSIGN system service as  
10 described in the VAX/VMS System Services Guide published  
by Digital Equipment Corporation. This service assigns a  
channel number that is to be used by the user program in  
15 conjunction with all I/O calls to the UC Device. As with  
all I/O calls, VMS automatically uses this channel number  
to Figure out the appropriate UCB associated with the I/O  
request and passes the address of this UCB on to UC  
20 Driver.

          Different I/O operations like Read, Write, Set Up  
Monitoring and Take Down Monitoring are performed using  
the QIO system service provided by VMS again as described  
25 in the VAX/VMS Systems Service Guide. The user program  
simply passes a function code indicating which particular  
I/O service it desires from the device driver. The  
device driver protocol as described in the VAX/UMS Guide  
30 to Writing A Device Driver as referred to earlier allows  
a driver to define a variety of different functions and  
the addresses of the subroutines within the driver which  
35 are designed to handle those functions.



1           Whenever a program wishes to cause UC Driver to  
begin monitoring a given terminal, it assigns a free UC  
device and then performs a QIO call with the following  
5 input.

Function Code = Set Up Monitoring  
UC device identifier (VMS channel number)  
Name of terminal to monitor

10           UC Driver then breaks the logical link and inserts  
itself between the terminal independent device driver and  
the terminal dependent device driver. If insertion was  
successful, it returns a success status to the user;  
15 otherwise, it tells the user than an error occurred.

UC Driver actually breaks the normal logical link in  
VMS to move from Figure 2 to Figure 3 as follows:

1. It calls a VMS routine which takes the  
20 terminal name for input and returns the address to  
the corresponding terminal UCB as output.

2. The program allocates a section of memory  
large enough to hold the terminal's Port Vector  
25 table (it is pointed to by the terminal UCB).

3. The program copies the terminal's Port  
Vector Table to the new section of memory. This  
will become the New Port Vector Table.  
30

4. The address for the terminal's Port Vector  
table is stored in the UC device's UCB. This is now  
referred to as the Old Port Vector Table.

35

1           5. The addresses for the terminal's Start  
Terminal Output and Start Terminal DMA Output  
routines in the New Port Vector table are changed to  
5 point to UC Driver's own UC Start Terminal Output  
and UC Start DMA Terminal Output routines.

10           6. The terminal UCB and the UC Device's UCB  
are added to UC Driver's list of monitored  
terminals.

15           7. UC Driver allocates two memory buffers,  
one to hold the data being outputted to the terminal  
and another to hold the data being inputted from the  
terminal. The addresses of these buffers are stored  
in the UC Device's UCB. The starting addresses of  
these buffers are also placed in the current posi-  
20 tion pointers in the UC Device's UCB, thereby  
indicating that the buffers are empty.

25           8. The driver temporarily disables interrupts  
at this point to keep the terminal device it is  
monitoring from interrupting the CPU while the  
terminal's UCB is in an unstable state.

30           9. The address of the Port Vector table in  
the terminal UCB is changed to point to the New Port  
Vector table. Thus, TT Driver will call UC Driver's  
Start Terminal Output routines rather than the  
terminal dependent device driver's routine.

35           10. The address of the Get Output Data routine  
(Get Vector) and the address of the Put Input Data  
routine (Put Vector) that are stored in the

1 terminal's UCB are saved in the UC Device's UCB as  
the Old Get Vector and the Old Put Vector. Then UC  
Driver changes the addresses stored in the  
5 terminal's UCB so that UC Driver's UCGet Output Data  
and UCPut Input Data routines will be called  
instead. These changed addresses in the terminal's  
UCB are the New Get Vector and the New Put Vector.

10

11. Interrupts are reenabled.

At this point, the link has been broken and all data  
between the terminal independent device driver and the  
terminal will flow into UC Driver.

15

Once the connections have been made as in Figure 3  
and as described above, all terminal output passes  
through one of the UC Driver subroutines: UCStart  
20 Terminal Output, UCStart Terminal DMA Output, and UCGet  
Output Data. In order to emulate the normal Start  
Terminal Output and Start Terminal DMA Output routines,  
UC Driver's routines preserves the input passed to the  
25 routines and calls the device dependent driver's Start  
Terminal Output and Start Terminal DMA Output routines,  
just before exiting. UC Driver knows where these  
routines are because it stored the addresses to the Old  
30 Port Vector table in the UC Device's UCB. When the UC  
Start Terminal Output and UC Start Terminal DMA Output  
routines are called with the address of the terminal's  
UCB, UC Driver uses that address to look up the corres-  
35 ponding UC Device's UCB in UC Driver's list of monitored  
terminals. Now with the UC Device's UCB, it can find the

1 address of the Old Port Vector table for the terminal and  
finally through this table it can get the address of the  
appropriate Start Terminal Output routine in the terminal  
5 dependent device driver.

A significant amount of output data also flows  
through the UCGet Output Data routine. This routine is  
called by the terminal dependent device driver in order  
10 to get more data to output. As in the case of the Start  
Terminal Output routines, UC Driver figures out the  
corresponding UC Driver's UCB using UC Driver's list of  
monitored terminals. It then uses the Old Get Vector in  
15 the UC Device's UCB to call the Get Output Data routine  
in the terminal independent device driver which returns  
additional data to output as described before. UC Driver  
20 carefully preserves this output so that when it exits,  
the output is available to the calling terminal dependent  
device driver just as though the driver had called the  
terminal independent device driver's Get Output Data  
25 routine.

Although terminal output data flows through three  
routines in UC Driver, terminal input and any resulting  
echo back flows through only one routine, UC Put Input  
30 Data. This routine is entered with the same input as the  
terminal independent device driver's Put Input Data  
routine which is described earlier. As in the other  
three routines, UC Driver first looks up in the corres-  
35 ponding UC device's UCB for the given terminal UCB. It  
then uses the Old Put Vector in the UC device's UCB to

1 call the Put Input Data routine in the terminal indepen-  
dent device driver which returns any data to echo back to  
the terminal. UC Put Input Data carefully preserves the  
5 output which contains the echo back (if any) from the  
input. In this way, it appears to the device-dependent  
driver that it had just called the TT Driver Put Input  
Data routine directly.

10 The reason for passing all data through the UC  
Driver is to make such data available to an auxiliary  
terminal device, thus effectively paralleling the two  
terminal devices. For this purpose, the information  
15 passing through the UC Driver is stored in memory buffers  
associated with the UC Device and such information is  
then accessed and sent to the auxiliary terminal  
20 device. The memory is accessed and the information sent  
to the terminal device by means of a program.

As described earlier, the UCStart Terminal Output  
and UCStart Terminal DMA Output are both called with  
25 input parameters containing the data to output to the  
terminal. Before UC Driver exits, it records this infor-  
mation in the output buffer pointed to by the UC device's  
UCB. This buffer is shown as the Output Buffer in  
30 Figures 4, 6, and 8. Also, after UCGet Output Data calls  
TT Driver's Get Output Data routine, via the Old Get  
Vector it now has access to the terminal output returned  
by this routine and it records this output in the same  
35 output buffer. Similarly, after UCPut Input Data calls

1 TT Driver's Put Input Data routine via the Old Put  
Vector, it now has access to the echo back and records  
this echo back in the same output buffer.

5 The UCPut Input Data is called with general register  
3 containing the input character. This character is  
placed in the input buffer shown as the Input Buffer in  
Figures 4, 6, and 8.

10 Data is always placed into the buffer starting at  
the current position pointer which is stored in the UC  
Device's UCB. When data is placed into either of these  
15 buffers, the corresponding pointer to the current posi-  
tion is updated to point to the next free byte in the  
buffer. When both buffers are full, data will be  
discarded. However, if the user program accessing the UC  
20 Device empties the buffers on a regular basis, this will  
not happen.

Figures 4, 5, and 6 show general user program  
arrangements for accessing the information in the Input  
25 and Output Buffers and feeding such information to an  
auxiliary terminal device while Figures 7 and 8 show a  
general user program arrangement for accepting informa-  
tion from an auxiliary terminal device and forcing it  
30 into the Input and Output buffers so that it is supplied  
to the CPU as if it had come from the monitored  
terminal.

35 As shown in Figure 4 the user program, indicated as  
box labeled Program, is linked to the operating system  
input/output interface, here labeled VMS I/O Interface.

1 The VMS I/O Interface is linked to the UC Driver which is  
linked to the UC Device. Such linkage is shown by solid  
lines between the blocks. The UC Device is shown as  
5 having accesses to the Input Buffer and the Output  
Buffer. The access is shown by the broken lines. Thus,  
the Input buffer is an information structure that the UC  
Driver has access to through the UC Device, also an  
10 information structure. As described above, as  
information passes through the UC Driver and UC Device,  
UC Driver causes the information to be placed into the  
Input Buffer or Output Buffer as appropriate. To access  
15 such information, the program works through the VMS I/O  
interface to cause the UC Driver to transfer the informa-  
tion, if any, from the Input Buffer (information from the  
20 terminal to the CPU) to a First Temporary Buffer. Once  
in the First Temporary Buffer, the program, again through  
the VMS I/O interface, causes the information to be  
transferred to the Buffer. Moving to Figure 5, the  
25 Buffer, Program, and VMS I/O Interface are the same as  
shown in Figure 4, but have been separated for ease of  
illustration. The Program causes the information now in  
the Buffer to be transferred to a Secondary Temporary  
30 Buffer from where it is directed by the program through  
the VMS I/O Interface and the terminal independent device  
driver (TT Driver) to the terminal device in normal VMS  
manner as shown in Figure 1. Thus, in Figure 5, the  
35 Terminal Device is represented as a data structure. It  
actually includes everything to the right of TT Driver in

1 Figure 1 and the information from the Secondary Temporary  
Buffer is transferred via the TT Driver, Port Vectors and  
Get Vector, Terminal Dependent Device Driver, and Hard-  
5 ware Terminal Controller to the desired auxiliary  
terminal. After information from the Input Buffer of  
Figure 4 has been transferred, information from the  
Output Buffer is similarly transferred and then the  
10 process is repeated as long as new information is placed  
by UC Driver in either of the Input or Output Buffers.  
Figures 4 and 5 together show one way in which a program  
can access and cause the information stored in the Input  
15 and Output Buffers to be monitored by an auxiliary  
terminal device.

While the program described above takes information  
20 from the input buffer first and then from the output  
buffer, a program could take information from the output  
buffer first. It is currently preferred to look at the  
Input buffer first because generally there will be less  
25 information going into the Input buffer than into the  
Output buffer. This seems to provide a more equitable  
distribution in looking at both buffers. Also, while an  
input and output buffer memory is used and is preferred,  
30 a single memory buffer could also be used.

Figure 6 shows an alternate, and more direct way in  
which information from the Output and Input Buffers may  
be transferred to an auxiliary terminal device such as a  
35 disk or other secondary memory device for direct  
storage. In this embodiment, the Program through VMS I/O



1 Interface and through direct linkage with UC Driver, sets  
up a direct access to the Input Buffer and Output Buffer  
by the Program and the VMS I/O Interface so that  
5 information in the Output Buffer is transferred directly  
to the Disk Driver and placed in disk memory and  
similarly, information in the Input Buffer is transferred  
directly to the Disk Driver and placed in disk memory.  
10 The embodiment of Figure 6 is used when a terminal is to  
be monitored such that the information passing to or from  
the terminal is stored for later use. It is not used for  
interactive use of an auxiliary terminal device.

15 Figures 7 and 8 are similar to Figures 4 and 5 in  
that they show the same block arrangement, but show  
information access links differently. Figures 7 and 8  
20 show the linkage for forcing information from an  
auxiliary terminal into UC Driver from where it is sent  
to the CPU and appears on the monitored terminal. Figure  
7 shows the program linked to the VMS I/O Interface which  
25 is linked to TT Driver. As in Figure 5, the Terminal  
Device block is indicated as a data structure and  
includes the Put Vector, Get Vector, Terminal Dependent  
Device Driver, Hardware Terminal Controller and Terminal  
30 as shown in Figure 1. The terminal in this case is the  
auxiliary terminal. The information fed from the  
auxiliary terminal is transferred through VMS I/O Inter-  
face and TT Driver to the Secondary Temporary Buffer from  
35 where it is transferred to the Buffer. There, moving to  
Figure 8, the information is transferred from the Buffer

1 to the First Temporary Buffer where it is then treated as  
input from the monitored terminal by UC Driver and UC  
Device and placed in the Input Buffer generally by  
5 calling the UC Put Input Data routine. As that input is  
echoed back from the CPU, it goes into the Output Buffer  
and also appears on the monitored terminal as information  
entered by the terminal into the CPU.

10 From this general description of the program  
function, any person skilled in the computer programming  
art and familiar with VMS programming could set up the  
specific program necessary to access and monitor the  
15 information. The actual access by the program is  
initiated by doing a Q10 request with a read Function  
Code. The request passes the following output:

20 Input: Function Code = Read  
Assigned channel number for UC Device  
Address of user buffer to receive data  
Address of place to return extended status

Output: User buffer contains data read  
Extended status contains number of bytes  
read and a bit indicating whether the  
25 data read is terminal input or output

The user program uses the extended status word to  
determine how many bytes were read from the UC Driver and  
whether the data read is terminal input or output.

30 When the UC Driver receives a read request of this  
type, it follows these steps:

1. Disables Terminal Interrupts. A terminal  
interrupt could mess up the Terminal Input and  
35 Output buffers.

1           2.    If there is any data in the Terminal Input  
Buffer, UC Driver moves all of the data from the  
Terminal Input Buffer up until the current position  
5           pointer into the user program's read buffer and sets  
the status bit indicating that the data is terminal  
input.    The current position pointer in the UC  
device's UCB for the Terminal Input Buffer is reset  
10          to point to the beginning of the buffer to indicate  
that it is empty.

          3.    Otherwise, if there is no data in the  
Terminal Input Buffer, UC Driver moves all of the  
15          data from the Terminal Output Buffer up until the  
current position pointer into the user program's  
read buffer and sets the status bit indicating that  
20          the data is terminal output.    The current position  
pointer for the Terminal Output Buffer is reset to  
point to the beginning of the buffer to indicate  
that it is empty.

25          4.    UC Driver returns the number of bytes read  
in the buffer for extended status.    If no bytes were  
read, then a zero is returned.

30          5.    Interrupts are reenabled and the I/O call  
is completed.

          With the user program merely accessing the data, the  
data may be stored in secondary memory as a log file or  
audit trail for later use, or the data may be displayed  
35          on a CRT terminal or printed out on on a hardcopy  
device.

1       Where input from the auxiliary terminal is to be  
directed into the computer system a user program can  
force input to the process running on the user's  
5 terminal. This input looks just like the monitored user  
had typed that input at his own terminal, even though he  
didn't. A user program forces input of this type by  
issuing a Q10 call to the UC Device with the following  
10 input parameter:

Input:       Function Code = Write  
              Assigned channel number for UC Device  
              Data to write  
              Number of bytes to write

15

When UC Driver receives a request with a write  
function code it performs the following steps:

1. Look up the corresponding Terminal UCB for  
20 the UC Device and places this in general register 5  
as input for calling the Put Input Data routine.
2. Disable terminal interrupts.
3. Move a single byte from the data to write  
25 Q10 parameter into the register 3 as input parameter  
for Put Input Data.
4. Call UCPut Input Data which eventually  
causes TT Driver to think that a character has just  
30 been typed on the monitored terminal. Also, UCPut  
Input Data correctly buffers the echo back. See the  
previous description of the UC Put Input Data  
35 routine.

- 1           5. Move the next byte from the data to write  
          Q10 parameter into the input parameter for Put Input  
          Data. If there is no next byte, then go to step  
5           6. Otherwise loop back to step 4.  
          6. Reenable terminal interrupts.

          As described earlier, UC Driver just discards data  
          once the Terminal Output Buffer is full. However, in  
10          order to avoid this situation a feature may be built into  
          UC Driver which allows it to automatically turn off the  
          output to the terminal device until the Terminal Output  
15          buffer is emptied.

          In the terminal dependent device driver there is a  
          routine called Stop Output which stops terminal output.  
          The address of this routine is found in the Port Vector  
20          Table. Whenever, the output buffer is two-thirds full,  
          UC Driver calls this routine in order to stop output.  
          The only input this routine requires is the address of  
          the terminal's UCB. Once the buffer is almost empty, UC  
25          Driver calls another routine called Resume Output (its  
          address is also found in the Port Vector Table) which  
          starts the terminal output up again. This routine also  
          requires the address of the terminal's UCB as input.

30          When it is desired to merely monitor a terminal  
          device a program can be used which runs in the background  
          and records all of the data passing between a monitored  
          terminal and VMS. This data is stored in a log file (on  
35          disk or some other secondary memory device) that can be  
          kept for documentation or security purposes. The program

1 does this by executing a "Set up" Q10 call to UC Driver  
giving the terminal to monitor and then doing "Read" Q10  
calls to UC Driver in order to read the data passing  
5 between the terminal and VMS. The data the program reads  
is then outputed to the log file.

When it is desired to interact with a terminal  
device, a program can be used which does a "Set Up" Q10  
10 call to set up monitoring on a specified terminal. After  
this it does "Read" Q10 calls to retrieve the data going  
to and from the monitored terminal. This data is then  
displayed on the specified user's terminal so that he can  
15 watch what is happening.

In addition, the program also provides a mechanism  
so that if the user hits a character on his terminal, the  
20 program issues a "Write" Q10 call to the UC Driver so  
that the character is sent to the monitored user's  
process just as though the user had hit the character  
himself.

25 When a user program has finished monitoring a  
particular terminal, it can cause UC Driver to disconnect  
U.C. Driver and reconnect the normal link between the TT  
Driver and the terminal device dependent driver. It does  
30 this by issuing a Q10 call with the "take down" function  
code to UC Driver. As input the user program passes the  
VMS channel number assigned to the UC device.

35

1           When UC Driver executes the take down function it  
performs the following steps:

- 5           1. Remove the monitored terminal's UCB and  
the corresponding UC Device's UCB from UC Driver's  
list of monitored terminals.
2. Disable interrupts.
3. Restore the addresses saved in the UC  
10          Device's UCB to the terminal's UCB for the Get  
Vector, Put Vector the Port Vector table.
4. Reenable Interrupts. At this point the  
15          terminal is no longer being monitored and the  
logical link has been reconnected.
5. Deallocate the memory buffer used to hold  
the New Port Vector table created when the logical  
20          link was originally broken.
6. Deallocate the memory buffers used to hold  
the terminal input and output data.

          The invention also contemplates the method of  
25          connecting, in a computer system as described, an  
auxiliary terminal device effectively in parallel with a  
selected terminal device of the system so that the  
selected terminal device can be monitored by the  
30          auxiliary terminal. The method includes the steps of  
creating a UC Driver and a UC Device coupled to the UC  
Driver and coupling that combination in between the  
35          terminal independent device driver and the terminal  
dependent device driver associated with the terminal  
device to be paralleled so that the output of the

1 terminal independent device driver intended for the  
terminal dependent device driver passes through the UC  
Driver and UC Device before arriving at the terminal  
5 independent device driver and making the information  
passing through the UC Driver available to an auxiliary  
terminal device.

Whereas this invention is here illustrated and  
10 described with specific reference to an embodiment  
thereof presently contemplated as the best mode of  
carrying out such invention in actual practice, it is to  
be understood that various changes may be made in  
15 adapting the invention to different embodiments without  
departing from the broader inventive concepts disclosed  
herein and comprehended by the claims that follow.

20

25

30

35



1

CLAIMS

I Claim:

1. In a digital computing system having a central  
5 processing unit with an operating system, at least one  
terminal independent device driver, at least one terminal  
dependent device driver, a plurality of terminal devices  
each coupled to an associated one of the at least one  
10 terminal dependent device drivers, first directing means  
coupling the at least one terminal independent device  
driver with one of the at least one terminal dependent  
device drivers so that information from the CPU's  
15 operating system directed to a particular terminal device  
is properly identified for the intended terminal device  
and is directed to the appropriate terminal dependent  
20 device driver for that terminal device, second directing  
means coupling the at least one terminal dependent device  
driver with the at least one terminal independent device  
driver so that information from the particular terminal  
25 device is properly identified and directed to the  
terminal independent device driver and then to the CPU's  
operating system, a system for connecting an auxiliary  
terminal device effectively in parallel with a selected  
30 terminal device comprising a user controlling driver; a  
user controlling device coupled to the user controlling  
driver, the user controlling driver and user controlling  
device combination being coupled between the terminal  
35 independent device driver and the terminal dependent  
device driver associated with the terminal device to be

1 paralleled so that the output of the terminal independent  
device driver intended for the terminal dependent device  
driver passes through the user controlling driver and  
5 user controlling device before arriving at the terminal  
dependent device driver and the output of the terminal  
dependent device driver intended for the terminal  
independent device driver passes through the user  
10 controlling driver and user controlling device before  
arriving at the terminal independent device driver; and  
means in the user controlling driver for making the  
15 information passing therethrough available to an  
auxiliary terminal device.

20

25

30

35

1           2. A system for connecting an auxiliary terminal  
device effectively in parallel with a selected terminal  
device according to claim 1, wherein the first directing  
5 means is positioned to direct information from the user  
controlling device to the terminal dependent device  
driver; wherein new first directing means are provided to  
direct information from the terminal independent device  
10 driver to the user controlling driver; wherein the second  
directing means is positioned to direct information from  
the user controlling device to the terminal independent  
15 device driver, wherein new second directing means are  
provided to direct information from the terminal  
dependent device driver to the user controlling driver,  
and wherein the user controlling driver directs  
20 information to the user controlling device.

          3. A system for connecting an auxiliary terminal  
device effectively in parallel with a selected terminal  
25 device according to claim 1, wherein the means in the  
user controlling driver for making information available  
to an auxiliary terminal device includes a memory  
associated with the user controlling driver for storing  
30 information passing therethrough.

35

1           4. A system for connecting an auxiliary terminal  
device effectively in parallel with a selected terminal  
device according to claim 3, wherein means is included  
5 for accessing the memory and providing the information in  
the memory to the auxiliary terminal.

10           5. A system for connecting an auxiliary terminal  
device effectively in parallel with a selected terminal  
device according to claim 4, wherein the means for  
accessing the memory and providing the information to the  
auxiliary terminal is a computer program.  
15

20           6. A system for connecting an auxiliary terminal  
device effectively in parallel with a selected terminal  
device according to claim 5, wherein the program causes  
the information stored in the memory associated with the  
user controlling driver to be transferred to a first  
temporary buffer memory, then to be transferred from the  
25 first buffer memory to a second buffer memory, then to be  
transferred from the second buffer memory by reason of  
the terminal independent device driver to the auxiliary  
terminal in normal manner.  
30

35

1           7. A system for connecting an auxiliary terminal  
device effectively in parallel with a selected terminal  
device according to claim 5, wherein the program causes  
5 the information stored in the memory associated with the  
user controlling device to be transferred directly from  
such memory to the auxiliary terminal device.

10           8. A system for connecting an auxiliary terminal  
device effectively in parallel with a selected terminal  
device according to claim 7, wherein the auxiliary  
terminal device is a secondary memory device.  
15

          9. A system for connecting an auxiliary terminal  
device effectively in parallel with a selected terminal  
20 device according to claim 8, wherein the secondary memory  
device is a disc memory device.

          10. A system for connecting an auxiliary terminal  
25 device effectively in parallel with a selected terminal  
device according to claim 3, wherein the memory  
associated with the user controlling driver is an input  
memory buffer for storing information passing through the  
30 user controlling driver from the monitored terminal  
device to the CPU and an output memory buffer for storing  
information passing through the user controlling driver  
from the CPU to the monitored terminal device.  
35

1

11. A system for connecting an auxiliary terminal device effectively in parallel with a selected terminal device according to claim 10, wherein means is included for alternately accessing the memory buffers and providing the information in the accessed memory buffer to the auxiliary terminal.

10

12. A system for connecting an auxiliary terminal device effectively in parallel with a selected terminal device according to claim 11, wherein the input buffer is accessed prior to accessing the output buffer.

15

13. A system for connecting an auxiliary terminal device effectively in parallel with a selected terminal device according to claim 2, wherein means is additionally provided for supplying information from the auxiliary terminal device through the new second directing means to the user controlling device and on to the CPU and the terminal being monitored in normal fashion.

30

35

1           14. A system for connecting an auxiliary terminal  
device effectively in parallel with a selected terminal  
device according to claim 13, wherein the means for  
5 supplying information from the auxiliary terminal to the  
user controlling device is a computer program.

10           15. A system for connecting an auxiliary terminal  
device effectively in parallel with a selected terminal  
device according to claim 14, wherein the program causes  
the information from the auxiliary terminal device to be  
transferred to a second temporary buffer memory, then to  
15 be transferred from the second temporary buffer memory to  
a first temporary buffer memory from the second temporary  
buffer memory, then to be transferred from the first  
20 temporary buffer memory to the user controlling driver.

25           16. A system for connecting an auxiliary terminal  
device effectively in parallel with a selected terminal  
device according to claim 1, wherein means is  
additionally provided for supplying information from the  
auxiliary terminal through the user controlling driver to  
the CPU and the terminal being monitored.

30

35

1           17. In a digital computing system having a central  
processing unit with an operating system, at least one  
terminal independent device driver, at least one terminal  
5           dependent device driver, a plurality of terminal devices  
each coupled to an associated one of the at least one  
terminal dependent device drivers, first directing means  
10          coupling the at least one terminal independent device  
driver with one of the at least one terminal dependent  
device drivers so that information from the CPU's  
operating system directed to a particular terminal device  
15          is properly identified for the intended terminal device  
and is directed to the appropriate terminal dependent  
device driver for that terminal device, second directing  
means coupling the at least one terminal dependent device  
20          driver with the at least one terminal independent device  
driver so that information from the particular terminal  
device is properly identified and directed to the  
terminal independent device driver and then to the CPU's  
25          operating system, a method for connecting an auxiliary  
terminal device effectively in parallel with a selected  
terminal device comprising the steps of creating a user  
controlling driver; creating a user controlling device  
30          coupled to the user controlling driver; coupling the user  
controlling driver and user controlling device  
combination between the terminal independent device  
driver and the terminal dependent device driver  
35          associated with the terminal device to be paralleled so  
that the output of the terminal independent device driver



1 intended for the terminal dependent device driver passes  
through the user controlling driver and user controlling  
device before arriving at the terminal dependent device  
5 driver and the output of the terminal dependent device  
driver intended for the terminal independent device  
driver passes through the user controlling driver and  
user controlling device before arriving at the terminal  
10 independent device driver; and making the information  
passing through the user controlling driver available to  
an auxiliary terminal device.

15

20

25

30

35

## AMENDED CLAIMS

[received by the International Bureau on 5 April 1988 (05.04.88)  
original claims 1 - 17 replaced by new claims 1 - 50 ( 17 pages)]

1

1. A method of controlling the movement of data  
between a first driver program and a second driver program,  
the first and second driver programs included in an  
operating system for a digital computing system, the method  
comprising the steps of:

5

generating a third driver program;

10

identifying an original set of vectors used by the  
first and second driver programs to establish a  
communication link between each other;

15

generating a new set of vectors to establish  
communication links between (a) the first driver  
program and the third driver program and (b) the second  
device driver program and the third driver program;

inserting the new set of vectors in the locations  
of the original set of vectors; and

20

communicating data between the first and third  
device driver programs and between the second and third  
driver programs by way of the communication links  
established by the insertion of the new set of vectors  
such that all data passing between the first and second  
driver programs flows through the third driver program,  
the flow of data being controlled by the third driver  
program.

25

30

35

1           2.    A method as defined in claim 1 wherein the digital  
computing system further comprises an auxiliary terminal  
device and wherein the method further comprises the step of  
connecting the auxiliary terminal device to the digital  
5    computing system so as to monitor all data input to and  
output from the selected terminal device associated with the  
second driver program.

10           3.    A method as defined in claim 1 wherein the digital  
computing system further comprises an auxiliary terminal  
device and wherein the method further comprises the step of  
connecting the auxiliary terminal device to the digital  
computing system so as to monitor all data input to the  
selected terminal device associated with the second driver  
15    program.

20           4.    A method as defined in claim 1 wherein the digital  
computing system further comprises an auxiliary terminal  
device and wherein the method further comprises the step of  
connecting the auxiliary terminal device to the digital  
computing system so as to monitor all data output from the  
selected terminal device associated with the second driver  
program.

25           5.    A method as defined in claim 1 wherein the digital  
computing system further comprises an auxiliary terminal  
device and wherein the method further comprises the step of  
forcing input to the first driver program and to the second  
driver program from the auxiliary terminal.

30           6.    A method as defined in claim 1 wherein the step of  
generating a third driver program comprises the step of  
generating a third driver program which operates  
independently of the operating system.

35

1           7.    A method as defined in claim 1 wherein the step of  
generating a third driver program comprises the step of  
providing an executable code listing which emulates the  
first driver program when conducting input and output with  
5           the second driver program.

          8.    A method as defined in claim 1 wherein the step of  
generating a third driver program comprises the step of  
providing an executable code listing which emulates the  
10           second driver program when conducting input and output with  
the first driver program.

          9.    A method as defined in claim 1 wherein the digital  
computing system further includes a memory and wherein the  
15           method further comprises the step of storing the original  
set of vectors in the memory.

          10.   A method as defined in claim 1 wherein the digital  
computing system further comprises an auxiliary terminal  
20           device, a plurality of second driver programs, and a  
plurality of terminal devices, each second driver program  
associated with one terminal device, the method further  
comprising the step of effectively paralleling an auxiliary  
terminal device with any one of the plurality of second  
25           driver programs.

30

35

1           11.    A method as defined in claim 1 wherein the digital  
              computing system comprises a plurality of terminal devices  
              and a memory and wherein the method further comprises the  
              steps of:

5                 providing a first location in the memory for  
                  storing input to the third driver program;

                  providing a second location in the memory for  
                  storing output from the third driver program;

10                providing pointers indicating the next available  
                  byte in the first and second locations;

                  providing a pointer to a third location in the  
                  memory associated with a selected terminal device; and

                  providing a flow control flag for controlling the  
                  flow of data through the third driver program.

15

              12.    A method as defined in claim 1 wherein the digital  
              computing system comprises a buffer located in memory and  
              wherein the method further comprises the step of storing  
              input to the third driver program in the buffer.

20

              13.    A method as defined in claim 1 wherein the digital  
              computing system comprises a CPU and wherein the step of  
              inserting the new set of vectors comprises the step of  
              causing the execution of the CPU to jump to the third driver  
25                program.

25

              14.    A method as defined in claim 1 wherein the digital  
              computing system comprises an auxiliary terminal device and  
              wherein the method further comprises the step of providing a  
30                user program for (a) accessing the data flowing through the  
              third driver program and for (b) making the data available  
              to an auxiliary terminal device.

35

1           15.    A method as defined in claim 14 wherein the user  
program comprises the steps of:

              directing the third driver program to place a data  
word in the buffer; and

5                outputting the data word to the auxiliary terminal  
device.

              16.    A method as defined in claim 15 wherein the user  
program further comprises the steps of:

10                monitoring the buffer;

              halting the input to the buffer from the second  
driver program when the buffer is nearly full;

              directing a routine which is called from the user  
program to read the data in the buffer; and

15                resuming the input to the buffer from the second  
driver program after the user program has read the data  
in the buffer.

              17.    A method as defined in claim 14 wherein the user  
20 program comprises the steps of:

              accepting input from the auxiliary terminal  
device;

              placing the input in an allocated portion of the  
memory associated with the third driver program; and

25                forcing the input to both the first driver program  
and the second driver program.

              18.    A method as defined in claim 1 wherein the digital  
computing system comprises a memory and wherein the method  
30 further comprises the step of saving the original set of  
vectors in the memory.

1           19. A method as defined in claim 18 further comprising  
the steps of:

          breaking the communication link between the first,  
second, and third driver programs, such that no data  
5           flows through the third driver program;

          removing the new set of vectors from the locations  
of the original set of vectors;

          replacing the original set of vectors in their  
original locations; and

10           reestablishing the communication link between the  
first and second driver programs.

          20. A method as defined in claim 1 wherein the digital  
computing system comprises a selected terminal and wherein  
15           the step of inserting the new set of vectors comprises the  
steps of:

          disabling interrupts from a selected terminal  
device;

          inserting the new set of vectors; and

20           enabling interrupts from the selected terminal  
device.

          21. A method as defined in claim 1 wherein the digital  
computing system comprises an addressable memory and wherein  
25           the step of inserting the new set of vectors comprises the  
step of moving an address associated with a third driver  
program communication routine into the locations in an  
allocated portion of memory associated with the second  
driver program.

30           22. A method as defined in claim 21 wherein the  
allocated portion of memory structure comprises a unit  
control block.

35

1           23. A method as defined in claim 1 wherein the first  
driver program comprises a terminal independent device  
driver and the second device driver program comprises a  
terminal dependent device driver and wherein the step of  
5 inserting the new set of vectors comprises the step of  
moving an address location of at least one communication  
routine associated with the third driver program into the  
locations for a vector associated with the terminal  
independent device driver and the terminal dependent device  
10 driver.

15

20

25

30

35



1           24. A method of programming a digital computing system  
so as to connect an auxiliary terminal device in parallel  
with a selected terminal device, wherein the digital  
computing system includes a CPU, a memory for storing  
5           executable instructions and data, the selected terminal  
device, an auxiliary terminal device, and an operating  
system for controlling the flow of data between the CPU and  
the selected terminal device, and wherein the operating  
system includes a first device driver associated with the  
10          CPU, a second device driver associated with the selected  
terminal device, and a data communication pathway connecting  
the first and second device drivers, the method comprising  
the steps of:

15                   generating a driver program adapted for  
communicating with both the first and second device  
drivers;

                  allocating a portion of the memory for receiving  
and storing data passed between the CPU and the  
selected terminal device;

20                   breaking the communication pathway and inserting  
in the pathway the driver program;

                  reestablishing the communication pathway between  
the first and second device drivers such that all data  
flows through the driver program without any apparent  
25                   interruption at the CPU or at the selected terminal  
device; and

                  placing the data passed between the CPU and the  
selected terminal device in a location in the allocated  
portion of memory whereby the data is accessible by the  
30                   auxiliary terminal device.

25. A method as defined in claim 24 further comprising  
the step of connecting the auxiliary terminal device to the  
digital computing system so as to monitor all data input to  
35                   and output from the selected terminal device.

1

26. A method as defined in claim 24 further comprising the step of connecting the auxiliary terminal device to the digital computing system so as to monitor all data input to the selected terminal device.

5

27. A method as defined in claim 24 further comprising the step of connecting the auxiliary terminal device to the digital computing system so as to monitor all data output from the selected terminal device.

10

28. A method as defined in claim 24 further comprising the step of forcing input to the first device driver and the second device driver from the auxiliary terminal device.

15

29. A method as defined in claim 24 wherein the step of generating a driver program comprises the step of generating a driver program which is executed independently of the operating system.

20

30. A method as defined in claim 24 wherein the step of allocating a portion of memory comprising the step of allocating sufficient space in the memory to store a plurality of vectors associated with the driver program.

25

31. A method as defined in claim 24 wherein the digital computing system comprises a plurality of second device drivers, and a plurality of selectable terminal devices, each second device driver associated with a selectable terminal device, and wherein the method further comprises the step of selectively connecting in parallel one auxiliary terminal device with any one of the selectable terminal devices.

30

35

1           32. A method as defined in claim 24 wherein the step  
of allocating a portion of the memory comprises the step of  
allocating space in the memory for containing pointers to  
vectors used to maintain the communication pathway between  
5 the driver program and the first device driver and the  
second device driver.

10           33. A method as defined in claim 24 wherein the step  
of breaking the communication pathway and inserting in the  
pathway the driver program comprises the step of causing the  
execution of the CPU to jump to the driver program.

15           34. A method as defined in claim 24 wherein the method  
further comprises the step of providing a user program and  
wherein the user program comprises the steps of:

          reading a data word received from the first device  
driver or the second device and stored in the allocated  
portion of memory;

          placing the data word in a buffer; and

20           outputting the data word to the auxiliary terminal  
device.

          35. A method as defined in claim 34 wherein the driver  
program further comprises the steps of:

25           monitoring the buffer;

          halting the input to the buffer from the selected  
terminal device when the buffer is nearly full;

          directing a routine which is called from the user  
program to read the data in the buffer; and

30           resuming the input to the buffer from the selected  
terminal device and the CPU after the user program has  
read the data in the buffer.

1           36. A method as defined in claim 24 wherein the method  
further comprises the step of providing a user program and  
wherein the user program comprises the steps of:

5                 accepting input from the auxiliary terminal  
device;

                  placing the input in the allocated portion of  
memory; and

10                 directing the driver program to force the input to  
both the first device driver and the second device  
driver.

                  37. A method as defined in claim 24 further comprising  
the steps of:

15                 breaking the communication pathway such that no  
data flows from the first device driver and the second  
device driver;

                  removing the driver program from the communication  
pathway; and

20                 reestablishing the communication link between the  
first device driver and the second device driver.

                  38. A method as defined in claim 24 wherein the  
communication pathway is maintained by an original set of  
vectors and wherein the step of breaking the communication  
25                 pathway and inserting a driver in the pathway comprises the  
steps of:

                  disabling interrupts from the selected terminal  
device;

30                 overwriting the original vectors which establish  
the communication pathway between the first and second  
device drivers with new vectors which point to the  
driver program; and

                  enabling interrupts from the selected terminal  
device.

35

1           39. A method as defined in claim 38 wherein the step  
of overwriting the original vectors comprises the step of  
moving the addresses of the communication routines  
associated with the driver program into the locations of the  
5 original vectors.

          40. A method as defined in claim 39 wherein the first  
device driver comprises a terminal independent device driver  
and the second device driver comprises a terminal dependent  
10 device driver and wherein the step of moving the addresses  
of the communication routines comprises the step of moving  
the locations of the communication routines into the vector  
locations associated with the terminal independent device  
driver and the terminal dependent device driver.

15

20

25

30

35

1           41. A method of programming a digital computing system  
including at least one terminal device, an auxiliary  
terminal device, an operating system having a terminal  
independent device driver program and at least one terminal  
5           dependent device driver program associated with each  
terminal device, the terminal independent device driver  
program and the terminal dependent device driver program  
having a communication link established between each other  
by way of an original set of vectors, the method comprising  
10          the steps of:

                generating a user driver program structured to (a)  
emulate the terminal dependent device driver program  
when communicating with the terminal independent device  
driver program and to (b) emulate the terminal  
15          independent device driver program when communicating  
with the terminal dependent device driver program;

                allocating a portion of memory to store the  
original set of vectors and to store data passing  
between the terminal independent device driver program  
and the terminal dependent device driver programs;  
20

                generating a new set of vectors to establish a  
communication link between (a) the terminal independent  
device driver program and the user driver program and  
(b) the terminal dependent device driver program and  
the user driver program;  
25

                establishing a communication link between the user  
driver program and the terminal independent device  
driver program and the terminal dependent device driver  
program by moving the new set of vectors into the  
30          location of the original set of vectors such that all  
data passing between the terminal independent device  
driver and the terminal dependent device driver flows  
through, and the flow may be controlled by, the user  
driver program; and

35

1 moving the data passing through the user driver  
program to a location where it may be monitored by an  
auxiliary terminal device.

5 42. A method as defined in claim 41 further comprising  
the step of forcing input to the terminal independent device  
driver program and the terminal dependent device driver  
program from the auxiliary terminal device.

10 43. A method as defined in claim 41 wherein the step  
of generating a user driver program comprises the step of  
generating a user driver program which method operates  
outside of the operating system.

15 44. A method as defined in claim 41 further comprising  
the step of saving the original vectors.

45. A method as defined in claim 44 further comprising  
the steps of:

20 disabling the flow of data from the terminal  
independent device driver program and the terminal  
dependent device driver program;

25 removing the communication link between the user  
driver program and the terminal independent device  
driver program and the terminal dependent device driver  
program; and

30 replacing the original set of vectors to  
reestablish the communication link between the terminal  
independent device driver program and the terminal  
dependent device driver program so as to restore the  
original communication link therebetween.

35 46. A method as defined in claim 41 further comprising  
the step of selectively connecting in parallel one auxiliary  
terminal device with any one of the terminal devices.

1

47. A method as defined in claim 41 further comprising the step of providing a user program, the user program comprising the steps of:

5

directing the user driver program to place a data word in a buffer; and  
outputting the data word to the auxiliary terminal device.

10

48. A method as defined in claim 47 wherein the user program further comprises the steps of:

monitoring the buffer;

halting the input to the buffer from a selected terminal dependent device driver program when the buffer is nearly full;

15

directing a routine which is called from the user program to read the data in the buffer; and

resuming the input to the buffer from the selected terminal dependent device driver program after the user program has read the data in the buffer.

20

49. A method as defined in claim 41 further comprising the step of providing a user program, the user program comprising the steps of:

25

accepting input from the auxiliary terminal device;

placing the input in the allocated portion of memory;

30

forcing the input to both the terminal independent device driver program and the terminal dependent device driver program.

35



1

50. A method for effectively paralleling an auxiliary terminal device with a selected terminal device, the auxiliary terminal device and the selected terminal device both connected to a digital computer, the digital computer including a CPU, an operating system, a terminal independent device driver, a plurality of terminal dependent device drivers, and a memory, the method comprising the steps of:

5

allocating a UC device data structure in the

10

memory;

generating a UC driver program comprising the steps of:

15

calling a subroutine to return the address of a unit control block associated with a first terminal dependent device driver which is associated with the selected terminal;

allocating sufficient space in the memory to hold a first port vector table associated with the first terminal dependent device driver;

20

copying the first port vector table to the allocated portion of memory to create a second port vector table;

storing the address of the first port vector table in a unit control block contained in the UC device;

25

changing the second port vector table pointers which point to (a) a start terminal output routine and (b) a start direct memory access output routine to point to their corresponding subroutines contained in the UC driver program;

30

adding the address of the unit control block associated with the first terminal dependent device driver and adding the address of the unit control block contained in the UC device to a list of monitored devices;

35

1           allocating a second block of memory for use as  
input and output buffers and to store the address of  
the input and output buffers in the unit control block  
contained in the UC device;

5           disabling interrupts from the selected terminal;  
          changing the pointers contained in the unit  
control block associated with the first terminal  
dependent device driver to point to the second port  
vector table;

10          copying the address of a get data routine, which  
retrieves data from the terminal independent device  
driver and makes that data available to the terminal  
dependent device driver, from the unit control block  
associated with the first terminal dependent device  
15          driver to the unit control block contained in the UC.  
device;

          copying the address of a put data routine, which  
moves data from the terminal dependent device driver to  
the terminal independent device driver, from the unit  
20          control block associated with the first terminal  
dependent device driver to the unit control block  
contained in the UC device;

          modifying two pointers in the unit control block  
associated with the first terminal dependent device  
25          driver such that the two pointers point to the get data  
routine and the put data routine associated with the  
unit control block contained in the UC device; and  
          enabling interrupts from the selected terminal.

30

35

## STATEMENT UNDER ARTICLE 19

Applicant believes that the new claims submitted herewith more accurately and distinctly point out the inventive concepts which Applicant wishes to claim. As is readily apparent, the originally filed claims were all directed to an apparatus, whereas the new claims submitted herewith are appropriately directed to a method for programing a digital computer.

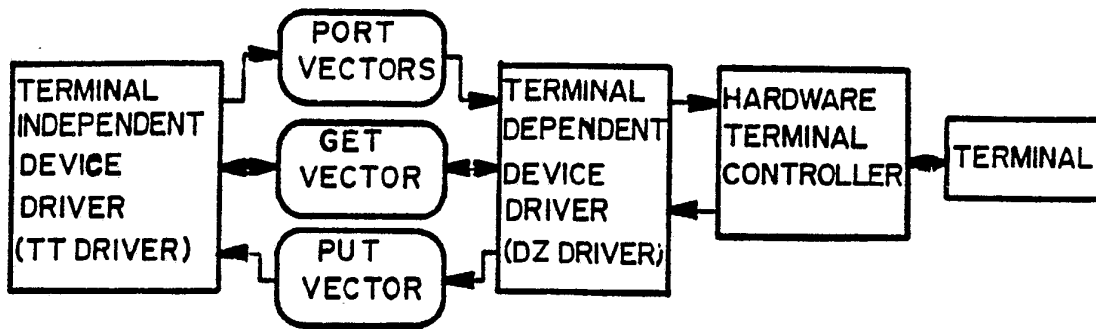


FIG. 1

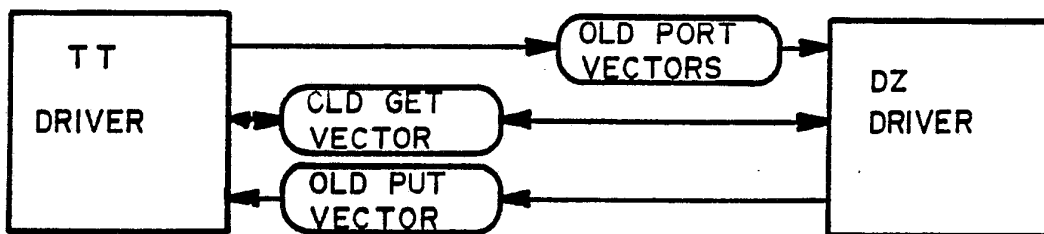
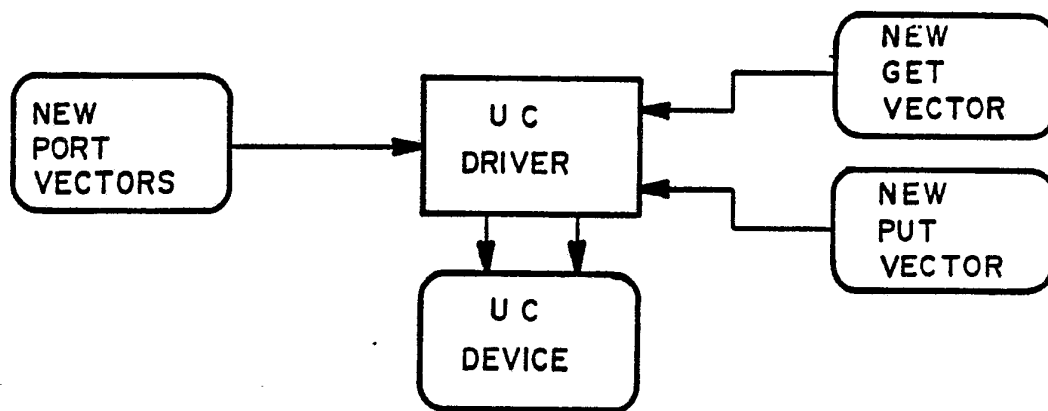


FIG. 2

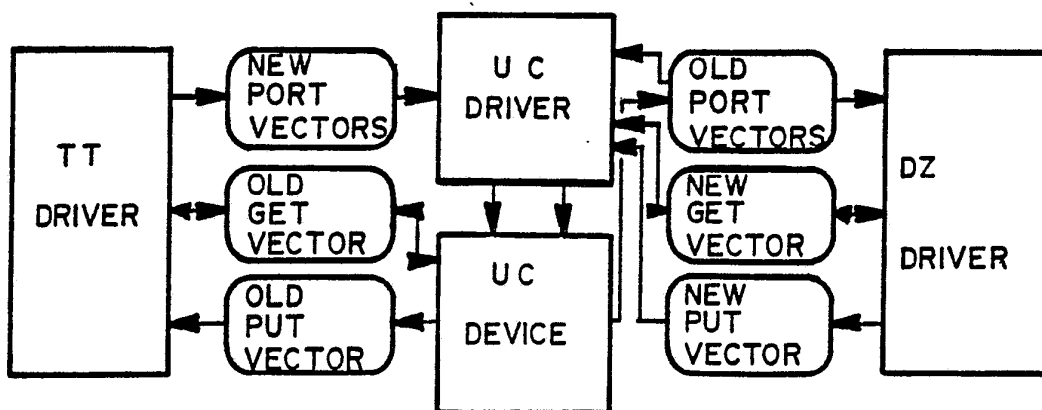


FIG. 3

SUBSTITUTE SHEET

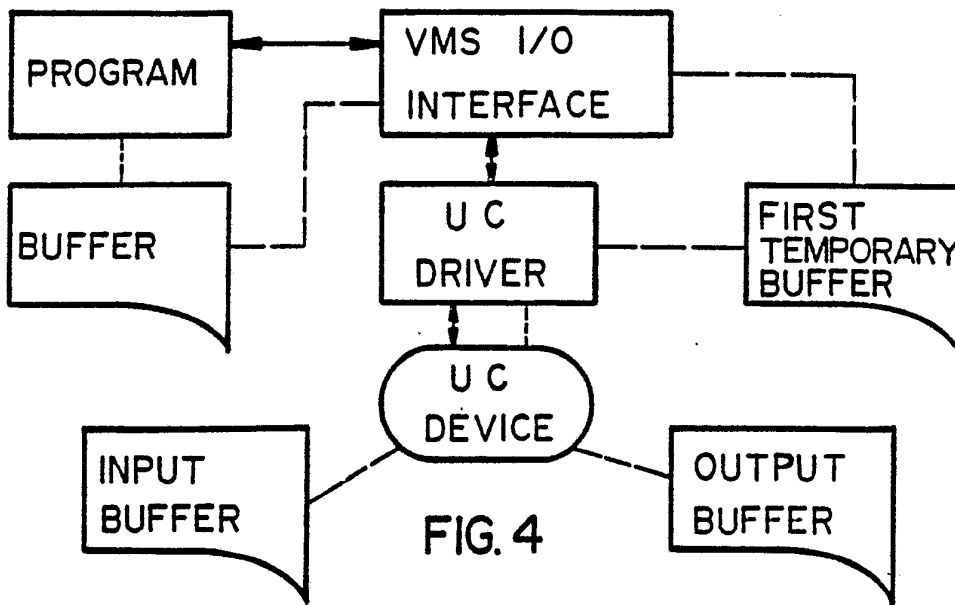


FIG. 4

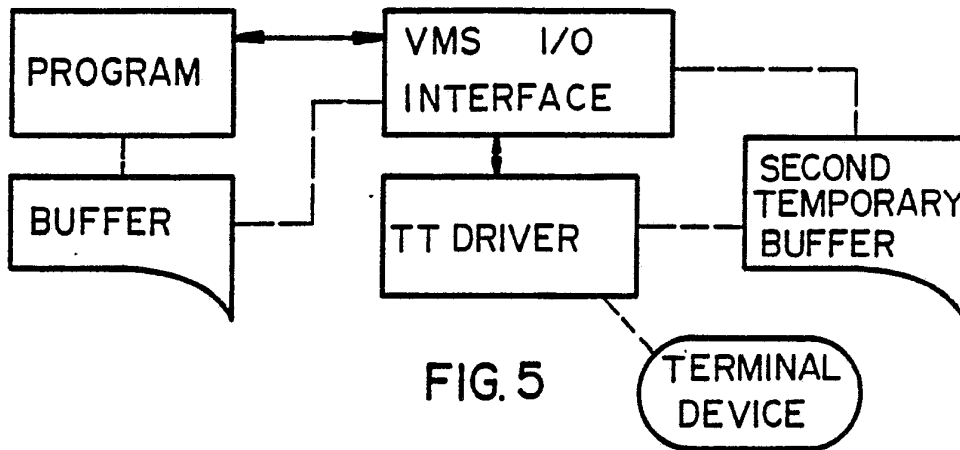


FIG. 5

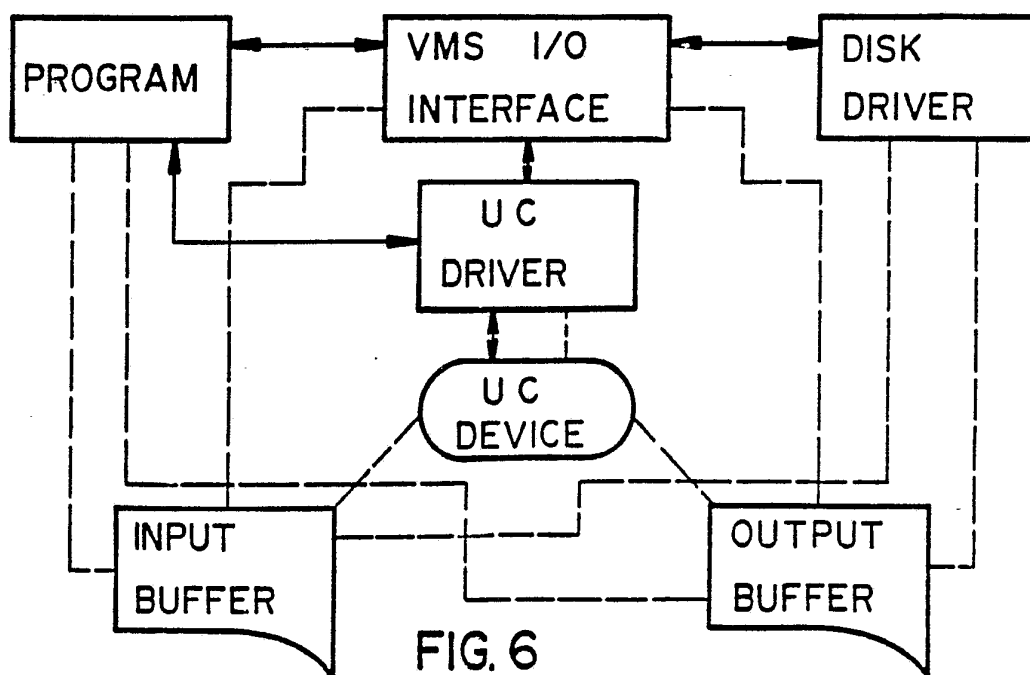


FIG. 6

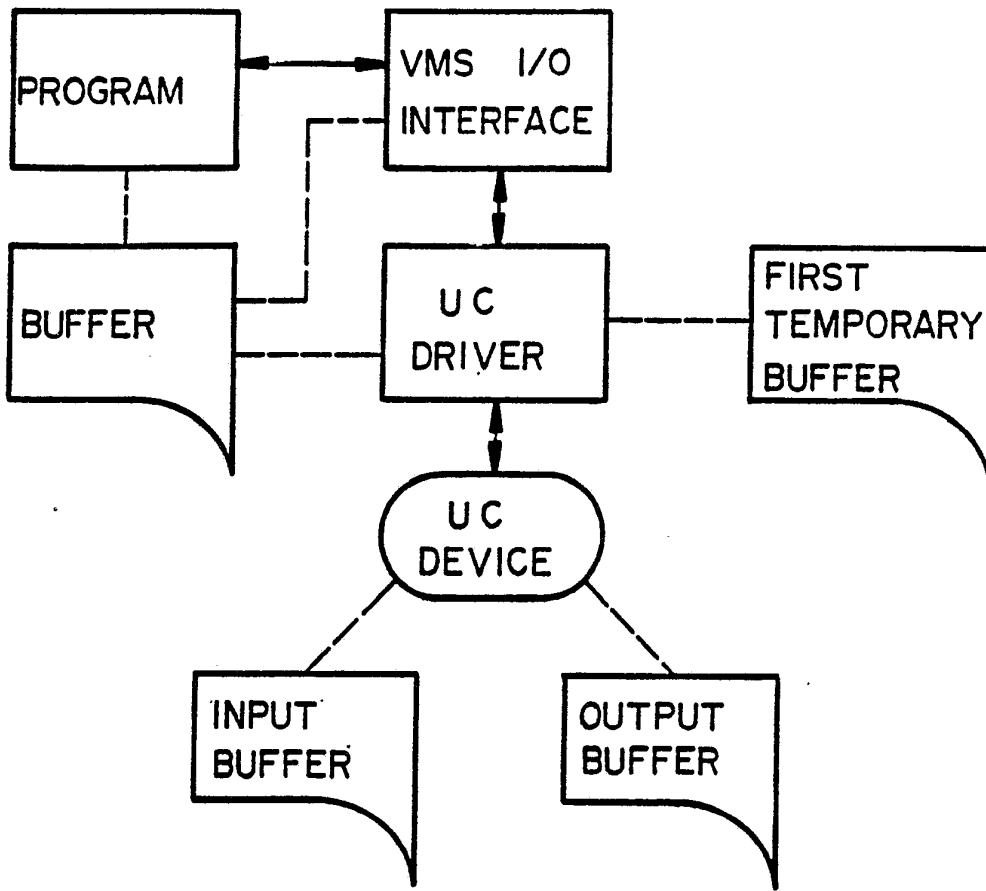


FIG. 8

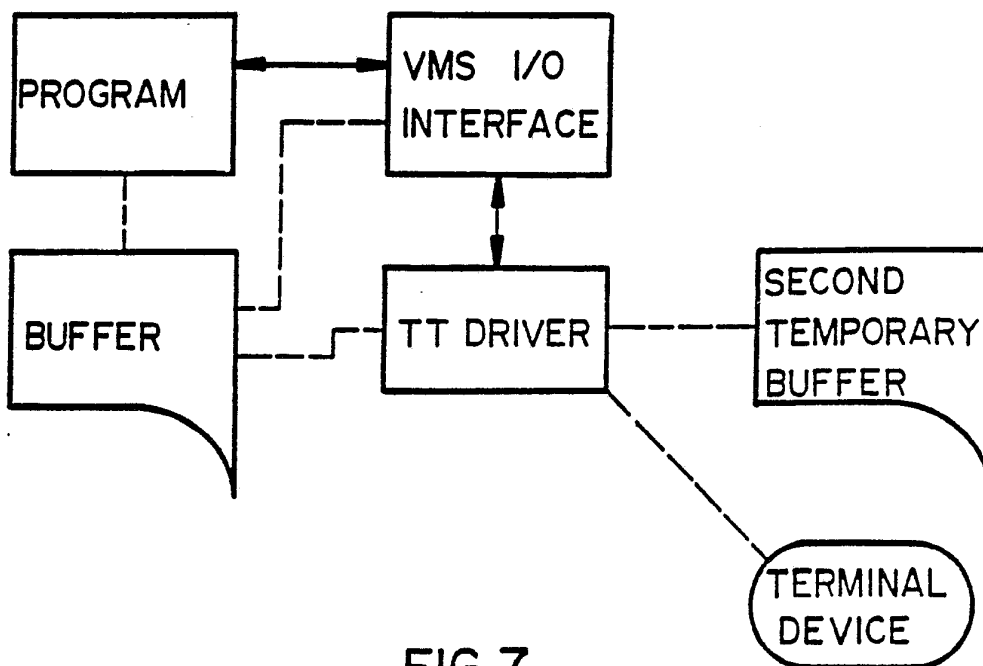
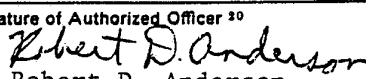


FIG. 7

# INTERNATIONAL SEARCH REPORT

International Application No PCT/US87/02653

<b>I. CLASSIFICATION OF SUBJECT MATTER</b> (if several classification symbols apply, indicate all) <sup>3</sup>		
According to International Patent Classification (IPC) or to both National Classification and IPC		
IPC (4) G06F 09/00 // G06F 11/00		
U.S. CL. 364/200		
<b>II. FIELDS SEARCHED</b>		
Minimum Documentation Searched <sup>4</sup>		
Classification System	Classification Symbols	
U.S. CL. 364/200, 900		
Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched <sup>5</sup>		
<b>III. DOCUMENTS CONSIDERED TO BE RELEVANT</b> <sup>14</sup>		
Category <sup>6</sup>	Citation of Document, <sup>16</sup> with indication, where appropriate, of the relevant passages <sup>17</sup>	Relevant to Claim No. <sup>18</sup>
E, X	US, A, 4,701,848 (CLYDE) 20 OCTOBER 1987 See the entire document.	1-17
X	US, A, 3,701,971 (SANNER) 31 OCTOBER 1972 See the entire document.	1-17
A	US, A, 4,034,351 (TAKEZOE) 5 JULY 1977 See the entire document.	
A	US, A, 4,057,847 (LOWELL) 8 NOVEMBER 1977 See the entire document.	
Y	US, A, 4,125,892 (FUKUDA) 14 NOVEMBER 1978 See the entire document.	1-17
X	US, A, 4,649,479 (ADVANI) 10 MARCH 1987 See the entire document.	1-17
A	US, A, 3,539,998 (BELCHER) 10 NOVEMBER 1970 See the entire document.	
A	US, A, 4,034,346 (HOSTEIN) 5 JULY 1977 See the entire document.	
<p><sup>15</sup> * Special categories of cited documents:</p> <p>"A" document defining the general state of the art which is not considered to be of particular relevance</p> <p>"E" earlier document but published on or after the international filing date</p> <p>"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)</p> <p>"O" document referring to an oral disclosure, use, exhibition or other means</p> <p>"P" document published prior to the international filing date but later than the priority date claimed</p> <p>"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention</p> <p>"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step</p> <p>"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.</p> <p>"&amp;" document member of the same patent family</p>		
<b>IV. CERTIFICATION</b>		
Date of the Actual Completion of the International Search <sup>19</sup>	Date of Mailing of this International Search Report <sup>20</sup>	
04 JANUARY 1988	04 FEB 1988	
International Searching Authority <sup>21</sup>	Signature of Authorized Officer <sup>20</sup>	
ISA/US	 Robert D. Anderson	

**FURTHER INFORMATION CONTINUED FROM THE SECOND SHEET**

A FREE, et al., "IBM Personal Computer Local/Remote Display and Keyboard Sharing," IBM Tech. Disclosure Bulletin, Vol. 27 No. 03, AUGUST 1984, page 1639.

A US, A, 4,034,339 (Free) 5 JULY 1977

**V.  OBSERVATIONS WHERE CERTAIN CLAIMS WERE FOUND UNSEARCHABLE <sup>10</sup>**

This international search report has not been established in respect of certain claims under Article 17(2) (a) for the following reasons:

1.  Claim numbers \_\_\_\_\_, because they relate to subject matter <sup>12</sup> not required to be searched by this Authority, namely:

2.  Claim numbers \_\_\_\_\_, because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out <sup>13</sup>, specifically:

**VI.  OBSERVATIONS WHERE UNITY OF INVENTION IS LACKING <sup>11</sup>**

This International Searching Authority found multiple inventions in this international application as follows:

1.  As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims of the international application.

2.  As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims of the international application for which fees were paid, specifically claims:

3.  No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claim numbers:

4.  As all searchable claims could be searched without effort justifying an additional fee, the International Searching Authority did not invite payment of any additional fee.

Remark on Protest

The additional search fees were accompanied by applicant's protest.

No protest accompanied the payment of additional search fees.