



SEARCHED

Class	Sub.	Date	Exmr.
713	2	1/27/04	SES
711	1	1/24/04	SES
712	600	1/21/04	SES
711	113	11/21/04	SES
711	120	11/19/04	SES
Updated Search		9/21/04 9/22/04 9/27/04 10/18/04	SES
Updated Search		6/17/05	SES
Search Updated		2/6/06	SES
711	118	2/6/06	SES

SEARCH NOTES (INCLUDING SEARCH STRATEGY)

East	Date	Exmr.
USPT US-PGUS EPO, SPO IBM, TSB NPL	1/28/04 1/29/04 2/4/04 2/4/04	SES
Updated Search	9/21/04 9/22/04 9/27/04 10/18/04	SES
Updated Search	5/27/05	SES
Updated Search	2/6/06	SES

Best Available Copy

INTERFERENCE SEARCHED

Class	Sub.	Date	Exmr.

(RIGHT OUTSIDE)

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
S1	6	(pre\$1load\$3 cach\$3) near3 boot adj (data program)	USPAT	OR	OFF	2006/02/06 12:02
S2	6	(pre\$1load\$3 cach\$3) near3 boot adj (data program)	US-PGPUB	OR	OFF	2006/02/06 12:04
S3	0	(pre\$1load\$3 cach\$3) near3 boot adj (data program)	EPO; JPO; IBM_TDB	OR	OFF	2006/02/06 12:04
S4	7	boot adj (data program) near3 cache	USPAT	OR	OFF	2006/02/06 12:15
S5	8	boot adj (data program) near3 cache	US-PGPUB	OR	OFF	2006/02/06 12:27
S6	0	boot adj (data program) near3 cache	EPO; JPO; IBM_TDB	OR	OFF	2006/02/06 12:27
S7	3	("6463509").URPN.	USPAT	OR	OFF	2006/02/06 12:31
S8	50	("3609665" "3806888" "4020466" "4215400" "4295205" "4342079" "4435775" "4500954" "4637024" "5128810" "5131089" "5146576" "5218689" "5226168" "5263142" "5287457" "5291584" "5293622" "5359713" "5396596" "5420998" "5437018" "5448719" "5459850" "5483641" "5493574" "5515525" "5519853" "5551000" "5555402" "5594885" "5603011" "5633484" "5657470" "5673394" "5680570" "5680573" "5692190" "5694567" "5694570" "5712811" "5732267" "5737619" "5745773" "5778418" "5787470" "5802566" "5895487" "5913224" "6003115").PN.	US-PGPUB; USPAT; USOCR	OR	OFF	2006/02/06 12:33
S9	3	("6539456").URPN.	USPAT	OR	OFF	2006/02/06 13:10
S10	6	("5307497" "6061788" "6073232" "6172936" "6189100" "6226740").PN.	US-PGPUB; USPAT; USOCR	OR	OFF	2006/02/06 13:18
S11	7	("6003115").URPN.	USPAT	OR	OFF	2006/02/06 13:25
S12	58	("5307497").URPN.	USPAT	OR	OFF	2006/02/06 13:28
S13	7	("6073232").URPN.	USPAT	OR	OFF	2006/02/06 14:15
S14	0	("6704840").URPN.	USPAT	OR	OFF	2006/02/06 14:17
S15	9	("5155833" "5301328" "5408636" "5671413" "5809531" "6073232" "6282644" "6401144" "6539456").PN.	US-PGPUB; USPAT; USOCR	OR	OFF	2006/02/06 14:18
S16	0	("6968450").URPN.	USPAT	OR	OFF	2006/02/06 14:22

S17	23	("20010039612" "20020156970" "20030212857" "20040003223" "5155833" "5269019" "5269022" "5307497" "5448719" "5581785" "5636355" "5724501" "5812883" "5832005" "5978922" "6073232" "6098158" "6101574" "6209088" "6434696" "6449683" "6745283" "6807630").PN.	US-PGPUB; USPAT; USOCR	OR	OFF	2006/02/06 14:24
-----	----	---	------------------------------	----	-----	------------------



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants: Fallon et al.

Examiner: Suryawanshi, Suresh

Serial No.: 09/776,267

Group Art Unit: 2115

Filed: February 2, 2001

Docket: 8011-15

For: **SYSTEMS AND METHODS FOR ACCELERATED LOADING OF
OPERATING SYSTEMS AND APPLICATION PROGRAMS**

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

AMENDMENT

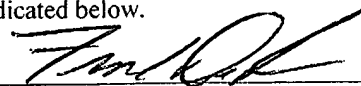
This is a response to the Office Action mailed on June 6, 2005. Please amend the claims as follows:

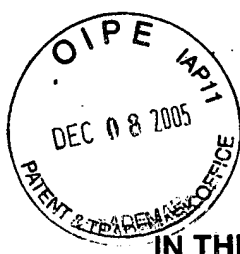
CERTIFICATE OF MAILING 37 C.F.R. §1.8(a)

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail, postpaid in an envelope, addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on the date indicated below.

Date:

12/6/05


Frank V. DeRosa



TFW 2115

Attorney Docket No.: 8011-15

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICANT(S): Fallon et al. Examiner: S. Suryawanshi

SERIAL NO.: 09/776,267 Group Art Unit: 2115

FILED: February 2, 2001 Dated: December 6, 2005

FOR: SYSTEMS AND METHODS FOR ACCELERATED LOADING OF OPERATING SYSTEMS AND APPLICATION PROGRAMS

Mail Stop Amendment
 Commissioner for Patents
 P.O. Box 1450
 Alexandria, VA 22313-1450

AMENDMENT TRANSMITTAL FORM

Sir:

Transmitted herewith is an amendment in the above-identified application.

- Small entity status of this application under 37 C.F.R. §§1.9 and 1.27 has been established by a verified statement previously submitted.
- A verified statement to establish small entity under 37 C.F.R. §§1.9 and 1.27 is enclosed.
- No additional fee is required.


For	Claims Remaining After Amendment	Highest No. Previously Paid For	Present Extra	Rate (Small Entity)	Addit. Fee	Rate	Addit. Fee
TOTAL CLAIMS	10	20	0	x 25 =	\$0	x 50 =	\$0
INDEPENDENT CLAIMS	2	3	0	x 100	\$0	x 200 =	\$0
<input type="checkbox"/> First Presentation of Multiple Dep. Claim				180		360	\$0

* If the entry in Col. 1 is less than entry in Col. 2, write "0" in Col. 3.
 ** If the "Highest No. Previously Paid for" IN THIS SPACE is less than 20, enter "20".
 *** If the "Highest No. Previously Paid For" IN THIS SPACE is less than 3, enter "3".
 The Highest No. Previously Paid For" (Total or indep.) is the highest number found in the appropriate box in Col. 1 of a prior amendment or the number of claims originally filed.

CERTIFICATE OF MAILING UNDER 37 C.F.R. §1.8(a)

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail, postpaid in an envelope, addressed to the: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on December 6, 2005.


Dated: December 6, 2005


 Frank V. DeRosa

- Please charge Deposit Account No. 50-0679 in the amount of \$____. Two (2) copies of this sheet are enclosed.
- The amount of \$_____ is authorized to be charged to a Credit Card. Form PTO-2038 is enclosed.
- Please charge any deficiency as well as any other fee(s) which may become due under 37 C.F.R. §§1.16 and/or 1.17 at any time during the pendency of this application, or credit any overpayment of such fee(s) to Deposit Account No. 50-0679. Also, in the event any extensions of time for responding are required for the pending application(s), please treat this paper as a petition to extend the time as required and charge Deposit Account No. 50-0679 therefor. TWO (2) COPIES OF THIS SHEET ARE ENCLOSED.

F. CHAU & ASSOCIATES, LLC
130 Woodbury Road
Woodbury, NY 11797
(516) 692-8888

Respectfully submitted,


Frank V. DeRosa
Reg No. 43,584
Attorney for Applicant(s)

IN THE CLAIMS:

1. (Previously Presented) A method for providing accelerated loading of an operating system, comprising the steps of:

- maintaining a list of boot data used for booting a computer system;
- initializing a central processing unit of the computer system;
- preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device; and
- servicing requests for boot data from the computer system using the preloaded boot data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data.

2. (Original) The method of claim 1, wherein the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof.

3. (Canceled)

4. (Previously Presented) The method of claim 1, wherein the method steps are performed by a data storage controller connected to the boot device.

5. (Original) The method of claim 1, further comprising the step of updating the list of boot data during the boot process.

6. (Original) The method of claim 5, wherein the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list.

7. (Original) The method of claim 5, wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system.

8. (Canceled)

9. (Original) The method of claim 1, wherein the method steps are program instructions that are tangibly embodied on a program storage device and readable by a machine to execute the method steps.

10. (Canceled)

11. (Canceled)

12. (Canceled)

13. (Previously Presented) A boot device controller for providing accelerated loading of an operating system of a host system, the boot device controller comprising:

a digital signal processor (DSP) or controller comprising a data compression engine (DCE) for compressing boot data stored to a boot device and for decompressing compressed boot data retrieved from the boot device;

a programmable volatile logic device, wherein the programmable volatile logic device is programmed by the DSP or controller prior to completion of initialization of a central processing unit of the host system, to (i) instantiate a first interface for operatively interfacing the boot device controller to the boot device and to (ii) instantiate a second interface for operatively interfacing the boot device controller to the host system;

a cache memory device; and

a non-volatile memory device, for storing logic code associated with the DSP or controller, the first interface and the second interface, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system, for preloading the compressed boot data into the cache memory device ~~upon~~ prior to completion of initialization of the central processing unit of the host system, and for decompressing the preloaded compressed boot data to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system.

14. (Canceled)

15. (Previously Presented) The boot device controller of claim 13, wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data.

16. (Canceled)

17. (Previously Presented) The method of claim 1, further comprising:
maintaining a list of application data associated with an application program;
preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing compressed application data from a boot device;
and

servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data.

REMARKS

Claims 1, 2, 4-7, 9, 10, 12, 13, 15 and 17 are pending and stand rejected. By the above amendment, claims 10 and 12 have been canceled without prejudice.

Reconsideration of the claim rejections is requested based on the following remarks.

Claims 1, 2, 4-7, 9, 10, 12, 13, 15 and 17 stand rejected as being unpatentable over U.S. Patent No. 6,073,232 to Kroeker in view of U.S. Patent No. 6,434,695 to Esfahani, et al. The rejection of claims 10 and 12 is rendered moot by cancellation of such claims.

At the very least, it is respectfully submitted that claims 1 and 13 are patentable and non-obvious over the combination of Kroeker and Esfahani. For instance, with respect to claim 1, the combination of Kroeker and Esfahani does not disclose or suggest *preloading the boot data into a cache memory prior to completion of initialization of a central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device*, as essentially recited in claim 1.

From the Response to Arguments (page 11) section of the Office Action, it appears that Examiner agrees with Applicants previous arguments that Esfahani does not disclose *preloading the boot data into a cache memory prior to completion of initialization of a central processing unit of the computer system*. However, the Examiner contends (p. 11, ¶ 16) that Kroeker “clearly discloses preloading the boot data into a cache memory prior to completion of initialization of a central process unit ...” Applicants respectfully disagree with Examiner’s characterization of Kroeker in this

regard.

Although Kroeker generally discloses that the *disk drive completes its booting process before the host computer system is ready for program transfer*, “ this does not specifically teach or suggest *preloading boot data into a cache memory prior to completion of initialization of a central processing unit of the computer system*. Indeed, Examiner should be aware that CPU initialization is one aspect of host computer initialization. Clearly, the Examiner has not specifically explained or demonstrated that Kroeker discloses preloading boot data before initialization of the host system CPU and, consequently, the rejections are seemingly based on surmise and conjecture. For at least this reason, the Office Action fails to present a *prima facie* case of obviousness against claim 1.

Moreover, with respect to claim 13, the combination of Kroeker and Esfahani does not teach or suggest, e.g., *a programmable volatile logic device, wherein the programmable volatile logic device is programmed by the DSP or controller prior to completion of initialization of a central processing unit of the host system*, much less the *DSP or controller preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system, and decompressing the preloaded compressed boot data to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system*, as essentially recited in claim 13.

At the very least, the same arguments as set forth above for claim 1 apply to claim 1 in that the Examiner has not shown with legal sufficiency to establish a *prima facie* case

of obviousness that the cited combination of references discloses preloading the compressed boot data into a cache memory device prior to completion of initialization of the central processing unit of the host system, as recited in claim 13.

Therefore, claims 1 and 13 are patentable and non-obvious over the combination of Kroeker and Esfahani. Moreover, the remaining dependent claims are patentable over the cited combination at least by virtue of their dependence from respective base claims 1 or 13.

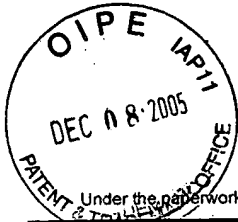
Early and favorable consideration by the Examiner is respectfully urged. Should the Examiner believe that a telephone or personal interview may facilitate resolution of any remaining matters, it is requested that the Examiner contact Applicants' undersigned attorney.

Respectfully submitted,



Frank V. DeRosa
Reg. No. 43,584
Attorney for Applicant(s)

F. Chau & Associates, LLC
130 Woodbury Road
Woodbury, New York 11797
TEL.: (516) 692-8888
FAX: (516) 692-8889



PETITION FOR EXTENSION OF TIME UNDER 37 CFR 1.136(a)
FY 2005
(Fees pursuant to the Consolidated Appropriations Act, 2005 (H.R. 4818).)

Docket Number (Optional)
8011-15

Application Number 09/776,267

Filed February 2, 2001

For Systems and Methods for Accelerated Loading of Operating Systems....

Art Unit 2115

Examiner James J. Fallon

This is a request under the provisions of 37 CFR 1.136(a) to extend the period for filing a reply in the above identified application.

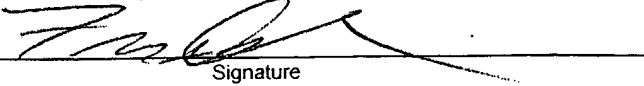
The requested extension and fee are as follows (check time period desired and enter the appropriate fee below):

	Fee	Small Entity Fee	
<input type="checkbox"/> One month (37 CFR 1.17(a)(1))	\$120	\$60	\$ _____
<input type="checkbox"/> Two months (37 CFR 1.17(a)(2))	\$450	\$225	\$ _____
<input checked="" type="checkbox"/> Three months (37 CFR 1.17(a)(3))	\$1020	\$510	\$ <u>510.00</u>
<input type="checkbox"/> Four months (37 CFR 1.17(a)(4))	\$1590	\$795	\$ _____
<input type="checkbox"/> Five months (37 CFR 1.17(a)(5))	\$2160	\$1080	\$ _____

- Applicant claims small entity status. See 37 CFR 1.27.
- A check in the amount of the fee is enclosed.
- Payment by credit card. Form PTO-2038 is attached.
- The Director has already been authorized to charge fees in this application to a Deposit Account.
- The Director is hereby authorized to charge any fees which may be required, or credit any overpayment, to Deposit Account Number _____ I have enclosed a duplicate copy of this sheet.

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

- I am the
- applicant/inventor.
 - assignee of record of the entire interest. See 37 CFR 3.71. Statement under 37 CFR 3.73(b) is enclosed (Form PTO/SB/96).
 - attorney or agent of record. Registration Number 43,584
 - attorney or agent under 37 CFR 1.34. Registration number if acting under 37 CFR 1.34 _____

 _____
Signature

December 6, 2005
Date

Frank V. DeRosa
Typed or printed name

516-692-8888
Telephone Number

NOTE: Signatures of all the inventors or assignees of record of the entire interest or their representative(s) are required. Submit multiple forms if more than one signature is required, see below.

Total of _____ forms are submitted.

This collection of information is required by 37 CFR 1.136(a). The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to take 6 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PATENT APPLICATION FEE DETERMINATION RECORD
Substitute for Form PTO-875

Application or Docket Number
09/776,267

CLAIMS AS FILED - PART I

FOR	(Column 1) NUMBER FILED	(Column 2) NUMBER EXTRA
BASIC FEE (37 CFR 1.16(a))		
TOTAL CLAIMS (37 CFR 1.16(c))	16 minus 20 =	0
INDEPENDENT CLAIMS (37 CFR 1.16(b))	3 minus 3 =	0
MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(d))		

SMALL ENTITY	
RATE	FEE
	355
x \$	=
x \$	=
+ \$	=
TOTAL	355

OTHER THAN SMALL ENTITY	
RATE	FEE
	\$
x \$	=
x \$	=
+ \$	=
TOTAL	

* If the difference in column 1 is less than zero, enter "0" in column 2.

CLAIMS AS AMENDED - PART II

AMENDMENT A	(Column 1)	(Column 2)	(Column 3)
	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total (37 CFR 1.16(c))	11	16	0
Independent (37 CFR 1.16(b))	3	3	0
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(d))			

SMALL ENTITY	
RATE	ADDITIONAL FEE
x \$	=
x \$	=
+ \$	=
TOTAL ADD'L FEE	0

OTHER THAN SMALL ENTITY	
RATE	ADDITIONAL FEE
x \$	=
x \$	=
+ \$	=
TOTAL ADD'L FEE	

5/2/05

AMENDMENT B	(Column 1)	(Column 2)	(Column 3)
	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total (37 CFR 1.16(c))	11	16	0
Independent (37 CFR 1.16(b))	3	3	0
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(d))			

SMALL ENTITY	
RATE	ADDITIONAL FEE
x \$	=
x \$	=
+ \$	=
TOTAL ADD'L FEE	0

OTHER THAN SMALL ENTITY	
RATE	ADDITIONAL FEE
x \$	=
x \$	=
+ \$	=
TOTAL ADD'L FEE	

12-8-05

AMENDMENT C	(Column 1)	(Column 2)	(Column 3)
	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total (37 CFR 1.16(c))	10	20	0
Independent (37 CFR 1.16(b))	2	3	0
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(d))			

SMALL ENTITY	
RATE	ADDITIONAL FEE
x \$	=
x \$	=
+ \$	=
TOTAL ADD'L FEE	7

OTHER THAN SMALL ENTITY	
RATE	ADDITIONAL FEE
x \$	=
x \$	=
+ \$	=
TOTAL ADD'L FEE	

- * If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
- ** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".
- *** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".

The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

This collection of information is required by 37 CFR 1.16. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/776,267	02/02/2001	James J. Fallon	8011-15	9730

22150 7590 06/06/2005
F. CHAU & ASSOCIATES, LLC
130 WOODBURY ROAD
WOODBURY, NY 11797

EXAMINER

SURYAWANSHI, SURESH

ART UNIT PAPER NUMBER

2115

DATE MAILED: 05/06/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No. 09/776,267	Applicant(s) FALLON ET AL.	
	Examiner Suresh K. Suryawanshi	Art Unit 2115	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) Responsive to communication(s) filed on 02 May 2005.
- 2a) This action is FINAL. 2b) This action is non-final.
- 3) Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) Claim(s) 1,2,4-7,9,10,12,13,15 and 17 is/are pending in the application.
- 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) Claim(s) _____ is/are allowed.
- 6) Claim(s) 1,2,4-7,9,10,12,13,15 and 17 is/are rejected.
- 7) Claim(s) _____ is/are objected to.
- 8) Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) The specification is objected to by the Examiner.
- 10) The drawing(s) filed on 02 February 2001 is/are: a) accepted or b) objected to by the Examiner.
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

- 12) Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) All b) Some * c) None of:
1. Certified copies of the priority documents have been received.
 2. Certified copies of the priority documents have been received in Application No. _____.
 3. Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

- | | |
|--|---|
| 1) <input checked="" type="checkbox"/> Notice of References Cited (PTO-892) | 4) <input type="checkbox"/> Interview Summary (PTO-413)
Paper No(s)/Mail Date. _____ |
| 2) <input type="checkbox"/> Notice of Draftsperson's Patent Drawing Review (PTO-948) | 5) <input type="checkbox"/> Notice of Informal Patent Application (PTO-152) |
| 3) <input type="checkbox"/> Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08)
Paper No(s)/Mail Date _____ | 6) <input type="checkbox"/> Other: _____ |



DETAILED ACTION

1. Claims 1-2, 4-7, 9-10, 12-13, 15 and 17 are presented for examination.

Drawings

2. This application, filed under former 37 CFR 1.60, lacks formal drawings. The informal drawings filed in this application are acceptable for examination purposes. When the application is allowed, applicant will be required to submit new formal drawings. In unusual circumstances, the formal drawings from the abandoned parent application may be transferred by the grant of a petition under 37 CFR 1.182.

Claim Rejections - 35 USC § 103

1. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

2. Claims 1-2, 4-7, 9-10, 12-13, 15 and 17 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kroeker et al (US Patent no 6,073,232¹) in view of Esfahani et al (US Patent no 6,434,695 B1¹).

¹ Prior art cited by the examiner in the prior office action.

Art Unit: 2115

3. As per claim 1, Kroeker et al teach

maintaining a list of boot data used for booting a computer system [col. 2, lines 30-47; col. 5, lines 1-7; a prefetch table containing a listing of the disk locations and length of data records that were requested by the host computer in the immediately previous power-on/reset];

initializing a central processing unit of the computer system [col. 2, lines 30-35; inherent to the system during power-up process];

preloading the boot data into a cache memory prior to completion of initialization of the central processing unit of the computer system [col. 1, lines 58-64; col. 2, lines 36-41; col. 3, lines 30-39; col. 5, lines 17-21; data is preloaded into the RAM cache according to the prefetch table prior to completion of initialization of the central processor unit as shown in Fig. 3 that the method enters an idle state to await a command from the host computer since the CPU of the host computer is not ready]; and

servicing requests for boot data from the computer system using the preloaded boot data after completion of initialization of the central processing unit of the computer system [col. 2, lines 41-47; col. 3, lines 30-39; data is communicated from the cache to the host computer as soon as the host computer requests the data upon completion of initialization of the CPU].

Art Unit: 2115

Kroeker et al do not disclose about accessing compressed boot data and decompressing the compressed boot data. However, Esfahani et al clearly disclose about loading a compressed boot data into a RAM cache and then the boot data is decompressed and executed [col. 2, lines 5-13, 63, 67; col. 10, line 65 – col. 11, line 4]. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the cited references as both are directed to minimize a computer's initial program load time or shortening the load time of the computer programs from a hard disk drive to a host computer. Moreover, the shortening load time method of Kroeker et al by loading the program codes into the RAM cache according to the prefetch table will definitely be benefited with the method of reading compressed data into the RAM cache and then decompressing and executing as needed. This way, one may not only have needed data into a fast access memory but also a large amount of data to avoid frequent accessing the storage device(s).

4. As per claim 2, Kroeker et al teach that the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof [col. 5, lines 41-51; requesting data records are part of a computer program such as DOS or Windows].

5. As per claim 4, Kroeker et al teach that the method steps are performed by a data storage controller connected to the boot device [fig. 1; controller].

Art Unit: 2115

6. As per claim 5, Kroeker et al teach the step of updating the list of boot data during the boot process [col. 8, lines 63-65; the prefetch table is updated].

7. As per claim 6, Kroeker et al teach the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list [col. 8, lines 63-68; the prefetch table is updated].

8. As per claim 7, Kroeker et al teach that the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system [col. 8; lines 63-65; updating the prefetch table].

9. As per claims 9 and 12, Kroeker et al teach that the method steps are program instructions that are tangibly embodied on a program storage device and readable by a machine to execute the method steps [col. 9, lines 27-30; computer program].

10. As per claim 10, Kroeker et al teach

maintaining a list of application data associated with an application program [col. 11; lines 30-34; a prefetch table containing disk storage location and length of the data records requested by the application program];

Art Unit: 2115

preloading the application data upon launching the application program [col. 11, lines 46-50; preloading the data cache prior to receiving a read command from the application]; and

servicing requests for application data from a computer system using the preloaded application data [col. 11, lines 51-57; communicating the prestored data records of the application from the data cache to the host computer].

Kroeker et al do not disclose about accessing compressed data and decompressing the compressed data. However, Esfahani et al clearly disclose about loading a compressed data into a RAM cache and then the compressed data is decompressed and executed [col. 2, lines 5-13, 63, 67; col. 10, line 65 – col. 11, line 4]. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the cited references as both are directed to minimize a computer's initial program load time or shortening the load time of the computer programs from a hard disk drive to a host computer. Moreover, the shortening load time method of Kroeker et al by loading the program codes into the RAM cache according to the prefetch table will definitely be benefited with the method of reading compressed data into the RAM cache and then decompressing and executing as needed. This way, one may not only have needed data into a fast access memory but also a large amount of data to avoid frequent accessing the storage device(s).

Art Unit: 2115

11. As per claim 13, Kroeker et al teach

a digital signal processor (DSP) [fig. 1; host computer];

a programmable volatile logic device [fig. 1, RAM cache], wherein the programmable volatile logic device is programmed by the DSP or controller prior to completion of initialization of a central processing unit of the host system [col. 1, lines 58-64; col. 2, lines 36-41; col. 3, lines 30-39; col. 5, lines 17-21; data is preloaded into the RAM cache according to the prefetch table prior to completion of initialization of the central processor unit as shown in Fig. 3 that the method enters an idle state to await a command from the host computer since the CPU of the host computer is not ready] to (i) instantiate a first interface for operatively interfacing the boot device controller to a boot device [fig. 1; controller] and to (ii) instantiate a second interface for operatively interfacing the boot device controller to the host system [inherent to the system as a bus interface is used to interface the controller with host computer];

a cache memory device [Fig. 1; RAM cache]; and

a non-volatile memory device, for storing logic code associated with the DSP, the first interface and the second interface, wherein the logic code comprises instructions executable by the DSP for maintaining a list of boot data used for booting the host system [fig. 1; col. 4, lines 10-28; instructions are embodied as microcode in a ROM; col. 5, lines 1-7; a prefetch table is read from a reserved area of the disks], for preloading the boot data into the cache memory

Art Unit: 2115

device prior to completion of initialization of the central processing unit of the host system [col. 1, lines 58-64; col. 2, lines 36-41; col. 3, lines 30-39; col. 5, lines 17-21; data is preloaded into the RAM cache according to the prefetch table prior to completion of initialization of the central processor unit as shown in Fig. 3 that the method enters an idle state to await a command from the host computer since the CPU of the host computer is not ready], and servicing requests for boot data from the host system after completion of initialization of the central processing unit of the host system using the preloaded boot data [col. 2, lines 41-47; col. 3, lines 30-39; data is communicated from the cache to the host computer as soon as the host computer requests the data upon completion of initialization of the CPU].

Kroeker et al do not disclose about accessing compressed boot data and decompressing the compressed boot data. However, Esfahani et al clearly disclose about loading a compressed boot data into a RAM cache and then the boot data is decompressed and executed [col. 2, lines 5-13, 63, 67; col. 10, line 65 – col. 11, line 4]. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the cited references as both are directed to minimize a computer's initial program load time or shortening the load time of the computer programs from a hard disk drive to a host computer. Moreover, the shortening load time method of Kroeker et al by loading the program codes into the RAM cache according to the prefetch table will definitely be benefited with the method of reading compressed data into the RAM cache and then decompressing and executing as needed. This way, one may not only have needed data into a fast access memory but also a large amount of data to avoid frequent accessing the storage device(s).

Art Unit: 2115

12. As per claim 15, Kroeker et al teach that the logic code in the non-volatile memory device further comprises program instructions executable by the DSP for maintaining a list of application data associated with an application program [col. 11; lines 30-34; a prefetch table containing disk storage location and length of the data records requested by the application program]; preloading the application data upon launching the application program [col. 11, lines 46-50; preloading the data cache prior to receiving a read command from the application], and servicing requests for the application data from the host system using the preloaded application data col. 11, lines 51-57; communicating the prestored data records of the application from the data cache to the host computer].

13. As per claim 17, Kroeker et al teach

maintaining a list of application data associated with an application program [col. 2, lines 30-47; col. 5, lines 1-7; a prefetch table containing a listing of the disk locations and length of data records that were requested by the host computer in the immediately previous power-on/reset; col. 5, lines 41-51; requesting data records are part of a computer program such as DOS or Windows; claims 28 and 32];

Art Unit: 2115

preloading the application data into the cache memory prior to completion of initialization of the central processing unit of the computer system, wherein preloading the application data comprises accessing application data from a boot device [col. 1, lines 58-64; col. 2, lines 36-41; col. 3, lines 30-39; col. 5, lines 17-21; data is preloaded into the RAM cache according to the prefetch table prior to completion of initialization of the central processor unit as shown in Fig. 3 that the method enters an idle state to await a command from the host computer since the CPU of the host computer is not ready]; and

servicing requests for application data from the computer system using the preloaded application data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing application data from the cache [col. 2, lines 41-47; col. 3, lines 30-39; data is communicated from the cache to the host computer as soon as the host computer requests the data upon completion of initialization of the CPU].

Response to Arguments

14. Applicant's arguments filed 05/02/2005 have been fully considered but they are not persuasive.

15. In the remarks, applicants argued in substance that (1) Kroeker does not disclose preloading the boot data into a cache memory prior to completion of initialization of a central processing unit of the computer system; (2) Kroeker does not teach a method for accelerated loading of an application program.

16. As to point (1), Kroeker clearly disclose preloading the boot data into a cache memory prior to completion of initialization of a central processing unit of the computer system [col. 1, lines 58-64; disk completes its booting process before the host computer is ready for program transfer; col. 2, lines 36-41; col. 3, lines 30-39; col. 5, lines 17-21; data is preloaded into the RAM cache according to the prefetch table prior to completion of initialization of the central processor unit as shown in Fig. 3 that the method enters an idle state to await a command from the host computer since the CPU of the host computer is not ready].

17. As to point (2), Kroeker clearly disclose a method for accelerated loading of an application program as claimed by Kroeker in claims 28 and 32. Kroeker expressly indicates about requests data records of an application program [col. 11, lines 27-29; col. 12, lines 25-28]. Kroeker expressly discloses another object of the present invention for rapidly communicating a computer program from a disk drive to a host computer [col. 2, lines 1-5].

Art Unit: 2115

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Suresh K. Suryawanshi whose telephone number is 571-272-3668. The examiner can normally be reached on 9:00am - 5:30pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Thomas C. Lee can be reached on 571-272-3667. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

sks

May 27, 2005

Chun Cao
CHUN CAO

Notice of References Cited	Application/Control No. 09/776,267	Applicant(s)/Patent Under Reexamination FALLON ET AL.	
	Examiner Suresh K. Suryawanshi	Art Unit 2115	Page 1 of 1

U.S. PATENT DOCUMENTS

*	Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
A	US-6,539,456	03-2003	Stewart, David C.	711/113
B	US-			
C	US-			
D	US-			
E	US-			
F	US-			
G	US-			
H	US-			
I	US-			
J	US-			
K	US-			
L	US-			
M	US-			

FOREIGN PATENT DOCUMENTS

*	Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
N					
O					
P					
Q					
R					
S					
T					

NON-PATENT DOCUMENTS

*	Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
U	
V	
W	
X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
 Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.



SEARCHED

Class	Sub.	Date	Exmr.
713	2	1/21/04	SB
713	1	1/24/04	SB
713	150	1/24/04	SB
711	113	1/29/04	SB
711	120	1/29/04	SB
Updated search		9/21/04 9/23/04 9/27/04 10/19/04	SB
Updated search		5/17/05	SB

SEARCH NOTES (INCLUDING SEARCH STRATEGY)

East	Date	Exmr.
USPAT US-PGPHS EPOY SPO IBM-TSP MPL	1/21/04 1/24/04 1/24/04 1/24/04	SB
Updated search	9/21/04 9/23/04 9/27/04 10/19/04	SB
Updated search	5/27/05	SB

est Available Copy

INTERFERENCE SEARCHED

Class	Sub.	Date	Exmr.

(RIGHT OUTSIDE)

ISSUE SLIP STAPLE AREA (for additional cross references)

POSITION	INITIALS	ID NO.	DATE
FEE DETERMINATION			
O.I.P.E. CLASSIFIER	ASD		3/3/01
FORMALTY REVIEW	NN	778	3/9/01
RESPONSE FORMALTY REVIEW	PL	1090	5-23-01

INDEX OF CLAIMS

- ✓ Rejected
- Allowed
- (Through numeral)... Canceled
- ⊖ Restricted
- N Non-elected
- I Interference
- A Appeal
- O Objected

Claim	Final	Original	Date
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			
26			
27			
28			
29			
30			
31			
32			
33			
34			
35			
36			
37			
38			
39			
40			
41			
42			
43			
44			
45			
46			
47			
48			
49			
50			

Claim	Final	Original	Date
51			
52			
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			
69			
70			
71			
72			
73			
74			
75			
76			
77			
78			
79			
80			
81			
82			
83			
84			
85			
86			
87			
88			
89			
90			
91			
92			
93			
94			
95			
96			
97			
98			
99			
100			

Claim	Final	Original	Date
101			
102			
103			
104			
105			
106			
107			
108			
109			
110			
111			
112			
113			
114			
115			
116			
117			
118			
119			
120			
121			
122			
123			
124			
125			
126			
127			
128			
129			
130			
131			
132			
133			
134			
135			
136			
137			
138			
139			
140			
141			
142			
143			
144			
145			
146			
147			
148			
149			
150			

Best Available Copy

If more than 150 claims or 10 actions
staple additional sheet here

(LEFT INSIDE)



UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
 UNITED STATES PATENT AND TRADEMARK OFFICE
 WASHINGTON, D.C. 20231
 www.uspto.gov



Bib. Data Sheet

CONFIRMATION NO. 9730

SERIAL NUMBER: 09/776,267	FILING DATE 02/02/2001 RULE	CLASS 713	GROUP ART UNIT 2182	ATTORNEY DOCKET NO. 8011-15
-------------------------------------	---	---------------------	-------------------------------	---------------------------------------

APPLICANTS

James J. Fallon, Armonk, NY;
 John Buck, Oceanside, NY;
 Paul F. Pickel, Bethpage, NY;
 Stephen J. McErlain, New York, NY;

**** CONTINUING DATA *******

THIS APPLN CLAIMS BENEFIT OF 60/180,114 02/03/2000
yes *yes*

**** FOREIGN APPLICATIONS *******

None *yes*

IF REQUIRED, FOREIGN FILING LICENSE
 GRANTED ** 03/09/2001

**** SMALL ENTITY ****

Foreign Priority claimed <input type="checkbox"/> yes <input checked="" type="checkbox"/> no	STATE OR COUNTRY NY	SHEETS DRAWING 13	TOTAL CLAIMS 10 12	INDEPENDENT CLAIMS 3
35 USC 119 (a-d) conditions met <input type="checkbox"/> yes <input type="checkbox"/> no <input type="checkbox"/> Met after Allowance				
Verified and Acknowledged Examiner's Signature <i>SJS</i> Initials				

ADDRESS

Frank Chau, Esq.
 F. CHAU & ASSOCIATES, LLP
 Suite 501
 1900 Hempstead Turnpike
 East Meadow, NY 11554

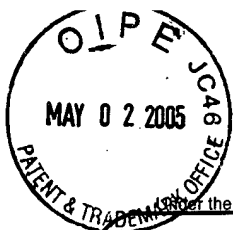
SEP 3 5 58 PM '01

TITLE

Systems and methods for accelerated loading of operating systems and application programs

FILING FEE RECEIVED 420	FEES: Authority has been given in Paper No. _____ to charge/credit DEPOSIT ACCOUNT No. _____ for following:	<input type="checkbox"/> All Fees
		<input type="checkbox"/> 1.16 Fees (Filing)
		<input type="checkbox"/> 1.17 Fees (Processing Ext. of time)
		<input type="checkbox"/> 1.18 Fees (Issue)
		<input type="checkbox"/> Other _____
		<input type="checkbox"/> Credit

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
S1	7	((accelerated fast) adj load\$3) near3 ((operating adj system) OS application program)	USPAT	OR	OFF	2005/05/27 10:23
S2	7	((accelerated fast) adj load\$3) near3 ((operating adj system) OS application program)	US-PGPUB; EPO; JPO; IBM_TDB	OR	OFF	2005/05/27 10:21
S3	1	preload\$3 near2 (boot adj data) near5 cache	USPAT	OR	OFF	2005/05/27 10:25
S4	1	preload\$3 near2 (boot adj data)	USPAT	OR	OFF	2005/05/27 10:25
S5	9	load\$3 near2 (boot adj data)	USPAT	OR	OFF	2005/05/27 10:30
S6	8	load\$3 near2 (boot adj data)	US-PGPUB; EPO; JPO; IBM_TDB	OR	OFF	2005/05/27 10:27
S7	43	(initial adj program adj load) near2 time	USPAT	OR	OFF	2005/05/27 10:42
S8	38	(initial adj program adj load) near2 time	US-PGPUB; EPO; JPO; IBM_TDB	OR	OFF	2005/05/27 10:42
S9	6	("6073232").URPN.	USPAT	OR	OFF	2005/05/27 10:46
S10	1	("6539456").URPN.	USPAT	OR	OFF	2005/05/27 11:09
S11	6	("5307497" "6061788" "6073232" "6172936" "6189100" "6226740").PN.	US-PGPUB; USPAT; USOCR	OR	OFF	2005/05/27 11:10
S12	0	list adj2 (boot adj data)	USPAT	OR	OFF	2005/05/27 11:14
S13	1	list adj2 (boot adj data)	US-PGPUB; EPO; JPO; IBM_TDB	OR	OFF	2005/05/27 11:14
S14	1	"20020069354"	US-PGPUB	OR	OFF	2005/05/27 11:18



RCE *[Signature]*
C-C

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

Request for Continued Examination (RCE) Transmittal

Address to:
Mail Stop RCE
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

<i>Application Number</i>	09/776,267
<i>Filing Date</i>	February 2, 2001
<i>First Named Inventor</i>	James J. Fallon
<i>Art Unit</i>	2115
<i>Examiner Name</i>	S. Suryawanshi
<i>Attorney Docket Number</i>	8011-15

This is a Request for Continued Examination (RCE) under 37 CFR 1.114 of the above-identified application.
Request for Continued Examination (RCE) practice under 37 CFR 1.114 does not apply to any utility or plant application filed prior to June 8, 1995, or to any design application. See Instruction Sheet for RCEs (not to be submitted to the USPTO) on page 2.

1. **Submission required under 37 CFR 1.114** Note: If the RCE is proper, any previously filed unentered amendments and amendments enclosed with the RCE will be entered in the order in which they were filed unless applicant instructs otherwise. If applicant does not wish to have any previously filed unentered amendment(s) entered, applicant must request non-entry of such amendment(s).

- a. Previously submitted. If a final Office action is outstanding, any amendments filed after the final Office action may be considered as a submission even if this box is not checked.
 - i. Consider the arguments in the Appeal Brief or Rely Brief previously filed on _____
 - ii. Other _____
- b. Enclosed
 - i. Amendment/Reply
 - ii. Affidavit(s)/ Declaration(s)
 - iii. Information Disclosure Statement (IDS)
 - iv. Other _____

2. **Miscellaneous**

- a. Suspension of action on the above-identified application is requested under 37 CFR 1.103(c) for a period of _____ months. (Period of suspension shall not exceed 3 months; Fee under 37 CFR 1.17(i) required)
- b. Other _____

3. **Fees**

- The RCE fee under 37 CFR 1.17(e) is required by 37 CFR 1.114 when the RCE is filed. The Director is hereby authorized to charge the following fees, or credit any overpayments, to
- a. Deposit Account No. _____
 - i. RCE fee required under 37 CFR 1.17(e)
 - ii. Extension of time fee (37 CFR 1.136 and 1.17)
 - iii. Other _____
 - b. Check in the amount of \$ _____ enclosed
 - c. Payment by credit card (Form PTO-2038 enclosed)

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT REQUIRED

<i>Name (Print/Type)</i>	Frank V. DeRosa	<i>Registration No. (Attorney/Agent)</i>	43,584
<i>Signature</i>	<i>[Signature]</i>	<i>Date</i>	April 25, 2005

CERTIFICATE OF MAILING OR TRANSMISSION

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to: Mail Stop RCE, Commissioner for Patents, P. O. Box 1450, Alexandria, VA 22313-1450 or facsimile transmitted to the U.S. Patent and Trademark Office on the date shown below.

<i>Name (Print/Type)</i>	Frank V. DeRosa	<i>Date</i>	April 25, 2005
<i>Signature</i>	<i>[Signature]</i>		

This collection of information is required by 37 CFR 1.114. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Mail Stop RCE, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

05/03/2005 HVUONG1 00000029 09776267



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICANT(S): Fallon et al. Examiner: S. Suryawanshi
 SERIAL NO.: 09/776,267 Group Art Unit: 2115
 FILED: February 2, 2001 Dated: April 25, 2005
 FOR: SYSTEMS AND METHODS FOR ACCELERATED LOADING OF OPERATING SYSTEMS AND APPLICATION PROGRAMS

Mail Stop Amendment
 Commissioner for Patents
 P.O. Box 1450
 Alexandria, VA 22313-1450

AMENDMENT TRANSMITTAL FORM

Sir:

Transmitted herewith is an amendment in the above-identified application.

- Small entity status of this application under 37 C.F.R. §§1.9 and 1.27 has been established by a verified statement previously submitted.
- A verified statement to establish small entity under 37 C.F.R. §§1.9 and 1.27 is enclosed.
- No additional fee is required.

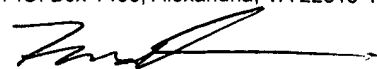
For	Claims Remaining After Amendment	Highest No. Previously Paid For	Present Extra	Rate (Small Entity)	Addit. Fee	Rate	Addit. Fee
TOTAL CLAIMS*	12	20	0	x 25 =	\$0	x 50 =	\$0
INDEPENDENT CLAIMS	3	3	0	x 100	\$0	x 200 =	\$0
<input type="checkbox"/> First Presentation of Multiple Dep. Claim				180		360	\$0

* If the entry in Col. 1 is less than entry in Col. 2, write "0" in Col. 3.
 ** If the "Highest No. Previously Paid for" IN THIS SPACE is less than 20, enter "20".
 *** If the "Highest No. Previously Paid For" IN THIS SPACE is less than 3, enter "3".
 The Highest No. Previously Paid For" (Total or indep.) is the highest number found in the appropriate box in Col. 1 of a prior amendment or the number of claims originally filed.

CERTIFICATE OF MAILING UNDER 37 C.F.R. §1.8(a)

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail, postpaid in an envelope, addressed to the: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on April 25, 2005.

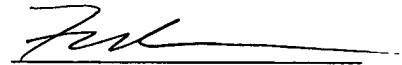
Dated: April 25, 2005


 Frank V. DeRosa

- Please charge Deposit Account No. 50-0679 in the amount of \$____. Two (2) copies of this sheet are enclosed.
- The amount of \$_____ is authorized to be charged to a Credit Card. Form PTO-2038 is enclosed.
- Please charge any deficiency as well as any other fee(s) which may become due under 37 C.F.R. §§1.16 and/or 1.17 at any time during the pendency of this application, or credit any overpayment of such fee(s) to Deposit Account No. 50-0679. Also, in the event any extensions of time for responding are required for the pending application(s), please treat this paper as a petition to extend the time as required and charge Deposit Account No. 50-0679 therefor. TWO (2) COPIES OF THIS SHEET ARE ENCLOSED.

F. CHAU & ASSOCIATES, LLC
130 Woodbury Road
Woodbury, NY 11797
(516) 692-8888

Respectfully submitted,



Frank V. DeRosa
Reg No. 43,584
Attorney for Applicant(s)



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants: Fallon et al.

Examiner: Suryawanshi, Suresh

Serial No.: 09/776,267

Group Art Unit: 2115

Filed: February 2, 2001

Docket: 8011-15

**For: SYSTEMS AND METHODS FOR ACCELERATED LOADING OF
OPERATING SYSTEMS AND APPLICATION PROGRAMS**

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450


AMENDMENT

This is a response to the Final Office Action mailed on October 25, 2004. An RCE has been filed herewith. Please amend the application as follows:

CERTIFICATE OF MAILING 37 C.F.R. §1.8(a)

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail, postpaid in an envelope, addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on the date indicated below.

Date: 4/25/05



Frank V. DeRosa

IN THE CLAIMS:

1. (Currently Amended) A method for providing accelerated loading of an operating system, comprising the steps of:

maintaining a list of boot data used for booting a computer system;

initializing a central processing unit of the computer system;

preloading the boot data into a cache memory ~~upon~~ prior to completion of initialization of the central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device; and

servicing requests for boot data from the computer system using the preloaded boot data after completion of initialization of the central processing unit of the computer system, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data.

2. (Original) The method of claim 1, wherein the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof.

3. (Canceled)

4. (Previously Presented) The method of claim 1, wherein the method steps are performed by a data storage controller connected to the boot device.

5. (Original) The method of claim 1, further comprising the step of updating the list of boot data during the boot process.

6. (Original) The method of claim 5, wherein the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list.

7. (Original) The method of claim 5, wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system.

8. (Canceled)

9. (Original) The method of claim 1, wherein the method steps are program instructions that are tangibly embodied on a program storage device and readable by a machine to execute the method steps.

10. (Previously Presented) A method for providing accelerated launching of an application program, comprising the steps of:

maintaining a list of application data associated with an application program;

preloading the application data into a cache memory upon launching the

application program, wherein preloading the application data comprises accessing

compressed application data from a persistent storage device; and

servicing requests for application data from a computer system using the preloaded application data, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data.

11. (Canceled)

12. (Original) The method of claim 10, wherein the method steps are program instructions that are tangibly embodied on a program storage device and readable by a machine to execute the method steps.

13. (Currently Amended) A boot device controller for providing accelerated loading of an operating system of a host system, the boot device controller comprising:

a digital signal processor (DSP) or controller comprising a data compression engine (DCE) for compressing boot data stored to a boot device and for decompressing compressed boot data retrieved from the boot device;

a programmable volatile logic device, wherein the programmable volatile logic device is programmed by the DSP or controller prior to completion of initialization of a central processing unit of the host system, to (i) instantiate a first interface for operatively interfacing the boot device controller to the boot device and to (ii) instantiate a second interface for operatively interfacing the boot device controller to the host system;

a cache memory device; and

a non-volatile memory device, for storing logic code associated with the DSP or controller, the first interface and the second interface, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system, for preloading the compressed boot data into the cache memory device ~~upon~~ prior to completion of initialization of the central processing unit of the host system, and for decompressing the preloaded compressed boot data to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system.

14. (Canceled)

15. (Previously Presented) The boot device controller of claim 13, wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data.

16. (Canceled)

17. (New) The method of claim 1, further comprising:
maintaining a list of application data associated with an application program;
preloading the application data into the cache memory prior to completion of
initialization of the central processing unit of the computer system, wherein preloading the
application data comprises accessing compressed application data from a boot device; and
servicing requests for application data from the computer system using the
preloaded application data after completion of initialization of the central processing unit
of the computer system, wherein servicing requests comprises accessing compressed
application data from the cache and decompressing the compressed application data.

REMARKS

Claims 1, 2, 4-7, 9, 10, 12, 13 and 15 are pending in the application. Claims 1 and 13 have been amended. New claim 17 has been added. Examiner's reconsideration of the rejections in view of the above amendments and following remarks is respectfully requested.

Claim Objections

Claim 13 has been amended above, and the status identifier of claim has been changed to "Currently Amended".

Claim Rejections - 35 U.S.C. § 103

Claims 1-2, 4-7, 8-10, 12-13 and 15 stand rejected as being unpatentable over U.S. Patent No. 6,073,232 to Kroeker in view of U.S. Patent No. 6,434,695 to Esfahani, et al. At the very least, it is respectfully submitted that claims 1, 10 and 13 are patentable and non-obvious over the combination of Kroeker and Esfahani.

For instance, with respect to claim 1, the combination of Kroeker and Esfahani does not disclose or suggest *preloading the boot data into a cache memory prior to completion of initialization of a central processing unit of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device*, as essentially recited in claim 1. Although Esfahani arguably discloses loading compressed boot data into a RAM cache, it is submitted that Esfahani does not cure the deficiencies of Kroeker in that Esfahani does not disclose *preloading the boot data into a cache memory prior to completion of initialization of a central processing unit of the computer system*. Indeed, Esfahani specifically discloses (Col. 8, line 40, through Col. 9,

line 6, for example) that a *Boot Info file* (40) is located and loaded into *RAM* (12) after completion of initialization of the computer system, to begin booting the operating system.

The is in stark contrast to the claimed *preloading the boot data into a cache memory prior to completion of initialization of a central processing unit of the computer system*, as claimed in claim 1.

Further, with respect to claim 10, the combination of Kroeker and Esfahani does not teach or suggest *preloading the application data into a cache memory upon launching the application program, wherein preloading the application data comprises accessing compressed application data from a persistent storage device*, as essentially recited in claim 10. Indeed, both Kroeker and Esfahani are directed to booting operating systems. In contrast, the claimed invention is directed to a method for accelerated loading of an application program, which is not taught by Kroeker and Esfahani, alone or in combination.

Moreover, with respect to claim 13, the combination of Kroeker and Esfahani does not teach or suggest, e.g., *a programmable volatile logic device, wherein the programmable volatile logic device is programmed by the DSP or controller prior to completion of initialization of a central processing unit of the host system*, much less *the DSP or controller preloading the compressed boot data into the cache memory device prior to completion of initialization of the central processing unit of the host system, and decompressing the preloaded compressed boot data to service requests for boot data from the host system after completion of initialization of the central processing unit of the host system*, as essentially recited in claim 13.

Therefore, claims 1, 10 and 13 are patentable and non-obvious over the combination of Kroeker and Esfahani. Moreover, the remaining dependent claims are patentable over the cited combination at least by virtue of their dependence from respective base claims 1, 10 or 13.

Early and favorable consideration by the Examiner is respectfully urged. Should the Examiner believe that a telephone or personal interview may facilitate resolution of any remaining matters, it is requested that the Examiner contact Applicants' undersigned attorney.

Respectfully submitted,

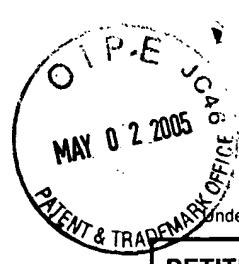


Frank V. DeRosa

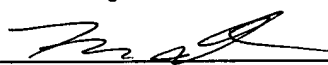
Reg. No. 43,584

Attorney for Applicant(s)

F. Chau & Associates, LLC
130 Woodbury Road
Woodbury, New York 11797
TEL.: (516) 692-8888
FAX: (516) 692-8889



Under the paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PETITION FOR EXTENSION OF TIME UNDER 37 CFR 1.136(a) FY 2005 <i>(Fees pursuant to the Consolidated Appropriations Act, 2005 (H.R. 4818).)</i>		Docket Number (Optional) 8011-15																									
Application Number 09/776,267		Filed February 2, 2001																									
For Systems and Methods for Accelerated Loading of Operating Systems and Application....																											
Art Unit 2115		Examiner S. Suryawanshi																									
<p>This is a request under the provisions of 37 CFR 1.136(a) to extend the period for filing a reply in the above identified application.</p> <p>The requested extension and fee are as follows (check time period desired and enter the appropriate fee below):</p> <table border="0"> <thead> <tr> <th></th> <th style="text-align: center;"><u>Fee</u></th> <th colspan="2" style="text-align: center;"><u>Small Entity Fee</u></th> </tr> </thead> <tbody> <tr> <td><input type="checkbox"/> One month (37 CFR 1.17(a)(1))</td> <td style="text-align: center;">\$120</td> <td style="text-align: center;">\$60</td> <td style="text-align: center;">\$ _____</td> </tr> <tr> <td><input type="checkbox"/> Two months (37 CFR 1.17(a)(2))</td> <td style="text-align: center;">\$450</td> <td style="text-align: center;">\$225</td> <td style="text-align: center;">\$ _____</td> </tr> <tr> <td><input checked="" type="checkbox"/> Three months (37 CFR 1.17(a)(3))</td> <td style="text-align: center;">\$1020</td> <td style="text-align: center;">\$510</td> <td style="text-align: center;">\$ <u>510.00</u></td> </tr> <tr> <td><input type="checkbox"/> Four months (37 CFR 1.17(a)(4))</td> <td style="text-align: center;">\$1590</td> <td style="text-align: center;">\$795</td> <td style="text-align: center;">\$ _____</td> </tr> <tr> <td><input type="checkbox"/> Five months (37 CFR 1.17(a)(5))</td> <td style="text-align: center;">\$2160</td> <td style="text-align: center;">\$1080</td> <td style="text-align: center;">\$ _____</td> </tr> </tbody> </table> <p><input checked="" type="checkbox"/> Applicant claims small entity status. See 37 CFR 1.27.</p> <p><input type="checkbox"/> A check in the amount of the fee is enclosed.</p> <p><input checked="" type="checkbox"/> Payment by credit card. Form PTO-2038 is attached.</p> <p><input type="checkbox"/> The Director has already been authorized to charge fees in this application to a Deposit Account.</p> <p><input type="checkbox"/> The Director is hereby authorized to charge any fees which may be required, or credit any overpayment, to Deposit Account Number <u>50-0679</u>. I have enclosed a duplicate copy of this sheet.</p> <p>WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.</p> <p>I am the <input type="checkbox"/> applicant/inventor.</p> <p><input type="checkbox"/> assignee of record of the entire interest. See 37 CFR 3.71. Statement under 37 CFR 3.73(b) is enclosed (Form PTO/SB/96).</p> <p><input checked="" type="checkbox"/> attorney or agent of record. Registration Number <u>43,584</u></p> <p><input type="checkbox"/> attorney or agent under 37 CFR 1.34. Registration number if acting under 37 CFR 1.34 _____</p> <p style="text-align: center;">  _____ Signature </p> <p style="text-align: right;"> _____ April 25, 2005 Date </p> <p> _____ Frank V. DeRosa Typed or printed name </p> <p style="text-align: right;"> _____ 516-692-8888 Telephone Number </p> <p>NOTE: Signatures of all the inventors or assignees of record of the entire interest or their representative(s) are required. Submit multiple forms if more than one signature is required, see below.</p> <p><input type="checkbox"/> Total of _____ forms are submitted.</p>					<u>Fee</u>	<u>Small Entity Fee</u>		<input type="checkbox"/> One month (37 CFR 1.17(a)(1))	\$120	\$60	\$ _____	<input type="checkbox"/> Two months (37 CFR 1.17(a)(2))	\$450	\$225	\$ _____	<input checked="" type="checkbox"/> Three months (37 CFR 1.17(a)(3))	\$1020	\$510	\$ <u>510.00</u>	<input type="checkbox"/> Four months (37 CFR 1.17(a)(4))	\$1590	\$795	\$ _____	<input type="checkbox"/> Five months (37 CFR 1.17(a)(5))	\$2160	\$1080	\$ _____
	<u>Fee</u>	<u>Small Entity Fee</u>																									
<input type="checkbox"/> One month (37 CFR 1.17(a)(1))	\$120	\$60	\$ _____																								
<input type="checkbox"/> Two months (37 CFR 1.17(a)(2))	\$450	\$225	\$ _____																								
<input checked="" type="checkbox"/> Three months (37 CFR 1.17(a)(3))	\$1020	\$510	\$ <u>510.00</u>																								
<input type="checkbox"/> Four months (37 CFR 1.17(a)(4))	\$1590	\$795	\$ _____																								
<input type="checkbox"/> Five months (37 CFR 1.17(a)(5))	\$2160	\$1080	\$ _____																								

This collection of information is required by 37 CFR 1.136(a). The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to take 6 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PATENT APPLICATION FEE DETERMINATION RECORD
 Substitute for Form PTO-875

Application or Docket Number
09/776,267

CLAIMS AS FILED – PART I

FOR	(Column 1) NUMBER FILED	(Column 2) NUMBER EXTRA
BASIC FEE (37 CFR 1.16(a))		
TOTAL CLAIMS (37 CFR 1.16(c))	16	0
INDEPENDENT CLAIMS (37 CFR 1.16(b))	3	0
MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(d))		

SMALL ENTITY	
RATE	FEE
	355
x \$ _____ =	
x \$ _____ =	
+ \$ _____ =	
TOTAL	355

OTHER THAN SMALL ENTITY	
RATE	FEE
	\$ _____
x \$ _____ =	
x \$ _____ =	
+ \$ _____ =	
TOTAL	

* If the difference in column 1 is less than zero, enter "0" in column 2.

CLAIMS AS AMENDED – PART II

AMENDMENT A	(Column 1)	(Column 2)	(Column 3)
	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total (37 CFR 1.16(c))	11	16	0
Independent (37 CFR 1.16(b))	3	3	0
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(d))			

SMALL ENTITY	
RATE	ADDITIONAL FEE
x \$ _____ =	
x \$ _____ =	
+ \$ _____ =	
TOTAL ADD'L FEE	0

OTHER THAN SMALL ENTITY	
RATE	ADDITIONAL FEE
x \$ _____ =	
x \$ _____ =	
+ \$ _____ =	
TOTAL ADD'L FEE	

5/2/05

AMENDMENT B	(Column 1)	(Column 2)	(Column 3)
	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total (37 CFR 1.16(c))	11	16	0
Independent (37 CFR 1.16(b))	3	3	0
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(d))			

SMALL ENTITY	
RATE	ADDITIONAL FEE
x \$ _____ =	
x \$ _____ =	
+ \$ _____ =	
TOTAL ADD'L FEE	0

OTHER THAN SMALL ENTITY	
RATE	ADDITIONAL FEE
x \$ _____ =	
x \$ _____ =	
+ \$ _____ =	
TOTAL ADD'L FEE	

AMENDMENT C	(Column 1)	(Column 2)	(Column 3)
	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
Total (37 CFR 1.16(c))			
Independent (37 CFR 1.16(b))			
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(d))			

SMALL ENTITY	
RATE	ADDITIONAL FEE
x \$ _____ =	
x \$ _____ =	
+ \$ _____ =	
TOTAL ADD'L FEE	

OTHER THAN SMALL ENTITY	
RATE	ADDITIONAL FEE
x \$ _____ =	
x \$ _____ =	
+ \$ _____ =	
TOTAL ADD'L FEE	

- * If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
- ** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".
- *** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".

The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

This collection of information is required by 37 CFR 1.16. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

174



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/776,267	02/02/2001	James J. Fallon	8011-15	9730

22150 7590 10/25/2004
F. CHAU & ASSOCIATES, LLC
130 WOODBURY ROAD
WOODBURY, NY 11797

EXAMINER

SURYAWANSHI, SURESH

ART UNIT PAPER NUMBER

2115

DATE MAILED: 10/25/2004

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No. 09/776,267	Applicant(s) FALLON ET AL.	
	Examiner Suresh K Suryawanshi	Art Unit 2115	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) Responsive to communication(s) filed on 8/16/04 amendments.
- 2a) This action is FINAL. 2b) This action is non-final.
- 3) Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) Claim(s) 1,2,4-7,9,10,12,13 and 15 is/are pending in the application.
4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) Claim(s) _____ is/are allowed.
- 6) Claim(s) 1,2,4-7,9,10,12,13 and 15 is/are rejected.
- 7) Claim(s) _____ is/are objected to.
- 8) Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) The specification is objected to by the Examiner.
- 10) The drawing(s) filed on 02 February 2001 is/are: a) accepted or b) objected to by the Examiner.
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

- 12) Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
a) All b) Some * c) None of:
1. Certified copies of the priority documents have been received.
 2. Certified copies of the priority documents have been received in Application No. _____.
 3. Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).
- * See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

- | | |
|--|---|
| 1) <input checked="" type="checkbox"/> Notice of References Cited (PTO-892) | 4) <input type="checkbox"/> Interview Summary (PTO-413)
Paper No(s)/Mail Date. _____ |
| 2) <input type="checkbox"/> Notice of Draftsperson's Patent Drawing Review (PTO-948) | 5) <input type="checkbox"/> Notice of Informal Patent Application (PTO-152) |
| 3) <input type="checkbox"/> Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08)
Paper No(s)/Mail Date _____ | 6) <input type="checkbox"/> Other: _____ |

DETAILED ACTION

1. Claims 1-2, 4-7, 9-10, 12-13 and 15 are presented for examination.

Drawings

2. This application, filed under former 37 CFR 1.60, lacks formal drawings. The informal drawings filed in this application are acceptable for examination purposes. When the application is allowed, applicant will be required to submit new formal drawings. In unusual circumstances, the formal drawings from the abandoned parent application may be transferred by the grant of a petition under 37 CFR 1.182.

Claim Objections

3. Claim 13 is objected to because of the following informalities: "Original" should be replaced with "Currently amended" as the claim has been amended. Appropriate correction is required.

Art Unit: 2115

Claim Rejections - 35 USC § 103

4. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

5. Claims 1-2, 4-7, 9-10, 12-13 and 15 are rejected under 35 U.S.C. 103(a) as being unpatentable over Krockner et al (US Patent no 6,073,232) in view of Esfahani et al (US Patent no 6,434,695 B1).

6. As per claim 1, Krockner et al teach

maintaining a list of boot data used for booting a computer system [col. 2, lines 30-47; col. 5, lines 1-7; a prefetch table containing a listing of the disk locations and length of data records that were requested by the host computer in the immediately previous power-on/reset];

preloading the boot data upon initialization of the computer system [col. 2, lines 36-41; col. 3, lines 30-39; col. 5, lines 17-21; data is preloaded into the RAM cache according to the prefetch table]; and

servicing requests for boot data from the computer system using the preloaded boot data [col. 2, lines 41-47; col. 3, lines 30-39; data is communicated from the cache to the host computer].

Krocker et al do not disclose about accessing compressed boot data and decompressing the compressed boot data. However, Esfahani et al clearly disclose about loading a compressed boot data into a RAM cache and then the boot data is decompressed and executed [col. 2, lines 5-13, 63, 67; col. 10, line 65 – col. 11, line 4]. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the cited references as both are directed to minimize a computer's initial program load time or shortening the load time of the computer programs from a hard disk drive to a host computer. Moreover, the shortening load time method of Krocker et al by loading the program codes into the RAM cache according to the prefetch table will definitely be benefited with the method of reading compressed data into the RAM cache and then decompressing and executing as needed. This way, one may not only have needed data into a fast access memory but also a large amount of data to avoid frequent accessing the storage device(s).

7. As per claim 2, Krocker et al teach that the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof [col. 5, lines 41-51; requesting data records are part of a computer program such as DOS or Windows].

8. As per claim 4, Krocker et al teach that the method steps are performed by a data storage controller connected to the boot device [fig. 1; controller].

9. As per claim 5, Krocker et al teach the step of updating the list of boot data during the boot process [col. 8, lines 63-65; the prefetch table is updated].

10. As per claim 6, Krocker et al teach the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list [col. 8, lines 63-68; the prefetch table is updated].

11. As per claim 7, Krocker et al teach that the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system [col. 8; lines 63-65; updating the prefetch table].

12. As per claims 9 and 12, Krocker et al teach that the method steps are program instructions that are tangibly embodied on a program storage device and readable by a machine to execute the method steps [col. 9, lines 27-30; computer program].

13. As per claim 10, Krocker et al teach

maintaining a list of application data associated with an application program [col. 11; lines 30-34; a prefetch table containing disk storage location and length of the data records requested by the application program];

preloading the application data upon launching the application program [col. 11, lines 46-50; preloading the data cache prior to receiving a read command from the application]; and

servicing requests for application data from a computer system using the preloaded application data [col. 11, lines 51-57; communicating the prestored data records of the application from the data cache to the host computer].

Krocker et al do not disclose about accessing compressed data and decompressing the compressed data. However, Esfahani et al clearly disclose about loading a compressed data into a RAM cache and then the compressed data is decompressed and executed [col. 2, lines 5-13, 63, 67; col. 10, line 65 – col. 11, line 4]. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the cited references as both are directed to minimize a computer's initial program load time or shortening the load time of the computer programs from a hard disk drive to a host computer. Moreover, the shortening load time method of Krocker et al by loading the program codes into the RAM cache according to the prefetch table will definitely be benefited with the method of reading compressed data into the RAM cache and then decompressing and executing as needed. This way, one may not only have needed data into a fast access memory but also a large amount of data to avoid frequent accessing the storage device(s).

Art Unit: 2115

14. As per claim 13, Krockner et al teach

a digital signal processor (DSP) [fig. 1; host computer];

a programmable logic device [fig. 1, disk], wherein the programmable logic device is programmed by the digital signal processor [fig. 1; host computer] to (i) instantiate a first interface for operatively interfacing the boot device controller to a boot device [fig. 1; controller] and to (ii) instantiate a second interface for operatively interfacing the boot device controller to the host system [inherent to the system as a bus interface is used to interface the controller with host computer]; and

a non-volatile memory device [fig. 1; disk;], for storing logic code associated with the DSP, the first interface and the second interface, wherein the logic code comprises instructions executable by the DSP for maintaining a list of boot data used for booting the host system [col. 5, lines 1-7; a prefetch table is read from a reserved area of the disks], preloading the boot data upon initialization of the host system [col. 2, lines 36-41; col. 3, lines 30-39; col. 5, lines 17-21; data is preloaded into the cache according to the prefetch table], and servicing requests for boot data from the host system using the preloaded boot data [col. 2, lines 41-47; col. 3, lines 30-39].

Krockner et al do not disclose about accessing compressed boot data and decompressing the compressed boot data. However, Esfahani et al clearly disclose about loading a compressed boot data into a RAM cache and then the boot data is decompressed and executed [col. 2, lines 5-

Art Unit: 2115

13, 63, 67; col. 10, line 65 – col. 11, line 4]. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the cited references as both are directed to minimize a computer's initial program load time or shortening the load time of the computer programs from a hard disk drive to a host computer. Moreover, the shortening load time method of Krockner et al by loading the program codes into the RAM cache according to the prefetch table will definitely be benefited with the method of reading compressed data into the RAM cache and then decompressing and executing as needed. This way, one may not only have needed data into a fast access memory but also a large amount of data to avoid frequent accessing the storage device(s).

15. As per claim 15, Krockner et al teach that the logic code in the non-volatile memory device further comprises program instructions executable by the DSP for maintaining a list of application data associated with an application program [col. 11; lines 30-34; a prefetch table containing disk storage location and length of the data records requested by the application program]; preloading the application data upon launching the application program [col. 11, lines 46-50; preloading the data cache prior to receiving a read command from the application], and servicing requests for the application data from the host system using the preloaded application data col. 11, lines 51-57; communicating the prestored data records of the application from the data cache to the host computer].

Art Unit: 2115

Conclusion

THIS ACTION IS MADE FINAL. Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a). The practice of automatically extending the shortened statutory period an additional month upon the filing of a timely first response to a final rejection has been discontinued by the Office. See 1021 TMOG 35.

A SHORTENED STATUTORY PERIOD FOR RESPONSE TO THIS FINAL ACTION IS SET TO EXPIRE THREE MONTHS FROM THE DATE OF THIS ACTION. IN THE EVENT A FIRST RESPONSE IS FILED WITHIN TWO MONTHS OF THE MAILING DATE OF THIS FINAL ACTION AND THE ADVISORY ACTION IS NOT MAILED UNTIL AFTER THE END OF THE THREE-MONTH SHORTENED STATUTORY PERIOD, THEN THE SHORTENED STATUTORY PERIOD WILL EXPIRE ON THE DATE THE ADVISORY ACTION IS MAILED, AND ANY EXTENSION FEE PURSUANT TO 37 CFR 1.136(a) WILL BE CALCULATED FROM THE MAILING DATE OF THE ADVISORY ACTION. IN NO EVENT WILL THE STATUTORY PERIOD FOR RESPONSE EXPIRE LATER THAN SIX MONTHS FROM THE DATE OF THIS FINAL ACTION.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Suresh K Suryawanshi whose telephone number is 571-272-3668. The examiner can normally be reached on 9:00am - 5:30pm.

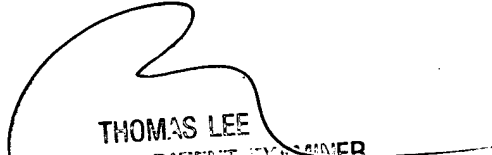
If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Thomas C. Lee can be reached on 571-272-3667. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Art Unit: 2115

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

sks

October 18, 2004


THOMAS LEE
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100

Notice of References Cited	Application/Control No. 09/776,267	Applicant(s)/Patent Under Reexamination FALLON ET AL.	
	Examiner Suresh K Suryawanshi	Art Unit 2115	Page 1 of 1

U.S. PATENT DOCUMENTS

*	Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
A	US-6,434,695	08-2002	Esfahani et al.	713/2
B	US-			
C	US-			
D	US-			
E	US-			
F	US-			
G	US-			
H	US-			
I	US-			
J	US-			
K	US-			
L	US-			
M	US-			

FOREIGN PATENT DOCUMENTS

*	Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
N					
O					
P					
Q					
R					
S					
T					

NON-PATENT DOCUMENTS

*	Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
U	
V	
W	
X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.



UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
 UNITED STATES PATENT AND TRADEMARK OFFICE
 WASHINGTON, D.C. 20231
 www.uspto.gov



Bib.Data Sheet

CONFIRMATION NO. 9730

SERIAL NUMBER: 09/776,267	FILING DATE 02/02/2001 RULE	CLASS 713	GROUP ART UNIT 2182	ATTORNEY DOCKET NO. 8011-15
-------------------------------------	---	---------------------	-------------------------------	---------------------------------------

APPLICANTS
 James J. Fallon, Armonk, NY;
 John Buck, Oceanside, NY;
 Paul F. Pickel, Bethpage, NY;
 Stephen J. McErlain, New York, NY;

**** CONTINUING DATA *******
 THIS APPLN CLAIMS BENEFIT OF 60/180,114 02/03/2000
yes yes

**** FOREIGN APPLICATIONS *******
None yes

IF REQUIRED, FOREIGN FILING LICENSE GRANTED ** 03/09/2001 **** SMALL ENTITY ****

Foreign Priority claimed 35 USC 119 (a-d) conditions met Verified and Acknowledged	<input type="checkbox"/> yes <input checked="" type="checkbox"/> no <input type="checkbox"/> yes <input type="checkbox"/> no <input type="checkbox"/> Met after Allowance	STATE OR COUNTRY NY	SHEETS DRAWING 13	TOTAL CLAIMS 18 11	INDEPENDENT CLAIMS 3
--	--	------------------------	----------------------	-------------------------------------	-------------------------

ADDRESS
 Frank Chau, Esq.
 F. CHAU & ASSOCIATES, LLP
 Suite 501
 1900 Hempstead Turnpike
 East Meadow, NY 11554

TITLE
 Systems and methods for accelerated loading of operating systems and application programs

FILING FEE RECEIVED 420	FEES: Authority has been given in Paper No. _____ to charge/credit DEPOSIT ACCOUNT No. _____ for following:	<input type="checkbox"/> All Fees
		<input type="checkbox"/> 1.16 Fees (Filing)
		<input type="checkbox"/> 1.17 Fees (Processing Ext. of time)
		<input type="checkbox"/> 1.18 Fees (Issue)
		<input type="checkbox"/> Other _____
		<input type="checkbox"/> Credit

ISSUE SLIP STAPLE AREA (for additional cross references)

POSITION	INITIALS	ID NO.	DATE
FEE DETERMINATION			
O.I.P.E. CLASSIFIER	ASD		3/1/01
FORMALITY REVIEW	NV	778	3/9/01
RESPONSE FORMALITY REVIEW	ML	1030	5-22-01

INDEX OF CLAIMS

- ✓ Rejected
- = Allowed
- (Through numeral) ... Canceled
- ÷ Restricted
- N Non-elected
- I Interference
- A Appeal
- O Objected

Claim	Date
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	
41	
42	
43	
44	
45	
46	
47	
48	
49	
50	

Claim	Date
51	
52	
53	
54	
55	
56	
57	
58	
59	
60	
61	
62	
63	
64	
65	
66	
67	
68	
69	
70	
71	
72	
73	
74	
75	
76	
77	
78	
79	
80	
81	
82	
83	
84	
85	
86	
87	
88	
89	
90	
91	
92	
93	
94	
95	
96	
97	
98	
99	
100	

Claim	Date
101	
102	
103	
104	
105	
106	
107	
108	
109	
110	
111	
112	
113	
114	
115	
116	
117	
118	
119	
120	
121	
122	
123	
124	
125	
126	
127	
128	
129	
130	
131	
132	
133	
134	
135	
136	
137	
138	
139	
140	
141	
142	
143	
144	
145	
146	
147	
148	
149	
150	

Best Available Copy

If more than 150 claims or 10 actions
staple additional sheet here

(LEFT INSIDE)



Best Available Copy

SEARCHED

Class	Sub.	Date	Exmr.
713	2	11/28/04	SBS
713	1	11/24/04	SBS
712	600	11/24/04	SBS
711	113	11/23/04	SBS
711	120	11/24/04	SBS
Updated Search		9/21/04 9/22/04 9/27/04 10/19/04	SBS

SEARCH NOTES (INCLUDING SEARCH STRATEGY)

	Date	Exmr.
East USPAT US-PGPHS EPOS IPO IBM-TSP NYL	11/28/04 11/24/04 11/24/04 11/24/04	SBS
Updated Search	9/21/04 9/22/04 9/27/04 10/19/04	SBS

INTERFERENCE SEARCHED

Class	Sub.	Date	Exmr.

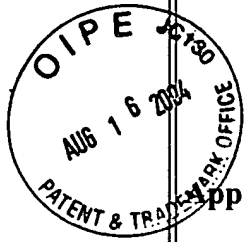
(RIGHT OUTSIDE)

	Hits	Search Text	DBs	Time Stamp
1	4	(accelerat\$3 adj load\$3). near3 ((operating adj system) os application)	USPAT	2004/09/22 11:33
2	0	6748457.URPN.	USPAT	2004/09/21 15:08
3	5	("5187793" "5226176" "5530845" "5652857" "6442659").PN.	USPAT	2004/09/21 15:09
4	5	(accelerat\$3 adj load\$3) near3 ((operating adj system) os application)	US-PGPUB; EPO; JPO; IBM_TDB	2004/09/21 15:22
5	0	cache with (compressed adj boot)	USPAT	2004/09/21 15:26
6	0	cache same (compressed adj boot)	USPAT	2004/09/21 15:35
7	104	cache with (compressed adj1 data)	USPAT	2004/09/21 17:23
8	0	cache with (compressed adj1 data) with pre\$1load\$3	USPAT	2004/09/21 15:26
9	5	cache with (compressed adj1 data) with load\$3	USPAT	2004/09/21 15:30
10	2	cache with (compressed adj1 data) with load\$3	US-PGPUB; EPO; JPO; IBM_TDB	2004/09/21 15:30
11	0	compress\$3 adj boot adj data	USPAT	2004/09/21 15:37
12	1	preload\$3 near3 (boot adj data)	USPAT	2004/09/21 15:39
13	2	(fast\$2 adj load\$3) near3 ((operating adj system) os application)	USPAT	2004/09/21 17:20
14	1	(quick adj load\$3) near3 ((operating adj system) os application)	USPAT	2004/09/21 17:20
15	27	(cache with (compressed adj1 data)).clm.	USPAT	2004/09/27 15:25
16	142	(fast\$2 quick\$2) adj boot\$3	USPAT	2004/09/22 11:04
17	5	(fast\$2 quick\$2) adj boot\$3 with cache	USPAT	2004/09/22 11:22
18	23	(fast\$2 quick\$2) adj boot\$3 same (computer laptop pc desktop workstation notebook)	USPAT	2004/09/22 11:19
19	24	(fast\$2 quick\$2) adj boot\$3 same (computer laptop pc desktop workstation notebook)	US-PGPUB; EPO; JPO; IBM_TDB	2004/09/22 11:19
20	3	(fast\$2 quick\$2) adj boot\$3 with cache	US-PGPUB; EPO; JPO; IBM_TDB	2004/09/22 11:22
21	3	pre\$1load\$3 near3 (initializ\$5 adj data)	USPAT	2004/09/22 11:37

	Hits	Search Text	DBs	Time Stamp
22	101	load\$3 near3 (initializ\$5 adj data)	USPAT	2004/09/22 11:37
23	0	load\$3 near3 (initializ\$5 adj data) with cache	USPAT	2004/09/22 11:36
24	0	load\$3 near3 (initializ\$5 adj data) same cache	USPAT	2004/09/22 11:36
25	1	pre\$1load\$3 near3 (initializ\$5 adj data)	US-PGPUB; EPO; JPO; IBM_TDB	2004/09/22 11:37
26	0	load\$3 near3 (boot\$3 adj data) with cache	USPAT	2004/09/22 11:38
27	1	load\$3 near3 (boot\$3 adj data) same cache	USPAT	2004/09/22 11:38
28	42	BIOS near5 compressed	USPAT	2004/09/27 15:26
29	3483	(load\$3 stor\$3) near3 (compressed adj (data image))	USPAT	2004/09/27 16:54
30	1456	((load\$3 stor\$3) near3 (compressed adj (data image))) near3 (ram (system adj memory) cache memory)	USPAT	2004/09/27 16:56
31	1348	((load\$3 stor\$3) near3 (compressed adj (data image))) near3 memory	USPAT	2004/09/27 16:56
32	135	((load\$3 stor\$3) near3 (compressed adj (data image))) near3 ram	USPAT	2004/09/27 17:07
33	16	((load\$3 stor\$3) near3 (compressed adj (data image))) near3 (system adj memory)	USPAT	2004/09/27 17:02
34	0	((load\$3 stor\$3) near3 (compressed adj (boot adj (data image)))) near3 (system adj memory)	USPAT	2004/09/27 17:41
35	0	((load\$3 stor\$3) near3 (compressed adj (boot adj (data image))))	USPAT	2004/09/27 17:03
36	86	((load\$3 stor\$3) near3 (compressed adj data)) near3 ram	USPAT	2004/09/27 17:46
37	25	((load\$3 stor\$3) near3 (compressed adj data)) near3 cache	USPAT	2004/09/27 17:34
38	3	((load\$3 stor\$3) near3 (compressed adj image)) near3 cache	USPAT	2004/09/27 17:44
39	3	((load\$3 stor\$3) near3 (compressed adj (initializ\$5 boot)))	USPAT	2004/09/27 17:42
40	0	((load\$3 stor\$3) near3 (compressed near2 ((system boot) adj image))) near3 cache	USPAT	2004/09/27 17:44
41	5	((load\$3 stor\$3) near3 (compressed near2 ((system boot) adj image)))	USPAT	2004/09/27 17:44
42	66	((load\$3 stor\$3) near3 (compressed adj2 image)) near3 ram	USPAT	2004/09/27 17:47
43	4	6073232.uref.	USPAT	2004/10/18 09:54

	Hits	Search Text	DBs	Time Stamp
44	1	6539456.URPN.	USPAT	2004/10/18 10:01
45	6	("5307497" "6061788" "6073232" "6172936" "6189100" "6226740").PN.	USPAT	2004/10/18 10:01

41 2115 ✓



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants: Fallon et al.

Examiner: Suryawanshi, Suresh

Serial No.: 09/776,267

Group Art Unit: 2115

Filed: February 2, 2001

Docket: 8011-15

**For: SYSTEMS AND METHODS FOR ACCELERATED LOADING OF
OPERATING SYSTEMS AND APPLICATION PROGRAMS**

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

RECEIVED

AUG 20 2004

Technology Center 2100

AMENDMENT

This is a response to the Office Action mailed on February 10, 2004. Please amend the application as follows:

CERTIFICATE OF MAILING 37 C.F.R. § 1.8(a)

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail, postpaid in an envelope, addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on August 10, 2004.

Date: 8/10/04

Frank V. DeRosa
Frank V. DeRosa

IN THE CLAIMS:

1. (Currently Amended) A method for providing accelerated loading of an operating system, comprising the steps of:

maintaining a list of boot data used for booting a computer system;

preloading the boot data into a cache memory upon initialization of the computer system, wherein preloading the boot data comprises accessing compressed boot data from a boot device;

and

servicing requests for boot data from the computer system using the preloaded boot data, wherein servicing requests comprises accessing compressed boot data from the cache and decompressing the compressed boot data.

2. (Original) The method of claim 1, wherein the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof.

3. (Canceled)

4. (Currently Amended) The method of claim 1 3, wherein the method steps are performed by a data storage controller connected to the boot device.

5. (Original) The method of claim 1, further comprising the step of updating the list of boot data during the boot process.

6. (Original) The method of claim 5, wherein the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list.

7. (Original) The method of claim 5, wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system.

8. (Canceled)

9. (Original) The method of claim 1, wherein the method steps are program instructions that are tangibly embodied on a program storage device and readable by a machine to execute the method steps.

10. (Currently Amended) A method for providing accelerated launching of an application program, comprising the steps of:

maintaining a list of application data associated with an application program;

preloading the application data into a cache memory upon launching the application program, wherein preloading the application data comprises accessing compressed application data from a persistent storage device; and

servicing requests for application data from a computer system using the preloaded application data, wherein servicing requests comprises accessing compressed application data from the cache and decompressing the compressed application data.

11. (Canceled)

12. (Original) The method of claim 10, wherein the method steps are program instructions that are tangibly embodied on a program storage device and readable by a machine to execute the method steps.

13. (Original) A boot device controller for providing accelerated loading of an operating system of a host system, the boot device controller comprising:

a digital signal processor (DSP) or controller comprising a data compression engine (DCE) for compressing boot data stored to a boot device and for decompressing compressed boot data retrieved from the boot device;

a programmable logic device, wherein the programmable logic device is programmed by the DSP or controller ~~digital signal processor~~ to (i) instantiate a first interface for operatively interfacing the boot device controller to a the boot device and to (ii) instantiate a second interface for operatively interfacing the boot device controller to the host system;

a cache memory device; and

a non-volatile memory device, for storing logic code associated with the DSP or controller, the first interface and the second interface, wherein the logic code comprises instructions executable by the DSP or controller for maintaining a list of boot data used for booting the host system, for preloading the compressed boot data into the cache memory device upon initialization of the host system, and for decompressing the preloaded compressed boot data

~~to service and servicing requests for boot data from the host system using the preloaded compressed boot data.~~

14. (Canceled)

15. (Currently Amended) The boot device controller of claim 13, wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP or controller for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for the application data from the host system using the preloaded application data.

16. (Canceled)

IN THE SPECIFICATION:

(i) Please amend the paragraph on Page 9, line 23, through Page 10, line 23, as follows:

The present invention is directed to data storage controllers that provide increased data storage/retrieval rates that are not otherwise achievable using conventional disk controller systems and protocols to store/retrieve data to/from mass storage devices. The concept of “accelerated” data storage and retrieval was introduced in ~~co~~pending U.S. Patent Application Serial No. 09/266,394, filed March 11, 1999, entitled “System and Methods For Accelerated Data Storage and Retrieval”, which is now U.S. Patent No. 6,601,104 and ~~co~~pending U.S. Patent Application Serial No. 09/481,243, filed January 11, 2000, entitled “System and Methods For Accelerated Data Storage and Retrieval,” which is now U.S. Patent No. 6,604,158, both of which are commonly assigned and incorporated herein by reference. In general, as described in the above-incorporated applications, “accelerated” data storage comprises receiving a digital data stream at a data transmission rate which is greater ~~that~~ than the data storage rate of a target storage device, compressing the input stream at a compression rate that increases the effective data storage rate of the target storage device and storing the compressed data in the target storage device. For instance, assume that a mass storage device (such as a hard disk) has a data storage rate of 20 megabytes per second. If a storage controller for the mass storage device is capable of compressing an input data stream with an average compression rate of 3:1, then data can be stored in the mass storage device at a rate of 60 megabytes per second, thereby effectively increasing the storage bandwidth (“storewidth”) of the mass storage device by a factor of three. Similarly, accelerated data retrieval comprises retrieving a compressed digital data stream from a target storage device at the rate equal to, e.g., the data access rate of the target storage device and then decompressing the compressed data at a rate that increases the effective data access rate of the target storage device. Advantageously, accelerated data storage/retrieval mitigates the traditional bottleneck associated with, e.g., local and network disk accesses.

(ii) Please amend the paragraph on Page 11, lines 1-13, as follows:

Referring now to Fig. 1, a high-level block diagram illustrates a data storage controller 10 according to one embodiment of the present invention. The data storage controller 10 comprises a data compression engine 12 for compressing/decompressing data (preferably in real-time or psuedo real-time) stored/retrieved from a hard disk 11 (or any other type of mass storage device) to provide accelerated data storage/retrieval. The DCE 12 preferably employs the data compression/decompression techniques disclosed in U.S. Serial No. 09/210,491 entitled "Content Independent Data Compression Method and System," filed on December 11, 1998, which is now U.S. Patent No. 6,195,024 which is commonly assigned and which is fully incorporated herein by reference. It is to be appreciated that the compression and decompression systems and methods disclosed in U.S. Serial No. 09/210,491 are suitable for compressing and decompressing data at rates, which provide accelerated data storage and retrieval. A detailed discussion of a preferred "content independent" data compression process will be provided below.

(iii) Please amend the paragraph on Page 14, line 23, through Page 15, line 11, as follows:

As discussed in greater detail below, upon host computer power-up or external user reset, the data storage controller 10 initializes the onboard interfaces 14, 15 prior to release of the external host bus 16 from reset. The processor of the host computer then requests initial data from the disk 11 to facilitate the computer's boot-up sequence. The host computer requests disk data over the Bus 16 via a command packet issued from the host computer. Command packets are preferably eight words long (in a preferred embodiment, each word comprises 32 bits). Commands are written from the host computer to the data storage controller 10 with the host computer as the Bus Master and the data storage controller 10 as the slave. The data storage controller 10 includes at least one Base Address Register (BAR) for decoding the address of a command queue of the data storage controller 10. The command queue resides within the cache 13 or within onboard memory of the DCE 12.

(iv) Please amend the paragraph on Page 18, lines 10-22, as follows:

The storage controller 20 further comprises computer reset and power up circuitry 28 (or “boot configuration circuit”) for controlling initialization (either cold or warm boots) of the host computer system and storage controller 20. A preferred boot configuration circuit and preferred computer initialization systems and protocols are described in U.S. Patent Application Serial No. 09/775,897 _____ (~~Attorney Docket No. 8011-10~~), filed concurrently herewith (~~Express Mail Label No. EL679454245US~~), which is commonly assigned and incorporated herein by reference.

Preferably, the boot configuration circuit 28 is employed for controlling the initializing and programming the programmable logic device 22 during configuration of the host computer system (i.e., while the CPU of the host is held in reset). The boot configuration circuit 28 ensures that the programmable logic device 22 (and possibly other volatile or partially volatile logic devices) is initialized and programmed before the bus 16 (such as a PCI bus) is fully reset.

(v) Please amend the paragraph on Page 22, lines 9-14, as follows:

Fig. 3 illustrates another embodiment of a data storage controller ~~30~~ 35 wherein the data storage controller 35 is embedded within the motherboard of the host computer system. This architecture provides the same functionality as the system of Fig. 2, and also adds the cost advantage of being embedded on the host motherboard. The system comprises additional RAM and ROM memory devices 23a, 24a, operatively connected to the DSP 21 via a local bus 25a.

(vi) Please amend the paragraph on Page 25, line 12, through Page 26, line 7, as follows:

Referring now to Fig. 6, a flow diagram illustrates a method for initializing the programmable logic device 22 according to one aspect of the invention. In the following discussion, it is assumed that the programmable logic device 22 is always reloaded, regardless of the type of boot process. Initially, in Fig. 6a, the DSP 21 is reset by asserting a DSP reset signal (step 50). Preferably, the DSP reset signal is generated by the boot circuit configuration circuit 28 (as described in the above-incorporated U.S. Serial No. 09/775,897 ~~_____~~ (Attorney Docket No. ~~8011-10~~). While the DSP reset signal is asserted (e.g., active low), the DSP is held in reset and is initialized to a prescribed state. Upon deassertion of the DSP Reset signal, the logic code for the DSP (referred to as the "boot loader") is copied from the non-volatile logic device 24 into memory residing in the DSP 21 (step 51). This allows the DSP to execute the initialization of the programmable logic device 22. In a preferred embodiment, the lower 1K bytes of EPROM memory is copied to the first 1k bytes of DSP's low memory (0x0000 0000 through 0x0000 03FF). As noted above, the memory mapping of the DSP 21 maps the CE1 memory space located at 0x9000 0000 through 0x9001 FFFF with the OTP EPROM. In a preferred embodiment using the Texas Instrument DSP TMS320c6211GFN-150, this ROM boot process is executed by the EDMA controller of the DSP. It is to be understood, however, that the EDMA controller may be instantiated in the programmable logic device (Xilinx), or shared between the DSP and programmable logic device.

(vii) Please amend the paragraph on Page 37, lines 14-19, as follows:

The DSP services request for its external bus from two requestors, the Enhanced Direct Memory Access (EDMA) Controller and an external shared memory device controller. The DSP can typically utilize the full 280 megabytes of bus bandwidth on an 8k through 64K byte (2k word through 16k word) burst basis. It should be noted that ~~the DSRA does not utilize the~~ SDRAM memory is not utilized for interim processing storage, and as such ~~only utilizes~~ bandwidth is only utilized in direct proportion to disk read and write commands.

(viii) Please amend the paragraph on Page 41, lines 12-18, as follows:

Once the data is preloaded, when the computer system bus issues its first read commands to the data storage controller seeking operating system data, the data will already be available in the cache memory of the data storage controller. The data storage controller will then be able to instantly start transmitting the data to the system bus. Before transmission to the bus, if the **data** was stored in compressed format on the boot device, the data will be decompressed. The process of preloading required (compressed) portions of the operating system significantly reduces the computer boot process time.

(ix) Please amend the paragraph on Page 49, line 21, through Page 50, line 12, as follows:

Again, it is to be understood that the embodiment of the data compression engine of Fig. 9 is exemplary of a preferred compression system which may be implemented in the present invention, and that other compression systems and methods known to those skilled in the art may be employed for providing accelerated data storage in accordance with the teachings herein. Indeed, in another embodiment of the compression system disclosed in the above-incorporated U.S. Patent No. 6,195,024 Serial No. 09/210,491, a timer is included to measure the time elapsed during the encoding process against an *a priori*-specified time limit. When the time limit expires, only the data output from those encoders (in the encoder module 125) that have completed the present encoding cycle are compared to determine the encoded data with the highest compression ratio. The time limit ensures that the real-time or pseudo real-time nature of the data encoding is preserved. In addition, the results from each encoder in the encoder module 125 may be buffered to allow additional encoders to be sequentially applied to the output of the previous encoder, yielding a more optimal lossless data compression ratio. Such techniques are discussed in greater detail in the above-incorporated U.S. Patent No. 6,195,024 Serial No. 09/210,491.

(x) Please amend the paragraph on Page 50, lines 13-20, as follows:

Referring now to FIG. 10, a detailed block diagram illustrates an exemplary decompression system that may be employed herein or accelerated data retrieval as disclosed in the above-incorporated U.S. Patent No. 6,195,024 ~~Serial No. 09/210,491~~. In this embodiment, the data compression engine 180 retrieves or otherwise accepts compressed data blocks from one or more data storage devices and inputs the data via a data storage interface. It is to be understood that the system processes the input data stream in data blocks that may range in size from individual bits through complete files or collections of multiple files. Additionally, the input data block size may be fixed or variable.

(xi) Please amend the paragraph on Page 51, line 20, through Page 52, line 6, as follows:

As with the data compression systems discussed in U.S. Patent No. 6,195,024 ~~Application Serial No. 09/210,491~~, the decoder module 165 may include multiple decoders of the same type applied in parallel so as to reduce the data decoding time. An output data buffer or cache 170 may be included for buffering the decoded data block output from the decoder module 165. The output buffer 70 then provides data to the output data stream. It is to be appreciated by those skilled in the art that the data compression system 180 may also include an input data counter and output data counter operatively coupled to the input and output, respectively, of the decoder module 165. In this manner, the compressed and corresponding decompressed data block may be counted to ensure that sufficient decompression is obtained for the input data block.

Please Amend the abstract on page 57 as follows:

ABSTRACT OF THE DISCLOSURE

Systems and methods ~~for providing~~ are provided for accelerated loading of operating system and application programs upon system boot or application launch. In one aspect, a method for providing accelerated loading of an operating system ~~includes~~ comprises the steps of: maintaining a list of boot data used for booting a computer system, ~~;~~ preloading the boot data upon initialization of the computer system, ; and servicing requests for boot data from the computer system using the preloaded boot data. The boot data may comprise program code associated with an operating system of the computer system, an application program, and a combination thereof. ~~In a preferred embodiment, the~~ The boot data is retrieved from a boot device and stored in a cache memory device. ~~In another aspect, the method for accelerated loading of an operating system comprises updating the list of boot data during the boot process, wherein updating comprises adding to the list any boot data requested by the computer system not previously stored in the list and/or removing from the list any boot data previously stored in the list and not requested by the computer system. In yet another aspect, the~~ The boot data is stored in a compressed format on the boot device and the preloaded boot data is decompressed prior to transmitting the preloaded boot data to the requesting system. ~~In another aspect, a method for providing accelerated launching of an application program comprises the steps of: maintaining a list of application data associated with an application program; preloading the application data upon launching the application program; and servicing requests for application data from a computer system using the preloaded application data.~~

REMARKS

Claims 1-16 are pending in the application. Claims 1, 4, 10, 13 and 15 have been amended. Claims 3, 8, 11, 14 and 16 have been canceled without prejudice. Examiner's reconsideration of the rejections and objections in view of the above amendments and following remarks is respectfully requested.

Specification Objections

The specification was objected to for the reasons set forth on pages 2-4 of the Office Action. Applicants have amended the specification to address the issues raised by the Examiner. Accordingly, withdrawal of the objections is respectfully requested.

Claim Rejections - 35 U.S.C. § 102

Claims 1-7, 9-10, and 12-15 stand rejected under 35 U.S.C. § 102(e) as being anticipated by U.S. Patent No. 6,073,232 to Kroeker et al, for the reasons set forth on pages 4-9 of the Office Action.

Claims 1, 10 and 13 are believed to be patentably distinct and patentable over Kroeker since, at the very minimum, Kroeker does not disclose or suggest *preloading compressed boot data or compressed application data into cache and servicing requests for boot/application data by decompressing the preloaded data*, as essentially claims in claims 1, 10, and/or 13. Indeed, even Examiner acknowledges on Page 9 of the Office Action that Kroeker does not disclose compressing/decompressing data. Accordingly, withdrawal of the anticipation rejections is respectfully requested.

Claim Rejections - 35 U.S.C. § 103

Claims 8, 11 and 16 stand rejected as being unpatentable over Kroeker in view of U.S.

Patent No. 6,173,381 to Dye. Claims 8, 11 and 16 have been canceled without prejudice and thus, the specific rejection is moot. However, Applicants will address the rejection with regard to amended claims 1, 10 and 13.

To establish a *prima facie* case of obviousness, various criteria must be met. For instance, the cited references *must* teach or suggest all the claim limitations. Further, there must be some suggestion or motivation in the references or in the knowledge generally available to one skilled in the art to combine their teachings. The teaching or suggestion to make the claimed combination must both be found in the prior art and not based on applicant's disclosure (see, e.g., MPEP 2141, 2143, 2143.03). Applicants respectfully submit that the combination of Kroeker and Dye does not render claims 1, 10 or 13 obvious. For example, Applicants submit that such combination does not fairly teach or *suggest preloading compressed boot data or compressed application into a cache memory device and decompressing the preloaded compressed data to service requests for such data from the host system.*

Although Dye arguably discloses a data controller having compression/decompression functions, it is submitted that other than through hindsight knowledge gleaned from Applicants' specification, there would be no motivation to combine data compression taught by Dye into the Kroeker system to derive the claimed inventions. Indeed, Dye discloses (Col. 11, lines 59-66, for example) a protocol whereby compressed data is accessed, decompressed and then stored in system memory. In contrast, with the claimed inventions, compressed boot/application data is first accessed and preloaded into a cache, and then decompressed to service requests for such data by the host system. These steps are not disclosed by either Dye or Kroecker.

The claimed inventions provide an advantage over the Kroecker system in terms of

accelerated processing. For example, with respect to accelerated booting, more acceleration could be obtained by accessing and storing the compressed data before decompressing to service requests. Indeed, a significant amount of boot data (in compressed form) can be quickly accessed and stored in cache (fast access memory) prior to commencement of the boot process.

Thereafter, when requested by the host system, the compressed data is readily accessible (short access time) and can be decompressed in real-time to service such requests.

In contrast, if the boot data was first compressed before preloaded into cache (e.g., Kroeker modified by the teachings of Dye (Col. 11, lines 60-66), less acceleration would be obtained because, e.g., the time for decompressing the boot data upon access from the boot device could add latency preventing more data from being accessed from the boot device before commencement of the boot process. In such case, if system requests for boot data must be serviced by first accessing compressed data from the boot device, the added latency in accessing compressed data from hard disk or other boot device (as opposed to cache) could slow the system. For at least these reasons, it is believed that the combination of Kroeker and Dye would not reasonable disclose, suggest or render obvious, claims 1, 10 or 13.

Early and favorable consideration by the Examiner is respectfully urged. Should the Examiner believe that a telephone or personal interview may facilitate resolution of any

remaining matters, it is requested that the Examiner contact Applicants' undersigned attorney.

Respectfully submitted,

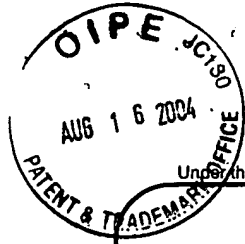


Frank V. DeRosa

Reg. No. 43,584

Attorney for Applicant(s)

F. Chau & Associates, LLC
130 Woodbury Road
Woodbury, New York 11797
TEL.: (516) 692-8888
FAX: (516) 692-8889



Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

CHANGE OF CORRESPONDENCE ADDRESS *Application*

Address to:
 Commissioner for Patents
 P.O. Box 1450
 Alexandria, VA 22313-1450.

Application Number	09/776,267
Filing Date	February 2, 2001
First Named Inventor	Fallon et al.
Art Unit	2115
Examiner Name	A. Elamin
Attorney Docket Number	8011-15

Please change the Correspondence Address for the above-identified patent application to:

Customer Number : 22150

RECEIVED

AUG 20 2004

OR

Firm or Individual Name F. Chau & Associates, LLC

Technology Center 2100

Address

Address

130 Woodbury Road

City

Woodbury

State

NY

Zip

11797

Country

US

Telephone

(516) 692-8888

Fax

(516) 692-8889

This form cannot be used to change the data associated with a Customer Number. To change the data associated with an existing Customer Number use "Request for Customer Number Data Change" (PTO/SB/124).

I am the:

- Applicant/Inventor
- Assignee of record of the entire interest.
Statement under 37 CFR 3.73(b) is enclosed. (Form PTO/SB/96).
- Attorney or Agent of record. Registration Number 43,584
- Registered practitioner named in the application transmittal letter in an application without an executed oath or declaration. See 37 CFR 1.33(a)(1). Registration Number _____

Typed or Printed Name Frank V. DeRosa

Signature

Date

8/10/04

Telephone

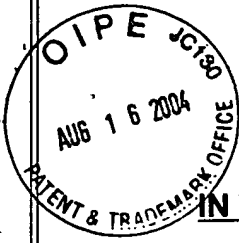
(516) 692-8888

NOTE: Signatures of all the inventors or assignees of record of the entire interest or their representative(s) are required. Submit multiple forms if more than one signature is required, see below*.

*Total of _____ forms are submitted.

This collection of information is required by 37 CFR 1.33. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 3 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICANT(S): Fallon et al. Examiner: A. Elamin
 SERIAL NO.: 09/776,267 Group Art Unit: 2115
 FILED: February 2, 2001 Dated: August 10, 2004
 FOR: SYSTEMS AND METHODS FOR ACCELERATED LOADING OF OPERATING SYSTEMS AND APPLICATION PROGRAMS

Mail Stop Amendment
 Commissioner for Patents
 P.O. Box 1450
 Alexandria, VA 22313-1450

RECEIVED

AUG 20 2004

Technology Center 2100

AMENDMENT TRANSMITTAL FORM

Sir:

Transmitted herewith is an amendment in the above-identified application.

- Small entity status of this application under 37 C.F.R. §§1.9 and 1.27 has been established by a verified statement previously submitted.
- A verified statement to establish small entity under 37 C.F.R. §§1.9 and 1.27 is enclosed.
- No additional fee is required.

For	Claims Remaining After Amendment	Highest No. Previously Paid For	Present Extra	Rate (Small Entity)	Addit. Fee	Rate	Addit. Fee
TOTAL CLAIMS*	11	20	0	x 9 =	\$0	x 18 =	\$0
INDEPENDENT CLAIMS	3	3	0	x 43	\$0	x 86 =	\$0
<input type="checkbox"/> First Presentation of Multiple Dep. Claim				145		290	\$0

* If the entry in Col. 1 is less than entry in Col. 2, write "0" in Col. 3.
 ** If the "Highest No. Previously Paid for" IN THIS SPACE is less than 20, enter "20".
 *** If the "Highest No. Previously Paid For" IN THIS SPACE is less than 3; enter "3".
 The Highest No. Previously Paid For" (Total or indep.) is the highest number found in the appropriate box in Col. 1 of a prior amendment or the number of claims originally filed.

CERTIFICATE OF MAILING UNDER 37 C.F.R. §1.8(a)

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail, postpaid in an envelope, addressed to the: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on August 10, 2004.

Dated: August 10, 2004


 Frank V. DeRosa

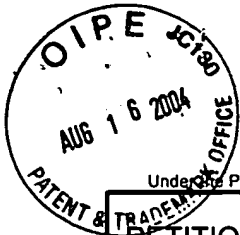
- Please charge Deposit Account No. 50-0679 in the amount of \$____. Two (2) copies of this sheet are enclosed.
- The amount of \$_____ is authorized to be charged to a Credit Card. Form PTO-2038 is enclosed.
- Please charge any deficiency as well as any other fee(s) which may become due under 37 C.F.R. §§1.16 and/or 1.17 at any time during the pendency of this application, or credit any overpayment of such fee(s) to Deposit Account No. 50-0679. Also, in the event any extensions of time for responding are required for the pending application(s), please treat this paper as a petition to extend the time as required and charge Deposit Account No. 50-0679 therefor. TWO (2) COPIES OF THIS SHEET ARE ENCLOSED.

F. CHAU & ASSOCIATES, LLC
130 Woodbury Road
Woodbury, NY 11797
(516) 692-8888

Respectfully submitted,



Frank V. DeRosa
Reg No. 43,584
Attorney for Applicant(s)



Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PETITION FOR EXTENSION OF TIME UNDER 37 CFR 1.136(a)		Docket Number (Optional) 8011-15	
In re Application of Fallon et al.			
Application Number 09/776,267	Filed 2/2/01		
For Systems and Methods for Accelerated Loading of...			
Group Art Unit 2115	Examiner A. Elamin		

This is a request under the provisions of 37 CFR 1.136(a) to extend the period for filing a reply in the above identified application.

The requested extension and appropriate non-small-entity fee are as follows (check time period desired):

- | | | |
|--|--|-------------|
| <input type="checkbox"/> One month (37 CFR 1.17(a)(1)) | RECEIVED
AUG 20 2004
Technology Center 2100 | \$ 110.00 |
| <input type="checkbox"/> Two months (37 CFR 1.17(a)(2)) | | \$ 420.00 |
| <input checked="" type="checkbox"/> Three months (37 CFR 1.17(a)(3)) | | \$ 950.00 |
| <input type="checkbox"/> Four months (37 CFR 1.17(a)(4)) | | \$ 1,480.00 |
| <input type="checkbox"/> Five months (37 CFR 1.17(a)(5)) | | \$ 2,010.00 |

- Applicant claims small entity status. See 37 CFR 1.27. Therefore, the fee amount shown above is reduced by one-half, and the resulting fee is: \$ 475.00
 - A check in the amount of the fee is enclosed.
 - Payment by credit card. Form PTO-2038 is attached.
 - The Commissioner has already been authorized to charge fees in this application to a Deposit Account.
 - The Commissioner is hereby authorized to charge any fees which may be required, or credit any overpayment, to Deposit Account Number _____.
- I have enclosed a duplicate copy of this sheet.

- I am the applicant/inventor
- assignee of record of the entire interest. See 37 CFR 3.71. Statement under 37 CFR 3.73(b) is enclosed. (Form PTO/SB/96).
 - attorney or agent of record.
 - attorney or agent under 37 CFR 1.34(a). Registration number if acting under 37 CFR 1.34(a) _____.

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

August 10, 2004
Date

Signature

08/16/2004 CCHAU1 00000064 09776267
01 FC:2258 475.00 OP

Frank V. DeRosa
Typed or printed name

NOTE: Signatures of all the inventors or assignees of record of the entire interest or their representative(s) are required. Submit multiple forms if more than one signature is required, see below.

Total of _____ forms are submitted.

CERTIFICATE OF MAILING UNDER 37 C.F.R. §1.8(a)

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail, postpaid in an envelope, addressed to the: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on August 10, 2004.

Dated: August 10, 2004

Frank V. DeRosa

PATENT APPLICATION FEE DETERMINATION RECORD
Effective October 1, 2000

Application or Docket Number

09/776267

CLAIMS AS FILED - PART I

	(Column 1)	(Column 2)
TOTAL CLAIMS	16	
FOR	NUMBER FILED	NUMBER EXTRA
TOTAL CHARGEABLE CLAIMS	16 minus 20=	0
INDEPENDENT CLAIMS	3 minus 3=	0
MULTIPLE DEPENDENT CLAIM PRESENT <input type="checkbox"/>		

SMALL ENTITY TYPE

OR OTHER THAN SMALL ENTITY

RATE	FEE		RATE	FEE
BASIC FEE	355.00	OR	BASIC FEE	710.00
X\$ 9=		OR	X\$18=	
X40=		OR	X80=	
+135=		OR	+270=	
TOTAL	355.00	OR	TOTAL	

* If the difference in column 1 is less than zero, enter "0" in column 2

CLAIMS AS AMENDED - PART II

	(Column 1)	(Column 2)	(Column 3)
AMENDMENT A	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
	Total	11 Minus 16 =	0
	Independent	3 Minus 3 =	0
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM <input type="checkbox"/>			

SMALL ENTITY

OR OTHER THAN SMALL ENTITY

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X\$ 9=		OR	X\$18=	
X40=		OR	X80=	
+135=		OR	+270=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

	(Column 1)	(Column 2)	(Column 3)
AMENDMENT B	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
	Total		
	Independent		
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM <input type="checkbox"/>			

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X\$ 9=		OR	X\$18=	
X40=		OR	X80=	
+135=		OR	+270=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

	(Column 1)	(Column 2)	(Column 3)
AMENDMENT C	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
	Total		
	Independent		
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM <input type="checkbox"/>			

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X\$ 9=		OR	X\$18=	
X40=		OR	X80=	
+135=		OR	+270=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
 ** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
 *** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."
 The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in column 1.

Best Available Copy

BEST AVAILABLE COPY



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/776,267	02/02/2001	James J. Fallon	8011-15	9730

7590 02/10/2004
Frank Chau, Esq.
F. CHAU & ASSOCIATES, LLP
Suite 501
1900 Hempstead Turnpike
East Meadow, NY 11554

EXAMINER

SURYAWANSHI, SURESH

ART UNIT PAPER NUMBER

2115

DATE MAILED: 02/10/2004

5

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary

Application No. 09/776,267	Applicant(s) FALLON ET AL.	
Examiner Suresh K Suryawanshi	Art Unit 2115	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --
Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) Responsive to communication(s) filed on 02 February 2001.
- 2a) This action is FINAL.
- 2b) This action is non-final.
- 3) Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) Claim(s) 1-16 is/are pending in the application.
 - 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) Claim(s) _____ is/are allowed.
- 6) Claim(s) 1-16 is/are rejected.
- 7) Claim(s) _____ is/are objected to.
- 8) Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) The specification is objected to by the Examiner.
- 10) The drawing(s) filed on 02 February 2001 is/are: a) accepted or b) objected to by the Examiner.
 - Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
 - Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

- 12) Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
 - a) All b) Some * c) None of:
 - 1. Certified copies of the priority documents have been received.
 - 2. Certified copies of the priority documents have been received in Application No. _____.
 - 3. Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).
- * See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

- 1) Notice of References Cited (PTO-892)
- 2) Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08)
 - Paper No(s)/Mail Date 4.
- 4) Interview Summary (PTO-413)
 - Paper No(s)/Mail Date. _____.
- 5) Notice of Informal Patent Application (PTO-152)
- 6) Other: _____.

DETAILED ACTION

1. Claims 1-16 are presented for examination.

Drawings

2. This application, filed under former 37 CFR 1.60, lacks formal drawings. The informal drawings filed in this application are acceptable for examination purposes. When the application is allowed, applicant will be required to submit new formal drawings. In unusual circumstances, the formal drawings from the abandoned parent application may be transferred by the grant of a petition under 37 CFR 1.182.

Specification

3. The abstract of the disclosure is objected to because it contains more than 150 words. Correction is required. See MPEP § 608.01(b).
4. The disclosure is objected to because of the following informalities: U.S. Patent Application Serial No. 09/266,394 is now Patent 6,601,103 at page 10, line 4.
Appropriate correction is required.
5. The disclosure is objected to because of the following informalities: U.S. Patent Application Serial No. 09/481,243 is now Patent 6,604,158 at page 10, line 6.
Appropriate correction is required.

6. The disclosure is objected to because of the following informalities: "greater that" should be "greater than" at page 10, line 10.

Appropriate correction is required.

7. The disclosure is objected to because of the following informalities: U.S. Patent Application Serial No. 09/210,491 is now Patent 6,195,024; page 11, line 7; page 50, line 3, 12, and 15.

Appropriate correction is required.

8. The disclosure is objected to because of the following informalities: symbol "5" is not in any figure; page 15, line 1.

Appropriate correction is required.

9. The disclosure is objected to because of the following informalities: blank space should be filled with Serial No. "09/775,897 at page 18, line 14.

Appropriate correction is required.

10. The disclosure is objected to because of the following informalities: symbol "30" at page 22, line 9 should be "35".

Appropriate correction is required.

Art Unit: 2115

11. The disclosure is objected to because of the following informalities: symbol "DSRA" is used without any definition in spec; page 37, line 18; page 38, line 7.

Appropriate correction is required.

12. The disclosure is objected to because of the following informalities: word "data" should be inserted after "the" and before "was" at page 41, line 16.

Appropriate correction is required.

Claim Rejections - 35 USC § 102

13. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

14. Claims 1 are rejected under 35 U.S.C. 102(e) as being anticipated by Krockner et al (US Patent no 6,073,232).

15. As per claim 1, Krockner et al teach

maintaining a list of boot data used for booting a computer system [col. 2, lines 30-47; col. 5, lines 1-7; a prefetch table containing a listing of the disk locations and length of data records that were requested by the host computer in the immediately previous power-on/reset];

preloading the boot data upon initialization of the computer system [col. 2, lines 36-41; col. 3, lines 30-39; col. 5, lines 17-21; data is preloaded into the cache according to the prefetch table]; and

servicing requests for boot data from the computer system using the preloaded boot data [col. 2, lines 41-47; col. 3, lines 30-39; data is communicated from the cache to the host computer].

16. As per claim 2, Krockner et al teach that the boot data comprises program code associated with one of an operating system of the computer system, an application program, and a combination thereof [col. 5, lines 41-51; requesting data records are part of a computer program such as DOS or Windows].

17. As per claim 3, Krockner et al teach that the step of preloading the boot data comprises retrieving boot data from a boot device and storing the retrieved data in a cache memory [col. 3, lines 30-39].

Art Unit: 2115

18. As per claim 4, Krocker et al teach that the method steps are performed by a data storage controller connected to the boot device [fig. 1; controller].

19. As per claim 5, Krocker et al teach the step of updating the list of boot data during the boot process [col. 8, lines 63-65; the prefetch table is updated].

20. As per claim 6, Krocker et al teach the step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list [col. 8, lines 63-68; the prefetch table is updated].

21. As per claim 7, Krocker et al teach that the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system [col. 8; lines 63-65; updating the prefetch table].

22. As per claims 9 and 12, Krocker et al teach that the method steps are program instructions that are tangibly embodied on a program storage device and readable by a machine to execute the method steps [col. 9, lines 27-30; computer program].

Art Unit: 2115

23. As per claim 10, Krocke et al teach

maintaining a list of application data associated with an application program [col. 11; lines 30-34; a prefetch table containing disk storage location and length of the data records requested by the application program];

preloading the application data upon launching the application program [col. 11, lines 46-50; preloading the data cache prior to receiving a read command from the application]; and

servicing requests for application data from a computer system using the preloaded application data [col. 11, lines 51-57; communicating the prestored data records of the application from the data cache to the host computer].

24. As per claim 13, Krocke et al teach

a digital signal processor (DSP) [fig. 1; host computer];

a programmable logic device [fig. 1, disk], wherein the programmable logic device is programmed by the digital signal processor [fig. 1; host computer] to (i) instantiate a first interface for operatively interfacing the boot device controller to a boot device [fig. 1; controller] and to (ii) instantiate a second interface for operatively interfacing the boot device controller to

Art Unit: 2115

the host system [inherent to the system as a bus interface is used to interface the controller with host computer]; and

a non-volatile memory device [fig. 1; disk;], for storing logic code associated with the DSP, the first interface and the second interface, wherein the logic code comprises instructions executable by the DSP for maintaining a list of boot data used for booting the host system [col. 5, lines 1-7; a prefetch table is read from a reserved area of the disks], preloading the boot data upon initialization of the host system [col. 2, lines 36-41; col. 3, lines 30-39; col. 5, lines 17-21; data is preloaded into the cache according to the prefetch table], and servicing requests for boot data from the host system using the preloaded boot data [col. 2, lines 41-47; col. 3, lines 30-39].

25. As per claim 14, Krockner et al teach a cache memory device for storing the preloaded boot data [fig. 1; cache].

26. As per claim 15, Krockner et al teach that the logic code in the non-volatile memory device further comprises program instructions executable by the DSP for maintaining a list of application data associated with an application program [col. 11; lines 30-34; a prefetch table containing disk storage location and length of the data records requested by the application program]; preloading the application data upon launching the application program [col. 11, lines 46-50; preloading the data cache prior to receiving a read command from the application], and

servicing requests for the application data from the host system using the preloaded application data col. 11, lines 51-57; communicating the prestored data records of the application from the data cache to the host computer].

Claim Rejections - 35 USC § 103

27. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

28. Claims 8, 11 and 16 are rejected under 35 U.S.C. 103(a) as being unpatentable over Krockner et al (US Patent no 6,073,232) in view of Dye (US Patent no 6,173,381 B1).

29. As per claims 8, 11 and 16, Krockner et al disclose the invention substantially. Krockner et al do not disclose about the data is being compressed and decompressed. But a routineer in the art would know about the data compression as the data compression is well known for purpose of space saving and increasing system bandwidth and efficiency. However Dye expressly disclose this technique [col. 8, lines 6-14; col. 11, lines 62-66]. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the cited references as both are directed to improve a system with respect to bandwidth and efficiency.

Conclusion

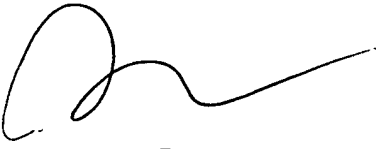
Any inquiry concerning this communication or earlier communications from the examiner should be directed to Suresh K Suryawanshi whose telephone number is 703-305-3990. The examiner can normally be reached on 9:00am - 5:30pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Thomas C. Lee can be reached on 703-305-9717. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

sks

February 4, 2004


THOMAS LEE
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100

Notice of References Cited

Application/Control No. 09/776,267	Applicant(s)/Patent Under Reexamination FALLON ET AL.	
Examiner Suresh K Suryawanshi	Art Unit 2115	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-6,073,232	06-2000	Kroeker et al.	713/1
	B	US-6,173,381	01-2001	Dye, Thomas A.	711/170
	C	US-6,539,456	03-2003	Stewart, David C.	711/113
	D	US-6,226,740	05-2001	Iga, Kazuhisa	713/2
	E	US-6,226,667	05-2001	Matthews et al.	709/203
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N	JP 06051989 A	02-1994	Japan	FURUSAWA, SHIGERU	G06F 09/445
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	IBM, Fast Dos Soft Boot, 2/1/1994, Vol 37, Issue 2B, pages 185-186
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

PAT-NO: - JP406051989A
DOCUMENT-IDENTIFIER: JP 06051989 A
TITLE: FAST LOADING SYSTEM OF OPERATING
SYSTEM IN COMPUTER SYSTEM
PUBN-DATE: February 25, 1994

INVENTOR-INFORMATION:
NAME
FURUSAWA, SHIGERU

ASSIGNEE-INFORMATION:
NAME COUNTRY
NEC CORP N/A

APPL-NO: JP04218728
APPL-DATE: July 27, 1992

INT-CL (IPC): G06F009/445

ABSTRACT:

PURPOSE: To perform the fast load processing of an operating system without performing the read processing of volume information for the whole device in spite of the presence/absence of a restore command in the initial load processing of the operating system.

CONSTITUTION: This system is provided with the device name instruction means 2 of a system storage file volume which instructs a volume in which a system storage file is stored in a device name by a REST command to instruct the generation of a system residence volume, the acquiring

Best Available Copy

means 3 of the
correspondence table of the device name with a channel
number, a channel number
acquiring means 4 to acquire the channel number from the
device name of the
volume, a restore processing means 5 which performs the
generation processing
of the system residence volume based on the device name and
the channel number,
and a device name/channel number correspondence table 6.

COPYRIGHT: (C)1994, JPO&Japio

(19)日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11)特許出願公開番号

特開平6-51989

(43)公開日 平成6年(1994)2月25日

(51)Int.Cl. ⁵ G 0 6 F 9/445	識別記号 9367-5B	庁内整理番号 9367-5B	F I G 0 6 F 9/ 06	技術表示箇所 4 2 0 G
---	-----------------	-------------------	----------------------	-------------------

審査請求 未請求 請求項の数1(全 4 頁)

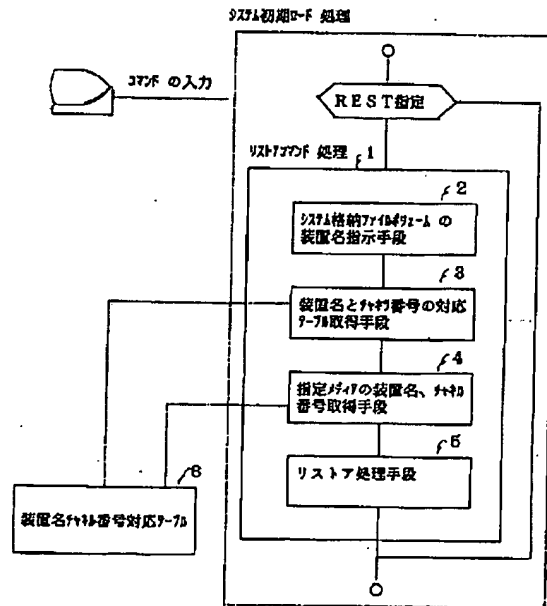
(21)出願番号	特願平4-218728	(71)出願人	000004237 日本電気株式会社 東京都港区芝五丁目7番1号
(22)出願日	平成4年(1992)7月27日	(72)発明者	古澤 茂 東京都港区芝五丁目7番1号 日本電気株式会社内
		(74)代理人	弁理士 山下 稔平

(54)【発明の名称】 計算機システムにおけるオペレーティングシステムの高速度ロード方式

(57)【要約】

【目的】 オペレーティングシステムの初期ロード処理において、リストアコマンドの有無にかかわらず全装置に対するボリューム情報のリード処理を行う事なくオペレーティングシステムの高速度ロード処理を行なう。

【構成】 システム常駐ボリュームの作成を指示する R E S Tコマンドによって、システム格納ファイルを記憶しているボリュームを装置名で指示するシステム格納ファイルボリュームの装置名指示手段2と、装置名とチャネル番号の対応テーブルの取得手段3と、ボリュームの装置名からチャネル番号を得る、チャネル番号取得手段4と、装置名、チャネル番号をもとに、システム常駐ボリュームの作成処理を行うリストア処理手段5と、装置名チャネル番号対応テーブル6を有する。



1

【特許請求の範囲】

【請求項1】 IPL時のオペレータ指示によって、システム格納ファイルよりシステム常駐ボリュームを作成する計算機システムにおいて、IPL時に、システム常駐ボリュームの作成を指示するRESTコマンドによってシステム格納ファイルを記憶しているボリュームを装置名で指示するシステム格納ファイルボリュームの装置名指示手段と、前記装置名とチャンネル番号の対応テーブルの取得手段と、前記ボリュームの装置名からチャンネル番号を得るチャンネル番号取得手段と、前記装置名、チャンネル番号をもとにシステム常駐ボリュームの作成処理を行うリストア処理手段とを有することを特徴とするオペレーティングシステムの高速度ロード方式。

【発明の詳細な説明】

【0001】

【産業上の利用分野】本発明は、計算機システムに関し、特にオペレーティングシステムのシステム初期ロード処理に関する。

【0002】

【従来の技術】従来、この種の計算機システムにおいて、オペレーティングシステムのシステム初期ロード時には、リストア処理の有無によらず、全ての装置に対するボリューム情報のリード処理を行い、ボリューム情報をテーブルにセットしていた。そして、RESTコマンドが指定された場合には、指定されたメディア名に対する装置名、チャンネル番号をこのテーブルから取得し、それらをもとにリストア処理を行っていた。

【0003】

【発明が解決しようとする課題】上述した従来の計算機システムにおいては、オペレーションシステムのシステム初期ロード時に、リストア処理の有無によらず、全ての装置のボリューム情報のリード処理を行っていた。また、再立上げ時には、再び全ての装置に対するリード処理を行わなければならないため、オペレーティングシステムのシステム初期ロード処理にかなりの時間がかかるという、欠点があった。

【0004】本発明は上の問題点に鑑みて、リストアコマンドの要求の有無に関わらず全装置に対するボリューム情報のリード処理を行うことなくオペレーションシステムの高速度ロード処理を可能とする計算機システムにおけるオペレーティングシステムの高速度ロード方式を提供することを目的としている。

【0005】

【課題を解決するための手段】本発明の計算機システムにおけるオペレーティングシステムの高速度ロード方式は、IPL時のオペレータ指示によって、システム格納ファイルよりシステム常駐ボリュームを作成する計算機

2

システムにおいて、IPL時に、システム常駐ボリュームの作成を指示するRESTコマンドによって、システム格納ファイルを記憶しているボリュームを装置名で指示するシステム格納ファイルボリュームの装置名指示手段と、装置名とチャンネル番号の対応テーブルの取得手段と、ボリュームの装置名からチャンネル番号を取得する、チャンネル番号取得手段と、装置名、チャンネル番号をもとに、システム常駐ボリュームの作成処理を行うリストア処理手段とを有する。

10 【0006】

【作用】上記構成によれば、IPL（初期プログラムローディング）時にシステム常駐ボリュームの作成を指示するRESTコマンドの投入によりリストアコマンド処理が動作し、システム格納ファイルボリュームの装置名指示手段がシステム格納ファイルのボリュームを装置名で指示する。装置名とチャンネル番号の対応テーブルの取得手段は指示された装置名とチャンネル番号の対応テーブルを参照して装置名からチャンネル番号を取得する。リストア処理手段はその装置名とチャンネル番号をもとに常駐ボリュームの作成処理を行うので、システムの初期ロード処理が高速化される。

【0007】

【実施例】次に、本発明の一実施例について図面を参照して説明する。

【0008】図1は本発明の実施例の構成図である。

【0009】図1を参照すると、リストアコマンド処理1と、システム格納ファイルボリュームの装置名指示手段2と、装置名とチャンネル番号の対応テーブル取得手段3と、チャンネル番号取得手段4と、リストア処理手段5とを有する。

30

【0010】つぎに動作について説明する。IPL時のオペレータ指示によって、システム格納ファイルよりシステム常駐ボリュームを作成する計算機システムにおいて、IPL時に、システム常駐ボリュームの作成を指示するRESTコマンドの投入によって、リストアコマンド処理1が動作する。そして、システム格納ファイルを記憶するボリュームを指定する。システム格納ボリュームの装置名指示手段2によりシステム格納ファイルの装置名を指示する。これにより、指定されたシステム格納ファイルの装置名が、装置名とチャンネル番号の対応テーブル取得手段3に渡る。

40

【0011】図2は図1に示す装置名、チャンネル番号対応テーブルの一例を示す図であり、図2のように、オペレーティングシステムを構成する各装置はそれぞれ固有な装置名61と装置名に対応する物理的な接続状態を示す為のチャンネル番号62を有しており、一般的に、オペレーティングシステムでは装置名とチャンネル番号の対応テーブル6をハードウェア又はソフトウェア、或は双方が保有している。装置名とチャンネル番号の対応テーブル

50

3

取得手段3は、上記装置名とチャンネル番号の対応テーブル6を取得した後チャンネル番号取得手段4を起動する。チャンネル番号取得手段4は、装置名とチャンネル番号の対応テーブル6を参照し、装置名61からチャンネル番号62を取得する。次に、システム格納ファイルボリュームの装置名指示手段2により指示された装置名61、および、チャンネル番号取得手段4により取得したチャンネル番号62を入力としてリストア処理手段5を起動する。リストア処理手段5は、装置名61、チャンネル番号62をもとに、システム常駐ボリュームの作成処理を行う。

【0012】さらに、システムの初期ロード処理において、リストア処理の必要がなければ、リストアコマンド処理1は実行されず、全装置のボリューム情報のリード処理等の不必要な処理は行われない。

【0013】またリストアの指示が無い場合であっても、全装置のボリューム情報のリード処理を行わないためオペレーティングシステムの高ロードが可能となる。

【0014】このような、本実施例においてはリストアコマンド処理においては、メディア名のかわりに装置名

4

を指定することにより、オペレーションシステムのシステム初期ロード処理の高速化を行うことを可能とする。

【0015】

【発明の効果】以上説明したように、本発明は、オペレーションシステムのシステム初期ロード処理において、リストアコマンドの要求の有無に関わらず全装置に対するボリューム情報のリード処理を行うことなく、オペレーションシステムの高ロード処理を行うことを可能とする効果がある。

10 【図面の簡単な説明】

【図1】本発明の一実施例の構成を表す図である。

【図2】図1に示す装置名チャンネル番号対応テーブルの内容を示す図である。

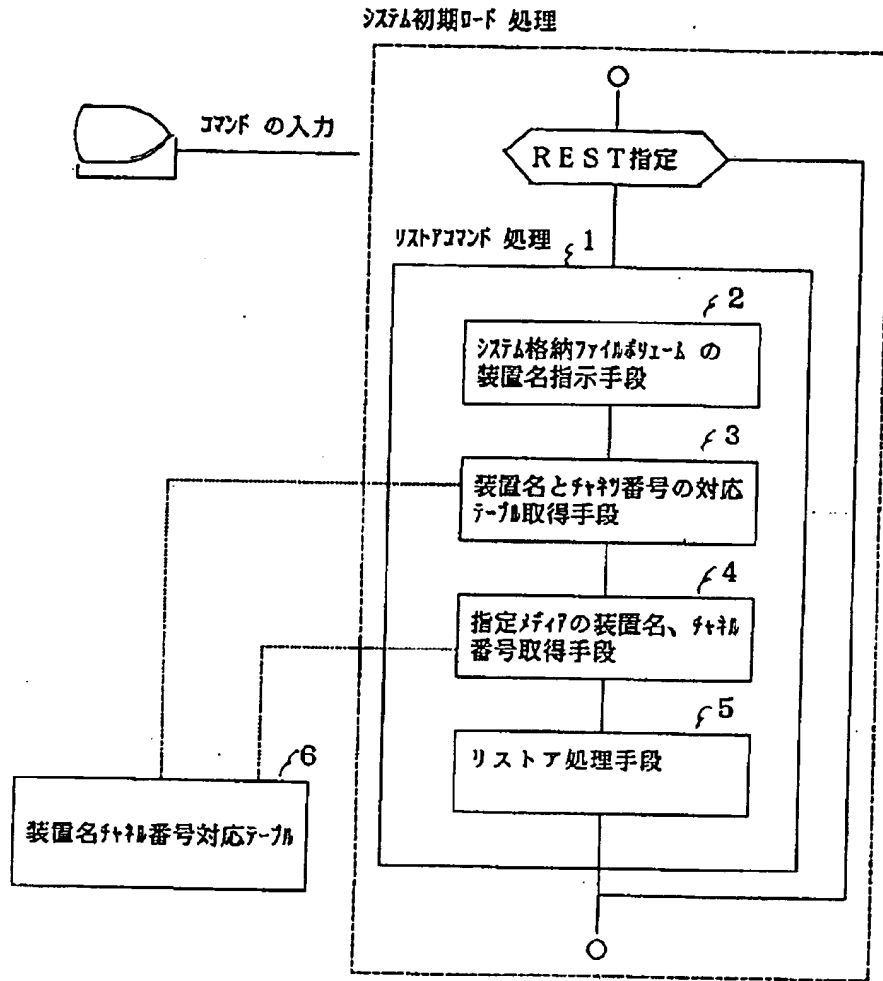
【符号の説明】

- 1 リストアコマンド処理
- 2 システム格納ファイルボリュームの装置名指示手段
- 3 装置名とチャンネル番号の対応テーブル取得手段
- 4 指定装置のチャンネル番号取得手段
- 5 リストア処理手段
- 20 6 装置名とチャンネル番号の対応テーブル

【図2】

61	62
装置名1	チャンネル番号1
装置名2	チャンネル番号2
~	
装置名n	チャンネル番号n

【図1】



**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- BLACK BORDERS**
- IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- FADED TEXT OR DRAWING**
- BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- SKEWED/SLANTED IMAGES**
- COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- GRAY SCALE DOCUMENTS**
- LINES OR MARKS ON ORIGINAL DOCUMENT**
- REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- OTHER:** _____

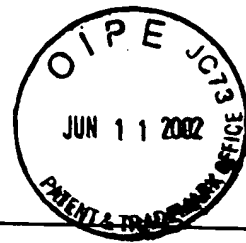
IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.

Form PTO-1449 U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
INFORMATION DISCLOSURE
STATEMENT BY APPLICANT

ATTY. DOCKET NO.
8011-15

SERIAL NO.
09/776.267



(Use several sheets if necessary)

APPLICANT

James J. Fallon et al.

FILING DATE

February 2, 2001

GROUP

Art Unit 2182

U.S. PATENT DOCUMENT

EXAMINER INITIALS	DOCUMENT NUMBER	DATE	NAME	CLASS	SUBCLASS	FILING DATE IF APPROPRIATE
Sls	5 1 8 7 7 9 3	2/16/1993	Keith et al.			
Sls	5 2 2 6 1 7 6	7/6/1993	Westaway et al.			
Sls	5 6 5 2 8 5 7	7/29/1997	Shimoi et al.			

RECEIVED

JUN 18 2002

Technology Center 2100

FOREIGN PATENT DOCUMENTS

	DOCUMENT NUMBER	DATE	COUNTRY	CLASS	SUBCLASS	TRANSLATION	
						YES	NO
Sls	DE 41 27 5 1 8 A 1	2/27/1992	Germany				
Sls	EP 07 18 7 5 1 A 2	6/26/1996	Europe				X
Sls	EP 07 18 7 5 1 A 3	6/26/1992	Europe				

OTHER DOCUMENTS (Including Author, Title, Date, Pertinent Pages, Etc.)

EXAMINER

Suresh K Suryawanshi

DATE CONSIDERED

2/4/04

EXAMINER: Initial if citation considered, whether or not citation is in conformance with MPEP 609; Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

(Form PTO-1449 [6-4])

	Hits	Search Text	DBs	Time Stamp
1	0	accelerat\$3 near3 (OS (operating adj1 system)) with load\$3	USPAT	2004/01/28 14:50
2	1	accelerat\$3 near3 (OS (operating adj1 system)) with boot\$3	USPAT	2004/01/28 14:50
3	77	accelerat\$3 near3 (OS (operating adj1 system))	USPAT	2004/01/28 15:02
4	7	(fast\$2 near2 load\$3) with (OS (operating adj1 system))	USPAT	2004/01/28 15:12
5	1	(maintain\$3 near2 list) near5 boot\$3	USPAT	2004/01/28 15:18
6	3	(maintain\$3 near2 list) near5 boot\$3	US-PGPUB; EPO; JPO; IBM_TDB	2004/01/28 15:16
7	184	pre\$1load\$3 near5 initializ\$5	USPAT	2004/01/28 15:19
8	36	pre\$1load\$3 near5 initializ\$5 with data	USPAT	2004/01/28 15:29
9	8	pre\$1load\$3 near5 initializ\$5 with computer	USPAT	2004/01/28 15:35
10	0	(updat\$3 near2 (list near2 boot)) near5 (boot adj1 process)	USPAT	2004/01/28 15:36
11	2	(updat\$3 near2 (list near2 boot))	USPAT	2004/01/28 15:43
12	3	(updat\$3 near2 (list near2 boot))	US-PGPUB; EPO; JPO; IBM_TDB	2004/01/28 15:39
13	32	(accelerat\$3 fast\$2) near2 (launch\$3 load\$3) near3 application	USPAT	2004/01/28 15:51
14	0	pre\$1load\$3 near2 (boot adj1 data)	USPAT	2004/01/28 15:52
15	22	pre\$1load\$3 near2 data with (boot\$3 initializ\$5)	USPAT	2004/01/28 16:08
16	14	pre\$1load\$3 near2 data with (boot\$3 initializ\$5)	US-PGPUB; EPO; JPO; IBM_TDB	2004/01/28 15:58
17	4	pre\$1load\$3 with (boot adj1 time)	USPAT	2004/01/28 16:13
18	4	pre\$1load\$3 with (boot adj1 time)	US-PGPUB; EPO; JPO; IBM_TDB	2004/01/28 16:12
19	3	decreas\$3 near3 (boot adj1 time)	USPAT	2004/01/28 16:32
20	6	decreas\$3 near3 (boot adj1 time)	US-PGPUB; EPO; JPO; IBM_TDB	2004/01/28 16:32
21	11	controller near2 pre\$1load\$3 same (boot reset)	USPAT	2004/01/28 16:35
22	29	portions near2 (os (operating adj1 system)) near3 load\$3	USPAT	2004/01/28 16:46

	Hits	Search Text	DBs	Time Stamp
23	0	portions near2 (os (operating adj1 system)) near3 load\$3	EPO; JPO; IBM_TDB	2004/01/28 16:45
24	38	preload\$3 with star\$1tup	USPAT	2004/01/28 16:55
25	2	preload\$3 with star\$1tup	EPO; JPO; IBM_TDB	2004/01/28 16:55
26	2	6226667.URPN.	USPAT	2004/01/28 17:07
27	75	instant near2 boot\$3	USPAT	2004/01/28 16:58
28	1	instant near2 boot\$3 with (OS (operating adj1 system))	USPAT	2004/01/28 17:01
29	0	instant near2 boot\$3 with (OS (operating adj1 system))	EPO; JPO; IBM_TDB	2004/01/28 17:00
30	0	instant near2 boot\$3 with pre\$1load\$3	USPAT	2004/01/28 17:01
31	2	quick near2 boot\$3 with (OS (operating adj1 system))	USPAT	2004/01/28 17:02
32	0	quick near2 launch\$3 with (OS (operating adj1 system))	USPAT	2004/01/28 17:02
33	0	quick near2 launch\$3 with (OS (operating adj1 system))	EPO; JPO; IBM_TDB	2004/01/28 17:02
34	0	quick near2 boot\$3 with (OS (operating adj1 system))	EPO; JPO; IBM_TDB	2004/01/28 17:02
35	1	6539456.pn.	USPAT	2004/01/29 09:50
36	0	6539456.URPN.	USPAT	2004/01/29 10:27
37	6	("5307497" "6061788" "6073232" "6172936" "6189100" "6226740").PN.	USPAT	2004/01/29 10:27
38	20	fast adj1 boot	USPAT	2004/02/02 17:32
39	1	fast adj1 boot	EPO; JPO; IBM_TDB	2004/02/02 17:31
40	5	fast near2 boot	EPO; JPO; IBM_TDB	2004/02/02 17:31
41	44	fast near2 boot	USPAT	2004/02/02 17:34
42	24	(fast near2 boot) not (fast adj1 boot)	USPAT	2004/02/02 17:32
43	2	fast near2 booting	USPAT	2004/02/02 17:36
44	18	fast\$2 near2 booting	USPAT	2004/02/02 17:39
45	0	fast\$2 near2 booting	EPO; JPO; IBM_TDB	2004/02/02 17:38
46	1	quick near2 booting	EPO; JPO; IBM_TDB	2004/02/02 17:38

	Hits	Search Text	DBs	Time Stamp
47	4	quick near2 booting	USPAT	2004/02/02 17:38
48	1	(fast\$2 near2 loading) with ((operating adj1 system) os)	USPAT	2004/02/02 17:42
49	1	(fast\$2 near2 loading) with ((operating adj1 system) os)	US-PGPUB; EPO; JPO; IBM_TDB	2004/02/02 17:42
50	0	(quick near2 loading) with ((operating adj1 system) os)	US-PGPUB; EPO; JPO; IBM_TDB	2004/02/02 17:42
51	0	(quick near2 loading) with ((operating adj1 system) os)	USPAT	2004/02/02 17:44
52	0	boot with (pre\$1loaded near2 data)	USPAT	2004/02/02 17:45
53	0	boot\$3 with (pre\$1loaded near2 data)	USPAT	2004/02/02 17:46
54	4	boot\$3 with (pre\$1loaded near2 data)	US-PGPUB; EPO; JPO; IBM_TDB	2004/02/02 17:45
55	10	(pre\$1load\$3 near2 data) near5 (initializ\$5 start\$lup restart)	USPAT	2004/02/02 17:49
56	7	(pre\$1load\$3 near2 data) near5 (initializ\$5 start\$lup restart)	US-PGPUB; EPO; JPO; IBM_TDB	2004/02/02 17:49
57	24	((boot application) adj1 data) near3 (compressed decompress\$3)	USPAT	2004/02/04 10:03



US006173381B1

(12) **United States Patent**
Dye

(10) **Patent No.:** US 6,173,381 B1
(45) **Date of Patent:** *Jan. 9, 2001

(54) **MEMORY CONTROLLER INCLUDING EMBEDDED DATA COMPRESSION AND DECOMPRESSION ENGINES**

4,929,946 * 5/1990 O'Brien et al. 341/87

(List continued on next page.)

(75) **Inventor:** Thomas A. Dye, Austin, TX (US)

* cited by examiner

(73) **Assignee:** Interactive Silicon, Inc., Austin, TX (US)

Primary Examiner—Eddie P. Chan

Assistant Examiner—Hong Kim

(74) *Attorney, Agent, or Firm*—Conley, Rose & Tayon, PC; Jeffrey C. Hood

(*) **Notice:** This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Under 35 U.S.C. 154(b), the term of this patent shall be extended for 0 days.

This patent is subject to a terminal disclaimer.

(57) **ABSTRACT**

An integrated memory controller (IMC) which includes data compression and decompression engines for improved performance. The memory controller (IMC) of the present invention preferably sits on the main CPU bus or a high speed system peripheral bus such as the PCI bus and couples to system memory. The IMC preferably uses a lossless data compression and decompression scheme. Data transfers to and from the integrated memory controller of the present invention can thus be in either two formats, these being compressed or normal (non-compressed). The IMC also preferably includes microcode for specific decompression of particular data formats such as digital video and digital audio. Compressed data from system I/O peripherals such as the hard drive, floppy drive, or local area network (LAN) are decompressed in the IMC and stored into system memory or saved in the system memory in compressed format. Thus, data can be saved in either a normal or compressed format, retrieved from the system memory for CPU usage in a normal or compressed format, or transmitted and stored on a medium in a normal or compressed format. Internal memory mapping allows for format definition spaces which define the format of the data and the data type to be read or written. Software overrides may be placed in applications software in systems that desire to control data decompression at the software application level. The integrated data compression and decompression capabilities of the IMC remove system bottle-necks and increase performance. This allows lower cost systems due to smaller data storage requirements and reduced bandwidth requirements. This also increases system bandwidth and hence increases system performance. Thus the IMC of the present invention is a significant advance over the operation of current memory controllers.

(21) **Appl. No.:** 08/916,464

(22) **Filed:** Aug. 8, 1997

Related U.S. Application Data

(60) Continuation of application No. 08/463,106, filed on Jun. 5, 1995, now abandoned, which is a division of application No. 08/340,667, filed on Nov. 16, 1994, now Pat. No. 6,002,411.

(51) **Int. Cl.⁷** G06F 13/00

(52) **U.S. Cl.** 711/170; 711/159; 711/160; 711/165; 711/155; 710/68; 709/247; 345/202; 345/521; 382/232

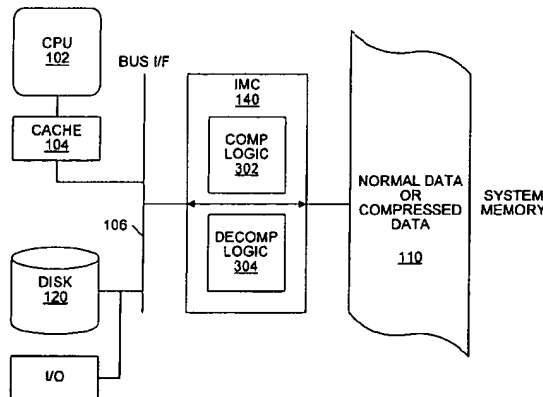
(58) **Field of Search** 395/341, 888, 395/133, 159; 711/203, 170, 160, 133, 134, 136, 155, 159, 165; 709/247; 345/521, 202, 509; 710/68; 714/763, 764; 382/232

(56) **References Cited**

U.S. PATENT DOCUMENTS

- 4,008,460 * 2/1977 Bryant et al. 395/463
- 4,688,108 * 8/1987 Cotton et al. 358/261.1
- 4,881,075 * 11/1989 Weng 341/87

97 Claims, 19 Drawing Sheets



U.S. PATENT DOCUMENTS

5,237,460	*	8/1993	Miller et al.	395/888	5,559,978	*	9/1996	Spilo	711/203
5,247,638	*	9/1993	O'Brien et al.	395/888	5,563,595		10/1996	Strohacker	341/106
5,247,646	*	9/1993	Osterlund et al.	395/888	5,584,008	*	12/1996	Shimada et al.	711/114
5,353,425	*	10/1994	Matamy et al.	711/144	5,602,976	*	2/1997	Cooper et al.	358/1.17
5,357,614	*	10/1994	Pattisam et al.	395/250	5,606,428	*	2/1997	Hanselman	358/404
5,396,343	*	3/1995	Hanselman	358/426	5,652,878	*	7/1997	Craft	707/1
5,420,696	*	5/1995	Wegeng et al.	358/468	5,696,912	*	12/1997	Bicevskis et al.	395/308
5,455,577	*	10/1995	Slivka et al.	341/51	5,696,926	*	12/1997	Culbert et al.	711/203
5,479,587	*	12/1995	Campbell et al.	358/1.17	5,699,539	*	12/1997	Garber et al.	711/2
5,483,622	*	1/1996	Zimmerman et al.	358/1.15	5,708,763	*	1/1998	Peltzer	395/115
5,504,842	*	4/1996	Gentile	358/1.15	5,812,817	*	9/1998	Hovis et al.	711/173
5,548,742	*	8/1996	Wang et al.	711/128	5,828,877		10/1998	Pearce et al.	395/670

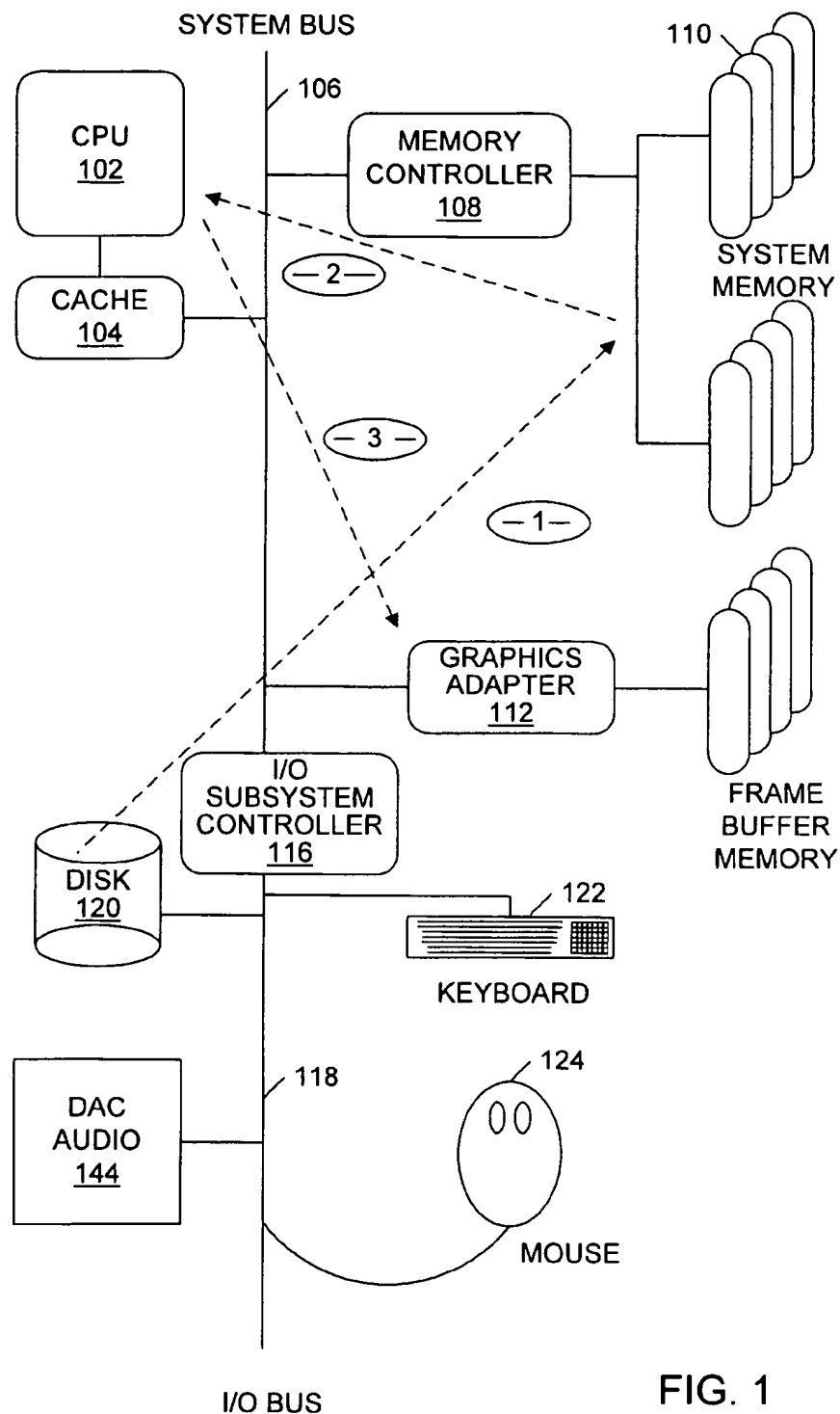


FIG. 1
(PRIOR ART)

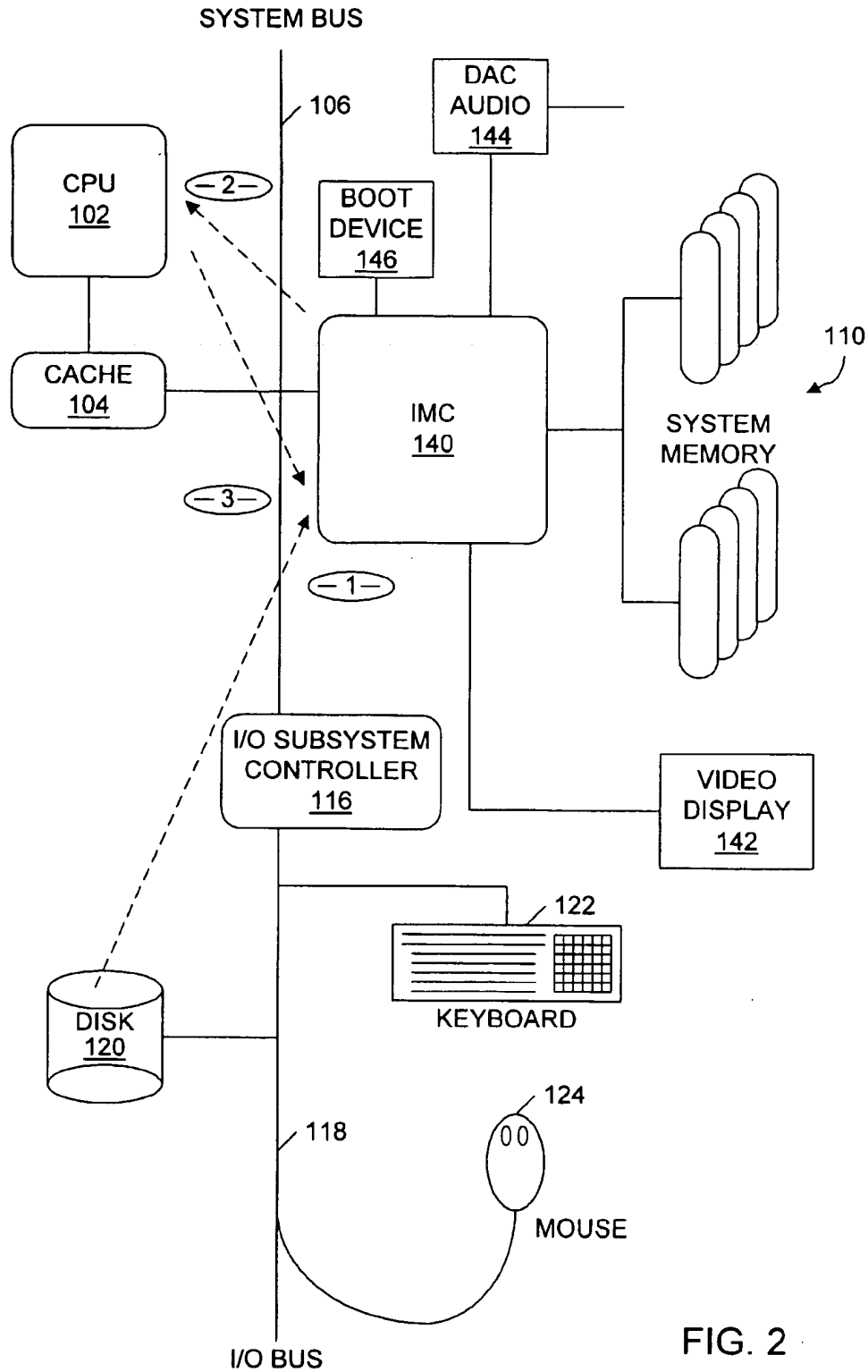


FIG. 2

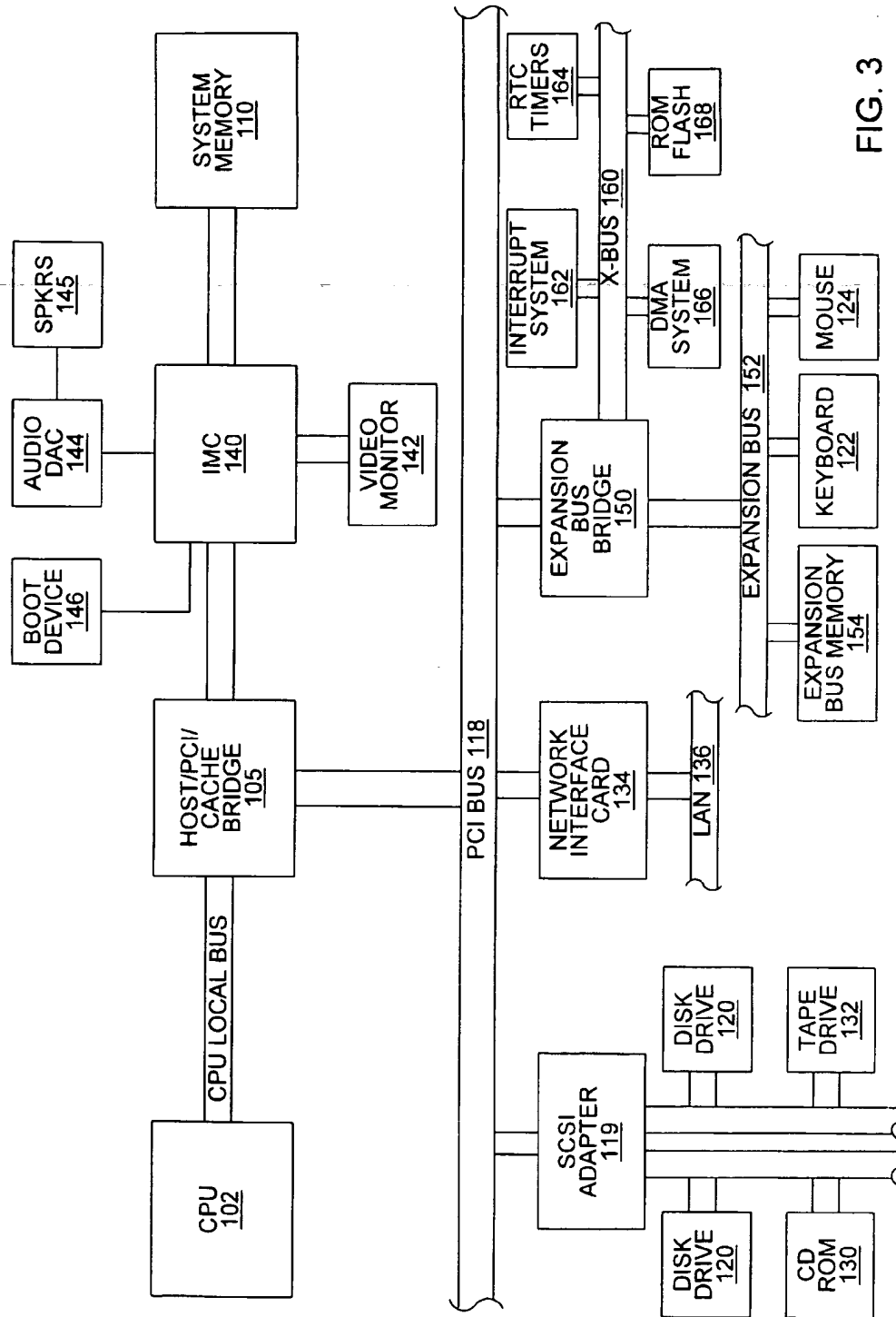


FIG. 3

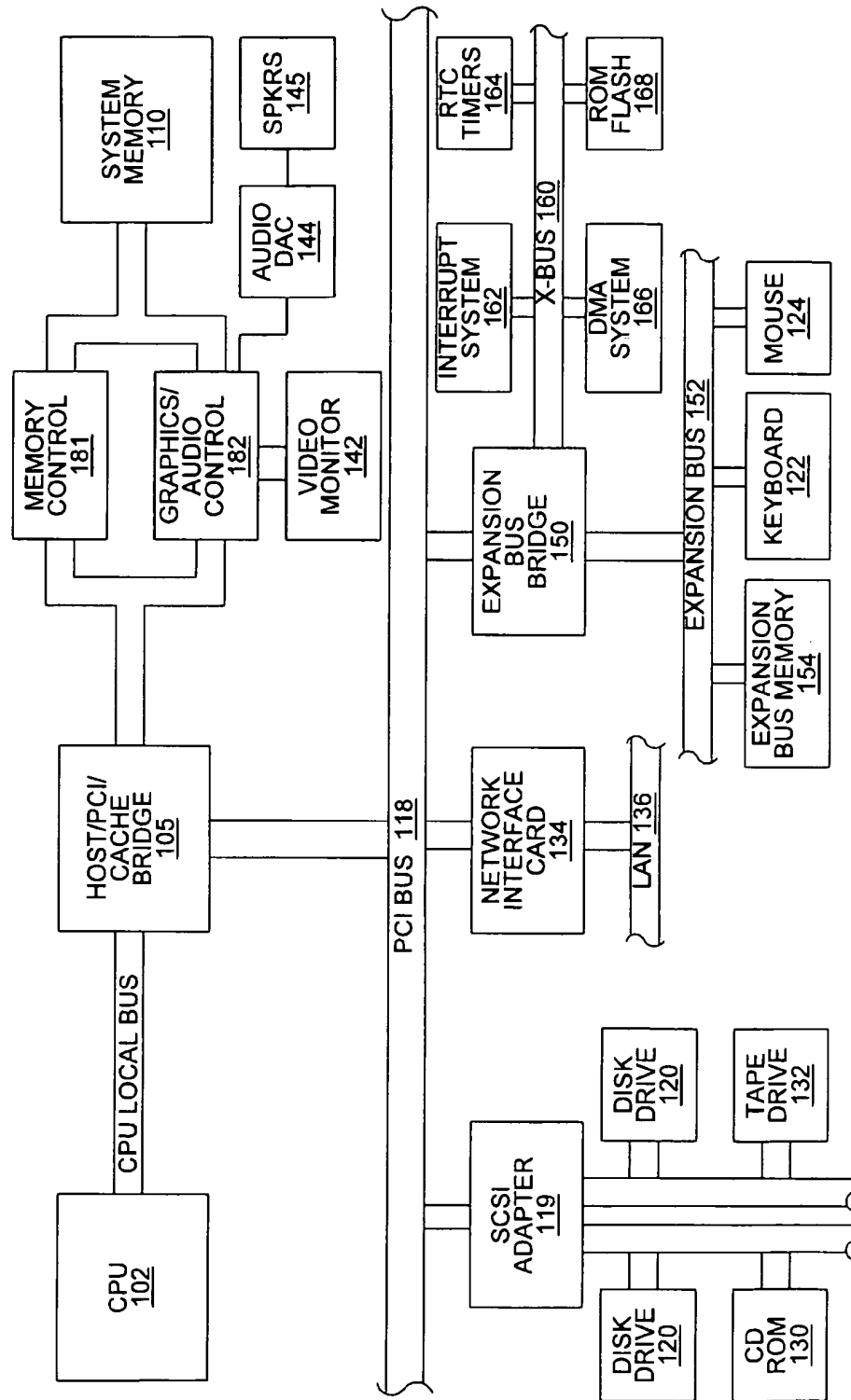


FIG. 3A

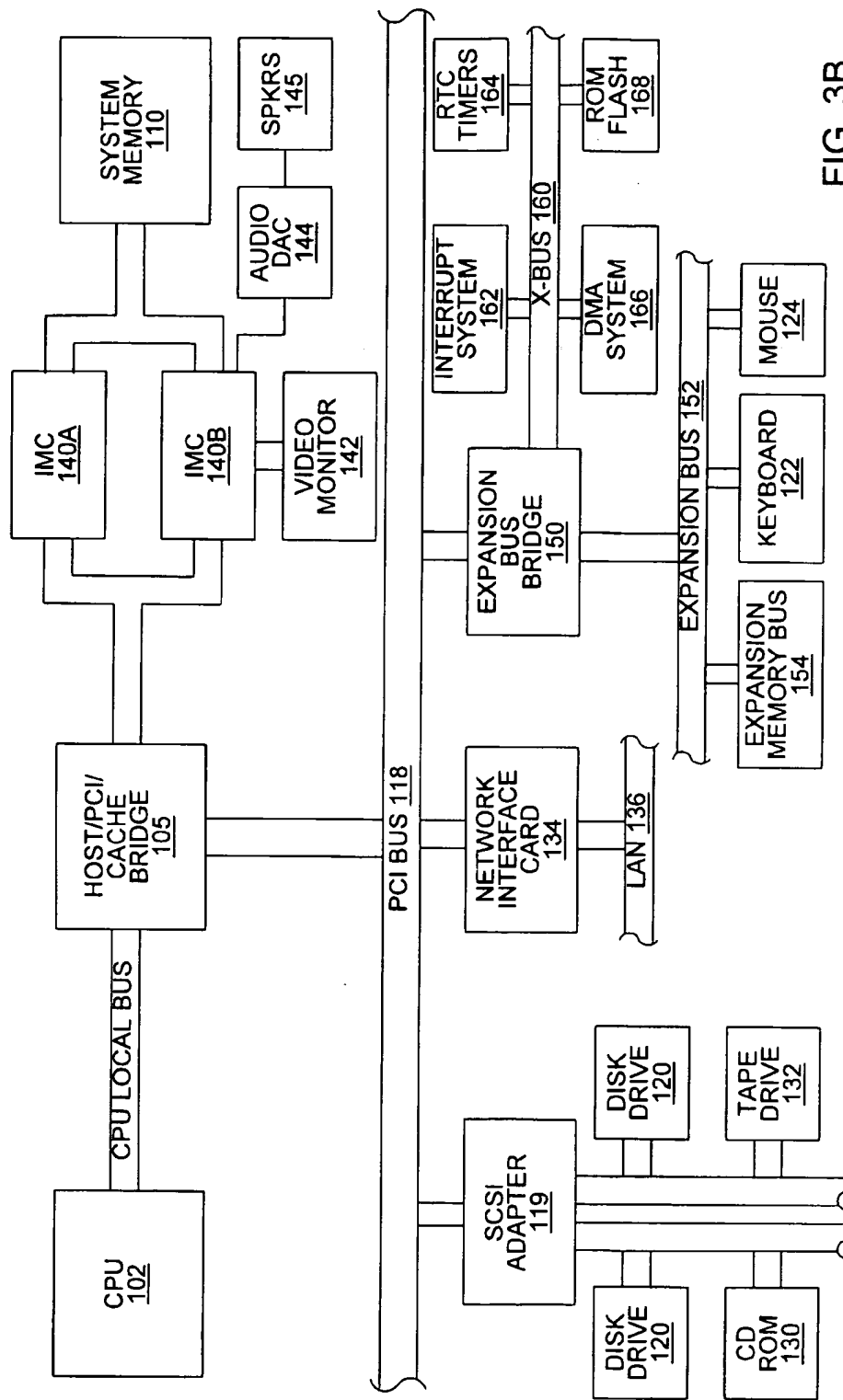


FIG. 3B

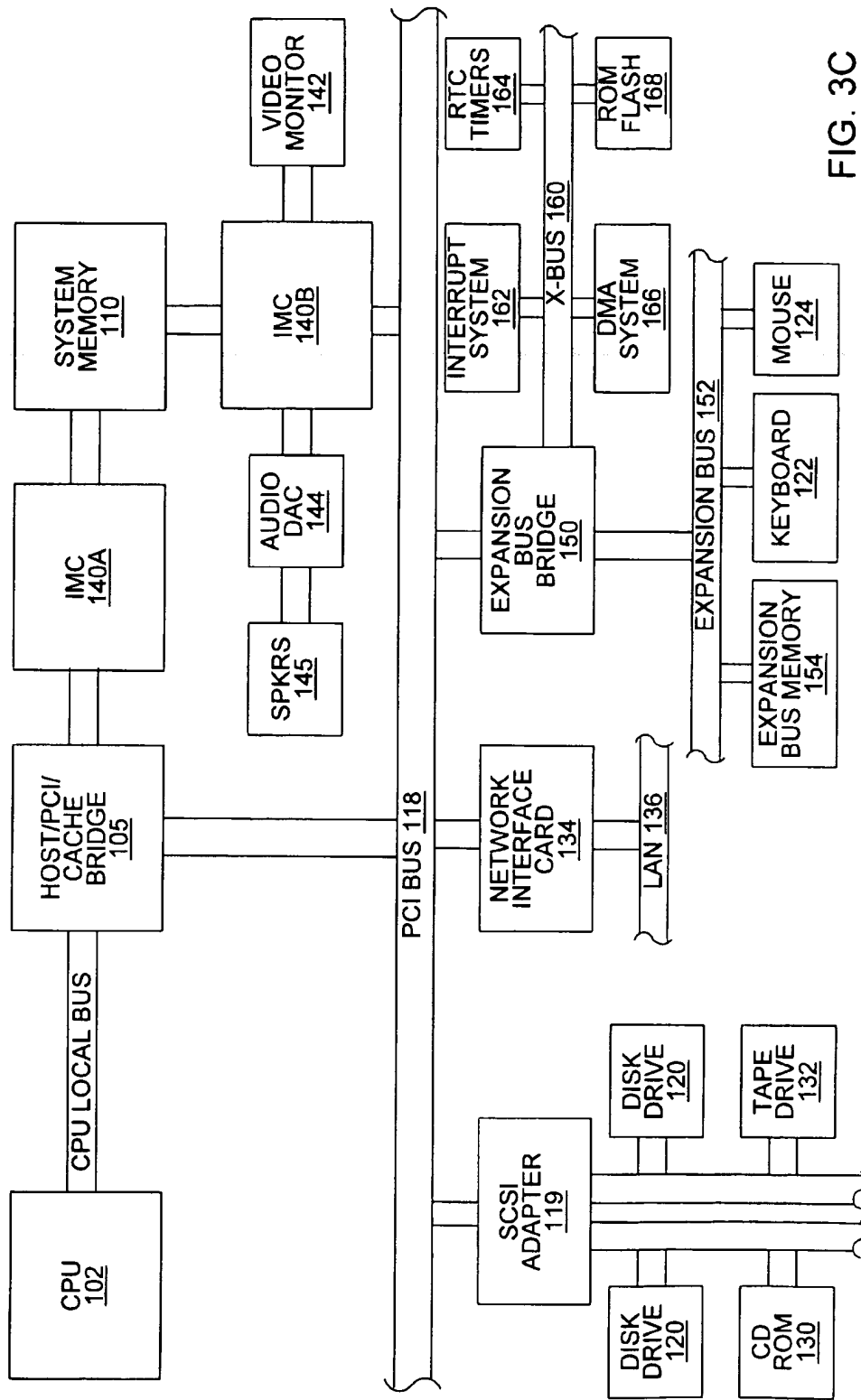


FIG. 3C

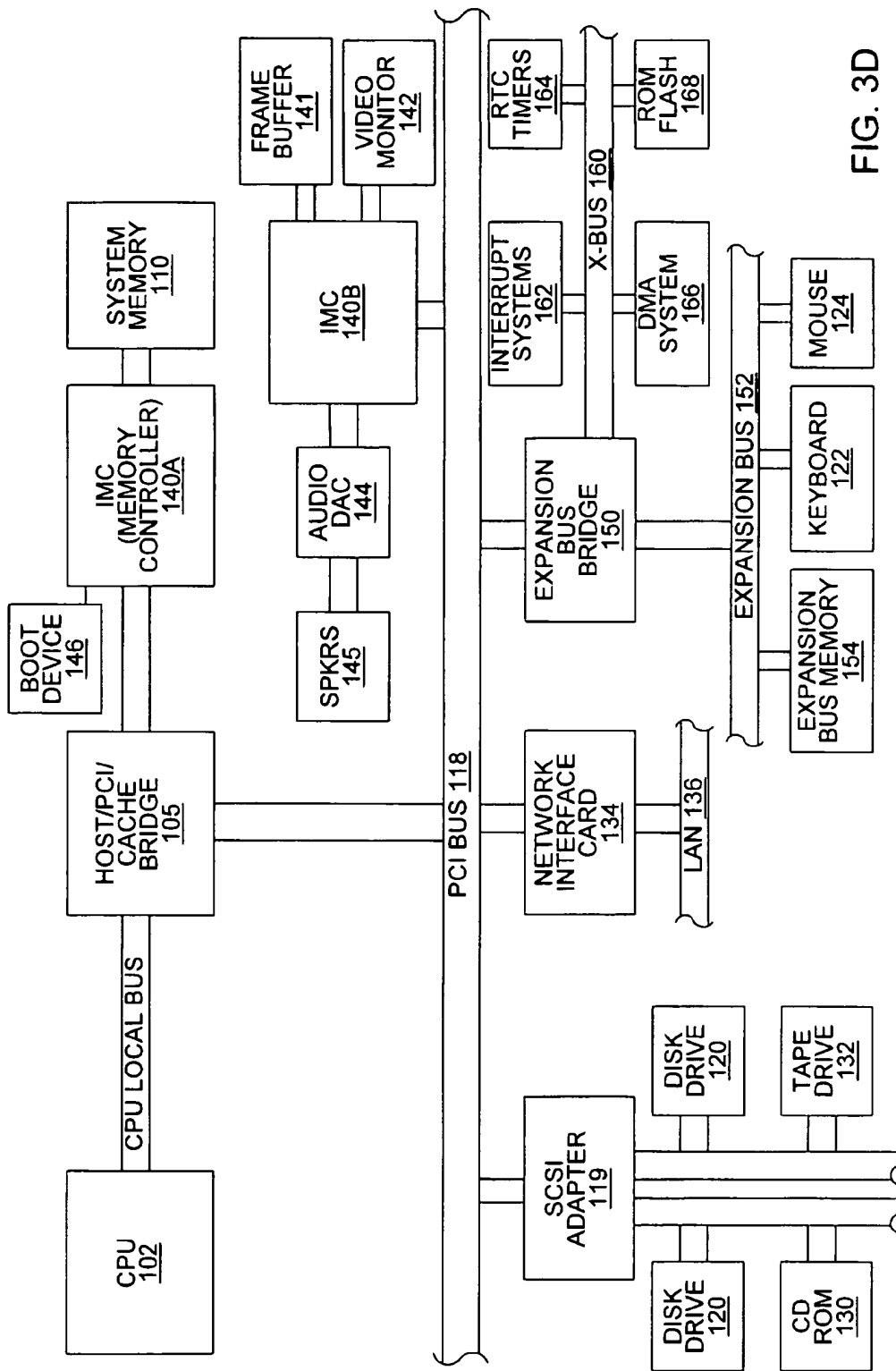


FIG. 3D

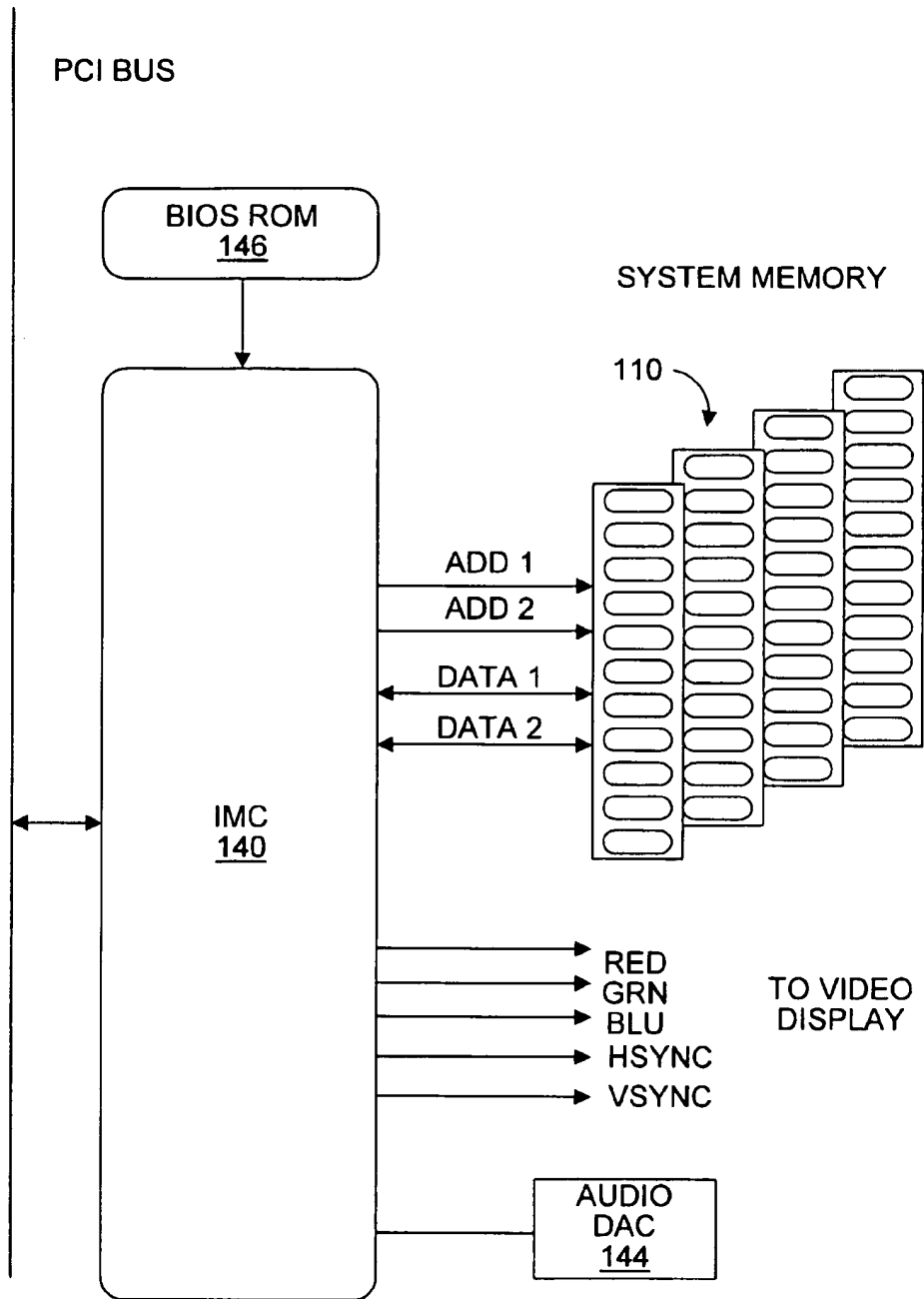


FIG. 4

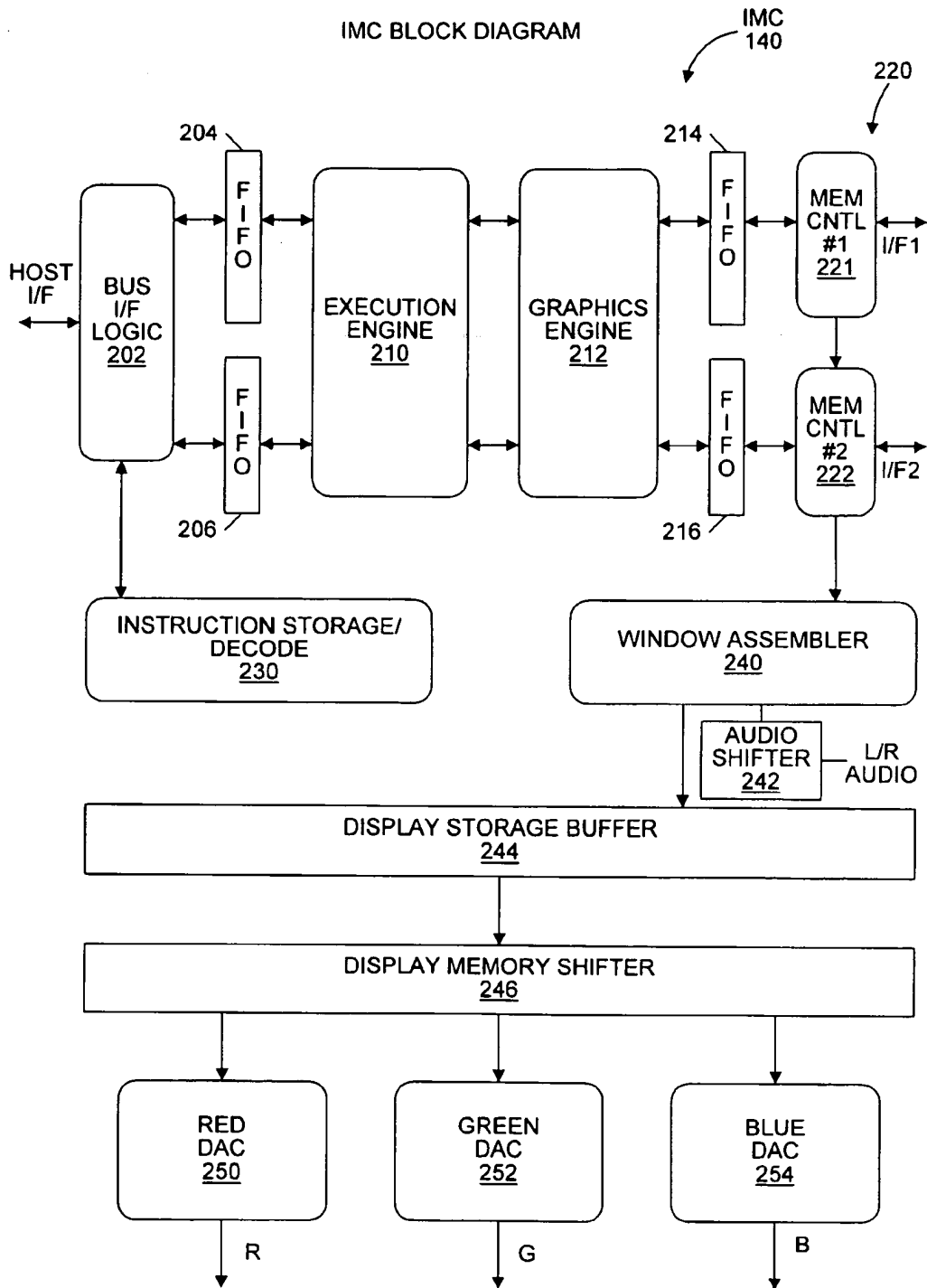


FIG. 5

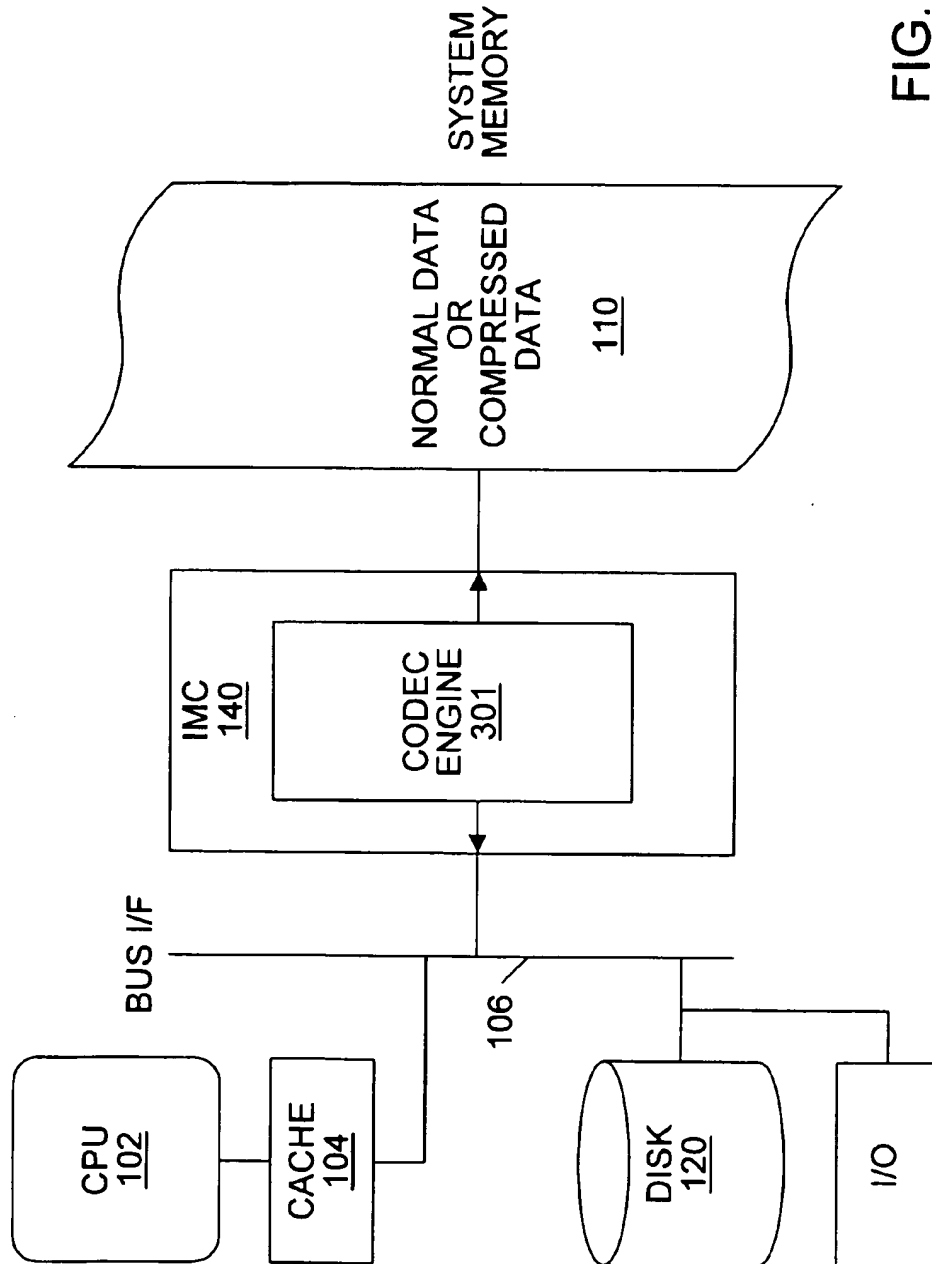


FIG. 6

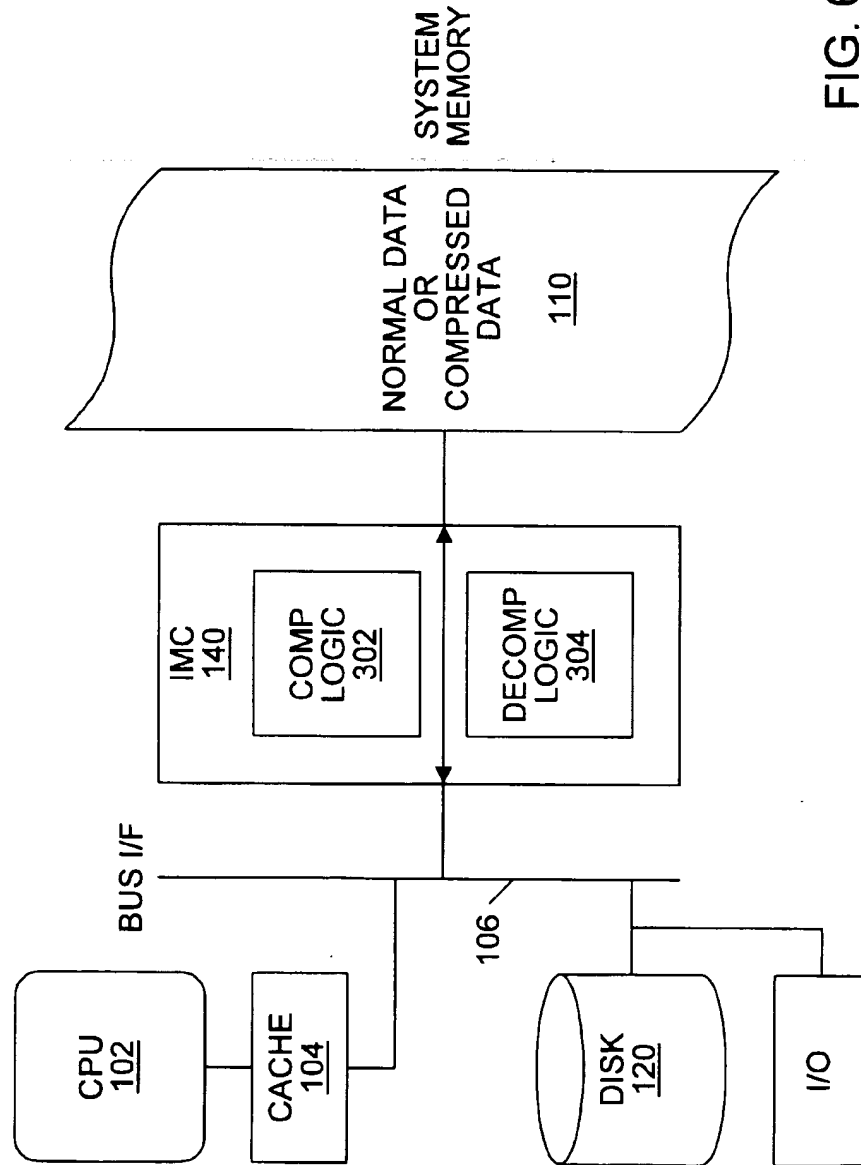
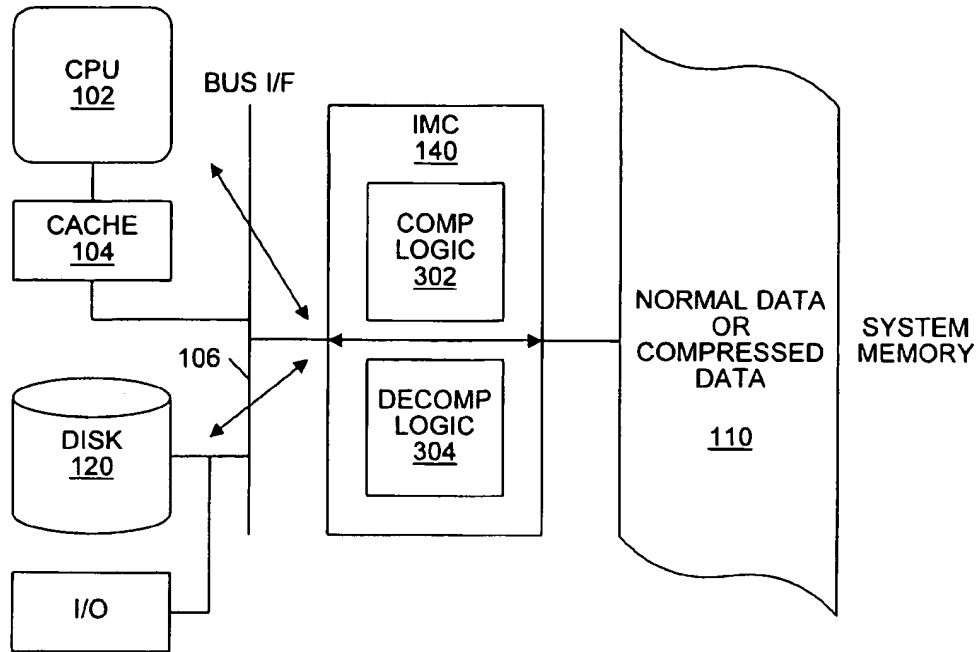
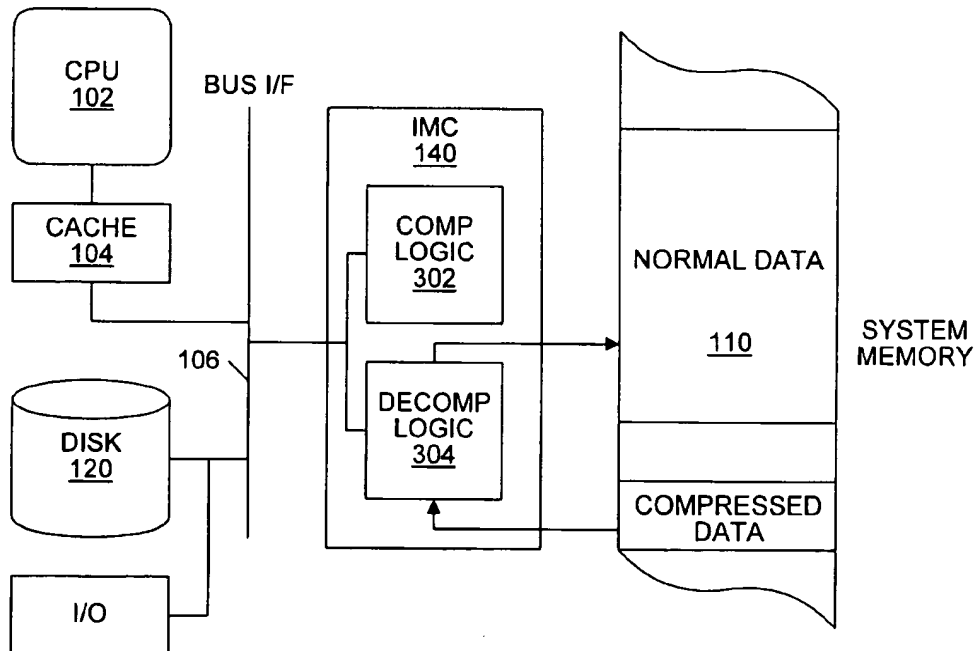


FIG. 6A



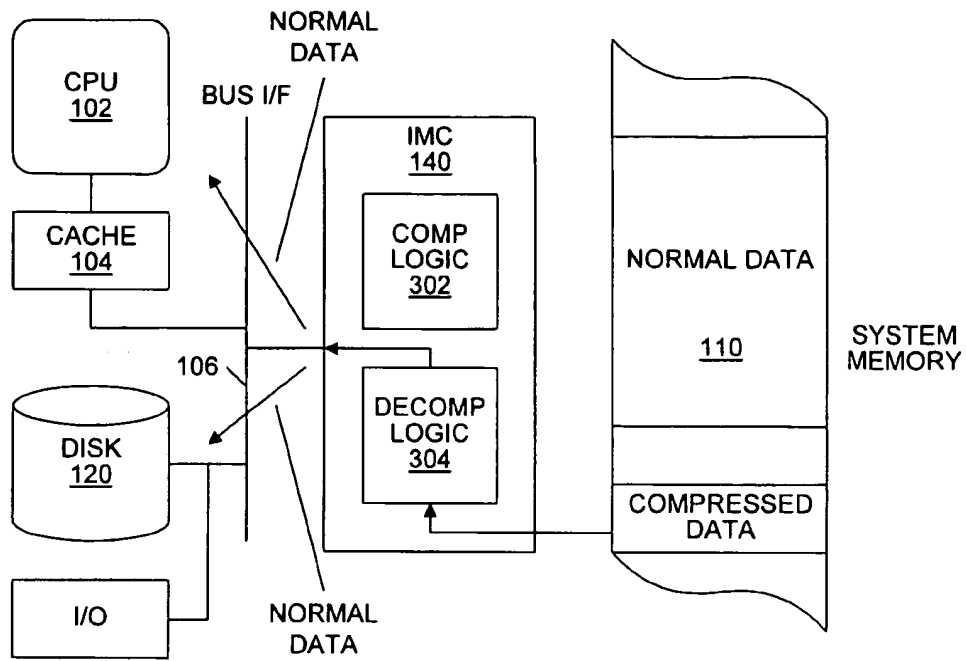
Normal or compressed data transfer, No modification by IMC

FIG. 7



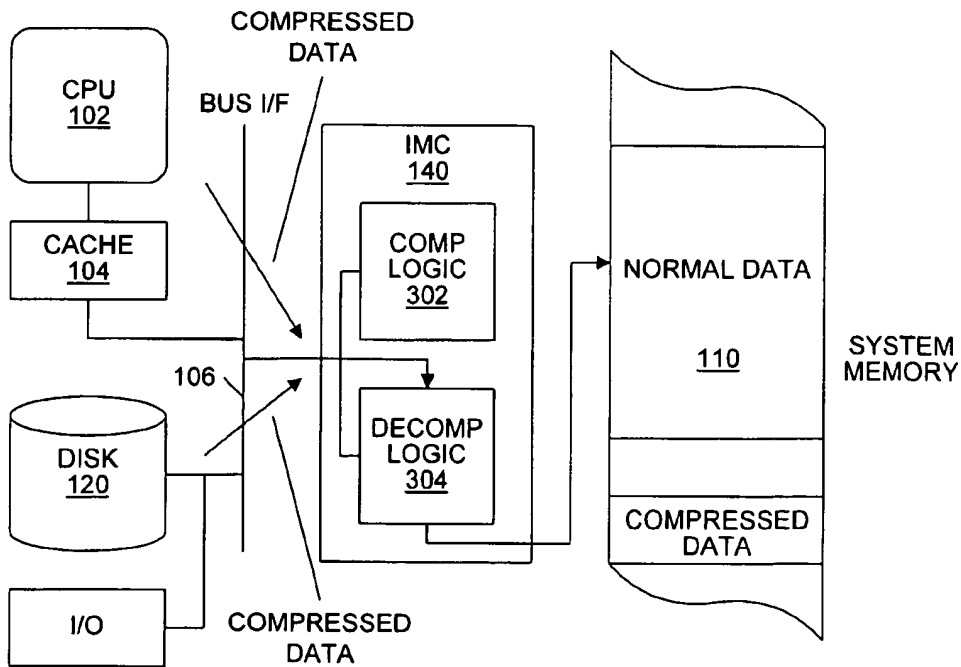
Memory to memory decompression

FIG. 8



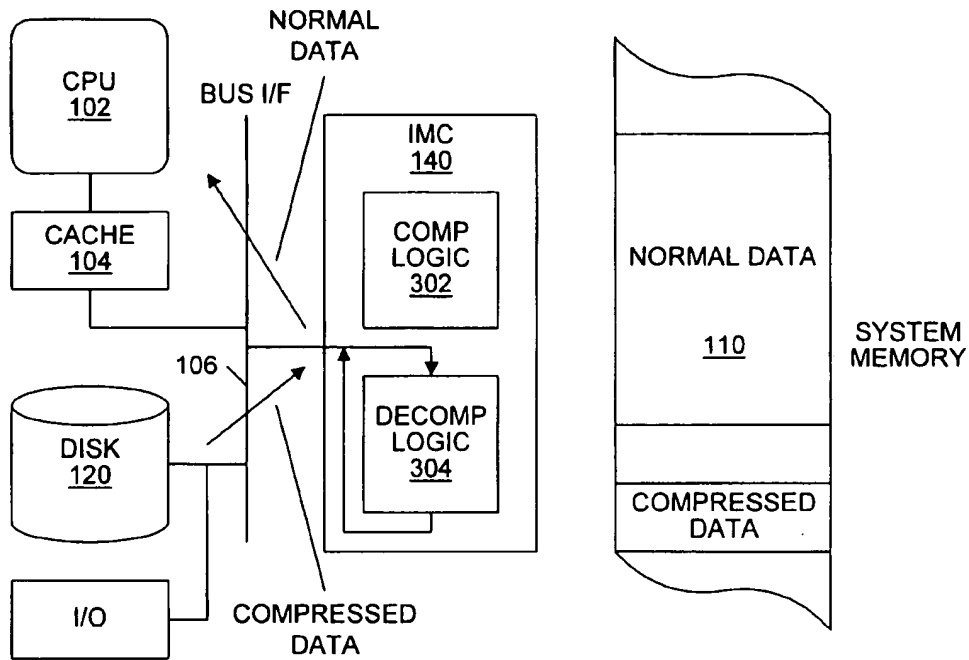
Memory decompression to CPU or Disk

FIG. 9



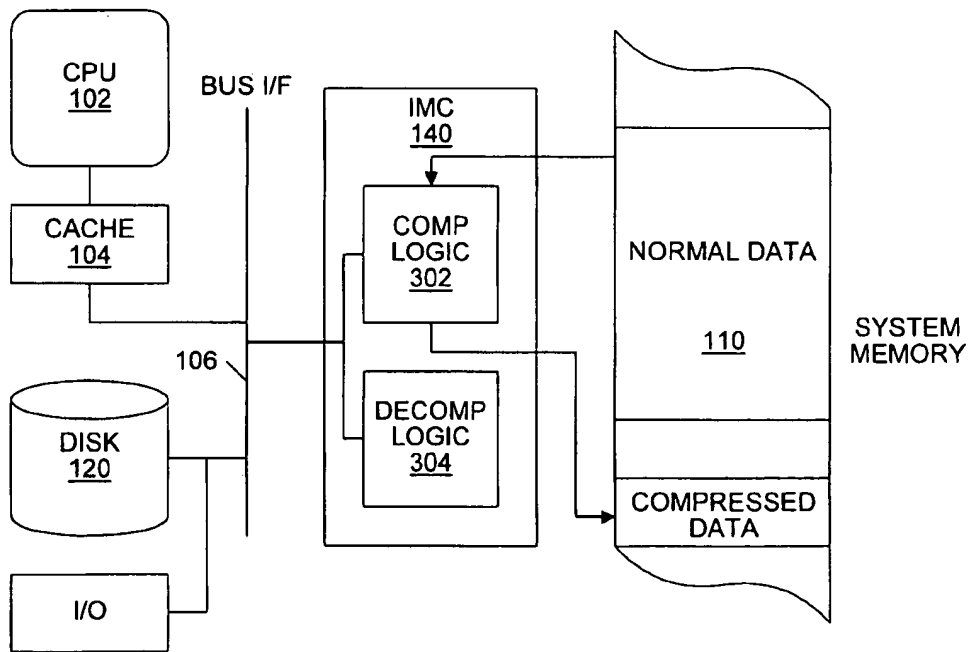
Decompression from Disk or CPU to memory

FIG. 10



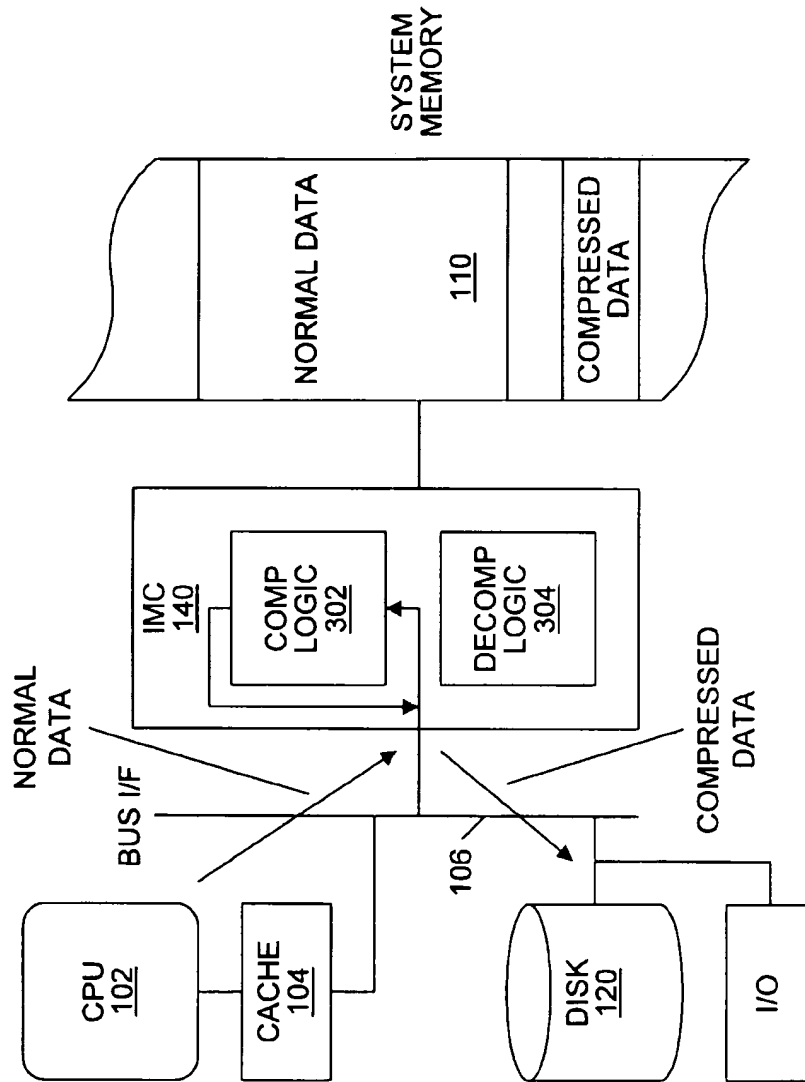
Decompression of disk to CPU

FIG. 11



Memory to memory Compression

FIG. 12



Compression of CPU data to Disk or I/O subsystem
FIG. 15

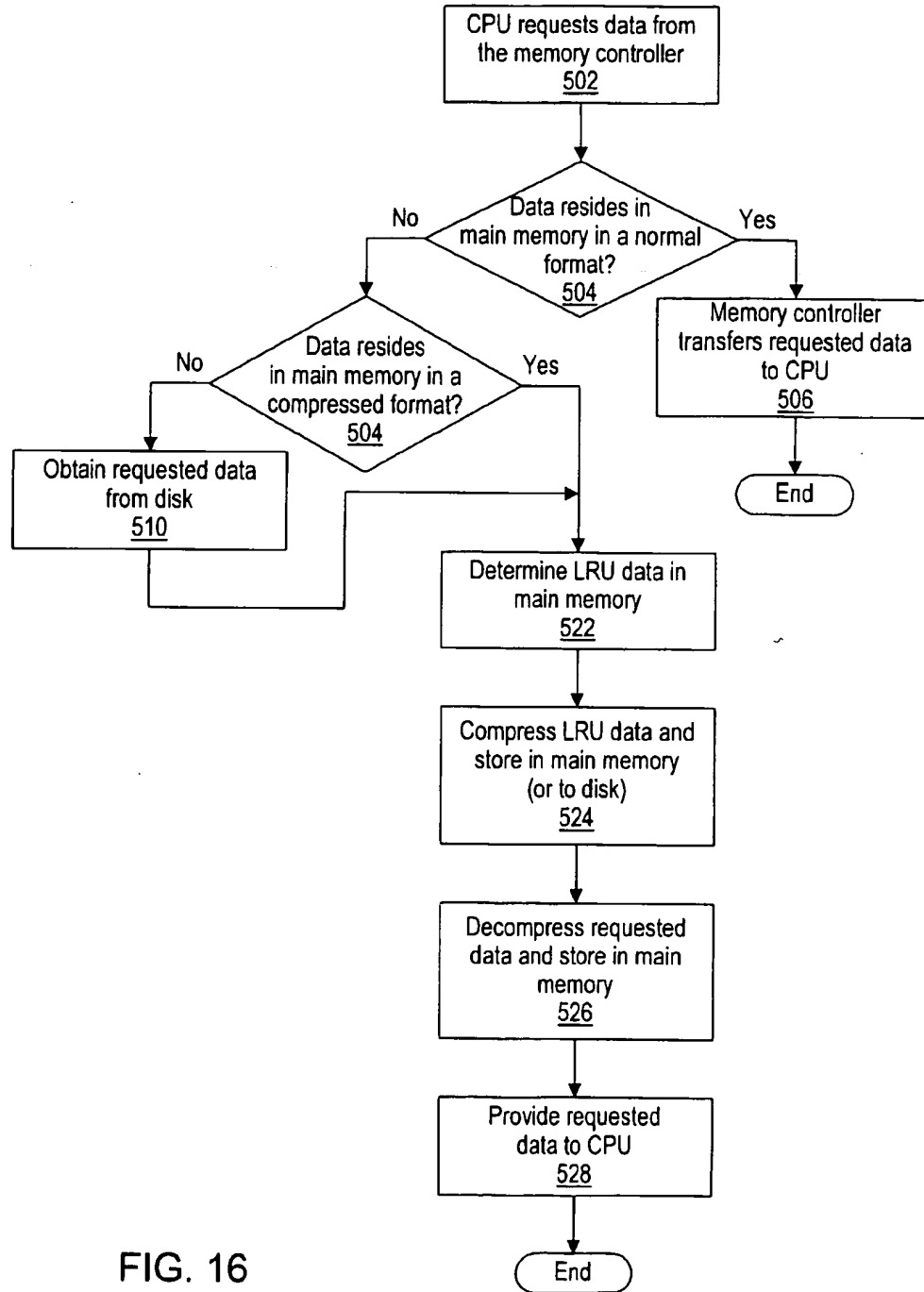


FIG. 16

Mapping Registers

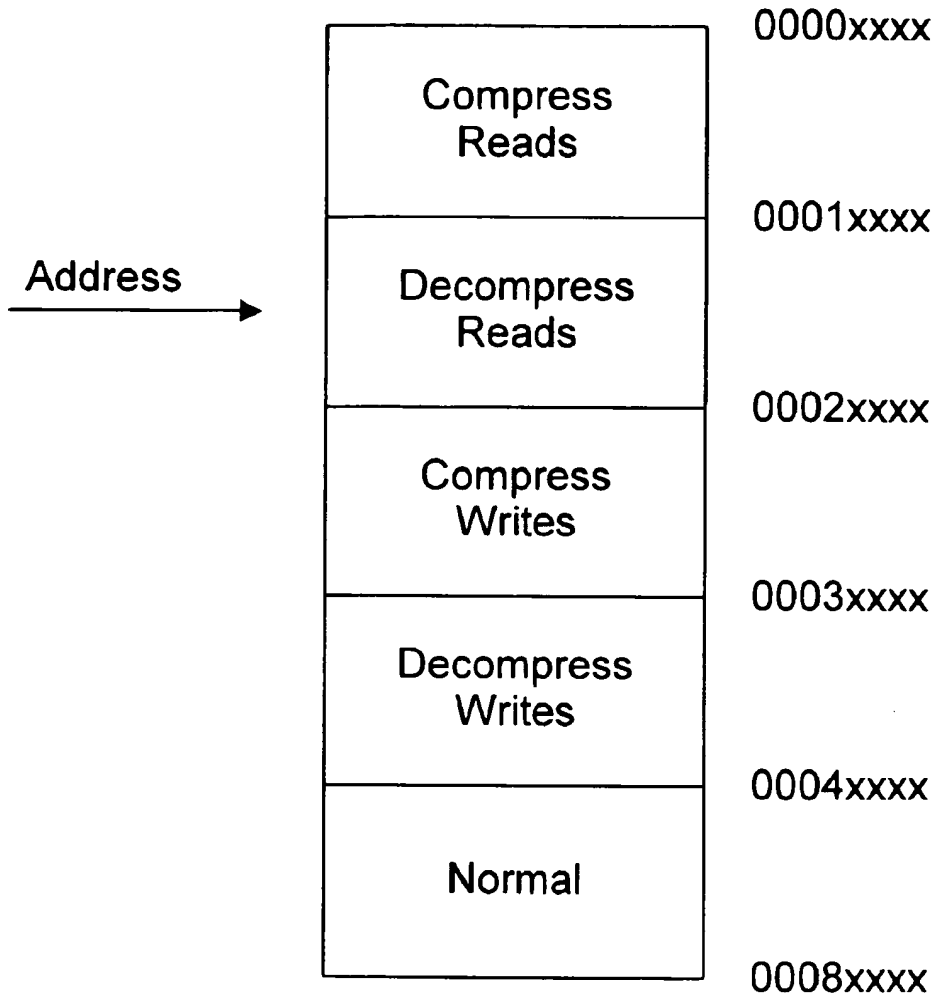


FIG. 17

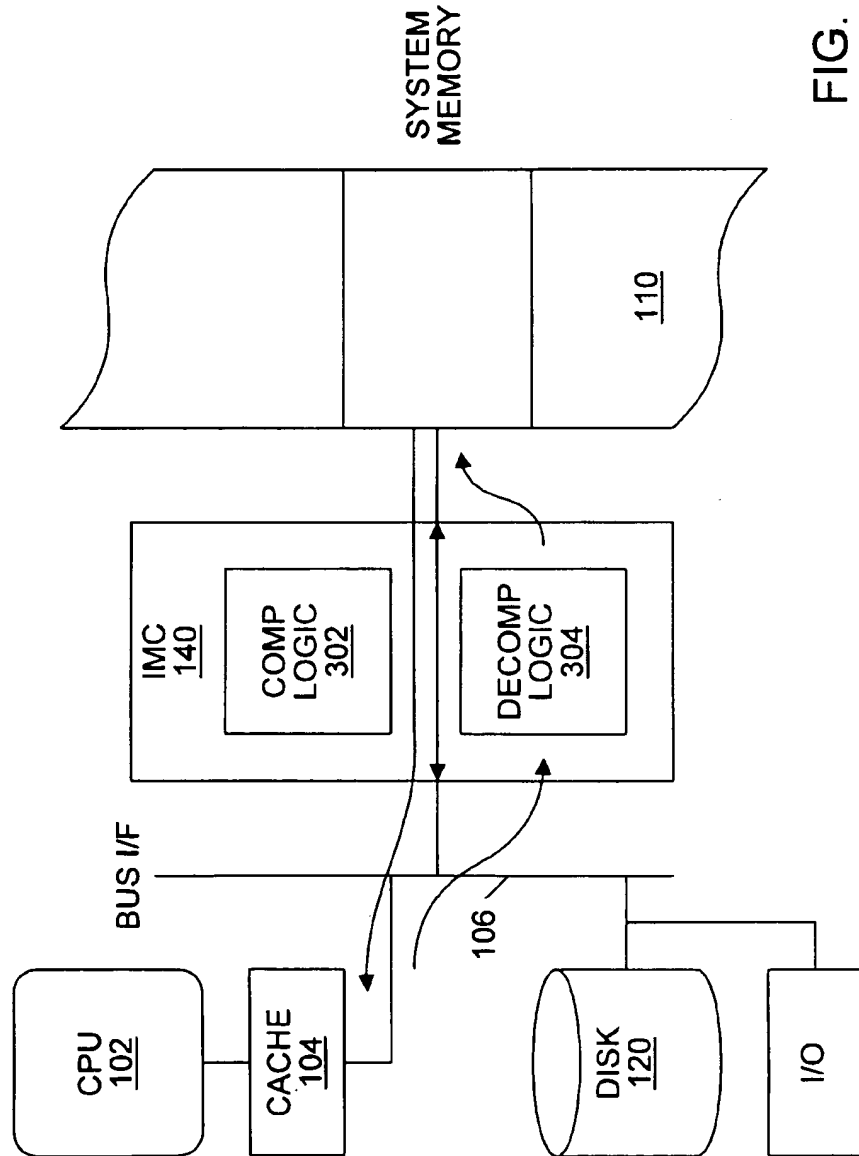


FIG. 18

1

MEMORY CONTROLLER INCLUDING EMBEDDED DATA COMPRESSION AND DECOMPRESSION ENGINES

This is a continuation of application Ser. No. 08/463,106, now abandoned titled "Memory Controller Including Embedded Data Compression and Decompression Engines" filed Jun. 5, 1995, whose inventor is Thomas A. Dye, which is a divisional of application Ser. No. 08/340,667, now U.S. Pat. No. 6,002,411 titled "Integrated Video and Memory Controller with Data Processing and Graphical Processing Capabilities" and filed Nov. 16, 1994, whose inventor is Thomas A. Dye.

FIELD OF THE INVENTION

The present invention relates to computer system architectures, and more particularly to an integrated memory and graphics controller which includes an embedded data compression and decompression engine for increased system bandwidth and efficiency.

DESCRIPTION OF THE RELATED ART

Since their introduction in 1981, the architecture of personal computer systems has remained substantially unchanged. The current state of the art in computer system architectures includes a central processing unit (CPU) which couples to a memory controller interface that in turn couples to system memory. The computer system also includes a separate graphical interface for coupling to the video display. In addition, the computer system includes input/output (I/O) control logic for various I/O devices, including a keyboard, mouse, floppy drive, hard drive, etc.

In general, the operation of a modern computer architecture is as follows. Programs and data are read from a respective I/O device such as a floppy disk or hard drive by the operating system, and the programs and data are temporarily stored in system memory. Once a user program has been transferred into the system memory, the CPU begins execution of the program by reading code and data from the system memory through the memory controller. The application code and data are presumed to produce a specified result when manipulated by the system CPU. The code and data are processed by the CPU and data is provided to one or more of the various output devices. The computer system may include several output devices, including a video display, audio (speakers), printer, etc. In most systems, the video display is the primary output device.

Graphical output data generated by the CPU is written to a graphical interface device for presentation on the display monitor. The graphical interface device may simply be a video graphics array (VGA) card, or the system may include a dedicated video processor or video acceleration card including separate video RAM (VRAM). In a computer system including a separate, dedicated video processor, the video processor includes graphics capabilities to reduce the workload of the main CPU. Modern prior art personal computer systems typically include a local bus video system based on either the peripheral component interconnect (PCI) bus or the VESA (Video Electronics Standards Association) VL bus, or perhaps a proprietary local bus standard. The video subsystem is generally positioned on a local bus near the CPU to provide increased performance.

Therefore, in summary, program code and data are first read from the hard disk to the system memory. The program code and data are then read by the CPU from system memory, the data is processed by the CPU, and graphical

2

data is written to the video RAM in the graphical interface device for presentation on the display monitor. The CPU typically reads data from system memory across the system bus and then writes the processed data or graphical data back to the I/O bus or local bus where the graphical interface device is situated. The graphical interface device in turn generates the appropriate video signals to drive the display monitor. It is noted that this operation requires the data to make two passes across the system bus and/or the I/O subsystem bus. In addition, the program which manipulates the data must also be transferred across the system bus from the main memory. Further, two separate memory subsystems are required, the system memory and the dedicated video memory, and video data is constantly being transferred from the system memory to the video memory frame buffer. FIG. 1 illustrates the data transfer paths in a typical computer system using prior art technology.

Computer systems are being called upon to perform larger and more complex tasks that require increased computing power. In addition, modern software applications require computer systems with increased graphics capabilities. Modern software applications typically include graphical user interfaces (GUIs) which place increased burdens on the graphics capabilities of the computer system. Further, the increased prevalence of multimedia applications also demands computer systems with more powerful graphics capabilities. Therefore, a new computer system and method is desired which provides increased system performance and in particular, increased video and/or graphics performance, than that possible using prior art computer system architectures.

SUMMARY OF THE INVENTION

The present invention comprises an integrated memory controller (IMC) which includes data compression/decompression engines for improved performance. The memory controller (IMC) of the present invention preferably sits on the main CPU bus or a high speed system peripheral bus such as the PCI bus. The IMC includes one or more symmetric memory ports for connecting to system memory. The IMC also includes video outputs to directly drive the video display monitor as well as an audio interface for digital audio delivery to an external stereo digital-to-analog converter (DAC).

The IMC transfers data between the system bus and system memory and also transfers data between the system memory and the video display output. Therefore, the IMC architecture of the present invention eliminates the need for a separate graphics subsystem. The IMC also improves overall system performance and response using main system memory for graphical information and storage. The IMC system level architecture reduces data bandwidth requirements for graphical display since the host CPU is not required to move data between main memory and the graphics subsystem as in conventional computers, but rather the graphical data resides in the same subsystem as the main memory. Therefore, for graphical output, the host CPU or DMA master is not limited by the available bus bandwidth, thus improving overall system throughput.

The integrated memory controller of the preferred embodiment includes a bus interface unit which couples through FIFO buffers to an execution engine. The execution engine includes a compression/decompression engine according to the present invention as well as a texture mapping engine according to the present invention. In the preferred embodiment the compression/decompression

engine comprises a single engine which performs both compression and decompression. In an alternate embodiment, the execution engine includes separate compression and decompression engines.

The execution engine in turn couples to a graphics engine which couples through FIFO buffers to one or more symmetrical memory control units. The graphics engine is similar in function to graphics processors in conventional computer systems and includes line and triangle rendering operations as well as span line interpolators. An instruction storage/decode block is coupled to the bus interface logic which stores instructions for the graphics engine and memory compression/decompression engines. A Window Assembler is coupled to the one or more memory control units. The Window Assembler in turn couples to a display storage buffer and then to a display memory shifter. The display memory shifter couples to separate digital to analog converters (DACs) which provide the RGB signals and the synchronization signal outputs to the display monitor. The window assembler includes a novel display list-based method of assembling pixel data on the screen during screen refresh, thereby improving system performance. In addition, a novel antialiasing method is applied to the video data as the data is transferred from system memory to the display screen. The internal graphics pipeline of the IMC is optimized for high end 2D and 3D graphical display operations, as well as audio operations, and all data is subject to operation within the execution engine and/or the graphics engine as it travels through the data path of the IMC.

As mentioned above, according to the present invention the execution engine of the IMC includes a compression/decompression engine for compressing and decompressing data within the system. The IMC preferably uses a lossless data compression and decompression scheme. Data transfers to and from the integrated memory controller of the present invention can thus be in either two formats, these being compressed or normal (non-compressed). The execution engine also preferably includes microcode for specific decompression of particular data formats such as digital video and digital audio. Compressed data from system I/O peripherals such as the hard drive, floppy drive, or local area network (LAN) are decompressed in the IMC and stored into system memory or saved in the system memory in compressed format. Thus, data can be saved in either a normal or compressed format, retrieved from the system memory for CPU usage in a normal or compressed format, or transmitted and stored on a medium in a normal or compressed format. Internal memory mapping allows for format definition spaces which define the format of the data and the data type to be read or written. Graphics operations are achieved preferably by either a graphics high level drawing protocol, which can be either a compressed or normal data type, or by direct display of pixel information, also in a compressed or normal format. Software overrides may be placed in applications software in systems that desire to control data decompression at the software application level. In this manner, an additional protocol within the operating system software for data compression and decompression is not required.

The compression/decompression engine in the IMC is also preferably used to cache least recently used (LRU) data in the main memory. Thus, on CPU memory management misses which occur during translation from a virtual address to a physical address, the compression/decompression engine compresses the LRU block of system memory and stores this compressed LRU block in system memory. Thus the LRU data is effectively cached in a compressed format

in the system memory. As a result of the miss, if the address points to a previously compressed block cached in the system memory, the compressed block is now decompressed and tagged as the most recently used (MRU) block. After being decompressed, this MRU block is now accessible to the CPU.

The use of the compression/decompression engine to cache LRU data in compressed format in the system memory greatly improves system performance, in many instances by as much as a factor of 10, since transfers to and from disk generally have a maximum transfer rate of 10 Mbytes/sec, whereas the decompression engine can perform at over 100 Mbytes/second.

The integrated data compression and decompression capabilities of the IMC remove system bottle-necks and increase performance. This allows lower cost systems due to smaller data storage requirements and reduced bandwidth requirements. This also increases system bandwidth and hence increases system performance. Thus the IMC of the present invention is a significant advance over the operation of current memory controllers.

BRIEF DESCRIPTION OF THE DRAWINGS

A better understanding of the present invention can be obtained when the following detailed description of the preferred embodiment is considered in conjunction with the following drawings, in which:

FIG. 1 is a prior art diagram illustrating data flow in a prior art computer system;

FIG. 2 is a block diagram illustrating data flow in a computer system including an integrated memory controller (IMC) according to the present invention;

FIG. 3 illustrates a block diagram of a computer system including an IMC according to the present invention;

FIG. 3A illustrates an alternate embodiment of the computer system of FIG. 3 including memory control and graphics/audio blocks coupled to the system memory;

FIG. 3B illustrates an alternate embodiment of the computer system of FIG. 3 including two IMCs coupled to the system memory;

FIG. 3C illustrates an alternate embodiment of the computer system of FIG. 3 including a first IMC coupled to the cache bridge which couples to system memory and a second IMC coupled to the PCI bus which couples to system memory;

FIG. 3D illustrates a computer system including the IMC and using a prior art architecture where the IMC couples to the PCI bus and uses a separate frame buffer memory for video data;

FIG. 4 is a block diagram illustrating the IMC interfacing to system memory and a video display monitor;

FIG. 5 is a block diagram illustrating the internal architecture of the integrated memory controller (IMC) of the present invention;

FIG. 6 illustrates the compression/decompression logic comprised in the IMC 140 according to the present invention;

FIG. 6A illustrates an alternate embodiment including separate compression and decompression engines comprised in the IMC 140 according to the present invention;

FIG. 7 illustrates normal or compressed data transfers in a computer system incorporating the IMC where the IMC does not modify data during the transfer;

FIG. 8 illustrates a memory-to-memory decompression operation performed by the IMC according to the present invention;

5

FIG. 9 illustrates a memory decompression operation performed by the IMC on data being transferred to the CPU or to a hard disk according to the present invention;

FIG. 10 illustrates decompression of data received from the hard disk or CPU that is transferred in normal format in system memory according to the present invention;

FIG. 11 illustrates operation of the IMC decompressing data retrieved from the hard disk that is provided in normal format to the CPU;

FIG. 12 illustrates a memory-to-memory compression operation performed by the IMC according to the present invention;

FIG. 13 illustrates operation of the IMC 140 compressing data retrieved from the system memory and providing the compressed data to either the CPU or hard disk;

FIG. 14 illustrates compression of data in a normal format received from the CPU or hard disk that is stored in compressed form in the system memory;

FIG. 15 illustrates operation of the IMC in compressing normal data obtained from the CPU that is stored in compressed form on the hard disk 120;

FIG. 16 is a flowchart diagram illustrating operation of a computer system where least recently used data in the system memory is cached in a compressed format to the system memory using the compression/decompression engine of the present invention;

FIG. 17 illustrates memory mapping registers which delineate compression and decompression operations for selected memory address spaces; and

FIG. 18 illustrates read and write operations for an address space shown in FIG. 17.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Incorporation by Reference

U.S. patent application Ser. No. 08/340,667 titled "Integrated Video and Memory Controller with Data Processing and Graphical Processing Capabilities" and filed Nov. 16, 1994, is hereby incorporated by reference in its entirety. Prior Art Computer System Architecture

FIG. 1 illustrates a block diagram of a prior art computer system architecture. As shown, prior art computer architectures typically include a CPU 102 coupled to a cache system 104. The CPU 102 and cache system 104 are coupled to the system bus 106. A memory controller 108 is coupled to the system bus 106 and the memory controller 108 in turn couples to system memory 110. In FIG. 1, graphics adapter 112 is shown coupled to the system bus 106. However, it is noted that in modern computer systems the graphics adapter 112 is typically coupled to a separate local expansion bus such as the peripheral component interface (PCI) bus or the VESA VL bus. Prior art computer systems also typically include bridge logic coupled between the CPU 102 and the memory controller 108 wherein the bridge logic couples to the local expansion bus where the graphics adapter 112 is situated. For example, in systems which include a PCI bus, the system typically includes a host/PCI/cache bridge which integrates the cache logic 104, host interface logic, and PCI interface logic. The graphics adapter 112 couples to frame buffer memory 114 which stores the video data that is actually displayed on the display monitor. Modern prior art computer systems typically include between 1 to 4 Megabytes of video memory. An I/O subsystem controller 116 is shown coupled to the system bus 106. In computer systems which include a PCI bus, the I/O subsystem controller 116 typically is coupled to the PCI bus. The I/O subsystem

6

controller 116 couples to an input/output (I/O) bus 118. Various peripheral I/O devices are generally coupled to the I/O bus 18, including a hard disk 120, keyboard 122, mouse 124, and audio digital-to-analog converter (DAC) 144.

Prior art computer system architectures generally operate as follows. First, programs and data are generally stored on the hard disk 120. If a software compression application is being used, data may be stored on the hard disk 120 in compressed format. At the direction of the CPU 102, the programs and data are transferred from the hard disk 120 through the I/O subsystem controller 116 to system memory 110 via the memory controller 108. If the data being read from the hard disk 120 is stored in compressed format, the data is decompressed by software executing on the CPU 102 prior to being transferred to system memory 110. Thus software compression applications require the compressed data to be transferred from the hard disk 120 to the CPU 120 prior to storage in the system memory 110.

The CPU 102 accesses programs and data stored in the system memory 110 through the memory controller 108 and the system bus 106. In processing the program code and data, the CPU 102 generates graphical data or graphical instructions that are then provided over the system bus 106 and generally the PCI bus (not shown) to the graphics adapter 112. The graphics adapter 112 receives graphical instructions or pixel data from the CPU 102 and generates pixel data that is stored in the frame buffer memory 114. The graphics adapter 112 generates the necessary video signals to drive the video display monitor (not shown) to display the pixel data that is stored in the frame buffer memory 114. When a window on the screen is updated or changed, the above process repeats whereby the CPU 102 reads data across the system bus 106 from the system memory 110 and then transfers data back across the system bus 106 and local expansion bus to the graphics adapter 112 and frame buffer memory 114.

When the computer system desires to store or cache data on the hard disk 120 in a compressed format, the data is read by the CPU 102 and compressed by the software compression application. The compressed data is then stored on the hard disk 120. If compressed data is stored in system memory 110 which must be decompressed, the CPU 102 is required to read the compressed data, decompress the data and write the decompressed data back to system memory 110.

Computer Architecture of the Present Invention

Referring now to FIG. 2, a block diagram illustrating the computer architecture of a system incorporating the present invention is shown. Elements in FIG. 2 that are similar or identical to those in FIG. 1 include the same reference numerals for convenience. As shown, the computer system of the present invention includes a CPU 102 preferably coupled to a cache system 104. The CPU 102 may include a first level cache system and the cache 104 may comprise a second level cache. Alternatively, the cache system 104 may be a first level cache system or may be omitted as desired. The CPU 102 and cache system 104 are coupled to a system bus 106. The CPU 102 and cache system 104 are also directly coupled through the system bus 106 to an integrated memory controller (IMC) 140 according to the present invention. The integrated memory controller (IMC) 140 includes a compression/decompression engine for greatly increasing the performance of the computer system. It is noted that the IMC 140 can be used as the controller for main system memory 110 or can be used to control other memory subsystems as desired. The IMC 140 may also be used as the graphics controller in computer systems using prior art architectures having separate memory and video subsystems.

The IMC 140 couples to system memory 110, wherein the system memory 110 comprises one or more banks of memory. In the preferred embodiment, the system memory 110 comprises two banks of memory, and the IMC 140 preferably includes two symmetric memory ports for coupling to the two banks in system memory 110. The IMC 140 of the present invention may couple to any of various types of memory, as desired. In the preferred embodiment, the IMC 140 couples to the system memory 110 through a RAMBUS implementation. For more information on the RAMBUS memory architecture, please see "RAMBUS Architectural Overview," version 2.0, published July 1993 by RAMBUS, Inc., and "Applying RAMBUS Technology to Desktop Computer Main Memory Subsystems," version 1.0, published March 1992 by RAMBUS, Inc., which are both hereby incorporated by reference. In an alternate embodiment, the system memory 110 comprises SGRAM or single in-line memory modules (SIMMs). As noted above, the IMC 140 of the present invention may couple to any of various types of memory, as desired.

The IMC 140 also generates appropriate video signals for driving video display monitor 142. The IMC 140 preferably generates red, green, blue (RGB) signals as well as vertical and horizontal synchronization signals for generating images on the video display 142. Therefore, the integrated memory controller 140 of the present invention integrates memory controller and video and graphics controller capabilities into a single logical unit. This greatly reduces bus traffic and increases system performance. In one embodiment, the IMC 140 also generates appropriate data signals that are provided to Audio DAC 144 for audio presentation. Alternatively, the IMC 140 integrates audio processing and audio DAC capabilities and provides audio signal outputs that are provided directly to speakers. A boot device 146 is also coupled to the IMC 140 to configure or boot the IMC 140, as described further below.

The IMC 140 of the present invention is preferably situated either on the main CPU bus or a high speed system peripheral bus. In the preferred embodiment, as shown in FIGS. 2 and 3, the IMC 140 is coupled directly to the system bus 106 or CPU bus, wherein the IMC 140 interfaces through a cache system 104 to the CPU 102. In an alternate embodiment, the IMC 140 is situated on the peripheral component interconnect (PCI) bus, which is a high speed peripheral local bus standard developed by Intel Corporation. For more information on the PCI bus, please see "PCI System Architecture" by Tom Shanley and Don Anderson, copyright 1993 by MindShare Inc., which is hereby incorporated by reference. Please also see PCI documentation available from Intel Corporation. In this embodiment, the cache 104 preferably comprises a PCI/cache bridge, and the system bus 106 is preferably a PCI bus. However, it is noted that the IMC 140 can sit on any various types of buses as desired.

An I/O subsystem controller 116 is coupled to the system bus 106. The I/O subsystem controller 116 in turn is coupled to an I/O bus 118. Various I/O devices are coupled to the I/O bus including a hard disk 120, keyboard 122, and mouse 124, as shown. In an embodiment including a PCI bus, the I/O subsystem Controller 116 is coupled to the PCI bus.

Typical computer programs require more system bus bandwidth for the transfer of application data than the transfer of program code executed by the CPU. Examples of application data include a bit mapped image, font tables for text output, information defined as constants, such as table or initialization information, etc. Graphical and/or video data, for example, is processed by the CPU 102 for display

before the video data is written to the graphical output device. Therefore, in virtually all cases, the actual program code executed by the CPU 102 which manipulates the application data consumes considerably less system memory 110 for storage than the application data itself.

The IMC 140 includes a novel system architecture which helps to eliminate system bandwidth bottlenecks and removes extra operations required by the CPU 102 to move and manipulate application data. According to the present invention, the IMC 140 includes a data compression/decompression engine which allows application data to move about the system in a compressed format. The operation of the compression/decompression engine in the IMC 140 is discussed in greater detail below.

The IMC 140 also includes a high level protocol for the graphical manipulation of graphical data or video data which greatly reduces the amount of bus traffic required for video operations and thus greatly increases system performance. This high level protocol includes a display list based video refresh system and method whereby the movement of objects on the video display screen 142 does not require movement of pixel data in the system memory 110, but rather only requires the manipulation of display address pointers in a Display Refresh List, thus greatly increasing the performance of pixel bit block transfers, animation, and manipulation of 2D and 3D objects.

FIG. 2 illustrates the data transfer path of data within a computer system including the IMC 140 according to the present invention. As mentioned above, in typical computer systems, the program code and data is initially stored on the hard disk drive 122. First, the IMC 140 reads program code and data stored on the disk 120 using a direct memory access (DMA) and burst control methods where the IMC 140 acts as a master on the system bus 106. The program code and data are read from the disk 120 by the IMC 140 and stored in the system memory 110. In an alternative embodiment, the program code and data are transferred from the disk 120 to the IMC 140 under CPU control. The data is transferred from the hard disk 120 to the system memory 110 preferably in a compressed format, and thus the data requires less disk storage and reduced system bus bandwidth. As the data is transferred from the disk 120 to the IMC 140, the data is preferably decompressed by the decompression engine within the IMC 140 and stored in the system memory bank 110. In general, disk I/O transfer rates are sufficiently slow to allow decompression and storage of the data as the compressed data is received from the disk 120.

The CPU 102 begins program execution by reading the recently decompressed program code from the system memory 110. Portions of the program code contain information necessary to write data and/or instructions back to the IMC 140 using a special graphical protocol to direct the IMC 140 to control the display output on the video display 142. In many cases, the graphical data is not required to leave the system memory 110 and is not required to move to another location in system memory 110, but rather the display list-based operation and high level graphical protocol of the IMC 140 of the present invention enables the CPU 102 to instruct the IMC 104 how window and other graphical data is presented on the screen. This provides a tremendous improvement over prior art systems.

The IMC 140 of the present invention integrates a data compression/decompression engine into the memory controller unit. This reduces the amount of disk storage or archive storage requirements and thus reduces overall system costs. This also reduces the required amount of system memory because, when data is compressed for storage, more

offscreen or non-recently-used data can be stored in system memory 110. This allows faster memory access time since less time is required to decompress the compressed data in system memory 110 than to retrieve the data from the hard disk 120. The incorporation of data compression and decompresses engines in the memory controller unit and also offloads compression tasks from the CPU 102 and avoids use of the cache system for decompression, thereby increasing system performance.

Therefore, the IMC 140 of the present invention reduces the amount of data required to be moved within the system for processing, thus reducing the overall cost while improving the performance of the computer system. According to the present invention, the CPU 102 spends much less time moving data between the various subsystems. This frees up the CPU 102 and allows the CPU 102 greater time to work on the application program rather than moving data around the system.

Computer System Block Diagram

Referring now to FIG. 3, a block diagram illustrating the preferred embodiment of a computer system incorporating the IMC 140 according to the present invention is shown. It is noted that the present invention may be incorporated into any of various types of computer systems having various system architectures. As shown, the computer system includes a central processing unit (CPU) 102 which is coupled through a CPU local bus to a host/PCI/cache bridge 105. The bridge 105 incorporates the cache 104 and I/O subsystem controller 116 of FIG. 2.

The IMC 140 of the present invention couples to the bridge 105. In the preferred embodiment, the IMC 140 comprises a single chip, as shown. However, it is noted that the IMC 140 may comprise two or more separate chips or controllers, as desired. Main memory or system memory 110 couples to the IMC 140. The IMC 140 provides video outputs to video monitor 142 and audio outputs to Audio DAC 144. Speakers 145 are connected to the Audio DAC 144. A boot device 146 is preferably coupled to the IMC 140. The host/PCI/cache bridge 105 also interfaces to a peripheral component interconnect (PCI) bus 118. In the preferred embodiment, a PCI local bus is used. However, it is noted that other local buses may be used, such as the VESA (Video Electronics Standards Association) VL bus or a proprietary bus. In an alternate embodiment, the IMC 140 is coupled directly to the PCI bus 118 as a PCI device. Alternatively, the IMC 140 is adapted to the P6.0 bus, which is a high-speed interconnect for Intel P6 processors and related devices. In one embodiment, the IMC 140 includes a pin-strappable interface which can couple either to the PCI bus or to an address/data CPU bus.

Various types of devices may be connected to the PCI bus 118. It is noted that, in prior art computer systems, a video adapter and video frame buffer would be coupled to the PCI bus 118 for controlling video functions. However, in the computer system of the present invention, video functions are performed by the IMC 140. Also, video data is stored in system memory 110, and thus a separate video frame buffer is not required.

As shown in FIG. 3, a SCSI (small computer systems interface) adapter 119 is coupled to the PCI bus 118. In the embodiment shown in FIG. 3, the SCSI adapter connects to two disk drive units 120, a CD-ROM 130, and a tape drive 132. Various other devices may be connected to the PCI bus 118, such as a network interface card 134. As shown, the network interface card 134 interfaces to a local area network (LAN) 136.

In the embodiment shown, expansion bus bridge logic 150 is coupled to the PCI bus 118. The expansion bus bridge

logic 150 is coupled to the PCI bus 118. The expansion bus bridge logic 150 interfaces to an expansion bus 152. The expansion bus 152 may be any of varying types, including the industry standard architecture (ISA) bus, also referred to as the AT bus, the extended industry standard architecture (EISA) bus, or the microchannel architecture (MCA) bus. Various devices may be coupled to the expansion bus 152, including expansion bus memory 154, a keyboard 122 and a mouse 124. The expansion bus bridge logic 150 also couples to a peripheral expansion bus referred to as the X-bus 160. The X-bus 160 is used for connecting various peripherals to the computer system, such as an interrupt system 162, a real time clock (RTC) and timers 164, a direct memory access (DMA) system 166, and ROM/Flash memory 168, among others.

Alternate Computer System Embodiments

FIG. 3A illustrates an alternate embodiment of the computer system of FIG. 3 including memory control and graphics/audio blocks coupled to the system memory 110. In this embodiment, the host/PCI/cache bridge 105 couples to a memory control block 181 which couples to system memory 110. The host/PCI/cache bridge 105 also couples to a graphics/audio control block 182 which couples to system memory 110. Video monitor 142 and audio DAC 144 are coupled to the graphics/audio control block 182. Speakers 145 connect to the Audio DAC 144. Thus, in this embodiment, the internal logic of the IMC 140 is split into two chips 181 and 182, one comprising the memory control logic 181 and the other comprising the graphics/audio control logic 182. This embodiment is preferably used where it is impractical to include both the memory and graphical capabilities of the IMC 140 of the present invention on a single chip.

FIG. 3B illustrates an alternate embodiment of the computer system of FIG. 3 including two IMCs 140a and 140b coupled between the host/PCI/cache bridge 105 and the system memory 110. In one embodiment the IMC 140a is used solely for memory control functions and the IMC 140b is used solely for graphical and audio functions. Alternatively, the IMCs 140a and 140b each perform both memory and graphics/audio functions for increased performance. For example, the video monitor 142 may optionally be coupled to both IMCs 140a and 140b.

FIG. 3C illustrates an alternate embodiment of the computer system of FIG. 3 including a first IMC 140a coupled between the host/PCI/cache bridge 105 and the system memory 110. A second IMC 140b is coupled to the PCI bus 118, and the second IMC 140b also couples to the system memory 110. Video monitor 142 and Audio DAC 144 are coupled to the IMC 140b and speakers 145 connect to the Audio DAC 145. Alternatively, the first IMC 140a can simply be a memory controller without graphical or audio capabilities.

FIG. 3D illustrates a computer system including the IMC and using a prior art architecture similar to that of FIG. 1. A first IMC 140a or memory controller is coupled between the host/PCI/cache bridge 105 and the system memory 110. A second IMC 140b couples to the PCI bus 118. A frame buffer 141 separate from system memory 110 is coupled to the IMC 140b. Video monitor 142 and Audio DAC 144 are coupled to the IMC 140b and speakers 145 connect to the Audio DAC 145. This embodiment does not have many of the same advantages as the embodiments described above because a separate frame buffer 141 is used. Also, this system requires graphical data or pixel data transfers between the system memory 110 and the frame buffer 141, which are not required in the above systems. Alternatively, the computer system includes a dedicated (non-IMC) memory controller,

and the IMC 140 is used as the graphics accelerator in the graphics adapter 112.

IMC as a Bus Master

In the preferred embodiment, the IMC 140 is a system bus master, thus providing a better cost/performance ratio. In the preferred embodiment of FIG. 3, the IMC 140 can act as a master on the PCI bus 118 in a similar manner that the CPU 102 acts as a master on the PCI bus 118. In one embodiment, the PCI/cache bridge 105 includes arbitration logic, and the CPU 102 and the IMC 140 arbitrate for control of the PCI bus 118. As is well known, a PCI master is able to initiate burst mode or DMA data transfers onto or off-of the system bus, and such transfers minimize the amount of work the CPU 102 and IMC 140 must perform to move data around the system. Since the IMC 140 is a PCI master, memory acquisition or data transfers of certain data-types which are stored in permanent storage (disks) or across the network (LAN) do not consume CPU resources. It is noted that the CPU 102 must service the request to transfer, (IMC register initialization for the transfer). However, the CPU 102 is not required to actually perform the data transfer once the link has been established, and thus CPU processing time is saved. In the preferred embodiment where the IMC 140 is a bus master, once the CPU 102 has set up the data transfer, data movement is controlled by the IMC 140. In this case the IMC 140 may be tasked with decompression of data coming off of the system hard drive. Another example is an external MPEG decoder for live video. Once initialized, the IMC 140 moves and prepares the data for display without CPU intervention. With the IMC's ability to control transfer, decompression and display, the CPU 102 is not required to use processing power in order to transfer data between subsystems.

IMC Interface

Referring now to FIG. 4, a block diagram illustrating how the IMC 140 interfaces to various devices is shown. In the embodiment shown in FIG. 4, the IMC 140 is coupled to a PCI bus wherein the PCI bus is the system bus 106. However, in the preferred embodiment, the IMC 140 is coupled to an expansion bus/cache bridge 105, as shown in FIG. 3. An external BIOS ROM 146 is coupled to the IMC 140 for boot and initialization of the computer system. As mentioned above, in the preferred embodiment the IMC 140 includes dual memory control units for connection of up to 512 Megabytes of system memory. Each memory control unit generates respective address and data signals as shown. For example, a first memory control unit generates address and data signals (Add1 and Data1) and a second memory control unit also generates address and data signals (Add2 and Data2). In an alternate embodiment, the IMC 140 includes a single memory control unit. The IMC 140 also generates the appropriate video signals for driving the video display monitor 142. As shown, the IMC 140 generates red, green and blue signals referred to as red, grn and blu, for driving the video display monitor 142 and generates horizontal and vertical synchronization signals referred to as HSYNC and VSYNC and VSYNC, respectively. The IMC 140 further generates audio signals to an Audio DAC 144, which in turn provides analog audio signals to one or more speakers (not shown).

IMC System Boot Procedure

The BIOS ROM 146 stores boot data, preferably in a compressed format. At power-up, the IMC 140 reads and decompresses the BIOS data from the BIOS ROM 146 into a normal format and loads the data into the system memory 110. In the preferred embodiment, all memory accesses are suspended until the boot code has been transferred to the

system memory 110 and is ready to be read. All internal IMC mapping registers default to point to the boot code for power on operation. Once the boot code has been loaded into system memory 110, the CPU 102 traps the starting address of the boot code to begin boot operations.

The boot code is responsible for a number of configuration options of the IMC 140. When a reset input to the IMC 140 referred to as nRESET goes inactive high, configuration resistors tied to inactive signals determine the start up procedures. If the configuration is set to boot from the IMC boot code, the data is read by the IMC 140, optionally decompressed, and transferred into the system memory 110. Before this operation can take place, the IMC 140 must also be programmed. When the boot device 146 is connected to the IMC 140, the first portion of the boot code is specific to the IMC 140. This code is read from the boot device 146 into the IMC instruction register FIFO. IMC instructions such as load and store registers set up the initialization of the IMC. These operations include but are not limited to: set refresh, map PCI memory bounds, initialize display timing, and read main CPU boot code to specific system memory address. In addition, if the boot code is in a compressed format, the IMC initialization routine sets up the IMC for decompression of such code. It is noted that all boot code for the IMC is in a "non-compressed" format. Once the system boot and driver have been initialized, the IMC protocol for instruction processing can be in a compressed format.

Once the boot code is transferred to the system memory 110 by the IMC 140, an NMI or high level interrupt is generated from the IMC interrupt output pin. Optionally, the IMC can communicate a "NOT READY" status to the CPU 102 to prevent access until the boot memory 146 is in place. After the IMC 140 has set the memory bounds and configured the PCI interface configuration, set display and memory refresh timings, decompressed and/or loaded host CPU boot code into system memory, an interrupt out instruction from the IMC 140 directs the host CPU 102 to begin instruction execution for completion of system initialization.

Non-IMC System Boot Procedure

In an alternate embodiment, the computer system does not include a boot device coupled to the IMC boot device port. In this embodiment, the IMC 140 resides in the system as a coprocessor. A waiting register loads into the IMC 140 to enable access to the main memory 110. In an embodiment where the IMC 140 is coupled to the PCI bus, the IMC 140 contains the correct configuration information in order for the system to recognize the IMC 140 as a PCI peripheral device. In this architecture the host CPU 102 is responsible for register loads to initialize the IMC 140. Such initialization sets up the decode memory map for non-compressed and compressed data storage, as well as the display for output and any other set-up required to boot the operating system.

IMC Block Diagram

FIG. 5 illustrates a more detailed block diagram of the internal components comprising the IMC 140 of the present invention. It is noted that various of the elements in FIG. 5 are interconnected with each other, wherein many of the various interconnections are not illustrated in FIG. 5 for simplicity.

As shown, the IMC 140 includes bus interface logic 202 for coupling to the host computer system, i.e., for coupling to the system bus 106. In the preferred embodiment, the system bus 106 is the CPU bus or host bus. Alternatively, the system bus 106 is the PCI bus, and the bus interface logic 202 couples to the PCI bus. Instruction storage/decode logic 230 is coupled to the bus interface logic 202.

The bus interface logic 202 couples to an execution engine 210 through two first in first out (FIFO) buffers 204 and 206. In other words, the two FIFO buffers 204 and 206 are coupled between the bus interface logic 202 and the execution engine 210. The FIFO buffers 204 and 206 decouple data transfers between the external asynchronous computer system and the synchronous logic comprised within the IMC 140. The execution engine 210 includes a data compression/decompression (codec) engine according to the present invention, as described further below. The execution engine 210 also include texture mapping logic for performing texture mapping on pixel data. In one embodiment, the execution engine 210 includes separate compression and decompression engines.

The execution engine 210 couples to a graphics engine 212. The graphics engine 212 essentially serves as the graphical adapter or graphics processor and includes various graphical control logic for manipulating graphical pixel data and rendering objects. The graphics engine 212 includes polygon rendering logic for drawing lines, triangles, etc., i.e., for interpolating objects on the display screen 142. The graphics engine 212 also includes other graphical logic, including ASCII to font conversion logic, among others. The instruction storage/decode logic 230 stores instructions for execution by the graphics engine 212.

In one embodiment, the execution engine 210 comprises a DSP engine which performs both codec functions as well as graphical functions. In one embodiment, the DSP engine includes one or more ROMs which store different microcode depending on the task being performed, and the DSP engine dynamically switches between different sets of microcode to perform different tasks.

The graphics engine 212 couples to respective memory control units referred to as memory control unit #1 220 and memory control unit #2 222 via respective FIFO buffers 214 and 216, respectively. Memory control unit #1 220 and memory control #2 222 provide interface signals to communicate with respective banks of system memory 110. In an alternate embodiment, the IMC 140 includes a single memory control unit. The graphics engine 212 reads graphical data from system memory 110, performs various graphical operations on the data, such as formatting the data to the correct x, y addressing, and writes the data back to system memory 110. The graphics engine 212 performs operations on data in the system memory 110 under CPU control using the high level graphical protocol. In many instances, the graphics engine 212 manipulates or resets pointers and manipulates data in windows workspace areas in system memory 110, rather than transferring the pixel data to a new location in system memory 110.

The two memory control units 220 and 222 can each preferably address up to 256 Megabytes of system memory 110. Each memory control unit 220 and 222 comprises a complete address and data interface for coupling to system memory 110. Each memory control unit 220 and 222 also includes internal collision logic for tracking of operations to avoid data coherency problems. The memory control units 220 and 222 are coupled internally and include a complete display list of memory operations to be performed. Multiple display lists are used for memory transfers as well as screen refresh and DRAM refresh operations. Both memory control units 220 and 222 span the entire memory interface address space and are capable of reading any data comprised within the system memory 110.

A Window Assembler 240 is coupled to each of the memory control units 220 and 222. The Window Assembler 240 includes logic according to the present invention which

assembles video refresh data on a per window or per object basis using a novel pointerbased Display Refresh List method. This considerably improves system and video performance. The Display Refresh List is stored in system memory 110 and uses pointers which reference video data for display. The Window Assembler 240 also uses a respective window workspace located in system memory 110 for each window or object on the display screen 142. In other words, the Window Assembler 240 includes memory mapped I/O registers which point to applications-specific memory areas within the system memory 110, i.e., areas of system memory 110 which are mapped as windows workspace memory. Each window workspace contains important information pertaining to the respective window or application, including the position of the window on the display, the number of bits per pixel or color composition matrix, depth and alpha blending values, and respective address pointers for each function. Thus each window on the display screen includes an independent number of colors, depth, and alpha planes. The information in each respective window workspace is used by the Window Assembler 240 during screen refresh to draw the respective window information on the display screen 142.

Therefore, the system memory 110 includes workspace areas which specify data types, color depths, 3D depth values, screen position, etc. for each window on the screen. A Display Refresh List or queue is located in system memory 110, and the Window Assembler 240 dynamically adjusts and/or constructs the Display Refresh List according to the movement of data objects which appear on the video display screen 142. Thus, when an object or window is moved to a new position on the video screen, the data comprising the object does not transfer to another location in system memory 110. Rather, only the display pointer address is changed in the system memory 110, and this change is reflected in the Display Refresh List. This provides the effect of moving data from a source address to a destination address, i.e., a bit block transfer (bit blit), without ever having to move data comprising the object to a new location in system memory 110. This provides greatly increased performance over conventional bit blit operations commonly used in graphical systems.

The Window Assembler 240 is coupled to a display storage buffer 244 where the screen refresh pixel data is stored. The display storage buffer 244 is coupled to a display memory shifter 246 which in turn is coupled to respective red, green and blue digital to analog converters (DACs) which provide the respective red, green and blue signals to the display unit 142. The IMC 140 also provides horizontal and vertical synchronization signals (not shown in FIG. 4). In one embodiment, the Window Assembler 240 also provides audio signal outputs to an Audio Shifter 242 which provides audio output signals, as shown.

The IMC 140 includes a bursting architecture designed to preferably burst 8 bytes or 64 bits of data during single transfers, and can also burst 32 bit (4 byte) transfers for PCI bus transfers. The IMC 140 also includes logic for single byte and multiple byte operations using either big or little endian formats. The IMC 140 transfers data between the system bus and main memory 110 and also transfers data between the system memory 110 and the internal shift registers 244 and 246 for graphical display output. All data transferred within the IMC 140 is subject to operation within the execution engine 210 and/or the graphics engine 212 as the data traverses through the data path of the IMC 140.

Compression/Decompression Engine

Referring now to FIG. 6, the execution engine 210 preferably includes a single compression/decompression

engine 301 which performs compression and decompression functions. This single engine 301 is preferably a dedicated codec hardware engine. In one embodiment, the codec engine 301 comprises a DSP core with one or more ROMs which store different sets of microcode for certain functions, such as compression, decompression, special types of graphical compression and decompression, and bit blit operations, as desired. In this embodiment, the codec engine 301 dynamically shifts between the different sets of microcode in the one or more ROMs depending on the function being performed.

As shown in FIG. 6A, in one embodiment, the execution engine 210 in the IMC 140 preferably includes an embedded lossless data compression engine 302 and decompression engine 304 designed to compress and decompress data as data is transferred to/from system memory 110. In the following description, the execution engine 210 is described as having separate compression and decompression engines 302 and 304. In the present disclosure, the term "compression/decompression engine" includes a single integrated engine which performs compression and decompression functions as well as separate compression and decompression engines.

Thus, the IMC 140 includes two data formats referred to as "compressed" data and "normal" data. The compressed data format requires less storage and thus is less expensive. The compressed format also requires less system bandwidth to transfer data between system memory 110 and I/O subsystems. Compression of normal data format to compressed data format results in a small performance penalty. However, the decompression of compressed data format to normal data format does not have an associated penalty. In one embodiment, the compression engine 302 is implemented in software by the CPU 102.

In the preferred embodiment, the compression engine 302 and decompression engine 304 comprise hardware engines in the IMC 140, or alternatively use pieces of the same engine for compression and decompression. In the preferred embodiment, the compression engine 302 and decompression engine 304 in the IMC 140 comprise one or more hardware engines which perform LZRW compression and decompression. For more information on a data compression and decompression system using LZRW compression, please see U.S. Pat. No. 4,701,745, titled "Data Compression System," which issued Oct. 20, 1987 and which is hereby incorporated by reference in its entirety. In an alternate embodiment, the data compression and decompression engines 302 and 304 utilize the data compression/decompression processor hardware disclosed in U.S. Pat. No. 5,410,671, titled "Data Compression/Decompression Processor," which issued Apr. 25, 1995 and which is hereby incorporated by reference in its entirety. Other types of data compression/decompression methods may be used. For examples of other data compression/decompression methods which can be used in the hardware engines 302 and 304 of the present invention, please see U.S. Pat. Nos. 4,464,650 and 4,558,302 which are both hereby incorporated by reference. The above two patents present implementations of a data compression method described by Lempel and Ziv in "Compression of Individual Sequences Via Variable-Rate Coding," IEEE Transactions on Information Theory, IT-5, September 1977, pages 530-537, and "A Universal Algorithm for Sequential Data Compression," IEEE Transactions on Information Theory, IT-23-3, May 1977, pages 337-343 and the above two articles are both hereby incorporated by reference.

The compression engine 302 and decompression engine 304 of the present invention may also include specialized

compression/decompression engines for image data. For example, one embodiment utilizes compression and decompression engines 302 and 304, which are shown and described in U.S. Pat. No. 5,408,542, titled "Method and Apparatus for Real-Time Lossless Compression and Decompression of Image Data," which issued Apr. 18, 1995 and which is hereby incorporated by reference in its entirety. In an alternative embodiment, the compression and decompression engines 302 and 304 utilize lossy decompression techniques and comprise the system and method taught in U.S. Pat. No. 5,046,119 titled "Method and Apparatus for Compressing and Decompressing Color Video Data with an Anti-Aliasing Mode," this patent being hereby incorporated by reference in its entirety. For related information on compression and decompression engines for video applications, please see U.S. Pat. No. 5,379,356 titled "Decompression Processor for Video Applications," U.S. Pat. No. 5,398,066 titled "Method and Apparatus for Compression and Decompression of Digital Color Images," U.S. Pat. No. 5,402,146 titled "System and Method for Video Compression with Artifact Dispersement Control," and U.S. Pat. No. 5,379,351 titled "Video Compression/Decompression Processing and Processors," all of which are hereby incorporated by reference in their entirety.

For other types of data compression and decompression methods which may be used in the compression and decompression engines 302 and 304 of the present invention, please see U.S. Pat. No. 5,406,279 titled "General Purpose, Hash-Based Technique for Single Pass Lossless Data Compression," U.S. Pat. No. 5,406,278 titled "Method and Apparatus for Data Compression Having an Improved Matching Algorithm which Utilizes a Parallel Hashing Technique," U.S. Pat. No. 5,396,595 titled "Method and System for Compression and Decompression of Data."

In the preferred embodiment of the invention, the compression engine 302 and decompression engine 304 use a lossless compression method. Any of various lossless compression methods may be used as desired. As noted above, in the preferred embodiment, LZRW compression is used as shown in U.S. Pat. No. 4,701,745. However, it is noted that other lossless compression methods may be used, and in some embodiments lossy compression methods may be used as desired.

In the preferred embodiment of the invention, the compression engine 302 and decompression engine 304 are hardware engines comprised of logic circuitry. In an alternate embodiment, the compression and decompression engines 302 and 304 include a dedicated compression/decompression processor which executes instructions out of a ROM or RAM memory. Various other implementations may be used to embed a compression/decompression within the memory controller according to the present invention.

According to the present invention, a software subroutine executing on the CPU 102 directs the IMC to compress data before the data is written to system memory 110 or hard disk 120. This is preferably accomplished after the compilation period of the software and thus does not affect the performance of run time executables. During program execution, the compressed data, in the form of either executables or data files, is decompressed by the decompression engine 304 in the IMC 140 as data is retrieved from the system memory 110. Data stored in compressed format either on the hard disk 120 or on other I/O subsystems such as a LAN (local area network), serial ports, etc., is transferred to the system memory 110 and is either decompressed to normal data by the decompression engine 304 in the IMC 140 during the transfer or is stored as compressed data in the system memory 110 for later decompression.

The operation of the compression unit 302 and the decompression unit 304 in the IMC 140 are completely transparent to system level application software. According to the present invention, special directives are included in the computer's operating system software which imbed directives used in file and data transfers, where the directives are used by the IMC 140 for data manipulation. In this manner, the IMC 140 predicts the necessary data manipulation required, i.e., compression or decompression, ahead of the actual execution requirements of the software application. This system level architecture provides a mechanism for the determination of when and how data is to be transferred and the particular data format, either normal or compressed format, in which the data is to be represented. Software overrides may also be included in software applications in systems where it is desired to control decompression of data at the software application level. In this manner, an additional protocol for data compression or decompression is not required.

Data decompression is particularly important for live video system throughput and texture map storage. In prior art computer systems, live video is limited by the data transfer rate of the raw digital video data between the storage device, the system bus, and the system memory 110 or video subsystem. The IMC 140 of the present invention provides video acceleration with minimal CPU overhead because the IMC 140 decompresses the incoming video data. It is noted that the IMC 140 requires external video input digitization for live video. The IMC 140 also may require an external device for compression of some video formats, such as MPEG.

In addition, while incoming video input is received by the IMC 140, decompressed, and transferred to the hard disk 120 or other I/O device, the video data may also be stored in normal format in the system memory 110 for immediate display on the video monitor 142. The video data stored in the system memory 110 is displayed according to the refresh display list system and method of the present invention comprised in the Window Assembler 240. Thus, this provides the mechanism for receiving video, storing it in compressed format on the disk 120, and also displaying the live video on the display screen 142 in real time during video capture with minimal CPU involvement. Also, as discussed further below, the pointer-based display list video refresh system and method of the present invention provides greatly improved video display capabilities than that found in the prior art. In the 3-D video game market large amounts of memory storage are required to store and manipulate texture images for texture mapping. By storing the texture source (or texels) in compressed format, the IMC 140 reduces both hard disk and memory capacity requirements. The IMC 140 can then be directed by the CPU 102 to expand the compressed textures before texture mapping of display objects is required.

FIGS. 7-15 illustrate various examples of data compression, data decompression, and data transfer within a computer system including an IMC 140 according to the present invention. FIG. 7 illustrates data transfer in either a normal format or compressed format within the computer system without modification by the IMC 140. Thus, the IMC allows data transfers by the system DMA logic or CPU without performing any type of compression or decompression operations, i.e., without any special functions or operations on the data stream. The data is stored in memory or is transferred to the disk or I/O subsystem without any modifications. It is noted that this mode represents the standard prior art method for system data transfer where no compression

or decompression operations are performed on the data by the memory controller. In this mode, the IMC 140 is unaware of the data format type and whether the data is for transfer or storage.

FIG. 8 illustrates a memory-to-memory decompression operation implemented by the IMC 140 according to the present invention. As shown, the IMC 140 performs decompression of data within the system memory 110 without host CPU intervention, i.e., without requiring intervention of software routines executing on the host CPU 102. As shown in FIG. 8, compressed data stored in the system memory is expanded into a normal data format by passing through the decompression engine 304 in the IMC 140. This operation is necessary for preparation of executables which contain instructions and operands directly responsible for CPU program execution. The IMC 140 is directed by initialization code in the form of a malloc instruction to allocate a block for executable storage and to decompress the existing routines already present in the memory subsystem.

FIG. 9 illustrates operation of the decompression engine 304 in the IMC 140 obtaining compressed data from the system memory 110, decompressing the data, and transferring the data to the CPU 102 or hard disk 120. Thus, the CPU 102 or hard disk 120 or respective I/O subsystem is capable of reading normal noncompressed data for storage and/or execution from the system memory 110 even when the data stored in system memory is stored in a compressed format. The decompression engine 304 and the IMC 140 operates transparently relative to the remainder of the computer system and operates to transform compressed memory data stored in system memory 110 into noncompressed data or data in the normal format. The decompression operation is transparent and occurs during a read operation from the CPU to system memory 110. The IMC 140 also includes a look ahead architecture system which ensures that the data being read is always available. Thus, stall-out, i.e., the decompression engine 304 failing to keep up with the CPU requests, only occurs when the CPU reads blocks of nonsequential data.

FIG. 10 illustrates operation of the IMC 140 in decompressing data from either the CPU 102 or hard disk 120 and storing the decompressed or normal data into system memory 110. Thus, data can be transferred from hard disk 120 and I/O subsystem or from the CPU 102 can be decompressed and stored in a normal format for later execution or use. This mode of operation is preferably the standard mode. This method allows smaller data files and smaller amounts of information to be transferred on the system bus as data is read from a hard disk 120 or from a local area network (LAN) via a network interface card. The CPU 102 may also obtain and/or move data from a compressed format and store the data in a normal format in the system memory 110 without the CPU 102 having to execute a decompression algorithm in software. This enables executable programs that are stored on the hard disk 120 in compressed format that are transferred by the CPU 102 in compressed format to be expanded within the IMC 140 into a normal format during memory storage.

FIG. 11 illustrates compressed data transferred from the hard disk 120 decompressed within the IMC 140 and read as normal data by the CPU 102. This is for cases where it is desirable for the CPU to read data from the hard disk 120 or an I/O subsystem where the data is stored in a compressed format and CPU 102 desires to read the data in a normal format or noncompressed format. The IMC 140 includes a special transfer mode by which the data is not required to be temporarily stored in the system memory 110 in order for

decompression to occur. It is noted, however, that the data transfer time may actually be increased in this mode due to the duality of the single interface bus at the interface of the IMC 140. In one embodiment of the invention, the decompression logic 304 includes a dual ported nature with FIFOs at each end wherein compressed data is read into one end and decompressed data is output from the other to increase decompression operations.

FIG. 12 illustrates operation of the IMC 140 in converting normal data, i.e., data in a normal format, in the system memory 110 into data stored in a compressed format within the system memory 110. In one embodiment, the IMC 140 includes a compression engine 302 which accompanies software compression performed by the CPU 102. In some applications, it is faster and more convenient to be able to compress data off line without CPU intervention. This compression operation may generally be used for areas of "cached-out" program or operand data, i.e., data stored in the system memory 110 that is either non-cacheable or is not currently in the cache memory. Thus, the IMC 140 allows for memory compaction during a software application's memory allocation and cleanup routine. FIG. 12 illustrates how the IMC 140 can read data in its normal format from the system memory 110, compress the data, and then write the data back to system memory 110 for later decompression. This is a dynamic operation and can be imbedded into software applications as desired.

FIG. 13 illustrates operation of the compression engine 302 in the IMC 140 retrieving data stored in a normal format in the system memory 110 and providing compressed data to either the CPU 102 or the hard disk 120. In a computer system incorporating the IMC 140 according to the preferred embodiment, this operation of the compression engine 302 in transferring data stored in a normal format from system memory 110 and storing the data in a compressed format on the hard disk 120 is preferably one of the most common uses for the IMC compression engine 302.

As shown, data stored in the normal format in the system memory 110 can effectively be "cached" onto the hard disk 120 or an I/O subsystem in compressed format for later use. This method is substantially more efficient than normal data transfers because, due to the compression, the amount of data transferred is less. When a memory miss occurs, i.e., when the CPU requests data from the system memory 110 and the data is not present in the system memory 110 because the data has been stored in a compressed format on the hard disk 120, data in the system memory 110 that has been least recently used is written in compressed format to the disk to make room for the data requested by the CPU 102. Thus, this operation is similar to a cache system where, on a cache miss, the least recently used (LRU) data is overwritten with the requested data because this data is the least likely to be requested in the future. If the CPU 102 includes an internal first level cache system and the cache system 104 is a second level cache system, the system memory 110 effectively acts as a third level cache system storing LRU data in a compressed format in main memory rather than writing the data back to the hard disk 120.

As shown in FIG. 12, instead of transferring the LRU data from system memory 110 to the hard disk 120, the data is not cached to disk but rather is compressed by the compression engine 302 and stored in system memory 110 in compressed format. For example, when a page miss occurs the data is conventionally transferred to the hard disk. However, according to the present invention, the data is stored in system memory 110 in compressed format. This allows faster recall of data when a page miss occurs since the

requested data is still in system memory 110, albeit in compressed format.

The compression engine 302 in the IMC 140 provides that only compressed data is transferred between the hard disk 120 and the system memory 110, thus providing substantially faster transfers because of the reduced amount of data required to be transferred. This greatly increases the performance and storage capability of computer systems which implement virtual memory by swapping data from the system memory 110 to and from the hard disk 120. It is further noted that the IMC 140 compresses data stored in the normal format in system memory 110 and transfers this compressed data to the CPU if the CPU 102 desires to obtain the data in a compressed format. It is anticipated that this will not be as common as the transfer of data in a normal format in system memory 110 to a compressed format on the hard disk 120 as described above.

FIG. 14 illustrates data in a normal noncompressed format transferred from either the hard disk 120 or CPU 102 to the IMC 140 where the compression engine 302 in the IMC 140 converts the data into compressed data and stores the compressed data in the system memory 110. It is noted that there are generally rare occasions when the hard disk 120, an I/O subsystem, or even the CPU 102 transfers data in normal format to the IMC where it is desirable to store the data in compressed format in the system memory 110. This could typically occur from foreign applications programs loaded into from the floppy drive or retrieved from a local area network where it is desirable to compress this information before use or storage in the main system memory 110. Another usage is for storage of bitmaps and texture maps which must be animated in real time. Here the disk or LAN is too slow to load and register the image data for animation. In this example, the IMC 140 registers compressed bit maps (stored in compressed format on disk) and then uses the method shown in FIG. 8 on an "as needed" basis.

FIG. 15 illustrates compression of data from the CPU 102 and storage of the compressed data on the hard disk 120 or transferred over another I/O subsystem. Thus, another feature of the compression engine 302 of the present invention is the ability to write CPU data in normal format directly onto the system disk 120 or I/O subsystem in a compressed format. This is performed without requiring the CPU 102 to implement a special software compression algorithm, thus saving CPU resources.

Compression/Decompression Engine for Caching Data in a Compressed Format

The compression/decompression engine 301 in the IMC 140 is also preferably used to cache least recently used (LRU) data in the main memory 110. Thus, on CPU memory management misses, which occur during translation from a virtual address to a physical address, the compression/decompression engine 301 compresses the LRU block of system memory 110 and stores this compressed LRU block in system memory 110. Thus the LRU data is effectively cached in a compressed format in the system memory 110. As a result of the miss, if the address points to a previously compressed block cached in the system memory 110, the compressed block is decompressed and tagged as the most recently used (MRU) block. After being decompressed, this MRU block is now accessible to the CPU 102.

Referring now to FIG. 16, a flowchart diagram is shown illustrating operation of the computer system where the compression/decompression engine is used to store or "cache" LRU data in a compressed format in the system memory 110. In step 502 the CPU 102 requests data from the system memory 110, i.e., the CPU provides addresses of

requested data to the IMC 140. In step 504 the IMC 140 determines if the data resides in the main memory 110 in a normal format, i.e., the IMC 140 determines if the data resides in the "system memory cache". If so, then in step 506 the IMC 140 transfers the requested data to the CPU 102, and operation completes.

If the data is determined to not reside in the main memory 110 in a normal format, then in step 508 the IMC 140 determines if the data resides in the main memory 110 in a compressed format. It is noted that the determinations of steps 504 and 508 may essentially be performed in the same step. If the data does not reside in the main memory 110 in a compressed format, then the data must be cached on the disk subsystem 120, and in step 510 the requested data is retrieved from the disk subsystem 120.

If the data resides in the main memory 110 in a compressed format, then in step 522 the IMC 140 determines the least recently used data in main memory 110. Step 522 involves either determining the "true" LRU data or determining "pseudo LRU" data according to a desired replacement algorithm. In the present disclosure, the term "least recently used data" or "LRU data" refers to the data the IMC 140 decides to compress and store (cache) in the system memory 110, presumably because this data was determined to be the least likely to be accessed by the CPU 102 in the future.

In step 524 the IMC 140 compresses the LRU data and stores the compressed LRU data in main memory 110. The compressed LRU data may also be cached to the disk subsystem 120 if additional free system memory space is needed. In step 526 the IMC 140 decompresses the requested data and stores the uncompressed requested data back to main memory 110. The IMC 140 also preferably marks this data as most recently used (MRU) data. In step 528 the IMC 140 provides the requested data to the CPU 102, and operation completes.

It is noted that if the requested data resides in the disk subsystem 120, then the data is retrieved by the IMC 140 in step 510 and steps 522-528 are then performed as described above. In this instance, step 526 is performed only if the data was stored on the disk subsystem 120 in a compressed format, which is typically the case.

The use of the compression/decompression engine to cache LRU data in compressed format in the system memory greatly improves system performance, in many instances by as much as a factor of 10, since transfers to and from disk generally have a maximum transfer rate of 10 Mbytes/sec, whereas the decompression engine can perform at over 100 Mbytes/second.

Mapping System Memory as Compressed and Normal

Under normal operations where the compression/decompression engine is not used, the operating system software maps the IMC 140 as normal "physically addressed" memory. For certain applications it is more advantageous to map the system memory 110 into compressed and normal data storage areas. This allows the operating system to read and write to alternate address ranges where the data is compressed or decompressed during access or operation. This stage is preferably determined by information in an "attributes" list which stores attributes about each window or object on the screen. The attributes list is used by the Window Assembler 240 to maintain information about windows or objects on the screen. For more information on the attributes list and the operation of the Window Assembler 240, please see FIG. 18 and the associated text in U.S. patent application Ser. No. 08/340,667, referenced above.

FIG. 17 illustrates an example of mapping registers which determine whether the system memory space is mapped into compressed or normal data storage areas. Thus, as the address is input to the mapping registers, the compression/decompression engine is engaged depending on the pre-defined "locked" memory bounds for each system memory region.

As shown in FIG. 17, address range 0000xxxx to 0001xxxx is designated with "compress reads", address range 0001xxxx to 0002xxxx is designated with "decompress reads", address range 0002xxxx to 0003xxxx is designated with "compress writes", address range 0003xxxx to 0004xxxx is designated with "decompress writes", and address range 0004xxxx to 0008xxxx is designated with "normal". Thus, if an address is in the range 0003xxxx to 0004xxxx, then reads are normal and writes are decompressed, which is shown in FIG. 18. It is noted that all combinations are possible, including any combination of normal, compressed, and decompressed transfers for reads and writes.

Thus, according to the present invention, the operating system tags system memory 110 for usage. In addition, the IMC 140 maps areas of system memory as compressed or decompressed.

Conclusion

Therefore, the IMC 140 of the present invention includes a compression/decompression engine 301 which off loads work from the CPU 102 and provides increased data transfer capabilities that reduce the amount of data required to be transferred. The IMC 140 of the present invention incorporates compression and decompression in the memory subsystem and thus off loads the host CPU 102 from having to perform this function. Thus, as shown above, multiple choices are available for cost and performance enhancements, and the IMC of the present invention provides numerous advances over the prior art.

Although the system and method of the present invention has been described in connection with the preferred embodiment, it is not intended to be limited to the specific form set forth herein, but on the contrary, it is intended to cover such alternatives, modifications, and equivalents, as can be reasonably included within the spirit and scope of the invention as defined by the appended claims.

What is claimed is:

1. A method for managing memory accesses in a system including a CPU, a system memory for storing data, and a memory controller coupled to the system memory, wherein the memory controller performs memory control functions for the system memory, wherein the memory controller includes a hardware compression and decompression engine, the method comprising:

- the CPU initiating an access of data in the system memory, wherein the system memory is a volatile memory which stores uncompressed data currently being used for execution by the CPU, wherein the uncompressed data includes most recently used data;
- determining a replacement block of data in the system memory after said initiating;
- the memory controller compressing said replacement block of data;
- the memory controller storing said compressed replacement block of data in said system memory after said compressing said replacement block of data;
- wherein said compressing said replacement block of data and storing said compressed replacement block of data in said system memory operates to free up at least a portion of said system memory;

23

the memory controller performing said access of data in the system memory.

2. The method of claim 1, wherein the CPU initiating an access of data in the system memory comprises the CPU initiating a read of requested data in the system memory, wherein the memory controller performing said access of data in the system memory includes:

the memory controller providing said requested data to the CPU.

3. The method of claim 2, wherein the requested data resides in the system memory in a compressed format, wherein the memory controller providing said requested data to the CPU includes:

the memory controller decompressing said requested data after the CPU initiating the access to produce uncompressed requested data; and

the memory controller storing said uncompressed requested data in the system memory.

4. The method of claim 3, further comprising: marking said uncompressed requested data as most recently used data.

5. The method of claim 2, further comprising: marking said requested data as most recently used data.

6. The method of claim 2, wherein the computer system includes a non-volatile memory coupled to the memory controller, wherein the requested data resides in the non-volatile memory, wherein the memory controller performing said access of data in the system memory comprises:

the memory controller accessing said requested data from the non-volatile memory; and

the memory controller storing said requested data in the system memory.

7. The method of claim 2, wherein the computer system includes a non-volatile memory coupled to the memory controller, wherein the requested data resides in the non-volatile memory in a compressed format, wherein the memory controller performing said access of data in the system memory comprises:

the memory controller accessing said requested data from the non-volatile memory;

the memory controller decompressing the requested data to produce uncompressed requested data; and

the memory controller storing said uncompressed requested data in the system memory.

8. The method of claim 1, further comprising: marking said data as most recently used data.

9. The method of claim 1, wherein the CPU initiating an access of data in the system memory comprises the CPU initiating a write of first data to the system memory, wherein the memory controller performing said access of data in the system memory comprises:

the memory controller writing said first data to the system memory after the memory controller compressing said replacement block of data and storing said compressed replacement block of data in said system memory.

10. The method of claim 9, further comprising: marking said first data as most recently used data.

11. The method of claim 1, wherein the CPU initiating an access of data in the system memory comprises the CPU initiating a write of first data to the system memory, wherein the memory controller performing said access of data in the system memory comprises:

the memory controller compressing the first data to produce compressed first data; and

the memory controller writing said compressed first data to the system memory after the memory controller

24

compressing said replacement block of data and storing said compressed replacement block of data in said system memory.

12. The method of claim 1,

the memory controller determining if the data resides in the system memory in an uncompressed format in response to the CPU initiating the access of data in the system memory;

wherein the memory controller compresses the replacement block of data and stores the compressed replacement block of data in the system memory in response to the memory controller determining that the data does not reside in the system memory in an uncompressed format.

13. The method of claim 12,

wherein the memory controller determining if the data resides in the system memory in an uncompressed format comprises the memory controller determining if a page hit occurs;

wherein the memory controller compresses the replacement block of data and stores the compressed replacement block of data in the system memory in response to the memory controller determining that a page miss has occurred.

14. The method of claim 1, wherein the memory controller compressing said replacement block of data comprises the memory controller performing a lossless compression on said replacement block of data.

15. The method of claim 1, wherein the memory controller compressing said replacement block of data comprises the memory controller performing a lossy compression on said replacement block of data.

16. The method of claim 1, wherein the system memory stores application data used by the CPU for executing one or more applications.

17. The method of claim 16,

wherein the CPU initiating an access of data in the system memory comprises the CPU initiating an access of application data in the system memory; and

wherein the memory controller performing said access of data in the system memory comprises the memory controller accessing the application data in the system memory.

18. The method of claim 16, wherein the replacement block of data in the system memory comprises application data.

19. The method of claim 1, wherein the computer system includes a display, wherein the system memory stores graphics data used for presenting images on the display;

wherein the CPU initiating an access of data in the system memory comprises the CPU initiating an access of graphics data in the system memory; and

wherein the memory controller performing said access of data in the system memory comprises the memory controller accessing the graphics data in the system memory.

20. The method of claim 1, wherein the computer system includes a display, wherein the system memory stores graphics data used for presenting images on the display;

wherein the replacement block of data in the system memory comprises graphics data.

21. The method of claim 1, wherein said determining a replacement block of data in the system memory comprises determining a least recently used block of data in the system memory.

22. The method of claim 1, wherein said determining a replacement block of data in the system memory comprises

25

determining a true least recently used block of data in the system memory.

23. The method of claim 1, wherein said determining a replacement block of data in the system memory comprises determining a pseudo least recently used block of data in the system memory.

24. The method of claim 1, wherein the memory controller comprises a hardware compression engine and a hardware decompression engine.

25. The method of claim 1, wherein the data includes application code and application data.

26. A method for managing memory accesses in a system a system including a CPU, a system memory for storing data, and a memory controller coupled to the system memory, wherein the memory controller performs memory control functions for the system memory, wherein the memory controller includes a hardware compression and decompression engine, the method comprising:

the CPU requesting data from the memory controller, wherein the data resides in the system memory in a compressed format, wherein the system memory is a volatile memory which stores uncompressed data currently being used for execution by the CPU, wherein the uncompressed data includes most recently used data;

determining a replacement block of data in the system memory after said requesting;

the memory controller compressing said replacement block of data;

the memory controller storing said compressed replacement block of data in said system memory after said compressing said replacement block of data;

the memory controller decompressing said requested data after said requesting to produce uncompressed requested data; and

the memory controller providing said uncompressed requested data to the CPU.

27. The method of claim 26, further comprising: marking said uncompressed requested data as most recently used data.

28. The method of claim 26, wherein the memory controller comprises a hardware compression engine and a hardware decompression engine.

29. The method of claim 26, wherein the data includes application code and application data.

30. A method for managing data accesses in a system including a CPU, a system memory for storing data, a memory controller coupled to the system memory, and a non-volatile memory coupled to the memory controller, wherein the memory controller performs memory control functions for the system memory, wherein the memory controller includes a hardware compression and decompression engine, the method comprising:

the CPU initiating an access of data in the system memory, wherein the system memory is a volatile memory which stores uncompressed data currently being used for execution by the CPU, wherein the uncompressed data includes most recently used data;

determining a replacement block of data in the system memory after said initiating;

the memory controller compressing said replacement block of data;

the memory controller transferring said compressed replacement block to the non-volatile memory for storage after said compressing said replacement block of data;

26

wherein said compressing said replacement block of data and transferring said compressed replacement block of data to the non-volatile memory operates to free up at least a portion of said system memory;

the memory controller performing said access of data in the system memory.

31. The method of claim 30, further comprising: marking said data as most recently used data.

32. The method of claim 30, wherein the CPU initiating an access of data in the system memory comprises the CPU initiating a read of requested data in the system memory, wherein the memory controller performing said access of data in the system memory includes:

the memory controller providing said requested data to the CPU.

33. The method of claim 32, further comprising: marking said requested data as most recently used data.

34. The method of claim 32, wherein the requested data resides in the system memory in a compressed format, wherein the memory controller providing said requested data to the CPU includes:

the memory controller decompressing said requested data after the CPU initiating the access to produce uncompressed requested data; and

the memory controller storing said uncompressed requested data in the system memory.

35. The method of claim 32, wherein the requested data resides in the non-volatile memory, wherein the memory controller providing said requested data to the CPU includes:

the memory controller accessing said requested data from the non-volatile memory; and

the memory controller storing said requested data in the system memory.

36. The method of claim 32, wherein the requested data resides in the non-volatile memory in a compressed format, wherein the memory controller providing said requested data to the CPU includes:

the memory controller accessing said requested data from the non-volatile memory;

the memory controller decompressing the requested data to produce uncompressed requested data; and

the memory controller storing said uncompressed requested data in the system memory.

37. The method of claim 30, wherein the CPU initiating an access of data in the system memory comprises the CPU initiating a write of first data to the system memory, wherein the memory controller performing said access of data in the system memory comprises:

the memory controller writing said first data to the system memory after the memory controller compressing said replacement block of data and storing said compressed replacement block of data in said system memory.

38. The method of claim 37, further comprising:

marking said first data as most recently used data.

39. The method of claim 30, wherein the CPU initiating an access of data in the system memory comprises the CPU initiating a write of first data to the system memory, wherein the memory controller performing said access of data in the system memory comprises:

the memory controller compressing the first data to produce compressed first data; and

the memory controller writing said compressed first data to the system memory after the memory controller

compressing said replacement block of data and storing said compressed replacement block of data in said system memory.

40. The method of claim 30, the memory controller determining if the data resides in the system memory in an uncompressed format in response to the CPU initiating the access of data in the system memory;

wherein the memory controller compresses the replacement block of data and transfers the compressed replacement block of data to the non-volatile memory in response to the memory controller determining that the data does not reside in the system memory in an uncompressed format.

41. The method of claim 40, wherein the memory controller determining if the data resides in the system memory in an uncompressed format comprises the memory controller determining if a page hit occurs;

wherein the memory controller compresses the replacement block of data and transfers the compressed replacement block of data to the non-volatile memory in response to the memory controller determining that a page miss has occurred.

42. The method of claim 30, wherein the memory controller compressing said replacement block of data comprises the memory controller performing a lossless compression on said replacement block of data.

43. The method of claim 30, wherein the memory controller compressing said replacement block of data comprises the memory controller performing a lossy compression on said replacement block of data.

44. The method of claim 30, wherein the system memory stores application data used by the CPU for executing one or more applications.

45. The method of claim 44, wherein the CPU initiating an access of data in the system memory comprises the CPU initiating an access of application data in the system memory; and wherein the memory controller performing said access of data in the system memory comprises the memory controller accessing the application data in the system memory.

46. The method of claim 44, wherein the replacement block of data in the system memory comprises application data.

47. The method of claim 30, wherein the computer system includes a display, wherein the system memory stores graphics data used for presenting images on the display;

wherein the CPU initiating an access of data in the system memory comprises the CPU initiating an access of graphics data in the system memory; and

wherein the memory controller performing said access of data in the system memory comprises the memory controller accessing the graphics data in the system memory.

48. The method of claim 30, wherein the computer system includes a display, wherein the system memory stores graphics data used for presenting images on the display;

wherein the replacement block of data in the system memory comprises graphics data.

49. The method of claim 30, wherein said determining a replacement block of data in the system memory comprises determining a least recently used block of data in the system memory.

50. The method of claim 30, wherein said determining a replacement block of data in the system memory comprises

determining a true least recently used block of data in the system memory.

51. The method of claim 30, wherein said determining a replacement block of data in the system memory comprises determining a pseudo least recently used block of data in the system memory.

52. The method of claim 30, wherein the memory controller comprises a hardware compression engine and a hardware decompression engine.

53. The method of claim 30, wherein the data includes application code and application data.

54. A method for managing memory accesses in a system including a CPU, a system memory for storing data, a memory controller coupled to the system memory, and a non-volatile memory coupled to the memory controller, wherein the memory controller performs memory control functions for the system memory, wherein the memory controller includes a hardware compression and decompression engine, the method comprising:

the CPU requesting data from the memory controller, wherein the data resides in the system memory in a compressed format, wherein the system memory is a volatile memory which stores uncompressed data currently being used for execution by the CPU, wherein the uncompressed data includes most recently used data;

determining a replacement block of data in the system memory after said requesting;

the memory controller compressing said replacement block of data;

the memory controller transferring said compressed replacement block of data to the non-volatile memory after said compressing said replacement block of data;

the memory controller decompressing said requested data after said requesting to produce uncompressed requested data; and

the memory controller providing said uncompressed requested data to the CPU.

55. The method of claim 54, further comprising: marking said uncompressed requested data as most recently used data.

56. The method of claim 54, wherein the memory controller comprises a hardware compression engine and a hardware decompression engine.

57. The method of claim 54, wherein the data includes application code and application data.

58. A system with improved memory access management, the system comprising:

a CPU;

a system memory, wherein the system memory is a volatile memory for storing data, wherein the data includes uncompressed data currently being used for execution by the CPU, wherein the uncompressed data includes most recently used data; and

a memory controller coupled to the CPU and to the system memory, wherein the memory controller performs memory control functions for the system memory, wherein the memory controller includes a hardware compression/decompression engine;

wherein the CPU is operable to initiate an access of data in the system memory;

wherein, in response to the access, the memory controller is operable to access a replacement block of data in the system memory, compress said replacement block of data, and store said compressed replacement block of

29

data in said system memory, wherein said compression of said replacement block of data and storage of said compressed replacement block of data in said system memory operates to free up at least a portion of said system memory;

wherein the memory controller is operable to perform said access of data in the system memory after freeing up said at least a portion of said system memory.

59. The system of claim 58, wherein said data is marked as most recently used data.

60. The system of claim 58, wherein the CPU is operable to initiate a read of requested data in the system memory; wherein, in performing said access of data in the system memory, the memory controller is operable to provide said requested data to the CPU.

61. The system of claim 60, wherein the requested data resides in the system memory in a compressed format; wherein, in providing said requested data to the CPU, the memory controller is operable to decompress said requested data and store said uncompressed requested data in the system memory.

62. The system of claim 60, wherein the system further includes:

a non-volatile memory coupled to the memory controller, wherein the requested data resides in the non-volatile memory;

wherein, in providing said requested data to the CPU, the memory controller is operable to access said requested data from the non-volatile memory and store said requested data in the system memory.

63. The system of claim 60, wherein the system further includes:

a non-volatile memory coupled to the memory controller, wherein the requested data resides in the non-volatile memory in a compressed format;

wherein, in providing said requested data to the CPU, the memory controller is operable to access said requested data from the non-volatile memory, decompress the requested data to produce uncompressed requested data, and store said uncompressed requested data in the system memory.

64. The system of claim 58, wherein the CPU is operable to initiate a write of first data to the system memory; wherein, in performing said access of data in the system memory, the memory controller is operable to write said first data to the system memory.

65. The system of claim 58, wherein the CPU is operable to initiate a write of first data to the system memory; wherein, in performing said access of data in the system memory, the memory controller is operable to compress the first data to produce compressed first data and write said compressed first data to the system memory.

66. The system of claim 58, wherein, in response to the access, the memory controller is operable to determine if the data resides in the system memory in an uncompressed format;

wherein the memory controller is operable to access the replacement block of data, compress the replacement block of data, and store the compressed replacement block of data in the system memory in response to the memory controller determining that the data does not reside in the system memory in an uncompressed format.

67. The system of claim 66, wherein the memory controller is operable to provide the data to the CPU in response to the memory controller

30

determining that the data resides in the system memory in an uncompressed format.

68. The system of claim 66, wherein, in determining if the data resides in the system memory in an uncompressed format, the memory controller is operable to determine if a page hit occurs;

wherein the memory controller accesses the replacement block of data, compresses the replacement block of data, and stores the compressed replacement block of data in the system memory in response to the memory controller determining that a page miss has occurred.

69. The system of claim 58, wherein the compression/decompression engine comprised in the memory controller is operable to perform a lossless compression on said replacement block of data.

70. The system of claim 58, wherein the compression/decompression engine comprised in the memory controller is operable to perform a lossy compression on said replacement block of data.

71. The system of claim 58, wherein the system memory stores application data used by the CPU for executing one or more applications;

wherein the data comprises application data.

72. The system of claim 58, wherein the system further includes a display;

wherein the system memory stores graphics data used for presenting images on the display;

wherein the data comprises graphics data.

73. The system of claim 58, wherein the replacement block of data comprises a least recently used block of data in the system memory.

74. The system of claim 58, wherein the replacement block of data comprises a true least recently used block of data in the system memory.

75. The system of claim 58, wherein the replacement block of data comprises a pseudo least recently used block of data in the system memory.

76. The system of claim 58, wherein the system comprises a computer system.

77. The system of claim 58, wherein the memory controller comprises a hardware compression engine and a hardware decompression engine.

78. The system of claim 58, wherein the data includes application code and application data.

79. A system with improved memory access management, the system comprising:

a CPU;

a system memory, wherein the system memory is a volatile memory for storing data, wherein the data includes uncompressed data currently being used for execution by the CPU, wherein the uncompressed data includes most recently used data;

a memory controller coupled to the CPU and to the system memory, wherein the memory controller performs memory control functions for the system memory, wherein the memory controller includes a hardware compression/decompression engine; and

a non-volatile memory coupled to the memory controller; wherein the CPU is operable to initiate an access of data in the system memory;

wherein, in response to the access, the memory controller is operable to access a replacement block of data in the system memory, compress said replacement block of data, and transfer said compressed replacement block of data to the non-volatile memory, wherein said com-

31

pression of said replacement block of data and transfer of said compressed replacement block of data to the non-volatile memory operates to free up at least a portion of said system memory;

wherein the memory controller is operable to perform said access of data in the system memory after freeing up said at least a portion of said system memory.

80. The system of claim 79,

wherein said data is marked as most recently used data.

81. The system of claim 79, wherein the CPU is operable to initiate a read of requested data in the system memory;

wherein, in performing said access of data in the system memory, the memory controller is operable to provide said requested data to the CPU.

82. The system of claim 81, wherein the requested data resides in the system memory in a compressed format;

wherein, in providing said requested data to the CPU, the memory controller is operable to decompress said requested data and store said uncompressed requested data in the system memory.

83. The system of claim 81,

wherein the requested data resides in the non-volatile memory;

wherein, in providing said requested data to the CPU, the memory controller is operable to access said requested data from the non-volatile memory and store said requested data in the system memory.

84. The system of claim 81,

wherein the requested data resides in the non-volatile memory in a compressed format;

wherein, in providing said requested data to the CPU, the memory controller is operable to access said requested data from the non-volatile memory, decompress the requested data to produce uncompressed requested data, and store said uncompressed requested data in the system memory.

85. The system of claim 79, wherein the CPU is operable to initiate a write of first data to the system memory;

wherein, in performing said access of data in the system memory, the memory controller is operable to write said first data to the system memory.

86. The system of claim 79, wherein the CPU is operable to initiate a write of first data to the system memory;

wherein, in performing said access of data in the system memory, the memory controller is operable to compress the first data to produce compressed first data and write said compressed first data to the system memory.

32

87. The system of claim 79,

wherein, in response to the access, the memory controller is operable to determine if the data resides in the system memory in an uncompressed format;

wherein the memory controller is operable to access the replacement block of data, compress the replacement block of data, and transfer the compressed replacement block of data to the non-volatile memory in response to the memory controller determining that the data does not reside in the system memory in an uncompressed format.

88. The system of claim 79, wherein the compression/decompression engine comprised in the memory controller is operable to perform a lossless compression on said replacement block of data.

89. The system of claim 79, wherein the compression/decompression engine comprised in the memory controller is operable to perform a lossy compression on said replacement block of data.

90. The system of claim 79, wherein the system memory stores application data used by the CPU for executing one or more applications;

wherein the data comprises application data.

91. The system of claim 79, wherein the system further includes a display;

wherein the system memory stores graphics data used for presenting images on the display;

wherein the data comprises graphics data.

92. The system of claim 79, wherein the replacement block of data comprises a least recently used block of data in the system memory.

93. The system of claim 79, wherein the replacement block of data comprises a true least recently used block of data in the system memory.

94. The system of claim 79, wherein the replacement block of data comprises a pseudo least recently used block of data in the system memory.

95. The system of claim 79, wherein the system comprises a computer system.

96. The system of claim 79, wherein the memory controller comprises a hardware compression engine and a hardware decompression engine.

97. The system of claim 79, wherein the data includes application code and application data.

* * * * *



US006539456B2

(12) **United States Patent**
Stewart

(10) Patent No.: **US 6,539,456 B2**

(45) Date of Patent: ***Mar. 25, 2003**

(54) **HARDWARE ACCELERATION OF BOOT-UP UTILIZING A NON-VOLATILE DISK CACHE**

(75) Inventor: **David C. Stewart, Beaverton, OR (US)**

(73) Assignee: **Intel Corporation, Santa Clara, CA (US)**

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/417,000**

(22) Filed: **Oct. 13, 1999**

(65) **Prior Publication Data**

US 2002/0156970 A1 Oct. 24, 2002

(51) Int. Cl.⁷ **G06F 12/00; G06F 9/24**

(52) U.S. Cl. **711/113; 711/133; 711/144; 713/2**

(58) Field of Search **711/113, 133, 711/144; 713/2**

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,307,497 A	*	4/1994	Feigenbaum et al.	713/1
6,061,788 A	*	5/2000	Reynaud et al.	713/2
6,073,232 A	*	6/2000	Kroeker et al.	713/1
6,172,936 B1	*	1/2001	Kitazaki	365/233
6,189,100 B1	*	2/2001	Barr et al.	713/182
6,226,740 B1	*	5/2001	Iga	713/2

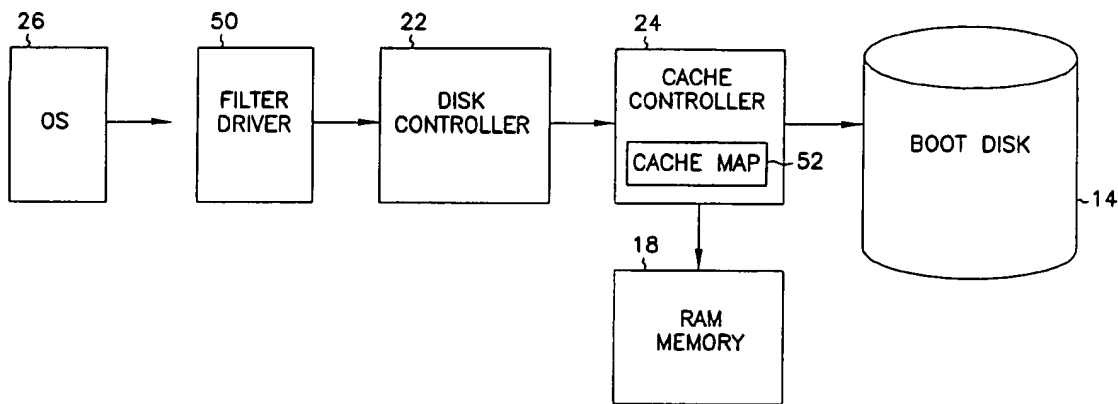
* cited by examiner

Primary Examiner—Reginald G. Bragdon
(74) *Attorney, Agent, or Firm*—Schwegman, Lundberg, Woessner & Kluth, P.A.

(57) **ABSTRACT**

A computer system includes a nonvolatile memory positioned between a disk controller and a disk drive storing a boot program, in a computer system. Upon an initial boot sequence, the boot program is loaded into a cache in the nonvolatile memory. Subsequent boot sequences retrieve the boot program from the cache. Cache validity is maintained by monitoring cache misses, and/or by monitoring writes to the disk such that a write to a sector held in the cache results in the cache line for that sector being invalidated until such time as the cache is updated. A filter driver is provided to monitor writes to the disk and determine if a cache line is invalidated.

38 Claims, 5 Drawing Sheets



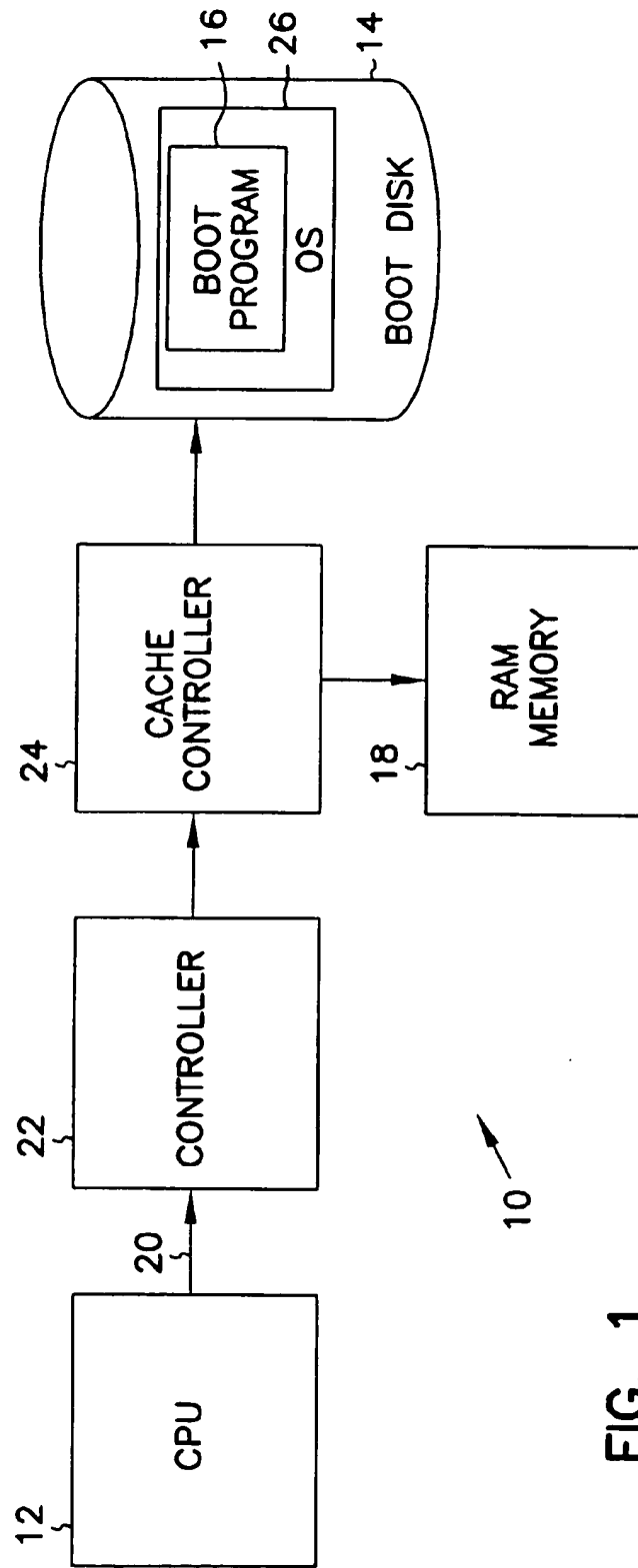
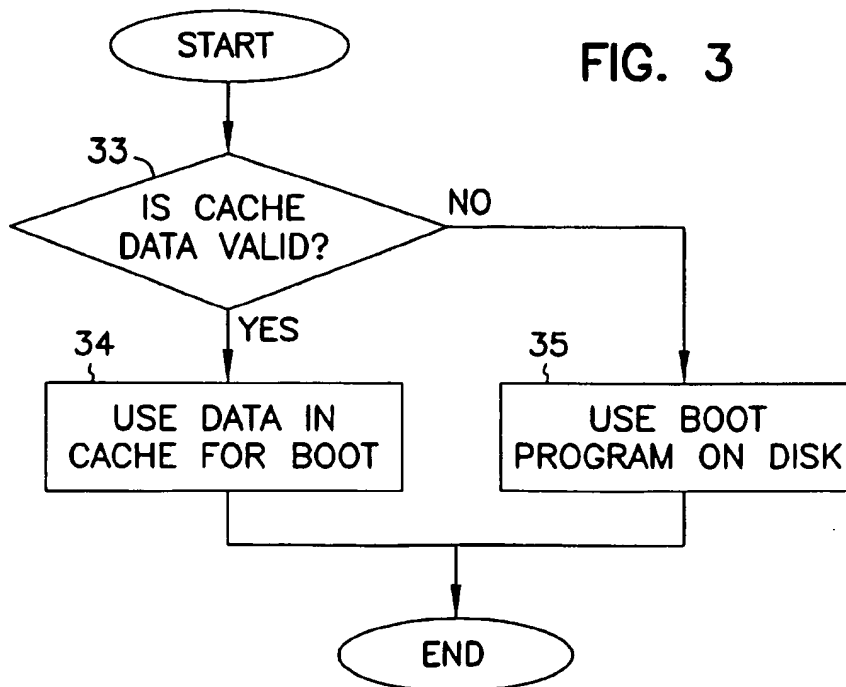
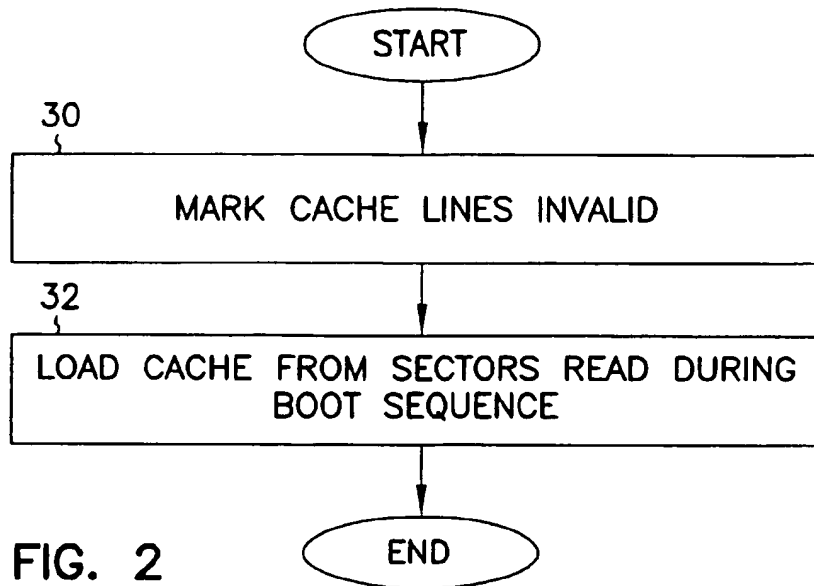


FIG. 1



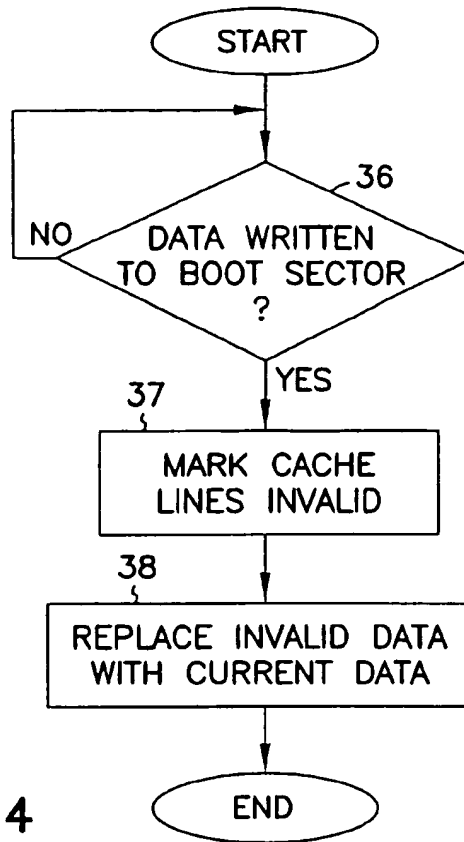


FIG. 4

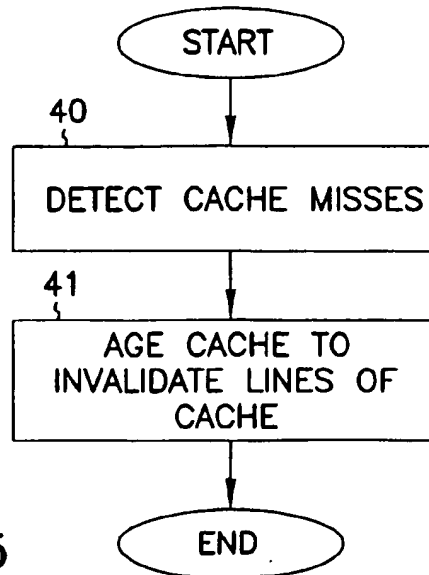


FIG. 5

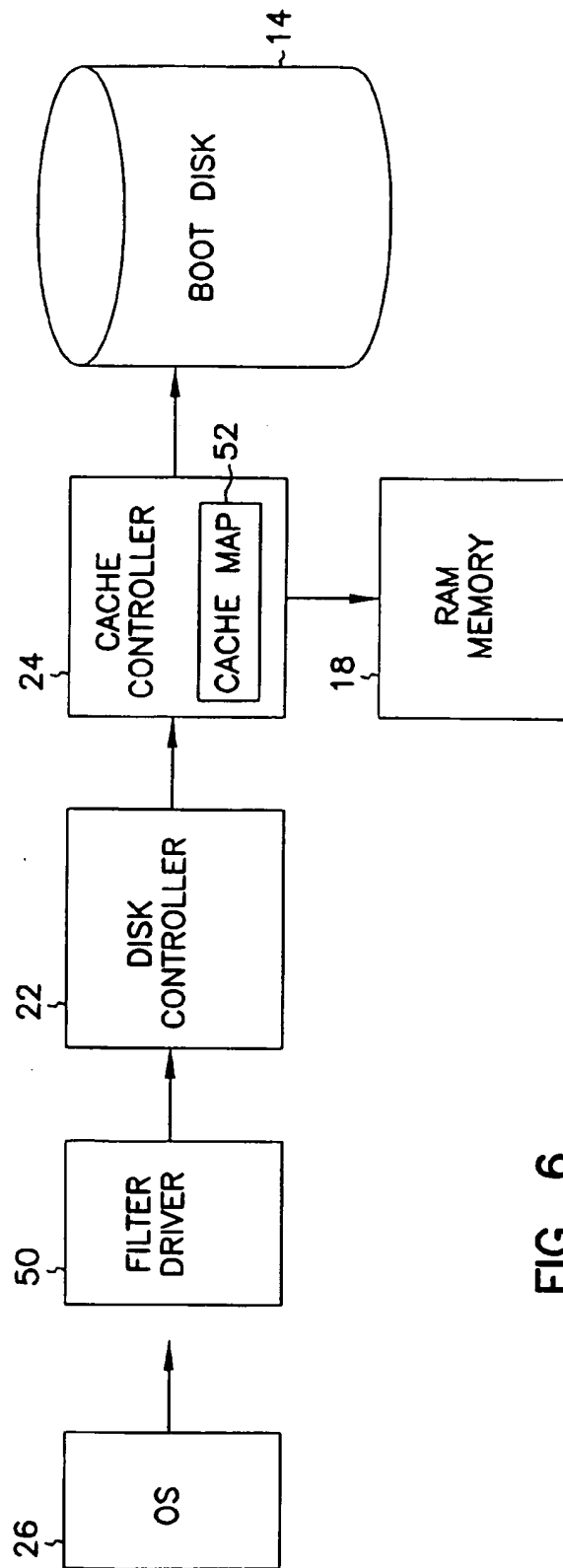


FIG. 6

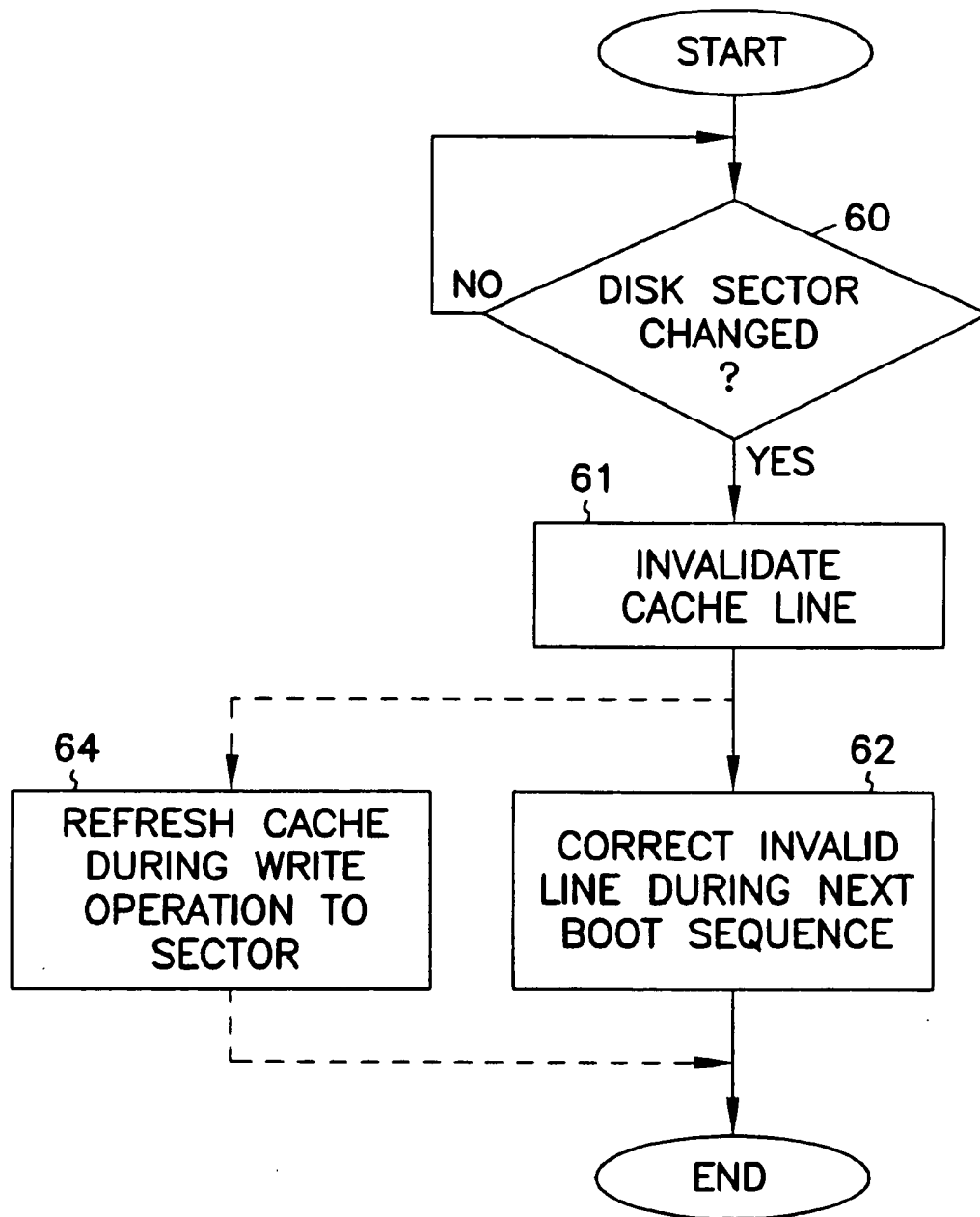


FIG. 7

1

HARDWARE ACCELERATION OF BOOT-UP UTILIZING A NON-VOLATILE DISK CACHE

TECHNICAL FIELD OF THE INVENTION

The present invention pertains generally to computers, and more particularly to method and apparatus for speeding the boot-up process in computers.

BACKGROUND OF THE INVENTION

Booting up a computer, and in particular an IBM-compatible personal computer (PC), often takes longer than desired. For example, it is not atypical for a PC using the Windows® 98 operating system to require one minute or more to boot up. This delay can be untenable when the PC needs to be activated on an expedited basis. For instance, if the user needs a phone number quickly, it can be more expeditious to look the number up in a telephone directory as opposed to a PC if the PC requires booting. Thus, unless PC's can be booted more quickly than as is currently the case, their use in applications that require fast initialization is limited. Thus, there is a need for a PC with a shorter boot up time than is currently available.

SUMMARY OF THE INVENTION

The present invention provides method and apparatus for speeding the boot-up of a computer. According to one embodiment of the invention, a boot program stored on a boot disk is cached in a nonvolatile memory, and retrieved by the system from the cache during the boot sequence instead of from the boot disk, thereby increasing the speed of access to the boot program. This and various other embodiments of the invention are described below.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates a first embodiment of the apparatus of the invention.

FIGS. 2-5 illustrate various alternate embodiments of the method of using the cache according to the present invention.

FIG. 6 illustrates an alternate embodiment of the apparatus of the invention.

FIG. 7 illustrates yet another embodiment of the method of the invention.

DETAILED DESCRIPTION OF THE INVENTION

In the following detailed description of the invention reference is made to the accompanying drawings which form a part hereof, and in which is shown, by way of illustration, specific embodiments in which the invention may be practiced. In the drawings, like numerals describe substantially similar components throughout the several views. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention. Other embodiments may be utilized and structural, logical, and electrical changes may be made without departing from the scope of the present invention.

Referring now to FIG. 1, there is shown a first embodiment of the invention. A computer system 10 includes a Central Processing Unit (CPU) 12, a boot disk 14 storing a boot program 16 used by the computer system 10 to boot, and a nonvolatile random access memory 18 used as a disk cache. Memory 18 receives all or a portion of the boot

2

program 16 from the boot disk 14 and stores it for access by the CPU 12 so that the computer system 10 can boot in whole or in part from the disk cache in memory 18. A data bus 20 couples the CPU 12 to a controller 22 that controls the boot disk 14, and a cache controller 24 is coupled between the bus 20 and the boot disk 14, and wherein the memory 18 is coupled to the cache controller 22. In one example embodiment, the computer system 10 may comprise an IBM-compatible computer with a Pentium class microprocessor and an IDE controller for controller 22, or an Apple Macintosh computer with a Motorola microprocessor. The invention, however, is not limited in this respect, and other types of computer systems and processors can be used. Nonvolatile memory 18 may be a FLASH memory, or any suitable form of nonvolatile memory, and, preferably in at least some embodiments of the invention, random access memory.

In operation, the computer system 10 operates under the control of an operating system 26, which includes as a portion thereof boot program 16. Boot program 16 has a boot-time disk footprint of an ascertainable size. The memory 18 is sized to be substantially as large as the boot-time disk footprint, so that the boot program 16 can be cached in the memory 18. However, the memory 18 could be smaller than the footprint, and store only a portion of the entire boot program 16. Alternatively, memory 18 could exceed the size of program 16. All or a portion of boot program 16 can therefore be stored in memory 18, from where it can be more quickly retrieved, as opposed to being retrieved from the boot disk 14, during boot-up of the system 10. If only a portion of the boot program 16 is stored in memory 18, that portion may be retrieved therefrom, with the remaining portion retrieved from the boot disk 14.

According to another example embodiment, the boot program cache in memory 18 is formed of lines, the boot program 16 is stored in linear sectors on the boot disk 14, and the lines of the cache are mapped to the linear sectors of the boot disk 14 read in a boot sequence upon boot up of system 10. Referring to FIGS. 2-5, there is shown an example method for using the boot program cache. Initially, the cache lines are marked invalid (30). The cache is loaded with data from sectors of disk 14 read during an initial boot sequence (32). As shown in FIG. 3, during boots of the system 10 subsequent to the initial boot sequence, data in the cache is used (34) instead of the corresponding sector data from the boot disk, if the sector data in the cache is valid (33). Otherwise, the boot program or the disk is used (35). According to another example variant of this embodiment shown in FIG. 4, if data is written to a sector read during the initial boot sequence (36), the cache lines corresponding to the sector are marked invalid (37). The invalid cache line can be subsequently replaced with new data from the boot disk and the cache line marked valid (38). According to yet another example embodiment of the method of the present invention, illustrated in FIG. 5, cache coherency is maintained by detecting cache misses (40), and if a miss is detected, aging the cache, to invalidate lines from the cache (41). According to one approach, the cache is aged in a first-in first-out (FIFO) manner.

According to yet another embodiment of the invention diagrammatically illustrated in FIG. 6, a filter driver 50 is positioned between the operating system 26 and the disk controller 22, and the filter driver 50 has access to all input-output (I/O) requests to the boot disk 14, and to a cache map 52 in cache controller 24. Filter driver 50 can detect writes to the disk 14 which are in the same sector as a sector in the cache. In one embodiment, filter driver 50 can

3

monitor all I/O operations without significantly slowing performance of the system.

According to a method of operation using the embodiment of FIG. 6, illustrated in FIG. 7, if a disk sector cached in the cache is changed (60), as detected by filter driver 50, the corresponding cache line is invalidated (61). The invalidated line can be refreshed with the correct contents during the next boot sequence (62). In one embodiment, the cache is not updated by the filter driver so that performance is not degraded. However, according to another embodiment, the cache is refreshed during the write operation to the corresponding sector in the disk drive (64) using a cache write-back queue.

Thus, as described above, there is provided method and apparatus for speeding the boot-up of a computer. The invention is applicable to all manner of computer systems, including appliance-like, sealed case systems, where the loadable files and configuration are seldom changed.

What is claimed is:

1. A computer system comprising a CPU, a boot disk storing a boot program used by the computer system to boot, and a nonvolatile memory disk cache receiving all or a portion of the boot program from the boot disk and storing it for access by the CPU so that the computer system can boot in whole or in part from the disk cache, wherein the computer system further includes an IDE controller for controlling the boot disk, wherein the cache has lines, wherein the lines of the cache are mapped to linear sectors read in a boot sequence, wherein the cache lines are initially marked invalid, wherein the cache is loaded with data from sectors read during an initial boot sequence, wherein during boots of the system subsequent to the initial boot sequence sector data in the cache is used instead of the corresponding sector data from the boot disk if the sector data in the cache is valid, and wherein if data is written to a sector read during the initial boot sequence, the cache line corresponding to the sector is marked invalid.

2. The computer system of claim 1, wherein the invalid cache line is replaced with new data from the boot disk and the cache line marked valid.

3. A computer system comprising a CPU, a boot disk storing a boot program used by the computer system to boot, and a nonvolatile memory disk cache receiving all or a portion of the boot program from the boot disk and storing it for access by the CPU so that the computer system can boot in whole or in part from the disk cache, wherein the computer system further includes an IDE controller for controlling the boot disk, wherein the cache has lines, wherein the lines of the cache are mapped to linear sectors read in a boot sequence, wherein the cache lines are initially marked invalid, wherein the cache is loaded with data from sectors read during an initial boot sequence, wherein during boots of the system subsequent to the initial boot sequence sector data in the cache is used instead of the corresponding sector data from the boot disk if the sector data in the cache is valid, and wherein cache coherency is maintained by detecting cache misses, and if a miss is detected, the cache is aged, to invalidate lines from the cache.

4. The computer system of claim 3, wherein the cache is aged in a first-in first-out (FIFO) manner.

5. A computer system comprising a CPU, a boot disk storing a boot program used by the computer system to boot, and a nonvolatile memory disk cache receiving all or a portion of the boot program from the boot disk and storing it for access by the CPU so that the computer system can boot in whole or in part from the disk cache, wherein the computer system further includes an IDE controller for

4

controlling the boot disk, wherein the cache has lines, wherein the lines of the cache are mapped to linear sectors read in a boot sequence, wherein the cache lines are initially marked invalid, wherein the cache is loaded with data from sectors read during an initial boot sequence, wherein during boots of the system subsequent to the initial boot sequence sector data in the cache is used instead of the corresponding sector data from the boot disk if the sector data in the cache is valid, and wherein the computer system further includes a filter driver between the CPU and the IDE controller, wherein the filter driver has access to all input-output (I/O) requests to the boot disk, and wherein the filter driver has access to a cache map and can detect writes to the disk which are in the same sector as a sector in the cache.

6. The computer system of claim 5, further wherein if such a sector is changed, the corresponding cache line is invalidated, and refreshed with the correct contents during the next boot sequence.

7. The computer system of claim 5, wherein the cache is not updated by the filter driver.

8. The computer system of claim 5, wherein the cache is refreshed during the write operation to the corresponding sector in the disk drive.

9. A method comprising storing a boot program used by a computer system in a nonvolatile memory disk cache which receives all or a portion of the boot program from a system boot disk, the boot program stored in the cache for access by the CPU so that the computer system can boot in whole or in part from the disk cache, wherein an IDE controller is used to control the boot disk, wherein the cache is organized in lines, and wherein the lines of the cache are mapped to linear sectors read in a boot sequence, wherein the cache lines are initially marked invalid, wherein the cache is loaded with data from sectors read during an initial boot sequence, wherein during boots of the system subsequent to the initial boot sequence sector data in the cache is used instead of the corresponding sector data from the boot disk if the sector data in the cache is valid, and wherein if data is written to a sector read during the initial boot sequence, the cache line corresponding to the sector is marked invalid.

10. The method of claim 9, wherein the invalid cache line is replaced with new data from the boot disk and the cache line marked valid.

11. A method comprising storing a boot program used by a computer system in a nonvolatile memory disk cache which receives all or a portion of the boot program from a system boot disk, the boot program stored in the cache for access by the CPU so that the computer system can boot in whole or in part from the disk cache, wherein an IDE controller is used to control the boot disk, wherein the cache is organized in lines, and wherein the lines of the cache are mapped to linear sectors read in a boot sequence, wherein the cache lines are initially marked invalid, wherein the cache is loaded with data from sectors read during an initial boot sequence, wherein during boots of the system subsequent to the initial boot sequence sector data in the cache is used instead of the corresponding sector data from the boot disk if the sector data in the cache is valid, and wherein cache coherency is maintained by detecting cache misses, and if a miss is detected, the cache is aged, to invalidate lines from the cache.

12. The method of claim 11, wherein the cache is aged in a first-in first-out (FIFO) manner.

13. A method comprising storing a boot program used by a computer system in a nonvolatile memory disk cache which receives all or a portion of the boot program from a

5

system boot disk, the boot program stored in the cache for access by the CPU so that the computer system can boot in whole or in part from the disk cache, wherein an IDE controller is used to control the boot disk, wherein the cache is organized in lines, and wherein the lines of the cache are mapped to linear sectors read in a boot sequence, wherein the cache lines are initially marked invalid, wherein the cache is loaded with data from sectors read during an initial boot sequence, wherein during boots of the system subsequent to the initial boot sequence sector data in the cache is used instead of the corresponding sector data from the boot disk if the sector data in the cache is valid, and further wherein a filter driver is positioned between the CPU and the IDE controller, and wherein the filter driver has access to all input-output (I/O) requests to the boot disk, and wherein the filter driver has access to a cache map and can detect writes to the disk which are in the same sector as a sector in the cache.

14. The method of claim 13, and further wherein if such a sector is changed, the corresponding cache line is invalidated, and refreshed with the correct contents during the next boot sequence.

15. The method of claim 13, wherein the cache is not updated by the filter driver.

16. The method of claim 13, wherein the cache is refreshed during the write operation to the corresponding sector in the disk drive.

17. A computer system, comprising:

a CPU;

a boot disk storing a boot program used by the computer system to boot;

a nonvolatile memory disk cache receiving all or a portion of the boot program from the boot disk and storing it for access by the CPU, so that the computer system can boot in whole or in part from the disk cache; and

a controller for controlling the boot disk, wherein the cache has lines that are initially marked invalid, and are mapped to linear sectors read in a boot sequence,

wherein the cache is loaded with data from sectors read during an initial boot sequence,

wherein during boots of the system subsequent to the initial boot, sequence sector data in the cache is used instead of the corresponding sector data from the boot disk if the sector data in the cache is valid, and

wherein if data is written to a sector read during the initial boot sequence, the cache line corresponding to the sector is marked invalid.

18. The computer system of claim 17, further comprising:

a data bus coupled to the CPU; and

a cache controller coupled between the bus and the boot disk,

wherein the disk cache is coupled to the cache controller.

19. The computer system of claim 17, wherein the computer system operates under the control of an operating system, and wherein the operating system has a boot-time disk footprint size and the cache is sized substantially as large as the size of the footprint.

20. The computer system of claim 17, wherein the invalid cache line is replaced with new data from the boot disk and the cache line marked valid.

21. A computer system, comprising:

a CPU;

a boot disk storing a boot program used by the computer system to boot;

6

a nonvolatile memory disk cache receiving all or a portion of the boot program from the boot disk and storing it for access by the CPU, so that the computer system can boot in whole or in part from the disk cache; and

a controller for controlling the boot disk,

wherein the cache has lines that are initially marked invalid, and are mapped to linear sectors read in a boot sequence,

wherein the cache is loaded with data from sectors read during an initial boot sequence,

wherein during boots of the system subsequent to the initial boot, sequence sector data in the cache is used instead of the corresponding sector data from the boot disk if the sector data in the cache is valid, and

wherein cache coherency is maintained by detecting cache misses, and if a miss is detected, the cache is aged, to invalidate lines from the cache.

22. The computer system of claim 21, wherein the cache is aged in a first-in first-out (FIFO) manner.

23. The computer system of claim 21, further comprising:

a data bus coupled to the CPU; and

a cache controller coupled between the bus and the boot disk,

wherein the disk cache is coupled to the cache controller.

24. The computer system of claim 21, wherein the computer system operates under the control of an operating system, and wherein the operating system has a boot-time disk footprint size and the cache is sized substantially as large as the size of the footprint.

25. A computer system, comprising:

a CPU;

a boot disk storing a boot program used by the computer system to boot;

a nonvolatile memory disk cache receiving all or a portion of the boot program from the boot disk and storing it for access by the CPU, so that the computer system can boot in whole or in part from the disk cache;

a controller for controlling the boot disk; and

a filter driver between the CPU and the controller, wherein the cache has lines that are initially marked invalid, and are mapped to linear sectors read in a boot sequence,

wherein the cache is loaded with data from sectors read during an initial boot sequence,

wherein during boots of the system subsequent to the initial boot, sequence sector data in the cache is used instead of the corresponding sector data from the boot disk if the sector data in the cache is valid,

wherein the filter driver has access to all input-output (I/O) requests to the boot disk, and

wherein the filter driver has access to a cache map and can detect writes to the disk which are in the same sector as a sector in the cache.

26. The computer system of claim 25, further wherein if such a sector is changed, the corresponding cache line is invalidated, and refreshed with the correct contents during the next boot sequence.

27. The computer system of claim 25, wherein the cache is not updated by the filter driver.

28. The computer system of claim 25, wherein the cache is refreshed during the write operation to the corresponding sector in the disk drive.

29. A method comprising:

storing a boot program used by a computer system in a nonvolatile memory disk cache that receives all or a

7

portion of the boot program from a system boot disk, the boot program stored in the cache for access by a CPU so that the computer system can boot in whole or in part from the disk cache;

controlling the boot disk with a controller;

organizing the cache in lines;

initially marking the lines of the cache as invalid;

mapping the lines of the cache to linear sectors read in a boot sequence;

loading the cache with data from sectors read during an initial boot sequence;

using sequence sector data in the cache instead of the corresponding sector data from the boot disk during boots of the system subsequent to the initial boot if the sector data in the cache is valid; and

marking the cache line corresponding to a sector as invalid if data is written to the sector read during the initial boot sequence.

30. The method of claim 29, further comprising providing a data bus that couples the CPU to a cache controller coupled between the bus and the boot disk, wherein the disk cache is coupled to the cache controller.

31. The method of claim 29, further comprising:

replacing the invalid cache line with new data from the boot disk; and

marking the cache line as valid.

32. A method comprising:

storing a boot program used by a computer system in a nonvolatile memory disk cache that receives all or a portion of the boot program from a system boot disk, the boot program stored in the cache for access by a CPU so that the computer system can boot in whole or in part from the disk cache;

controlling the boot disk with a controller;

organizing the cache in lines;

initially marking the lines of the cache as invalid;

mapping the lines of the cache to linear sectors read in a boot sequence;

loading the cache with data from sectors read during an initial boot sequence;

using sequence sector data in the cache instead of the corresponding sector data from the boot disk during boots of the system subsequent to the initial boot if the sector data in the cache is valid;

maintaining cache coherency by detecting cache misses; and

8

if a miss is detected, aging the cache to invalidate lines from the cache.

33. The method of claim 32, further comprising providing a data bus that couples the CPU to a cache controller coupled between the bus and the boot disk, wherein the disk cache is coupled to the cache controller.

34. The method of claim 32, wherein the aging of the cache includes aging the cache in a first-in first-out (FIFO) manner.

35. A method comprising:

storing a boot program used by a computer system in a nonvolatile memory disk cache that receives all or a portion of the boot program from a system boot disk, the boot program stored in the cache for access by a CPU so that the computer system can boot in whole or in part from the disk cache;

controlling the boot disk with a controller;

organizing the cache in lines;

initially marking the lines of the cache as invalid;

mapping the lines of the cache to linear sectors read in a boot sequence;

loading the cache with data from sectors read during an initial boot sequence;

using sequence sector data in the cache instead of the corresponding sector data from the boot disk during boots of the system subsequent to the initial boot if the sector data in the cache is valid; and

positioning a filter driver between the CPU and the controller, wherein the filter driver has access to all input-output (I/O) requests to the boot disk, and wherein the filter driver has access to a cache map and can detect writes to the disk which are in the same sector as a sector in the cache.

36. The method of claim 35, and further wherein the cache is not updated by the filter driver.

37. The method of claim 35, further comprising:

invalidating a cache line if a sector mapped to the cache line is changed; and

refreshing the cache line with correct contents during a next boot sequence.

38. The method of claim 35, further comprising refreshing the cache during a write operation to a corresponding sector in the disk.

* * * * *



US006226740B1

(12) **United States Patent**
Iga

(10) **Patent No.:** **US 6,226,740 B1**

(45) **Date of Patent:** **May 1, 2001**

(54) **INFORMATION PROCESSING APPARATUS
AND METHOD THAT USES FIRST AND
SECOND POWER SUPPLIES FOR
REDUCING BOOTING TIME**

6,061,788 * 5/2000 Reynaud et al. 713/2
6,115,814 * 9/2000 Lieber et al. 713/2

FOREIGN PATENT DOCUMENTS

62-242257 10/1987 (JP).
3-14153 1/1991 (JP).
4-170647 6/1992 (JP).
8-161176 6/1996 (JP).

(75) **Inventor:** **Kazuhisa Iga, Tokyo (JP)**

(73) **Assignee:** **NEC Corporation, Tokyo (JP)**

(*) **Notice:** Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

* cited by examiner

Primary Examiner—Thomas M. Heckler

(74) *Attorney, Agent, or Firm*—Foley & Lardner

(21) **Appl. No.:** **09/216,610**

(22) **Filed:** **Dec. 21, 1998**

(30) **Foreign Application Priority Data**

Dec. 19, 1997 (JP) 9-350771

(51) **Int. Cl.⁷** **G06F 9/445**

(52) **U.S. Cl.** **713/2**

(58) **Field of Search** **713/2**

(57) **ABSTRACT**

A host controller performs a centralized controlled over the operation of an information processing apparatus as a whole. Codes which are read out for booting the information processing apparatus are stored in a ROM in advance. The host controller copies the contents of the ROM into a high speed storage element and reads out the storage contents of the high speed storage element to boot the apparatus when a V_{CC} power supply is turned on after the V_{CC} power supply is once turned off. This allows the apparatus to be booted based on the contents read out from the high speed storage element which can be read much faster than the ROM.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,568,641 * 10/1996 Nelson et al. 713/2

3 Claims, 2 Drawing Sheets

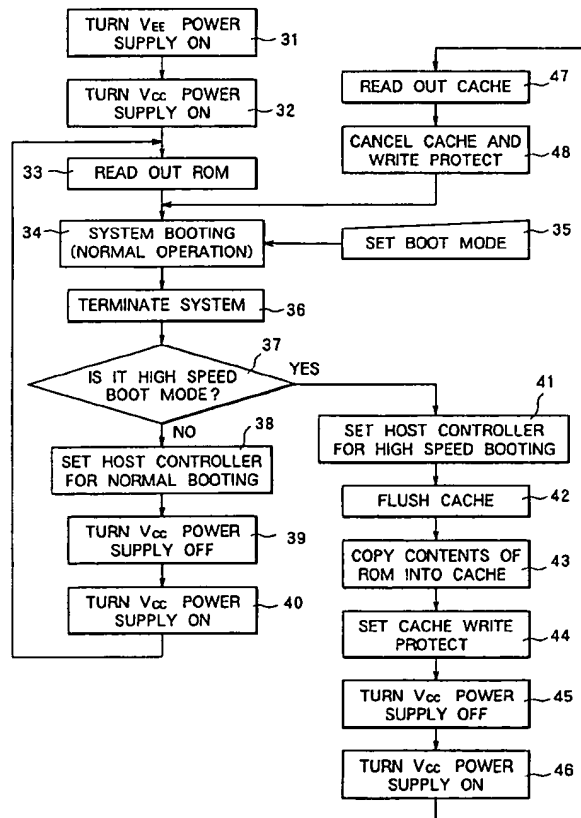


FIG. 1

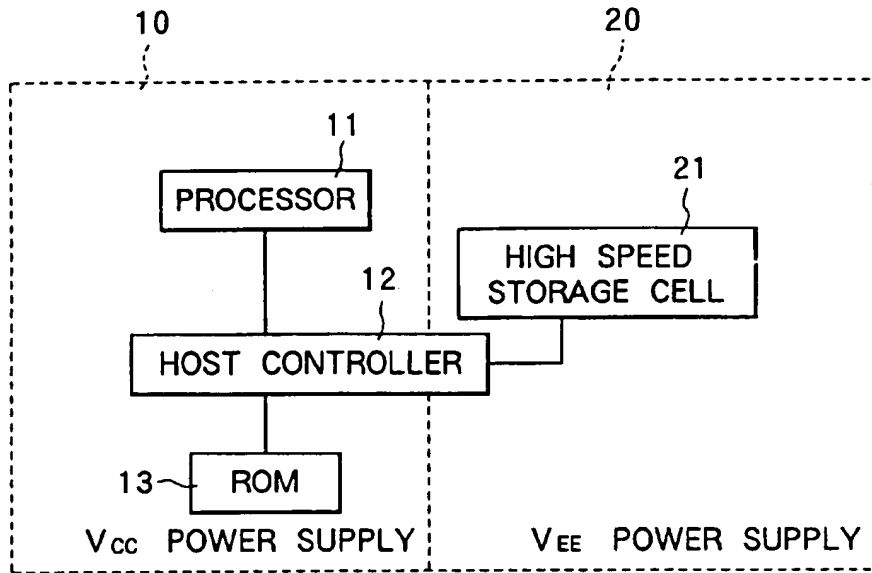


FIG. 2

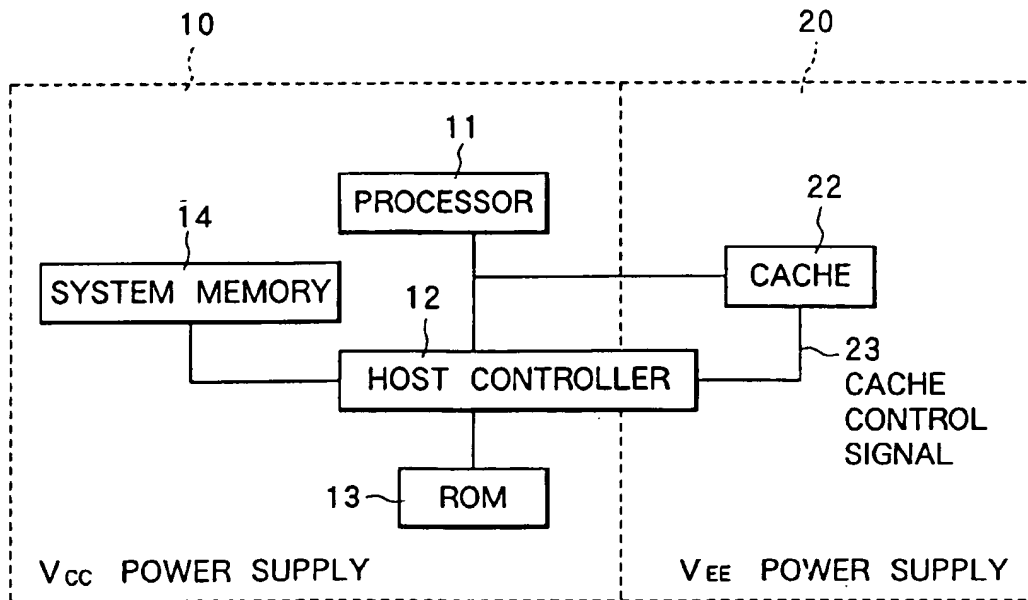
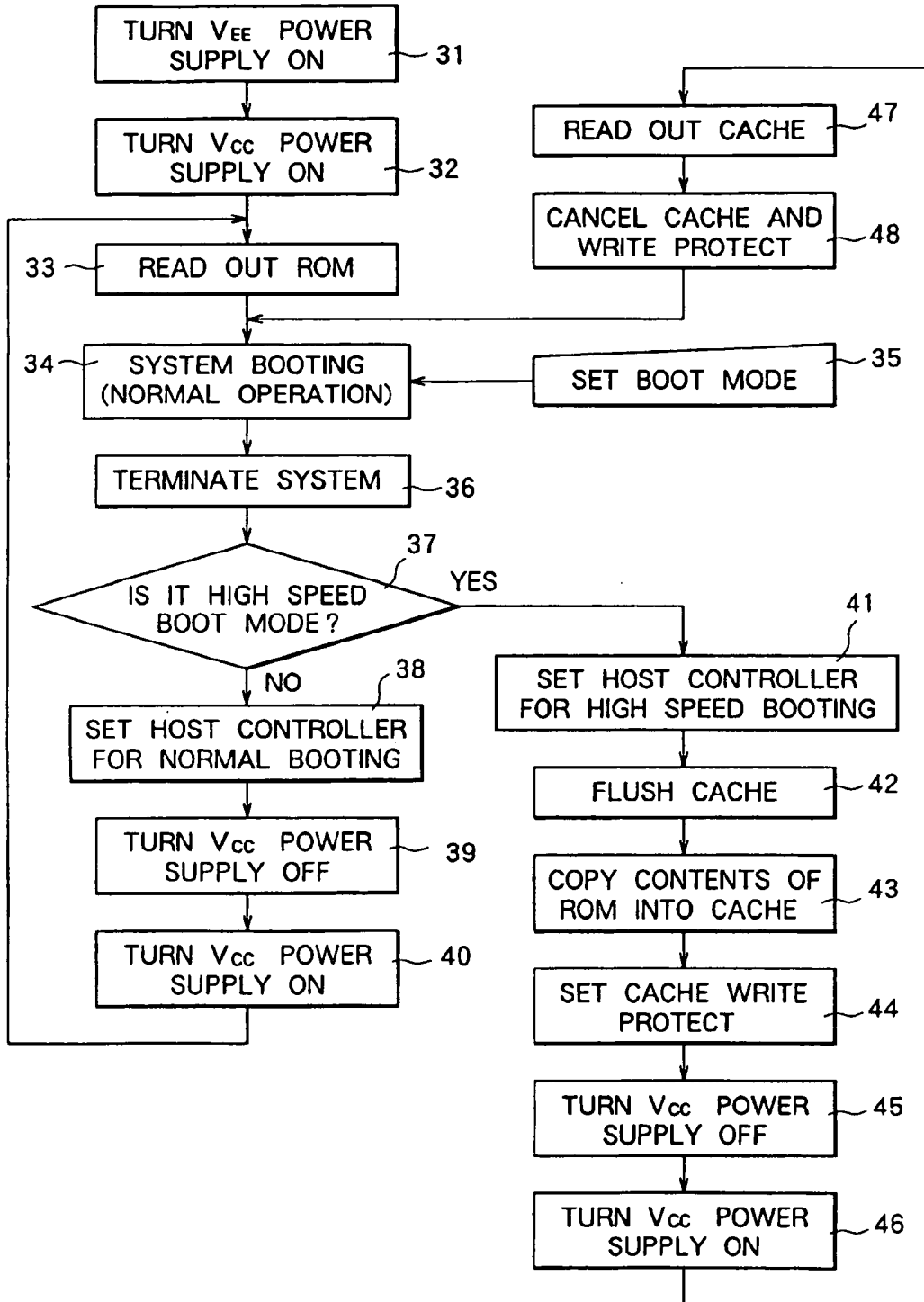


FIG. 3



1

**INFORMATION PROCESSING APPARATUS
AND METHOD THAT USES FIRST AND
SECOND POWER SUPPLIES FOR
REDUCING BOOTING TIME**

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to an information processing apparatus and a method for booting the same. More particularly, the present invention relates to an information processing apparatus having a first power supply for supplying power only in a normal operating state and a second power supply for supplying power not only in a normal operating state but also in a power supply off state and a method for booting the same.

2. Description of the Related Art

In some information processing apparatuses such as personal computers, codes for a basic input/output system (BIOS) to serve as an interface between the operating system(OS) and the hardware are stored in a read only memory (ROM). In such an information processing apparatus, the BIOS codes are read from the ROM when the apparatus is booted.

Such a conventional information processing apparatus has a problem in that a long time is spent before the apparatus is booted because the BIOS codes stored in the ROM are read at a low speed and hence the user must spend wasteful time until the apparatus is booted.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide an information processing apparatus which can be booted at a high speed and a method for booting the same.

It is another object of the invention to provide an information processing apparatus whose booting time can be reduced with an inexpensive configuration and a method for booting the same.

An information processing apparatus according to the present invention comprises:

- a processor;
- a storage device;
- a first power supply;
- a host controller connected to the processor, the host controller reading out codes stored in the storage device to boot the apparatus when the first power supply is turned on;
- a high speed storage element which can be read out at a speed higher than that for the storage device;
- a second power supply for supplying power to each of the host controller and the high speed storage element at all times whether the first power supply is turned on or turned off; and
- a controller for reading out codes stored in advance in the high speed storage element having the same storage contents as those in the storage device to boot the apparatus when the first power supply is turned on.

According to the present invention, there is provided a method for booting an information processing apparatus in which a host controller connected to a processor reads codes stored in a storage device to boot the apparatus when a first power supply is turned on, comprises the steps of:

- supplying power from a second power supply to each of a high speed storage element which can be read out at a speed higher than that for the storage device and the

2

host controller at all times whether the first power supply is on or off; and

reading out codes stored in advance in the high speed storage element having the same storage contents as those in the storage device to boot the apparatus when the first power supply is turned on.

According to the present invention, since booting is performed by reading out codes stored in advance in the high speed storage element having the same contents as those in the storage device when the first power supply is turned on, the time required for booting can be significantly reduced compared to the related art.

Further, according to the present invention, the storage contents of the high speed storage element are cleared during the period from a system terminating operation until the time at which the first power supply is turned off as a result of the system terminating operation; thereafter, a process is performed to copy the codes used for booting read out from the storage device into the high speed storage element and set a write protect therein; and, when the first power supply is turned on again thereafter, booting is carried out using the codes read out from the high speed storage element and the write protect is cancelled. Since this makes it possible to use an existing high speed storage element for booting, the present invention can be carried out at a low cost without the need for additional components.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a configuration diagram showing the principle of the present invention;

FIG. 2 is a block diagram of an embodiment of the present invention; and

FIG. 3 is a flow chart illustrating the operation of the embodiment shown in FIG. 2.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

Preferred embodiments of the present invention will now be described specifically with reference to the accompanying drawings. FIG. 1 is a block diagram of a first embodiment of the present invention. Referring to FIG. 1, an information processing apparatus according to the embodiment of the present invention comprises a processor 11, a host controller 12, a read only memory (ROM) 13 which is a storage device, and a high speed storage element 21. The processor 11, host controller 12 and ROM 13 are provided in a V_{CC} power supply circuit 10 to which power is supplied from a V_{CC} power supply as a first power supply only in a normal operating state.

The host controller 12 and high speed storage element 21 are provided in a V_{EE} power supply circuit 20 to which power is supplied from a V_{EE} power supply as a second power supply not only in a normal operating state but also when the V_{CC} power supply is off. That is, power is supplied to the host controller 12 from both of the V_{CC} and V_{EE} power supplies.

The host controller 12 carries out centralized control over the operation of the information processing apparatus as a whole. Codes to be read to boot the information processing apparatus are stored in the ROM 13 in advance. Codes having the same storage contents as those stored in the ROM 13 are stored in advance in the high speed storage element 21 which is configured such that it can be read out at a much higher speed than the ROM 13. According to the present invention, the storage contents of the high speed storage

3

element 21 are read out to boot the apparatus when the V_{CC} power supply is turned on after the V_{CC} power supply is once turned off.

Further, according to the present invention, when a booting mode is set at a high speed mode during a normal operation for which the first power supply V_{CC} is on, the storage contents of the high speed storage element 21 are cleared during the period from a system terminating operation until the time at which the first power supply V_{CC} is turned off as a result of the system terminating operation under the control of the host controller 12. Thereafter, a process is performed to copy the codes used for booting read out from the ROM 13 into the high speed storage element 21 and set write protect therein. When the first power supply V_{CC} is turned on again thereafter, booting is carried out using the codes read out from the high speed storage element 21 and the write protect is cancelled. This makes it possible to use an existing high speed storage element for booting.

A second embodiment of the present invention will now be described. FIG. 2 is a block diagram of an information processing apparatus according to the second embodiment of the present invention. In FIG. 2, components identical to those in FIG. 1 are indicated by same reference numbers. Referring to FIG. 2, the information processing apparatus comprises a processor 11, a host controller 12, a ROM 13, a system memory 14 and a cache memory (hereinafter referred to as "cache") 22.

The processor 11, the host controller 12, the ROM 13 and the system memory 14 are provided in a V_{CC} power supply circuit 10 to which power is supplied from a V_{CC} power supply only in a normal operating state. The host controller 12 and cache 22 are provided in a V_{EE} power supply circuit 20 to which power is supplied from a V_{EE} power supply not only in a normal operating state but also when the V_{CC} power supply is off. The cache 22 is a memory which is an example of the high speed storage element 21, and write and read operations on the same is controlled by the host controller 12. The same storage contents as those stored in the system memory 14 are stored in the cache 22 during a normal operation.

The operation of the present embodiment will now be described with reference to the flow chart in FIG. 3. In the information processing apparatus of the present embodiment, a V_{EE} power supply is first turned on (step 31) to supply power to each of the host controller 12 and the cache 22 not only in a normal operating state but also when the power supply to the information processing apparatus is off, thereby putting them in an operating state at all times. Then, the V_{CC} power supply is turned on (step 32). Since this is the beginning of the use of the apparatus and a system boot mode has not been set by the user yet, the storage contents of the ROM 13 (BIOS codes) are read out in a default state (step 33).

This puts the information processing apparatus in a normal operating mode after booting (step 34) in which the user sets a boot mode (step 35). The present embodiment has a normal boot mode in which the storage contents of the ROM 13 are read out for booting and a high speed boot mode in which the storage contents of the cache 22 are read out for booting.

If the user chooses to terminate the system thereafter (step 36), it is determined which boot mode has been set (step 37). If the user has set the normal mode described above, the host controller 12 is set such that it reads out the contents of the ROM 13 when booting is carried out (step 38). Therefore, the V_{CC} power supply is turned off as a result of system

4

termination (step 39). When the V_{CC} power supply is turned on again later to use the apparatus (step 40), the host controller 12 reads out the storage contents of the ROM 13 for booting (steps 33, 34).

In the case that the user has set the above-described high speed mode in the normal operating state, if the user chooses to terminate the system (step 36) and it is determined that the high speed mode has been set for the host controller 12 (step 37), the host controller 12 is set in the high speed mode (step 41).

Thereafter, the V_{CC} power supply is turned off when the system is terminated. Since the contents of the system memory 14 have been written into the cache 22 until that time through a normal operation, the cache 22 is flushed to clear the storage contents (step 42) and, thereafter, the storage contents (BIOS codes) of the ROM 13 are copied to the cache 22 of the host controller 12 (step 43). In order to maintain the copied contents, the host controller 12 provides a write protect by adding a cache control signal 23 (step 44) during writing. The system is terminated after the process described above and, as a result, the V_{CC} power supply is turned off (step 45).

When the V_{CC} power supply is turned on again to use the apparatus later (step 46), the host controller 12 reads out the storage contents (BIOS codes) of the cache 22 (step 47), cancels the write protect in the cache 22 with the cache control signal 23 when the reading-out is complete to enable the cache 22 to start operating as a normal cache (step 48) and boots the system (step 34). Thus, the apparatus starts a normal operation.

As described above, booting is carried out using the contents of the cache 22 which can be read out at a high speed instead of the contents of the ROM 13 which is read out at a lower speed. This makes it possible to reduce the apparatus boot time significantly compared to the related art.

In addition, an existing cache 22 which has been provided in an information processing apparatus is used as the high speed storage element 21 to be used for increasing the reading-out speed of the BIOS codes. This eliminates the need for any external additional circuit and therefore the need for additional components, which allows inexpensive configurations.

Since the cache 22 is required to have the memory of the codes having the same storage contents of the ROM 13 when the V_{CC} power supply is turned on after the V_{CC} power supply is once turned off, it may be a dedicated memory in which the codes having the same storage contents of the ROM 13 are stored in advance.

What is claimed is:

1. An information processing apparatus comprising:

- a processor;
- a storage device;
- a first power supply coupled to the processor and the storage device;
- a host controller connected to said processor and said first power supply for reading out basic input/output system codes stored in said storage device to boot the apparatus when said first power supply is turned on;
- a high speed storage element which can be read out at a speed higher than that for said storage device wherein said high speed storage element is a cache memory used during a normal operation for which said first power supply is on and wherein said host controller copies the codes used for booting read out from said storage device into said high speed storage element

5

during the period from a system terminating operation at the host controller until the time said first power supply is turned off, sets a write protect thereafter, reads the codes used for booting from said cache memory when said first power supply is turned on and cancels said write protect thereafter; and

a second power supply for supplying power to each of said host controller and said high speed storage element at all times whether said first power supply is on or off; and

wherein said host controller reads out BIOS codes stored in advance in said high speed storage element having the same storage contents as those in said storage device to boot the apparatus when said first power supply is turned on.

2. A method for booting an information processing apparatus in which a host controller connected to a processor reads out basic input/output system codes stored in a storage device to boot the apparatus when a first power supply is turned on, said method comprising the steps of:

supplying power from a second power supply to each of a high speed storage element which can be read out at a speed higher than that for said storage device and said host controller at all times whether said first power supply is on or off; and

reading out BIOS codes stored in advance in said high speed storage element having the same storage contents as those in said storage device to boot the apparatus when said first power supply is turned on; and

performing booting based on the codes used for booting read out from said storage device in a default state when said first power supply is initially turned on; and copying the codes used for booting read from said storage device into said high speed storage element immediately before said first power supply is turned off only

6

when a high speed boot mode is set during a subsequent normal operation.

3. A method for booting an information processing apparatus, in which a host controller connected to a processor reads out basic input/output system codes stored in a storage device to boot the apparatus when a first power supply is turned on, said method comprising the steps of:

supplying power from a second power supply to each of a high speed storage element which can be read out at a speed higher than that for said storage device and said host controller at all times whether said first power supply is on or off; and

reading out BIOS codes stored in advance in said high speed storage element having the same storage contents as those in said storage device to boot the apparatus when said first power supply is turned on;

wherein if a boot mode is set at the high speed mode during a normal operation for which said first power supply is on;

the storage contents of said high speed storage element are cleared in the period from a system terminating operation until the time at which said first power supply is turned off as a result of said system terminating operation;

a process is performed thereafter to copy the codes used for booting read out from said storage device into said high speed storage element and to set a write protect further; and

booting is performed using the codes read out from said high speed storage element and said write protect is canceled when said first power supply is turned on again thereafter.

* * * * *



US006073232A

United States Patent [19]

[11] Patent Number: **6,073,232**

Krocker et al.

[45] Date of Patent: ***Jun. 6, 2000**

[54] **METHOD FOR MINIMIZING A COMPUTER'S INITIAL PROGRAM LOAD TIME AFTER A SYSTEM RESET OR A POWER-ON USING NON-VOLATILE STORAGE**

5,210,875	5/1993	Bealkowski et al.	395/700
5,230,052	7/1993	Dayan et al.	395/700
5,269,022	12/1993	Shinjo et al.	395/700
5,307,497	4/1994	Feigenbaum et al.	395/700
5,325,532	6/1994	Crosswy et al.	395/700
5,355,498	10/1994	Proving et al.	395/700
5,404,527	4/1995	Irwin et al.	395/700
5,410,699	4/1995	Bealkowski et al.	395/700

[75] Inventors: **Richard Mark Krocker, Morgan Hill; Richard Henry Mandel, III, San Jose, both of Calif.**

[73] Assignee: **International Business Machines Corporation, Armonk, N.Y.**

*Primary Examiner—Ayaz R. Sheikh
Assistant Examiner—Tim Vo
Attorney, Agent, or Firm—Gray Cary Ware Freidenrich*

[*] Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

[57] ABSTRACT

A method for increasing boot speed of a host computer with associated hard disk drive generates a prefetch table that contains pointers to disk locations and lengths of the records of an application program requested by the host computer during an initial power-on/reset. During the next power-on/reset, before the host computer is ready for data but after the disk drive has completed its reset routine, using the prefetch table the disk drive accesses the previously requested data and copies it onto the cache of the disk drive, from where it is transferred to the host computer when the host computer requests it. The prefetch table is updated to reflect disk location changes for the various records, or to reflect new records that were requested by the host computer but not found in cache during the previous power-on/reset.

[21] Appl. No.: **08/806,135**

[22] Filed: **Feb. 25, 1997**

[51] Int. Cl.⁷ **G06F 9/445**

[52] U.S. Cl. **713/1; 713/2; 713/100**

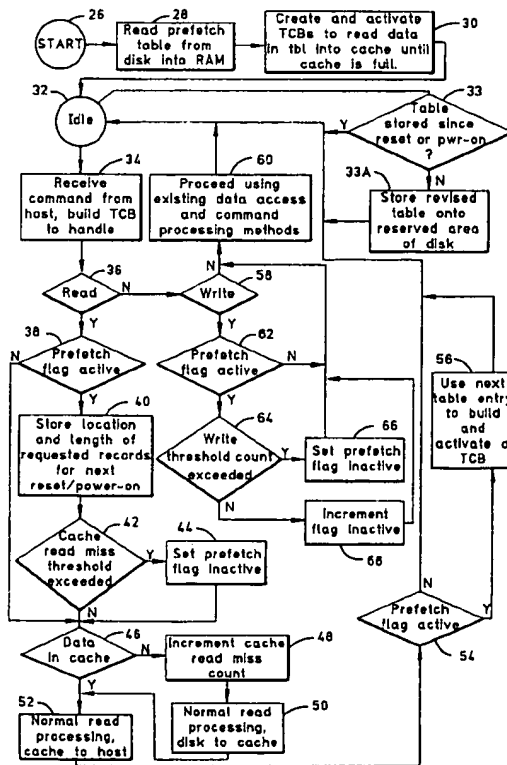
[58] Field of Search **395/651, 652, 395/653; 713/1, 2, 100**

[56] References Cited

U.S. PATENT DOCUMENTS

4,663,707 5/1987 Dawson 364/200

33 Claims, 2 Drawing Sheets



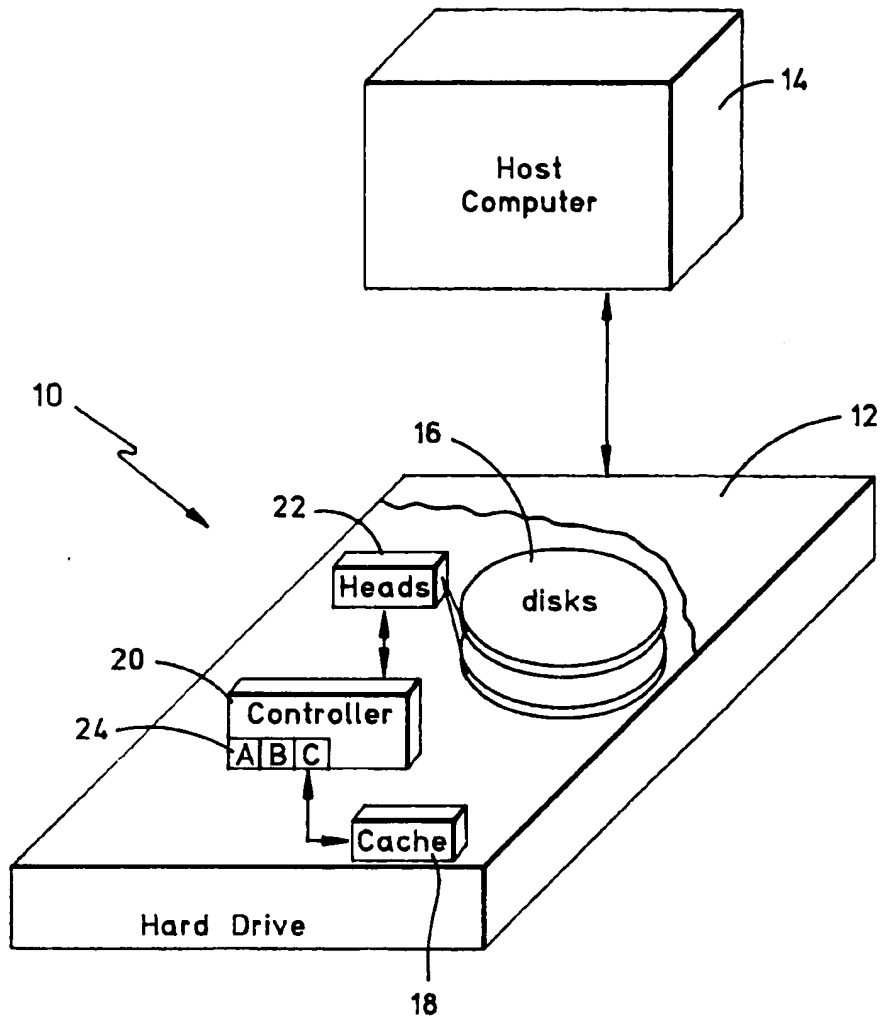


FIG. 1

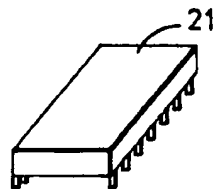


FIG. 2

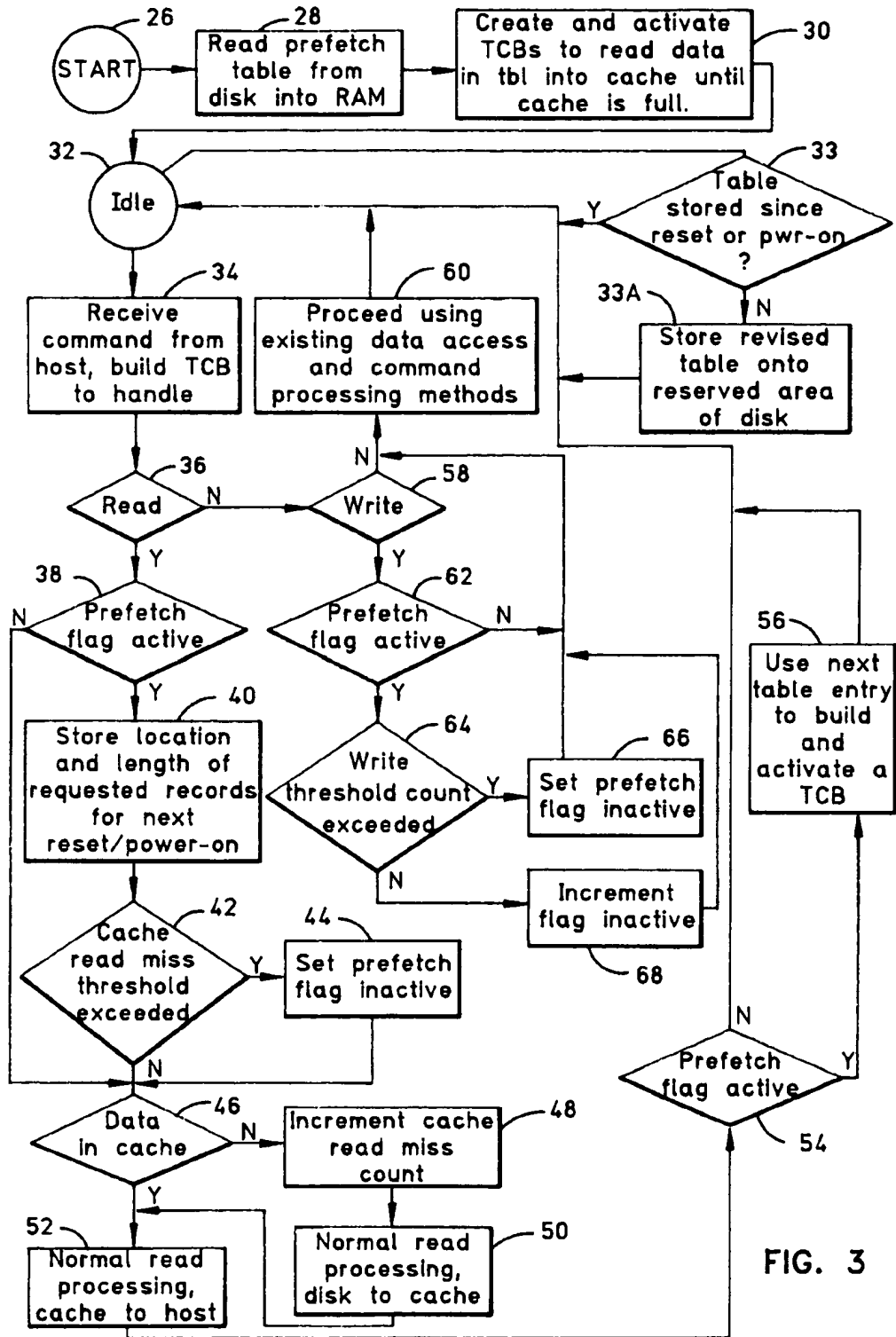


FIG. 3

**METHOD FOR MINIMIZING A
COMPUTER'S INITIAL PROGRAM LOAD
TIME AFTER A SYSTEM RESET OR A
POWER-ON USING NON-VOLATILE
STORAGE**

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to peripheral storage apparatus for computers, and more particularly to shortening the load time of computer programs from a hard disk drive to a host computer.

2. Description of the Related Art

When a computer undergoes a hardware reset (i.e., a power-on or reset), the computer executes procedures embodied in its power-on/reset firmware which prepare for loading an operating system into the computer to condition it for operation. Typically, execution of such procedures begins what is referred to as a "boot". While the computer is executing these power on/reset procedures, peripheral devices that are associated with the computer, such as, for example, a hard disk drive, also execute their own power-on/reset procedures embodied in firmware. When the computer finishes the above-described firmware-implemented portion of the "booting" process, it typically requests data from the disk drive as part of initializing user-selected software, e.g., a program marketed under one of the trade-names DOS, Windows, UNIX, OS/2, AIX, etc.

It happens that the transfer of the selected program to the computer is relatively slow, particularly when the program is a modern large operating system. Accordingly, methods have been disclosed for increasing the speed with which computers "boot". One example of such a method is disclosed in U.S. Pat. No. 5,307,497, which teaches that a portion of an operating system can be stored in read-only memory (ROM) for fast access of the portion during power-on or reset. Unfortunately, the portion stored in ROM is unchangeable. In other words, the method disclosed in the '497 patent does not adapt to changing user preferences regarding operating systems, or indeed to updated versions of a particular operating system.

Another example of a previous attempt to shorten the load time from a disk drive to its host computer is set forth in U.S. Pat. No. 5,269,022. As disclosed in the '022 patent, a snapshot of computer memory is stored in a backup memory that is separate from the disk drive associated with the computer, for use during the next succeeding boot. Unfortunately, the backup memory must be large, because it must store the entire computer memory. Also, the method disclosed in the '022 patent requires operating system intervention, which, because of security features common to many modern operating systems, renders the '022 invention unfeasible.

As recognized by the present invention, however, it is possible to provide, without operating system intervention, a method for adaptively preparing a disk drive to effect rapid application program loading to a host computer. Specifically, we have found that during hardware resets the disk drive associated with a host computer typically completes its booting process before the host computer is ready for program transfer, and as recognized by the present invention, the disk drive can be configured during this period for rapidly communicating a program to the host computer.

Accordingly, it is an object of the present invention to provide a method for rapidly communicating a computer program from a disk drive to a host computer.

Another object of the present invention is to provide a method for rapidly communicating a computer program from a disk drive to a host computer which adapts to changes in user program preference, and to changes in program storage location on the disks of the disk drive. Yet another object of the present invention is to provide a method for rapidly communicating a computer program from a disk drive to a host computer which does not require excessively large storage space, and which can be undertaken entirely by the disk drive itself, transparent to the host computer. Still another object of the present invention is to provide a method for rapidly communicating a computer program from a disk drive to a host computer which is easy to use and cost-effective.

SUMMARY OF THE INVENTION

The invention is embodied in an article of manufacture—a machine component—that is used by a digital processing apparatus and that tangibly embodies a program of instructions that are executable by the digital processing apparatus to rapidly communicate a computer program from a hard disk drive to the drive's host computer.

This invention is realized in a critical machine component that causes a digital processing apparatus to adaptively store a computer program on the cache of the hard disk drive and communicate the program to the host computer. Hereinafter, the machine component is referred to as a "computer program product".

In accordance with the present invention, steps executed by the digital processing apparatus include, after an initial power-up or reset of the hard disk drive and a host computer associated with the drive, receiving an initial read command from the host computer for transferring to the host computer a plurality of data records of a program stored on the disk. A prefetch table is then generated, with the table representing a disk location and length of each data record requested by the initial read command. Then, after a subsequent power-on or reset of the hard disk drive, and during a second power-on or reset of the host computer, the prefetch table is accessed to read into the data cache the data records. In response to a subsequent read command from the host computer, it is determined whether records requested by the subsequent read command are stored in the data cache. If they are, the records are communicated from the cache to the host computer; otherwise, the records are communicated from the disk to the host computer.

Preferably, the accessing and determining steps are repeated for each power-on or reset of the host computer. The steps executed by the digital processing apparatus further include either incrementing a read counter toward a predetermined value, fixed in the algorithm or programmed by the user or adaptively determined based on system environmental conditions, or decrementing a counter towards zero using the same process described for the incrementing condition when it is determined during the determining step that records requested by the subsequent read command are not stored in the data cache. As used generally herein, then, "incrementing" a counter refers both to incrementing and decrementing a counter. Additionally, the steps further include updating the data prefetch table, communicating the records from the disk to the host computer when the read counter exceeds a predetermined threshold, and setting a prefetch flag to inactive when the read counter exceeds the predetermined threshold.

In the presently preferred embodiment, the invention also includes setting the prefetch flag to inactive when a prede-

terminated number of write commands from the host computer to the hard disk drive have been received. Moreover, the invention can include, after the communicating step and when the prefetch flag is inactive, determining whether the hard disk drive is idle and if so, storing the prefetch table on the disk. The computer program product is disclosed in combination with the hard disk drive, and in combination with the host computer.

In another aspect, a computer program product is disclosed for use with a host computer that is coupled to a hard disk drive. The hard disk drive includes at least one storage disk having a program stored thereon and a data cache, and the computer program product includes a data storage device which includes a computer usable medium having computer readable program means. As disclosed in detail below, these code means are for enhanced loading of the program from the hard disk drive to the host computer during power-on or reset of the host computer. In accordance with the present invention, code means receive a command from the host computer during a power-up or reset of the host computer. When the command is a read command, code means generate a prefetch table representative of at least the disk location of the records requested by the read command for transfer of the records from the disk to the cache for a subsequent power-on or reset of the host computer. Moreover, code means are provided for determining whether the records have been stored in the cache in response to a previous power-on or reset of the host computer, and the records are communicated to the host computer in response.

In still another aspect, a computer hard disk drive includes at least one data storage disk and a data storage cache. Furthermore, the hard disk drive includes means for recording onto the cache, immediately after a hardware reset of the hard disk drive, data on the disk that has been requested by a host computer during a first hardware reset of the host computer. Additionally, the disk drive includes means for communicating the data from the cache to the host computer during a second hardware reset of the host computer.

The details of the present invention, both as to its structure and operation, can best be understood in reference to the accompanying drawings, in which like reference numerals refer to like parts, and in which:

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a partially schematic view of a disk drive with associated host computer, with portions broken away for clarity;

FIG. 2 is an illustration of memory such as read-only memory (ROM), random access memory (RAM), electrically-erasable programmable read only memory (EEPROM), or dynamic random access memory (DRAM) containing microcode, that embodies the invention as a program storage product; and

FIG. 3 is a flow chart showing the method steps of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring initially to FIG. 1, a system is shown, generally designated 10, for promoting rapid communication of a computer program from a hard disk drive 12 to a host computer 14 that is in data communication with the disk drive 12 in accordance with principles well-known in the art. In one intended embodiment, the host computer 14 may be a personal computer (PC) or laptop computer made by IBM

Corp. of Armonk, N.Y. Or, the host computer 14 may be a Unix computer, or OS/2 server, or Windows NT server, or IBM RS/6000 250 workstation. Indeed, the host computer 14 can be an embedded controller that is part of a music synthesizer, or part of an industrial instrument. And, the hard disk drive 12 can be any hard disk drive suitable for computer applications, provided that the hard disk drive 12 includes at least one, and typically a plurality of, data storage disks 16 and an on-board, solid state, random access memory (RAM) data cache 18.

As shown in FIG. 1, the hard disk drive 12 also includes an onboard controller 20. In accordance with principles well-known in the art, the onboard controller 20 is a digital processor which, among other things, controls read heads 22 in the disk drive 12 for effecting data transfer to and from the disks 16.

Additionally, as intended by the present invention the onboard controller 20 includes an adaptive cache module 24. Per the present invention, the adaptive cache module 24 is executed by the onboard controller 20 as a series of computer-executable instructions. These instructions are embodied as microcode in a memory, e.g., read-only memory (ROM) of the onboard controller 20. Such a ROM is indicated by reference numeral 21 in FIG. 2. The ROM 21 contains microinstructions that embody means and program steps that perform according to the invention. When in the ROM 21, the microinstructions become part of the ROM 21, and therefore, part of the hardware of the disk drive 12.

Those skilled in the art will appreciate that the hard disk drive is merely illustrative of a particular tangible environment that is useful for understanding the concepts of our invention. Broadly, the hard disk drive 12 represents a peripheral storage apparatus. The hard disks 16 of the hard disk drive 12 represent data storage elements that are found in the general peripheral storage apparatus. The invention, therefore, applies to such a peripheral storage apparatus and a data storage element, and should not be limited to a hard disk drive.

FIG. 3 illustrates the structure of such microinstructions as embodied in a computer program. Those skilled in the art will appreciate that FIG. 3 illustrates the structures of computer program code elements that function according to this invention. Manifestly, the invention may be practiced in its essential embodiment by a machine component, embodied by the ROM 21, that renders the computer program code elements in a form that instructs a digital processing apparatus (e.g., the onboard controller 20) to perform a sequence of function steps corresponding to those shown in the Figures. The machine component is shown in FIGS. 1 and 2 as a combination of program code elements A-C in computer readable form that are embodied in a computer-usable data medium (the ROM 21) of the onboard controller 20. Such media can also be found in other semiconductor devices, on magnetic tape, on optical disks, on floppy diskettes, on a DASD array, on a conventional hard disk drive, in logic circuits, in other data storage devices, or even in a node of a network. In an illustrative embodiment of the invention, the computer-executable instructions would be in object code form, compiled or assembled from a C++ language program and stored, by conventional means, in the ROM 21. Or, the code used can be an interpretative code such as Forth, Smalltalk, or Java and its derivatives.

Referring in detail to FIG. 3, the method of the present invention can be seen. It is to be understood that in the presently preferred embodiment, the method begins immediately after the hard disk drive 12 has completed its power-on/reset (i.e., hardware reset) routine.

Commencing at start state 26, the process moves to block 28, wherein a prefetch table is read from a reserved area of the disks 16 into the RAM cache 18. As discussed further below, the prefetch table contains a listing of the disk locations and lengths of data records that were requested by the host computer 14 in the immediately previous power-on/reset. Additionally, a copy of a prefetch flag, if enabled by the user, is created and set active at block 28. In other words, the prefetch flag is preferably defined to be active by the user of the hard disk drive 12 to enable the adaptive caching of the present invention. If desired, the default setting of the prefetch flag can be set to active. The original prefetch flag resides on the disk and is settable by the user (setting the features for IDE, and setting the mode pages for SCSI), and the copy of the prefetch flag is what is used for all steps discussed below, except where indicated otherwise.

Next, at block 30, task control blocks (TCBs) are created and initiated in accordance with well-known principles to read the data represented by the prefetch table from the disks 16 into the RAM cache 18. The method then enters an idle state 32 to await a command from the host computer 14.

During the idle state 32, the process can move to decision diamond 33 to determine whether the prefetch table has been stored since the latest power on or reset. Also, it is determined whether the original prefetch flag is active and whether the copy of the flag is inactive. If the original flag is active, the copy is not, and the table has not been stored since the latest power on/reset, (and, if desired, a disk drive idle time has been exceeded), then the process moves to block 33A to store the table onto the area of the disk reserved for the table. From block 33A or from decision diamond 33 if any one of the above-noted conditions have not been met, the logic moves back to the idle state 32.

At block 34, a command is received from the host computer 14, and a task control block (TCB) is accordingly built to support the command. From block 34, the process moves to decision diamond 36 to determine whether the command received from the host computer 14 is a read command.

If the command is a read command, indicating that the host computer 14, pursuant to its initialization, is requesting data records that are part of a computer program such as DOS or Windows, the process moves to decision diamond 38 wherein it is determined whether the prefetch flag is active. If it is, the process continues to block 40, wherein the disk location and length of the record requested by the read command is recorded in the prefetch table for the next power-on/reset. Thus, at block 40 the prefetch table is updated to reflect a newly requested record, or to reflect a new disk location of a previously-requested record. Then, it is determined at decision diamond 42 whether a read miss counter exceeds a predetermined read miss threshold. If so, the prefetch flag is set to inactive at block 44.

The skilled artisan will recognize that the read miss threshold represents a predetermined number of cache misses. Per the present invention, the read miss threshold can be a programmed integer, or it can be an adaptively determined integer. For example, the read miss threshold can be calculated as a predetermined fraction of total cache "hits". Or, the read miss threshold can be calculated as the number of misses beyond which a predetermined percentage of the records requested by the host computer 14 cannot be retrieved from the cache 18.

From block 44, or from decision diamonds 38 or 42 when the decisions there are negative, the process moves to decision diamond 46 to determine whether the requested

data exists in cache. If not, a cache read miss counter (initialized at zero) is incremented by one at block 48, and the requested record is transferred from the disks 16 to the host computer 14 using normal processing at block 50. (As the skilled artisan will recognize, a counter can instead be initialized to a predetermined value and then decremented instead of incremented, in which case the test at decision diamond 42 would be changed to "is the read miss counter greater than zero?"). From block 50, or from decision diamond 46 if it was determined that the requested data exists in cache, the process moves to block 52 to transfer the record from cache 18 to the host computer 14.

After the read processing just described, the method proceeds to decision diamond 54 to determine whether the copy of the prefetch flag is active. If it is active, the logic, at block 56, uses the next entry in the prefetch table to build a task control block (TCB) to fetch data into the same segment of the cache 18 that the just-transferred record had occupied prior to being communicated to the host computer 14. In accordance with the present invention, the TCB in block 50 is activated as though a command otherwise was received across the device/file interface. In other words, when the host computer 14 is a PC, the TCB in block 50 is activated as though a command otherwise was received across the SCSI (or IDE)—disk drive interface. In this way, the relatively small amount of cache storage space can be optimally used during the adaptive caching process until all records designated in the prefetch table have been loaded into cache and then transferred to the host computer 14. Also, if desired the process updates the data in the cache 18 of the disk drive 12 in response to the step undertaken at block 56. Stated differently, at block 56 the next entry in the prefetch table is copied from the disks 16 to the data cache 18. The control then loops back to idle state 32.

Thus, the above discussion is directed to the condition wherein a read command is received. Recall that this decision is made at decision diamond 36. In contrast to the steps executed as described above, when it is determined at decision diamond 36 that the command received at block 34 is not a read command, the logic moves to decision diamond 58 to determine whether the command is a write command. If the command is not a write command, the process moves to block 60 to proceed using existing data access and command processing methods, and then the process continues back to the idle state 32.

If, however, the command received at block 34 is a write command, the process moves to decision diamond 62 to determine whether the copy of the prefetch flag is active. If not, the logic loops back to block 60, but otherwise the logic moves to decision diamond 64 to determine whether a write miss counter exceeds a predetermined write miss threshold. If it does, the prefetch flag copy is set to inactive at block 66. Otherwise, the write miss counter is incremented at block 68. From blocks 66 and 68 the process loops back to block 60. If desired, while the disk drive 12 is idle, the data on the disks 16 that was requested by the host computer 14 is reordered on the disks 16 for accessing the data into cache during the next power-on/reset with a minimum of latency and seek times.

The method just described and illustrated in FIG. 3 may be realized in a computer program in, for example, the C++ language, when commonly known programming techniques are employed with reference to the pseudo code representation in Table I.

TABLE I

A high-level pseudo-code representation of a computer program embodying the invention for a hard disk drive sequence after power-on through the first X number of READ commands from a host computer.

- 1) disk drive diagnostics successfully complete
- 2) all drive cache segments are filled with data that was requested from the previous power-on
 - the power-on prefetch table is read from the reserved area on the disk into SRAM
 - a series of tasks are initiated, by creating task control blocks (TCB) and starting them much like they would normally be started upon command receipt, to read the data the power-on prefetch table points to into data buffer RAM.
- 3) drive goes into its idle loop
- 4) command is received from host
- 5) a TCB is built to handle command
- 6) is the command a read?
 - Y) is the prefetch flag active?
 - Y) store the location and length of the requested record for the subsequent power-on or reset cycle.
 - cache read miss threshold exceeded?
 - Y) set prefetch flag inactive
 - data in cache?
 - N) increment cache read miss count
 - normal read processing, set up hardware and start transfer, disk to cache
 - prefetch flag active?
 - Y) when transfer complete, use next entry in prefetch table to build TCB to fetch data into same segment as last transfer. TCB is activated as if a command was received across the interface.
 - N) is the command a write?
 - Y) Prefetch flag active?
 - Y) write threshold count exceeded?
 - Y) set prefetch flag inactive
 - N) increment write threshold count
 - proceed using existing data access and command processing methods
- 7) whenever command is received from host, go to (3).
- 8) during idle time the revised table is stored onto the reserve area of the disk (only done once per power on/reset session).

Note <a>: The file side code (software that works very close to the disk side of the hardware, as opposed to the IDE/SCST or host side of the hardware) operates quasi-independently from host side code (it is linked together through TCBs and other global variables). The hardware can support concurrent host and file side operations.

While the particular METHOD FOR MINIMIZING A
COMPUTER'S INITIAL PROGRAM LOAD TIME
AFTER A SYSTEM RESET OR A POWER-ON USING
NON-VOLATILE STORAGE as herein shown and
described in detail is fully capable of attaining the above-
described objects of the invention, it is to be understood that
it is the presently preferred embodiment of the present
invention and is thus representative of the subject matter
which is broadly contemplated by the present invention, that
the scope of the present invention fully encompasses other
embodiments which may become obvious to those skilled in
the art, and that the scope of the present invention is
accordingly to be limited by nothing other than the appended
claims.

We claim:

1. A computer program product for use with a peripheral storage apparatus including at least one data storage element and a data cache, comprising:
 - a computer program storage medium readable by a digital processing apparatus; and
 - a program means on the program storage medium and including instructions executable by the digital processing apparatus for causing the digital processing apparatus to copy data stored on the data storage element to the data cache by:
 - after an initial power-up or reset of the peripheral storage apparatus and a host computer associated with the peripheral storage apparatus, receiving a read command from the host computer for transfer-

- ring to the host computer a data record of a program stored on the data storage element;
 - in response to receiving, generating and storing in the peripheral storage apparatus a prefetch table representative of a storage location and length of the data record requested by the initial read command;
 - after a subsequent power-on or reset of the peripheral storage apparatus, and during a second power-on or reset of the host computer, accessing by the peripheral storage apparatus the prefetch table to read the data record into the data cache; and
 - in response to subsequent read commands from the host computer, determining whether records requested by the subsequent read commands are stored in the data cache, and if so, communicating the records from the data cache to the host computer, and otherwise communicating the records from the data storage element to the host computer.
2. The computer program product of claim 1, wherein accessing and determining are repeated for each power-on or reset of the host computer, the program means further causing the digital processing apparatus to:
 - increment a read counter when it is determined that records requested by the subsequent read command are not stored in the data cache.
 3. The computer program product of claim 1, wherein the program means further causes the digital processing apparatus to update the data prefetch table.
 4. The computer program product of claim 1, in combination with the storage apparatus.

5. The computer program product of claim 2, wherein the program means further causes the digital processing apparatus to:

communicate the records from the data storage element to the host computer when the read counter exceeds a predetermined threshold; and

set a prefetch flag to inactive when the read counter exceeds the predetermined threshold.

6. The combination of claim 4, in further combination with the host computer.

7. The computer program product of claim 4, wherein the storage apparatus is a disk drive.

8. The computer program product of claim 5, wherein the program means further causes the digital processing apparatus to set the prefetch flag to inactive when a predetermined number of write commands from the host computer to the peripheral storage apparatus have been received.

9. The computer program product of claim 5, wherein the program means further causes the digital processing apparatus to:

after communicating, when the prefetch flag is inactive, determine whether the peripheral storage apparatus is idle and if so, store the prefetch table on the data storage element.

10. The computer program product of claim 7, wherein the data storage element is a disk.

11. A computer program product for use with a host computer coupled to a peripheral storage apparatus including at least one data storage element having a program stored thereon and a data cache, comprising:

a data storage device for use within a peripheral storage apparatus, the data storage device including a computer usable medium having computer readable program means for loading the program from the peripheral storage apparatus to the host computer during power-on or reset of the host computer, comprising:

computer readable code means for receiving a command from the host computer during a power-up or reset of the host computer;

computer readable code means for, when the command is a read command, generating and storing in the peripheral storage apparatus a prefetch table representative of at least a storage location of the records requested by the read command for transfer of the records from the data storage element to the data cache for a subsequent power-on or reset of the host computer;

computer readable code means for determining whether the records have been stored in the data cache in response to a previous power-on or reset of the host computer; and

computer readable code means for communicating the records to the host computer in response to the code means for determining.

12. The computer program product of claim 11, wherein the code means for communicating transfers the records from the cache if the records are in the cache, and otherwise from the data storage element.

13. The computer program product of claim 12, wherein the length of the records is recorded in the prefetch table, and the computer program product further comprises:

computer readable code means for updating the prefetch table when a record or the storage location of a record requested during a subsequent power-on or reset of the host computer is different from the storage location or record represented in the prefetch table.

14. The computer program product of claim 13, further comprising:

computer readable code means for incrementing a read counter when records requested by the read command are not stored in the cache;

computer readable code means for communicating the records from the data storage element to the host computer when the read counter exceeds a predetermined threshold;

computer readable code means for setting a prefetch flag to inactive when the read counter exceeds the predetermined threshold; and

computer readable code means for setting the prefetch flag to inactive when a predetermined number of write commands from the host computer to the peripheral storage apparatus have been received.

15. The computer program product of claim 14, further comprising computer readable code means for, after the communicating step and when the prefetch flag is inactive, determining whether the peripheral storage apparatus is idle and if so, storing the prefetch table on the data storage element.

16. The computer program product of claim 15, in combination with the peripheral storage apparatus.

17. The computer program product of claim 16, wherein the peripheral storage apparatus is a disk drive.

18. The combination of claim 16, in further combination with the host computer.

19. The computer program product of claim 17, wherein the data storage element is a disk.

20. A method for causing a computer program stored in a data storage element in a peripheral storage apparatus of a computer system to be loaded to a host computer, the method comprising the steps of:

after power-up or reset, receiving a read command from the host computer for transferring a data record of a computer program stored on the data storage element to the host computer;

in response to receiving, generating and storing in the peripheral storage apparatus a prefetch table representative of a storage location and length of the data record requested by the initial read command;

after a subsequent power-on or reset of the peripheral storage apparatus, and during a second power-on or reset of the host computer, accessing by the peripheral storage apparatus the prefetch table to read the data record into the data cache; and

in response to subsequent read commands from the host computer, determining whether records requested by the subsequent read commands are stored in the data cache, and if so, communicating the records from the data cache to the host computer, and otherwise communicating the records from the data storage element to the host computer.

21. The method of claim 20, wherein the accessing and determining steps are repeated for each power-on or reset of the host computer, and the method steps further comprise: incrementing a read counter when it is determined during the determining step that records requested by the subsequent read command are not stored in the data cache.

22. The method of claim 20, wherein the method steps further include updating the data prefetch table.

23. The method of claim 21, wherein the method steps further comprise:

communicating the records from the data storage element to the host computer when the read counter exceeds a predetermined threshold; and

setting a prefetch flag to inactive when the read counter exceeds the predetermined threshold.

24. The method of claim 23, wherein the method steps further comprise setting the prefetch flag to inactive when a predetermined number of write commands from the host computer to the peripheral storage apparatus have been received.

25. The method of claim 23, wherein the method steps further comprise:

after the communicating step, when the prefetch flag is inactive, determining whether the peripheral storage apparatus is idle and if so, storing the prefetch table on the data storage element.

26. The method of claim 23, wherein the peripheral storage apparatus is a disk drive.

27. The method of claim 26, wherein the data storage element is a disk.

28. A disk drive, comprising:

one or more read heads for reading from one or more disks of the disk drive;

a data cache comprising a random access memory (RAM);

a controller;

the controller being operative to execute the following steps after a power-up or reset of the disk drive and a host computer associated with the disk drive:

receiving, from the host computer, a read command which requests data records of an application program stored on the one or more disks;

generating and storing a prefetch table in the disk drive in response to receiving the read command, the prefetch table representing disk storage location and length of the data records requested by the read command;

the controller being operative to execute the following steps for subsequent power-up or resets of the disk drive and host computer:

receiving, from the host computer, a subsequent read command which requests data records stored on the one or more disks;

prior to receiving the subsequent read command, identifying and locating the data records of the application program on the one or more disks based on the disk storage location and length represented in the prefetch table;

prior to receiving the subsequent read command, and upon identifying and locating the data records, reading the data records of the application program from the one or more disks and storing them in the data cache; and

in response to receiving the subsequent read command, communicating the prestored data records of the application program from the data cache to the host computer if the subsequent read command requests those data records, and otherwise accessing data records from the one or more disks for communicating.

29. The disk drive according to claim 28, wherein the one or more disks comprise one or more hard disks and the disk drive comprises a hard drive.

30. The disk drive according to claim 28, wherein the controller is further operative to execute the following steps for subsequent power-up or resets of the disk drive and host computer:

determining whether the subsequent read command requests the prestored data records in the data cache; and

updating the prefetch table with disk storage location and length corresponding to those data records being requested by the subsequent read command when those data records are not the same as the prestored data records in the data cache.

31. The disk drive according to claim 28, wherein the one or more disks comprise one or more hard disks and the disk drive comprises a hard drive, and wherein the controller is further operative to execute the following steps for subsequent power-up or resets of the disk drive and host computer:

determining whether the subsequent read command requests the prestored data records in the data cache; and

updating the prefetch table with disk storage location and length corresponding to those data records being requested by the subsequent read command when those data records are not the same as the prestored data records in the data cache.

32. A method for reducing an initial program load time with use of a hard drive, the method comprising:

after a power-up or reset of the hard drive and a host computer associated therewith, the following steps being performed by and at the hard drive:

receiving, from the host computer, a read command which requests data records of an application program stored on one or more hard disks of the hard drive;

generating and storing a prefetch table in response to receiving the read command, the prefetch table representing disk storage location and length of the data records requested by the read command;

for subsequent power-up or resets of the hard drive and host computer, the following steps being performed by and at the hard drive:

receiving, from the host computer, a subsequent read command which requests data records stored on the one or more hard disks;

prior to receiving the subsequent read command, identifying and locating the data records of the application program on the one or more hard disks based on the disk storage location and length represented in the prefetch table;

prior to receiving the subsequent read command, and upon identifying and locating the data records, reading the data records of the application program from the one or more hard disks and storing them in a data cache in the hard drive; and

in response to receiving the subsequent read command, communicating the prestored data records of the application program from the data cache if the subsequent read command requests those data records, and otherwise accessing data records from the one or more hard disks for communicating.

33. The method according to claim 32, further comprising for the subsequent power-up or resets of the hard drive and host computer:

determining whether the subsequent read command requests the prestored data records in the data cache; and

updating the prefetch table with disk storage location and length of those data records being requested by the subsequent read command when those data records are not the same as the prestored data records in the data cache.

* * * * *



US006226667B1

(12) **United States Patent**
Matthews et al.

(10) Patent No.: **US 6,226,667 B1**
(45) Date of Patent: ***May 1, 2001**

(54) **METHOD AND APPARATUS FOR
PRELOADING DATA IN A DISTRIBUTED
DATA PROCESSING SYSTEM**

(75) Inventors: **Gareth Christopher Matthews, Cedar
Park; David Medina, Austin; Allen
Chester Wynn, Round Rock, all of TX
(US)**

(73) Assignee: **International Business Machines
Corporation, Armonk, NY (US)**

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 0 days.

This patent is subject to a terminal dis-
claimer.

(21) Appl. No.: **09/084,277**

(22) Filed: **May 26, 1998**

(51) Int. Cl.⁷ **G06F 15/177; G06F 9/00**

(52) U.S. Cl. **709/203; 709/203; 713/1**

(58) Field of Search **709/220-222,
709/203, 228; 713/1-2, 100**

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,885,770	*	12/1989	Croll	379/269
5,146,568	*	9/1992	Flaherty et al.	716/3
5,367,688	*	11/1994	Croll	713/2
5,404,527	*	4/1995	Irwin et al.	709/222
5,444,850	*	8/1995	Chang	709/222
5,452,454		9/1995	Basu	395/700
5,485,609	*	1/1996	Vitter et al.	707/101
5,530,862	*	6/1996	Wadsworth et al.	713/1
5,666,293	*	9/1997	Metz et al.	709/220
5,708,820	*	1/1998	Park et al.	713/323
5,715,456	*	2/1998	Bennett et al.	713/2
5,752,042	*	5/1998	Cole et al.	717/11
5,758,072	*	5/1998	Filepp et al.	709/220
5,758,165		5/1998	Shuff	717/11
5,778,443		7/1998	Swanberg et al.	711/162

5,832,283	11/1998	Chou et al.	713/300
5,872,968	2/1999	Knox et al.	713/2
5,987,506	11/1999	Carter et al.	709/213
6,074,435	* 6/2000	Rojestal	717/11
6,101,601	* 8/2000	Matthews et al.	713/2
6,108,697	* 8/2000	Raymond et al.	709/218
6,131,159	* 10/2000	Hecht et al.	713/1
6,134,616	* 10/2000	Beatty	710/104

OTHER PUBLICATIONS

Gralla, P., "How the Internet Works," Ziff-Davis Press, pp. 126-127, 1994.*

Bestavros, Azer "Using Speculation to Reduce Server Load and Service Time on the WWW", pp. 403-410, 1995 Computer CIKM International Conference On Information Communications Review and Knowledge Management. vol. 26, No. 3, pp. 22-36 Jul. 1996.

* cited by examiner

Primary Examiner—Zarni Maung

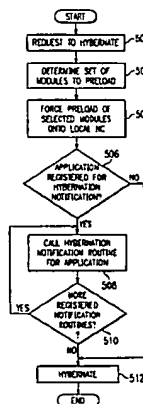
Assistant Examiner—Andrew Caldwell

(74) *Attorney, Agent, or Firm*—Duke W. Yee; Jeffrey S. LaBaw

(57) **ABSTRACT**

A method and apparatus for reducing time needed to initialize a data processing system and to execute applications on the data processing system: In accordance with a preferred embodiment of the present invention, pages for an application are pre-loaded onto a client from a server. The pre-loading of the application includes loading pages that will be required for execution of the application in preparation for hibernation. These pages may include other pages for executable code or data that will be used during execution of the application. Subsequently, the application is executed using the locally stored pages without having to retrieve pages from the server. In addition, an application is provided with an opportunity to prepare itself for hibernation via hibernation notification. For example, the application may read and process files from the server. This processing is done once prior to hibernation and is not required for later executions of the application.

25 Claims, 4 Drawing Sheets



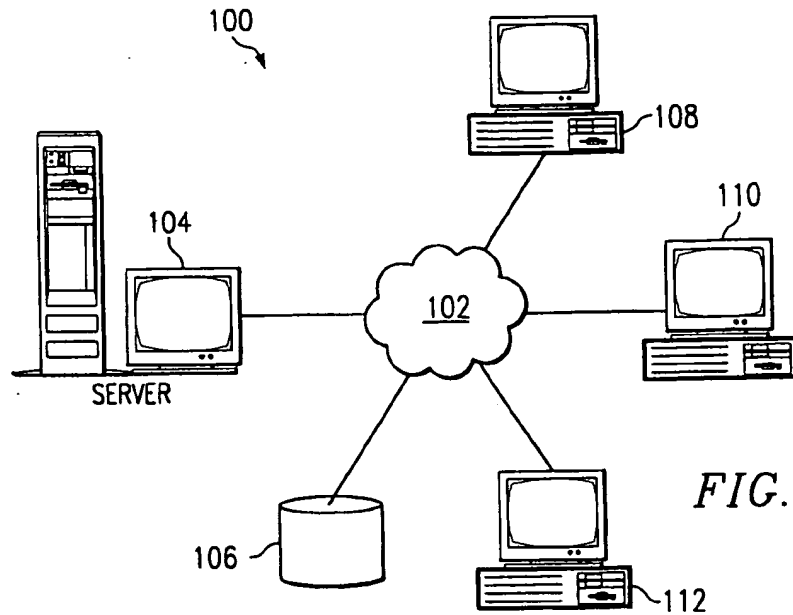


FIG. 1

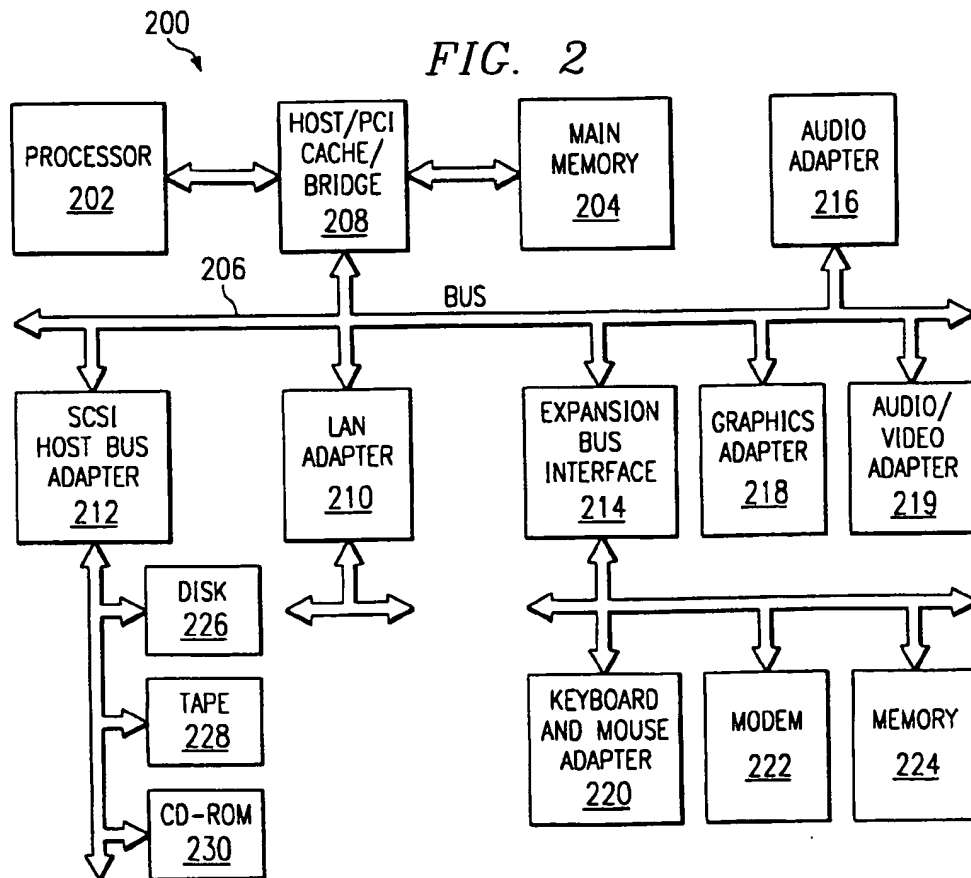


FIG. 2

FIG. 3

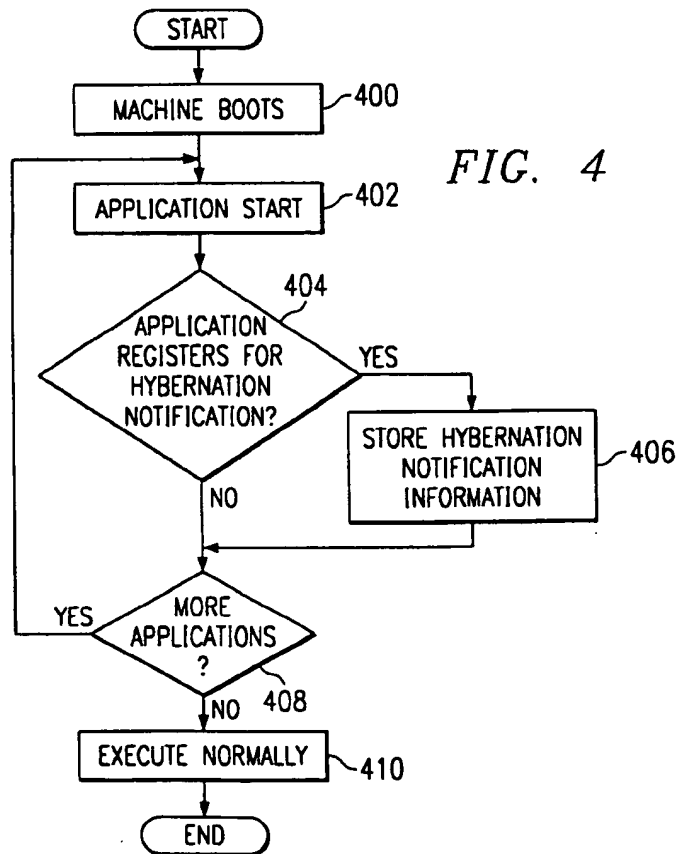
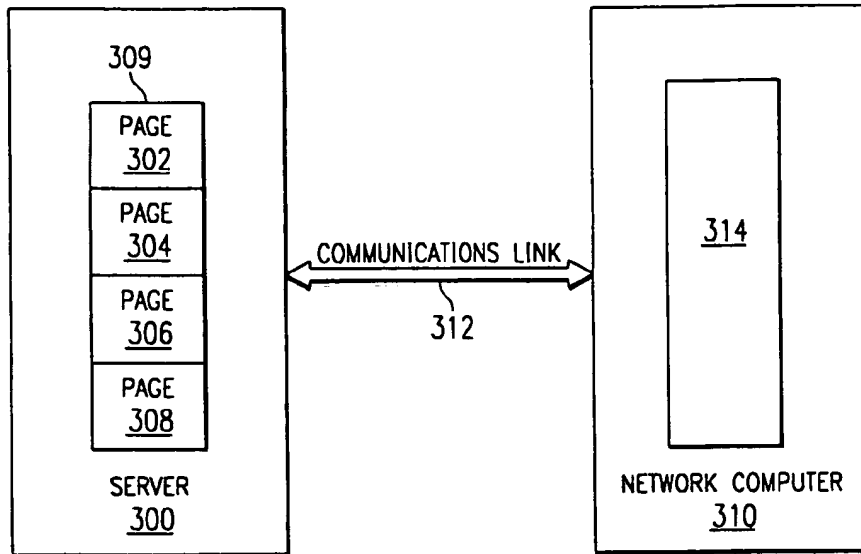


FIG. 4

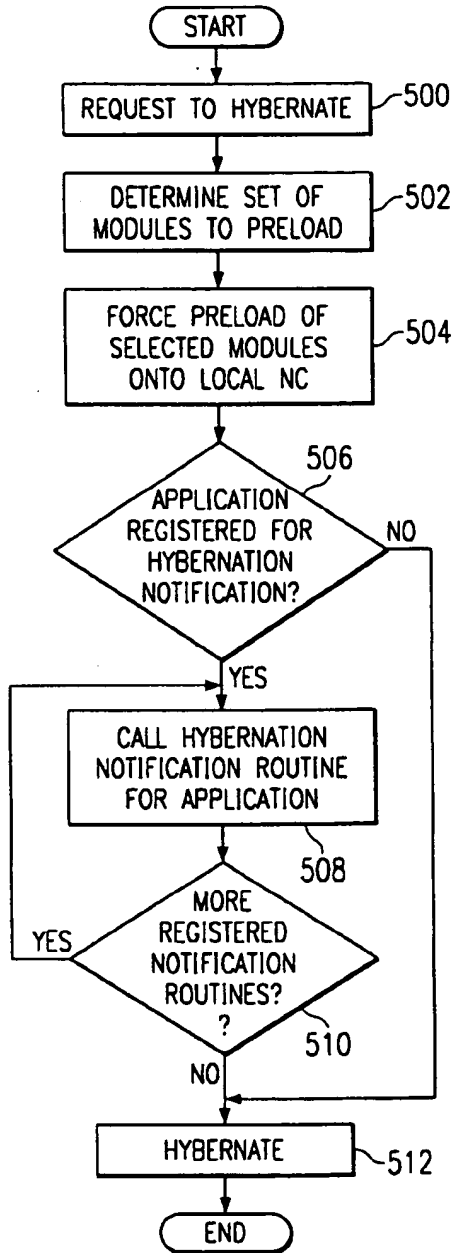


FIG. 5

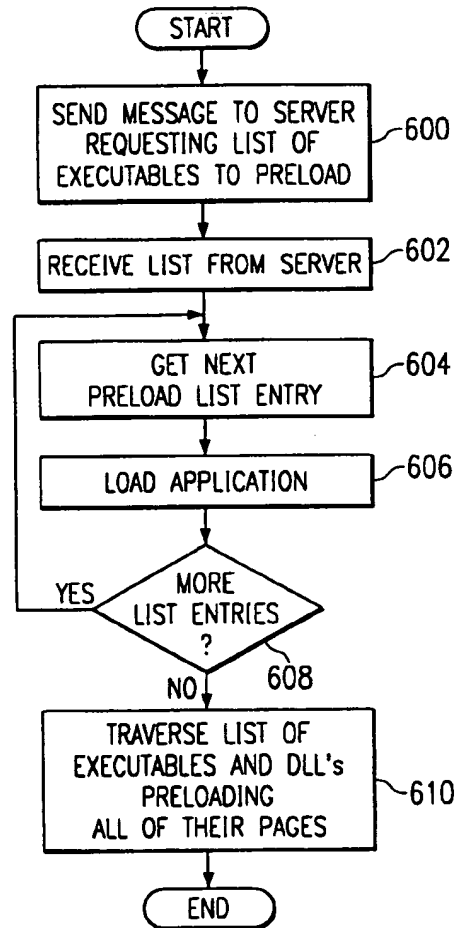


FIG. 6

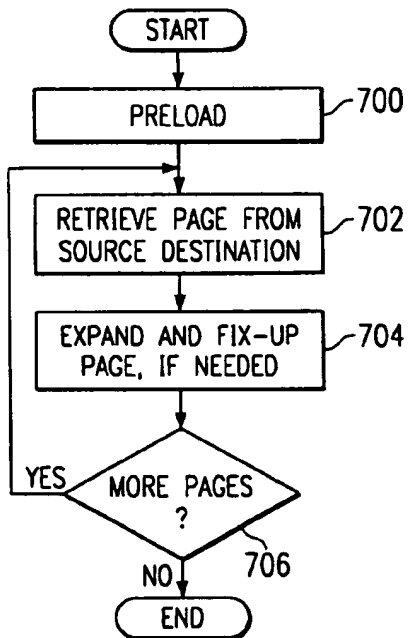


FIG. 7

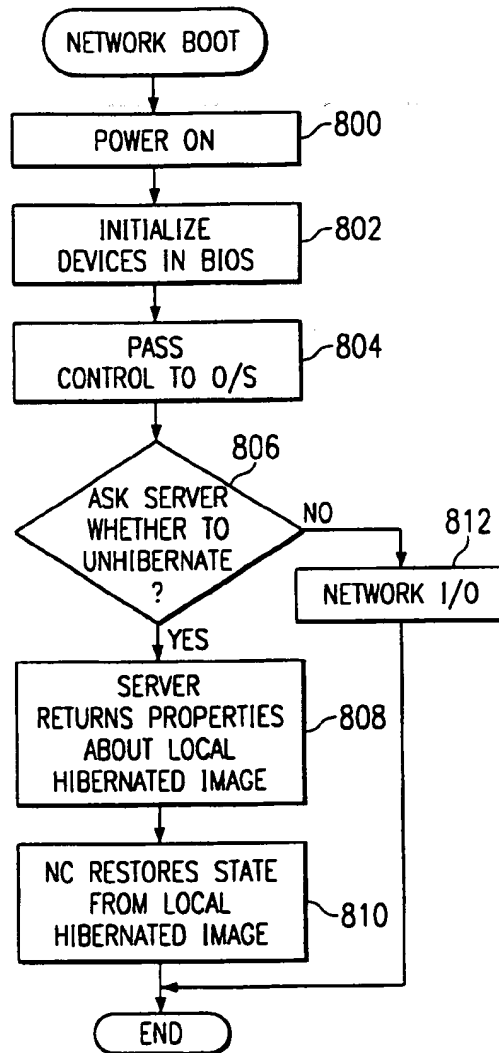


FIG. 8

1

METHOD AND APPARATUS FOR PRELOADING DATA IN A DISTRIBUTED DATA PROCESSING SYSTEM

CROSS REFERENCE TO RELATED APPLICATION

The present invention is related to an application entitled Method and Apparatus For Hibernation Within A Distributed Data Processing System, Ser. No. 09/062,885, filed Apr. 20, 1998, now U.S. Pat. No. 6,101,601, assigned to the same assignee and incorporated herein by reference.

BACKGROUND OF THE INVENTION

1. Technical Field

The present invention relates to an improved distributed data processing system and in particular to an improved method and apparatus for managing data processing systems within a distributed data processing system. Still more particularly, the present invention relates to a method and apparatus for pre-loading data within a distributed data processing system.

2. Description of Related Art

A computer includes both a physical machine, namely the hardware, and the instructions which cause the physical machine to operate, namely the software. Software includes both application and operating system programs. If the program is simply to do tasks for a user, such as solving specific problems, it is referred to as application software. If a program controls the hardware of the computer and the execution of the application programs, it is called operating system software. System software further includes the operating system, the program which controls the actual computer or central processing unit (CPU), and device drivers which control the input and output devices (I/O) such as printers and terminals.

A general purpose computer is fairly complicated. Usually a queue of application programs is present waiting to use the CPU. The operating system will need to determine which program will run next, how much of the CPU time it will be allowed to use and what other computer resources the application will be allowed to use. Further, each application program will require a special input or output device and the application program must transfer its data to the operating system, which controls the device drivers.

A network containing a number of computers may be formed by having these computers, also referred to as "nodes" or "network computers", communicate with each other over one or more communications links, which is an aggregation which is a computer network. Today, many computer workstations are connected to other workstations, file servers, or other resources over a local area network (LAN). Each computer on a network is connected to the network via an adapter card or other similar means, which provides an ability to establish a communications link to the network.

Recently, in a many network computer paradigms, applications are stored on a server and are sent to other network computers (NCs), also referred to as "clients".

A problem exists for network-loaded applications or images. In a network computing environment, when a client's system applications are accessed across the network, a run time problem in the application may occur due to constraints, such as, network traffic, application or image size, or initialization of the image. Such a situation may result in minutes being required to execute an application or

2

portions of the application. In addition, as an application is executed on a client, additional code in the form of pages may be required to continue execution of the application. In such an instance, these additional pages are downloaded from the server, requiring additional access across the network. A "page" is a fixed size block of memory. When a page is used in a paging memory system, a page is a block of memory whose physical address can be changed via mapping hardware.

Therefore, it would be advantageous to have an improved method and apparatus for executing applications across a network.

SUMMARY OF THE INVENTION

It is one object of the present invention to provide an improved distributed data processing system.

It is another object of the present invention to provide an improved method and apparatus for managing data processing systems within a distributed data processing system.

It is yet another object of the present invention to provide a method and apparatus for pre-loading data within a distributed data processing system.

The present invention provides a method and apparatus for reducing time needed to initialize a data processing system and to execute applications on the data processing system. In accordance with a preferred embodiment of the present invention, pages for an application are pre-loaded onto a client from a server. The pre-loading of the application includes loading pages that will be required for execution of the application in preparation for hibernation. These pages may include other pages for executable code or data that will be used during execution of the application. Subsequently, the application is executed using the locally stored pages without having to retrieve pages from the server.

In addition, an application is provided with an opportunity to prepare itself for hibernation via hibernation notification. For example, the application may read and process files from the server. This processing is done once prior to hibernation and is not required for later executions of the application.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

FIG. 1 is a diagram of a distributed data processing system in accordance with a preferred embodiment of the present invention;

FIG. 2 is a block diagram of a data processing system in which the present invention may be implemented;

FIG. 3 is a block diagram of components used in managing a network computer in accordance with a preferred embodiment of the present invention;

FIG. 4 is a flowchart of a process for registering an application for hibernation notification in accordance with a preferred embodiment of the present invention;

FIG. 5 is a flowchart of a process used by a network computer to pre-load modules from a server in accordance with a preferred embodiment of the present invention;

FIG. 6 is a flowchart of a process used by a network computer for determining which modules to pre-load in accordance with a preferred embodiment of the present invention;

FIG. 7 is a flowchart of a process for pre-loading pages in accordance with a preferred embodiment of the present invention; and

FIG. 8, a flowchart of a process for booting a network computer in accordance with a preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

With reference now to the figures, and in particular with reference to FIG. 1, a diagram of a distributed data processing system is depicted in accordance with a preferred embodiment of the present invention. Distributed data processing system 100 is a network of computers in which the present invention may be implemented. Distributed data processing system 100 contains a network 102, which is the medium used to provide communications links between various devices and computers connected together within distributed data processing system 100. Network 102 may include permanent connections, such as wire or fiber optic cables, or temporary connections made through telephone connections.

In the depicted example, a server 104 is connected to network 102 along with storage unit 106. In addition, network computers (NCs) 108, 110, and 112 also are connected to network 102. For purposes of this application, a network computer is any computer, coupled to a network, which receives a boot image from another computer coupled to the network and also may be a server managed computer. Server 104 provides data, such as boot files, operating system images, and applications to NCs 108-112. NCs 108, 110, and 112 are clients to server 104. Distributed data processing system 100 may include additional servers, NCs, and other devices not shown. FIG. 1 is intended as an example, and not as an architectural limitation for the processes of the present invention.

Turning next to FIG. 2, a block diagram of a data processing system 200 in which the present invention may be implemented is illustrated. Data processing system 200 employs a peripheral component interconnect (PCI) local bus architecture. Although the depicted example employs a PCI bus, other bus architectures such as Micro Channel and ISA may be used. Processor 202 and main memory 204 are connected to PCI local bus 206 through PCI bridge 208. PCI bridge 208 also may include an integrated memory controller and cache memory for processor 202. Additional connections to PCI local bus 206 may be made through direct component interconnection or through add-in boards. In the depicted example, local area network (LAN) adapter 210, SCSI host bus adapter 212, and expansion bus interface 214 are connected to PCI local bus 206 by direct component connection. In contrast, audio adapter 216, graphics adapter 218, and audio/video adapter (A/V) 219 are connected to PCI local bus 206 by add-in boards inserted into expansion slots. Expansion bus interface 214 provides a connection for a keyboard and mouse adapter 220, modem 222, and additional memory 224. SCSI host bus adapter 212 provides a connection for hard disk drive 226, tape drive 228, and CD-ROM 230 in the depicted example. Typical PCI local bus implementations will support three or four PCI expansion slots or add-in connectors.

Those of ordinary skill in the art will appreciate that the hardware in FIG. 2 may vary depending on the implementation. For example, other peripheral devices, such as optical disk drives and the like may be used in addition to or in place of the hardware depicted in FIG. 2. Implemented as an NC,

data processing system 200 may include fewer components than illustrated in FIG. 2. For example, many NCs may be diskless or have only a single storage device, such as hard disk drive 226. Data processing system 200 also may be implemented as a server. The depicted example is not meant to imply architectural limitations with respect to the present invention.

The present invention provides a method and system for minimizing traffic within a distributed data processing system by pre-loading client images for applications onto client network computers for use in a hibernated environment. In the depicted example, the processes of the present invention are implemented in the operating system on the client data processing system.

Turning next to FIG. 3, a block diagram of components used in managing a network computer is depicted in accordance with a preferred embodiment of the present invention. In the depicted example in FIG. 3, server 300 downloads pages 302-308 for application 309 to NC 310 using communications link 312. In the depicted example, a page is a fixed-size block of memory. When used in the context of a paging memory system, a page is a block of memory whose physical address can be changed via mapping hardware. These pages may include execution code and data for an application used on NC 310. Typically, only some pages for the application are initially sent to NC 310. Thus, hibernating NC 310 at this point would provide some benefits at boot up, but would still require accessing server 300 to load any other pages that the application would need during runtime. According to the present invention, in response to NC 310 receiving a hibernation request from server 300, all of the pages for an application would be sent to NC 310. After all of the pages have been sent to NC 310, NC 310 is hibernated in a known or selected state to produce image 314, which may be used to restart NC 310, when NC 310 is restarted and unhibernation is appropriate. These pages are stored in a paging space or paging file in image 314. Hibernation involves saving the state of a computer, such as a NC. Hibernation is implementation specific and the data may be saved in a file or raw I/O format located on a storage device, such as a hard disk drive or a static RAM. More information on hibernation may be found in Method and Apparatus for Hibernation Within A Distributed Data Processing System, serial no. 09/062,885, filed on Apr. 20, 1998.

In this manner, applications are pre-loaded onto NC 310 such that network traffic within the distributed data processing system is reduced by reducing the number of accesses to server 300. In addition, the present invention may notify an application that hibernation will occur and allows the application to perform whatever processes are necessary to prepare for hibernation. These processes may include, for example, reading and processing additional files from the server or processing data. In the depicted example, an application is notified if the application has registered for hibernation notification. In the depicted example, each application individually registers hibernation notification for hibernation.

With reference now to FIG. 4, a flowchart of a process for registering an application for hibernation notification is illustrated in accordance with a preferred embodiment of the present invention. The process begins with the booting of the NC (step 400). Thereafter, an application starts (step 402), and a determination is made as to whether the application is one that will register for hibernation notification. (step 404). If the application is to register for hibernation notification it issues an API call. Thereafter hibernation notification information for the application is stored (step 406). Registration

5

involves employing an API to inform the operating system to call a selected function for the application when hibernation occurs. Thereafter, a determination is made as to whether more applications are started (step 408). If more applications are started, the process returns to step 402. Otherwise, the NC executes in a normal fashion (step 410). With reference again to step 404, if the application does not register for hibernation notification, the process proceeds directly to step 408.

Turning next to FIG. 5, a flowchart of a process used by a network computer to pre-load modules from a server is depicted in accordance with a preferred embodiment of the present invention. The process begins by receiving a request to hibernate the network computer (step 500). Thereafter, the set of modules that are to be pre-loaded is identified (step 502). These modules may be, for example, executable programs for the application or images containing the executable programs, dynamic link library (DLL) files, or memory pages. The identified modules are then loaded onto the local network computer (step 504). Identified modules are modules that are identified by the system administrator as a set of executables that will normally be executed on a particular network computer. Thereafter, a determination is made as to whether applications are registered for hibernation notification (step 506). If any application has been registered for hibernation notification, the process then calls the hibernation notification routine for the applications (step 508). Calling the hibernation notification routine for an application allows the application to pull and process application specific information, such as, for example, files from the server. A determination is then made as to whether more registered notification routines are present for execution (step 510). If additional hibernation notification routines are present, the process returns to step 508. Otherwise, hibernation of the network computer is initiated (step 512). With reference again to step 506, if no applications are registered for hibernation notification, the process also proceeds to step 512 to initiate hibernation.

Turning now to FIG. 6, a flowchart of a process used by a network computer for determining which modules to pre-load is illustrated in accordance with a preferred embodiment of the present invention. In the depicted example, the end user tells system administrator what applications will be used on the network computer. Of course, other mechanisms may be used such as having the system administrator select the applications. FIG. 6 is a more detailed description of steps 502 and 504 in FIG. 5. The process begins by sending a message to the server, requesting a list of modules or executable files to pre-load onto the network computer (step 600). In response to this request, a list of the modules is received from the server (step 602). The process then obtains the next pre-load entry on the list (step 604). The list of modules may take various forms depending on the implementation. A list may include, for example, the path and name of the module. Alternatively, the list may include the name of module and a list of pages from the module to load from. In this manner a network computer that has limited resources can have crucial portions of an application loaded while omitting pages that are not frequently used. Thereafter, the application or modules are loaded onto the network computer from the server (step 606). After loading of the application, a determination is made as to whether additional entries are present on the list (step 608). If additional entries are present, the process returns to step 604 to obtain the next entry. Otherwise, the list of executables and DLLs currently in use are traversed and all of the pages for these files are pre-loaded onto the

6

network computer (step 610) with the process terminating thereafter. After pre-loading all of the pages for all applications, hibernation is used to save an image of the network computer.

Turning next to FIG. 7, a flowchart of a process for pre-loading pages is depicted in accordance with a preferred embodiment of the present invention. FIG. 7 is a more detailed description of step 610 in FIG. 6. The process begins by receiving a request to pre-load pages (step 700). Thereafter, a page is retrieved from the source (the server) and across the network and stored in the network computer (step 702). After the page has been retrieved, the page is expanded and fixed up, if required (step 704). In the depicted example, pages may be retrieved from the module on the server and may be in a compressed form. In such an instance, decompression is performed so that each time the page is accessed by the network computer, it does not have to be uncompressed. Fix up of code occurs to provide any needed fix up of relocatable addresses referenced in the code. Such a feature allows for relocation of code in memory. A determination is then made as to whether additional pages are present for retrieval from the source (step 706). If additional pages are present for retrieval from the source, the process then returns to step 702 to retrieve the next page. Otherwise, the process of pre-loading pages is finished.

With reference now to FIG. 8, a flowchart of a process for booting a network computer is depicted in accordance with a preferred embodiment of the present invention. Process begins with a system event such as the power being turned on at the network computer (step 800). Thereafter, devices are initialized in the BIOS (step 802), and control is passed to the operating system (step 804). The network computer then polls the server to determine whether to unhibernate (step 806). The polling may be accomplished using a known network protocol. Unhibernate means to restore the stored state of the system back to the physical state, e.g. restore device states, load physical memory, and set paging space back to pre-hibernated state. If the NC is to unhibernate locally, the server returns information describing properties and location of the hibernated image (step 808). Examples of criteria that may be used by a server to determine whether to unhibernate locally or to use an image from the server include the addition of new devices to the hardware, changes in applications used by the network computer, or updates to operating systems. The network computer then restores itself to the desired state from the local hibernation image (step 810), with the boot process terminating thereafter.

With reference again to step 806, if the server indicates that a normal network boot is to be employed, normal network I/O occurs to remotely boot the NC (step 812) with the boot process terminating thereafter. The server may indicate that a normal network boot is to be employed in order to download new data, such as, for example, updated operating systems, applications, or configurations. Additionally, new applications may be loaded from the server to the NC. After a normal network boot, the NC may then be instructed to save an image of the system for hibernation using the processes of the present invention.

Thus, the present invention provides an improved method and apparatus for reducing the amount of network access needed to execute applications on a network computer. The present invention also decreases boot time and run time using the hibernation mechanism described above. The present invention provides this advantage by pre-loading pages for modules required for execution of applications at the time the NC is requested to hibernate. In addition, the

present invention provides hibernation notification to registered applications, which allows an application to perform necessary tasks for hibernation. For example, when notified, a word processing program may pre-load macros in response to receiving hibernation notification, rather than waiting for a macro to be executed. This mechanism reduces the need to access the server at a later time for additional pages and eliminates the need for the network computer to re-load the application from the server each time the application is run.

It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in a form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media such as a floppy disc, a hard disk drive, a RAM, and CD-ROMs and transmission-type media such as digital and analog communications links.

The description of the present invention has been presented for purposes of illustration and description, but is not limited to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiment was chosen and described in order to best explain the principles of the invention, the practical application, and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated.

What is claimed is:

1. A method for reducing network traffic in a distributed data processing system, the method comprising:

pre-loading a plurality of pages associated with an application onto a client data processing system from a server data processing system;

hibernating the client data processing system after the plurality of pages has been pre-loaded onto the client data processing system to form an image containing the plurality of pages; and

executing the application using plurality of pages on the client data processing system, wherein the client data processing system is unhibernated, the application is executed using the image without accessing the server data processing system.

2. The method of claim 1, wherein each page is locally stored in a paging file within the client data processing system.

3. The method of claim 1, wherein the plurality of pages are in a compressed format and further comprising decompressing the plurality of pages.

4. The method of claim 1, wherein the step of hibernating the client data processing system includes notifying the application that hibernation of the client data processing system will occur.

5. A method in a data processing system for reducing time needed to access client images in a distributed data processing system, the method comprising:

transferring a plurality of memory pages for an application from a server to a client, wherein the plurality of pages includes code needed to execute the application each time the application is started;

storing the plurality of memory pages for the application within the client;

accessing the stored plurality of memory pages when executing the application without accessing the server to execute the application;

hibernating the client;

unhibernating the client; and

executing the application only using the plurality of pages stored within client.

6. The method of claim 5, wherein the plurality of memory pages is stored in a paging file within the client.

7. A method in a distributed data processing system for executing an application on a client data processing system, wherein the application is stored on a server and includes a plurality of portions, the method comprising:

pre-loading the plurality of portions for the application onto a client data processing system from a server within the distributed data processing system;

hibernating the client data processing system after the plurality of portions have been pre-loaded onto the client data processing system, wherein the plurality of portions for the application are stored within the client data processing system; and

executing the application after unhibernating the client data processing system using the stored plurality of portions for the application without accessing the server data processing system to execute the application.

8. The method of claim 7, wherein the plurality of portions is a plurality of pages.

9. The method of claim 7, wherein the plurality of portions includes a number of executable files.

10. The method of claim 7, wherein the plurality of portions includes a number of dynamic-link library files.

11. A data processing system for reducing network traffic in a distributed data processing system, the data processing system comprising:

pre-loading means for pre-loading a plurality of pages associated with an application onto a client data processing system from a server data processing system;

hibernation means for hibernating the client data processing system after the plurality of pages has been pre-loaded onto the client data processing system to form an image containing the plurality of pages; and

execution means for executing the application using plurality of pages on the client data processing system each time the application is run using the image without accessing the server data processing system.

12. The data processing system of claim 11, wherein each page is locally stored in a paging file within the client data processing system.

13. The data processing system of claim 11, wherein the plurality of pages is in a compressed format and further comprising decompression means for decompressing the plurality of pages.

14. The data processing system of claim 11, wherein the hibernation means includes notification means for notifying the application that hibernation of the client data processing system will occur.

15. A data processing system for reducing time needed to access client images in a distributed data processing system, the data processing system comprising:

transferring means for transferring a plurality of memory pages for an application from a server to a client;

storing means for storing the plurality of memory pages for an application within the client;

accessing means for accessing the stored plurality of memory pages when executing the application without accessing the server to execute the application;

hibernating means for hibernating the client;

unhiberating means for unhibernating the client; and
 executing means for executing the application only using
 the plurality of pages stored within client.

16. The data processing system of claim 15, wherein the
 plurality of memory pages is stored on a hard disk drive
 within the client.

17. The data processing system of claim 15, wherein the
 plurality of memory pages is stored on a tape drive within
 the client.

18. The data processing system of claim 15, wherein the
 plurality of memory pages is stored in a paging file within
 the client.

19. A data processing system for executing an application
 on a client data processing system, wherein the application
 is stored on a server and includes a plurality of portions, the
 data processing system comprising:

pre-loading means for pre-loading the plurality of por-
 tions for the application onto the client data processing
 system from a server within the distributed data process-
 ing system;

hibernation means for hibernating the client data process-
 ing system after the plurality of portions have been
 pre-loaded onto the data processing system, wherein
 the plurality of portions for the application are stored
 within the client data processing system; and

execution means for executing the application after unhi-
 bernating the client data processing system, using the
 stored plurality of portions for the application without
 accessing the server to execute the application.

20. The data processing system of claim 19, wherein the
 plurality of portions is a plurality of pages.

21. The data processing system of claim 19, wherein the
 plurality of portions includes a number of executable files.

22. The data processing system of claim 19, wherein the
 plurality of portions includes a number of dynamic-link
 library files.

23. The data processing system of claim 19 further
 comprising notification means for notifying the application
 to prepare for hibernation after the pre-loading means has
 pre-loaded the plurality of pages.

24. A computer program product for executing an appli-
 cation on a client data processing system within a distributed
 data processing system, the computer program product
 comprising:

first instructions for pre-loading a plurality of pages
 associated with an application onto the client data
 processing system from a server data processing sys-
 tem;

second instructions for hibernating the client data pro-
 cessing system after the plurality of pages has been
 pre-loaded onto the client data processing system to
 form an image containing the plurality of pages; and

third instructions for executing the application using the
 plurality of pages on the client data processing system
 after each time the application is executed using the
 image without accessing the server data processing
 system.

25. The computer program product of claim 24, further
 comprising fourth instructions for notifying the application
 that hibernation of the client data processing system will
 occur, wherein the application may prepare for hibernation.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,226,667 B1
DATED : May 1, 2001
INVENTOR(S) : Gareth Christopher Matthews et al.

Page 1 of 4

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Title page.

The Title page should be deleted and substitute therefore the attached Title page.

Drawings.

Delete Figs. 4 and 5 and substitute Figs. 4 and 5 as shown on the attached pages.

Signed and Sealed this

Twenty-third Day of July, 2002

Attest:



Attesting Officer

JAMES E. ROGAN
Director of the United States Patent and Trademark Office

(12) **United States Patent**
 Matthews et al.

(10) Patent No.: **US 6,226,667 B1**
 (45) Date of Patent: ***May 1, 2001**

(54) **METHOD AND APPARATUS FOR PRELOADING DATA IN A DISTRIBUTED DATA PROCESSING SYSTEM**

(75) Inventors: **Gareth Christopher Matthews, Cedar Park; David Medina, Austin; Allen Chester Wynn, Round Rock, all of TX (US)**

5,832,283 11/1998 Chou et al. 713/300
 5,872,968 2/1999 Knox et al. 713/2
 5,987,506 11/1999 Carter et al. 709/213
 6,074,435 * 6/2000 Rojesta 717/11
 6,101,601 * 8/2000 Matthews et al. 713/2
 6,108,697 * 8/2000 Raymond et al. 709/218
 6,131,159 * 10/2000 Hecht et al. 713/1
 6,134,616 * 10/2000 Beatty 710/104

(73) Assignee: **International Business Machines Corporation, Armonk, NY (US)**

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

OTHER PUBLICATIONS

Gralla, P., "How the Internet Works," Ziff-Davis Press, pp. 126-127, 1994.*

Bestavros, Azer "Using Speculation to Reduce Server Load and Service Time on the WWW", pp. 403-410, 1995 Computer CIKM International Conference On Information Communications Review and Knowledge Management, vol. 26, No. 3, pp. 22-36 Jul. 1996.

This patent is subject to a terminal disclaimer.

* cited by examiner

(21) Appl. No.: **09/084,277**

Primary Examiner—Zarni Maung
Assistant Examiner—Andrew Caldwell

(22) Filed: **May 26, 1998**

(74) *Attorney, Agent, or Firm—Duke W. Yee; Jeffrey S. LaBaw*

(51) Int. Cl.⁷ **G06F 15/177; G06F 9/00**
 (52) U.S. Cl. **709/203; 709/203; 713/1**
 (58) Field of Search **709/220-222, 709/203, 228; 713/1-2, 100**

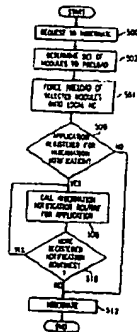
(57) ABSTRACT

A method and apparatus for reducing time needed to initialize a data processing system and to execute applications on the data processing system. In accordance with a preferred embodiment of the present invention, pages for an application are pre-loaded onto a client from a server. The pre-loading of the application includes loading pages that will be required for execution of the application in preparation for hibernation. These pages may include other pages for executable code or data that will be used during execution of the application. Subsequently, the application is executed using the locally stored pages without having to retrieve pages from the server. In addition, an application is provided with an opportunity to prepare itself for hibernation via hibernation notification. For example, the application may read and process files from the server. This processing is done once prior to hibernation and is not required for later executions of the application.

(56) **References Cited**
U.S. PATENT DOCUMENTS

4,885,770 * 12/1989 Croll 379/269
 5,146,568 * 9/1992 Flaherty et al. 716/3
 5,367,688 * 11/1994 Croll 713/2
 5,404,527 * 4/1995 Irwin et al. 709/222
 5,444,850 * 8/1995 Chang 709/222
 5,452,454 9/1995 Basu 395/700
 5,485,609 * 1/1996 Vitter et al. 707/101
 5,530,862 * 6/1996 Wadsworth et al. 713/1
 5,666,293 * 9/1997 Metz et al. 709/220
 5,708,820 * 1/1998 Park et al. 713/323
 5,715,456 * 2/1998 Bennett et al. 713/2
 5,752,042 * 5/1998 Cole et al. 717/11
 5,758,072 * 5/1998 Filepp et al. 709/220
 5,758,165 5/1998 Shuff 717/11
 5,778,443 7/1998 Swanberg et al. 711/162

25 Claims, 4 Drawing Sheets



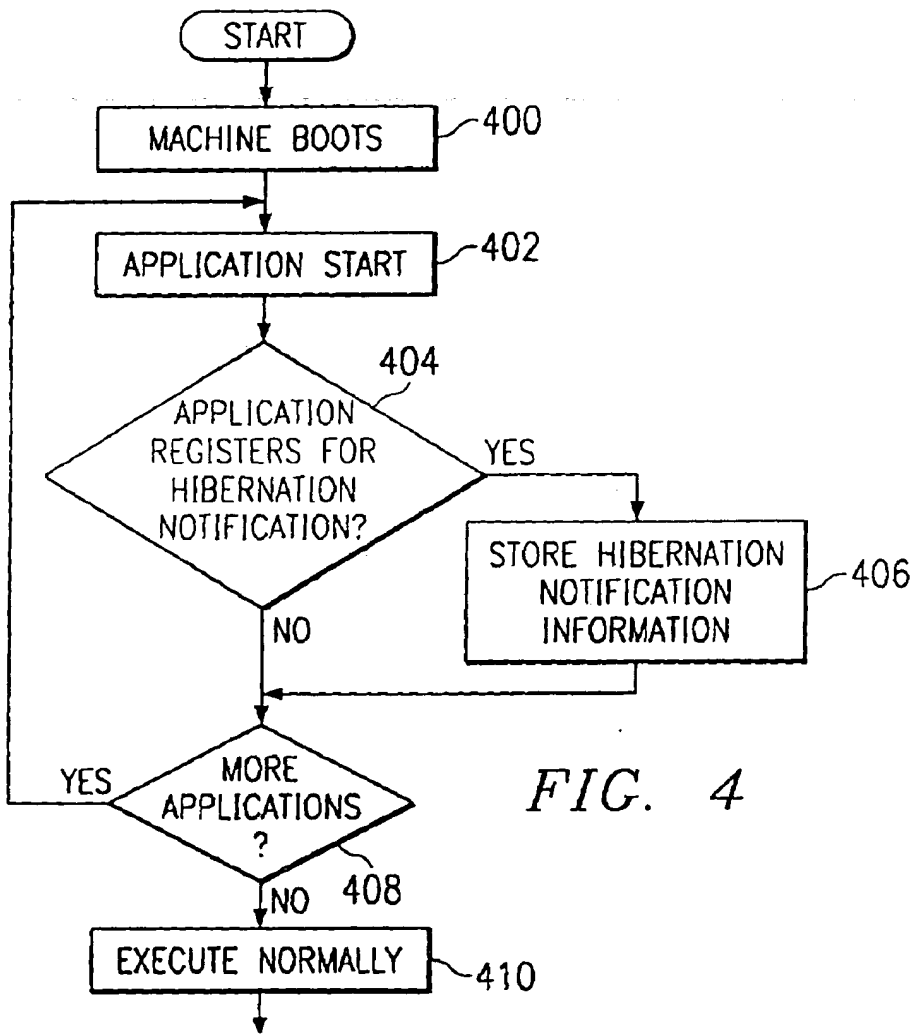


FIG. 4

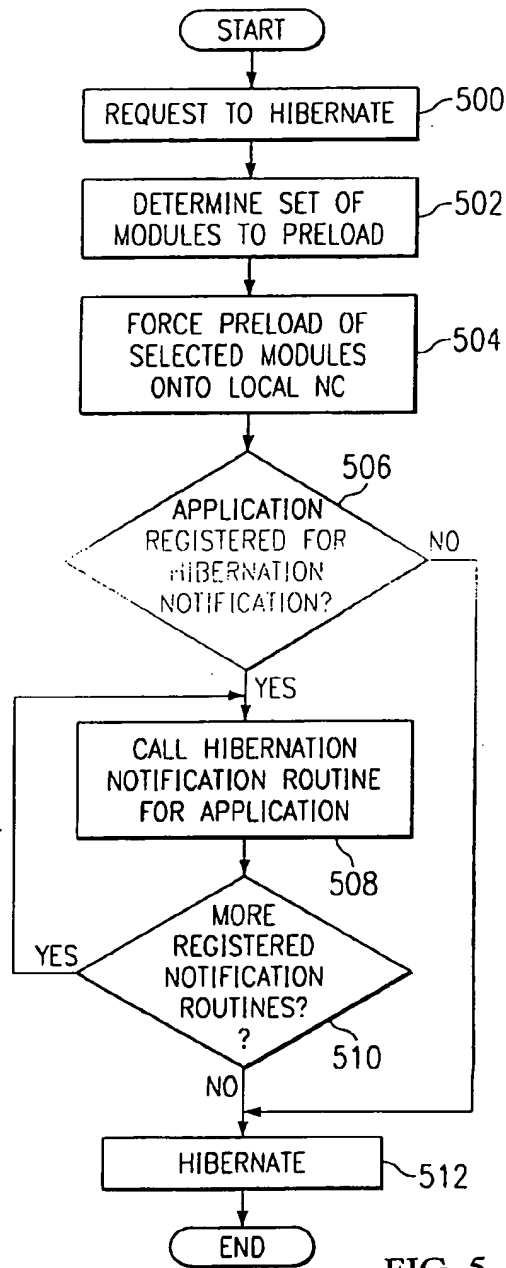
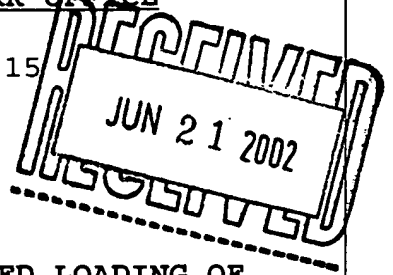
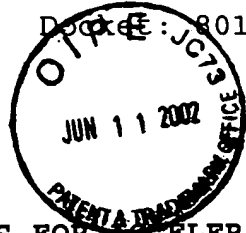


FIG. 5

GM 2182 #4
C
6-21-02

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Fallon et al.
Serial No.: 09/776,267
Filed: February 2, 2001
For: **SYSTEMS AND METHODS FOR ACCELERATED LOADING OF
OPERATING SYSTEMS AND APPLICATION PROGRAMS**



Assistant Commissioner for Patents
Washington, D.C. 20231

RECEIVED
JUN 18 2002
Technology Center 2100

INFORMATION DISCLOSURE STATEMENT

Sir:

Pursuant to Applicant(s) duty of disclosure, the information listed in the attached form PTO-1449 is brought to the attention of the Examiner. Copies of the listed items are enclosed.

The citation of the listed items is not a representation that they constitute a complete or exhaustive listing of the relevant art or that the references are prior art. The items listed are submitted in good faith, but are not intended to substitute for the Examiner's search. It is hoped,

CERTIFICATION UNDER 37 C.F.R. § 1.10

I hereby certify that this New Application Transmittal and the documents referred to as enclosed therein are being deposited with the United States Postal Service on this date June 6, 2002 in an envelope addressed to: Assistant Commissioner for Patents, Washington, D.C. 20231.

Dated: 6/6/02

Frank DeRosa
Frank DeRosa

however, that in addition to apprising the Examiner of these particular items, they will assist in identifying fields of search and in making as full and complete a search as possible.

The filing of this information disclosure statement is not an admission that the information cited herein is, or is considered to be, material to patentability as defined in 37 C.F.R. § 1.56(b).

- This information disclosure statement is being filed within three (3) months of the filing date of this application.

- This information disclosure statement is being filed within three (3) months of the date of entry of the national stage as set forth in 37 C.F.R. § 1.491 in an international application.

- To the best of Applicant(s) knowledge, this information disclosure statement is being filed before the date of mailing of a first Office Action on the merits in connection with this case.

Statement under 37 C.F.R. § 1.97(e):

I hereby state that each item of information contained in this Information Disclosure Statement was cited in a communication from a foreign patent office in a counterpart foreign application not more than 3 months prior to the filing of this statement.

Enclosed herewith is a petition under 37 C.F.R. § 1.97(d)(ii).

Enclosed by check is the petition fee of \$130.00. (37 C.F.R. § 1.17(i)(1))

Please charge the \$130.00 petition fee to Deposit Account No. _____

Enclosed by check is the \$180.00 fee required by 37 C.F.R. § 1.17(p).

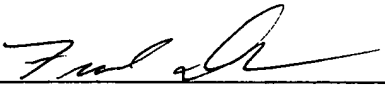
Please charge the \$180.00 fee required by 37 C.F.R. § 1.17(p) to Deposit Account No. _____

[] Please charge any deficiency as well as any other fee(s) which may become due under 37 C.F.R. § 1.16 and/or 1.17 at any time during the pendency of this application, or credit any overpayment of such fee(s) to Deposit Account _____. Also, in the event any extensions of time for responding are required for the pending application(s), please treat this paper as a petition to extend the time as required and charge Deposit Account No. _____ therefor. **TWO (2) COPIES OF THIS SHEET ARE ENCLOSED.**

The claims of the application as now presented are believed to patentably distinguish over the prior art and to be in condition for allowance. Early and favorable consideration of the case is respectfully requested.

Respectfully submitted,

By:


Frank DeRosa
Reg. No. 43,584
Attorney for Applicant(s)

Mailing Address:

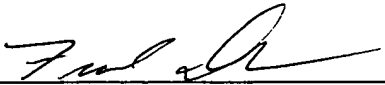
**F. Chau & Associates, LLP
1900 Hempstead Turnpike, Suite 501
East Meadow, New York 11554
(516) 351-0091
(516) 351-0092 (FAX)**

[] Please charge any deficiency as well as any other fee(s) which may become due under 37 C.F.R. § 1.16 and/or 1.17 at any time during the pendency of this application, or credit any overpayment of such fee(s) to Deposit Account _____. Also, in the event any extensions of time for responding are required for the pending application(s), please treat this paper as a petition to extend the time as required and charge Deposit Account No. _____ therefor. **TWO (2) COPIES OF THIS SHEET ARE ENCLOSED.**

The claims of the application as now presented are believed to patentably distinguish over the prior art and to be in condition for allowance. Early and favorable consideration of the case is respectfully requested.

Respectfully submitted,

By:


Frank DeRosa
Reg. No. 43,584
Attorney for Applicant(s)

Mailing Address:

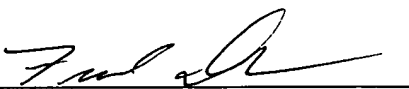
**F. Chau & Associates, LLP
1900 Hempstead Turnpike, Suite 501
East Meadow, New York 11554
(516) 351-0091
(516) 351-0092 (FAX)**

[] Please charge any deficiency as well as any other fee(s) which may become due under 37 C.F.R. § 1.16 and/or 1.17 at any time during the pendency of this application, or credit any overpayment of such fee(s) to Deposit Account _____. Also, in the event any extensions of time for responding are required for the pending application(s), please treat this paper as a petition to extend the time as required and charge Deposit Account No. _____ therefor. **TWO (2) COPIES OF THIS SHEET ARE ENCLOSED.**

The claims of the application as now presented are believed to patentably distinguish over the prior art and to be in condition for allowance. Early and favorable consideration of the case is respectfully requested.

Respectfully submitted,

By:


Frank DeRosa
Reg. No. 43,584
Attorney for Applicant(s)

Mailing Address:

**F. Chau & Associates, LLP
1900 Hempstead Turnpike, Suite 501
East Meadow, New York 11554
(516) 351-0091
(516) 351-0092 (FAX)**



19 BUNDESREPUBLIK
DEUTSCHLAND



DEUTSCHES
PATENTAMT

12 **Offenlegungsschrift**
10 **DE 41 27 518 A 1**

51 Int. Cl.5:
G 06 F 13/10
G 06 F 9/445

21 Aktenzeichen: P 41 27 518.7
22 Anmeldetag: 20. 8. 91
43 Offenlegungstag: 27. 2. 92

DE 41 27 518 A 1

30 Unionspriorität: 32 33 31
21.08.90 JP 2-219357

71 Anmelder:
Tokico Ltd., Kawasaki, Kanagawa, JP

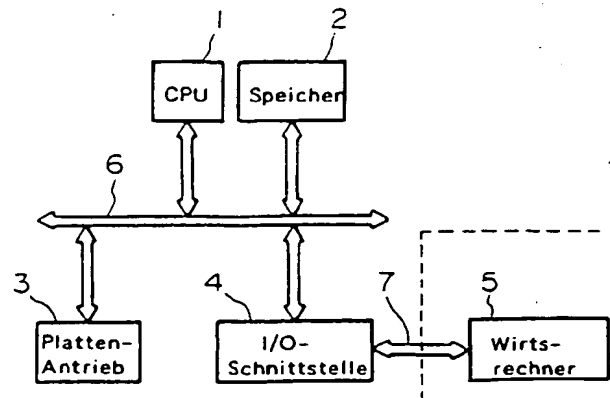
74 Vertreter:
Reinhard, H., Dipl.-Chem. Dr.rer.nat.; Skuhra, U.,
Dipl.-Ing.; Weise, R., Dipl.-Ing., Pat.-Anwälte, 8000
München

72 Erfinder:
Urabe, Masayuki, Tokio/Tokyo, JP

Prüfungsantrag gem. § 44 PatG ist gestellt

54 Magnetplattenspeichervorrichtung

57 Es wird eine Magnetplattenspeichervorrichtung angegeben, die dazu geeignet ist, mit einem Wirtsrechner (5) verbunden zu werden und als äußerer Speicher für denselben verwendet zu werden. Die Vorrichtung besteht im wesentlichen aus (A) wenigstens einer Magnetplatte zum Speichern von Daten, die einen Anlaufdatenbereich aufweist, in dem Anlaufdaten zum Inbetriebsetzen eines Systems gespeichert sind, das aus dem Wirtsrechner (5) und der Magnetplattenspeichervorrichtung besteht; (B) einem Pufferspeicher (2) zum vorübergehenden Speichern von Daten, die von der wenigstens einen Magnetplatte/Wirtsrechner (5) übertragen werden, und (C) einer Datenübertragungsschaltung zum Übertragen der Anlaufdaten aus dem Anlaufdatenbereich in den Pufferspeicher (2) zum Zeitpunkt des Anlaufens des Systems.



DE 41 27 518 A 1

Best Available Copy

Beschreibung

Die vorliegende Erfindung bezieht sich auf Magnetplattenspeichervorrichtungen und speziell auf Magnetplattenantriebe, die in der Lage sind, sehr kurze Anlaufzeiten für Speichersysteme zu erzielen.

Es sind in letzter Zeit verschiedene intelligente Magnetplattenantriebs- und Steuerungskombinationen zu dem Zweck entwickelt worden, die Arbeitsbelastung von Wirtsrechnern zu vermindern und den Eingabe/Ausgabe-Durchsatz und die Datenverarbeitungsgeschwindigkeiten von Computersystemen zu steigern.

Fig. 5 ist ein Blockschaltbild, das den Gesamtaufbau einer solchen konventionellen, intelligenten Magnetplattenantriebs- und Steuervorrichtung zeigt. Wie man in dieser Figur erkennt, besteht eine solche Vorrichtung im wesentlichen aus einer CPU 1, einem Speicher 2, einem Plattenantrieb 3 und einer Eingabe/Ausgabe-(I/O-) Schnittstelle 4. Die erwähnte CPU 1 ist zur Steuerung jeder Komponente des Magnetplattenantriebs vorgesehen. Der Speicher 2 besteht aus einem Festspeicher ROM, der verschiedene Arten von Steuerprogrammen und dgl. speichert, die die CPU 1 benötigt, um die ihr zugewiesene Funktion zu erfüllen, und einen Speicher mit wahlfreiem Zugriff RAM, der vorübergehend verschiedene Arten von Daten speichert. Insbesondere besteht der RAM aus einem Lese-Schreib-Pufferbereich zum vorübergehenden Speichern von Daten, die von Magnetplatten (nicht dargestellt) abgelesen und auf solche eingeschrieben werden sollen. Der Plattenantrieb 3 enthält Schaltungen, die bewirken, daß Magnetköpfe (nicht dargestellt) eine Zielspur suchen, sowie Schaltungen zum Steuern der Rotation der Platte. Die Verbindung zwischen der CPU 1 und dem Wirtsrechner 5 findet mittels der I/O-Schnittstelle 4 statt. Mittels eines internen Bus 6 ist die CPU 1 mit dem Speicher 2, dem Plattenantrieb 3 und der I/O-Schnittstelle 4 verbunden. Die I/O-Schnittstelle 4 ist mit dem Wirtsrechner 5 mittels eines äußeren Bus 7 verbunden.

Bei einer solchen Art von Magnetplattenantriebsvorrichtung und speziell bei solchen, die das sogenannte eingebaute Servoverfahren verwenden, umfaßt der Anlaufvorgang der Vorrichtung verschiedene Initialisierungsvorgänge mit folgenden Schritten: a) Testen des Speichers 2; b) Starten des Spindelmotors (nicht dargestellt) unter Regelung der Drehzahl desselben; und c) Auslesen verschiedener Arten von Systemanlaufparametern.

Nach den Initialisierungsvorgängen sendet die CPU 1 ein Bereitschaftssignalsignal an den Wirtsrechner 5, wodurch dieser darüber informiert wird, daß die CPU 1 zur Datenverarbeitung bereit ist.

Bei der beschriebenen bekannten Vorrichtung müssen jedoch bei der Inbetriebnahme des Rechnersystems die Anlaufdaten von der Platte abgelesen werden, was zu einer langen Anlaufzeit des Rechnersystems führt und daher den anfänglichen Durchsatz des Rechnersystems begrenzt.

Außerdem sind kurz nach der Aufnahme der Stromversorgung des Magnetplattenantriebs die Betriebsdaten noch nicht in den Lese-Schreib-Pufferbereich eingespeichert.

In Anbetracht der vorgenannten Tatsachen ist es Aufgabe der Erfindung, eine Magnetplattenspeichervorrichtung anzugeben, die in der Lage ist, die Anlaufzeit für das Rechnersystem abzukürzen.

Gemäß einem ersten Aspekt der vorliegenden Erfindung ist eine Magnetplattenspeichervorrichtung vorge-

sehen, die geeignet ist, mit einem Wirtsrechner (5) verbunden zu werden und als äußere Speichervorrichtung desselben verwendet zu werden, mit folgenden Merkmalen:

- a) wenigstens eine Magnetplatte zum Speichern von Daten, die einen Anlaufdatenbereich umfaßt, in dem Anlaufdaten zum Anlaufen eines Systems gespeichert sind, das aus dem Wirtsrechner (5) und der Magnetplattenspeichervorrichtung besteht;
- b) einen Pufferspeicher (2) zum vorübergehenden Speichern von Daten, die von der wenigstens einen Magnetplatte/Wirtsrechner (5) übertragen werden;
- c) eine Datenübertragungseinrichtung (1) zum Übertragen der Anlaufdaten aus dem Anlaufdatenbereich in den Pufferspeicher (2) zum Zeitpunkt des Anlaufens des Systems.

Gemäß einem zweiten Aspekt der Erfindung ist ein Magnetplattenspeicher vorgesehen, der dazu geeignet ist, mit einem Wirtsrechner (5) verbunden und als eine äußere Speichervorrichtung desselben verwendet zu werden, gekennzeichnet durch die folgenden Merkmale:

- a) wenigstens eine Magnetplatte zum Speichern von Daten, die einen Anlaufdatenbereich und einen Plattentreibersystembereich aufweist, wobei der Anlaufdatenbereich Anlaufdaten zum Anlaufen eines Systems enthält, das aus dem Wirtsrechner (5) und der Magnetplattenspeichervorrichtung besteht, und wobei der Plattentreibersystembereich die Adreßdaten des Anlaufdatenbereichs speichert;
- b) einen Pufferspeicher (2) zum vorübergehenden Speichern von Daten, die von der wenigstens einen Magnetplatte/Wirtsrechner (5) übertragen werden;
- c) eine Datenübertragungseinrichtung (1) zum Übertragen der Anlaufdaten aus dem Anlaufdatenbereich in den Pufferspeicher (2) beim Anlaufvorgang des Systems; und
- d) eine Schreibeinrichtung (3) zum Einschreiben von Adreßdaten des Anlaufdatenbereichs in den Plattentreibersystembereich zum Zeitpunkt, zu dem der Anlaufdatenbereich bestimmt ist in Abhängigkeit von einem Befehl vom Wirtsrechner (5).

Bei der vorliegenden Erfindung werden die Anlaufdaten automatisch vorausgelesen, wenn zum Plattentreibersystem Strom zugeführt worden ist, auf deren Grundlage der Anlaufvorgang ausgeführt wird, was zu einer kurzen Anlaufzeit für das Rechnersystem führt, so daß gleich zu Anfang ein hoher Durchsatz des Rechners erzielt wird.

Die Erfindung wird nachfolgend unter Bezugnahme auf die Zeichnungen näher erläutert.

Fig. 1 zeigt ein Flußdiagramm des Anlaufvorgangs der CPU in der Magnetplattenantriebsvorrichtung in Übereinstimmung mit einer bevorzugten Ausführungsform der vorliegenden Erfindung.

Fig. 2 zeigt eine Darstellung eines bevorzugten Beispiels eines zulieferspezifischen Befehls, der in einer bevorzugten Ausführung der Erfindung verwendet wird.

Fig. 3 ist ein Flußdiagramm der Speicherverarbeitung der vorausgelesenen Adresse der CPU in der Magnetplattenantriebsvorrichtung in Übereinstimmung mit einer bevorzugten Ausführungsform der vorliegenden Erfindung.

Fig. 4 ist eine Darstellung eines vorteilhaften Musters

des Formats von Daten, die von dem Wirtsrechner übertragen werden in Übereinstimmung mit einer bevorzugten Ausführungsform der Erfindung.

Fig. 5 ist ein Blockschaltbild des Gesamtaufbaus einer Magnetplattenantriebsvorrichtung.

Bei der vorliegenden Erfindung ist der Gesamtaufbau des Magnetplattenantriebs vergleichbar dem nach dem Stand der Technik gemäß Fig. 5, weshalb auf eine Wiederholung der Erläuterung verzichtet wird.

Die Plattenantriebsvorrichtung dieses Beispiels unterscheidet sich indessen von konventionellen Plattenantriebsvorrichtungen dahingehend wesentlich, daß sie die folgenden Funktionen enthält:

- a) Einschreiben der Adressen des Anlaufdatenbereiches als Vorausleseadressen in einen Plattenantriebssystembereich zum Zeitpunkt, zu dem der Anlaufdatenbereich zuerst bezeichnet wird, in Abhängigkeit von einem Befehl vom Wirtsrechner 5, und
- b) Auslesen der Anlaufdaten an dem Anlaufdatenbereich in der Platte und Übertragen derselben zu dem oben erwähnten Lese-Schreib-Pufferbereich zum Zeitpunkt des Anlaufs des Systems.

Nachfolgend wird die Speicherverarbeitung der CPU 1 zum Einschreiben vorausgelesener Adressen in den Plattenantriebssystembereich unter Bezugnahme auf die Fig. 2 bis 4 erläutert.

Damit die CPU 1 die Speicherverarbeitung ausführen kann, muß ein Speicherbefehl vom Wirtsrechner 5 abgegeben werden. Bei dieser Speicherverarbeitung wird ein zulieferspezifischer Befehl, der in Fig. 2 gezeigt ist, verwendet, der vom Benutzer frei neu festgelegt werden kann.

Wenn die CPU 1 den oben beschriebenen Speicherbefehl vom Wirtsrechner 5 entgegengenommen hat, beginnt sie den Betrieb gemäß dem im Flußdiagramm von Fig. 3 dargestellten Ablauf. Zunächst empfängt im Schritt SA1 die CPU 1 Daten, die vom Wirtsrechner 5 übertragen werden, wodurch die CPU 1 mit der vorausgelesenen Adresse (sog. Teilnehmeradresse) LBA und mit der vorausgelesenen Sektorzählung versorgt wird. Wenn hier der Lese-Schreib-Puffer eine große Kapazität hat, dann ist die Vorgabe mehrerer Adressen möglich.

Fig. 4 ist eine Darstellung, die ein bevorzugtes Formatmuster von Daten zeigt, die vom Wirtsrechner 5 übertragen werden und das von der CPU 1 im Schritt SA1 empfangen wird.

Wenn die CPU 1 die oben beschriebenen Daten, die die vorausgelesene Adresse LBA und die vorausgelesene Sektorzählung vom Wirtsrechner 5 empfangen hat, geht sie zum Schritt SA2 über und schreibt dort die vorausgelesene Adresse LBA und die vorausgelesene Sektorzählung in den Plattenantriebssystembereich ein. Diese Routine endet hinter dem Schritt SA2.

Nachfolgend wird die Anlaufverarbeitung der CPU 1 zum Inbetriebsetzen des Plattenantriebssystems unter Bezugnahme auf Fig. 1 erläutert.

Wenn die Stromversorgung zum Plattenantriebssystem eingeschaltet worden ist, beginnt die CPU 1, beim Schritt SP1 und führt hier verschiedene Arten von Initialisierungen durch, wie beispielsweise die Einstellung der Betriebsart, des Stapels und dgl. Sodann geht die Routine zum Schritt SP2 über, wo die Gültigkeit der im Speicher 2 gespeicherten Daten geprüft wird, d. h. eine Wahrheitsprüfung für den RAM 1 und eine Summen-

prüfung für den ROM werden ausgeführt. Wenn hier der Speicher 2 in Ordnung ist, geht die Routine zum Schritt SA3 über, bei dem der Betrieb des Spindelmotors eingeleitet und geregelt wird, so daß Magnetplatten mit einer vorbestimmten Geschwindigkeit rotieren. Nachdem die Platte angelaufen ist, geht die CPU 1 zum Schritt SA4 über, wo die vorausgelesene Adresse LBA und die vorausgelesene Sektorzählung aus dem Plattenantriebssystembereich ausgelesen werden.

Anschließend geht die Routine zum Schritt SP5 über, wo die Anlaufdaten aus dem Anlaufdatenbereich ausgelesen werden auf der Grundlage der vorausgelesenen Adresse LBA und der vorausgelesenen Sektorzählung im Schritt SP4, die zu dem Lese-Schreib-Pufferbereich übertragen wurden. Hierbei können verschiedene Parameter, die für die Betriebsart des Magnetplattenantriebs, erforderlich sind, in den Anlaufdaten enthalten sein.

Außerdem wird die Größe der vorausgelesenen Daten in Übereinstimmung mit der Kapazität des Lese-Schreib-Pufferbereichs bestimmt.

Nach dem Übertragen der Anlaufdaten im Schritt SP5 geht die Routine zum Schritt SP6 über, bei dem das Kennzeichen zum Anzeigen, das Daten gegenwärtig in dem Lese-Schreib-Pufferbereich und die Adresse der ausgelesenen Daten (im vorliegenden Beispiel die Teilnehmeradresse) in den Lese-Schreib-Pufferbereich gesetzt werden.

Als nächstes geht die Routine zum Schritt SP7 über, bei dem die CPU 1 ein Bereitschaftszustandssignal zum Wirtsrechner 5 sendet, wodurch dieser darüber informiert ist, daß die CPU 1 zur Eingabe/Ausgabe-Verarbeitung bereit ist. Sodann wartet die CPU 1 auf einen Befehl vom Wirtsrechner 5.

Mit dem oben beschriebenen Aufbau werden die Anlaufdaten automatisch vorausgelesen, wenn die Stromversorgung des Plattenantriebssystems aufgenommen worden ist, und auf deren Grundlage wird die Anlaufverarbeitung ausgeführt, was zu einer kurzen Anlaufzeit für das Rechnersystem führt und dadurch der anfängliche Durchsatz des Rechnersystems erzielt wird.

Außerdem, weil der zulieferspezifische Befehl, wie oben beschrieben, bei der Speicherverarbeitung der vorausgelesenen Adresse verwendet wird, kann der Benutzer die Vorausleseadresse frei bestimmen.

Weiterhin kann die Plattenantriebsvorrichtung nach der vorliegenden Erfindung, bei der die Speicherkapazität des oben beschriebenen Lese-Schreib-Pufferbereichs groß ist, eine kürzere Anlaufzeit für das System erzielen, weil die vorausgelesene Sektorzählung vergrößert werden kann.

Patentansprüche

1. Magnetplattenspeichervorrichtung, die zur Verbindung mit einem Wirtsrechner (5) geeignet ist und als äußere Speichervorrichtung desselben verwendet werden kann, **gekennzeichnet durch** die folgenden Merkmale:

- a) wenigstens eine Magnetplatte zum Speichern von Daten, die einen Anlaufdatenbereich enthält, in den Anlaufdaten zum Anlaufen eines Systems gespeichert werden, das aus dem Wirtsrechner (5) und der Magnetplattenspeichervorrichtung besteht;
- b) einen Pufferspeicher (2) zum vorübergehenden Speichern von Daten, die von der wenigstens einen Magnetplatte/Wirtsrechner (5)

- übertragen werden; und
 c) eine Datenübertragungseinrichtung (1) zum Übertragen der Anlaufdaten aus dem Anlaufdatenbereich in den Pufferspeicher (2) zum Zeitpunkt des Anlaufvorgangs des Systems. 5
2. Magnetplattenspeichervorrichtung nach Anspruch 1, dadurch gekennzeichnet, daß der Inhalt der Anlaufdaten frei in Abhängigkeit von einem Befehl bestimmt werden kann, der vom Benutzer frei vorgegeben werden kann. 10
3. Magnetplattenspeichervorrichtung nach Anspruch 1, dadurch gekennzeichnet, daß der Anlaufdatenbereich in Abhängigkeit von einem Befehl frei bestimmt werden kann, der vom Benutzer frei definiert werden kann. 15
4. Magnetplattenspeichervorrichtung nach Anspruch 1, dadurch gekennzeichnet, daß die Datenübertragungseinrichtung (1) eine Leseeinrichtung zum Auslesen der Anlaufdaten aus dem Anlaufdatenbereich zum Zeitpunkt des Anlaufvorgangs des Systems enthält. 20
5. Magnetplattenspeichervorrichtung nach Anspruch 1, dadurch gekennzeichnet, daß sie ein Kennzeichen enthält, das anzeigt, daß Anlaufdaten gegenwärtig im Pufferspeicher (2) gespeichert sind. 25
6. Magnetplattenspeichervorrichtung nach Anspruch 1, dadurch gekennzeichnet, daß die wenigstens eine Magnetplatte weiterhin einen Plattenantriebssystembereich enthält, in den Adreßdaten aus dem Anlaufdatenbereich gespeichert werden. 30
7. Magnetplattenspeichervorrichtung nach Anspruch 1, dadurch gekennzeichnet, daß die Adreßdaten die Adresse und die Sektorzählung des Anlaufdatenbereichs enthalten. 35
8. Magnetplattenspeichervorrichtung nach Anspruch 6, dadurch gekennzeichnet, daß die Datenübertragungseinrichtung (1) die Anlaufdaten aus dem Anlaufdatenbereich in den Pufferspeicher (2) auf der Grundlage der Adreßdaten überträgt, die in dem Plattenantriebssystembereich gespeichert sind, zum Zeitpunkt des Anlaufvorgangs des Systems. 40
9. Magnetplattenspeicher, der zur Verbindung mit einem Wirtsrechner (5) geeignet ist und als äußere Speichervorrichtung für denselben verwendbar ist, gekennzeichnet durch folgende Merkmale: 45
- a) wenigstens eine Magnetplatte zum Speichern von Daten, die einen Anlaufdatenbereich und einen Plattenantriebssystembereich enthält, wobei der Anlaufdatenbereich Anlaufdaten zum Inbetriebsetzen eines Systems enthält, das aus dem Wirtsrechner (5) und der Magnetplattenspeichervorrichtung besteht, wobei der Plattenantriebssystembereich die Adreßdaten des Anlaufdatenbereichs speichert; 50
- b) einen Pufferspeicher (2) zum vorübergehenden Speichern von Daten, die von der wenigstens einen Magnetplatte/Wirtsrechner (5) übertragen werden; 60
- c) eine Datenübertragungseinrichtung (1) zum Übertragen der Anlaufdaten aus dem Anlaufdatenbereich in den Pufferspeicher (2) zum Zeitpunkt des Anlaufs des Systems; und
- d) eine Schreibeinrichtung (3) zum Einschreiben einer Adreßdate aus dem Anlaufdatenbereich in den Plattenantriebssystembereich zu einem Zeitpunkt, zu welchem der Anlaufda-

- tenbereich bezeichnet wird, in Abhängigkeit von einem Befehl vom Wirtsrechner (5).
10. Magnetplattenspeichervorrichtung nach Anspruch 9, bei der der Inhalt der Anlaufdaten frei in Abhängigkeit von dem Befehl bestimmt werden kann, wobei der Befehl vom Benutzer frei definiert werden kann.
11. Magnetplattenspeichervorrichtung nach Anspruch 9, dadurch gekennzeichnet, daß der Anlaufdatenbereich frei in Abhängigkeit von dem Befehl bezeichnet werden kann, wobei der Befehl vom Benutzer frei definiert werden kann.
12. Magnetplattenspeichervorrichtung nach Anspruch 9, dadurch gekennzeichnet, daß die genannte Adreßdate die Adresse und die Sektorzählung des Anlaufdatenbereichs enthält.
13. Magnetplattenspeichervorrichtung nach Anspruch 9, dadurch gekennzeichnet, daß die Datenübertragungseinrichtung (1) eine Leseeinrichtung zum Auslesen der Anlaufdaten aus dem Anlaufdatenbereich zum Zeitpunkt eines Anlaufvorgangs des Systems enthält.
14. Magnetplattenspeichervorrichtung nach Anspruch 9, dadurch gekennzeichnet, daß die Datenübertragungseinrichtung (1) die Anlaufdaten aus dem Anlaufdatenbereich in den Pufferspeicher (2) auf der Grundlage der Adreßdaten überträgt, die in dem Plattenantriebssystembereich gespeichert sind, zum Zeitpunkt eines Anlaufvorgangs des Systems.
15. Magnetplattenspeichervorrichtung nach Anspruch 9, dadurch gekennzeichnet, daß sie weiterhin ein Kennzeichen enthält zum Angeben, daß Anlaufdaten gegenwärtig im Pufferspeicher (2) gespeichert sind.
16. Magnetplattenspeichervorrichtung nach einem der Ansprüche 1 und 9, dadurch gekennzeichnet, daß der Pufferspeicher (2) ein Speicher mit wahlfreiem Zugriff RAM ist.
17. Magnetplattenspeichervorrichtung nach einem der Ansprüche 1 und 9, dadurch gekennzeichnet, daß er weiterhin eine Eingabe/ Ausgabe-Schnittstelle (4) zur Verbindung mit dem Wirtsrechner (5) aufweist.

Hierzu 3 Seite(n) Zeichnungen

— Leerseite —

FIG. 1

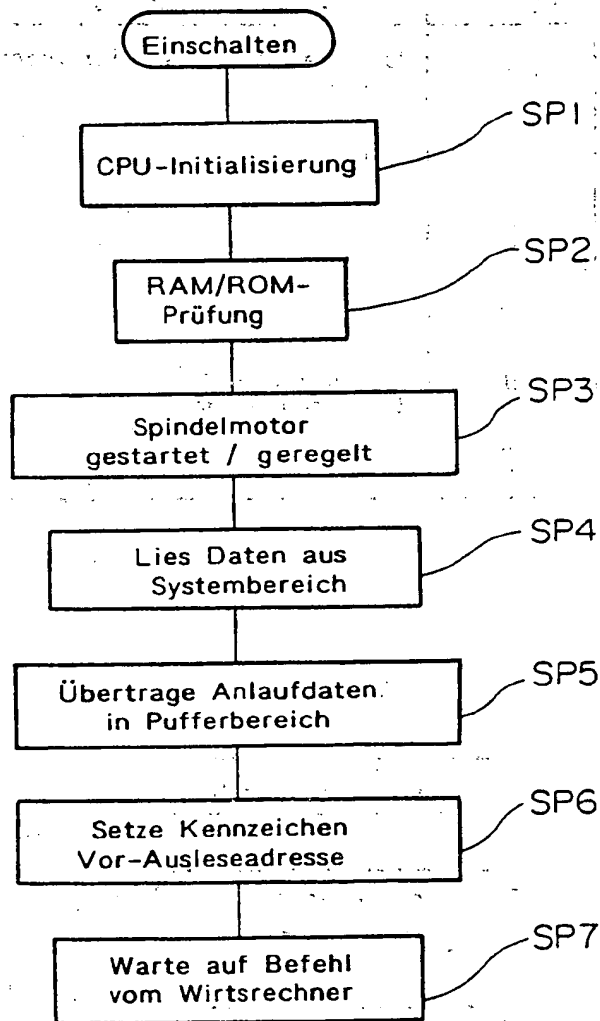


FIG.2

Byte \ bit	7	6	5	4	3	2	1	0
0	Die Definition des liverspez. Befehls							
1	L U N							
2								
3								
4	Die Länge der Übertragungsdaten							
5								

FIG.3

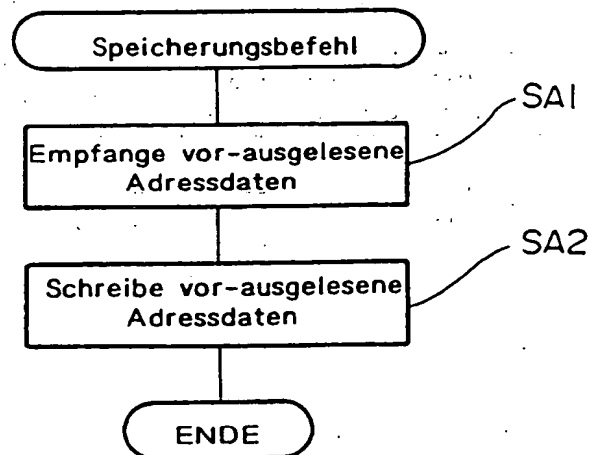


FIG. 4

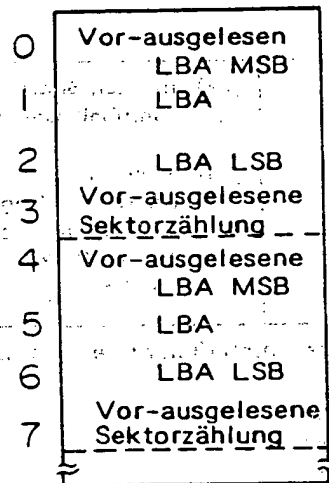
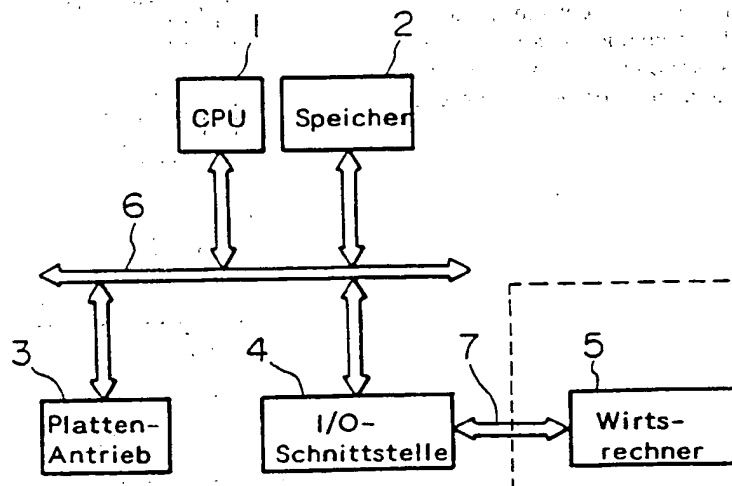


FIG. 5



**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

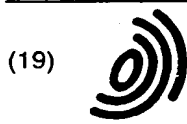
Defects in the images include but are not limited to the items checked:

- BLACK BORDERS**
- IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- FADED TEXT OR DRAWING**
- BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- SKEWED/SLANTED IMAGES**
- COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- GRAY SCALE DOCUMENTS**
- LINES OR MARKS ON ORIGINAL DOCUMENT**
- REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.

This Page Blank (uspto)



Europäisches Patentamt
European Patent Office
Office européen des brevets



(11) EP 0 718 751 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
26.06.1996 Bulletin 1996/26

(51) Int Cl.⁶: G06F 3/06

(21) Application number: 95309160.0

(22) Date of filing: 15.12.1995

(84) Designated Contracting States:
DE FR GB

(72) Inventor: Shafe, Mathew K.
Campbell, California 95008 (US)

(30) Priority: 23.12.1994 US 363464

(74) Representative: Moss, Robert Douglas
IBM United Kingdom Limited
Intellectual Property Department
Hursley Park
Winchester Hampshire SO21 2JN (GB)

(71) Applicant: International Business Machines Corporation
Armonk, N.Y. 10504 (US)

(54) Electronic circuit apparatus employing small disk drive with reconfigurable interface

(57) An electronic circuit apparatus is provided for electronic circuits and devices, comprising a component disk drive, a disk controller, and a programmable interface for dynamically adapting the component drive to communicate over a predetermined bus architecture to an external application. The interface is programmed by the disk controller using one of a library of microcode sets stored on the component drive. Alternatively, a microprocessor independent of the disk controller programs the interface from microcode stored in solid state memory. In an alternative embodiment, the apparatus further comprises an application circuit and the programmable interface adapts the component drive for use by the circuit. In another embodiment, the apparatus comprises a subcircuit that communicates with an external application over a predetermined bus architecture, and the programmable interface adapts the subcircuit for such communication.

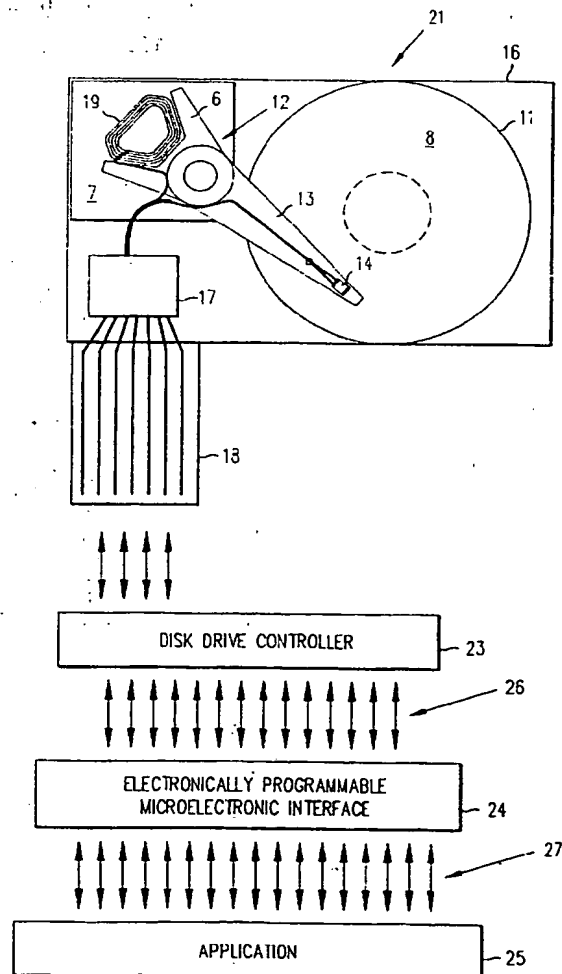


FIG. 1a

Best Available Copy

EP 0 718 751 A2

Description

Field of the Invention

The present invention relates generally to magnetic disk drive interfaces, and more particularly to an electronic circuit architecture including a component-sized disk drive.

Background of the Invention

Magnetic disk drives have been used broadly as peripheral storage devices. As disk drives grow smaller, they are also becoming practical as a cost-effective replacement for solid state memory in a variety of electronic devices, e.g. cameras, fax machines, cellular phones, modems, pagers, handheld computing devices, printers, and copiers.

Magnetic disk drives and other peripheral devices are generally designed to meet one of a number of industry standard bus architectures to assure compatibility with a host system. These include, for example, the small computer serial interface (SCSI), serial storage architecture (SSA) and the integrated drive electronics (IDE) interface. Each type of peripheral bus architecture defines its own unique set of communication protocols. The host system, which may include a microprocessor, memory devices, and other essential circuits, communicates with these elements via a system bus. The system bus may comprise an industry standard architecture (ISA) bus, or a microchannel, for example. The system bus similarly has its own set of communications protocols. As such, the host system requires a device adapter to interface between the system bus and the peripheral bus.

An electronic circuit implementing a component-sized magnetic disk drive in lieu of solid state memory must also provide means for allowing the disk drive to communicate with the circuit application. Moreover, the circuit may be housed in a card enclosure and plugged into an application external to the circuit. It then requires means for interfacing with the external application.

For example, circuits embodying fax machines, modems, or disk drives, and circuits related to the operation of cellular phones and cameras are presently being housed in credit-card sized formats of predefined dimensions that plug into a compatible socket of a computing device. Three standard formats that have emerged for such credit-card-type applications are the PCMCIA formats. A "type III" card measures 10.5 mm in height, 85.6 mm in length and 54 mm in width. The dimensions of a "type II" card are approximately 5 mm high X 85.6 mm long X 54 mm wide. A "type I" card is a modest 3.3 mm high X 85.6 mm long X 54 mm wide. A PCMCIA card includes a PCMCIA bus for communicating with the computing device. Thus circuit applications residing in PCMCIA-type cards will include some type of interface to the PCMCIA bus.

Some electronic devices are incorporated into a communication network, e.g. via a phone line. These devices must conform to industry standard communications protocols such as asynchronous transfer mode (ATM), integrated services digital networks (ISDN), RS-232, RS-422, V.35, and V.42. Wireless communications devices such as cellular phones and electronic pagers must also be adapted to meet industry standard communications protocols. Computer peripherals such as printers and printed circuit cards similarly require means for interfacing to the host system. Each of these circuit application is a candidate for implementing a component disk drive in lieu of solid state memory, and as such, would require a customized interface between the disk drive and the application.

To add flexibility to an electronic device incorporating a component disk drive, it is desirable to include an adaptable interface within the device circuit, enabling it to conform to more than one communication or host system protocol. A number of U.S. patents discuss the use of an adaptable interface between two systems having different communications protocols. For example, U.S. Patent No. 5,111,423 discloses a programmable interface, i.e. an EEPROM, inserted between a printed circuit card and a host system that is configurable for adapting a variety of printed circuit card applications to the same host system. U.S. Patent No. 4,899,306 describes a test interface circuit having an adaptable interface, i.e. random access memory, for adapting the test circuit to different types of host systems. Another testing device with an adaptable interface comprising programmable gate arrays is described in U.S. Patent No. 5,121,342. The testing device includes a microprocessor for receiving downloaded software from a removable floppy peripheral disk drive device and for selectively configuring the programmable gate arrays to a variety of communication protocols (e.g. RS-232, RS-422, V.35 and ISDN). U.S. Patent No. 5,243,273 describes a testing device for serial communications cards that is configurable to a plurality of communications protocols.

However, to date, an electronic circuit apparatus has not been implemented that includes a component-level disk drive that can be adapted to different types of communication protocols.

Summary of the Invention

Accordingly, a first embodiment of the present invention provides an electronic apparatus comprising a component disk drive for use as local storage by an electronic circuit in lieu of solid state memory, a hard disk controller (HDC) for controlling the low level operations of the drive, and a programmable interface for enabling communication between the component drive and the electronic circuit. The interface is programmed by the HDC. Alternatively, a simple central processing unit is provided in the apparatus for configuring the interface. The HDC resides in the component disk drive, in an in-

tegrated circuit chip external to the component drive, or within the programmable logic unit (i.e., the PLU is a microprocessor with programmable functions). The microcode used for programming the interface is stored in the component drive. Alternatively, a solid state memory component is provided in the apparatus for storing the microcode.

A second embodiment of the present invention is an electronic device comprising a component disk drive, an electronic circuit implementing the component disk drive in lieu of solid state memory as local storage, and a programmable interface provided therebetween for allowing the component drive to communicate with the circuit.

A third embodiment of the present invention comprises an electronic device that communicates with an application external to the device, comprising a component disk drive, an electronic circuit, and a programmable interface provided between the electronic circuit and the external application to enable communication therebetween.

A primary advantage of the present invention is to provide a component disk drive that provides local storage for an electronic circuit in lieu of solid state memory, and a programmable interface for adapting the component drive to the electronic circuit.

Another advantage of the present invention is to provide an electronic device comprising a component disk drive, an electronic circuit implementing the component disk drive in lieu of solid state memory as local storage, and a programmable interface provided therebetween for allowing the component drive to communicate with the circuit.

Another advantage of the present invention is to provide an electronic device that communicates with an application external to the device, comprising a component disk drive, an electronic circuit, and a programmable interface provided between the electronic circuit and the external application to enable communication therebetween.

Another advantage of the present invention is to provide a card-based electronic device having an electronic circuit, a component disk drive, and a programmable interface for adapting the component drive to the electronic circuit, e.g. for PCMCIA-type formats.

Each of the embodiments disclosed may be housed in a card-type enclosure such as a PCMCIA-type card.

The component disk drive preferably comprises a single disk with a diameter of no more than 1.3 inches and a single recording surface mounted directly to a rotatable flat motor having a diameter of up to 1.3 inches. The component drive further comprises a single suspension carrying at least one transducer for writing and retrieving data from the disk, and a parking zone at the center of the recording surface for parking the transducer during periods of inactivity or non-operation to provide high shock resistance.

Brief Description of the Drawing

The foregoing and other objects, features and advantages of the present invention will be apparent from the following detailed description of the preferred embodiments of the invention, and from the accompanying figures of the drawing:

Figs. 1(a) and 1(b) are functional block diagrams of the electronic circuit apparatus according to the present invention;

Fig. 2 is the electronic circuit apparatus of the present invention residing on an electronic circuit board;

Fig. 3 is a perspective view of a general card-type electronic circuit enclosure;

Fig. 4 is a first embodiment of the electronic circuit apparatus of the present invention in a card enclosure;

Figs. 5(a)-5(d) show side and top views of the component disk drive for use in the preferred embodiment of the electronic circuit apparatus of the present invention

Fig. 6 is a second embodiment of the electronic circuit apparatus of the present invention in a card enclosure; and

Fig. 7 is a third embodiment of the electronic circuit apparatus of the present invention in a card enclosure.

Description of the Preferred Embodiment

The present invention will now be described with reference to Fig. 1(a), which is a functional block diagram of the proposed electronic circuit apparatus. It comprises a component disk drive 21, a disk drive controller 23 (also commonly referred to as a hard disk controller or HDC), and an electronically programmable microelectronic interface 24 for adapting the component disk drive 21 and controller 23 to an application 25.

A disk drive generally comprises a disk 11 having at least one recording surface 8 for storing information, means such as a motor (not shown) for rotating the disk, an actuator assembly 12, arm electronics 17, and a housing 16. The actuator assembly 12 generally comprises a voice coil motor (VCM); an actuator arm 6, and at least one suspension 13 connected to the arm 6 and supporting a transducer adjacent 14 the recording surfaces 8 of the disk 11. The transducer 14 is held in close proximity to the disk surface by the combination of a downward force (relative to the disk surface) from the suspension 13 and an upward force caused by air flow

generated from the rotation of the disk 11. If the downward force exceeds the upward force, the transducer will come into contact with the disk surface.

The VCM comprises an inductive coil 19 disposed between an upper magnet (not shown) and a lower magnetic plate 7. The arm electronics 17 transmits electrical positioning current to the coil 19. The current signal induces a magnetic flux around the coil for repulsing and attracting the magnet and magnetic plate 7. The repulsing and attracting forces provide movement of the actuator arm in a plane substantially parallel to the recording surface 8, causing the suspension 13 to move along an arcuate path over the surface 8.

Data is generally recorded on concentric tracks of the recording surface 8. The disk region or track having the largest diameter is referred to as the outer diameter (OD) of the disk, and the region or track nearest to the hub and having the smallest diameter is referred to as the inner diameter (ID). Data to be stored on the disk 11 is first "encoded" by a read/write channel residing, e.g., in the disk drive controller 23. The data is encoded into a form suitable for the storage medium, then transmitted via a flex cable or other connector means 18 to the arm electronics 17 and then on to the transducer 14 for writing to the disk. For example, in a magnetic disk drive, digital data is encoded into a series of pulses. As is known in the art, the pulses are transmitted in the form of a current to the transducer, and cause a fluctuating magnetic field at the transducer pole tip that affects the magnetization of discrete regions on the disk surface. When a transducer senses or "reads" information from the disk, the data is transmitted in encoded form via the arm electronics 17 to the channel for "decoding". The arm electronics usually include means for amplifying and synchronizing the read signal.

The motor is fixedly attached to the disk 11. It may be encased in a hub, in which case the rotational force of the motor is translated to the hub and from the hub to the disk 11.

To protect a rotary disk drive from external forces during operation or movement, means may be implemented to park the transducer 14 when the disk is not operating and/or during periods of inactivity (i.e. times when data is not being written to or retrieved from the disk). Such means may include, for example, a load/unload ramp at the OD or a parking zone located at the ID.

As those skilled in the art of disk drive design will understand, the disk drive controller (HDC) 23 includes logic to control certain functions of the component drive 21. The HDC also serves as an intermediary interface between the component disk drive 21 and the programmable interface 24. Functions of the HDC 23 include, for example, servo control, data, address and command buffers, drive motor controls, and a read/write channel for coding and decoding data.

Typically, the HDC 23 is a function residing in one or more solid state die components dedicated to disk drive control functions and residing external to the disk

drive 21. Preferably, all functions are contained within a single custom chip. The HDC 23 may alternatively be integrated with the arm electronics 17 and reside within the disk drive enclosure 16 or external to the drive. It may also be merged with the programmable interface 24. For example, the Xilinx XC40000 processor is entirely programmable and may include a simple microcontroller and drive interface electronics as well as other functions. A custom hybrid-type die could also be provided, comprising a gate array having a normal, non-programmable CMOS for the HDC function, and programmable gates like the XC4000 as the PLU. This would minimize the cost of having an entire die of programmable gates permanently dedicated to the HDC function and make more programmable gates available to the PLU application coexisting on the die.

The programmable interface 24 comprises a programmable logic unit (PLU) of some kind, e.g. a field programmable gate array (FPGA), programmable array logic (PAL), or a programmable logic device (PLD). It communicates with the HDC over a bus 26. The particular PLU implemented in the electronic circuit apparatus of the present invention will be determined by the type of application that the PLU is interfaced to. For example, if the interface 24 is to a communications network with a communications protocol such as RS-232 or ISDN, the PLU of choice may be a programmable gate array such as the one described in US Patent No. 5,121,342.

As is well known in the art, a PLU is generally programmed by some kind of microprocessor. The processor receives a microcode set corresponding to a desired PLU function from local storage. The microcode set comprises both programming instructions and specifications. In the electronic circuit apparatus of Fig. 1(a), the HDC 23 serves as the programming processor of the PLU 24 and the component disk drive 21 provides local storage for one or a library of microcode sets. This arrangement enables dynamic programming of the interface. Alternatively, a simple processor 28 such as an Intel 80186 16-bit controller, and a solid state memory component 29 (e.g. SRAM, EEPROM, flash memory, etc.), are provided, as shown in Fig. 1(b). The microprocessor 28 is coupled directly to the PLU 24 and receives the microcode set from the memory component 29, which may contain a library of microcode sets. The microprocessor may only be temporarily coupled to the PLU 24 in a manufacturing step, or may be a permanent component of the apparatus to enable on the fly programming. Yet another alternative is to use a PLU that includes some basic microprocessing functions, enabling the PLU to essentially program itself from microcode provided by internal memory, external memory, or the component drive.

Programming instructions for the PLU are either selected from a library of functions associated with that device, or are specially designed for a non-standard function according to methods presently known in the art. More Detailed information on PLU programming

may be obtained by referring to "The Programmable Logic Data Book", a 1994 product guide and data book publicly available from Xilinx Inc., of San Jose, California.

The application 25 may be any of a number of devices. For example, it may be an element of a communications network that follows one of the standard communications protocols (e.g. RS-232, V.35, ISDN). It may be a computing device with a SCSI or IDE bus. It may be an electronic circuit application of an electronic device with a nonstandard bus communications protocol. The PLU 24 communicates with the application over bus 27.

Fig. 2 represents one possible implementation of the electronic circuit apparatus of the present invention. A general electronic circuit board 31 is shown, representative of circuit boards found in a variety of electronic devices. It includes an electronic circuit, comprising a plurality of interconnected circuit elements 35, e.g. integrated circuits, resistors, capacitors, oscillators, etc. The solid state components comprise, for example, a microprocessor, memory, an arithmetic logic unit, a programmable logic unit, etc. The apparatus of the present invention is also included on the circuit board, and is highlighted by dashed lines 32. It comprises a discrete, component-sized disk drive 21, a disk drive controller 23, and a PLU 24. The PLU 24 of the implementation shown is configured to adapt the component drive 21 to the electronic circuit's bus architecture, so that the component drive will appear to the circuit as solid state memory. Thus the component drive 21 provides local storage to the circuit for storing information used in the normal operation of the circuit.

A circuit board such as that represented in Fig. 2 may reside, for example, in a personal computer, a laptop, or other computing device, wherein the communications bus has an SSA architecture. It may also reside in devices peripheral to a computing device, e.g. controller cards for larger disk drives, printers, modems, and fax-modems. A circuit board is often present in electronic devices such as video cameras, fax machines, cellular phones, electronic pagers, photocopiers, and remote control devices, which may have nonstandard bus architectures. All of these are likely to have electronic circuits with some local storage requirements.

Fig. 3 is representative of a card enclosure for an electronic circuit, adapted to be plugged into a compatible computer slot at connector 60. It may, for example, be a PCMCIA card type I, II or III having a predefined length 64, width 63, and height 62. The card thickness 62 is generally the most critical dimension of a card enclosure.

Fig. 4 shows the preferred embodiment of the electronic circuit apparatus of the present invention, wherein a component disk drive 21, an HDC chip 23 containing an AMD AM80C186 microprocessor, and a PLU 24, e.g. a Xilinx XC4000, are housed in a card-type enclosure 36. A digital clocking signal is distributed throughout the

device along with power (not shown). A library of microcode sets defining a plurality of PLU interface configurations is stored on the component drive 21. The card enclosure preferably comprises a PCMCIA, type II or type III card. The PLU 24 is configured to interface to an application 25 external to the card enclosure 36 and adapted to communicate over a PCMCIA bus. The physical interface between the card enclosure 36 and the application 25 comprises card connector 60 and a compatible socket 61 on the application 25.

Figs. 5(a)-5(d) show the component disk drive assembly of the preferred circuit architecture according to the present invention. The drive comprises a disk 11, a motor 44, an actuator assembly 12, arm electronics 17, and a housing 16. The disk 11 is preferably magnetic and includes one recording surface 8 with a substantially planar region 55 at its center. It is preferably mounted directly to a flat motor 53 along its non-recording surface 9 by some appropriate means, e.g. mechanically, or by applying a bonding agent along the interface between the motor 53 and surface 9. Use of a single recording surface 8 and direct platter mounting allows a wider, thinner motor 53 to be used than would be feasible for a disk having two recording surfaces. The diameter of the motor 53 may be as large or larger than the diameter of the disk 11 itself. The advantages of this type of motor 53 will be described in further detail below.

The actuator assembly comprises a voice coil motor, an actuator arm 6, a single suspension 13 and a transducer 14 supported on an air-bearing slider. Preferably, the transducer 14 is a magnetoresistive (MR) head, allowing greater data capacity. Magnetoresistive heads are known in the disk drive industry and are preferred because their high sensitivity enables greater areal density (i.e. bits per inch) than conventional inductive heads. Using a state of the art magnetoresistive head in the preferred embodiment provides useful data storage capacity for applications requiring moderate data storage. However, it will be understood that other types of transducers may also be implemented. Moreover, the disk drive may be adapted to include a plurality of transducers 14 on suspension 13.

Direct mounting of disk 11 to motor 53 creates an unobstructed region 55 at the disk's center 52 accessible to the actuator assembly 12. The diameter of recording tracks in this region would be too small for practical use, so the region is used for center parking. Figs. 5(a) and 5(b) show head 14 and suspension 13 positioned over the data recording surface of disk 11. During times of inactivity, the head is "parked" in the central region 55 as shown in Figs. 5(c) and 5(d) so that head 14 is substantially aligned with the disk center 52. As the slider 14 nears the inner diameter of the disk 11, the upward force of the air flow is reduced and the slider begins to drag along region 55. It is then "parked" at the disk center 52 as shown. Parking structures, e.g. ramps, may be added to the center parking region to facilitate parking. Spacer structures may also be provided between this

region and the upper drive housing for added structural support.

The actuator assembly 12 moves the slider back onto the disk surface when necessary by applying a force to overcome the stiction between slider 14 and surface 8. "Stiction" is a term of art for the attractive and frictional forces between slider 14 and surface 8. It is greatest at the outer diameter of the disk and decreases in the direction of the disk center, being essentially reduced to zero at dead center. Since the slider is parked at disk center 52, stiction is virtually nonexistent, and very little force is required to overcome it.

The slider 14 is preferably also center parked during periods when the drive is not operational. When the drive is powered on, the disk 11 spins without any significant stiction impedence. Reduced stiction translates to a reduction in starting torque required from the motor 44. Reduced starting torque in turn leads to a reduction in the electric power requirements of the drive.

As mentioned previously, the preferred embodiment of the component disk drive allows a flatter, wider motor 53 to be used than conventional disk drives requiring hubs. The motor may take a variety of forms. For example, it may be fixed to the disk drive housing or integrated therein. It may have the shape of a disk, or be annular in shape. It may include a hub, and the disk may be mounted directly to the hub, or alternatively, the hub may penetrate the disk and lie substantially flush with the recording surface to preserve actuator access for center parking. An annular motor may surround a stationary hub structure that lies flush with the recording surface to provide a stationary parking zone at disk center.

Those skilled in the art of motor design generally understand that an increase in the diameter of the motor windings increases its moment arm to generate more torque with less force. The motor thus requires less electric current to perform the same operation as a center hub motor. Since the electrical power (P) is proportional to the square of the current, a reduction in the current requirement will result in a large reduction in the electric power requirement as shown below.

$$(1) \quad P = I \times I \times R,$$

and

$$(2) \quad I = k/D.$$

Therefore,

$$(3) \quad P = (k \times k \times R) / (D \times D),$$

where P is the electric power of the motor, I is the electric current used by the motor, D is the motor diameter, R is the electric resistance of the motor, and k is the inversely proportional constant of the motor current to diameter. Applying the above equations, an increase of motor diameter by, for example, a factor of 3 will result in a motor that can achieve the same torque with 1/9th of the power. The motor torque is transmitted directly to the bottom

surface of the disk through, for example, an adhesive or a coupling device. Examples of motors that may be used to implement the preferred embodiment include those implemented in the commercially available IBM Travelstar, and the Maxtor MobileMax Lite.

The disk drive assembly 21 of Fig. 4 preferably measures no more than 2 inches in length 72 X 1.6 inches in width 73 X 5 mm in height. For example, a 1.3" magnetic disk is mounted to a small, flat motor such as that implemented in Maxtor's MobileMax Lite. (At present there are no flat motors under 5 mm commercially available for implementation in a type II design. However, availability is anticipated in the near future and prototypes are currently being tested.) The actuator assembly is a conventional dual-suspension design such as that used in the Hewlett Packard KITTYHAWK 1.3" drive, modified to have a single suspension with a magnetoresistive head. Modifications required for operability include removal of the lower suspension and any actuator arm height adjustments necessitated by the height of the direct-mounted disk. It may also be desirable to make additional modifications, e.g. reducing the height of the actuator assembly. The manner of making such modifications will be readily apparent to a person of ordinary skill in the operation of disk drive assemblies in view of this specification and the state of the art.

The disk drive assembly 21 preferably occupies less than 50% of the available card area. The remaining card area is populated by a plurality of electronic components 71 comprising an application subcircuit.

If it later becomes desirable to adapt the card-enclosed apparatus of Fig. 4 to a non-PCMCIA application, e.g. to an application adapted to communicate over an IDE bus, the programmable interface 24 can be reprogrammed in the following manner.

The microcode set defining an IDE interface is stored on the component drive 21. It originates, however, from a designer's logic block diagram or schematic. Popular computer-aided software tools such as viewDraw are used to construct a logic block diagram. In this case, the diagram will represent an IDE interface option for the disk drive. Once the design is simulated, it is synthesized into a serial bitstream of microcode using a compiler program. In this case, a Xilinx tool called XACT generates the bitstream of microcode. For a Xilinx XC4000 FPGA, the microcode set can be 422,128 bits long and make up to 25,000 gates of logic (see diagram on page 2-26 of "The Programmable Logic Data Book") that will allow the disk drive to communicate with an IDE bus in this case. A unique microcode set is similarly generated for each interface application that the PLU 24 will support and is stored inside the component hard disk 21.

The interface change is initiated by some external means (e.g. a selectable switch toggled by a user of the apparatus). This will signal the microprocessor inside the HDC 23 to retrieve the appropriate IDE microcode set from the component drive 21 and will also signal the Xilinx XC4000 PLU 24 to go into a programming state.

The microcode set is fed serially from the component disk drive 21, through the HDC 23, and into the PLU 24. Inside the PLU 24, the code is stored in a series of latches. Long chains of latches form the IDE interface to the component disk drive 21. Once the last information is fed into the PLU 24, the HDC signals the PLU to go into operating mode. This will make the PLU 24 an active interface for the new IDE application.

To enable the drive 21 to communicate with up to ten different interfaces using the Xilinx XC4000 PLU 24, 4,221,280 bits of storage are required. This translates into about 1/2 megabyte of data. For the preferred component disk drive of the present invention, this storage space is a small fraction of the drive's total storage capacity (typically 20 to 40 MB). Thus, plenty of user storage remains available for storing additional application information.

PLU programming can alternatively be done at the time of manufacture of the device, but it would not have the flexibility of a field programmable drive interface. This approach may be desirable, however, for large-scale manufacturing of a card-based electronic circuit application intended for use in a plurality of external applications requiring different types of interfaces. Under such circumstances, it is desirable to keep as many physical parts of the apparatus in common. In the meantime, the PLU is configured with software to a specific bus architecture. For example, one microcode set would adapt the drive to work with an IDE interface, another with a SCSI interface, and another with a PCMCIA interface. The hardware connector on the interface (e.g. 50 pins for SCSI, 68 pins for PCMCIA) might represent the only physical modification required for the various versions.

The above examples represent the simplest form of this invention. Another embodiment illustrated in Fig. 6 includes an application 25 residing within the same enclosure 51 as the component disk drive 21, HDC 23, and PLU 24. The enclosure may be a device or a card such as a PCMCIA type II or type III. The application may comprise, for example, a wireless communication device. In this embodiment of the electronic circuit apparatus, the PLU 24 is actually a functional part of the application 25. Because wireless communications use many different protocols, part of the application circuitry, i.e. the PLU 24, needs to be reconfigured for each different protocol (e.g., one application may be for cellular communications, another for a local area network).

In this case, the PLU 24 is reprogrammed dynamically in the manner described above, but is a functional part of the application 25. The application 25 may contain a digital signal processor (DSP, not shown) to facilitate the waveform-to-digital processing of any communication protocols.

The processor which programs the PLU 24 can be a nonprogrammable reserved area inside the PLU 24. Alternatively, it can be contained inside the HDC 23 as an 8-bit or 16-bit processor such as an AMD 80C186

chip. Alternatively, the processor can be a stand alone chip (not shown) such as the AMD 80C186 which communicates with the HDC 23 and PLU 24. The processor may need some external RAM memory 52 to hold the necessary microcode sets, depending upon the complexity of its function.

Fig. 7 shows a third embodiment of the present invention, comprising a component disk drive 21, an HDC 23 merged into a PLU 24 (although it will be understood that the HDC may reside alone or in combination with the arm electronics 17, as previously discussed), and an electronic subcircuit 44 housed in the same enclosure 51. The enclosure 51 is linked to a communications network (not shown) via ISDN bus 27, which connects to a phone line 41. The PLU 24 serves as the interface between the subcircuit 44 and the communications network. The network presumably supports a plurality of different communications protocols, and the PLU adapts the subcircuit 44 to these protocols. Optionally, the enclosure is of a PCMCIA-type and further comprises a PCMCIA connector 43 and a microprocessor 45 such as the AMD 80C186 for providing the interface to the PCMCIA bus. The microprocessor 45 may also be a part of the subcircuit 44. The electronic circuit assembly shown may also optionally include a second programmable interface between the HDC 23 and the subcircuit 44 so that the storage of the component drive is made available to the subcircuit 44.

Although the electronic circuit apparatus of the present invention has been described in terms of specific embodiments, it is to be understood that this disclosure is not to be interpreted as limited to those embodiments shown. Various further alterations and modifications will no doubt become apparent to those skilled in the art after having read the above disclosure. Moreover, the examples provided are not intended to be exhaustive, and the scope of the present invention defined by the following claims is intended.

Claims

1. An electronic circuit apparatus, comprising:

a disk drive for storing and retrieving information;

a programmable logic unit (PLU);

means operatively coupled to said disk drive and said PLU for controlling the operation of said disk drive; and

means for configuring said PLU as an interface to enable communication by said disk drive over a bus.

2. Electronic circuit apparatus as claimed in claim 1,

further comprising a card enclosure housing said apparatus, and means for electrically coupling said card to an application adapted to communicate over said bus.

3. Electronic circuit apparatus as claimed in claim 2, wherein said card enclosure further comprises a PCMCIA-type card.

4. Electronic circuit apparatus as claimed in any preceding claim, wherein said means for controlling said disk drive further comprises a hard disk controller (HDC).

5. Electronic circuit apparatus as claimed in claim 4, wherein said HDC resides within said disk drive.

6. Electronic circuit apparatus as claimed in claim 4, wherein said HDC resides in an integrated circuit chip.

7. Electronic circuit apparatus as claimed in claim 4, wherein said HDC resides within said PLU.

8. Electronic circuit apparatus as claimed in any preceding claim, wherein said means for configuring said PLU further comprises a microcode set, means for storing said microcode set, a microprocessor, and means for providing said microcode set to said microprocessor for use in programming said PLU.

9. Electronic circuit apparatus as claimed in claim 8, wherein said microprocessor further comprises said means for controlling said disk drive.

10. Electronic circuit apparatus as claimed in claim 8 or claim 9, wherein said storing means further comprises a solid state memory component.

11. Electronic circuit apparatus as claimed in claim 8 or claim 9, wherein said storing means further comprises said disk drive.

12. Electronic circuit apparatus as claimed in claim 11, wherein said disk drive stores a plurality of microcode sets for use in configuring said PLU, each corresponding to a unique bus architecture, and wherein said PLU is dynamically configurable to any one of said bus architectures.

13. Electronic circuit apparatus as claimed in any preceding claim and further comprising:

an electronic circuit application;

said means for configuring being arranged to configure said PLU as a communications interface between said disk drive and said electronic

circuit application.

14. Electronic circuit apparatus as claimed in claim 13, further comprising a card enclosure housing said apparatus, and means for adapting said electronic circuit to communicate with an application external to said card enclosure over a bus.

15. Electronic circuit apparatus as claimed in any preceding claim and further comprising:

an electronic circuit application,

said means for configuring being arranged to configure said PLU as an interface to enable communication by said electronic circuit over a bus; and a second PLU operatively coupled between said controlling means and said electronic circuit application, and means for configuring said second PLU as a communications interface between said controlling means and said electronic circuit application.

5
10
15
20
25
30
35
40
45
50
55

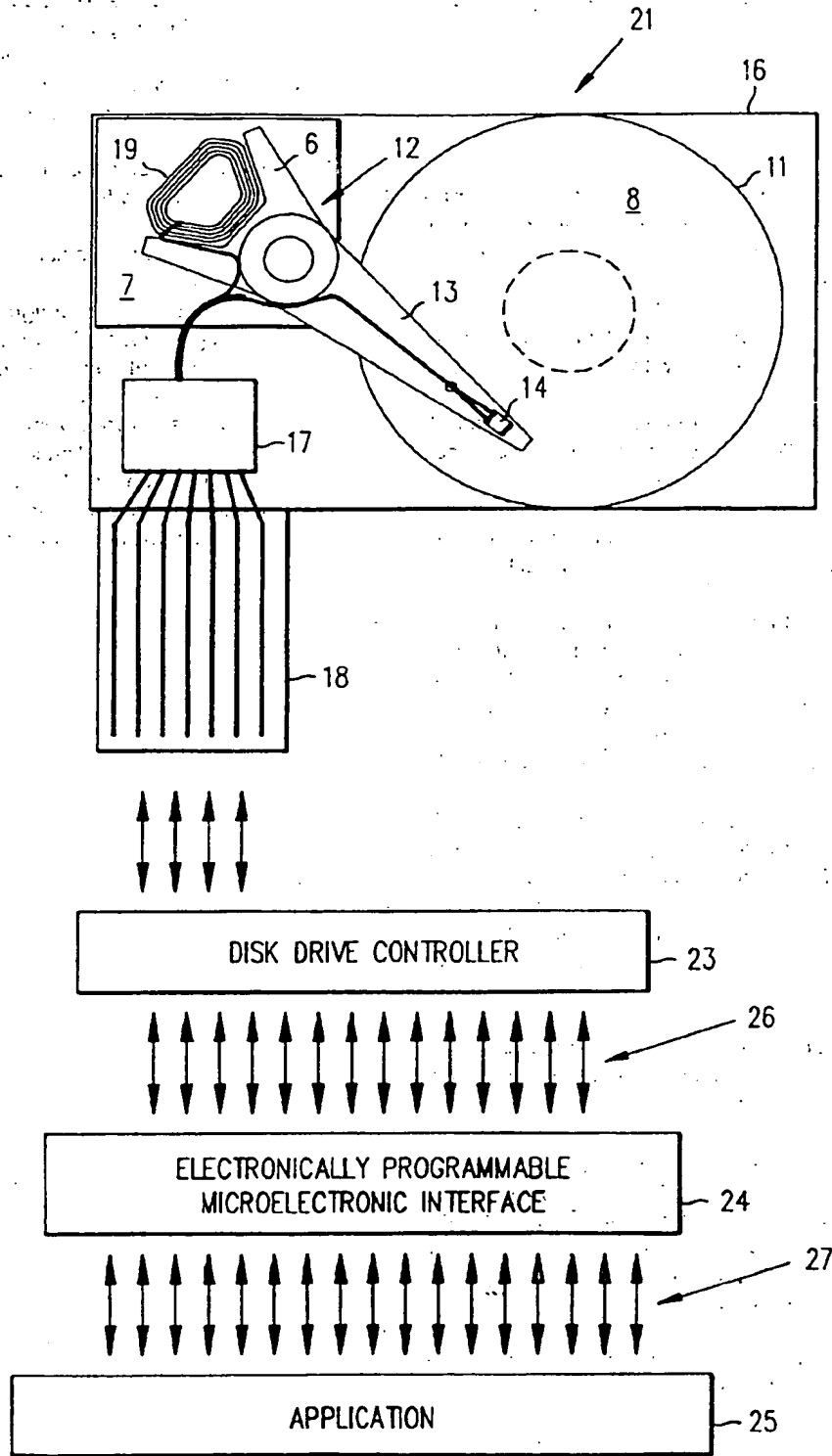


FIG. 1a

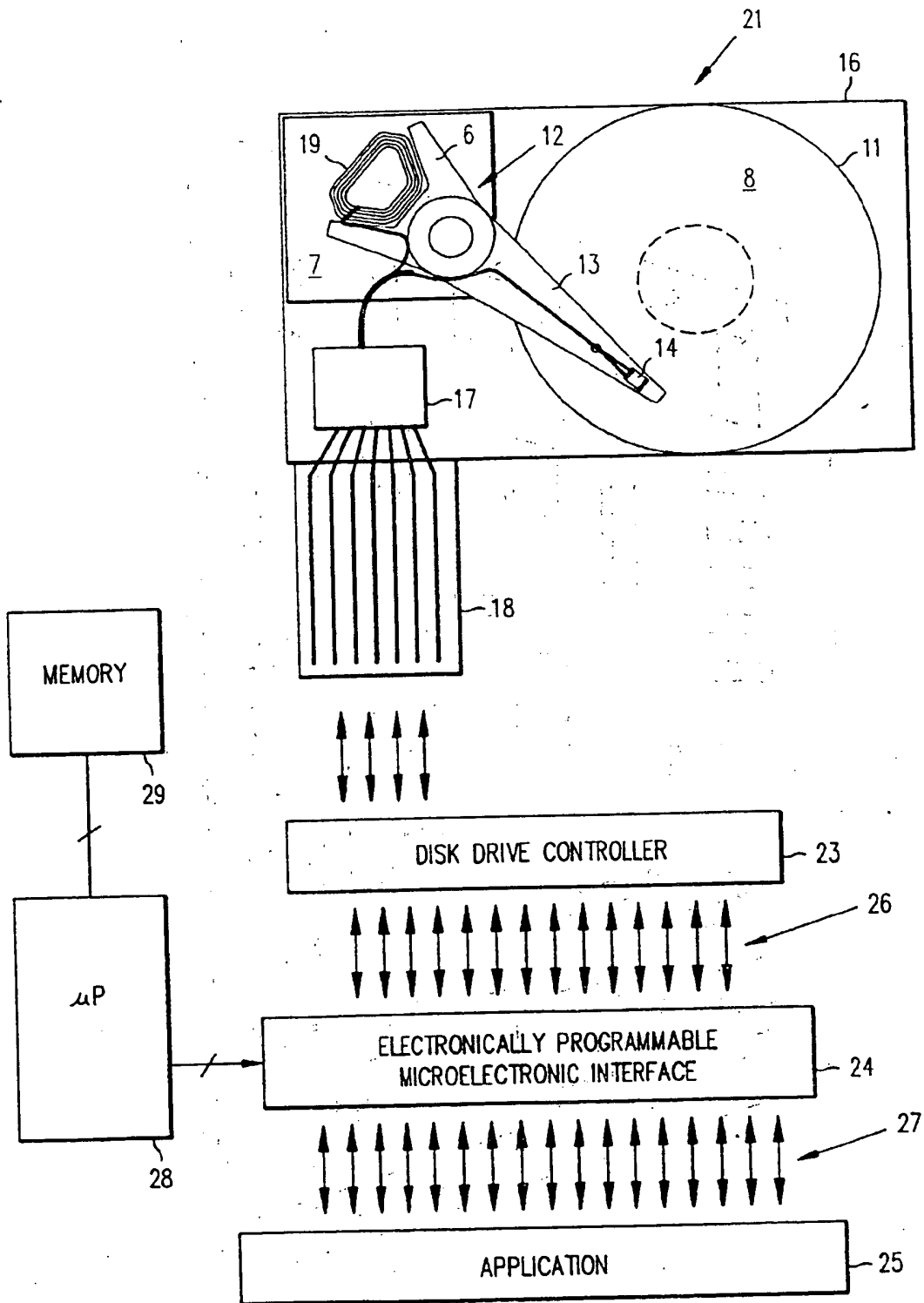


FIG. 1b

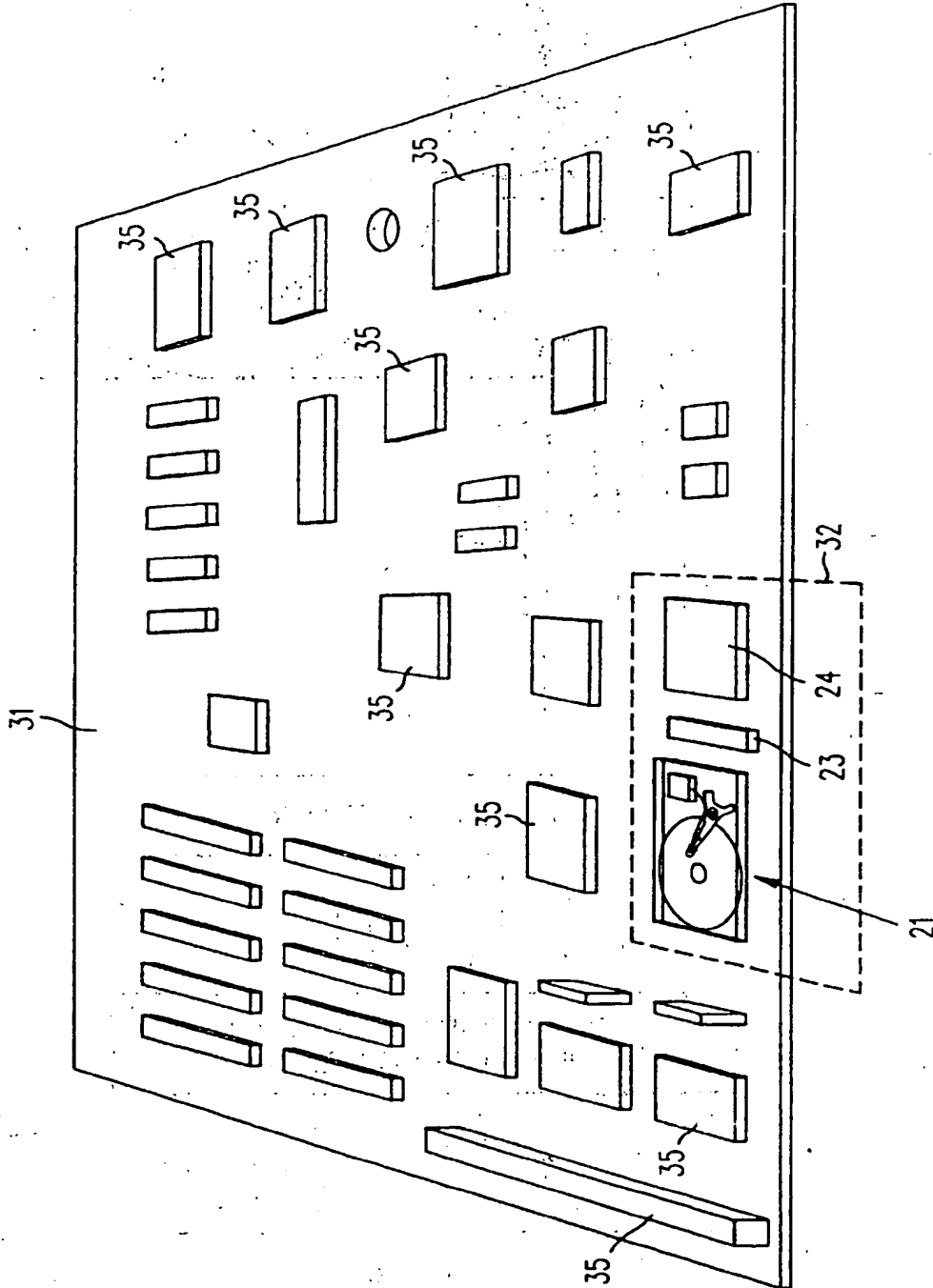


FIG. 2

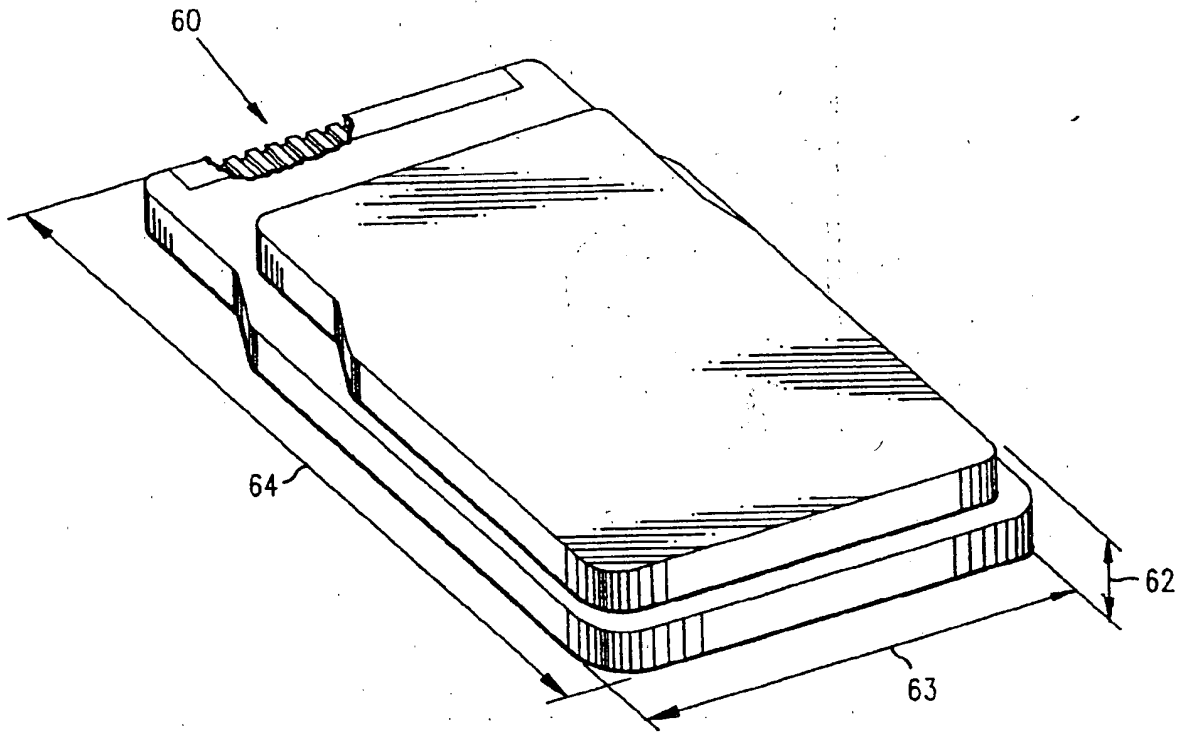


FIG. 3

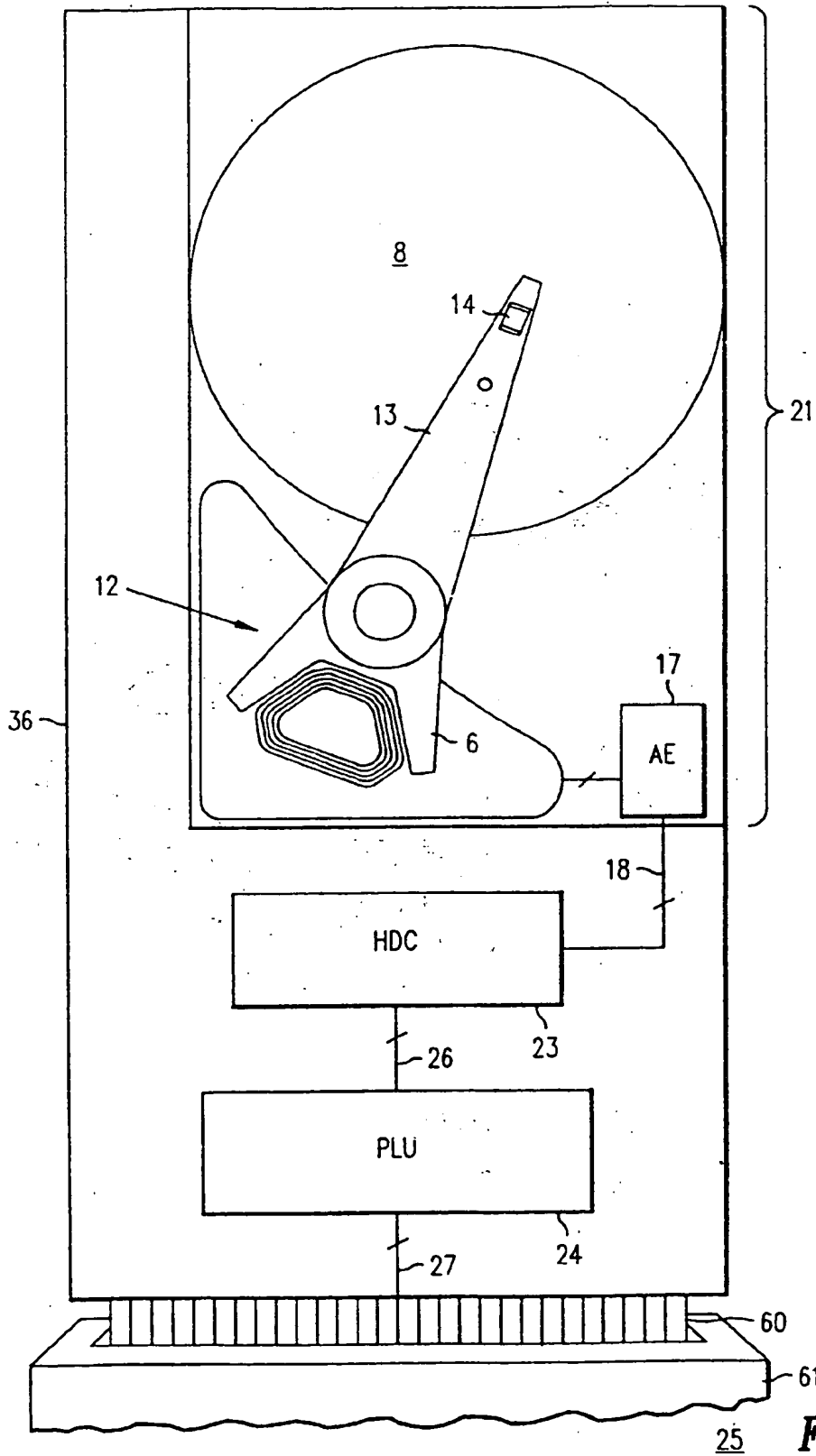


FIG. 4

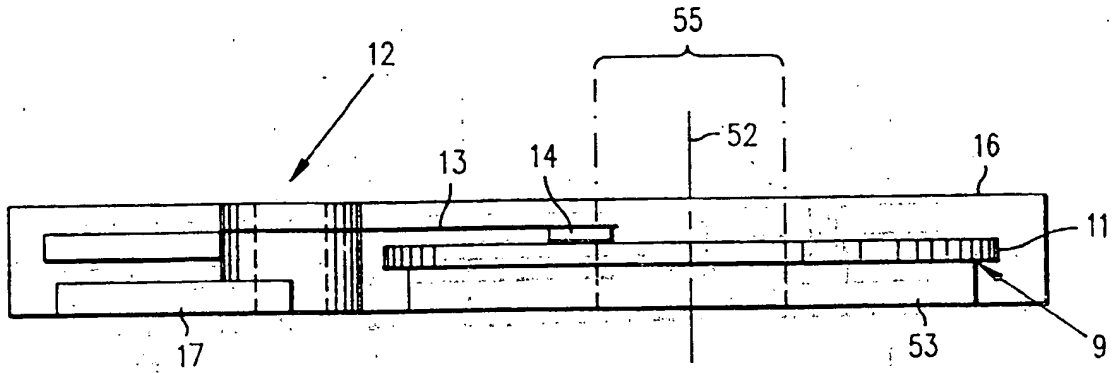


FIG. 5a

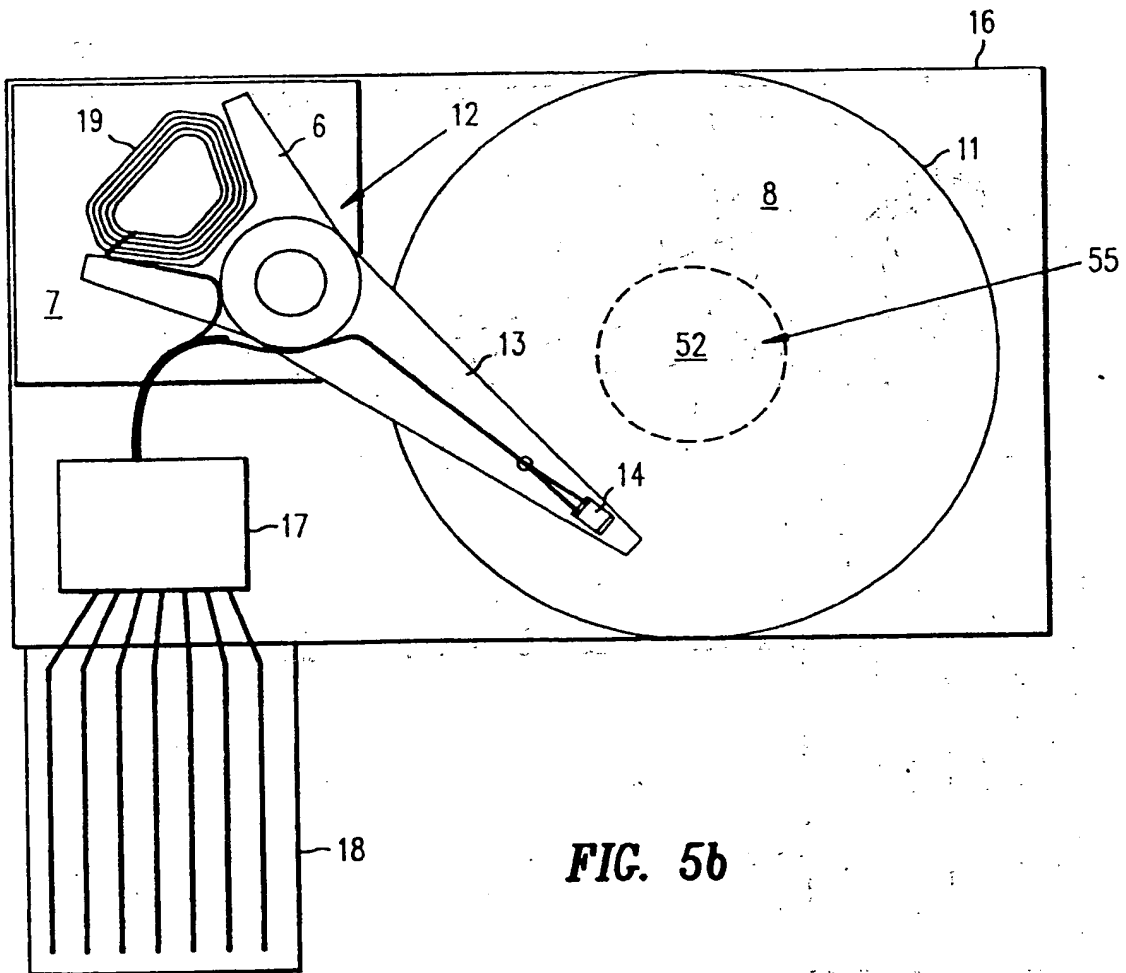


FIG. 5b

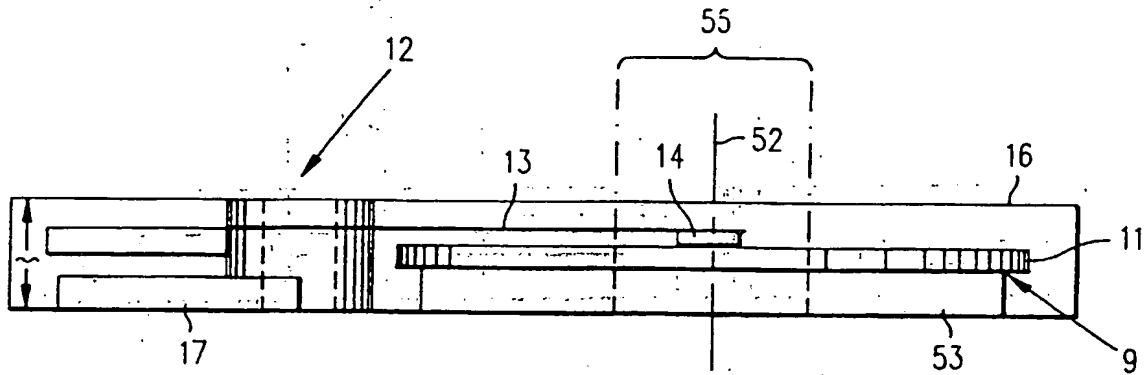


FIG. 5c

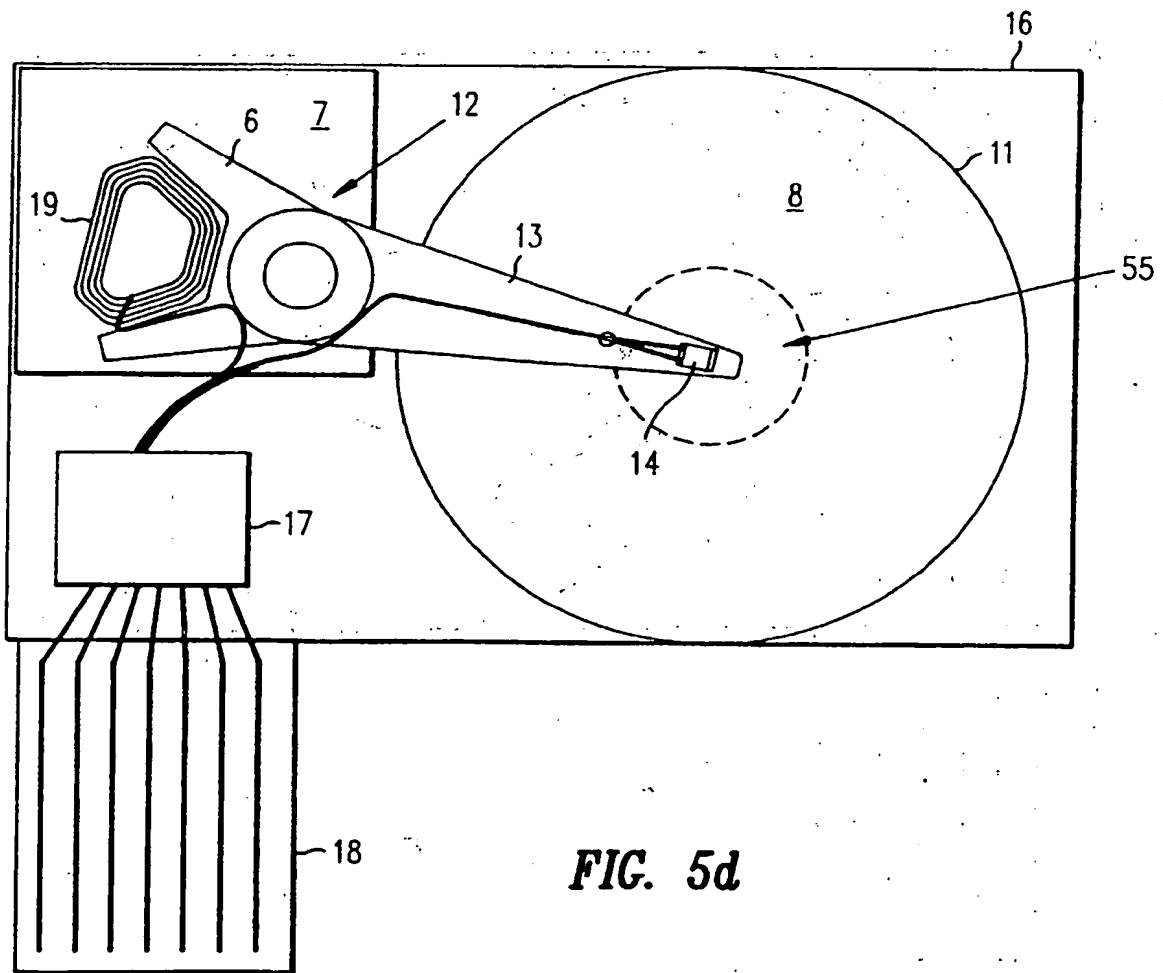


FIG. 5d

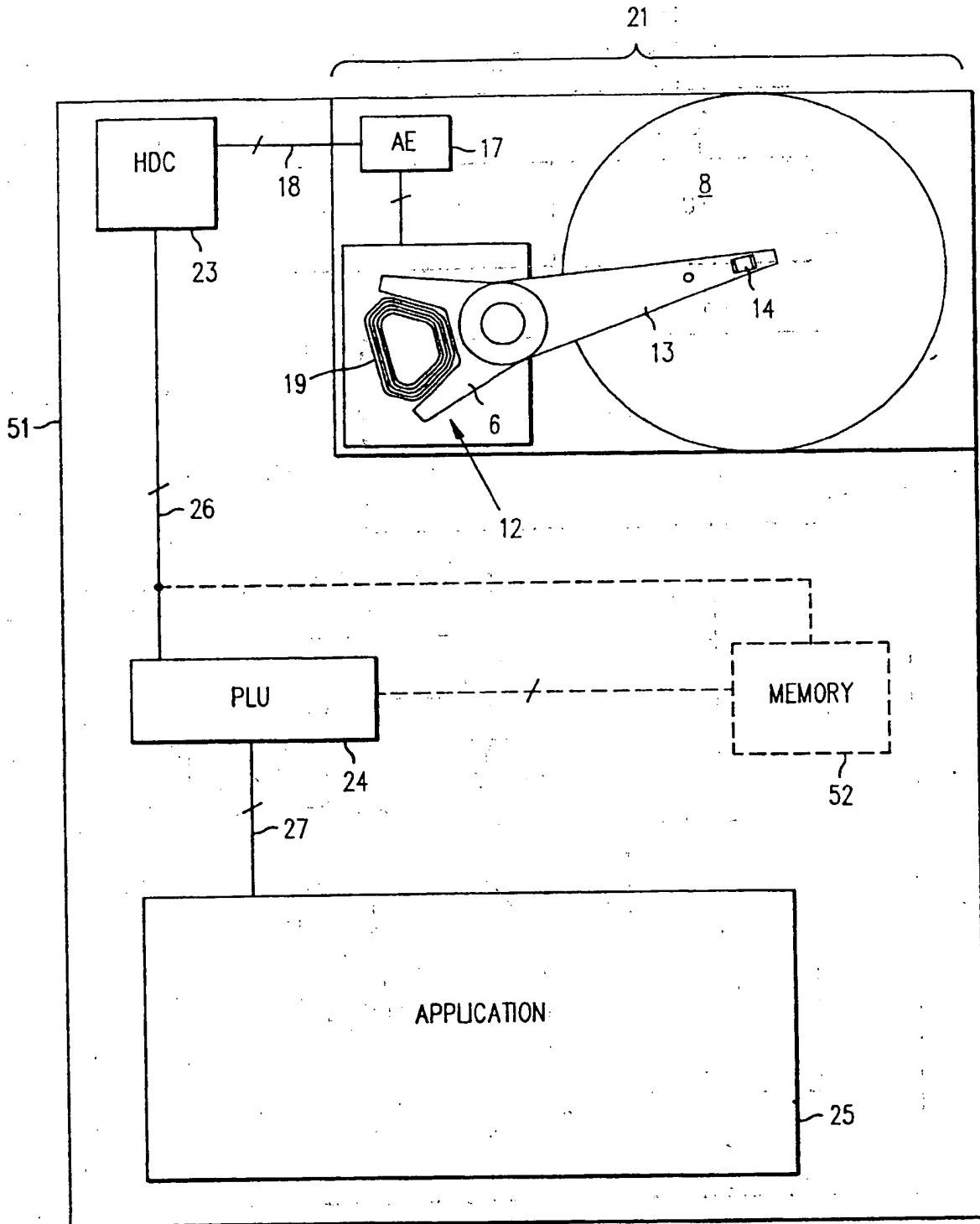


FIG. 6

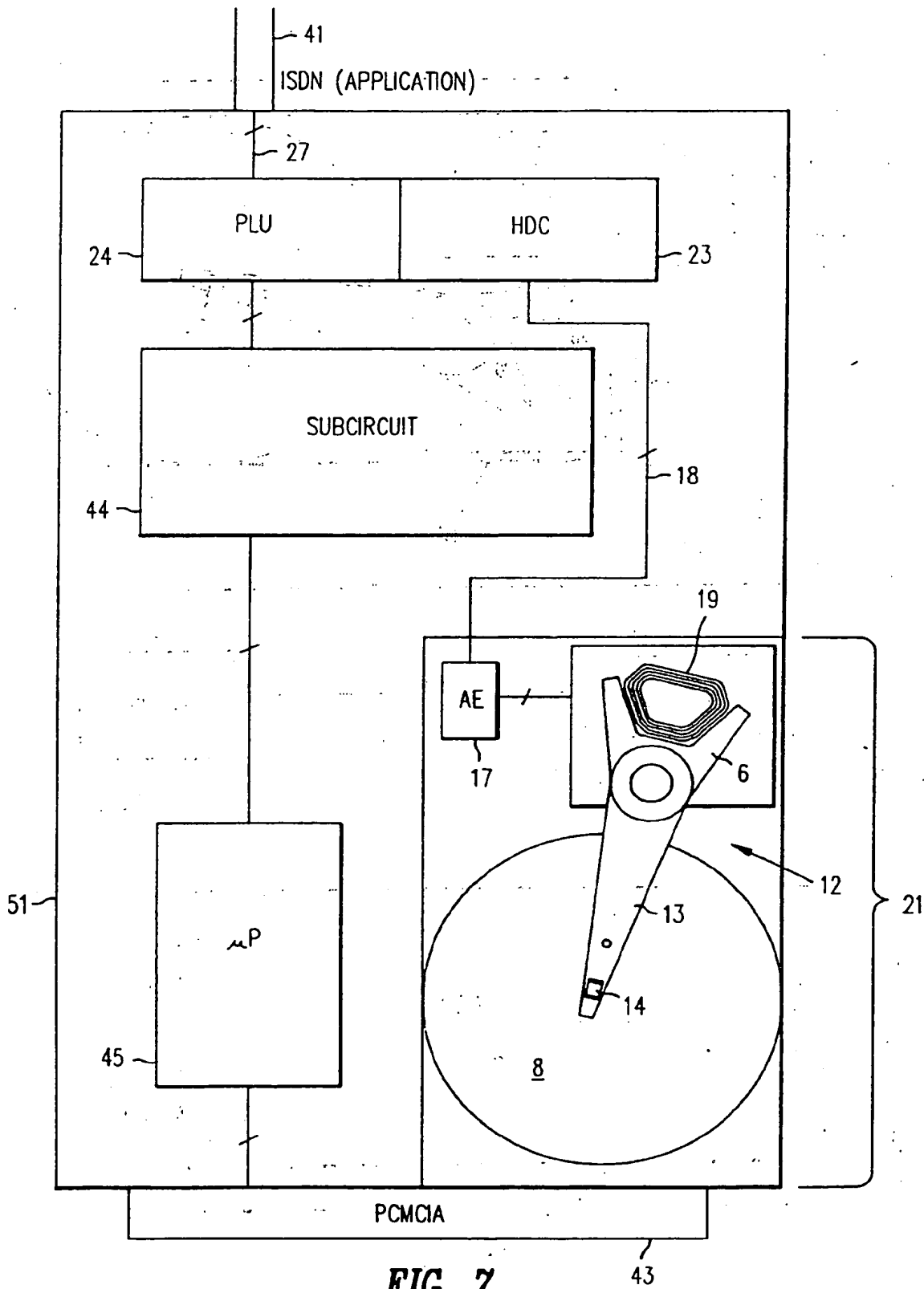


FIG. 7

This Page Blank (uspto)

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

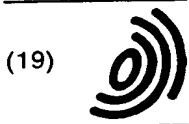
Defects in the images include but are not limited to the items checked:

- BLACK BORDERS
- IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT OR DRAWING
- BLURRED OR ILLEGIBLE TEXT OR DRAWING
- SKEWED/SLANTED IMAGES
- COLOR OR BLACK AND WHITE PHOTOGRAPHS
- GRAY SCALE DOCUMENTS
- LINES OR MARKS ON ORIGINAL DOCUMENT
- REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.

This Page Blank (uspto)



Europäisches Patentamt
European Patent Office
Office européen des brevets



(11) EP 0 718 751 A3

(12) EUROPEAN PATENT APPLICATION

(88) Date of publication A3:
12.02.1997 Bulletin 1997/07

(51) Int Cl.⁶: G06F 3/06

(43) Date of publication A2:
26.06.1996 Bulletin 1996/26

(21) Application number: 95309160.0

(22) Date of filing: 15.12.1995

(84) Designated Contracting States:
DE FR GB

(72) Inventor: **Shafe, Mathew K.**
Campbell, California 95008 (US)

(30) Priority: 23.12.1994 US 363464

(74) Representative: **Moss, Robert Douglas**
IBM United Kingdom Limited
Intellectual Property Department
Hursley Park
Winchester Hampshire SO21 2JN (GB)

(71) Applicant: **International Business Machines Corporation**
Armonk, N.Y. 10504 (US)

(54) **Electronic circuit apparatus employing small disk drive with reconfigurable interface**

(57) An electronic circuit apparatus is provided for electronic circuits and devices, comprising a component disk drive, a disk controller, and a programmable interface for dynamically adapting the component drive to an external application. The interface is programmed by the disk controller using one of a library of microcode sets stored on the component drive. Alternatively, a microprocessor independent of the disk controller programs the interface from microcode stored in solid state memory. In an alternative embodiment, the apparatus further comprises an application circuit and the programmable interface adapts the component drive for use by the circuit. In another embodiment, the apparatus comprises a subcircuit that communicates with an external application over a predetermined bus architecture, and the programmable interface adapts the subcircuit for such communication.

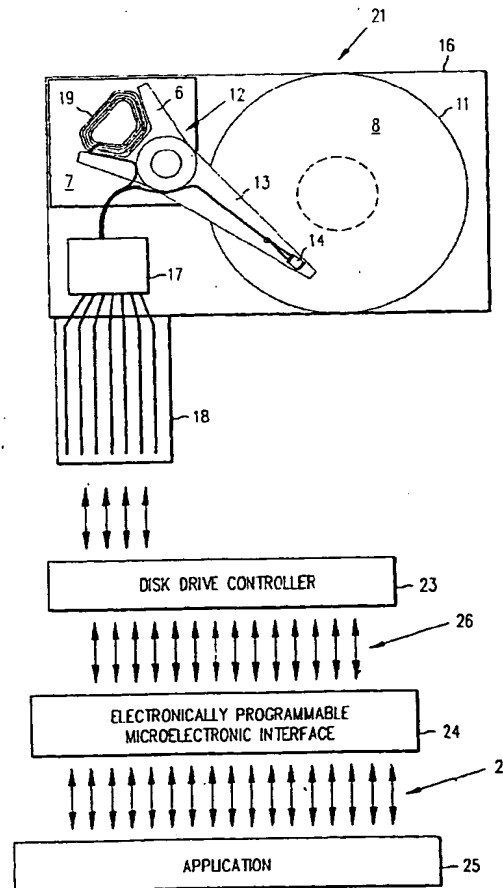


FIG. 1a

EP 0 718 751 A3

Best Available Copy



European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 95 30 9160

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
D,X	US-A-5 121 342 (SZYMBORSKI ROBERT C ET AL) 9 June 1992 * column 2, line 23 - line 41; figure 2A * ---	1,4	G06F3/06
A	EP-A-0 185 098 (HITACHI LTD) 25 June 1986 * page 1, paragraph 2 - page 9, paragraph 1; figure 2 * ---	1-15	TECHNICAL FIELDS SEARCHED (Int.Cl.6) G06F
A	WO-A-94 29852 (MAXTOR CORP) 22 December 1994 * page 17, line 1 - page 19, paragraph 2; figures 8,9 * -----	1-15	
The present search report has been drawn up for all claims			
Place of search THE HAGUE		Date of completion of the search 5 December 1996	Examiner Moens, R
CATEGORY OF CITED DOCUMENTS X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document		T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons ----- & : member of the same patent family, corresponding document	

EPO FORM 183 01/82 (P0401)

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- BLACK BORDERS**
- IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- FADED TEXT OR DRAWING**
- BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- SKEWED/SLANTED IMAGES**
- COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- GRAY SCALE DOCUMENTS**
- LINES OR MARKS ON ORIGINAL DOCUMENT**
- REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.

This Page Blank (uspto)

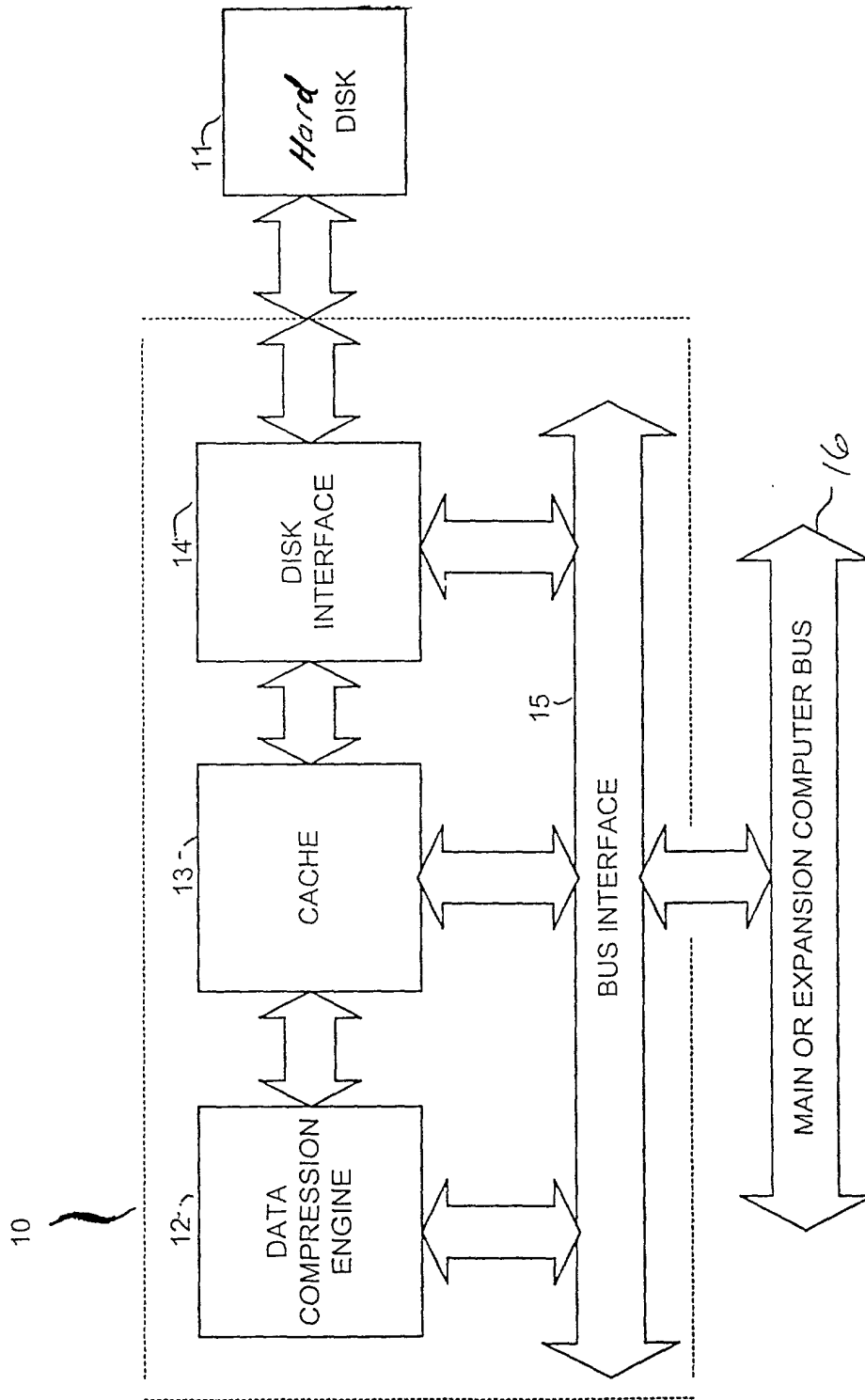


FIGURE 1

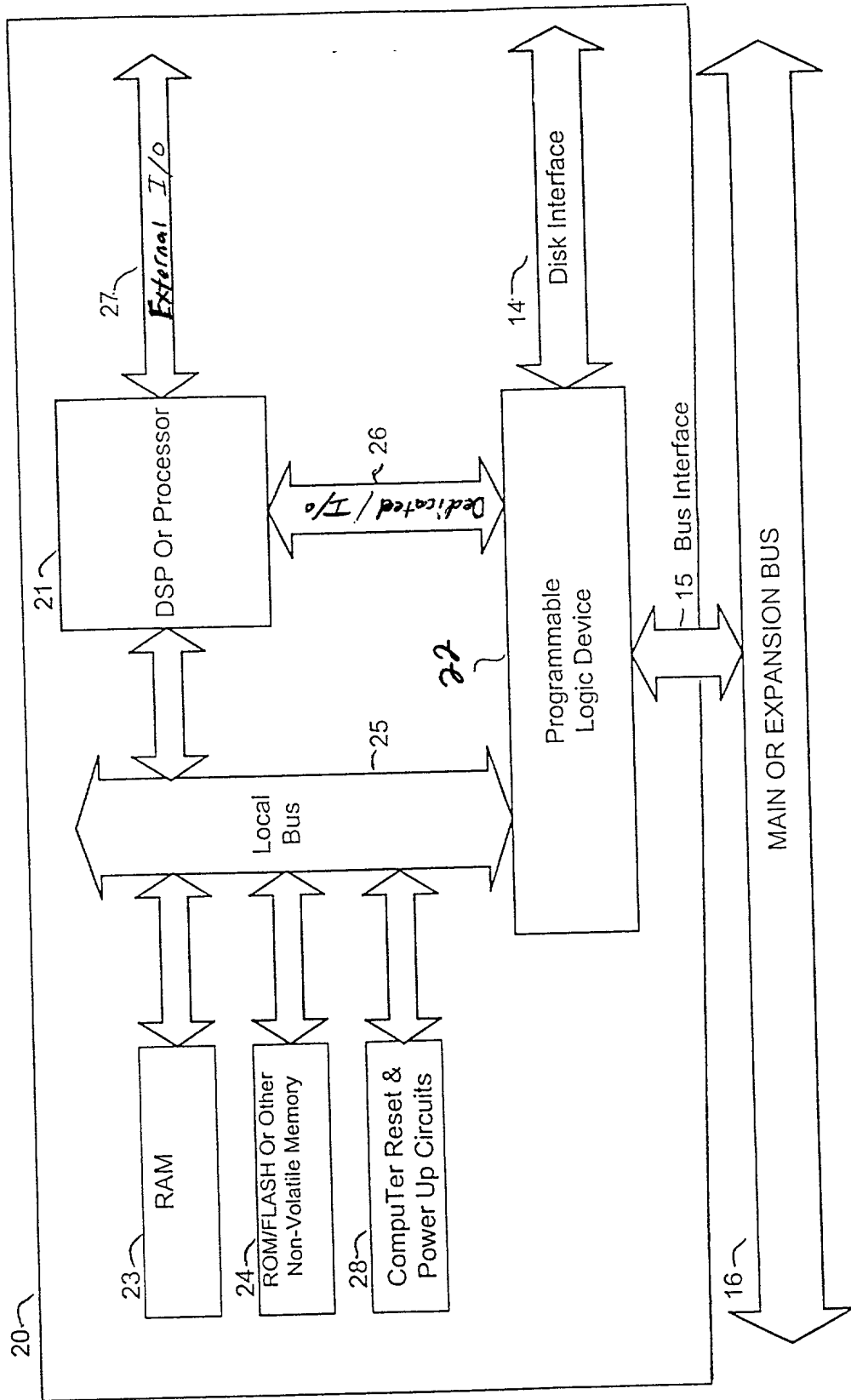


FIGURE 2

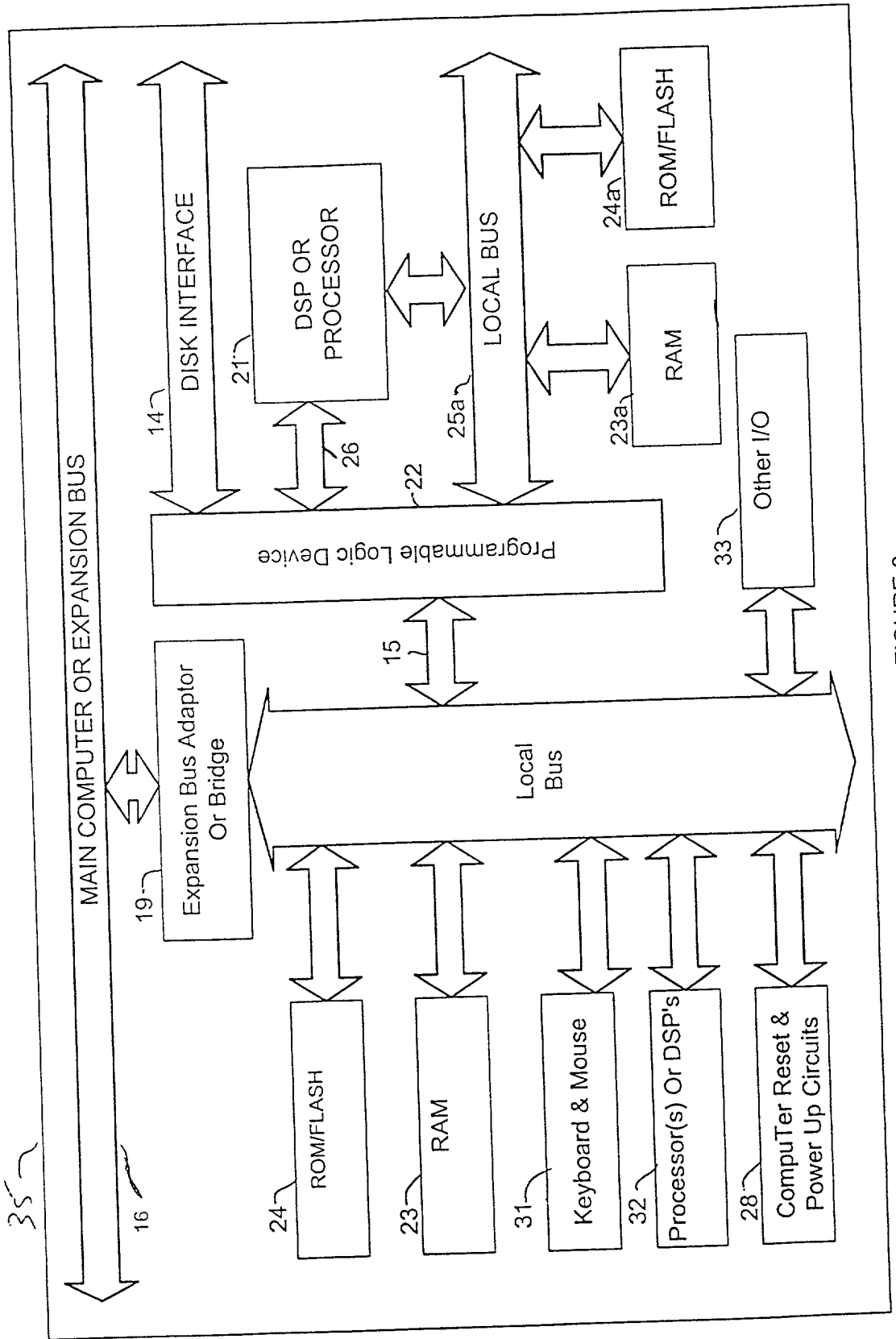


FIGURE 3

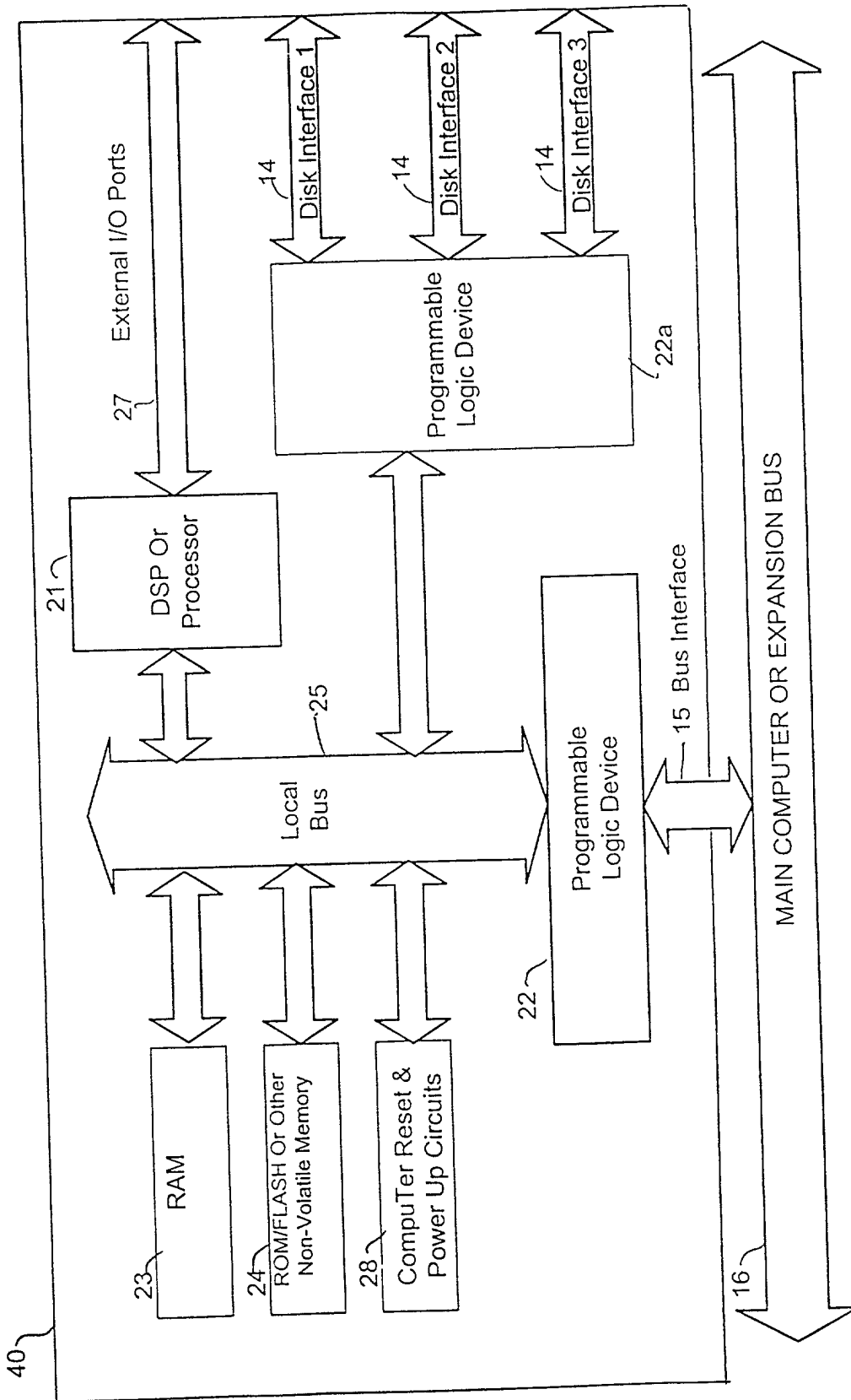


FIGURE 4

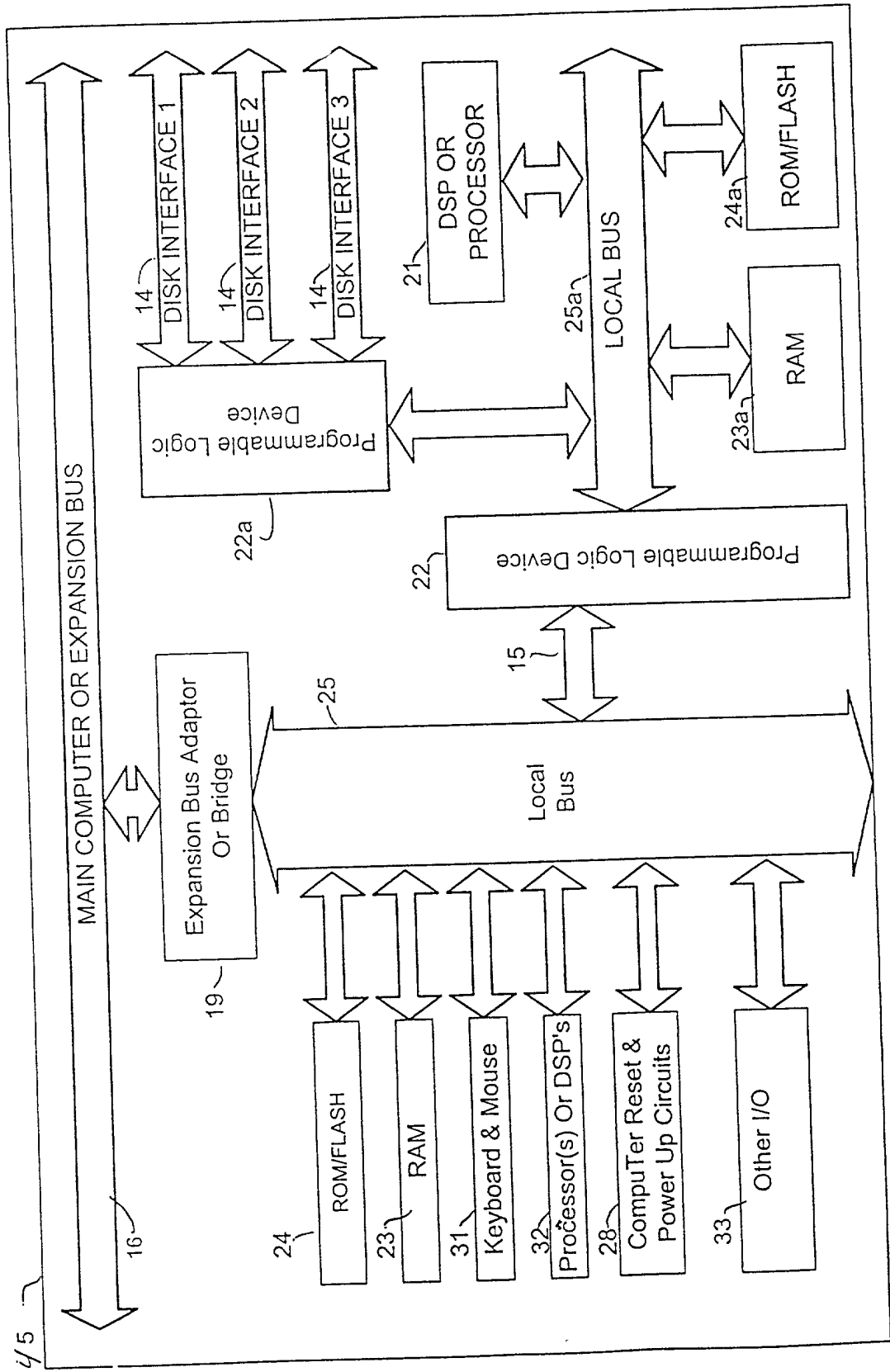


FIGURE 5

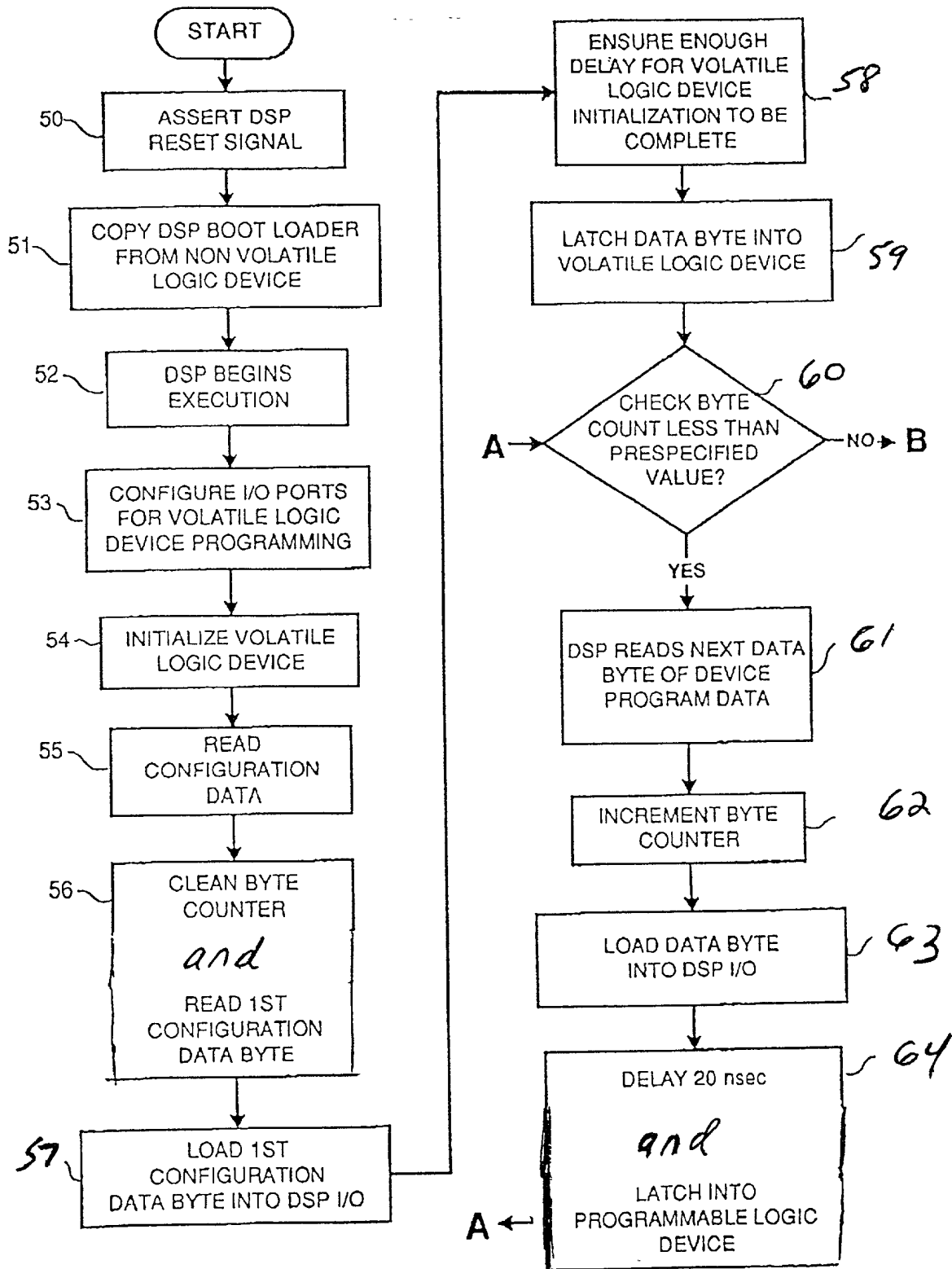


FIG. 6a

TOP SECRET

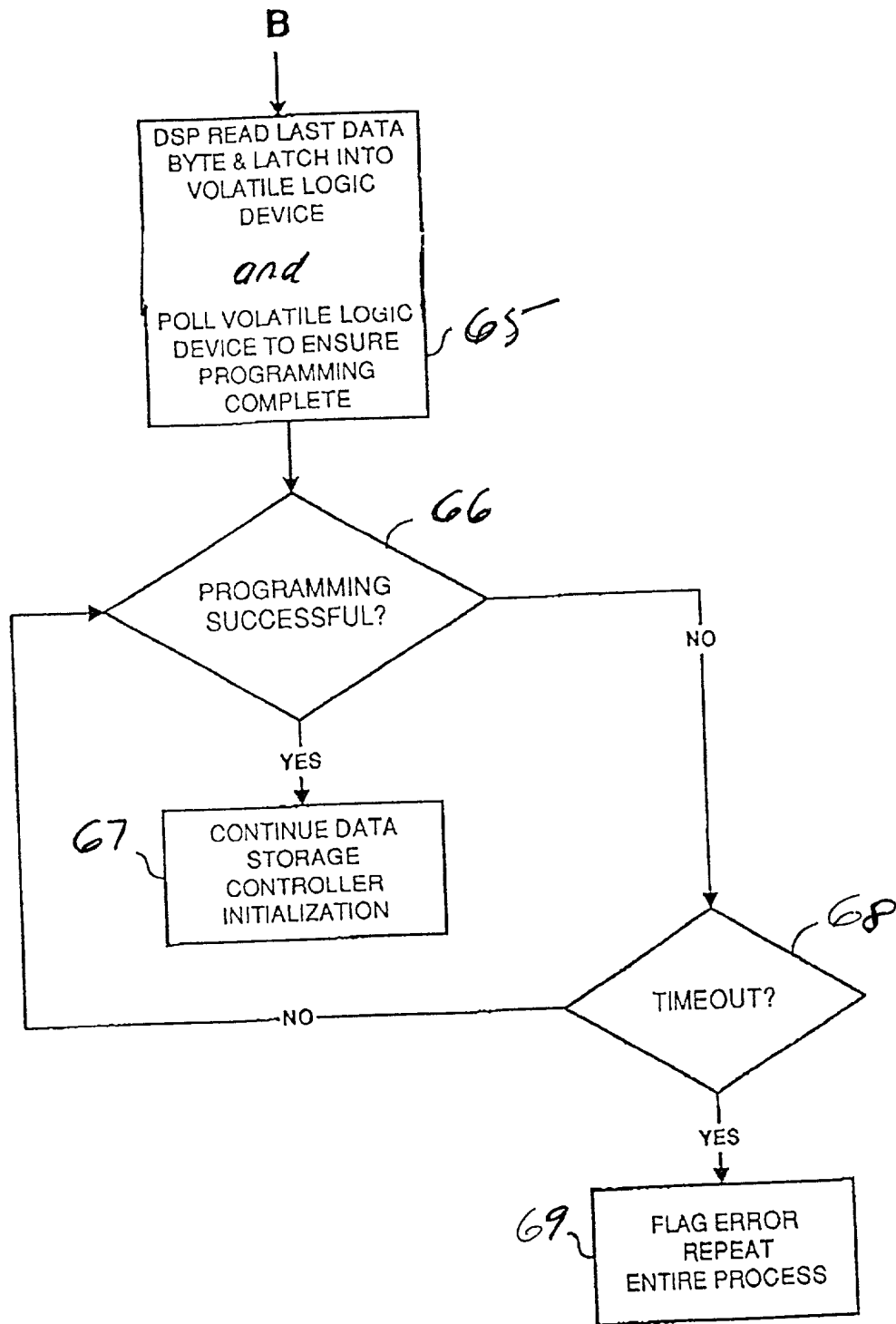


FIG. 6b

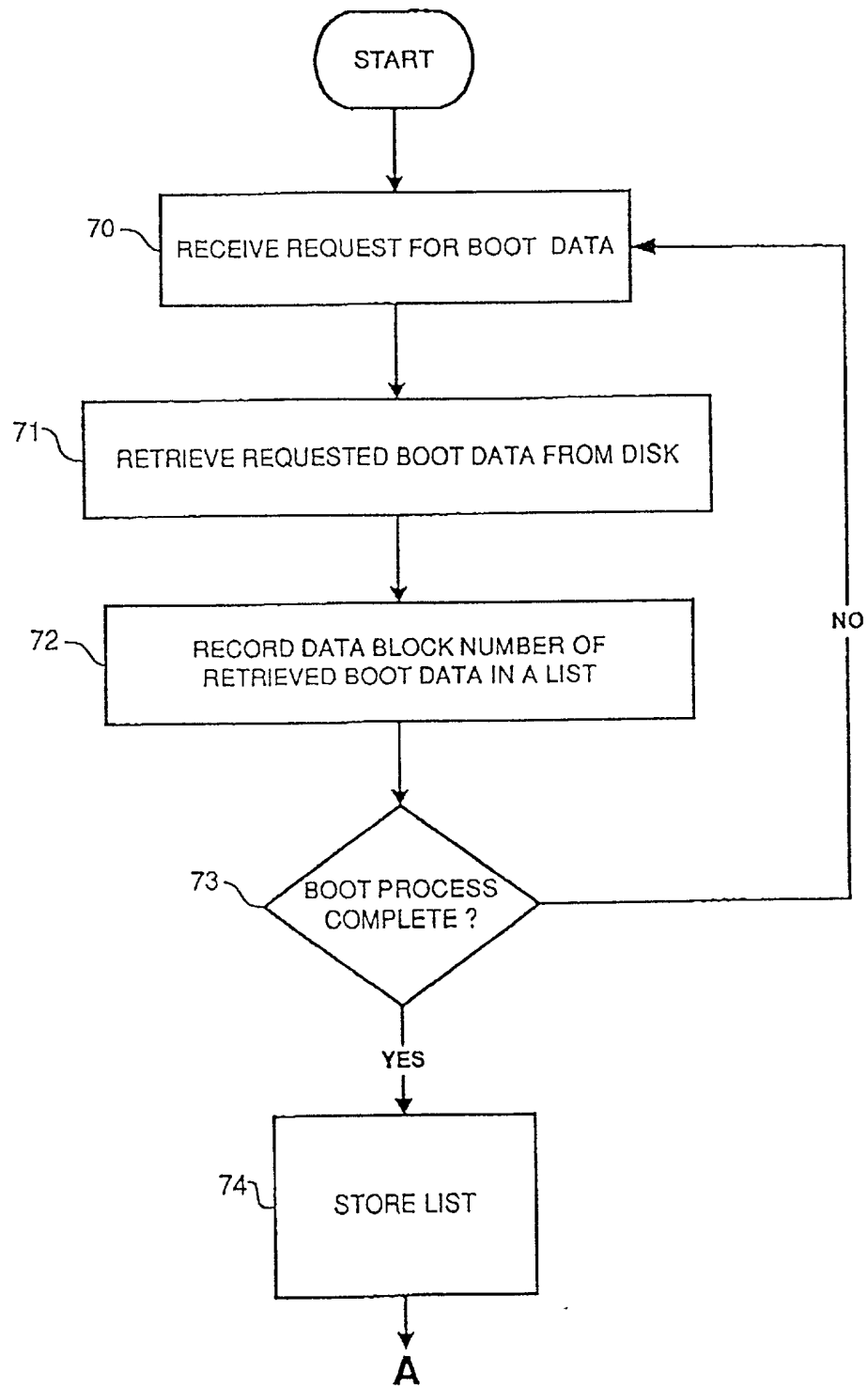


FIG. 7a

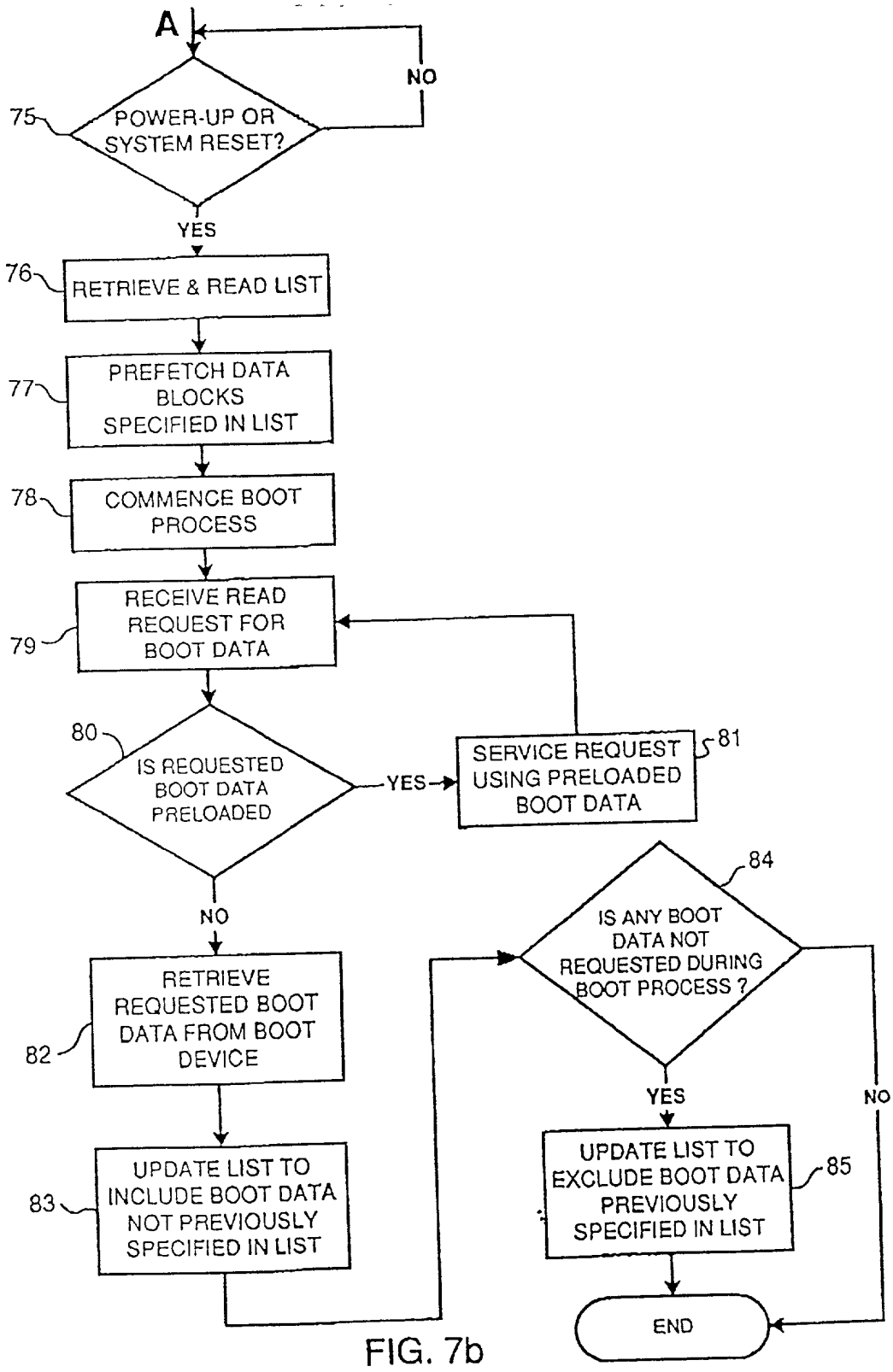


FIG. 7b

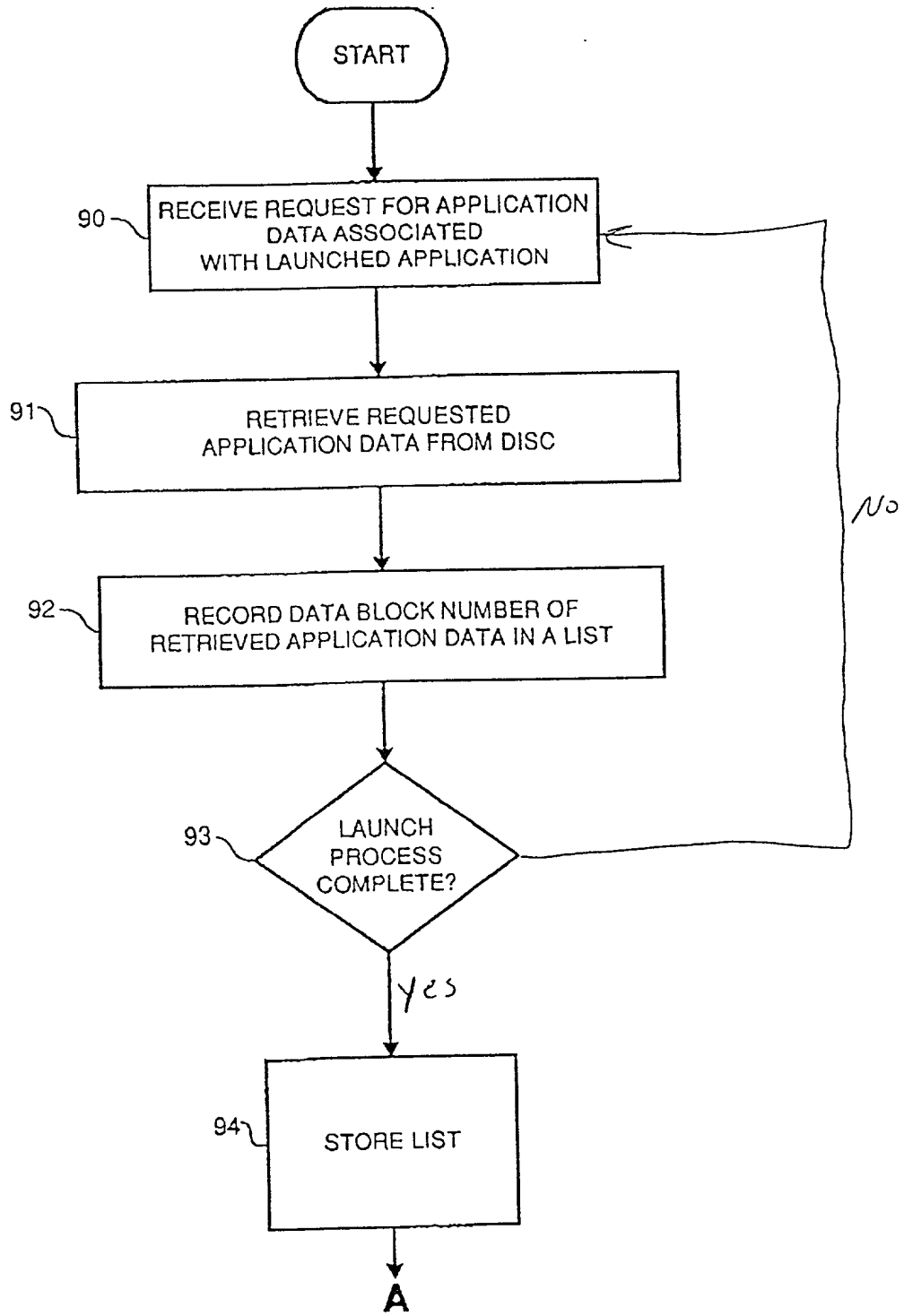


FIG. 8a

TOP SECRET 49292460

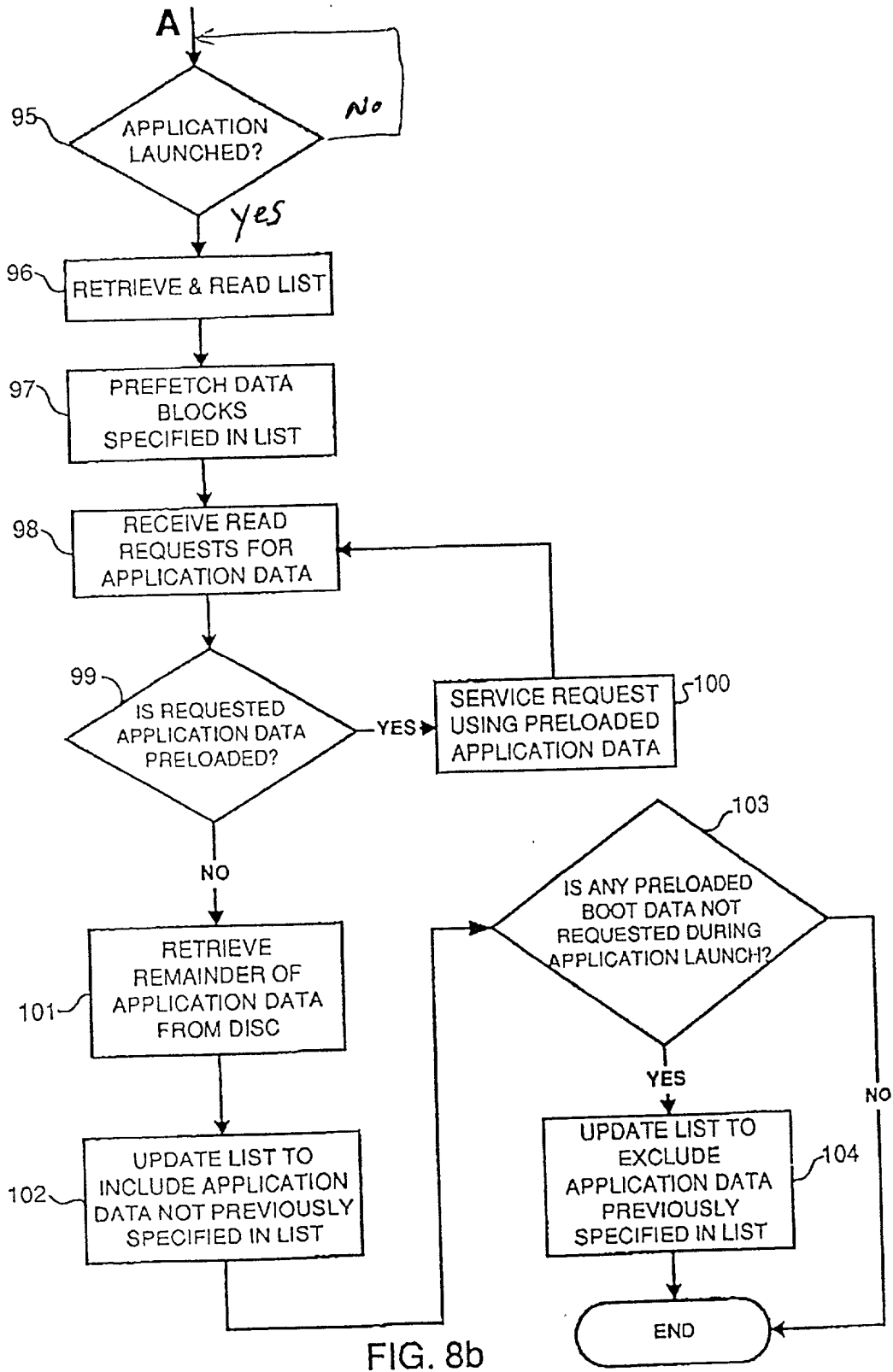


FIG. 8b

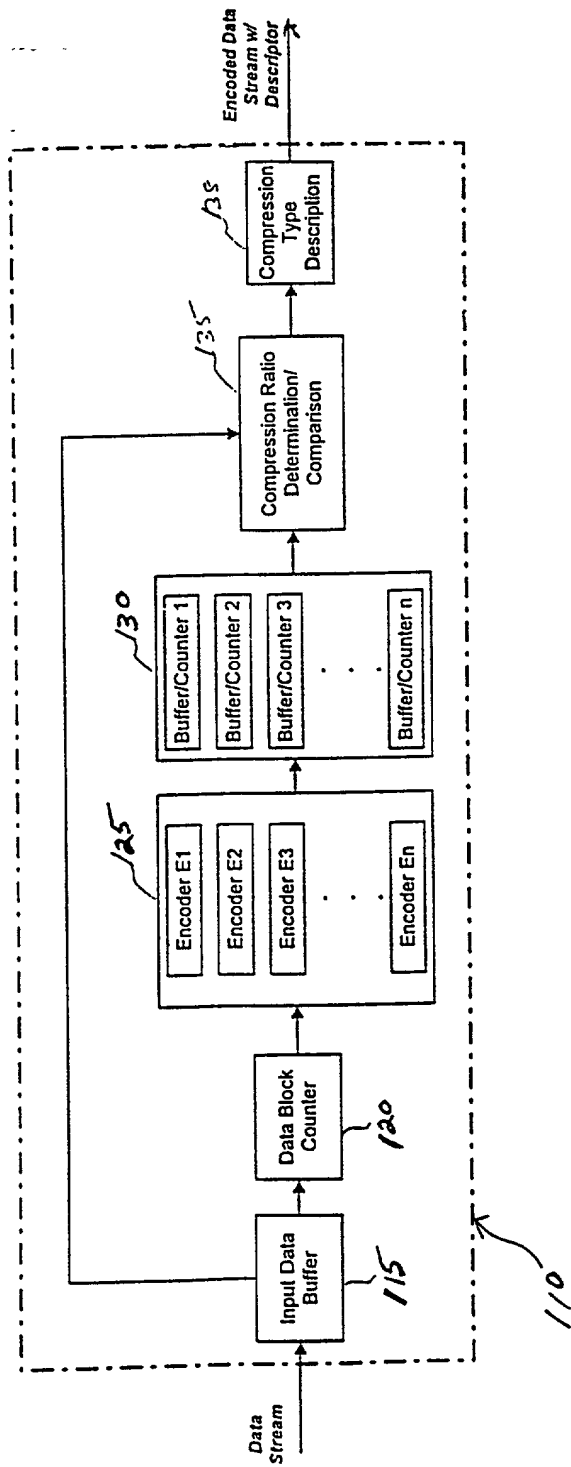


FIGURE 9

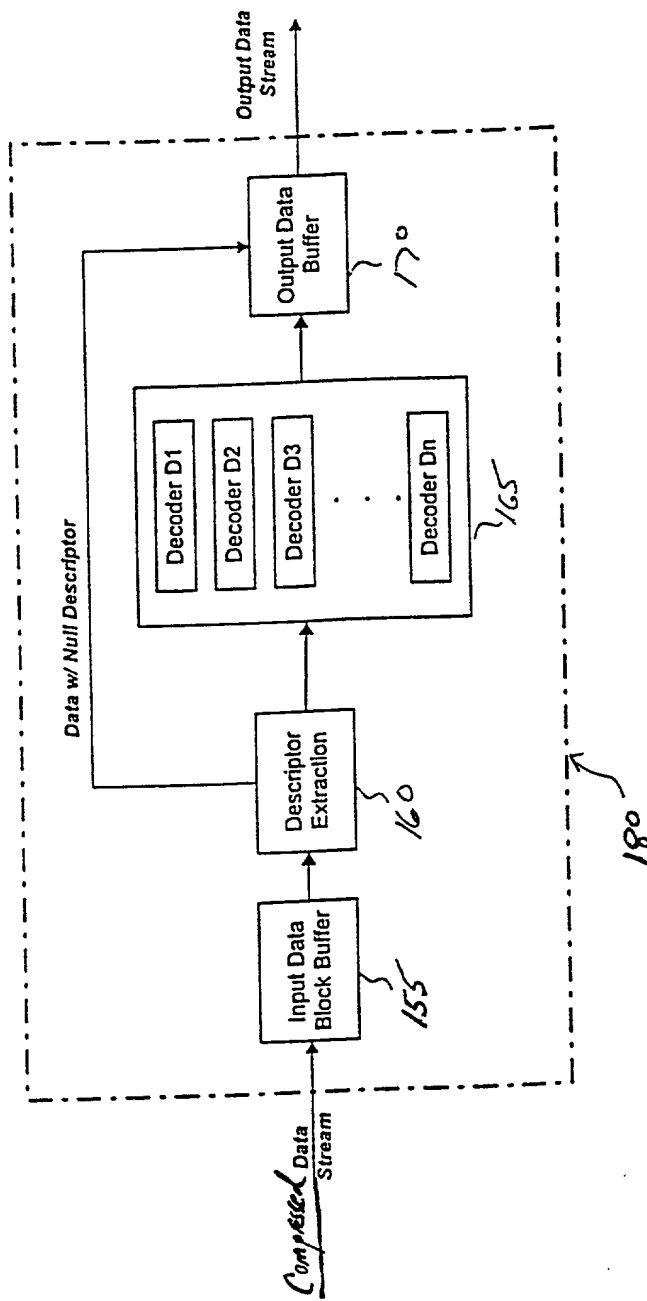
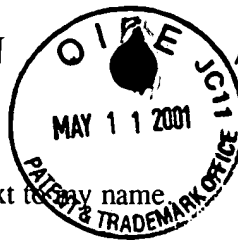


FIG. 10

DECLARATION



AS A BELOW NAMED INVENTOR, I hereby declare that:

My residence, post office address and citizenship are as stated next to my name

I believe that I am the original, first and sole (if only one name is listed below), or an original, first and joint inventor (if plural names are listed below), of the subject matter which is claimed and for which a patent is sought on the invention entitled:

TITLE: SYSTEMS AND METHODS FOR ACCELERATED LOADING OF OPERATING SYSTEMS AND APPLICATION PROGRAMS

the specification of which either is attached hereto or indicates an attorney docket no. _____, or:

[X] was filed in the U.S. Patent & Trademark Office on February 2, 2001 and assigned Serial No. 09/776,267,

[] and (if applicable) was amended on _____

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above. I acknowledge the duty to disclose information which is material to patentability and to the examination of this application in accordance with Title 37 of the Code of Federal Regulations §1.56. I hereby claim foreign priority benefits under Title 35, U.S. Code §119(a)-(d) or §365(b) of any foreign application(s) for patent or inventor's certificate, or §365(a) of any PCT international application which designated at least one country other than the United States, or §119(e) of any United States provisional application(s), listed below and have also identified below any foreign applications for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Priority Claimed: Yes [] No []

(Application Number)	(Country)	Day/Month/Year filed	Yes []	No []

I hereby claim the benefit under Title 35, U.S. Code, §120, of any United States application(s), or §365(c) of any PCT International application designating the United States, listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT International application(s) in the manner provided by the first paragraph of Title 35, U.S. Code, §112, I acknowledge the duty to disclose information material to patentability as defined in Title 37, The Code of Federal Regulations, §1.56(a) which became available between the filing date of the prior application and the national or PCT international filing date of this application:

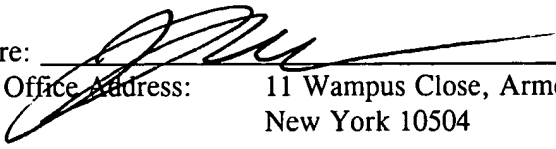
(Application Serial Number)	(Filing Date)	(STATUS: patented, pending, abandoned)
60/180,114	February 3, 2000	Pending

I hereby appoint the following attorneys: FRANK CHAU, Reg. No. 34,136; JAMES J. BITETTO, Reg. No. 40,513, FRANK V. DeROSA, Reg. No. 43,584; and GASPARE J. RANDAZZO, Reg. No. 41,528, each of them of F. CHAU & ASSOCIATES, LLP, 1900 Hempstead Turnpike, Suite 501, East Meadow, New York 11554 to prosecute this application and to transact all business in the U.S. Patent and Trademark Office connected therewith and with any divisional, continuation, continuation-in-part, reissue or re-examination application, with full power of appointment and with full power to substitute an associate attorney or agent, and to receive all patents which may issue thereon, and request that all correspondence be addressed to:

Frank Chau, Esq.
F. CHAU & ASSOCIATES, LLP
1900 Hempstead Turnpike, Suite 501
East Meadow, New York 11554
Area Code: 516-357-0091

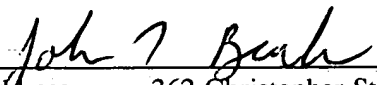
I HEREBY DECLARE that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under §1001 of Title 18 U.S. Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

FULL NAME OF FIRST OR SOLE INVENTOR: James J. Fallon Citizenship USA

Inventor's signature:  Date: 4/20/01

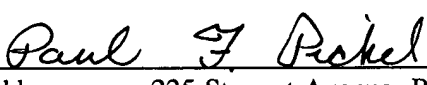
Residence & Post Office Address: 11 Wampus Close, Armonk
New York 10504

FULL NAME OF SECOND INVENTOR: John Buck Citizenship USA

Inventor's signature:  Date: 4/21/01

Residence & Post Office Address: 362 Christopher Street, Oceanside, New York 11572

FULL NAME OF THIRD INVENTOR: Paul F. Pickel Citizenship USA

Inventor's signature:  Date: 4/24/01

Residence & Post Office Address: 225 Stewart Avenue, Bethpage, New York 11714

FULL NAME OF FOURTH INVENTOR: Stephen J. McErlain McERLAIN Citizenship USA

Inventor's signature:  Date: 4/20/01

Residence & Post Office Address: 325 East 17th Street, New York, New York 10003

FULL NAME OF FIFTH INVENTOR: _____ Citizenship _____

Inventor's signature: _____ Date: _____

Residence & Post Office Address: _____

FULL NAME OF SIXTH INVENTOR: _____ Citizenship _____

Inventor's signature: _____ Date: _____

Residence & Post Office Address: _____

FULL NAME OF SEVENTH INVENTOR: _____ Citizenship _____

Inventor's signature: _____ Date: _____

Residence & Post Office Address: _____

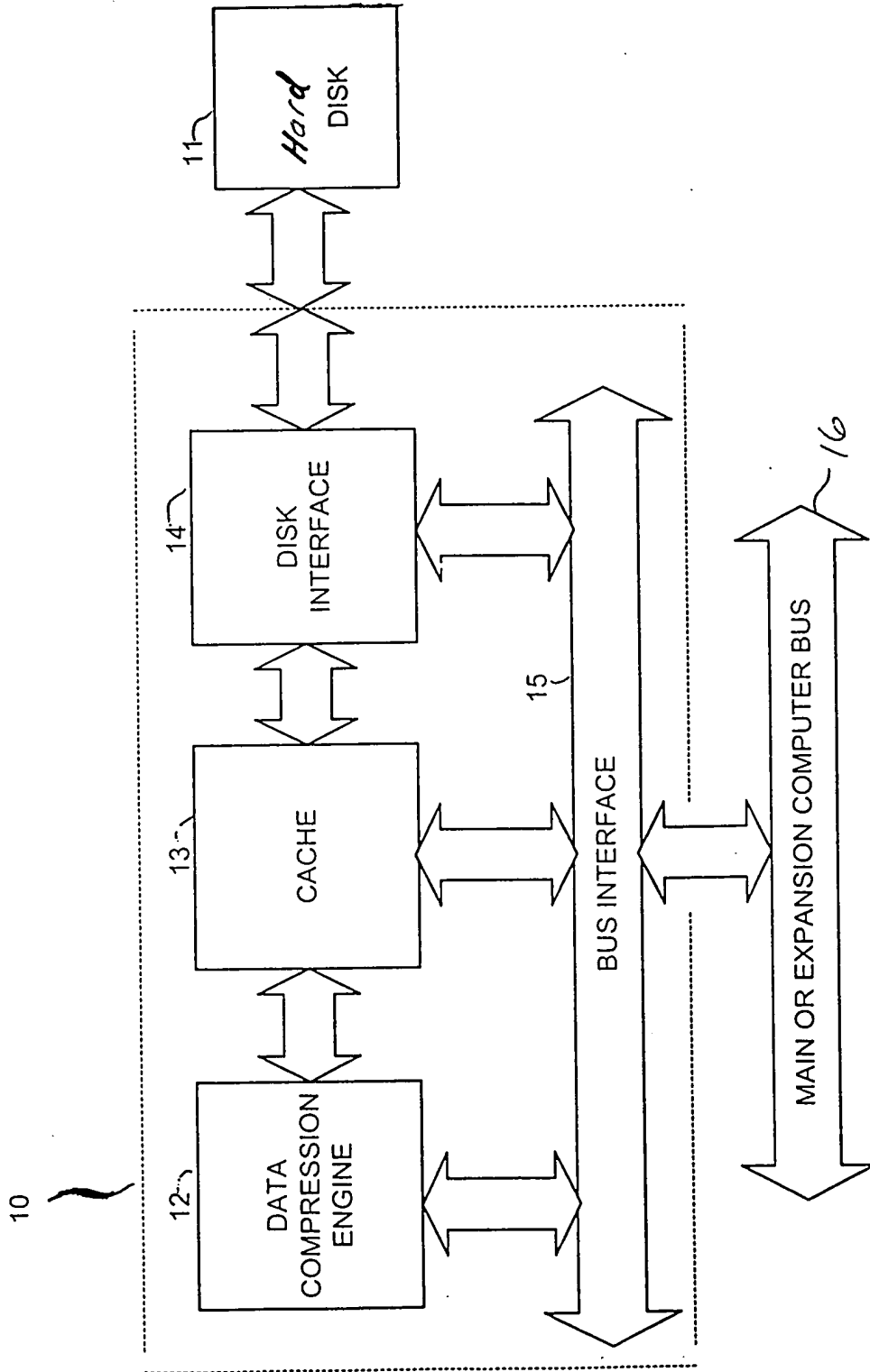


FIGURE 1

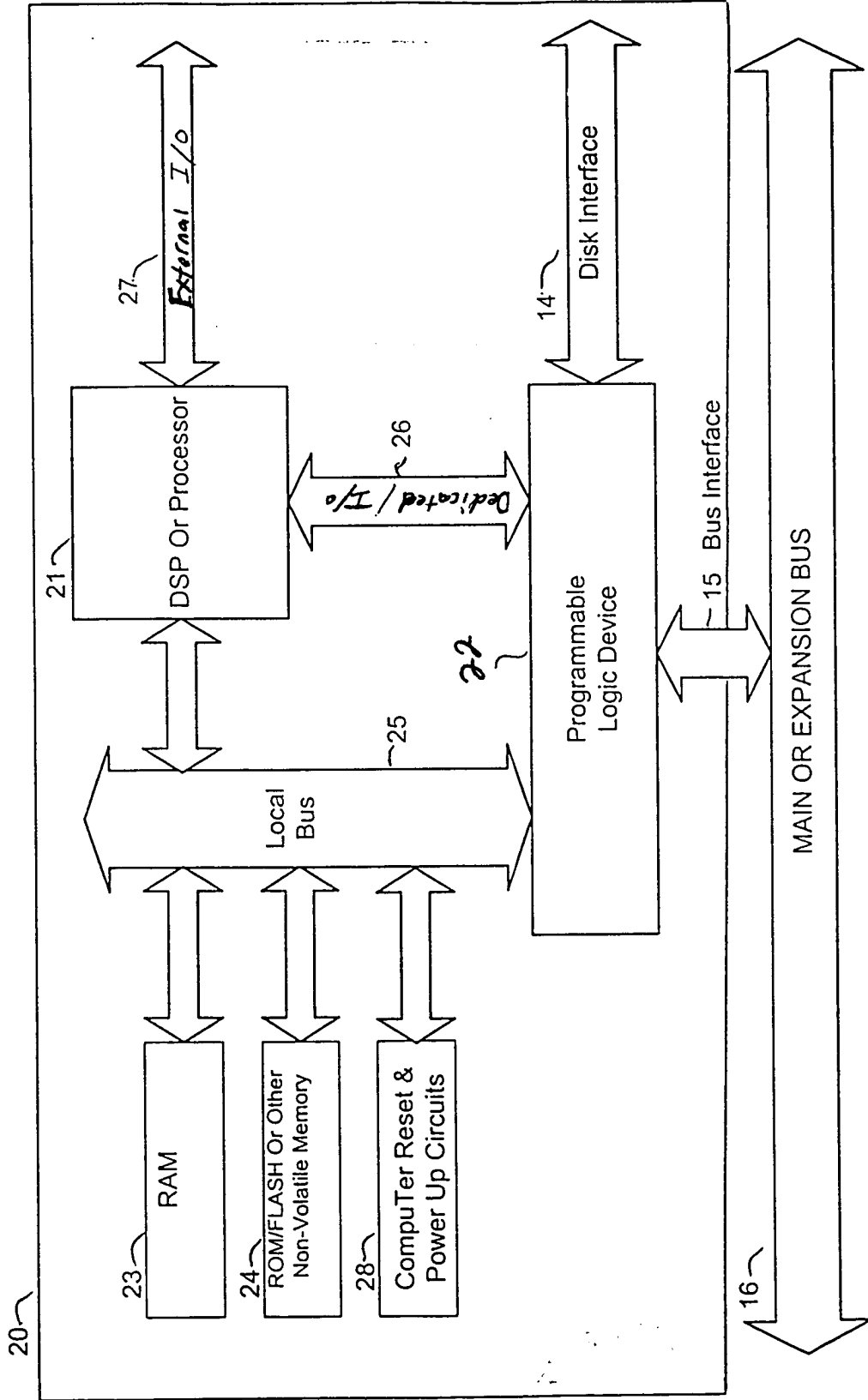


FIGURE 2

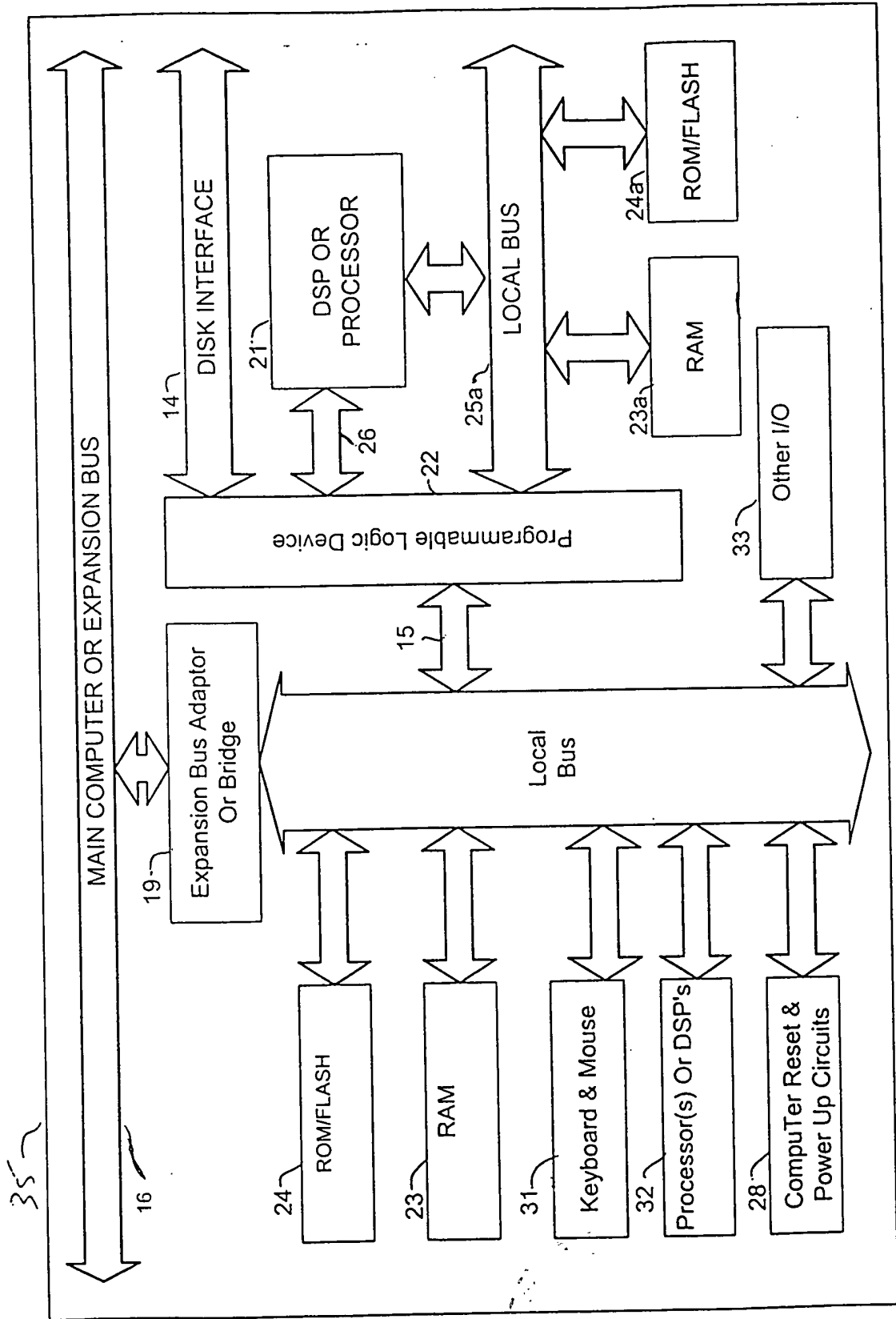


FIGURE 3

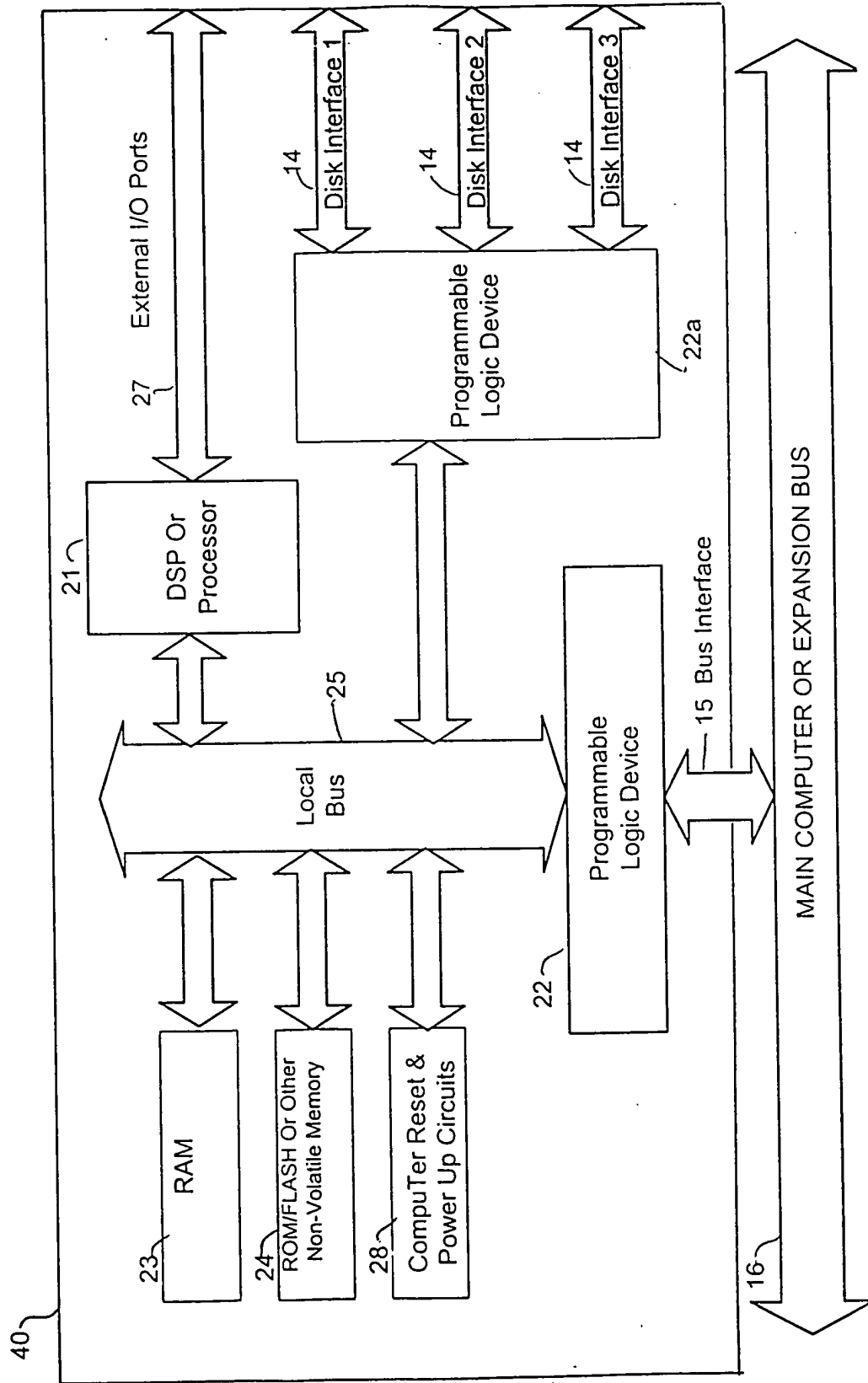


FIGURE 4

4/5

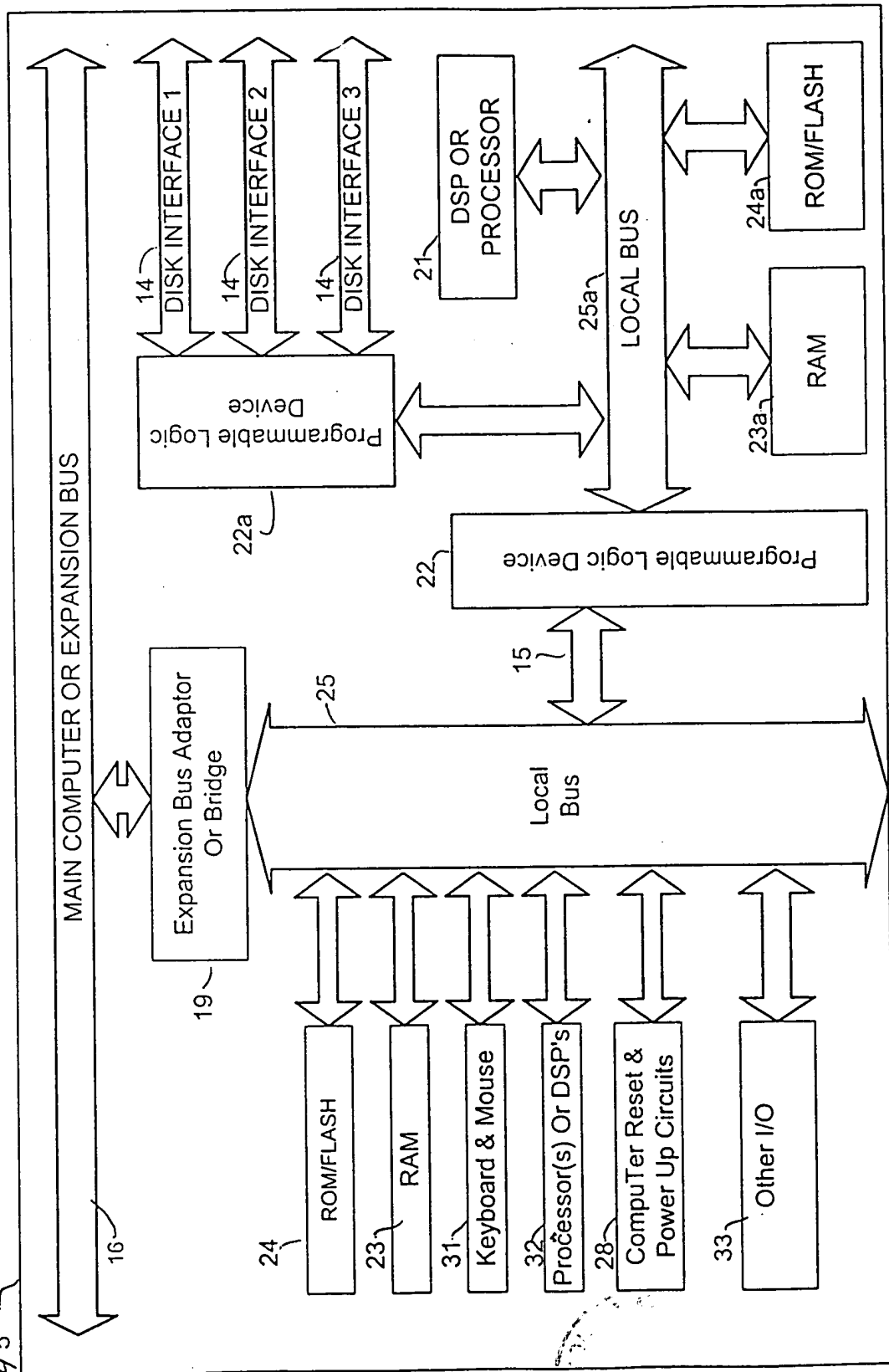


FIGURE 5

FOI 50 4929460

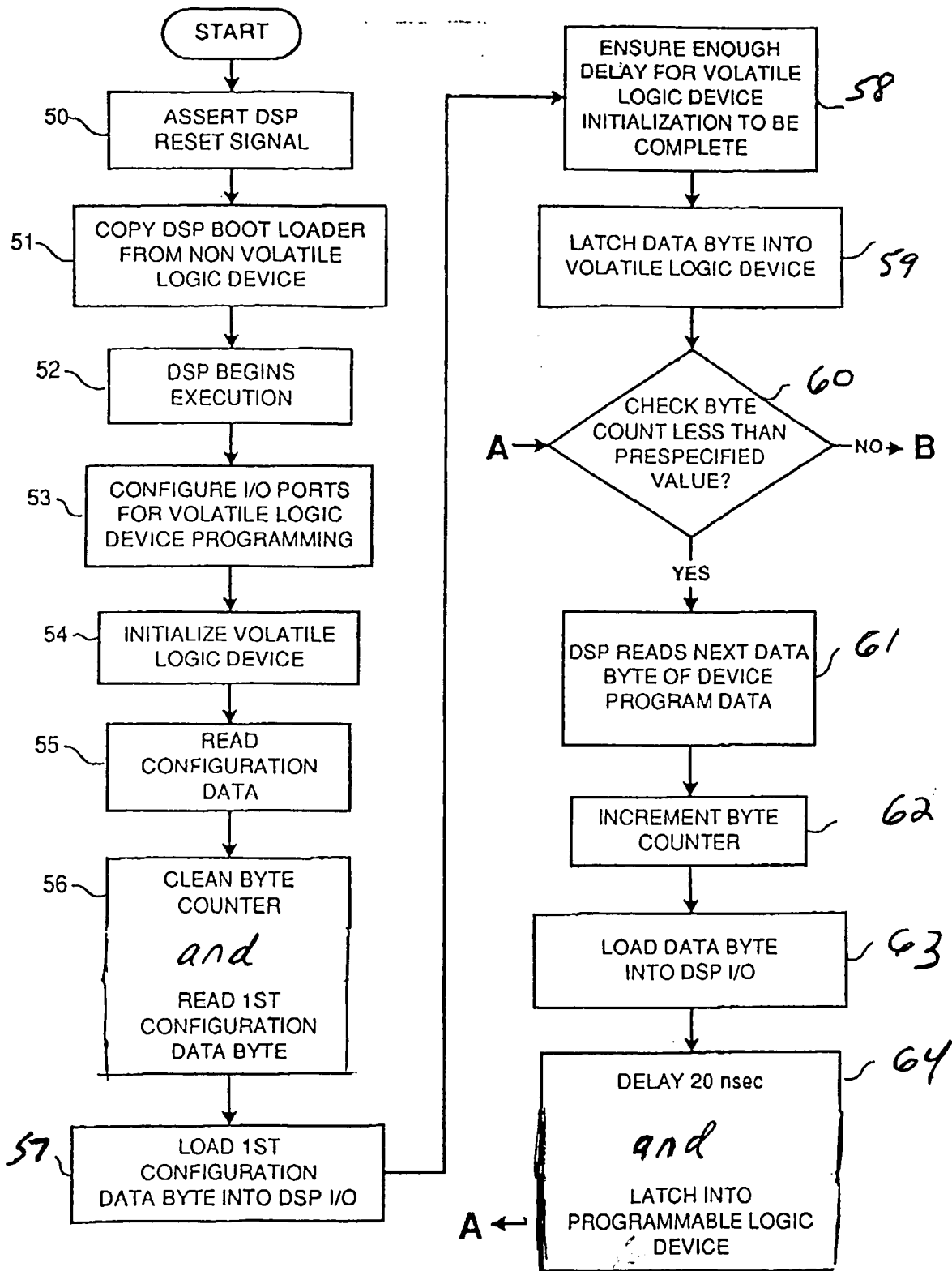


FIG. 6a

TOP SECRET 4929460

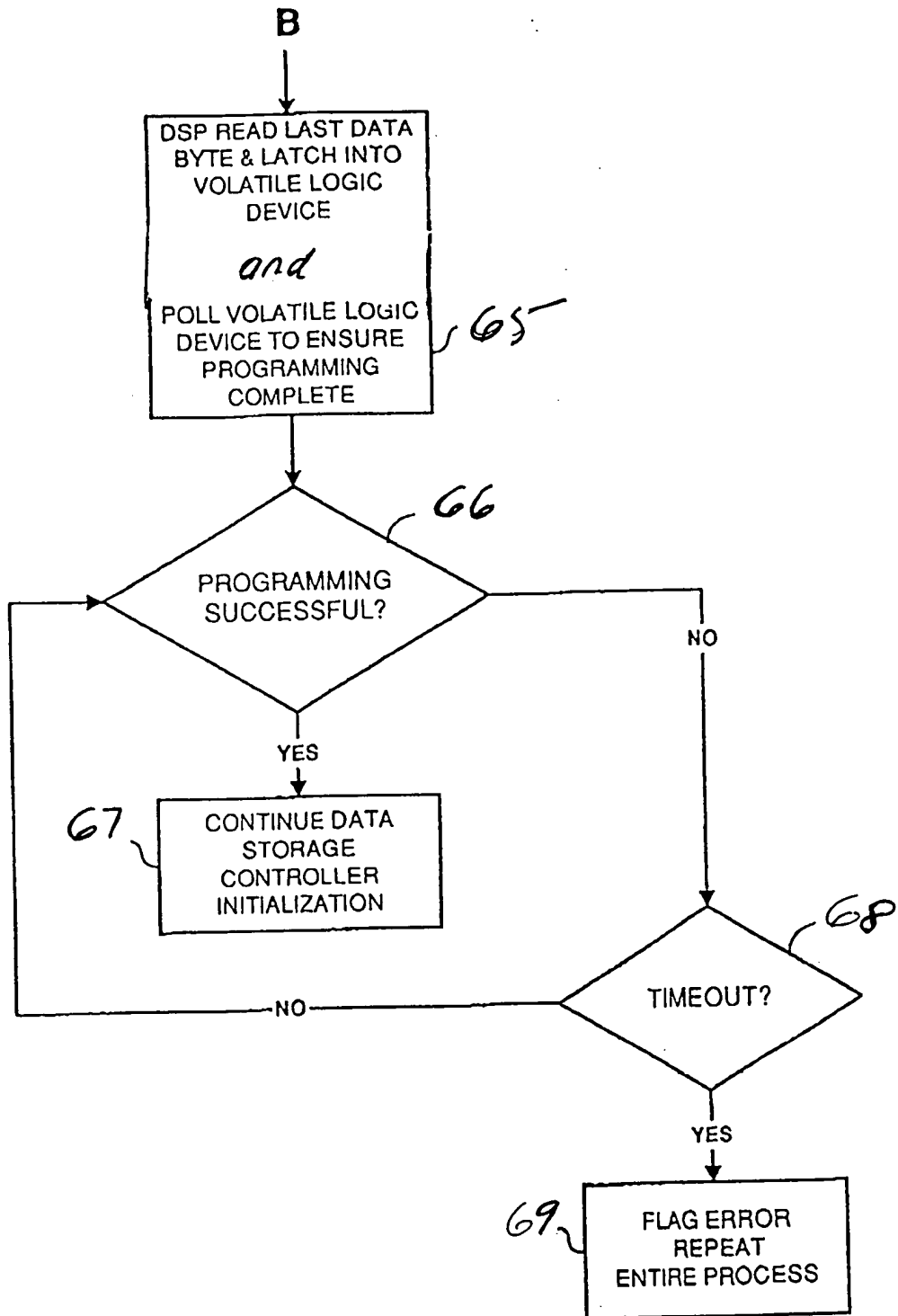


FIG. 6b

FOI b7c b7d

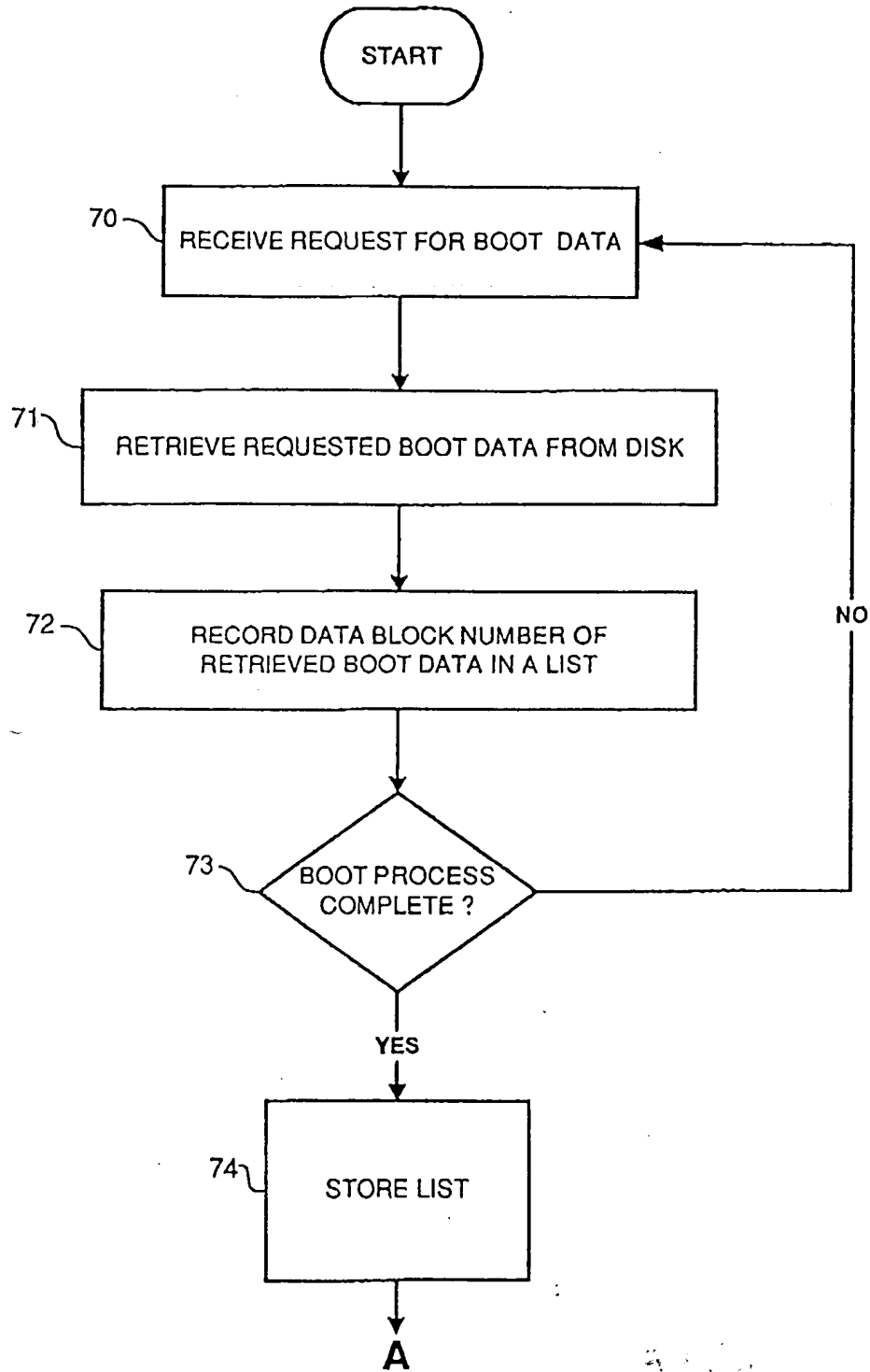


FIG. 7a

FOR F50 4929460

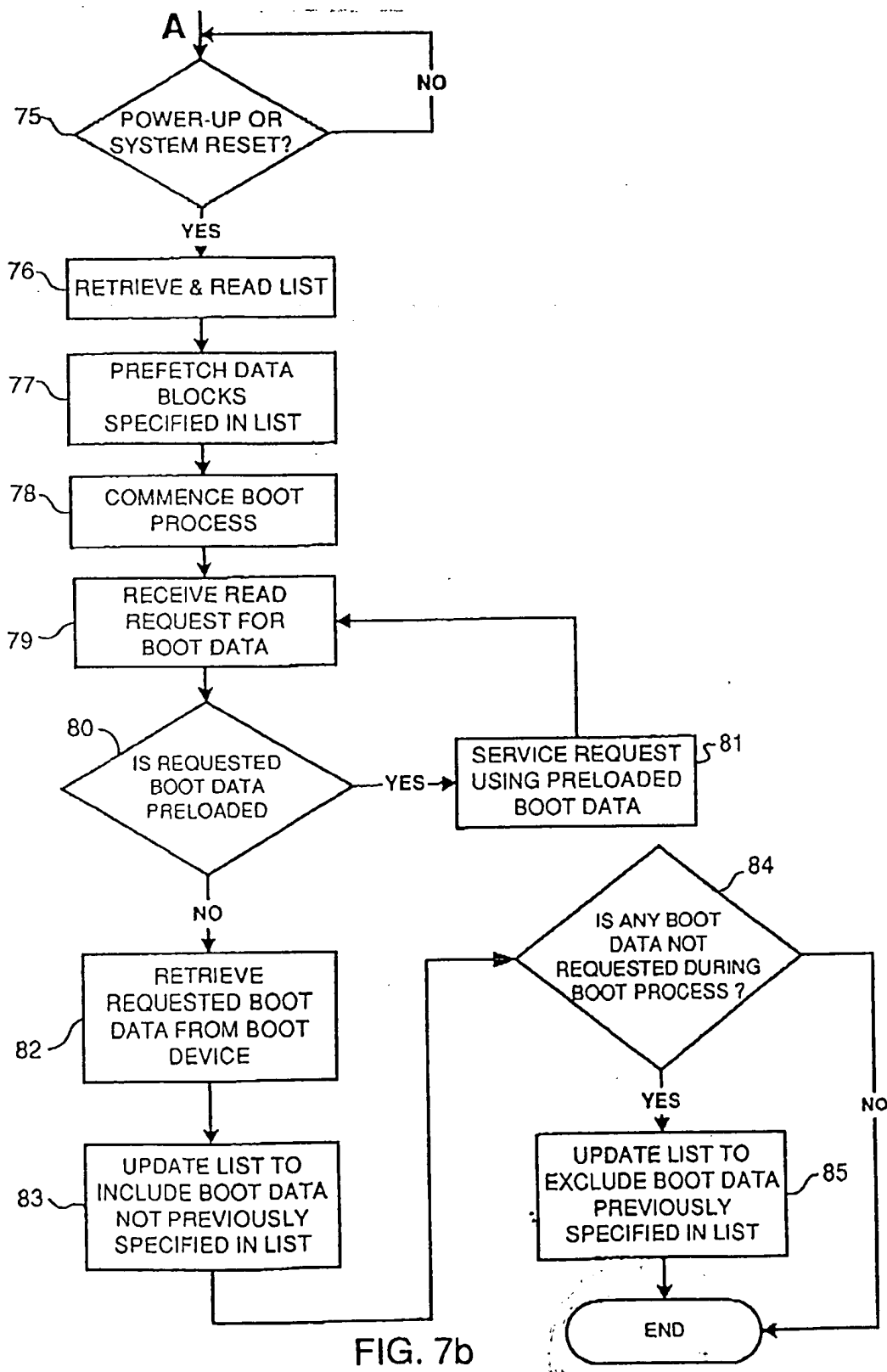


FIG. 7b

FOI 50 4929460

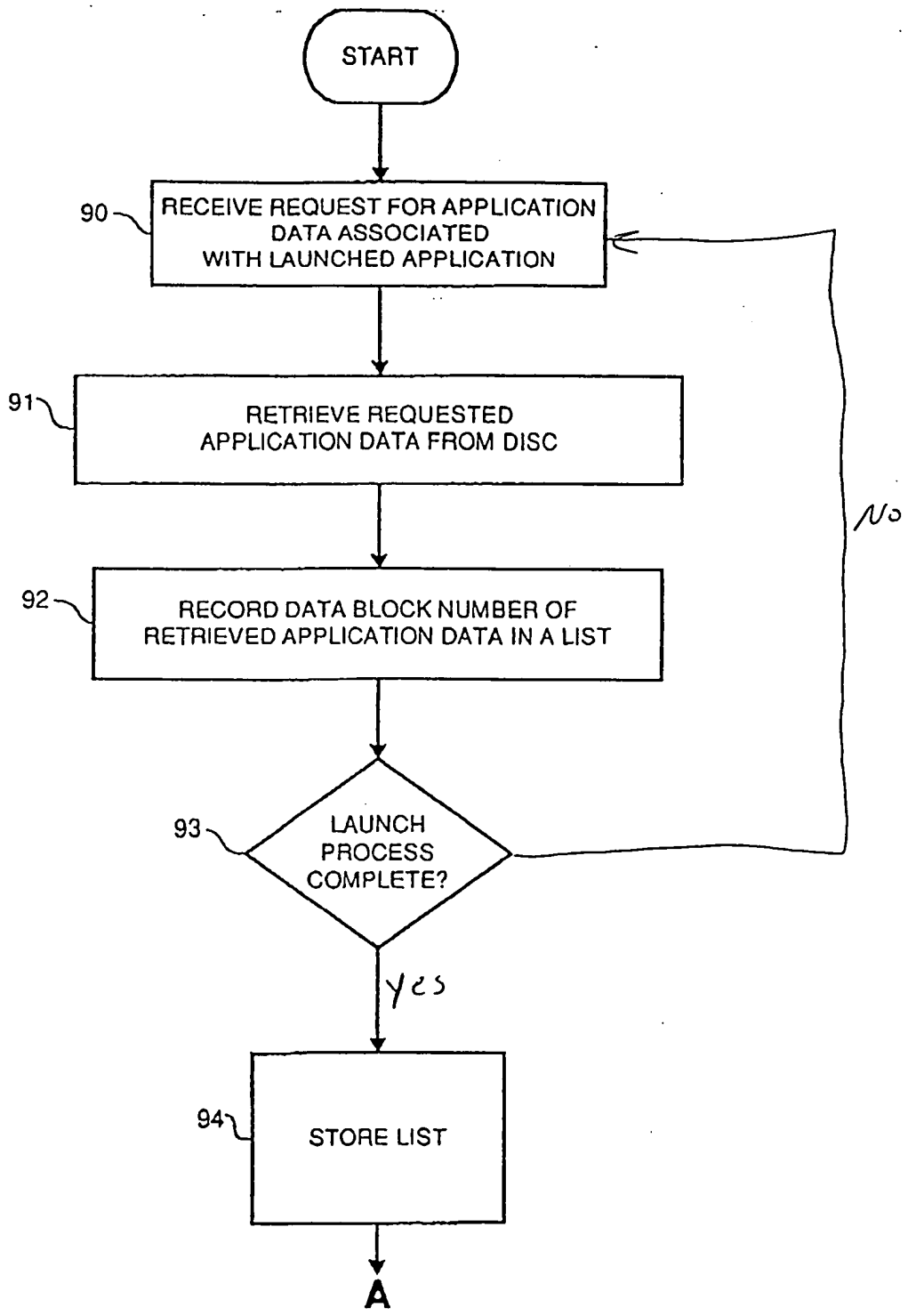


FIG. 8a

FOI 497629760

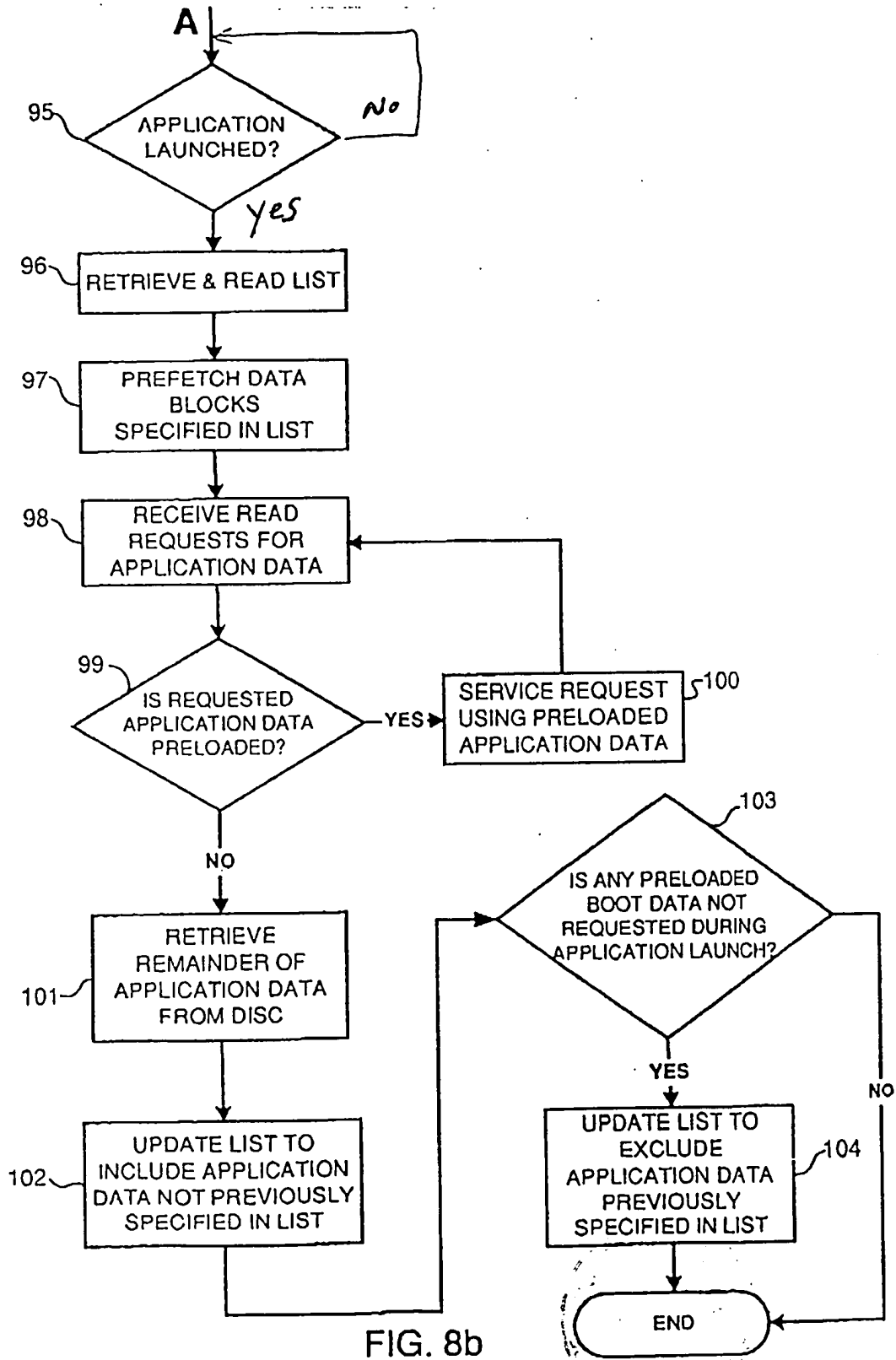


FIG. 8b

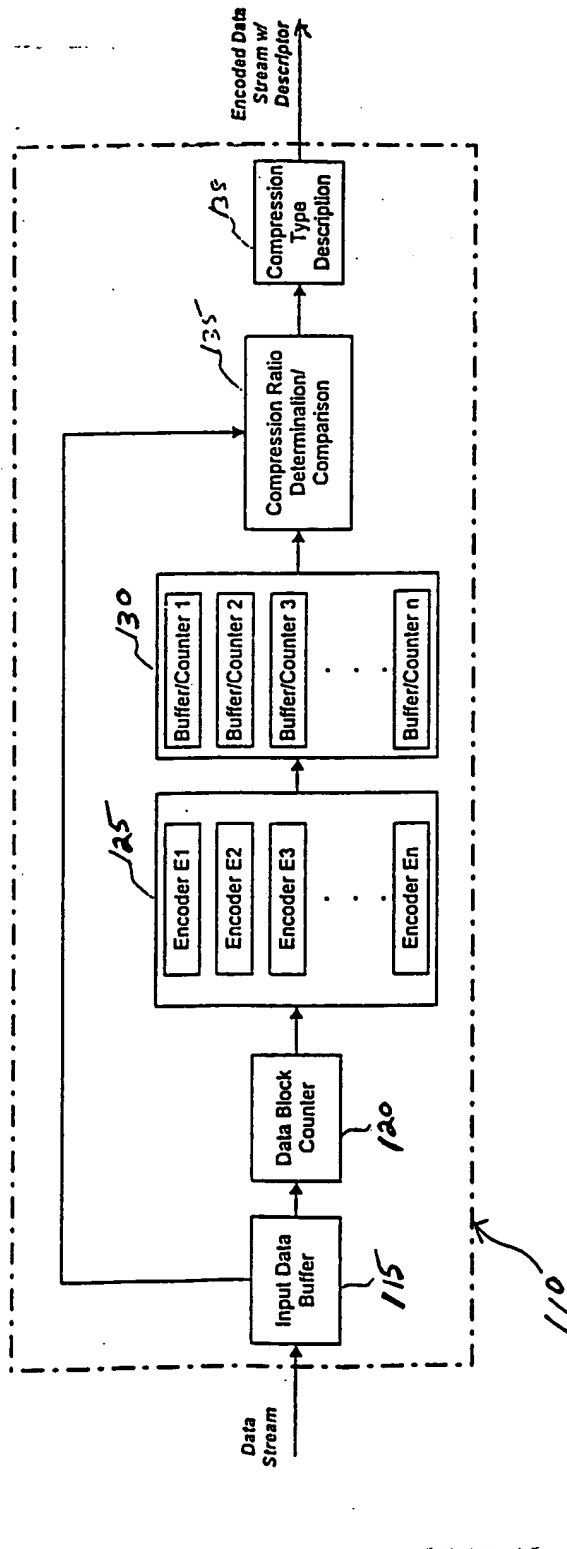


FIGURE 9

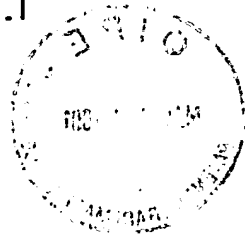
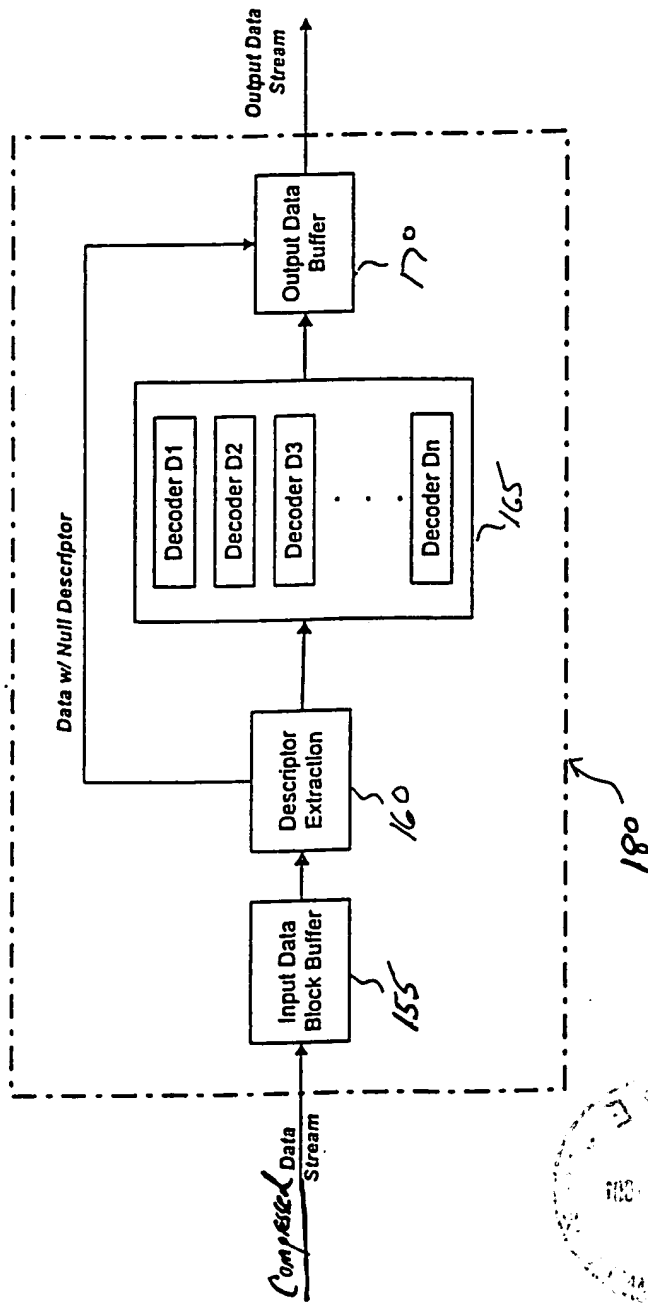
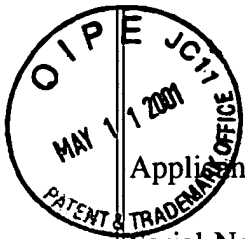


FIG. 10



H.A

\$ SECTOR PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): Fallon et al.

Docket: 8011-15

Serial No.: 09/776,267

Filed: February 2, 2001

For: SYSTEMS AND METHODS FOR ACCELERATED LOADING OF OPERATING SYSTEMS AND APPLICATION PROGRAMS

Assistant Commissioner for Patents
Washington, DC 20231

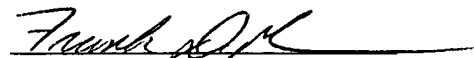
RESPONSE TO NOTICE TO FILE MISSING PARTS OF NONPROVISIONAL APPLICATION

Sir:

In response to the Notice to File Missing Parts of Nonprovisional Application dated March 9, 2001, Applicant encloses herewith an executed Declaration and Power of Attorney in compliance with 37 C.F.R. §1.63; together with substitute drawings in compliance with 37 CFR 1.84. Also enclosed is a check for \$65.00 to cover the surcharge under 37 C.F.R. § 1.16(e) along with a copy of the Notice to File Missing Parts of Nonprovisional Application.

If the enclosed check is insufficient for any reason or becomes detached, please charge the required fee under 37 C.F.R. § 1.16(e) to Deposit Account No. 50-0679. Also, in the event any additional extensions of time are required, please treat this paper as a petition to extend the time as required and charge Deposit Account No. 50-0679. TWO (2) COPIES OF THIS SHEET ARE ENCLOSED.

Respectfully submitted,


Frank V. DeRosa
Registration No. 43,584
Attorney for Applicant(s)

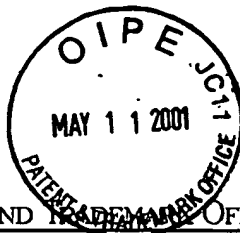
F. CHAU & ASSOCIATES, LLP
1900 Hempstead Turnpike, Suite 510
East Meadow, New York 11554
(516) 357-0091

CERTIFICATE OF MAILING UNDER 37 C.F.R. §1.8(a)

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail, postpaid in an envelope, addressed to the: Assistant Commissioner for Patents, Washington, D.C. 20231 on May 9, 2001.

Dated: 5/9/01


Frank V. DeRosa



UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
 UNITED STATES PATENT AND TRADEMARK OFFICE
 WASHINGTON, D.C. 20231
 www.uspto.gov

APPLICATION NUMBER	FILING/RECEIPT DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NUMBER
09/776,267	02/02/2001	James J. Fallon	8011-15

CONFIRMATION NO. 9730

FORMALITIES LETTER



OC00000005845946

Frank Chau, Esq.
 F. CHAU & ASSOCIATES, LLP
 Suite 501
 1900 Hempstead Turnpike
 East Meadow, NY 11554

Date Mailed: 03/09/2001

NOTICE TO FILE MISSING PARTS OF NONPROVISIONAL APPLICATION

FILED UNDER 37 CFR 1.53(b)

Filing Date Granted

An application number and filing date have been accorded to this application. The item(s) indicated below, however, are missing. Applicant is given TWO MONTHS from the date of this Notice within which to file all required items and pay any fees required below to avoid abandonment. Extensions of time may be obtained by filing a petition accompanied by the extension fee under the provisions of 37 CFR 1.136(a).

- The oath or declaration is missing.
- To avoid abandonment, a late filing fee or oath or declaration surcharge as set forth in 37 CFR 1.16(e) of \$65 for a small entity in compliance with 37 CFR 1.27, must be submitted with the missing items identified in this letter.
- **The balance due by applicant is \$ 65.**

The application is informal since it does not comply with the regulations for the reason(s) indicated below. Applicant is given TWO MONTHS from the date of this Notice within which to correct the informalities indicated below.

The required item(s) identified below must be timely submitted to avoid abandonment:

- Substitute drawings in compliance with 37 CFR 1.84 because:
 - drawing sheets do not have the appropriate margin(s) (see 37 CFR 1.84(g)). Each sheet must include a top margin of at least 2.5 cm. (1 inch), a left side margin of at least 2.5 cm. (1 inch), a right side margin of at least 1.5 cm. (5/8 inch), and a bottom margin of at least 1.0 cm. (3/8 inch);

*A copy of this notice **MUST** be returned with the reply.*

05/15/2001 AZERGAM1 00000050 09776267

01 FC:205

65.00 OP

Realtime 2023
 Page 801 of 964

Ngan

Customer Service Center
Initial Patent Examination Division (703) 308-1202

PART 2 - COPY TO BE RETURNED WITH RESPONSE


UNITED STATES PATENT AND TRADEMARK OFFICE

 COMMISSIONER FOR PATENTS
 UNITED STATES PATENT AND TRADEMARK OFFICE
 WASHINGTON, D.C. 20231
 www.uspto.gov

APPLICATION NUMBER	FILING/RECEIPT DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NUMBER
09/776,267	02/02/2001	James J. Fallon	8011-15

CONFIRMATION NO. 9730
FORMALITIES LETTER


OC00000005845946

 Frank Chau, Esq.
 F. CHAU & ASSOCIATES, LLP
 Suite 501
 1900 Hempstead Turnpike
 East Meadow, NY 11554

Date Mailed: 03/09/2001

NOTICE TO FILE MISSING PARTS OF NONPROVISIONAL APPLICATION
FILED UNDER 37 CFR 1.53(b)
Filing Date Granted

An application number and filing date have been accorded to this application. The item(s) indicated below, however, are missing. Applicant is given **TWO MONTHS** from the date of this Notice within which to file all required items and pay any fees required below to avoid abandonment. Extensions of time may be obtained by filing a petition accompanied by the extension fee under the provisions of 37 CFR 1.136(a).

- The oath or declaration is missing.
- To avoid abandonment, a late filing fee or oath or declaration surcharge as set forth in 37 CFR 1.16(e) of \$65 for a small entity in compliance with 37 CFR 1.27, must be submitted with the missing items identified in this letter.
- **The balance due by applicant is \$ 65.**

The application is informal since it does not comply with the regulations for the reason(s) indicated below. Applicant is given **TWO MONTHS** from the date of this Notice within which to correct the informalities indicated below.

The required item(s) identified below must be timely submitted to avoid abandonment:

- Substitute drawings in compliance with 37 CFR 1.84 because:
 - drawing sheets do not have the appropriate margin(s) (see 37 CFR 1.84(g)). Each sheet must include a top margin of at least 2.5 cm. (1 inch), a left side margin of at least 2.5 cm. (1 inch), a right side margin of at least 1.5 cm. (5/8 inch), and a bottom margin of at least 1.0 cm. (3/8 inch);

*A copy of this notice **MUST** be returned with the reply.*

Ngan

Customer Service Center
Initial Patent Examination Division (703) 308-1202

PART 3 - OFFICE COPY

02-03-01

A

PTO/SB/05 (08-00)

Approved for use through 10/31/2002. OMB 0651-0032

U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Please type a plus sign (+) inside this box → Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

02/02/01
1c965 U.S. PTO

10/20/01
02/02/01
192911/60
U.S. PTO

UTILITY PATENT APPLICATION TRANSMITTAL

Attorney Docket No.	8011-15
First Inventor	FALLON
Title	SYSTEMS AND METHODS FOR ACCELERATED
Express Mail Label No.	EL679454191US

(Only for new nonprovisional applications under 37 CFR 1.53(b))

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

- Fee Transmittal Form (e.g., PTO/SB/17)
(Submit an original and a duplicate for fee processing)
 - Applicant claims small entity status.
See 37 CFR 1.27.
 - Specification [Total Pages *(preferred arrangement set forth below)*
 - Descriptive title of the invention
 - Cross Reference to Related Applications
 - Statement Regarding Fed sponsored R & D
 - Reference to sequence listing, a table, or a computer program listing appendix
 - Background of the Invention
 - Brief Summary of the Invention
 - Brief Description of the Drawings (if filed)
 - Detailed Description
 - Claim(s)
 - Abstract of the Disclosure
 - Drawing(s) (35 U.S.C. 113) [Total Sheets - Oath or Declaration [Total Pages - Newly executed (original or copy)
Copy from a prior application (37 CFR 1.63 (d))
 - (for continuation/divisional with Box 17 completed)
 - DELETION OF INVENTOR(S)**
Signed statement attached deleting inventor(s) named in the prior application, see 37 CFR 1.63(d)(2) and 1.33(b).
- Application Data Sheet. See 37 CFR 1.76

ADDRESS TO: Assistant Commissioner for Patents
Box Patent Application
Washington, DC 20231

- CD-ROM or CD-R in duplicate, large table or Computer Program (Appendix)
- Nucleotide and/or Amino Acid Sequence Submission (if applicable, all necessary)
 - Computer Readable Form (CRF)
 - Specification Sequence Listing on:
 - CD-ROM or CD-R (2 copies); or
 - paper
 - Statements verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

- Assignment Papers (cover sheet & document(s))
- 37 CFR 3.73(b) Statement (when there is an assignee) Power of Attorney
- English Translation Document (if applicable)
- Information Disclosure Statement (IDS)/PTO-1449 Copies of IDS Citations
- Preliminary Amendment
- Return Receipt Postcard (MPEP 503) (Should be specifically itemized)
- Certified Copy of Priority Document(s) (if foreign priority is claimed)
- Other: Check for \$355.00

17. If a CONTINUING APPLICATION, check appropriate box, and supply the requisite information below and in a preliminary amendment, or in an Application Data Sheet under 37 CFR 1.76:

Continuation Divisional Continuation-in-part (CIP) of prior application No. _____ / _____

Prior application information: Examiner: _____ Group / Art Unit: _____

For CONTINUATION OR DIVISIONAL APPS only: The entire disclosure of the prior application, from which an oath or declaration is supplied under Box 5b, is considered a part of the disclosure of the accompanying continuation or divisional application and is hereby incorporated by reference. The incorporation can only be relied upon when a portion has been inadvertently omitted from the submitted application parts.

18. CORRESPONDENCE ADDRESS

Customer Number or Bar Code Label Correspondence address below

(insert Customer No. or Attach bar code label here)

Name	Frank Chau, Esq.			
Address	F. CHAU & ASSOCIATES, LLP			
	1900 Hempstead Turnpike, Suite 501			
City	East Meadow	State	New York	Zip Code 11554
Country	USA	Telephone	516-357-0091	Fax 516-357-0092

Name (Print/Type)	Frank V. DeRosa	Registration No. (Attorney/Agent)	43,584
Signature		Date	2/2/01

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

<h1 style="margin: 0;">FEE TRANSMITTAL</h1> <h2 style="margin: 0;">for FY 2001</h2> <p style="font-size: small; margin: 5px 0;">Patent fees are subject to annual revision.</p>	Complete if Known	
	Application Number	
	Filing Date	February 2, 2001
	First Named Inventor	James J. Fallon
	Examiner Name	
	Group Art Unit	
TOTAL AMOUNT OF PAYMENT	(\$ 355.00)	
	Attorney Docket No.	8011-15

METHOD OF PAYMENT	FEE CALCULATION (continued)																																																																																																																								
<p>1. <input type="checkbox"/> The Commissioner is hereby authorized to charge indicated fees and credit any overpayments to:</p> <p>Deposit Account Number: <u>50-0679</u></p> <p>Deposit Account Name: <u>F. Chau & Associates, LLP</u></p> <p><input type="checkbox"/> Charge Any Additional Fee Required Under 37 CFR 1.16 and 1.17</p> <p><input type="checkbox"/> Applicant claims small entity status See 37 CFR 1.27</p> <p>2. <input checked="" type="checkbox"/> Payment Enclosed: <input checked="" type="checkbox"/> Check <input type="checkbox"/> Credit card <input type="checkbox"/> Money Order <input type="checkbox"/> Other</p>	<p>3. ADDITIONAL FEES</p> <table border="1" style="width: 100%; border-collapse: collapse; font-size: x-small;"> <thead> <tr> <th>Large Entity Fee Code (\$)</th> <th>Small Entity Fee Code (\$)</th> <th>Fee Description</th> <th>Fee Paid</th> </tr> </thead> <tbody> <tr><td>105 130</td><td>205 65</td><td>Surcharge - late filing fee or oath</td><td></td></tr> <tr><td>127 50</td><td>227 25</td><td>Surcharge - late provisional filing fee or cover sheet</td><td></td></tr> <tr><td>139 130</td><td>139 130</td><td>Non-English specification</td><td></td></tr> <tr><td>147 2,520</td><td>147 2,520</td><td>For filing a request for ex parte reexamination</td><td></td></tr> <tr><td>112 920*</td><td>112 920*</td><td>Requesting publication of SIR prior to Examiner action</td><td></td></tr> <tr><td>113 1,840*</td><td>113 1,840*</td><td>Requesting publication of SIR after Examiner action</td><td></td></tr> <tr><td>115 110</td><td>215 55</td><td>Extension for reply within first month</td><td></td></tr> <tr><td>116 390</td><td>216 195</td><td>Extension for reply within second month</td><td></td></tr> <tr><td>117 890</td><td>217 445</td><td>Extension for reply within third month</td><td></td></tr> <tr><td>118 1,390</td><td>218 695</td><td>Extension for reply within fourth month</td><td></td></tr> <tr><td>128 1,890</td><td>228 945</td><td>Extension for reply within fifth month</td><td></td></tr> <tr><td>119 310</td><td>219 155</td><td>Notice of Appeal</td><td></td></tr> <tr><td>120 310</td><td>220 155</td><td>Filing a brief in support of an appeal</td><td></td></tr> <tr><td>121 270</td><td>221 135</td><td>Request for oral hearing</td><td></td></tr> <tr><td>138 1,510</td><td>138 1,510</td><td>Petition to institute a public use proceeding</td><td></td></tr> <tr><td>140 110</td><td>240 55</td><td>Petition to revive - unavoidable</td><td></td></tr> <tr><td>141 1,240</td><td>241 620</td><td>Petition to revive - unintentional</td><td></td></tr> <tr><td>142 1,240</td><td>242 620</td><td>Utility issue fee (or reissue)</td><td></td></tr> <tr><td>143 440</td><td>243 220</td><td>Design issue fee</td><td></td></tr> <tr><td>144 600</td><td>244 300</td><td>Plant issue fee</td><td></td></tr> <tr><td>122 130</td><td>122 130</td><td>Petitions to the Commissioner</td><td></td></tr> <tr><td>123 50</td><td>123 50</td><td>Petitions related to provisional applications</td><td></td></tr> <tr><td>126 240</td><td>126 240</td><td>Submission of Information Disclosure Stmt</td><td></td></tr> <tr><td>581 40</td><td>581 40</td><td>Recording each patent assignment per property (times number of properties)</td><td></td></tr> <tr><td>146 710</td><td>246 355</td><td>Filing a submission after final rejection (37 CFR § 1.129(a))</td><td></td></tr> <tr><td>149 710</td><td>249 355</td><td>For each additional invention to be examined (37 CFR § 1.129(b))</td><td></td></tr> <tr><td>179 710</td><td>279 355</td><td>Request for Continued Examination (RCE)</td><td></td></tr> <tr><td>169 900</td><td>169 900</td><td>Request for expedited examination of a design application</td><td></td></tr> <tr><td colspan="3">Other fee (specify) _____</td><td></td></tr> </tbody> </table>	Large Entity Fee Code (\$)	Small Entity Fee Code (\$)	Fee Description	Fee Paid	105 130	205 65	Surcharge - late filing fee or oath		127 50	227 25	Surcharge - late provisional filing fee or cover sheet		139 130	139 130	Non-English specification		147 2,520	147 2,520	For filing a request for ex parte reexamination		112 920*	112 920*	Requesting publication of SIR prior to Examiner action		113 1,840*	113 1,840*	Requesting publication of SIR after Examiner action		115 110	215 55	Extension for reply within first month		116 390	216 195	Extension for reply within second month		117 890	217 445	Extension for reply within third month		118 1,390	218 695	Extension for reply within fourth month		128 1,890	228 945	Extension for reply within fifth month		119 310	219 155	Notice of Appeal		120 310	220 155	Filing a brief in support of an appeal		121 270	221 135	Request for oral hearing		138 1,510	138 1,510	Petition to institute a public use proceeding		140 110	240 55	Petition to revive - unavoidable		141 1,240	241 620	Petition to revive - unintentional		142 1,240	242 620	Utility issue fee (or reissue)		143 440	243 220	Design issue fee		144 600	244 300	Plant issue fee		122 130	122 130	Petitions to the Commissioner		123 50	123 50	Petitions related to provisional applications		126 240	126 240	Submission of Information Disclosure Stmt		581 40	581 40	Recording each patent assignment per property (times number of properties)		146 710	246 355	Filing a submission after final rejection (37 CFR § 1.129(a))		149 710	249 355	For each additional invention to be examined (37 CFR § 1.129(b))		179 710	279 355	Request for Continued Examination (RCE)		169 900	169 900	Request for expedited examination of a design application		Other fee (specify) _____			
Large Entity Fee Code (\$)	Small Entity Fee Code (\$)	Fee Description	Fee Paid																																																																																																																						
105 130	205 65	Surcharge - late filing fee or oath																																																																																																																							
127 50	227 25	Surcharge - late provisional filing fee or cover sheet																																																																																																																							
139 130	139 130	Non-English specification																																																																																																																							
147 2,520	147 2,520	For filing a request for ex parte reexamination																																																																																																																							
112 920*	112 920*	Requesting publication of SIR prior to Examiner action																																																																																																																							
113 1,840*	113 1,840*	Requesting publication of SIR after Examiner action																																																																																																																							
115 110	215 55	Extension for reply within first month																																																																																																																							
116 390	216 195	Extension for reply within second month																																																																																																																							
117 890	217 445	Extension for reply within third month																																																																																																																							
118 1,390	218 695	Extension for reply within fourth month																																																																																																																							
128 1,890	228 945	Extension for reply within fifth month																																																																																																																							
119 310	219 155	Notice of Appeal																																																																																																																							
120 310	220 155	Filing a brief in support of an appeal																																																																																																																							
121 270	221 135	Request for oral hearing																																																																																																																							
138 1,510	138 1,510	Petition to institute a public use proceeding																																																																																																																							
140 110	240 55	Petition to revive - unavoidable																																																																																																																							
141 1,240	241 620	Petition to revive - unintentional																																																																																																																							
142 1,240	242 620	Utility issue fee (or reissue)																																																																																																																							
143 440	243 220	Design issue fee																																																																																																																							
144 600	244 300	Plant issue fee																																																																																																																							
122 130	122 130	Petitions to the Commissioner																																																																																																																							
123 50	123 50	Petitions related to provisional applications																																																																																																																							
126 240	126 240	Submission of Information Disclosure Stmt																																																																																																																							
581 40	581 40	Recording each patent assignment per property (times number of properties)																																																																																																																							
146 710	246 355	Filing a submission after final rejection (37 CFR § 1.129(a))																																																																																																																							
149 710	249 355	For each additional invention to be examined (37 CFR § 1.129(b))																																																																																																																							
179 710	279 355	Request for Continued Examination (RCE)																																																																																																																							
169 900	169 900	Request for expedited examination of a design application																																																																																																																							
Other fee (specify) _____																																																																																																																									
<p style="text-align: center; font-weight: bold;">FEE CALCULATION</p> <p>1. BASIC FILING FEE</p> <table border="1" style="width: 100%; border-collapse: collapse; font-size: x-small;"> <thead> <tr> <th>Large Entity Fee Code (\$)</th> <th>Small Entity Fee Code (\$)</th> <th>Fee Description</th> <th>Fee Paid</th> </tr> </thead> <tbody> <tr><td>101 710</td><td>201 355</td><td>Utility filing fee</td><td>355</td></tr> <tr><td>106 320</td><td>206 160</td><td>Design filing fee</td><td></td></tr> <tr><td>107 490</td><td>207 245</td><td>Plant filing fee</td><td></td></tr> <tr><td>108 710</td><td>208 355</td><td>Reissue filing fee</td><td></td></tr> <tr><td>114 150</td><td>214 75</td><td>Provisional filing fee</td><td></td></tr> <tr><td colspan="3" style="text-align: right;">SUBTOTAL (1)</td><td>(\$ 355.00)</td></tr> </tbody> </table> <p>2. EXTRA CLAIM FEES</p> <table style="width: 100%; font-size: x-small;"> <tr> <td>Total Claims</td> <td><u>16</u></td> <td>-20** =</td> <td><u>0</u></td> <td>x</td> <td>Fee from below</td> <td><u>9</u></td> <td>=</td> <td><u>0</u></td> <td>Fee Paid</td> </tr> <tr> <td>Independent Claims</td> <td><u>3</u></td> <td>-3** =</td> <td><u>0</u></td> <td>x</td> <td><u>40</u></td> <td></td> <td>=</td> <td><u>0</u></td> <td></td> </tr> <tr> <td>Multiple Dependent</td> <td></td> <td></td> <td></td> <td></td> <td><u>135</u></td> <td></td> <td>=</td> <td></td> <td></td> </tr> </table> <table border="1" style="width: 100%; border-collapse: collapse; font-size: x-small;"> <thead> <tr> <th>Large Entity Fee Code (\$)</th> <th>Small Entity Fee Code (\$)</th> <th>Fee Description</th> <th>Fee Paid</th> </tr> </thead> <tbody> <tr><td>103 18</td><td>203 9</td><td>Claims in excess of 20</td><td></td></tr> <tr><td>102 80</td><td>202 40</td><td>Independent claims in excess of 3</td><td></td></tr> <tr><td>104 270</td><td>204 135</td><td>Multiple dependent claim, if not paid</td><td></td></tr> <tr><td>109 80</td><td>209 40</td><td>** Reissue independent claims over original patent</td><td></td></tr> <tr><td>110 18</td><td>210 9</td><td>** Reissue claims in excess of 20 and over original patent</td><td></td></tr> <tr><td colspan="3" style="text-align: right;">SUBTOTAL (2)</td><td>(\$ 0)</td></tr> </tbody> </table> <p style="font-size: x-small;">**or number previously paid, if greater; For Reissues, see above</p>	Large Entity Fee Code (\$)	Small Entity Fee Code (\$)	Fee Description	Fee Paid	101 710	201 355	Utility filing fee	355	106 320	206 160	Design filing fee		107 490	207 245	Plant filing fee		108 710	208 355	Reissue filing fee		114 150	214 75	Provisional filing fee		SUBTOTAL (1)			(\$ 355.00)	Total Claims	<u>16</u>	-20** =	<u>0</u>	x	Fee from below	<u>9</u>	=	<u>0</u>	Fee Paid	Independent Claims	<u>3</u>	-3** =	<u>0</u>	x	<u>40</u>		=	<u>0</u>		Multiple Dependent					<u>135</u>		=			Large Entity Fee Code (\$)	Small Entity Fee Code (\$)	Fee Description	Fee Paid	103 18	203 9	Claims in excess of 20		102 80	202 40	Independent claims in excess of 3		104 270	204 135	Multiple dependent claim, if not paid		109 80	209 40	** Reissue independent claims over original patent		110 18	210 9	** Reissue claims in excess of 20 and over original patent		SUBTOTAL (2)			(\$ 0)	<p style="text-align: center; font-weight: bold;">FEE CALCULATION (3)</p> <p>SUBTOTAL (3) (\$ 0)</p> <p style="font-size: x-small;">* Reduced by Basic Filing Fee Paid</p>																																		
Large Entity Fee Code (\$)	Small Entity Fee Code (\$)	Fee Description	Fee Paid																																																																																																																						
101 710	201 355	Utility filing fee	355																																																																																																																						
106 320	206 160	Design filing fee																																																																																																																							
107 490	207 245	Plant filing fee																																																																																																																							
108 710	208 355	Reissue filing fee																																																																																																																							
114 150	214 75	Provisional filing fee																																																																																																																							
SUBTOTAL (1)			(\$ 355.00)																																																																																																																						
Total Claims	<u>16</u>	-20** =	<u>0</u>	x	Fee from below	<u>9</u>	=	<u>0</u>	Fee Paid																																																																																																																
Independent Claims	<u>3</u>	-3** =	<u>0</u>	x	<u>40</u>		=	<u>0</u>																																																																																																																	
Multiple Dependent					<u>135</u>		=																																																																																																																		
Large Entity Fee Code (\$)	Small Entity Fee Code (\$)	Fee Description	Fee Paid																																																																																																																						
103 18	203 9	Claims in excess of 20																																																																																																																							
102 80	202 40	Independent claims in excess of 3																																																																																																																							
104 270	204 135	Multiple dependent claim, if not paid																																																																																																																							
109 80	209 40	** Reissue independent claims over original patent																																																																																																																							
110 18	210 9	** Reissue claims in excess of 20 and over original patent																																																																																																																							
SUBTOTAL (2)			(\$ 0)																																																																																																																						

SUBMITTED BY		Complete (if applicable)	
Name (Print/Type)	Frank V. DeRosa	Registration No. (Attorney/Agent)	43,584
Telephone	(516) 357-0091	Signature	
Date	2/2/01		

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Washington, DC 20231.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Assistant Commissioner for Patents
Washington, D.C. 20231

UTILITY APPLICATION FEE TRANSMITTAL

11046 U.S. PTO
09/776267
02/02/01

Sir:

Transmitted herewith for filing is the patent application of

Inventor(s): James J. Fallon, John Buck, Paul F. Pickel,
Stephen J. McEerlain

For: SYSTEMS AND METHODS FOR ACCELERATED LOADING OF
OPERATING SYSTEMS AND APPLICATION PROGRAMS

Enclosed are:

- 52 page(s) of specification
- 1 page(s) of Abstract
- 4 page(s) of claims
- 13 sheets of drawings [] formal [X] informal
- [] _____ page(s) of Declaration and Power of Attorney
- [] An Assignment of the invention to: _____

CERTIFICATION UNDER 37 C.F.R. § 1.10

I hereby certify that this New Application Transmittal and the documents referred to as enclosed therein are being deposited with the United States Postal Service on this date February 2, 2001 in an envelope as "Express Mail Post Office to Addressee" Mail Label Number EL679454191US addressed to: Assistant Commissioner for Patents, Washington, D.C. 20231.

Frank V. DeRosa
(Type or print name of person mailing paper)


(Signature of person mailing paper)

11046 U.S. PTO
02/02/01

02/02/01

[X] This application claims the benefit under 35 U.S.C. §119(e) of U.S. Provisional Application(s) No(s):

APPLICATION NO(S): 60/180,114 FILING DATE February 3, 2000
_____/_____

[] Certified copy of applications

Country Appln. No. Filed

from which priority under Title 35 United States Code, § 119 is claimed

[] is enclosed.

[] will follow.

CALCULATION OF UTILITY APPLICATION FEE

For	Number Filed	Number Extra	Rate	Basic Fee \$710.00
Total Claims*	16	-20 = 0	x \$ 18.00	\$.00
Independent Claims	3	-3 = 0	x \$ 80.00	\$.00
Multiple Dependent Claims	[] yes	Add'l. Fee	\$270.00	\$
	[] no	Add'l. Fee	None	= \$

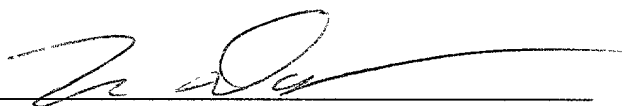
TOTAL \$ 710.00

[X] "Small Entity" Status Claimed Under 37 C.F.R. § 1.27. Reduced fees under 37 C.F.R. § 1.9(f) (50% of total) paid herewith \$355.00.

*Includes all independent and single dependent claims and all claims referred to in multiple claims. See 37 C.F.R. § 1.75(c).

- [] A check in the amount of \$ _____ is enclosed for recording the attached Assignment.
- [X] A check in the amount of \$355.00 to cover the filing fee is attached.
- [] Charge fee to Deposit Account No. 50-0679. Order No. 50-0679. TWO (2) COPIES OF THIS SHEET ARE ENCLOSED.
- [X] Please charge any deficiency as well as any other fee(s) which may become due under 37 C.F.R. § 1.16 and 1.17, at any time during the pendency of this application, or credit any overpayment of such fee(s) to Deposit Account No. 50-0679. Also, in the event any extensions of time for responding are required for the pending application(s), please treat this paper as a petition to extend the time as required and charge Deposit Account No. 50-0679 therefor. TWO (2) COPIES OF THIS SHEET ARE ENCLOSED.

Date: 2/2/01


SIGNATURE OF ATTORNEY
Frank V. DeRosa
Reg. No. 43,584

F. CHAU & ASSOCIATES, LLP
1900 Hempstead Turnpike
Suite 501
East Meadow, New York 11554
Tel. No. (516) 357-0091
Fax. (516) 357-0092
FVD:pg

U.S. Patent Application:

Title: SYSTEMS AND METHODS FOR ACCELERATED LOADING
OF OPERATING SYSTEMS AND APPLICATION PROGRAMS

Inventor(s): James J. Fallon, 11 Wampus Close, Armonk, New York, 10504;
John Buck, 362 Christopher Street, Oceanside, New York, 11572;
Paul F. Pickel, 225 Stewart Avenue, Bethpage, New York, 11714; and
Stephen J. McErlain, 325 East 17th Street, New York, New York 10003.

Filed: February 2, 2001

Assignee: Realtime Data LLC

F. Chau & Associates, LLP
1900 Hempstead Turnpike, Suite 501
East Meadow, NY 11554
Tel: (516) 357-0091
Fax: (516) 357-0092

**SYSTEMS AND METHODS FOR ACCELERATED LOADING OF
OPERATING SYSTEMS AND APPLICATION PROGRAMS**

CROSS-REFERENCE TO RELATED APPLICATION

5 This application is based on a United States provisional application Serial No. 60/180,114, filed on February 3, 2000, which is fully incorporated herein by reference.

BACKGROUND

1. Technical Field

10 The present invention relates generally to systems and methods for providing accelerated loading of operating system and application programs upon system boot or application launch and, more particularly, to data storage controllers employing lossless and/or lossy data compression and decompression to provide accelerated loading of operating systems and application programs.

15 **2. Description of the Related Art**

 Modern computers utilize a hierarchy of memory devices. To achieve maximum performance levels, modern processors utilize onboard memory and on board cache to obtain high bandwidth access to both program and data. Limitations in process technologies currently prohibit placing a sufficient quantity of onboard memory for most applications. Thus, in order to offer sufficient memory for the operating system(s), application programs, and user data, computers often use various forms of popular off-processor high speed memory including static random access memory (SRAM), synchronous dynamic random access memory (SDRAM), synchronous burst static ram (SBSRAM). Due to the prohibitive cost of the high-speed random access memory, 20 coupled with their power volatility, a third lower level of the hierarchy exists for non-

volatile mass storage devices.

Furthermore, mass storage devices offer increased capacity and fairly economical data storage. Mass storage devices (such as a “hard disk”) typically store the operating system of a computer system, as well as applications and data and rapid access to such data is critical to system performance. The data storage and retrieval bandwidth of mass storage devices, however, is typically much less as compared with the bandwidth of other elements of a computing system. Indeed, over the last decade, although computer processor performance has improved by at least a factor of 50, magnetic disk storage performance has only improved by a factor of 5. Consequently, memory storage devices severely limit the performance of consumer, entertainment, office, workstation, servers, and mainframe computers for all disk and memory intensive operations.

The ubiquitous Internet combined with new multimedia applications has put tremendous emphasis on storage volumetric density, storage mass density, storewidth, and power consumption. Specifically, storage density is limited by the number of bits that are encoded in a mass storage device per unit volume. Similarly mass density is defined as storage bits per unit mass. Storewidth is the data rate at which the data may be accessed. There are various ways of categorizing storewidth in terms, several of the more prevalent metrics include sustained continuous storewidth, burst storewidth, and random access storewidth, all typically measured in megabytes/sec. Power consumption is canonically defined in terms of power consumption per bit and may be specified under a number of operating modes including active (while data is being accessed and transmitted) and standby mode. Hence one fairly obvious limitation within the current art is the need for even more volume, mass, and power efficient data storage.

Magnetic disk mass storage devices currently employed in a variety of home, business, and scientific computing applications suffer from significant seek-time access delays along with profound read/write data rate limitations. Currently the fastest available disk drives support only a sustained output data rate in the tens of megabytes per second data rate (MB/sec). This is in stark contrast to the modern Personal Computer's Peripheral Component Interconnect (PCI) Bus's low end 32 bit / 33Mhz input/output capability of 264 MB/sec and the PC's internal local bus capability of 800 MB/sec.

Another problem within the current art is that emergent high performance disk interface standards such as the Small Computer Systems Interface (SCSI-3), Fibre Channel, AT Attachment UltraDMA/66/100, Serial Storage Architecture, and Universal Serial Bus offer only higher data transfer rates through intermediate data buffering in random access memory. These interconnect strategies do not address the fundamental problem that all modern magnetic disk storage devices for the personal computer marketplace are still limited by the same typical physical media restrictions. In practice, faster disk access data rates are only achieved by the high cost solution of simultaneously accessing multiple disk drives with a technique known within the art as data striping and redundant array of independent disks (RAID).

RAID systems often afford the user the benefit of increased data bandwidth for data storage and retrieval. By simultaneously accessing two or more disk drives, data bandwidth may be increased at a maximum rate that is linear and directly proportional to the number of disks employed. Thus another problem with modern data storage systems utilizing RAID systems is that a linear increase in data bandwidth requires a proportional number of added disk storage devices.

Another problem with most modern mass storage devices is their inherent unreliability. Many modern mass storage devices utilize rotating assemblies and other types of electromechanical components that possess failure rates one or more orders of magnitude higher than equivalent solid-state devices. RAID systems employ data
5 redundancy distributed across multiple disks to enhance data storage and retrieval reliability. In the simplest case, data may be explicitly repeated on multiple places on a single disk drive, on multiple places on two or more independent disk drives. More complex techniques are also employed that support various trade-offs between data bandwidth and data reliability.

10 Standard types of RAID systems currently available include RAID Levels 0, 1, and 5. The configuration selected depends on the goals to be achieved. Specifically data reliability, data validation, data storage /retrieval bandwidth, and cost all play a role in defining the appropriate RAID data storage solution. RAID level 0 entails pure data striping across multiple disk drives. This increases data bandwidth at best linearly with
15 the number of disk drives utilized. Data reliability and validation capability are decreased. A failure of a single drive results in a complete loss of all data. Thus another problem with RAID systems is that low cost improved bandwidth requires a significant decrease in reliability.

RAID Level 1 utilizes disk mirroring where data is duplicated on an independent
20 disk subsystem. Validation of data amongst the two independent drives is possible if the data is simultaneously accessed on both disks and subsequently compared. This tends to decrease data bandwidth from even that of a single comparable disk drive. In systems that offer hot swap capability, the failed drive is removed and a replacement drive is

inserted. The data on the failed drive is then copied in the background while the entire system continues to operate in a performance degraded but fully operational mode. Once the data rebuild is complete, normal operation resumes. Hence, another problem with RAID systems is the high cost of increased reliability and associated decrease in performance.

RAID Level 5 employs disk data striping and parity error detection to increase both data bandwidth and reliability simultaneously. A minimum of three disk drives is required for this technique. In the event of a single disk drive failure, that drive may be rebuilt from parity and other data encoded on disk remaining disk drives. In systems that offer hot swap capability, the failed drive is removed and a replacement drive is inserted. The data on the failed drive is then rebuilt in the background while the entire system continues to operate in a performance degraded but fully operational mode. Once the data rebuild is complete, normal operation resumes.

Thus another problem with redundant modern mass storage devices is the degradation of data bandwidth when a storage device fails. Additional problems with bandwidth limitations and reliability similarly occur within the art by all other forms of sequential, pseudo-random, and random access mass storage devices. These and other limitations within the current art are addressed by the present invention.

SUMMARY OF THE INVENTION

The present invention is directed to systems and methods for providing accelerated loading of operating system and application programs upon system boot or application launch and, more particularly, to data storage controllers employing lossless and/or lossy data compression and decompression to provide accelerated loading of

operating systems and application programs.

In one aspect of the present invention, a method for providing accelerated loading of an operating system comprises the steps of: maintaining a list of boot data used for booting a computer system; preloading the boot data upon initialization of the computer system; and servicing requests for boot data from the computer system using the preloaded boot data. The boot data may comprise program code associated with an operating system of the computer system, an application program, and a combination thereof. In a preferred embodiment, the boot data is retrieved from a boot device and stored in a cache memory device.

In another aspect, the method for accelerated loading of an operating system comprises updating the list of boot data during the boot process. The step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list and/or removing from the list any boot data previously stored in the list and not requested by the computer system.

In yet another aspect, the boot data is stored in a compressed format on the boot device and the preloaded boot data is decompressed prior to transmitting the preloaded boot data to the requesting system.

In another aspect, a method for providing accelerated launching of an application program comprises the steps of: maintaining a list of application data associated with an application program; preloading the application data upon launching the application program; and servicing requests for application data from a computer system using the preloaded application data.

In yet another aspect, a boot device controller for providing accelerated loading of

an operating system of a host system comprises: a digital signal processor (DSP); a programmable logic device, wherein the programmable logic device is programmed by the digital signal processor to (i) instantiate a first interface for operatively interfacing the boot device controller to a boot device and to (ii) instantiate a second interface for
5 operatively interfacing the boot device controller to the host system; and a non-volatile memory device, for storing logic code associated with the DSP, the first interface and the second interface, wherein the logic code comprises instructions executable by the DSP for maintaining a list of boot data used for booting the host system, preloading the boot data upon initialization of the host system, and servicing requests for boot data from the host
10 system using the preloaded boot data. The boot device controller further includes a cache memory device for storing the preloaded boot data.

The present invention is realized due to recent improvements in processing speed, inclusive of dedicated analog and digital hardware circuits, central processing units, (and any hybrid combinations thereof), that, coupled with advanced data compression and
15 decompression algorithms are enabling of ultra high bandwidth data compression and decompression methods that enable improved data storage and retrieval bandwidth

These and other aspects, features and advantages, of the present invention will become apparent from the following detailed description of preferred embodiments that is to be read in connection with the accompanying drawings.

20

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of a data storage controller according to one embodiment of the present invention;

Fig. 2 is a block diagram of a data storage controller according to another embodiment of the present invention;

Fig. 3 is a block diagram of a data storage controller according to another embodiment of the present invention;

5 Fig. 4 is a block diagram of a data storage controller according to another embodiment of the present invention;

Fig. 5 is a block diagram of a data storage controller according to another embodiment of the present invention;

10 Figs. 6a and 6b comprise a flow diagram of a method for initializing a data storage controller according to one aspect of the present invention;

Figs. 7a and 7b comprise a flow diagram of a method for providing accelerated loading of an operating system and/or application programs upon system boot, according to one aspect of the present invention;

15 Figs. 8a and 8b comprise a flow diagram of a method for providing accelerated loading of application programs according to one aspect of the present invention;

Fig. 9 is a diagram of an exemplary data compression system that may be employed in a data storage controller according to the present invention; and

Fig. 10 is a diagram of an exemplary data decompression system that may be employed in a data storage controller according to the present invention.

20

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

In the following description, it is to be understood that system elements having equivalent or similar functionality are designated with the same reference numerals in the Figures. It is to be further understood that the present invention may be implemented in
5 various forms of hardware, software, firmware, or a combination thereof. Preferably, the present invention is implemented on a computer platform including hardware such as one or more central processing units (CPU) or digital signal processors (DSP), a random access memory (RAM), and input/output (I/O) interface(s). The computer platform may also include an operating system, microinstruction code, and dedicated processing
10 hardware utilizing combinatorial logic or finite state machines. The various processes and functions described herein may be either part of the hardware, microinstruction code or application programs that are executed via the operating system, or any combination thereof.

It is to be further understood that, because some of the constituent system
15 components described herein are preferably implemented as software modules, the actual system connections shown in the Figures may differ depending upon the manner in that the systems are programmed. It is to be appreciated that special purpose microprocessors, dedicated hardware, or and combination thereof may be employed to implement the present invention. Given the teachings herein, one of ordinary skill in the related art will
20 be able to contemplate these and similar implementations or configurations of the present invention.

I. System Architectures

The present invention is directed to data storage controllers that provide increased

data storage/retrieval rates that are not otherwise achievable using conventional disk controller systems and protocols to store/retrieve data to/from mass storage devices. The concept of “accelerated” data storage and retrieval was introduced in copending U.S. Patent Application Serial No. 09/266,394, filed March 11, 1999, entitled “System and Methods For Accelerated Data Storage and Retrieval” and copending U.S. Patent Application Serial No. 09/481,243, filed January 11, 2000, entitled “System and Methods For Accelerated Data Storage and Retrieval,” both of which are commonly assigned and incorporated herein by reference. In general, as described in the above-incorporated applications, “accelerated” data storage comprises receiving a digital data stream at a data transmission rate which is greater than the data storage rate of a target storage device, compressing the input stream at a compression rate that increases the effective data storage rate of the target storage device and storing the compressed data in the target storage device. For instance, assume that a mass storage device (such as a hard disk) has a data storage rate of 20 megabytes per second. If a storage controller for the mass storage device is capable of compressing an input data stream with an average compression rate of 3:1, then data can be stored in the mass storage device at a rate of 60 megabytes per second, thereby effectively increasing the storage bandwidth (“storewidth”) of the mass storage device by a factor of three. Similarly, accelerated data retrieval comprises retrieving a compressed digital data stream from a target storage device at the rate equal to, e.g., the data access rate of the target storage device and then decompressing the compressed data at a rate that increases the effective data access rate of the target storage device. Advantageously, accelerated data storage/retrieval mitigates the traditional bottleneck associated with, e.g., local and network disk accesses.

Referring now to Fig. 1, a high-level block diagram illustrates a data storage controller 10 according to one embodiment of the present invention. The data storage controller 10 comprises a data compression engine 12 for compressing/decompressing data (preferably in real-time or psuedo real-time) stored/retrieved from a hard disk 11(or
5 any other type of mass storage device) to provide accelerated data storage/retrieval. The DCE 12 preferably employs the data compression/decompression techniques disclosed in U.S. Serial No. 09/210,491 entitled "Content Independent Data Compression Method and System," filed on December 11, 1998, which is commonly assigned and which is fully incorporated herein by reference. It is to be appreciated that the compression and
10 decompression systems and methods disclosed in U.S. Serial No. 09/210,491 are suitable for compressing and decompressing data at rates, which provide accelerated data storage and retrieval. A detailed discussion of a preferred "content independent" data compression process will be provided below.

The data storage controller 10 further comprises a cache 13, a disk interface (or
15 disk controller) 14 and a bus interface 15. The storage controller 10 is operatively connected to the hard disk 12 via the disk controller 14 and operatively connected to an expansion bus (or main bus) 16 of a computer system via the bus interface 15. The disk interface 14 may employ a known disk interface standard such as UltraDMA, SCSI, Serial Storage Architecture, FibreChannel or any other interface that provides suitable
20 disk access data rates. In addition, the storage controller 10 preferably utilizes the American National Standard for Information Systems (ANSI) AT Attachment Interface (ATA/ATAPI-4) to connect the data storage controller 10 to the hard disk 12. As is known in the art, this standard defines the connectors and cables for the physical

interconnects between the data storage controller and the storage devices, along with the electrical and logical characteristics of the interconnecting signals.

Further, the bus interface 15 may employ a known standard such as the PCI (Peripheral Component Interconnect) bus interface for interfacing with a computer system. The use of industry standard interfaces and protocols is preferable, as it allows the storage controller 10 to be backwards compatible and seamlessly integrated with current systems. However in new designs the present invention may be utilize any suitable computer interface or combination thereof.

It is to be understood that although Fig. 1 illustrates a hard disk 12, the storage controller 10 may be employed with any form of memory device including all forms of sequential, pseudo-random, and random access storage devices. Storage devices as known within the current art include all forms of random access memory, magnetic and optical tape, magnetic and optical disks, along with various other forms of solid-state mass storage devices. The current invention applies to all forms and manners of memory devices including, but not limited to, storage devices utilizing magnetic, optical, and chemical techniques, or any combination thereof. In addition, the cache 13 may comprise volatile or non-volatile memory, or any combination thereof. Preferably, the cache 13 is implemented in SDRAM (static dynamic random access memory).

The system of Fig. 1 generally operates as follows. When data is read from disk by the host computer, data flows from the disk 11 through the data storage controller 10 to the host computer. Data is stored in one of several proprietary compression formats on the disk 11 (e.g., "content independent" data compression). Data blocks are pre-specified in length, comprised of single or multiple sectors, and are typically handled in fractional

or whole equivalents of tracks, e.g. ½ track, whole track, multiple tracks, etc. To read disk data, a DMA transfer is setup from the disk interface 14 to the onboard cache memory 13. The disk interface 14 comprises integral DMA control to allow transfer of data from the disk 11 directly to the onboard cache 13 without intervention by the DCE 12. It should be noted that the DCE 12 acts as a system level controller and sets-up specific registers within both the disk interface 14 and bus interface 15 to facilitate DMA transfers to and from the cache memory 13. To initiate a transfer from the disk 11 to the cache 13, the DMA transfer is setup via specifying the appropriate command (read disk), the source address (disk logical block number), amount of data to be transferred (number of disk logical blocks), and destination address within the onboard cache memory 13. Then, a disk data interrupt signal (“DISKINT#”) is cleared (if previously set and not cleared) and the command is initiated by writing to the appropriate address space. Once data has been read from disk 11 and placed into onboard cache memory 13, the DISKINT# interrupt is asserted notifying the DCE 12 that requested data is now available in the cache memory 13. Data is then read by the DMA controller within the DCE 12 and placed into local memory for subsequent decompression. The decompressed data is then DMA transferred from the local memory of the DCE 12 back to the cache memory 13. Finally, data is DMA transferred via the bus interface controller 15 from the cache memory 13 to the bus 16. It is to be understood that in the read mode, the data storage controller acts as a bus master. A bus DMA transfer is then setup via specifying the appropriate command (write to host computer), the source address within the cache memory 13, the quantity of data words to be transferred (transfers are preferably in 4 byte increments), and the destination address on the host computer. When a bus 16 read or

write transaction has completed, the appropriate interrupt signals (respectively referred to as PCIRDINT# and PCIWRINT#) are asserted to the DCE 12. Either of these interrupts are cleared by a corresponding interrupt service routines through a read or write to the appropriate address of the DCE 12.

5 Similarly, when data is written to the disk 11 from the host computer, data flows from the host computer through the data storage controller 10 and onto disk 11. Data is normally received from the host computer in uncompressed (raw) format and is compressed by the DCE 12 and stored on the disk 11. Data blocks from the host are pre-specified in length and are typically handled in blocks that are a fixed multiplier higher
10 than fractional or whole equivalents of tracks, e.g. ½ track, whole track, multiple tracks, etc. This multiplier is preferably derived from the expected average compression ratio that is selected when the disk is formatted with the virtual file management system. To read host computer data, a bus DMA transfer is setup from the host bus 16 to the onboard cache memory 13. The bus interface controller 15 comprises integral DMA control that
15 allows large block transfers from the host computer directly to the onboard cache 13 without intervention by the DCE 12. The bus interface controller 15 acts as a host computer Bus Master when executing such transfer. Once data has been read from the host and placed into onboard cache memory 13, the data is read by the onboard DMA controller (residing on the DCE 12) and placed into local memory for subsequent
20 compression. The compressed data is then DMA transferred from the local memory of the DCE 12 back to the cache memory 13. Finally, data is DMA transferred via the disk controller 14 from the cache 13 to the disk 11.

As discussed in greater detail below, upon host computer power-up or external

user reset, the data storage controller 10 initializes the onboard interfaces 14, 5 prior to release of the external host bus 16 from reset. The processor of the host computer then requests initial data from the disk 11 to facilitate the computer's boot-up sequence. The host computer requests disk data over the Bus 16 via a command packet issued from the host computer. Command packets are preferably eight words long (in a preferred embodiment, each word comprises 32 bits). Commands are written from the host computer to the data storage controller 10 with the host computer as the Bus Master and the data storage controller 10 as the slave. The data storage controller 10 includes at least one Base Address Register (BAR) for decoding the address of a command queue of the data storage controller 10. The command queue resides within the cache 13 or within onboard memory of the DCE 12.

When a command is received from the host computer, an interrupt (referred to herein as PCICMDINT#) is generated to the DCE processor. The eight-word command is read by the DCE 12 and placed into the command queue. Because the commands occupy a very small amount of memory, the location of the command queue is at the discretion of software and the associated system level performance considerations. Commands may be moved from the bus interface 16 to the command queue by wither explicit reads and writes by the DCE processor or, as explained below, by utilizing programmed DMA from an Enhanced DMA Controller (EDMA) residing on the DCE 12. This second technique may better facilitate system throughput by allowing the EDMA to automatically load commands while the highly pipelined data compression and decompression processing in the DCE is executed fully undisturbed.

The DCE 12, disk interface 14 and bus interface 15 commonly share the cache 13.

As explained in detail below, the storage controller 10 preferably provides maximum system bandwidth by allowing simultaneous data transfers between the disk 12 and cache 13, the DCE 12 and the cache 13, and the expansion bus 16 and the cache 13. This is realized by employing an integral DMA (direct memory access) protocol that allows the DCE 12, disk interface 14 and bus interface 15 to transfer data without interrupting or interfering with other ongoing processes. In particular, as explained in detail below, an integral bandwidth allocation controller (or arbitrator) is preferably employed to allow the DCE 12, disk controller 14, and bus interface 15 to access the onboard cache with a bandwidth proportional to the overall bandwidth of the respective interface or processing element. The bandwidth arbitration occurs transparently and does not introduce latency in memory accesses. Bandwidth division is preferably performed with a high degree of granularity to minimize the size of requisite onboard buffers to synchronize data from the disk interface 14 and bus interface 15.

It is to be appreciated that the implementation of a storage controller according to the present invention significantly accelerates the performance of a computer system and significantly increases hard disk data storage capacity. For instance, depending on the compression rate, for personal computers running standard Microsoft Windows® based business application software, the storage controller provides: (1) an increase of n:1 in disk storage capacity (for example, assuming a compression ration of 3:1, a 20 gigabyte hard drive effectively becomes a 60 gigabyte hard drive) (2) a significant decrease in the computer boot-up time (turn-on and operating system load) and the time for loading application software and (3) User data storage and retrieval is increased by a factor of n:1.

Referring now to Fig. 2, a block diagram illustrates a data storage controller 20

according to another embodiment of the present invention. More specifically, Fig. 2 illustrates a PCB (printed circuit board) implementation of the data storage controller 10 of Fig. 1. The storage controller 20 comprises a DSP (digital signal processor) 21 (or any other micro-processor device) that implements the DCE 12 of Fig. 1. The storage controller 21 further comprises at least one programmable logic device 22 (or volatile logic device). The programmable logic device 22 preferably implements the logic (program code) for instantiating and driving both the disk interface 14 and the bus interface 15 and for providing full DMA capability for the disk and bus interfaces 14, 15. Further, as explained in detail below, upon host computer power-up and/or assertion of a system-level “reset” (e.g., PCI Bus reset), the DSP 21 initializes and programs the programmable logic device 22 before of the completion of initialization of the host computer. This advantageously allows the data storage controller 20 to be ready to accept and process commands from the host computer (via the bus 16) and retrieve boot data from the disk (assuming the data storage controller 20 is implemented as the boot device and the hard disk stores the boot data (e.g., operating system, etc.)).

The data storage controller 20 further comprises a plurality of memory devices including a RAM (random access memory) device 23 and a ROM (read only memory) device 24 (or FLASH memory or other types of non-volatile memory). The RAM device 23 is utilized as on-board cache and is preferably implemented as SDRAM (preferably, 32 megabytes minimum). The ROM device 24 is utilized for non-volatile storage of logic code associated with the DSP 21 and configuration data used by the DSP 21 to program the programmable logic device 22. The ROM device 24 preferably comprises a one time (erasable) programmable memory (OTP-EPROM) device.

The DSP 21 is operatively connected to the memory devices 23, 24 and the programmable logic device 22 via a local bus 25. The DSP 21 is also operatively connected to the programmable logic device 22 via an independent control bus 26. The programmable logic device 22 provides data flow control between the DSP 21 and the host computer system attached to the bus 16, as well as data flow control between the DSP 21 and the storage device. A plurality of external I/O ports 27 are included for data transmission and/or loading of one programmable logic devices. Preferably, the disk interface 14 driven by the programmable logic device 22 supports a plurality of hard drives.

The storage controller 20 further comprises computer reset and power up circuitry 28 (or “boot configuration circuit”) for controlling initialization (either cold or warm boots) of the host computer system and storage controller 20. A preferred boot configuration circuit and preferred computer initialization systems and protocols are described in U.S. Patent Application Serial No. _____ (Attorney Docket No. 8011-10), filed concurrently herewith (Express Mail Label No. EL679454245US), which is commonly assigned and incorporated herein by reference. Preferably, the boot configuration circuit 28 is employed for controlling the initializing and programming the programmable logic device 22 during configuration of the host computer system (i.e., while the CPU of the host is held in reset). The boot configuration circuit 28 ensures that the programmable logic device 22 (and possibly other volatile or partially volatile logic devices) is initialized and programmed before the bus 16 (such as a PCI bus) is fully reset.

In particular, when power is first applied to the boot configuration circuit 28, the

boot configuration circuit 28 generates a control signal to reset the local system (e.g., storage controller 20) devices such as a DSP, memory, and I/O interfaces. Once the local system is powered-up and reset, the controlling device (such as the DSP 21) will then proceed to automatically determine the system environment and configure the local system to work within that environment. By way of example, the DSP 21 of the disk storage controller 20 would sense that the data storage controller 20 is on a PCI computer bus (expansion bus) and has attached to it a hard disk on an IDE interface. The DSP 21 would then load the appropriate PCI and IDE interfaces into the programmable logic device 22 prior to completion of the host system reset. It is to be appreciated that this can be done for all computer busses and boot device interfaces including: PCI, NuBus, ISA, Fiber Channel, SCSI, Ethernet, DSL, ADSL, IDE, DMA, Ultra DMA, and SONET. Once the programmable logic device 22 is configured for its environment, the boot device controller is reset and ready to accept commands over the computer/expansion bus 16. Details of the boot process using a boot device comprising a programmable logic device will be provided below.

It is to be understood that the data storage controller 20 may be utilized as a controller for transmitting data (compressed or uncompressed) to and from remote locations over the DSP I/O ports 27 or system bus 16, for example. Indeed, the I/O ports 27 of the DSP 21 may be used for transmitting data (compressed or uncompressed) that is either retrieved from the disk 11 or received from the host system via the bus 16, to remote locations for processing and/or storage. Indeed, the I/O ports may be operatively connected to other data storage controllers or to a network communication channels. Likewise, the data storage controller 20 may receive data (compressed or uncompressed)

over the I/O ports 27 of the DSP 21 from remote systems that are connected to the I/O ports 27 of the DSP, for local processing by the data storage controller 20. For instance, a remote system may remotely access the data storage controller (via the I/O ports of the DSP or system bus 16) to utilize the data compression, in which case the data storage controller would transmit the compressed data back to the system that requested compression.

The DSP 21 may comprise any suitable commercially available DSP or processor. Preferably, the data storage controller 20 utilizes a DSP from Texas Instruments' 320 series, C62x family, of DSPs (such as TMS320C6211GFN-150), although any other DSP or processor comprising a similar architecture and providing similar functionalities may be employed. The preferred DSP is capable of up to 1.2 billion instructions per second. Additional features of the preferred DSP include a highly parallel eight processor single cycle instruction execution, onboard 4K byte L1P Program Cache, 4K L1D Data Cache, and 64K byte Unified L2 Program/Data Cache. The preferred DSP further comprises a 32 bit External Memory Interface (EMIF) that provides for a glueless interface to the RAM 23 and the non-volatile memory 24 (ROM). The DSP further comprises two multi-channel buffered serial ports (McBSPs) and two 32 bit general purpose timers. Preferably, the storage controller disables the I/O capability of these devices and utilizes the I/O ports of the DSP as general purpose I/O for both programming the programmable logic device 22 using a strobed eight bit interface and signaling via a Light Emitting Diode (LED). Ancillary DSP features include a 16 bit Host Port Interface and full JTAG emulation capability for development support.

The programmable logic device 22 may comprise any form of volatile or non-

volatile memory. Preferably, the programmable logic device 22 comprises a dynamically reprogrammable FPGA (field programmable gate array) such as the commercially available Xilinx Spartan Series XCS40XL-PQ240-5 FPGA. As discussed in detail herein, the FPGA instantiates and drives the disk and bus interfaces 14, 15.

5 The non-volatile memory device 24 preferably comprises a 128 Kbyte M27W101-80K one time (erasable) programmable read only memory, although other suitable non-volatile storage devices may be employed. The non-volatile memory device 24 is decoded at a designated memory space in the DSP 21. The non-volatile memory device 24 stores the logic for the DSP 21 and configuration data for the programmable logic
10 device 22. More specifically, in a preferred embodiment, the lower 80 Kbytes of the non-volatile memory device 24 are utilized for storing DSP program code, wherein the first 1k bytes are utilized for the DSP's boot loader. Upon reset of the DSP 21 (via boot configuration circuit 28), the first 1K of memory of the non-volatile memory device 24 is copied into an internal RAM of the DSP 21 by e.g., the DSP's Enhanced DMA Controller
15 (EDMA). Although the boot process begins when the CPU of the host system is released from external reset, the transfer of the boot code into the DSP and the DSP's initialization of the programmable logic device actually occurs while the CPU of the host system is held in reset. After completion of the 1K block transfer, the DSP executes the boot loader code and continues thereafter with executing the remainder of the code in non-volatile
20 memory device to program the programmable logic device 22.

More specifically, in a preferred embodiment, the upper 48K bytes of the non-volatile memory device 24 are utilized for storing configuration data associated with the programmable logic device 22. If the data storage controller 20 is employed as the

primary boot storage device for the host computer, the logic for instantiating and driving the disk and bus interfaces 14, 15 should be stored on the data storage controller 20 (although such code may be stored in remotely accessible memory locations) and loaded prior to release of the host system bus 16 from “reset”. For instance, revision 2.2 of the PCI Local Bus specification calls for a typical delay of 100msec from power-stable before release of PCI Reset. In practice this delay is currently 200msec although this varies amongst computer manufacturers. A detailed discussion of the power-on sequencing and boot operation of the data storage controller 20 will be provided below.

Fig. 3 illustrates another embodiment of a data storage controller 30 wherein the data storage controller 35 is embedded within the motherboard of the host computer system. This architecture provides the same functionality as the system of Fig. 2, and also adds the cost advantage of being embedded on the host motherboard. The system comprises additional RAM and ROM memory devices 23a, 24a, operatively connected to the DSP 21 via a local bus 25a.

Fig. 4 illustrates another embodiment of a data storage controller. The data storage controller 40 comprises a PCB implementation that is capable of supporting RAID levels 0,1 and 5. This architecture is similar to those of Fig. 1 and 2, except that a plurality of programmable logic devices 22, 22a are utilized. The programmable logic device 22 is dedicated to controlling the bus interface 15. The programmable logic device 22a is dedicated to controlling a plurality of disk interfaces 14, preferably three interfaces. Each disk interface 14 can connect up to two drives. The DSP in conjunction with the programmable logic device 22a can operate at RAID level 0, 1 or 5. At RAID level 0, which is disk striping, two interfaces are required. This is also true for RAID

level 1, which is disk mirroring. At RAID level 5, all three interfaces are required.

Fig. 5 illustrates another embodiment of a data storage controller according to the present invention. The data storage controller 45 provides the same functionality as that of Figure 4, and has the cost advantage of being embedded within the computer system motherboard.

II. Initializing A Programmable Logic Device

As discussed above with reference to Fig. 2, for example, the data storage controller 20 preferably employs an onboard Texas Instruments TMS320C6211 Digital Signal Processor (DSP) to program the onboard Xilinx Spartan Series XCS40XL FPGA upon power-up or system level PCI reset. The onboard boot configuration circuit 28 ensures that from system power-up and/or the assertion of a bus reset (e.g., PCI reset), the DSP 21 is allotted a predetermined amount of time (preferably a minimum of 10msec) to boot the DSP 21 and load the programmable logic device 22. Because of a potential race condition between either the host computer power-up or assertion of PCI Bus reset and configuration of the programmable logic device 20 (which is used for controlling the boot device and accepting PCI Commands), an “Express Mode” programming mode for configuring the SpartanXL family XCS40XL device is preferably employed. The XCS40XL is factory set to byte-wide Express-Mode programming by setting both the M1/M0 bits of the XCS40XL to 0x0. Further, to accommodate express mode programming of the programmable logic device 22, the DSP 21 is programmed to utilize its serial ports reconfigured as general purpose I/O. However, after the logic device 22 is programmed, the DSP 21 may then reconfigure its serial ports for use with other devices. Advantageously, using the same DSP ports for multiple purposes affords greater

flexibility while minimizing hardware resources and thus reducing product cost.

The volatile nature of the logic device 22 effectively affords the ability to have an unlimited number of hardware interfaces. Any number of programs for execution by the programmable logic device 22 can be kept in an accessible memory location (EPROM, hard disk, or other storage device). Each program can contain new disk interfaces, interface modes or subsets thereof. When necessary, the DSP 21 can clear the interface currently residing in the logic device 22 and reprogram it with a new interface. This feature allows the data storage controller 20 to have compatibility with a large number of interfaces while minimizing hardware resources and thus reducing product cost.

A preferred protocol for programming the programmable logic device can be summarized in the following steps: (1) Clearing the configuration memory; (2) Initialization; (3) Configuration; and (4) Start-Up. When either of three events occur: the host computer is first powered-up or a power failure and subsequent recovery occurs (cold boot), or a front panel computer reset is initiated (warm boot), the host computer asserts RST# (reset) on the PCI Bus. As noted above, the data storage controller 20 preferably comprises a boot configuration circuit 28 that senses initial host computer power turn-on and/or assertion of a PCI Bus Reset ("PCI RST#"). It is important to note that assuming the data storage controller 20 is utilized in the computer boot-up sequence, it should be available exactly 5 clock cycles after the PCI RST# is deasserted, as per PCI Bus Specification Revision 2.2. While exact timings vary from computer to computer, the typical PCI bus reset is asserted for approximately 200msec from initial power turn-on.

In general, PCI RST# is asserted as soon as the computer's power exceeds a nominal threshold of about 1 volt (although this varies) and remains asserted for 200msec

thereafter. Power failure detection of the 5volt or 3.3 volt bus typically resets the entire computer as if it is an initial power-up event (i.e., cold boot). Front panel resets (warm boots) are more troublesome and are derived from a debounced push-button switch input. Typical front panel reset times are a minimum of 20msec, although again the only
5 governing specification limit is 1msec reset pulse width.

As discussed in detail below, it may not be necessary to reload the programmable logic device 22 each time the DSP is reset. The boot configuration circuit 20 preferably comprises a state machine output signal that is readable by the DSP 21 to ascertain the type of boot process requested. For example, with a front-panel reset (warm boot), the
10 power remains stable on the PCI Bus, thus the programmable logic device 22 should not require reloading.

Referring now to Fig. 6, a flow diagram illustrates a method for initializing the programmable logic device 22 according to one aspect of the invention. In the following discussion, it is assumed that the programmable logic device 22 is always reloaded,
15 regardless of the type of boot process. Initially, in Fig. 6a, the DSP 21 is reset by asserting a DSP reset signal (step 50). Preferably, the DSP reset signal is generated by the boot circuit configuration circuit 28 (as described in the above-incorporated U.S. Serial No. _____ (Attorney Docket No. 8011-10). While the DSP reset signal is asserted (e.g., active low), the DSP is held in reset and is initialized to a prescribed state.

20 Upon deassertion of the DSP Reset signal, the logic code for the DSP (referred to as the "boot loader") is copied from the non-volatile logic device 24 into memory residing in the DSP 21 (step 51). This allows the DSP to execute the initialization of the programmable logic device 22. In a preferred embodiment, the lower 1K bytes of EPROM memory is

copied to the first 1k bytes of DSP's low memory (0x0000 0000 through 0x0000 03FF).

As noted above, the memory mapping of the DSP 21 maps the CE1 memory space

located at 0x9000 0000 through 0x9001 FFFF with the OTP EPROM. In a preferred

embodiment using the Texas Instrument DSP TMS320c6211GFN-150, this ROM boot

5 process is executed by the EDMA controller of the DSP. It is to be understood, however, that the EDMA controller may be instantiated in the programmable logic device (Xilinx), or shared between the DSP and programmable logic device.

After the logic is loaded in the DSP 21, the DSP 21 begins execution out of the

lower 1K bytes of memory (step 52). In a preferred embodiment, the DSP 21 initializes

10 with at least the functionality to read EPROM Memory (CE1) space. Then, as described

above, the DSP preferably configures its serial ports as general purpose I/O (step 53).

Next, the DSP 21 will initialize the programmable logic device 22 using one or more

suitable control signals. (step 54). After initialization, the DSP 21 begins reading the

configuration data of the programmable logic device 22 from the non-volatile memory 24

15 (step 55). This process begins with clearing a Data Byte Counter and then reading the

first data byte beginning at a prespecified memory location in the non-volatile memory 24

(step 56). Then, the first output byte is loaded into the DSP's I/O locations with LSB at

D0 and MSB at D7 (step 57). Before the first byte is loaded to the logic device 22, a

prespecified time delay (e.g., 5usec) is provided to ensure that the logic device 22 has

20 been initialized (step 58). In particular, this time delay should be of a duration at least

equal to the internal setup time of the programmable logic device 22 from completion of

initialization. Once this time delay has expired, the first data byte in the I/O bus 26 of the

DSP 21 is latched into the programmable logic device 22 (step 59).

Next, a determination is made as to whether the Data Byte Counter is less than a prespecified value (step 60). If the Data Byte Counter is less than the prespecified value (affirmative determination in step 60), the next successive data byte for the programmable logic device 22 is read from the non-volatile memory 24 (step 61) and the Data Byte Counter is incremented (step 62).

Next, the read data byte is loaded into the I/O of the DSP (step 63). A time delay of, e.g., 20 nsec is allowed to expire before the data byte is latched to the programmable logic device to ensure that a minimum data set-up time to the programmable logic device 21 is observed (step 64) and the process is repeated (return to step 60). It is to be appreciated that steps 60-64 may be performed while the current data byte is being latched to the programmable logic device. This provides “pipeline” programming of the logic device 22 and minimizes programming duration.

When the Data Byte Counter is not less than the prespecified count value (negative determination in step 60), as shown in Fig. 6b, the last data byte is read from the non-volatile memory and latched to the programmable logic device 22, and the DSP 21 will then poll a control signal generated by the programmable logic device 22 to ensure that the programming of the logic device 22 is successful (step 65). If programming is complete (affirmative determination in step 66), the process continues with the remainder of the data storage controller initialization (step 67). Otherwise, a timeout occurs (step 68) and upon expiration of the timeout, an error signal is provided and the programming process is repeated (step 69)..

III. Data Storage and Retrieval Protocols

A detailed discussion of operational modes of a data storage controller will now

be provided with reference to the embodiment of Fig 2 (although it is to be understood that the following discussion is applicable to all the above-described embodiments). The data storage controller 20 utilizes a plurality of commands to implement the data storage, retrieval, and disk maintenance functions described herein. Each command preferably

5 comprises eight thirty-two bit data words stored and transmitted in little endian format. The commands include: Read Disk Data; Write Disk Data; and Copy Disk Data, for example. For example, a preferred format for the “Read Disk Data” command is:

31	16	15	8	7	0		
Command Packet Number 0000h to FFFFh			Command Type 00h		Command Parameters (00h)		00h
Starting Block Address (Least Significant Word)							04h
Starting Block Address (Most Significant Word)							08h
Number of Blocks (Least Significant Word)							0Ch
Number of Blocks (Most Significant Word)							10h
Destination Address (Least Significant Word)							14h
Destination Address (Most Significant Word)							18h
Checksum				Reserved			1Ch

The host computer commands the data storage controller 20 over the PCI Bus 16,

10 for example. Upon computer power-up or reset, the host computer issues a PCI Bus Reset with a minimum pulse width of 100msec (in accordance with PCI Bus Specification Revision 2.2). Upon completion of the PCI Bus reset, the data storage controller 20 is fully initialized and waiting for completion of the PCI configuration cycle. Upon completion of the PCI configuration cycles, the data storage controller will

15 wait in an idle state for the first disk command.

During operation, the host operating system may issue a command to the data storage controller 20 to store, retrieve, or copy specific logical data blocks. Each command is transmitted over the PCI Bus 16 at the Address assigned to the Base Address Register (BAR) of the data storage controller 20.

5 The commands issued by the host system to the data storage controller and the data transmitted to and from the data storage controller are preferably communicated via a 32 bit, 33MHz, PCI Data Bus. As noted above, the PCI Interface is preferably housed within the onboard Xilinx Spartan XCS40XL-5 40,000 field programmable gate array which instantiates a PCI 32, 32 Bit, 33MHz PCI Bus Interface (as per PCI Bus Revision
10 2.2).

The PCI Bus interface operates in Slave Mode when receiving commands and as a Bus Master when reading or writing data. The source and destination for all data is specified within each command packet. When setting up data transfers, the Enhanced Direct Memory Access (EDMA) Controller of the DSP (or the Xilinx) utilizes two
15 Control Registers, a 16 Word Data Write to PCI Bus FIFO, a 16 Word Data Read From PCI Bus FIFO, and a PCI Data Interrupt (PCIDATINT). The 32 Bit PCI Address Register holds either the starting Source Address for data storage controller Disk Writes where data is read from the PCI Bus, or the starting Destination Address for data storage controller Disk Reads where data is written to the PCI Bus. The second control register is
20 a PCI Count Register that specifies the direction of the data transfer along with the number of 32 bit Data words to be written to or from the PCI bus.

Data is written to the PCI Bus from the DSP via a 16 Word PCI Data Write FIFO located within a prespecified address range. Data writes from the DSP to anywhere

within the address range place that data word in the next available location within the FIFO. Data is read from the PCI Bus to the DSP via a 16 Word PCI Data Read FIFO located within a prespecified address range and data read by the DSP from anywhere within this address range provides the next data word from the FIFO.

5 After completion of the Xilinx initialization by the DSP and subsequent negation of the PCI Bus Reset signal (RST#) by the host computer's PCI Bridge, the data storage controller is ready to accept commands from the host computer via the PCI Bus. When accepting commands it should be noted that the data storage controller is a PCI Target (Slave) Device. Commands are preferably fixed in length at exactly 8 (thirty-two bit)

10 words long. Commands are written from the host computer to the data storage controller via the PCI Bus utilizing the data storage controller's Base Address Register 0 (BAR0). The PCI Bus Reset initially sets the Command FIFO's Counter to zero and also signals the Xilinx's PCI Bus State Controller that the Command FIFO is empty and enable to accept a command.

15 Whenever a data write occurs within the valid data range of BAR0, the data word is accepted from PCI Bus and placed in the next available memory position within the Command FIFO. When the last of the 8 thirty-two bit data words is accepted by the PCI Bus (thus completing the command, i.e. last word for the command FIFO to be full), the PCI Bus State Controller is automatically set to Target Abort (within same PCI

20 Transaction) or Disconnect Without Data for all subsequent PCI transactions that try to writes to BAR0. This automatic setting is the responsibility of the Xilinx PCI Data Interface.

The PCI Command FIFO State Controller then asserts the Command Available

Interrupt to the DSP. The DSP services the Command Available Interrupt by reading the command data from a prespecified address range. It should be noted that the command FIFO is read sequentially from any data access that reads data within such address range. It is the responsibility of the DSP to understand that the data is read sequentially from any order of accesses within the data range and should thus be stored accordingly.

Upon completion of the Command Available Interrupt Service Routine the DSP executes a memory read or write to desired location within the PCI Control Register Space mapped into the DSP's CE3 (Xilinx) memory space. This resets the Command FIFO Counter back to zero. Next, the DSP executes a memory read or write to location in the DSP Memory Space that clears the Command Available Interrupt. Nested interrupts are not possible since the PCI Bus State Machine is not yet able to accept any Command Data at BAR0. Once the Command Available Interrupt routine has cleared the interrupt and exited, the DSP may then enable the PCI State Machine to accept a new command by reading or writing to PCI Command Enable location within the PCI Command FIFO Control Register Space.

A preferred architecture has been selected to enable the data storage controller to operate on one command at a time or to accept multiple prioritized commands in future implementations. Specifically, the decoupling of the Command Available Interrupt Service Routine from the PCI State Machine that accepts Commands at BAR0 enables the DSP's "operating system kernel" to accept additional commands at any time by software command. In single command operation, a command is accepted, the Command Available Interrupt Cleared, and the Command executed by the data storage controller in PCI Master Mode prior to the enabling of the PCI State machine to accept new

commands.

In a prioritized multi-command implementation, the “operating system kernel” may elect to immediately accept new commands or defer the acceptance of new commands based upon any software implemented decision criteria. In one embodiment, the O/S code might only allow a pre-specified number of commands to be queued. In another embodiment, commands might only be accepted during processor idle time or when the DSP is not executing time critical (i.e. highly pipelined) compress/decompress routines. In yet another embodiment, various processes are enabled based upon a pre-emptive prioritized based scheduling system.

As previously stated, the data storage controller retrieves commands from the input command FIFO in 8 thirty-two bit word packets. Prior to command interpretation and execution, a command’s checksum value is computed to verify the integrity of the data command and associated parameters. If the checksum fails, the host computer is notified of the command packet that failed utilizing the Command Protocol Error Handler. Once the checksum is verified the command type and associated parameters are utilized as an offset into the command “pointer” table or may other suitable command/data structure that transfers control to the appropriate command execution routine.

Commands are executed by the data storage controller with the data storage controller acting as a PCI Master. This is in direct contrast to command acceptance where the data storage controller acts as a PCI Slave. When acting as a PCI Bus Master, the data storage controller reads or writes data to the PCI Bus utilizing a separate PCI Bus Data FIFO (distinct & apart from the Command FIFO). The PCI Data FIFO is 64 (thirty-two bit) words deep and may be utilized for either data reads or data writes from the DSP

to the PCI Bus, but not both simultaneously.

For data to be written from the data storage controller to the Host Computer, the DSP must first write the output data to the PCI Bus Data FIFO. The Data FIFO is commanded to PCI Bus Data Write Mode by writing to a desired location within the Xilinx (CE3) PCI Control Register Space. Upon PCI Bus Reset the default state for the PCI Data FIFO is write mode and the PCI Data FIFO Available Interrupt is cleared. The PCI Data FIFO Available Interrupt should also be software cleared by writing to a prespecified location. Preferably, the first task for the data storage controller is for system boot-up or application code to be downloaded from disk. For reference, PCI Data Read Mode is commanded by writing to location BFF0 0104. The PCI Bus Reset initializes the Data FIFO Pointer to the first data of the 64 data words within the FIFO. However this pointer should always be explicitly initialized by a memory write to location BFF0 0108. This ensures that the first data word written to the FIFO by the DSP performing the data write anywhere in address range B000 0000 to B000 01FF is placed at the beginning of the FIFO. Each subsequent write to any location within this address range then places one thirty-two bit data word into the next available location within the PCI Data FIFO. The FIFO accepts up to 64 thirty-two bit data words although it should be clearly understood that not all data transfers to and from the PCI Bus will consist of a full FIFO. Counting the number of thirty-two bit data words written to the PCI Data FIFO is the responsibility of the DSP Code. It is envisioned that the DSP will, in general, use 64 word DMA data transfers, thus alleviating any additional processor overhead.

When the data has been transferred from the DSP to the PCI Data FIFO, the PCI Bus Controller also needs the address of the PCI Target along with the number of data

words to be transmitted. In the current data storage controller implementation, the PCI Bus Address is thirty-two bits wide, although future PCI bus implementations may utilize multiword addressing and/or significantly larger (64 bit & up) address widths. The single thirty-two bit address word is written by the DSP to memory location `aaaa+0x10` in the
5 PCI Control Register Space.

Finally, the PCI Bus Data Write transaction is initiated by writing the PCI Data FIFO word count to a prespecified memory address. The word count value is always decimal 64 or less (0x3F). When the count register is written the value is automatically transferred to the PCI Controller for executing the PCI Bus Master writes.

10 When the PCI Bus has completed the transfer of all data words within the PCI Data FIFO the PCI Data FIFO Available Interrupt is set. The DSP PCI Data FIFO Available Interrupt handler will then check to see if additional data is waiting or expected to be written to the PCI Data Bus. If additional data is required the interrupt is cleared and the data transfer process repeats. If no additional data is required to be transferred
15 then the interrupt is cleared and the routine must exit to a system state controller. For example, if the command is complete then master mode must be disabled and then slave mode (command mode) enabled – assuming a single command by command execution data storage controller.

For data to be read by the data storage controller from the Host Computer, the
20 DSP must command the PCI Bus with the address and quantity of data to be received.

The PCI Data FIFO is commanded to PCI Bus Data Read Mode by writing to a desired location within the Xilinx (CE3) PCI Control Register Space. Upon PCI Bus Reset the default state for the PCI Data FIFO is Write Mode and the PCI Data FIFO Full

Interrupt is cleared. The PCI Data FIFO Full Interrupt should also be cleared via software by writing to such location. The PCI Bus Reset also initializes the PCI Data FIFO Pointer to the first data word of the available 64 data words within the FIFO. However this pointer should always be explicitly initialized by a memory write to prespecified location.

5 For data to be read from the PCI Bus by the data storage controller, the Xilinx PCI Bus Controller requires the address of the PCI Target along with the number of data words to be received. In the current data storage controller implementation, the PCI Bus Address is thirty-two bits wide, although future PCI bus implementations may utilize multiword addressing and/or significantly larger (64 bit & up) address widths. The single
10 thirty-two bit address word is written by the DSP to prespecified memory location in the PCI Control Register Space.

Finally, the PCI Bus Data Read transaction is initiated by writing the PCI Data FIFO word count to prespecified memory address. The word count value is always decimal 64 or less (0x3F). When the count register is written the value is automatically
15 transferred to the PCI Controller for executing the PCI Bus Master Read.

When the PCI Bus has received all the requested data words PCI Data FIFO Full Interrupt is set. The DSP PCI Data FIFO Full Interrupt handler will then check to see if additional data is waiting or expected to be read from the PCI Data Bus. If additional data is required the interrupt is cleared and the data receipt process repeats. If no
20 additional data is required to be transferred, then the interrupt is cleared and the routine exits to a system state controller. For example, if the command is complete then master mode must be disabled and then slave mode (command mode) enabled – assuming a single command by command execution data storage controller.

It is clearly understood that there are other techniques for handling the PCI Data transfers. The current methodology has been selected to minimize the complexity and resource utilization of the Xilinx Gate Array. It should also be understood that the utilization of asynchronous memory reads and writes to initialize system states and synchronize events at a software level aids in both hardware and system level debug at the expense of increase software overhead. Subsequent embodiments of the gate array may automate resource intensive tasks if system level performance mandates.

IV. Memory Bandwidth Allocation

The onboard cache of the data storage controller is shared by the DSP, Disk Interface, and PCI Bus. The best case, maximum bandwidth for the SDRAM memory is 70 megawords per second, or equivalently, 280 megabytes per second. The 32 bit PCI Bus interface has a best case bandwidth of 132 megabytes per second, or equivalently 33 megawords per second. In current practice, this bandwidth is only achieved in short bursts. The granularity of PCI data bursts to/from the data storage controller is governed by the PCI Bus interface data buffer depth of sixteen words (64 bytes). The time division multiplexing nature of the current PCI Data Transfer Buffering methodology cuts the sustained PCI bandwidth down to 66 megabytes/second.

Data is transferred across the ultraDMA disk interface at a maximum burst rate of 66 megabytes/second. It should be noted that the burst rate is only achieved with disks that contain onboard cache memory. Currently this is becoming more and more popular within the industry. However assuming a disk cache miss, the maximum transfer rates from current disk drives is approximately six megabytes per second. Allotting for

technology improvements over time, the data storage controller has been designed for a maximum sustained disk data rate of 20 megabytes second (5 megawords/second). A design challenge is created by the need for continuous access to the SDRAM memory. Disks are physical devices and it is necessary to continuously read data from disk and place it into memory, otherwise the disk will incur a full rotational latency prior to continuing the read transaction. The maximum SDRAM access latency that can be incurred is the depth of the each of the two disk FIFO s or sixteen data. Assuming the FIFO is sixteen words deep the maximum latency time for emptying the other disk FIFO and restoring it to the disk interface is sixteen words at 5 megawords per second or $(16 \times 3.2\text{usec}) = 1\text{usec}$. Each EMIF clock cycle is 14.2857nsec, thus the maximum latency translates to 224 clock cycles. It should be noted that transfers across the disk interface are 16 bits wide, thus the FPGA is required to translate 32 bit memory transfers to 16 bit disk transfers, and vice-versa.

The DSP services request for its external bus from two requestors, the Enhanced Direct Memory Access (EDMA) Controller and an external shared memory device controller. The DSP can typically utilize the full 280 megabytes of bus bandwidth on an 8k through 64K byte (2k word through 16k word) burst basis. It should be noted that the DSRA does not utilize the SDRAM memory for interim processing storage, and as such only utilizes bandwidth in direct proportion to disk read and write commands.

For a single read from disk transaction data is transferred from and DMA transfer into SDRAM memory. This data is then DMA transferred by the DSP into onboard DSP memory, processed, and re transferred back to SDRAM in decompressed format (3 words

for every one word in). Finally the data is read from SDRAM by the PCI Bus Controller and placed into host computer memory. This equates to eight SDRAM accesses, one write from disk, one read by the DSP, three writes by the DSP and three by the PCI Bus. Disk write transactions similarly require eight SDRAM accesses, three from the PCI, 5 three DSP reads, one DSP write, and one to the disk.

Neglecting overhead for setting up DMA transfers, arbitration latencies, and memory wait states for setting up SDRAM transactions, the maximum DSRA theoretical SDRAM bandwidth limit for disk reads or writes is 280/8 megabytes second or 35 megabytes second. It should be noted that the best case allocation of SDRAM bandwidth 10 would be dynamic dependent upon the data compression and decompression ratios. Future enhancements to the data storage controller will utilize a programmable timeslice system to allocate SDRAM bandwidth, however this first embodiment will utilize a fixed allocation ratio as follows:

If all three requestors require SDRAM simultaneously:

15	PCI Bus Interface	3/8
	DSP Accesses	4/8
	UltraDMA Disk Interface	1/8

If only the PCI Bus and DSP require SDRAM:

	PCI Bus Interface	4/8
20	DSP Accesses	4/8

If only the DSP and Disk require SDRAM:

	DSP Accesses	6/8
--	--------------	-----

UltraDMA Disk Interface 2/8

If only the PCI Bus and Disk require SDRAM:

PCI Bus Interface 6/8

UltraDMA Disk Interface 2/8

5

If only one device requires SDRAM it receives the full SDRAM bandwidth. It should be noted that different ratios may be applied based upon the anticipated or actual compression and/or decompression ratios. For example in the case of all three requestors active the following equation applies. Assume that data storage accelerator achieves a compression ratio A:B for example 3:1. The Numerator and denominators of the various allocations are defined as follows:

10

PCI Bus Interface A/K

DSP Accesses (A+B)/K

UltraDMA Disk Interface B/K

15

Where Further define a sum K equal to the sum of the numerators of the PCI Bus interface fraction, the DSP Access fraction, and the UltraDMA Disk Interfaces, i.e. $K = 2(A+B)$. Similarly:

If only the PCI Bus and DSP require SDRAM:

PCI Bus Interface (A+B)/K

20

DSP Accesses (A+B)/K

If only the DSP and Disk require SDRAM:

DSP Accesses	2A/K
UltraDMA Disk Interface	2B/K

If only the PCI Bus and Disk require SDRAM:

PCI Bus Interface	2A/K
UltraDMA Disk Interface	2B/K

It should be noted that the resultant ratios may all be scaled by a constant in order to most effectively utilize the bandwidths of the internal busses and external interfaces. In addition each ratio can be scale by an adjustment factor based upon the time required to complete individual cycles. For example if PCI Bus interface takes 20% longer than all other cycles, the PCI time slice should be adjusted longer accordingly.

V. Instant Boot Device For Operating System, Application Program and Loading

Typically, with conventional boot device controllers, after reset, the boot device controller will wait for a command over the computer bus (such as PCI). Since the boot device controller will typically be reset prior to bus reset and before the computer bus starts sending commands, this wait period is unproductive time. The initial bus commands inevitably instruct the boot device controller to retrieve data from the boot device (such as a disk) for the operating system. Since most boot devices are relatively slow compared to the speed of most computer busses, a long delay is seen by the computer user. This is evident in the time it takes for a typical computer to boot.

It is to be appreciated that a data storage controller (having an architecture as described herein) may employ a technique of data preloading to decrease the computer system boot time. Upon host system power-up or reset, the data storage controller will

perform a self-diagnostic and program the programmable logic device (as discussed above) prior to completion of the host system reset (e.g., PCI bus reset) so that the logic device can accept PCI Bus commands after system reset. Further, prior to host system reset, the data storage controller can proceed to pre-load the portions of the computer operating system from the boot device (e.g., hard disk) into the on-board cache memory. The data storage controller preloads the needed sectors of data in the order in which they will be needed. Since the same portions of the operating system must be loaded upon each boot process, it is advantageous for the boot device controller to preload such portions and not wait until it is commanded to load the operating system. Preferably, the data storage controller employs a dedicated IO channel of the DSP (with or without data compression) to pre-load computer operating systems and applications.

Once the data is preloaded, when the computer system bus issues its first read commands to the data storage controller seeking operating system data, the data will already be available in the cache memory of the data storage controller. The data storage controller will then be able to instantly start transmitting the data to the system bus. Before transmission to the bus, if the was stored in compressed format on the boot device, the data will be decompressed. The process of preloading required (compressed) portions of the operating system significantly reduces the computer boot process time.

In addition to preloading operating system data, the data storage controller could also preload other data that the user would likely want to use at startup. An example of this would be a frequently used application such as a word processor and any number of document files.

There are several techniques that may be employed in accordance with the present

invention that would allow the data storage controller to know what data to preload from the boot device. One technique utilizes a custom utility program that would allow the user to specify what applications/data should be preloaded.

Another technique (illustrated by the flow diagram of Figs. 7a and 7b) that may be
5 employed comprises an automatic process that requires no input from the user. With this technique, the data storage controller maintain a list comprising the data associated with the first series of data requests received by the data storage controller by the host system after a power-on / reset. In particular, referring to Fig. 7a, during the computer boot process, the data storage controller will receive requests for the boot data (step 70). In
10 response, the data storage controller will retrieve the requested boot data from the boot device (e.g., hard disk) in the local cache memory (step 71). For each requested data block, the data storage controller will record the requested data block number in a list (step 72). The data storage controller will record the data block number of each data block requested by the host computer during the boot process (repeat steps 70-72). When
15 the boot process is complete (affirmative determination in step 73), the data storage controller will store the data list on the boot device (or other storage device) (step 74).

Then, upon each subsequent power-on / reset (affirmative result in step 75), the data storage controller would retrieve and read the stored list (step 76) and proceed to preload the boot data specified on the list (i.e., the data associated with the expected data
20 requests) into the onboard cache memory (step 77). It is to be understood that the depending on the resources of the given system (e.g., memory, etc.), the preloading process may be completed prior to commencement of the boot process, or continued after the boot process begins (in which case booting and preloading are performed

simultaneously).

When the boot process begins (step 78) (i.e., the storage controller is initialized and the system bus reset is deasserted), the data storage controller will receive requests for boot data (step 79). If the host computer issues a request for boot data that is pre-
5 loaded in the local memory of the data storage controller (affirmative result in step 80), the request is immediately serviced using the preloaded boot data (step 81). If the host computer issues a request for boot data that is not preloaded in the local memory of the data storage controller (negative determination in step 80), the controller will retrieve the requested data from the boot device, store the data in the local memory, and then deliver
10 the requested boot data to the computer bus (step 82). In addition, the data storage controller would update the boot data list by recording any changes in the actual data requests as compared to the expected data requests already stored in the list (step 83). Then, upon the next boot sequence, the boot device controller would pre-load that data into the local cache memory along with the other boot data previously on the list.

15 Further, during the boot process, if no request is made by the host computer for a data block that was pre-loaded into the local memory of the data storage controller (affirmative result in step 84), then the boot data list will be updated by removing the non-requested data block from the list (step 85). Thereafter, upon the next boot sequence, the data storage controller will not pre-load that data into local memory.

20 **VI. Quick Launch for Operating System, Application Program, and Loading**

It is to be appreciated that the data storage controller (having an architecture as described herein) may employ a technique of data preloading to decrease the time to load application programs (referred to as “quick launch”). Conventionally, when a user

launches an application, the file system reads the first few blocks of the file off the disk, and then the portion of the loaded software will request via the file system what additional data it needs from the disk. For example, a user may open a spreadsheet program, and the program may be configured to always load a company spreadsheet each
5 time the program is started. In addition, the company spreadsheet may require data from other spreadsheet files.

In accordance with the present invention, the data storage controller may be configured to “remember” what data is typically loaded following the launch of the spreadsheet program, for example. The data storage controller may then proceed to
10 preload the company spreadsheet and all the necessary data in the order in which such data is needed. Once this is accomplished, the data storage controller can service read commands using the preloaded data. Before transmission to the bus, if the preloaded data was stored in compressed format, the data will be decompressed. The process of preloading (compressed) program data significantly reduces the time for launching an
15 application.

Preferably, a custom utility program is employed that would allow the user to specify what applications should be made ready for quick launch.

Figs. 8a and 8b comprise a flow diagram of a quick launch method according to one aspect of the present invention. With this technique, the data storage controller
20 maintains a list comprising the data associated with launching an application. In particular, when an application is first launched, the data storage controller will receive requests for the application data (step 90). In response, the data storage controller will retrieve the requested application data from memory (e.g., hard disk) and store it in the

local cache memory (step 91). The data storage controller will record the data block number of each data block requested by the host computer during the launch process (step 92). When the launch process is complete (affirmative determination in step 93), the data storage controller will store the data list in a designated memory location (step 94).

5 Then, referring to Fig. 8b, upon each subsequent launch of the application (affirmative result in step 95), the data storage controller would retrieve and read the stored list (step 96) and then proceed to preload the application data specified on the list (i.e., the data associated with the expected data requests) into the onboard cache memory (step 97). During the application launch process, the data storage controller will receive
10 requests for application data (step 98). If the host computer issues a request for application data that is pre-loaded in the local memory of the data storage controller (affirmative result in step 99), the request is immediately serviced using the preloaded data (step 100). If the host computer issues a request for application data that is not preloaded in the local memory of the data storage controller (negative result in step 99),
15 the controller will retrieve the requested data from the hard disk memory, store the data in the local memory, and then deliver the requested application data to the computer bus (step 101). In addition, the data storage controller would update the application data list by recording any changes in the actual data requests as compared to the expected data requests already stored in the list (step 102).

20 Further, during the launch process, if no request is made by the host computer for a data block that was pre-loaded into the local memory of the data storage controller (affirmative result in step 103), then the application data list will be updated by removing the non-requested data block from the list (step 104). Thereafter, upon the next launch

sequence for the given application, the data storage controller will not pre-load that data into local memory.

It is to be understood that the quick boot and quick launch methods described above are preferably implemented by a storage controller according to the present invention and may or may not utilize data compression/decompression by the DSP. However, it is to be understood that the quick boot and quick launch methods may be implemented by a separate device, processor, or system, or implemented in software.

VII. Content Independent Data Compression

It is to be understood that any conventional compression/decompression system and method (which comply with the above mentioned constraints) may be employed in the data storage controller for providing accelerated data storage and retrieval in accordance with the present invention. Preferably, the present invention employs the data compression/decompression techniques disclosed in the above-incorporated U.S. Serial No. 09/210,491.

Referring to FIG. 9, a detailed block diagram illustrates an exemplary data compression system 110 that may be employed herein. Details of this data compression system are provided in U.S. Serial No. 09/210,491. In this embodiment, the data compression system 110 accepts data blocks from an input data stream and stores the input data block in an input buffer or cache 115. It is to be understood that the system processes the input data stream in data blocks that may range in size from individual bits through complete files or collections of multiple files. Additionally, the input data block size may be fixed or variable. A counter 120 counts or otherwise enumerates the size of input data block in any convenient units including bits, bytes, words, and double words.

It should be noted that the input buffer 115 and counter 120 are not required elements of the present invention. The input data buffer 115 may be provided for buffering the input data stream in order to output an uncompressed data stream in the event that, as discussed in further detail below, every encoder fails to achieve a level of compression that exceeds
5 an *a priori* specified minimum compression ratio threshold.

Data compression is performed by an encoder module 125 which may comprise a set of encoders E1, E2, E3 ... En. The encoder set E1, E2, E3 ... En may include any number "n" (where n may =1) of those lossless encoding techniques currently well known within the art such as run length, Huffman, Lempel-Ziv Dictionary Compression,
10 arithmetic coding, data compaction, and data null suppression. It is to be understood that the encoding techniques are selected based upon their ability to effectively encode different types of input data. It is to be appreciated that a full complement of encoders are preferably selected to provide a broad coverage of existing and future data types.

The encoder module 125 successively receives as input each of the buffered input
15 data blocks (or unbuffered input data blocks from the counter module 120). Data compression is performed by the encoder module 125 wherein each of the encoders E1 ... En processes a given input data block and outputs a corresponding set of encoded data blocks. It is to be appreciated that the system affords a user the option to enable/disable any one or more of the encoders E1.... En prior to operation. As is understood by those
20 skilled in the art, such feature allows the user to tailor the operation of the data compression system for specific applications. It is to be further appreciated that the encoding process may be performed either in parallel or sequentially. In particular, the encoders E1 through En of encoder module 125 may operate in parallel (i.e.,

simultaneously processing a given input data block by utilizing task multiplexing on a single central processor, via dedicated hardware, by executing on a plurality of processor or dedicated hardware systems, or any combination thereof). In addition, encoders E1 through En may operate sequentially on a given unbuffered or buffered input data block.

5 This process is intended to eliminate the complexity and additional processing overhead associated with multiplexing concurrent encoding techniques on a single central processor and/or dedicated hardware, set of central processors and/or dedicated hardware, or any achievable combination. It is to be further appreciated that encoders of the identical type may be applied in parallel to enhance encoding speed. For instance,
10 encoder E1 may comprise two parallel Huffman encoders for parallel processing of an input data block.

A buffer/counter module 130 is operatively connected to the encoder module 125 for buffering and counting the size of each of the encoded data blocks output from encoder module 125. Specifically, the buffer/counter 130 comprises a plurality of
15 buffer/counters BC1, BC2, BC3 ...BCn, each operatively associated with a corresponding one of the encoders E1...En. A compression ratio module 135, operatively connected to the output buffer/counter 130, determines the compression ratio obtained for each of the enabled encoders E1...En by taking the ratio of the size of the input data block to the size of the output data block stored in the corresponding buffer/counters BC1 ... BCn. In
20 addition, the compression ratio module 135 compares each compression ratio with an *a priori*-specified compression ratio threshold limit to determine if at least one of the encoded data blocks output from the enabled encoders E1...En achieves a compression that exceeds an *a priori*-specified threshold. As is understood by those skilled in the art,

the threshold limit may be specified as any value inclusive of data expansion, no data compression or expansion, or any arbitrarily desired compression limit. A description module 138, operatively coupled to the compression ratio module 135, appends a corresponding compression type descriptor to each encoded data block which is selected for output so as to indicate the type of compression format of the encoded data block. A data compression type descriptor is defined as any recognizable data token or descriptor that indicates which data encoding technique has been applied to the data. It is to be understood that, since encoders of the identical type may be applied in parallel to enhance encoding speed (as discussed above), the data compression type descriptor identifies the corresponding encoding technique applied to the encoded data block, not necessarily the specific encoder. The encoded data block having the greatest compression ratio along with its corresponding data compression type descriptor is then output for subsequent data processing, storage, or transmittal. If there are no encoded data blocks having a compression ratio that exceeds the compression ratio threshold limit, then the original unencoded input data block is selected for output and a null data compression type descriptor is appended thereto. A null data compression type descriptor is defined as any recognizable data token or descriptor that indicates no data encoding has been applied to the input data block. Accordingly, the unencoded input data block with its corresponding null data compression type descriptor is then output for subsequent data processing, storage, or transmittal.

Again, it is to be understood that the embodiment of the data compression engine of Fig. 9 is exemplary of a preferred compression system which may be implemented in the present invention, and that other compression systems and methods known to those

skilled in the art may be employed for providing accelerated data storage in accordance with the teachings herein. Indeed, in another embodiment of the compression system disclosed in the above-incorporated U.S. Serial No. 09/210,491, a timer is included to measure the time elapsed during the encoding process against an *a priori*-specified time limit. When the time limit expires, only the data output from those encoders (in the encoder module 125) that have completed the present encoding cycle are compared to determine the encoded data with the highest compression ratio. The time limit ensures that the real-time or pseudo real-time nature of the data encoding is preserved. In addition, the results from each encoder in the encoder module 125 may be buffered to allow additional encoders to be sequentially applied to the output of the previous encoder, yielding a more optimal lossless data compression ratio. Such techniques are discussed in greater detail in the above-incorporated U.S. Serial No. 09/210,491.

Referring now to FIG. 10, a detailed block diagram illustrates an exemplary decompression system that may be employed herein or accelerated data retrieval as disclosed in the above-incorporated U.S. Serial No. 09/210,491. In this embodiment, the data compression engine 180 retrieves or otherwise accepts compressed data blocks from one or more data storage devices and inputs the data via a data storage interface. It is to be understood that the system processes the input data stream in data blocks that may range in size from individual bits through complete files or collections of multiple files. Additionally, the input data block size may be fixed or variable.

The data decompression engine 180 comprises an input buffer 155 that receives as input an uncompressed or compressed data stream comprising one or more data blocks. The data blocks may range in size from individual bits through complete files or

collections of multiple files. Additionally, the data block size may be fixed or variable. The input data buffer 55 is preferably included (not required) to provide storage of input data for various hardware implementations. A descriptor extraction module 160 receives the buffered (or unbuffered) input data block and then parses, lexically, syntactically, or otherwise analyzes the input data block using methods known by those skilled in the art to extract the data compression type descriptor associated with the data block. The data compression type descriptor may possess values corresponding to null (no encoding applied), a single applied encoding technique, or multiple encoding techniques applied in a specific or random order (in accordance with the data compression system embodiments and methods discussed above).

A decoder module 165 includes one or more decoders D1...Dn for decoding the input data block using a decoder, set of decoders, or a sequential set of decoders corresponding to the extracted compression type descriptor. The decoders D1...Dn may include those lossless encoding techniques currently well known within the art, including: run length, Huffman, Lempel-Ziv Dictionary Compression, arithmetic coding, data compaction, and data null suppression. Decoding techniques are selected based upon their ability to effectively decode the various different types of encoded input data generated by the data compression systems described above or originating from any other desired source.

As with the data compression systems discussed in U.S. Application Serial No. 09/210,491, the decoder module 165 may include multiple decoders of the same type applied in parallel so as to reduce the data decoding time. An output data buffer or cache 170 may be included for buffering the decoded data block output from the decoder

module 165. The output buffer 70 then provides data to the output data stream. It is to be appreciated by those skilled in the art that the data compression system 180 may also include an input data counter and output data counter operatively coupled to the input and output, respectively, of the decoder module 165. In this manner, the compressed and
5 corresponding decompressed data block may be counted to ensure that sufficient decompression is obtained for the input data block.

Again, it is to be understood that the embodiment of the data decompression system 180 of FIG. 10 is exemplary of a preferred decompression system and method which may be implemented in the present invention, and that other data decompression
10 systems and methods known to those skilled in the art may be employed for providing accelerated data retrieval in accordance with the teachings herein.

Although illustrative embodiments have been described herein with reference to the accompanying drawings, it is to be understood that the present invention is not limited to those precise embodiments, and that various other changes and modifications may be
15 affected therein by one skilled in the art without departing from the scope or spirit of the invention. All such changes and modifications are intended to be included within the scope of the invention as defined by the appended claims.

WHAT IS CLAIMED IS:

1. A method for providing accelerated loading of an operating system, comprising the steps of:

maintaining a list of boot data used for booting a computer system;

5 preloading the boot data upon initialization of the computer system; and

servicing requests for boot data from the computer system using the preloaded boot data.

2. The method of claim 1, wherein the boot data comprises program code
10 associated with one of an operating system of the computer system, an application program, and a combination thereof.

3. The method of claim 1, wherein the step of preloading the boot data comprises
15 retrieving boot data from a boot device and storing the retrieved data in a cache memory.

4. The method of claim 3, wherein the method steps are performed by a data
storage controller connected to the boot device.

5. The method of claim 1, further comprising the step of updating the list of boot
20 data during the boot process.

6. The method of claim 5, wherein the step of updating comprises adding to the
list any boot data requested by the computer system not previously stored in the list.

7. The method of claim 5, wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system.

5 8. The method of claim 1, wherein the boot data is compressed and further comprising the step of decompressing the preloaded boot data.

9. The method of claim 1, wherein the method steps are program instructions that are tangibly embodied on a program storage device and readable by a machine to execute
10 the method steps.

10. A method for providing accelerated launching of an application program, comprising the steps of:

maintaining a list of application data associated with an application program;
15 preloading the application data upon launching the application program; and
servicing requests for application data from a computer system using the preloaded application data.

11. The method of claim 1, wherein the application data is compressed and
20 further comprising the step of decompressing the preloaded application data.

12. The method of claim 10, wherein the method steps are program instructions that are tangibly embodied on a program storage device and readable by a machine to

execute the method steps.

13. A boot device controller for providing accelerated loading of an operating system of a host system, the boot device controller comprising:

5 a digital signal processor (DSP);

a programmable logic device, wherein the programmable logic device is programmed by the digital signal processor to (i) instantiate a first interface for operatively interfacing the boot device controller to a boot device and to (ii) instantiate a second interface for operatively interfacing the boot device controller to the host system;

10 and

a non-volatile memory device, for storing logic code associated with the DSP, the first interface and the second interface, wherein the logic code comprises instructions executable by the DSP for maintaining a list of boot data used for booting the host system, preloading the boot data upon initialization of the host system, and servicing requests for boot data from the host system using the preloaded boot data.

14. The boot device controller of claim 13, further comprising a cache memory device for storing the preloaded boot data.

20 15. The boot device controller of claim 13, wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for

ABSTRACT OF THE DISCLOSURE

Systems and methods for providing accelerated loading of operating system and application programs upon system boot or application launch. In one aspect, a method for providing accelerated loading of an operating system comprises the steps of: maintaining
5 a list of boot data used for booting a computer system; preloading the boot data upon initialization of the computer system; and servicing requests for boot data from the computer system using the preloaded boot data. The boot data may comprise program code associated with an operating system of the computer system, an application program, and a combination thereof. In a preferred embodiment, the boot data is
10 retrieved from a boot device and stored in a cache memory device. In another aspect, the method for accelerated loading of an operating system comprises updating the list of boot data during the boot process, wherein updating comprises adding to the list any boot data requested by the computer system not previously stored in the list and/or removing from the list any boot data previously stored in the list and not requested by the computer
15 system. In yet another aspect, the boot data is stored in a compressed format on the boot device and the preloaded boot data is decompressed prior to transmitting the preloaded boot data to the requesting system. In another aspect, a method for providing accelerated launching of an application program comprises the steps of: maintaining a list of application data associated with an application program; preloading the application data
20 upon launching the application program; and servicing requests for application data from a computer system using the preloaded application data.

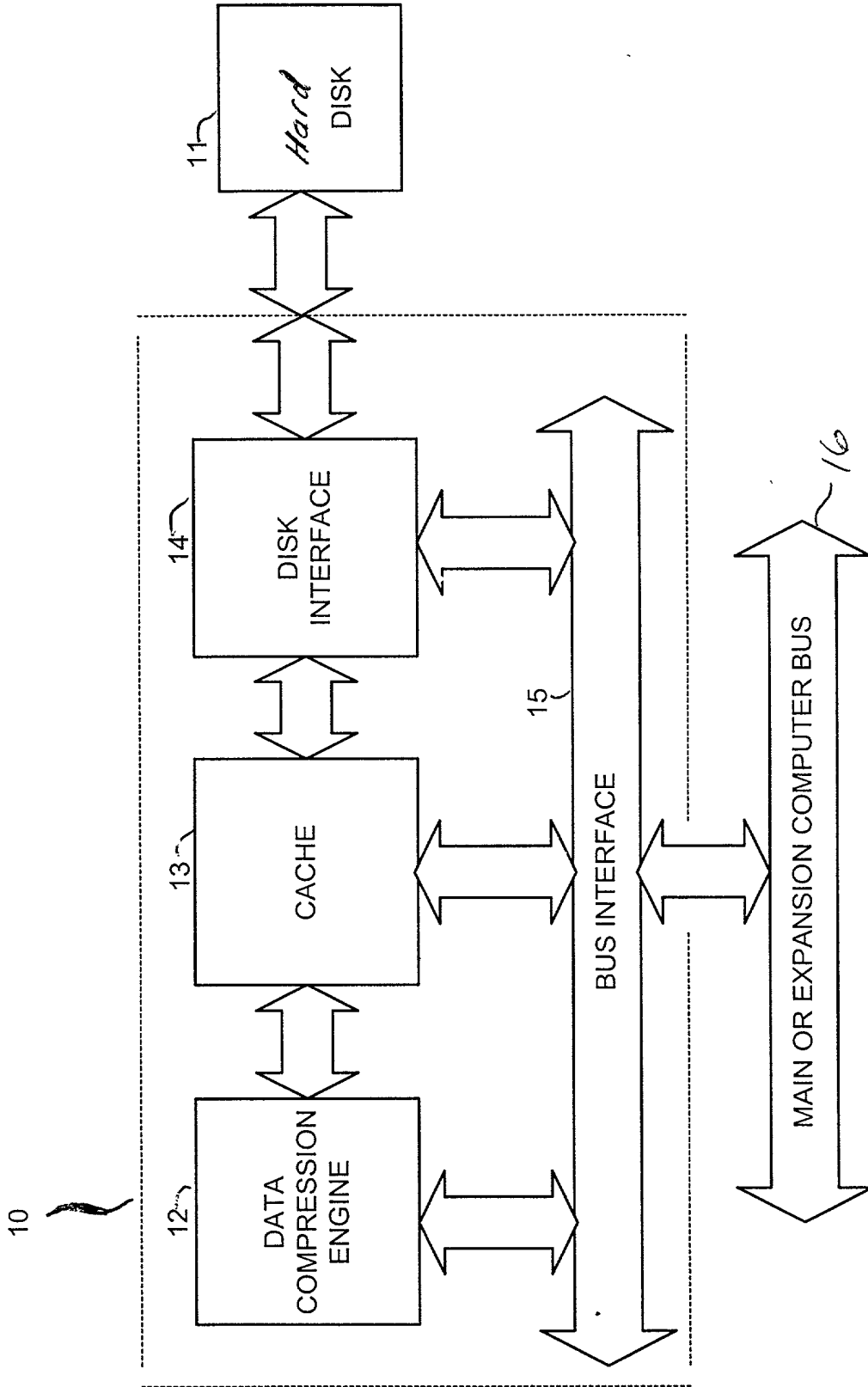


FIGURE 1

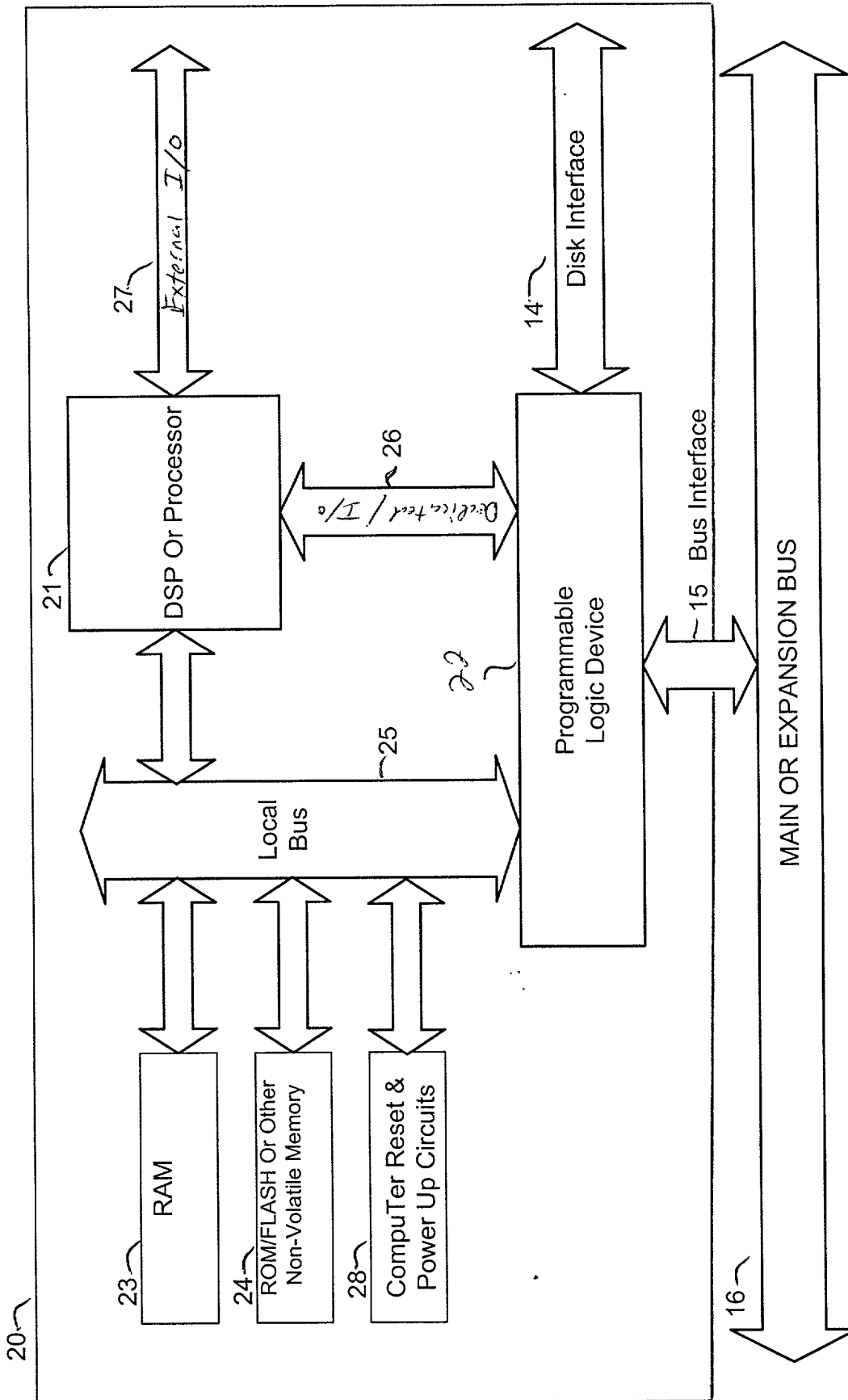


FIGURE 2

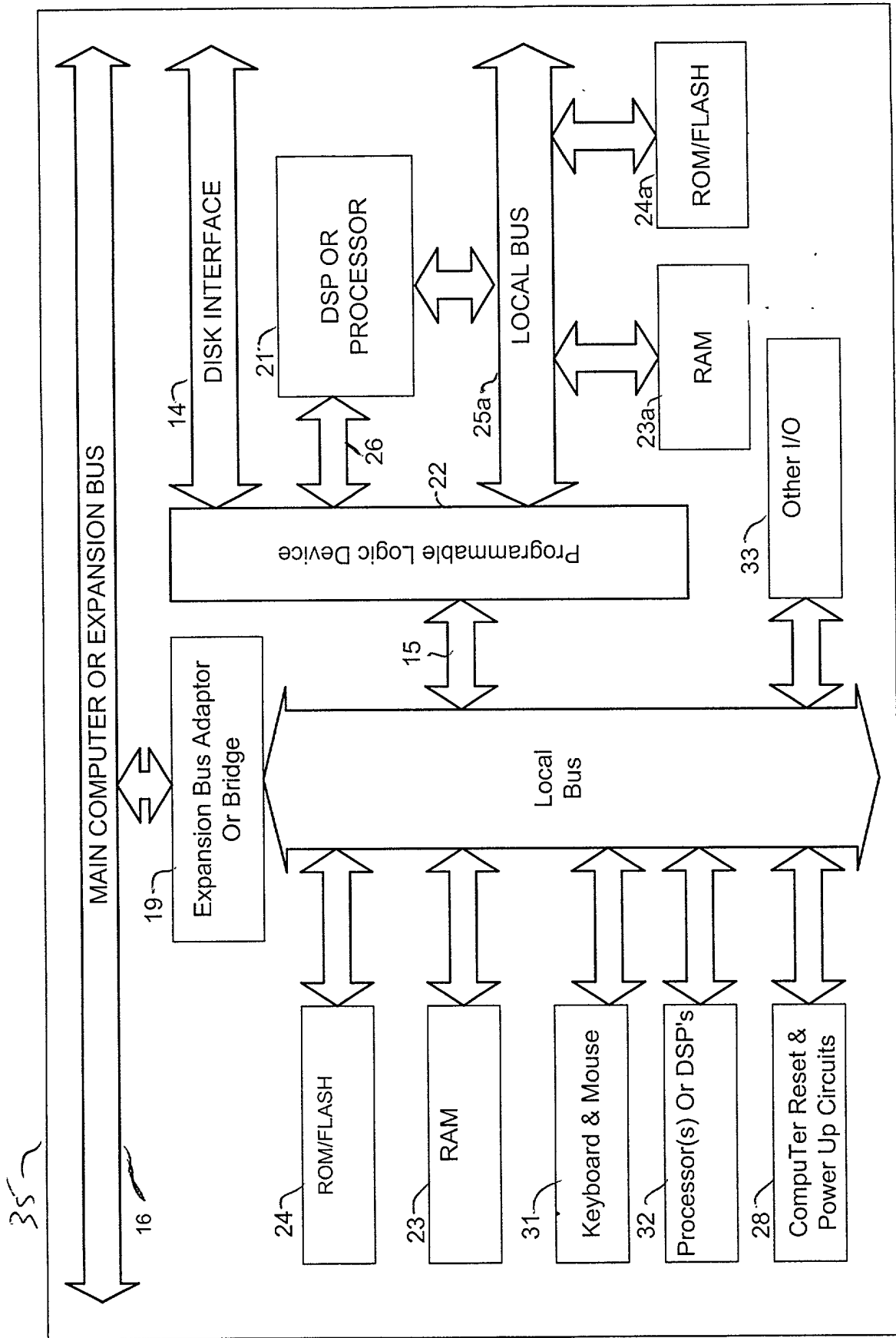


FIGURE 3

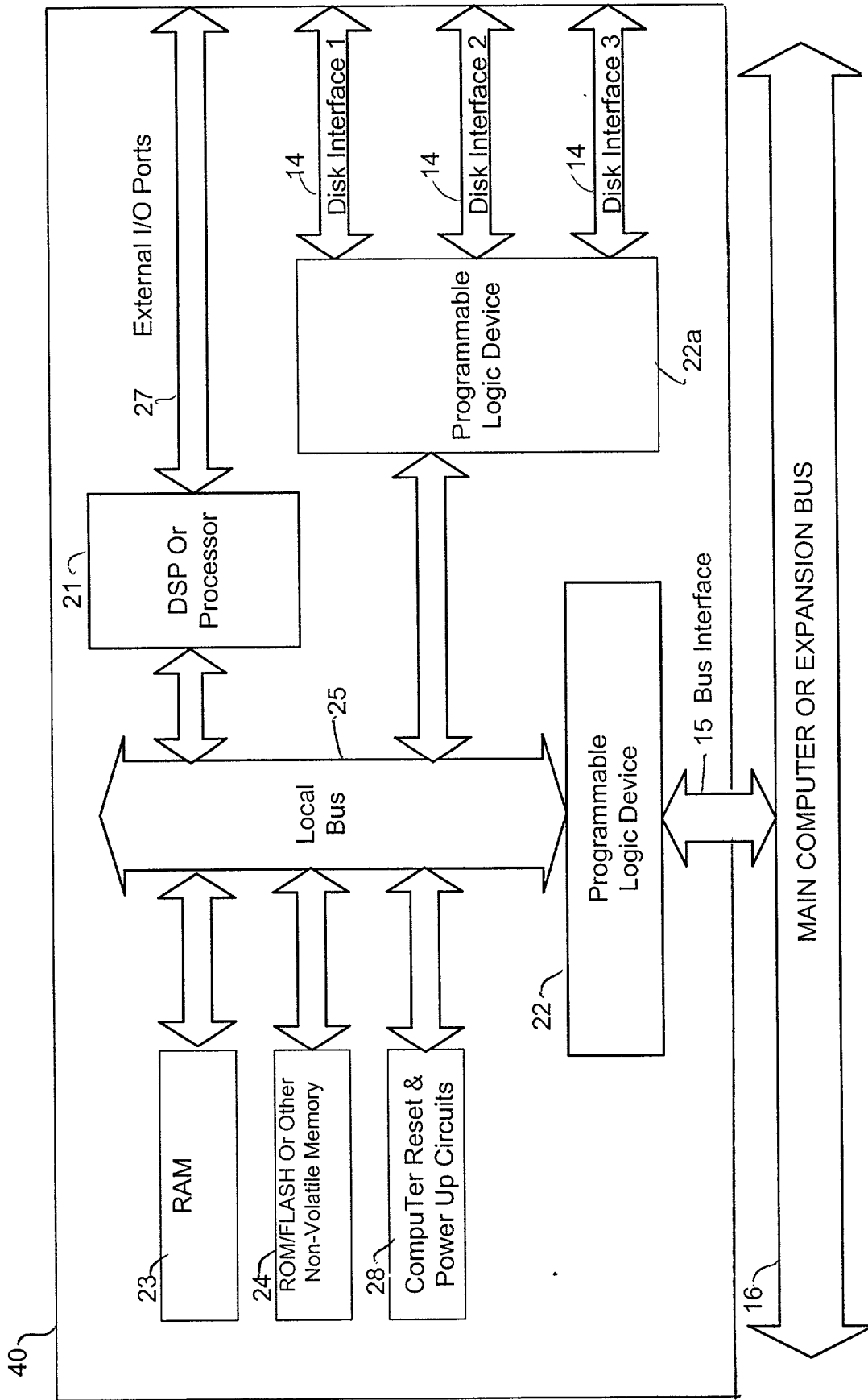


FIGURE 4

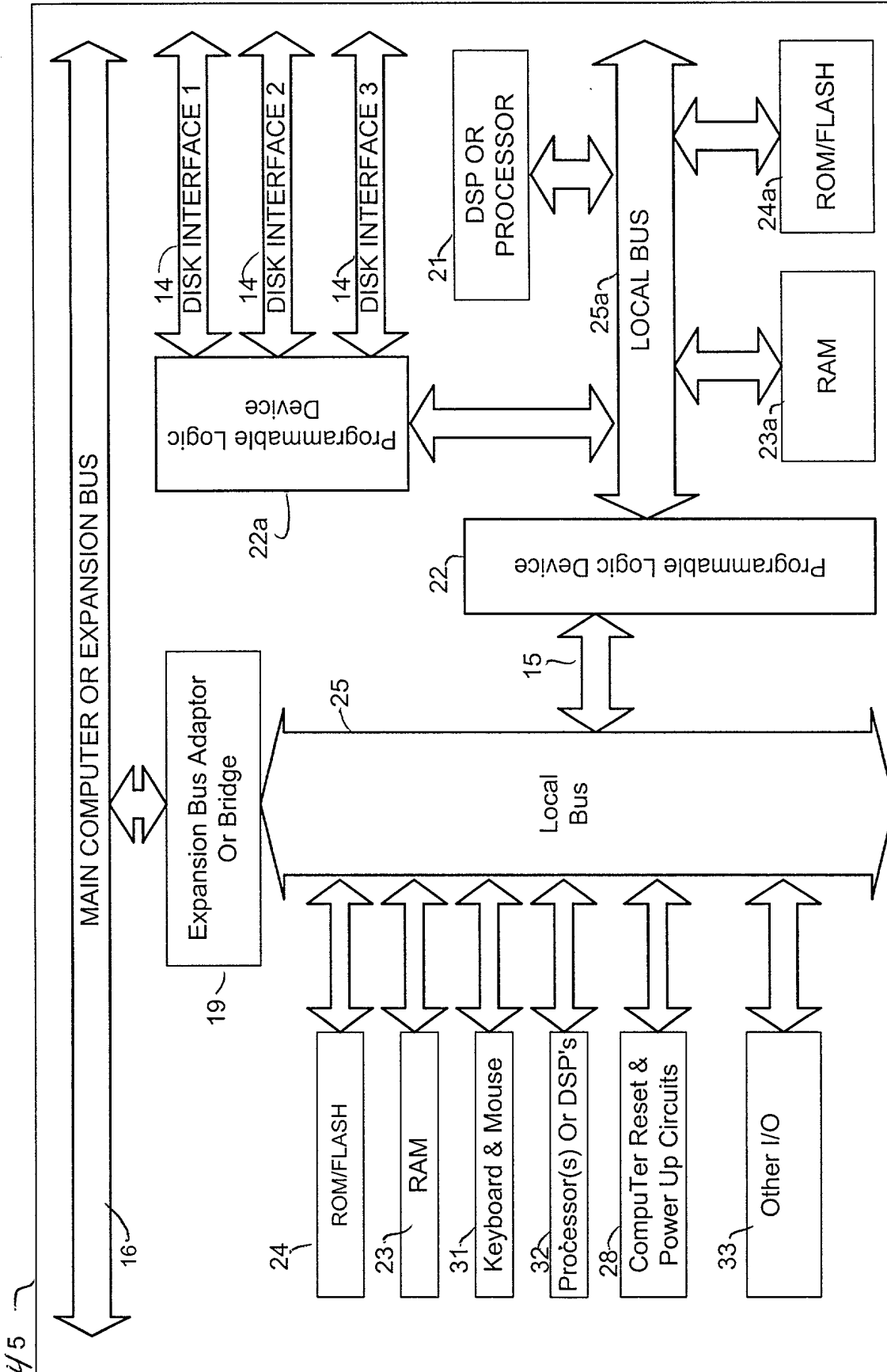


FIGURE 5

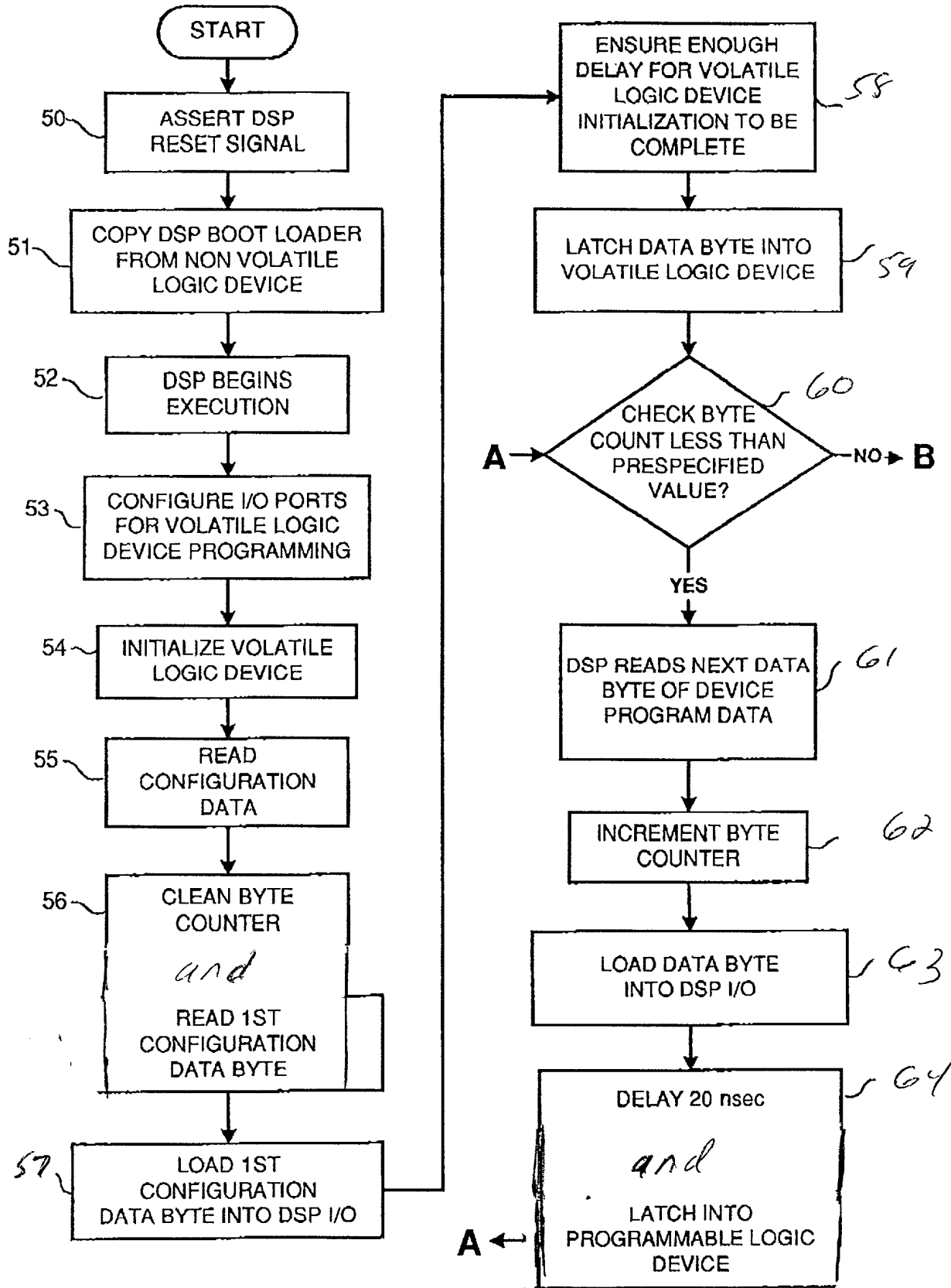


FIG. 6a

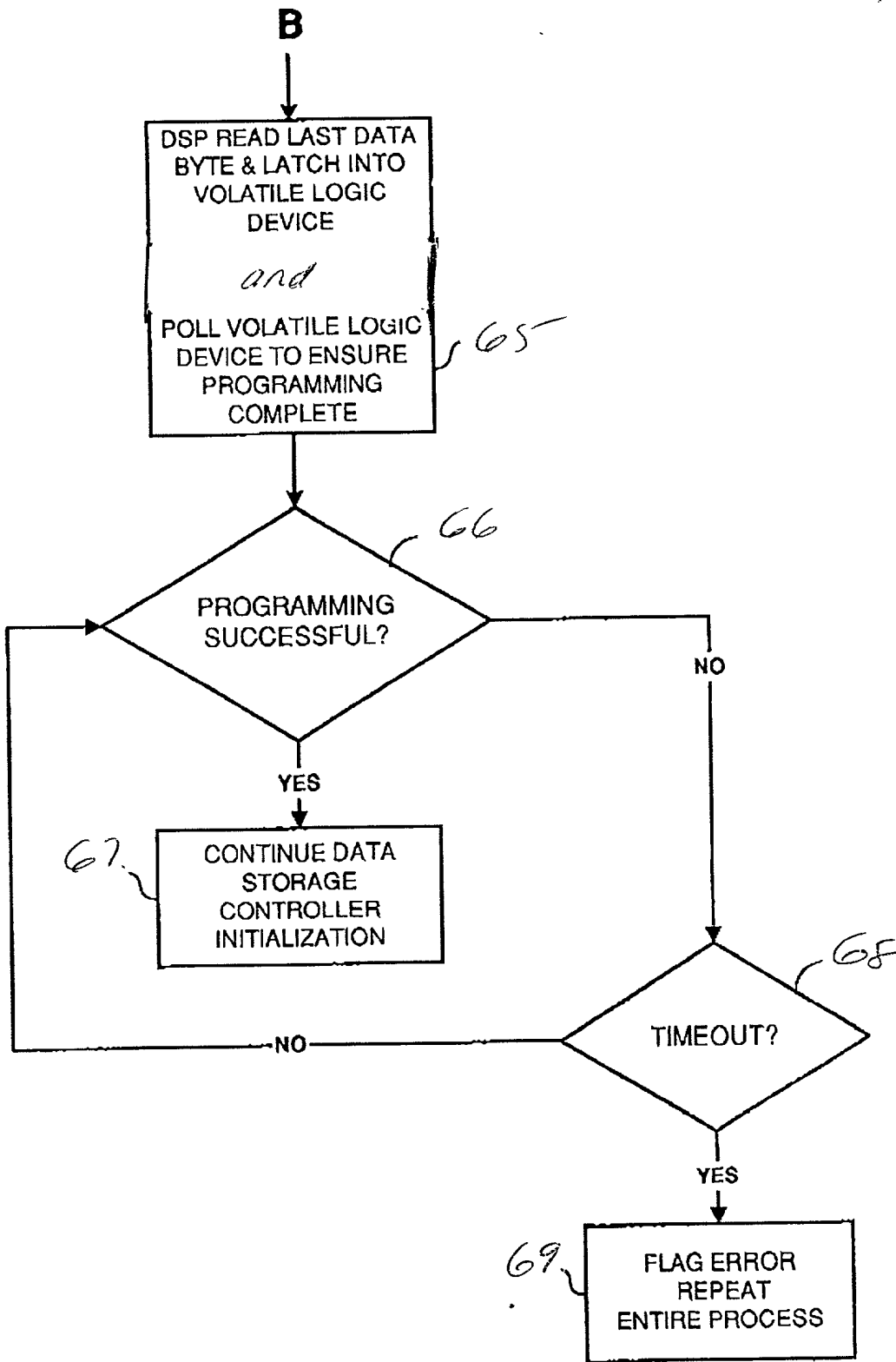


FIG. 6b

Patent # 2,992,263

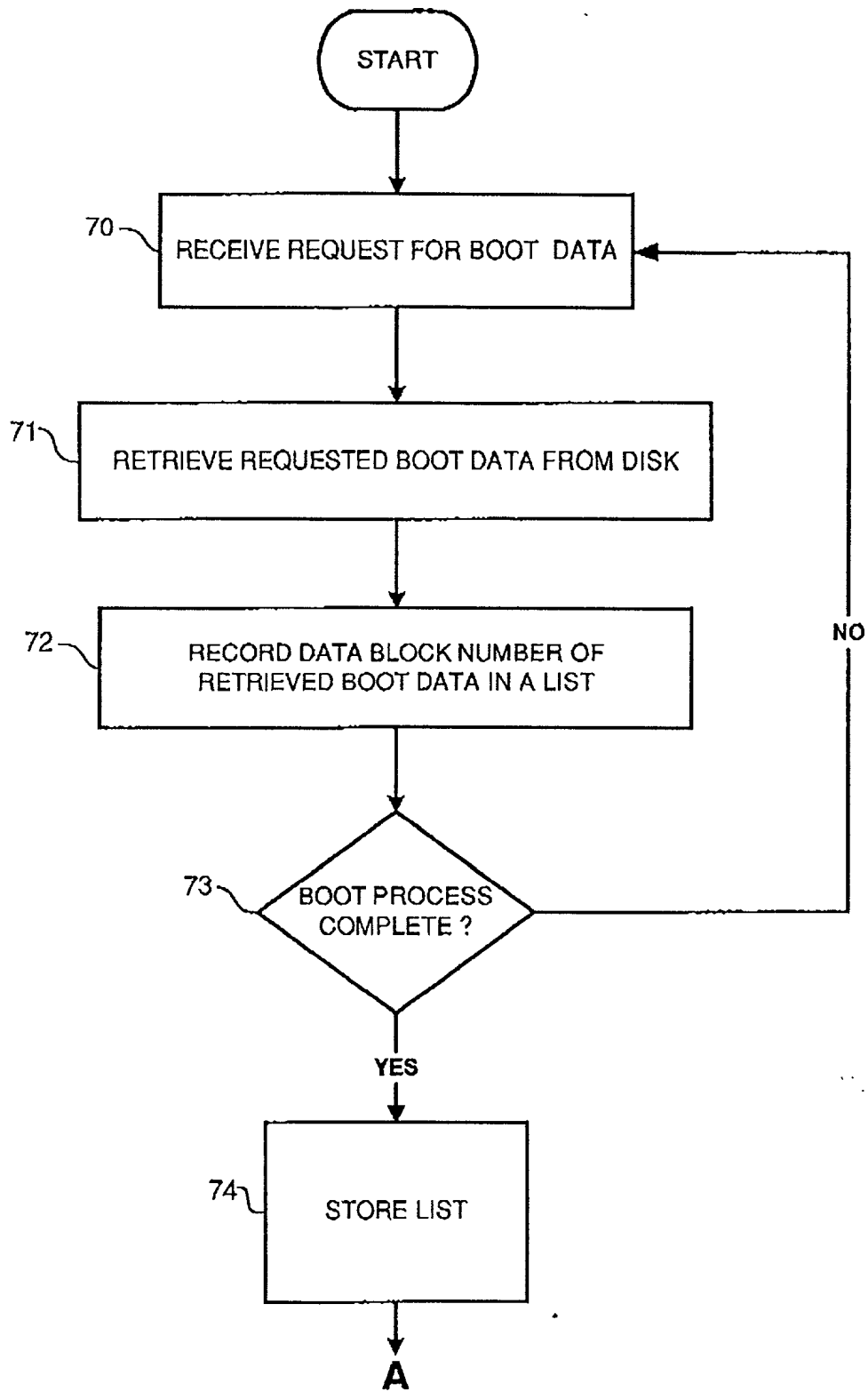


FIG. 7a

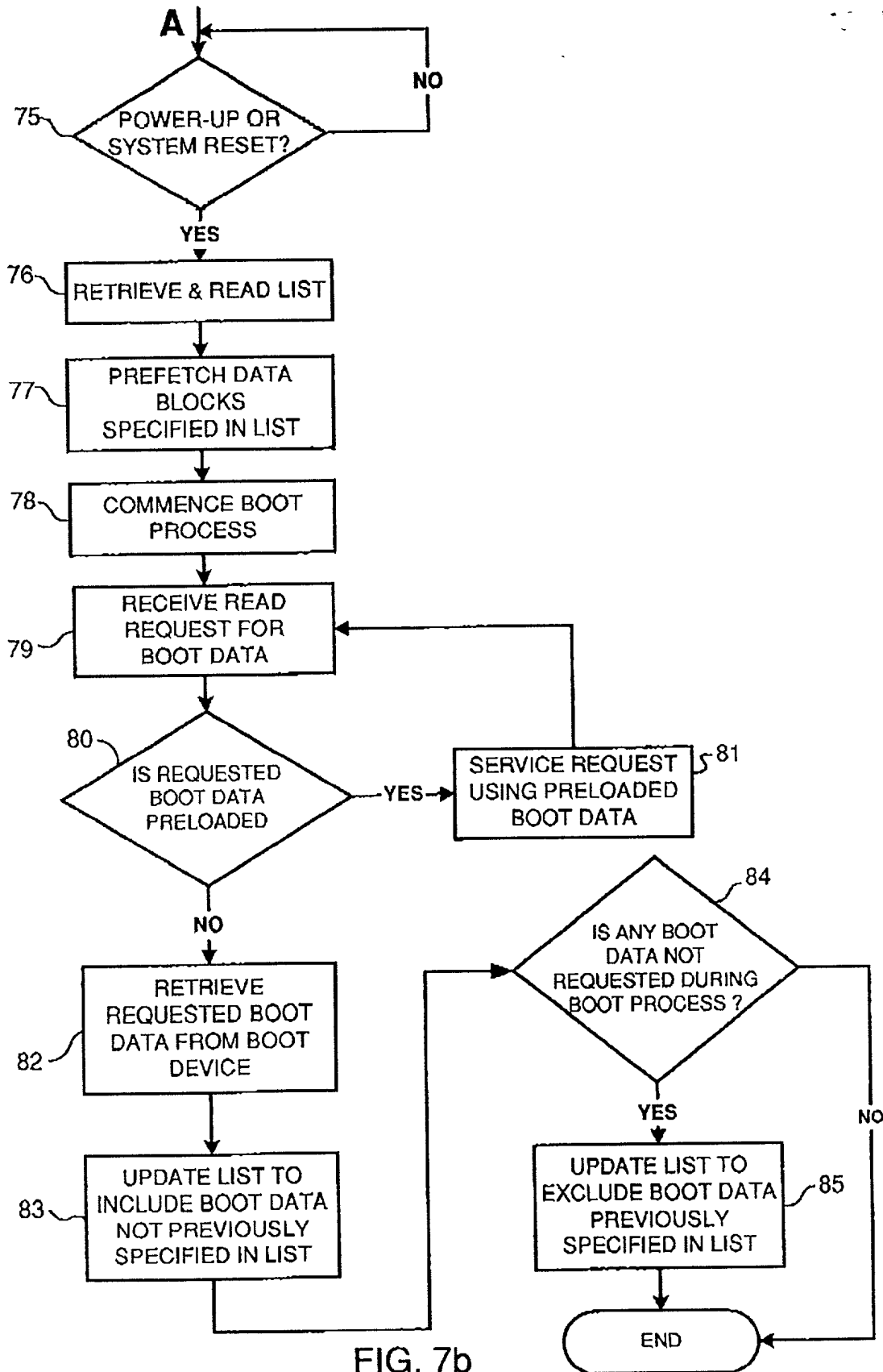


FIG. 7b

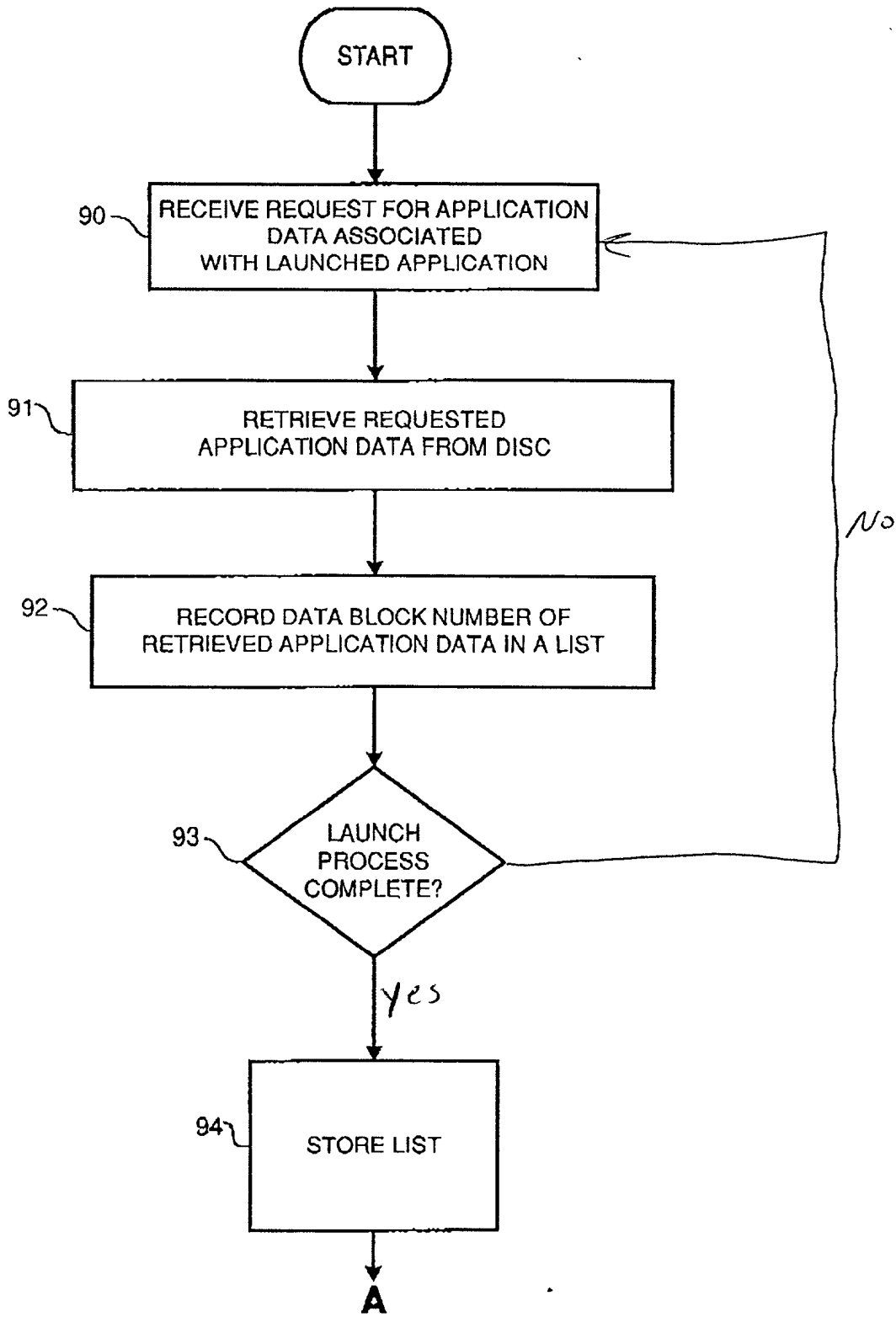


FIG. 8a

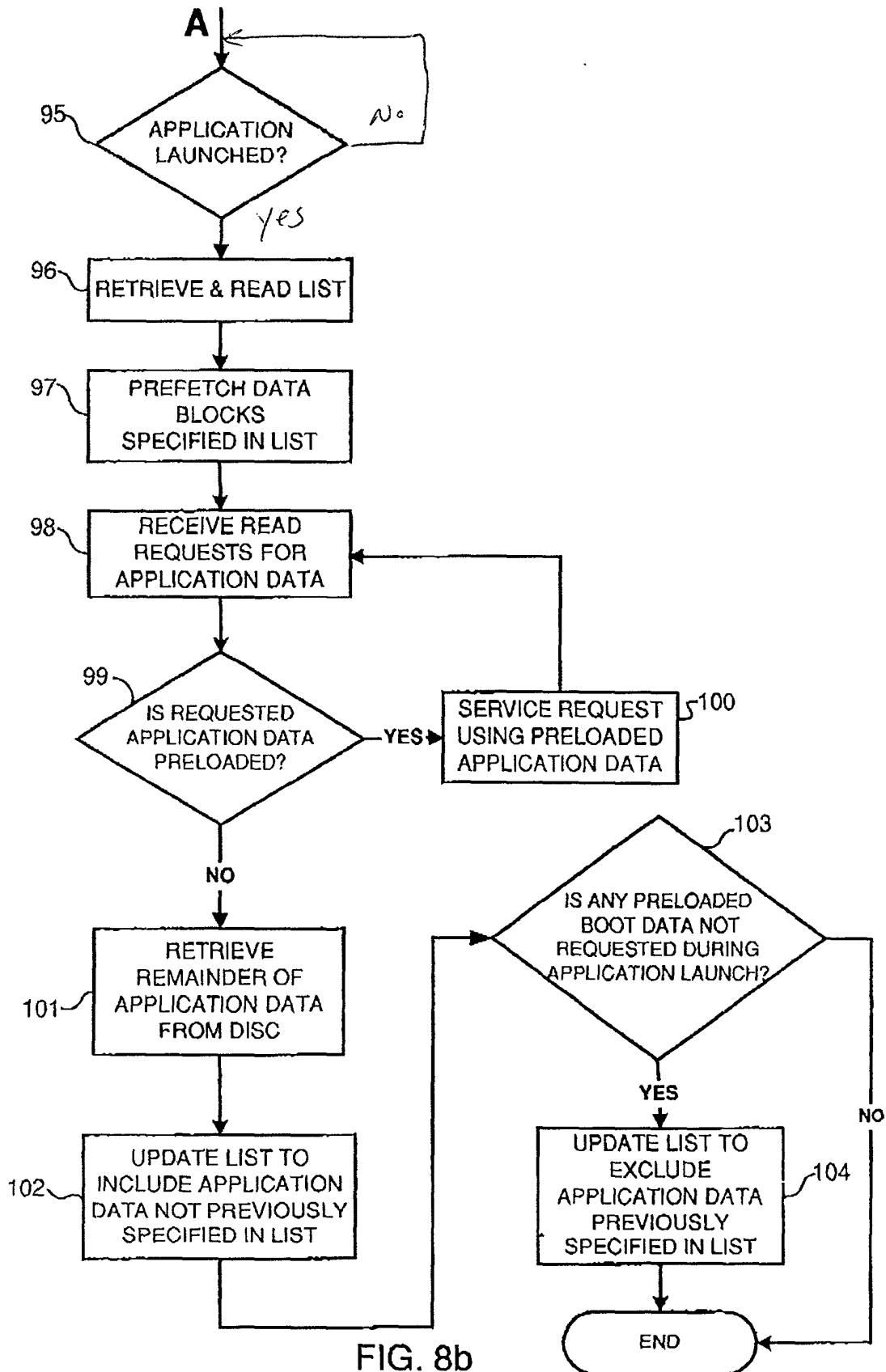


FIG. 8b

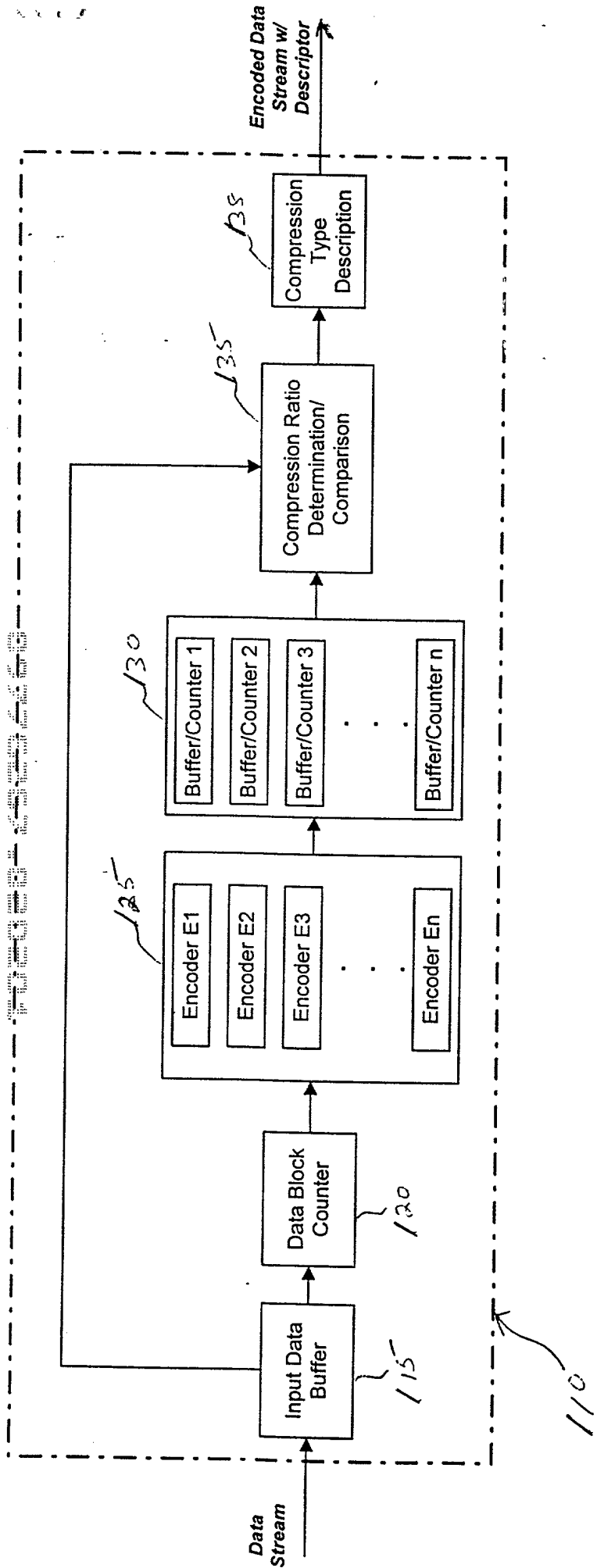


FIGURE 9

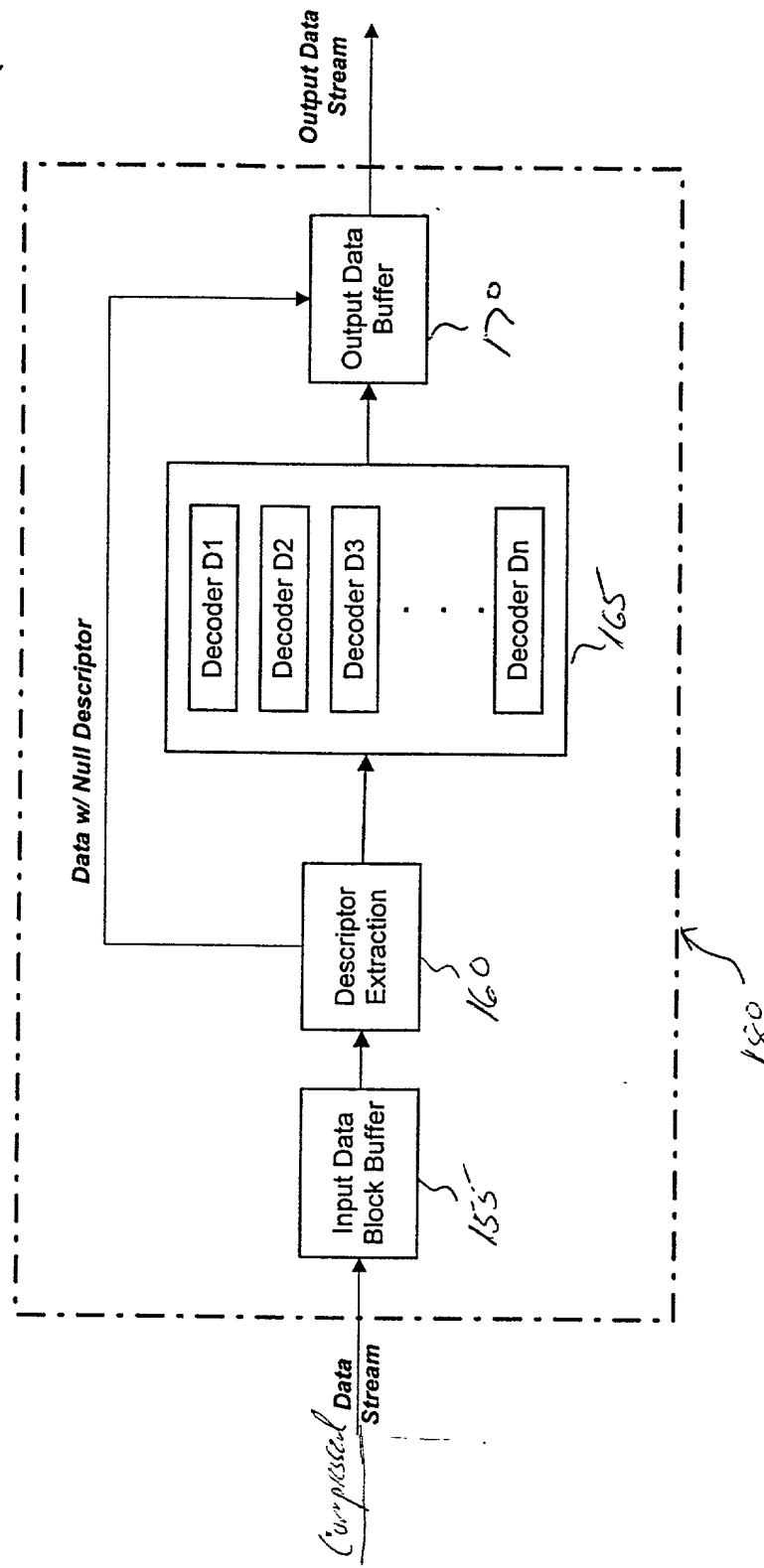


FIG. 10

United States Patent & Trademark Office
Office of Initial Patent Examination

Application papers not suitable for publication

SN 09776267 Mail Date 02-02-01

- Non-English Specification
- Specification contains drawing(s) on page(s) _____ or table(s) _____
- Landscape orientation of text Specification Claims Abstract
- Handwritten Specification Claims Abstract
- More than one column Specification Claims Abstract
- Improper line spacing Specification Claims Abstract
- Claims not on separate page(s)
- Abstract not on separate page(s)
- Improper paper size -- Must be either A4 (21 cm x 29.7 cm) or 8-1/2"x 11"
 - Specification page(s) _____ Abstract
 - Drawing page(s) _____ Claim(s)
- Improper margins
 - Specification page(s) _____ Abstract
 - Drawing page(s) FIG. 9 Claim(s)
- Not reproducible
 - Reason
 - Paper too thin
 - Glossy pages
 - Non-white background
 - Section
 - Specification page(s) _____
 - Drawing page(s) _____
 - Abstract
 - Claim(s)
- Drawing objection(s)
 - Missing lead lines, drawing(s) _____
 - Line quality is too light, drawing(s) _____
 - More than 1 drawing and not numbered correctly
 - Non-English text, drawing(s) _____
 - Excessive text, drawing(s) _____
 - Photographs capable of illustration, drawing(s) _____

FORM 2300-2000



Best Available Copy

SEARCHED

Class	Sub.	Date	Exmr.
713	2	1/28/04	SB3
713	1	1/28/04	SB3
713	100	1/28/04	SB3
711	113	1/29/04	SB3
711	120	1/29/04	SB3

SEARCH NOTES (INCLUDING SEARCH STRATEGY)

	Date	Exmr.
East		
USPAT	1/28/04	SB3
US-PGPHS	1/29/04	
EPAS 3P0	2/2/04	
IBM-T98	2/4/04	
NFL		

INTERFERENCE SEARCHED

Class	Sub.	Date	Exmr.

ISSUE SLIP STAPLE AREA (for additional cross references)

POSITION	INITIALS	ID NO.	DATE
FEE DETERMINATION			
O.I.P.E. CLASSIFIER	ASD		3/2/01
FORMALITY REVIEW	MN	778	3/9/01
RESPONSE FORMALITY REVIEW	pl	1020	5-22-01

INDEX OF CLAIMS

- ✓ Rejected
- = Allowed
- (Through numeral)... Canceled
- ÷ Restricted
- N Non-elected
- I Interference
- A Appeal
- O Objected

Claim	Final	Original	Date
1	✓	✓	
2	✓	✓	
3	✓	✓	
4	✓	✓	
5	✓	✓	
6	✓	✓	
7	✓	✓	
8	✓	✓	
9	✓	✓	
10	✓	✓	
11	✓	✓	
12	✓	✓	
13	✓	✓	
14	✓	✓	
15	✓	✓	
16	✓	✓	
17			
18			
19			
20			
21			
22			
23			
24			
25			
26			
27			
28			
29			
30			
31			
32			
33			
34			
35			
36			
37			
38			
39			
40			
41			
42			
43			
44			
45			
46			
47			
48			
49			
50			

Claim	Final	Original	Date
51			
52			
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			
69			
70			
71			
72			
73			
74			
75			
76			
77			
78			
79			
80			
81			
82			
83			
84			
85			
86			
87			
88			
89			
90			
91			
92			
93			
94			
95			
96			
97			
98			
99			
100			

Claim	Final	Original	Date
101			
102			
103			
104			
105			
106			
107			
108			
109			
110			
111			
112			
113			
114			
115			
116			
117			
118			
119			
120			
121			
122			
123			
124			
125			
126			
127			
128			
129			
130			
131			
132			
133			
134			
135			
136			
137			
138			
139			
140			
141			
142			
143			
144			
145			
146			
147			
148			
149			
150			

Best Available Copy

If more than 150 claims or 10 actions
staple additional sheet here

(LEFT INSIDE)

02-03-01

A

PTO/SB/05 (08-00)

Approved for use through 10/31/2002. OMB 0651-0032

U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Please type a plus sign (+) inside this box → Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

1c965 U.S. PTO 02/02/01

UTILITY PATENT APPLICATION TRANSMITTAL

Attorney Docket No.	8011-15
First Inventor	FALLON
Title	SYSTEMS AND METHODS FOR ACCELERATED
Express Mail Label No.	EL679454191US

(Only for new nonprovisional applications under 37 CFR 1.53(b))

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

- Fee Transmittal Form (e.g., PTO/SB/17) (Submit an original and a duplicate for fee processing)
- Applicant claims small entity status. See 37 CFR 1.27.
- Specification [Total Pages] (preferred arrangement set forth below)
 - Descriptive title of the invention
 - Cross Reference to Related Applications
 - Statement Regarding Fed sponsored R & D
 - Reference to sequence listing, a table, or a computer program listing appendix
 - Background of the Invention
 - Brief Summary of the Invention
 - Brief Description of the Drawings (if filed)
 - Detailed Description
 - Claim(s)
 - Abstract of the Disclosure
- Drawing(s) (35 U.S.C. 113) [Total Sheets]
- Oath or Declaration [Total Pages]
 - Newly executed (original or copy)
 - Copy from a prior application (37 CFR 1.63 (d)) (for continuation/divisional with Box 17 completed)
 - DELETION OF INVENTOR(S)**
Signed statement attached deleting inventor(s) named in the prior application, see 37 CFR 1.63(d)(2) and 1.33(b).
- Application Data Sheet. See 37 CFR 1.76

ADDRESS TO: Assistant Commissioner for Patents
Box Patent Application
Washington, DC 20231

- CD-ROM or CD-R in duplicate, large table or Computer Program (Appendix)
- Nucleotide and/or Amino Acid Sequence Submission (if applicable, all necessary)
 - Computer Readable Form (CRF)
 - Specification Sequence Listing on:
 - CD-ROM or CD-R (2 copies); or
 - paper
 - Statements verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

- Assignment Papers (cover sheet & document(s))
- 37 CFR 3.73(b) Statement of Attorney (when there is an assignee) Power of Attorney
- English Translation Document (if applicable)
- Information Disclosure Statement (IDS)/PTO-1449 Copies of IDS Citations
- Preliminary Amendment
- Return Receipt Postcard (MPEP 503) (Should be specifically itemized)
- Certified Copy of Priority Document(s) (if foreign priority is claimed)
- Other: Check for \$355.00

17. If a CONTINUING APPLICATION, check appropriate box, and supply the requisite information below and in a preliminary amendment, or in an Application Data Sheet under 37 CFR 1.76:

Continuation Divisional Continuation-in-part (CIP)


of prior application No.: _____ / _____

Prior application information: Examiner _____

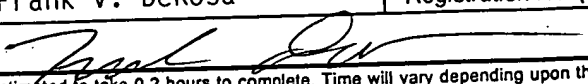
Group / Art Unit: _____

For CONTINUATION OR DIVISIONAL APPS only: The entire disclosure of the prior application, from which an oath or declaration is supplied under Box 5b, is considered a part of the disclosure of the accompanying continuation or divisional application and is hereby incorporated by reference. The incorporation can only be relied upon when a portion has been inadvertently omitted from the submitted application parts.

18. CORRESPONDENCE ADDRESS

Customer Number or Bar Code Label  or Correspondence address below

Name	Frank Chau, Esq.				
Address	F. CHAU & ASSOCIATES, LLP				
	1900 Hempstead Turnpike, Suite 501				
City	East Meadow	State	New York	Zip Code	11554
Country	USA	Telephone	516-357-0091	Fax	516-357-0092

Name (Print/Type)	Frank V. DeRosa	Registration No. (Attorney/Agent)	43,584
Signature		Date	2/2/01

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

FEE TRANSMITTAL for FY 2001

Patent fees are subject to annual revision.

Complete if Known

Application Number	
Filing Date	February 2, 2001
First Named Inventor	James J. Fallon
Examiner Name	
Group Art Unit	
Attorney Docket No.	8011-15

TOTAL AMOUNT OF PAYMENT (\$) 355.00

METHOD OF PAYMENT

1. The Commissioner is hereby authorized to charge indicated fees and credit any overpayments to:

Deposit Account Number: 50-0679
 Deposit Account Name: F. Chau & Associates, LLP

Charge Any Additional Fee Required Under 37 CFR 1.16 and 1.17

Applicant claims small entity status. See 37 CFR 1.27

2. Payment Enclosed:

Check Credit card Money Order Other

FEE CALCULATION

1. BASIC FILING FEE

Large Entity Code (\$)	Small Entity Code (\$)	Fee (\$)	Fee (\$)	Fee Description	Fee Paid
101	710	201	355	Utility filing fee	355
106	320	206	160	Design filing fee	
107	490	207	245	Plant filing fee	
108	710	208	355	Reissue filing fee	
114	150	214	75	Provisional filing fee	

SUBTOTAL (1) (\$) 355.00

2. EXTRA CLAIM FEES

Total Claims	16	-20** =	0	x	9	=	0	Fee Paid
Independent Claims	3	-3** =	0	x	40	=	0	
Multiple Dependent					135	=		

Large Entity Code (\$)	Small Entity Code (\$)	Fee (\$)	Fee (\$)	Fee Description	Fee Paid
103	18	203	9	Claims in excess of 20	
102	80	202	40	Independent claims in excess of 3	
104	270	204	135	Multiple dependent claim, if not paid	
109	80	209	40	** Reissue independent claims over original patent	
110	18	210	9	** Reissue claims in excess of 20 and over original patent	

SUBTOTAL (2) (\$) 0

**or number previously paid, if greater; For Reissues, see above

FEE CALCULATION (continued)

3. ADDITIONAL FEES

Large Entity Code (\$)	Small Entity Code (\$)	Fee (\$)	Fee (\$)	Fee Description	Fee Paid
105	130	205	65	Surcharge - late filing fee or oath	
127	50	227	25	Surcharge - late provisional filing fee or cover sheet	
139	130	139	130	Non-English specification	
147	2,520	147	2,520	For filing a request for <i>ex parte</i> reexamination	
112	920*	112	920*	Requesting publication of SIR prior to Examiner action	
113	1,840*	113	1,840*	Requesting publication of SIR after Examiner action	
115	110	215	55	Extension for reply within first month	
116	390	216	195	Extension for reply within second month	
117	890	217	445	Extension for reply within third month	
118	1,390	218	695	Extension for reply within fourth month	
128	1,890	228	945	Extension for reply within fifth month	
119	310	219	155	Notice of Appeal	
120	310	220	155	Filing a brief in support of an appeal	
121	270	221	135	Request for oral hearing	
138	1,510	138	1,510	Petition to institute a public use proceeding	
140	110	240	55	Petition to revive - unavoidable	
141	1,240	241	620	Petition to revive - unintentional	
142	1,240	242	620	Utility issue fee (or reissue)	
143	440	243	220	Design issue fee	
144	600	244	300	Plant issue fee	
122	130	122	130	Petitions to the Commissioner	
123	50	123	50	Petitions related to provisional applications	
126	240	126	240	Submission of Information Disclosure Stmt	
581	40	581	40	Recording each patent assignment per property (times number of properties)	
146	710	246	355	Filing a submission after final rejection (37 CFR § 1.129(a))	
149	710	249	355	For each additional invention to be examined (37 CFR § 1.129(b))	
179	710	279	355	Request for Continued Examination (RCE)	
169	900	169	900	Request for expedited examination of a design application	

Other fee (specify) _____

*Reduced by Basic Filing Fee Paid

SUBTOTAL (3) (\$) 0

SUBMITTED BY

Name (Print/Type)	Frank V. DeRosa	Registration No. (Attorney/Agent)	43,584	Telephone	(516)357-0091
Signature		Date	2/2/01		

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Washington, DC 20231.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Assistant Commissioner for Patents
Washington, D.C. 20231

UTILITY APPLICATION FEE TRANSMITTAL

PTO
11046 U.S. PTO
09/776267
02/02/01

Sir:

Transmitted herewith for filing is the patent application of

Inventor(s): James J. Fallon, John Buck, Paul F. Pickel,
Stephen J. McEerlain

For: SYSTEMS AND METHODS FOR ACCELERATED LOADING OF
OPERATING SYSTEMS AND APPLICATION PROGRAMS

Enclosed are:

- 52 page(s) of specification
- 1 page(s) of Abstract
- 4 page(s) of claims
- 13 sheets of drawings [] formal [X] informal
- [] _____ page(s) of Declaration and Power of Attorney
- [] An Assignment of the invention to: _____

CERTIFICATION UNDER 37 C.F.R. § 1.10

I hereby certify that this New Application Transmittal and the documents referred to as enclosed therein are being deposited with the United States Postal Service on this date February 2, 2001 in an envelope as "Express Mail Post Office to Addressee" Mail Label Number EL679454191US addressed to: Assistant Commissioner for Patents, Washington, D.C. 20231.

Frank V. DeRosa
(Type or print name of person mailing paper)


(Signature of person mailing paper)

1c965 U.S. PTO
02/02/01

0076887 00000004

[X] This application claims the benefit under 35 U.S.C. §119(e) of U.S. Provisional Application(s) No(s) .:

APPLICATION NO(S) .:

FILING DATE

60/180,114

February 3, 2000

/

[] Certified copy of applications

Country

Appln. No.

Filed

from which priority under Title 35 United States Code, § 119 is claimed

[] is enclosed.

[] will follow.

CALCULATION OF UTILITY APPLICATION FEE

For	Number Filed	Number Extra	Rate	Basic Fee
Total				\$710.00
Claims*	16	-20 = 0	x \$ 18.00	\$.00
Independent				
Claims	3	-3 = 0	x \$ 80.00	\$.00
Multiple	[] yes	Add'l. Fee	\$270.00	\$
Dependent				
Claims	[] no	Add'l. Fee	None	= \$
				TOTAL \$ 710.00

[X] "Small Entity" Status Claimed Under 37 C.F.R. § 1.27. Reduced fees under 37 C.F.R. § 1.9(f) (50% of total) paid herewith \$355.00.

*Includes all independent and single dependent claims and all claims referred to in multiple claims. See 37 C.F.R. § 1.75(c).

- [] A check in the amount of \$ _____ is enclosed for recording the attached Assignment.
- [X] A check in the amount of \$355.00 to cover the filing fee is attached.
- [] Charge fee to Deposit Account No. 50-0679. Order No. 50-0679. TWO (2) COPIES OF THIS SHEET ARE ENCLOSED.
- [X] Please charge any deficiency as well as any other fee(s) which may become due under 37 C.F.R. § 1.16 and 1.17, at any time during the pendency of this application, or credit any overpayment of such fee(s) to Deposit Account No. 50-0679. Also, in the event any extensions of time for responding are required for the pending application(s), please treat this paper as a petition to extend the time as required and charge Deposit Account No. 50-0679 therefor. TWO (2) COPIES OF THIS SHEET ARE ENCLOSED.

Date:

2/2/01



SIGNATURE OF ATTORNEY
Frank V. DeRosa
Reg. No. 43,584

F. CHAU & ASSOCIATES, LLP
1900 Hempstead Turnpike
Suite 501
East Meadow, New York 11554
Tel. No. (516) 357-0091
Fax. (516) 357-0092
FVD:pg

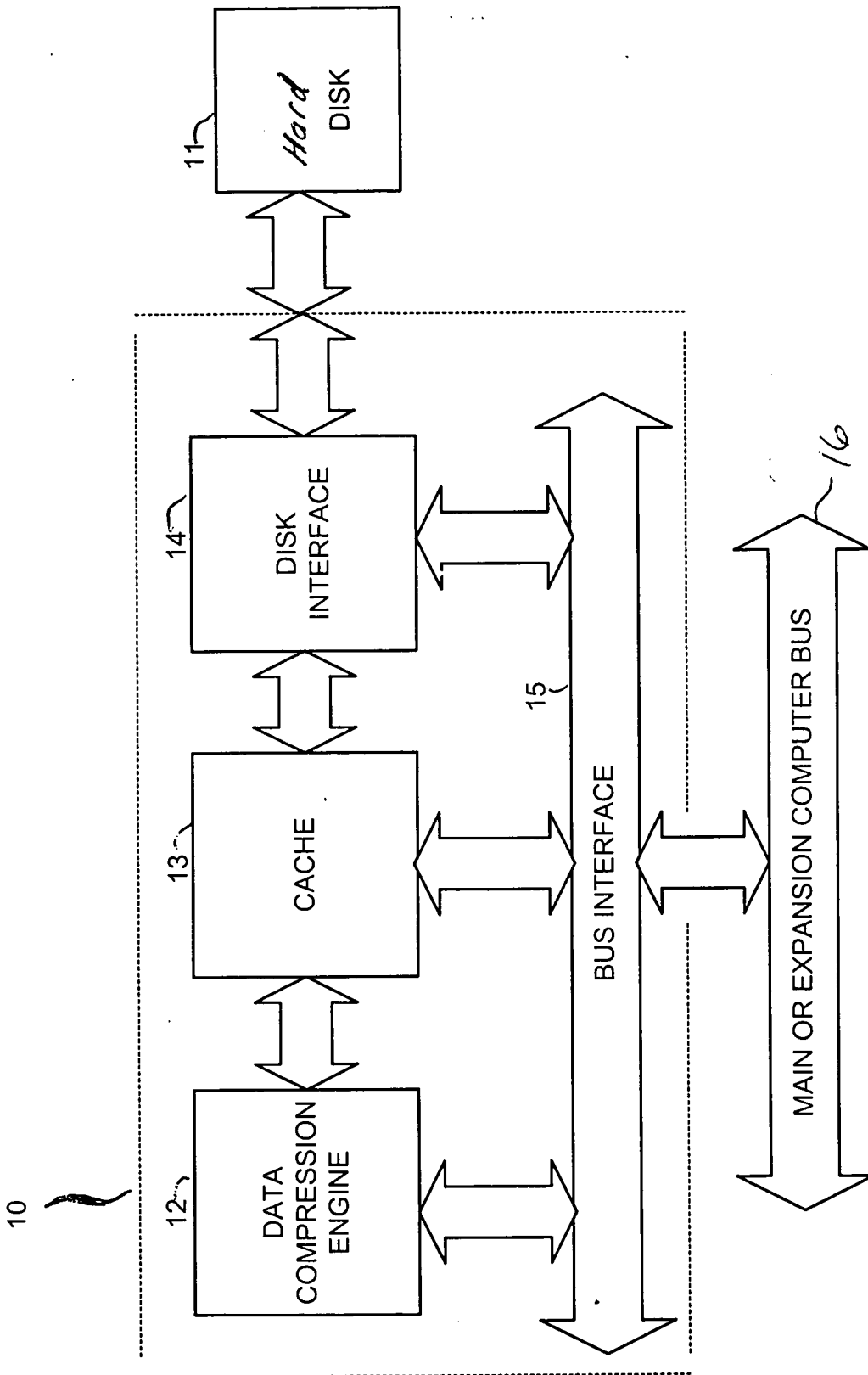


FIGURE 1

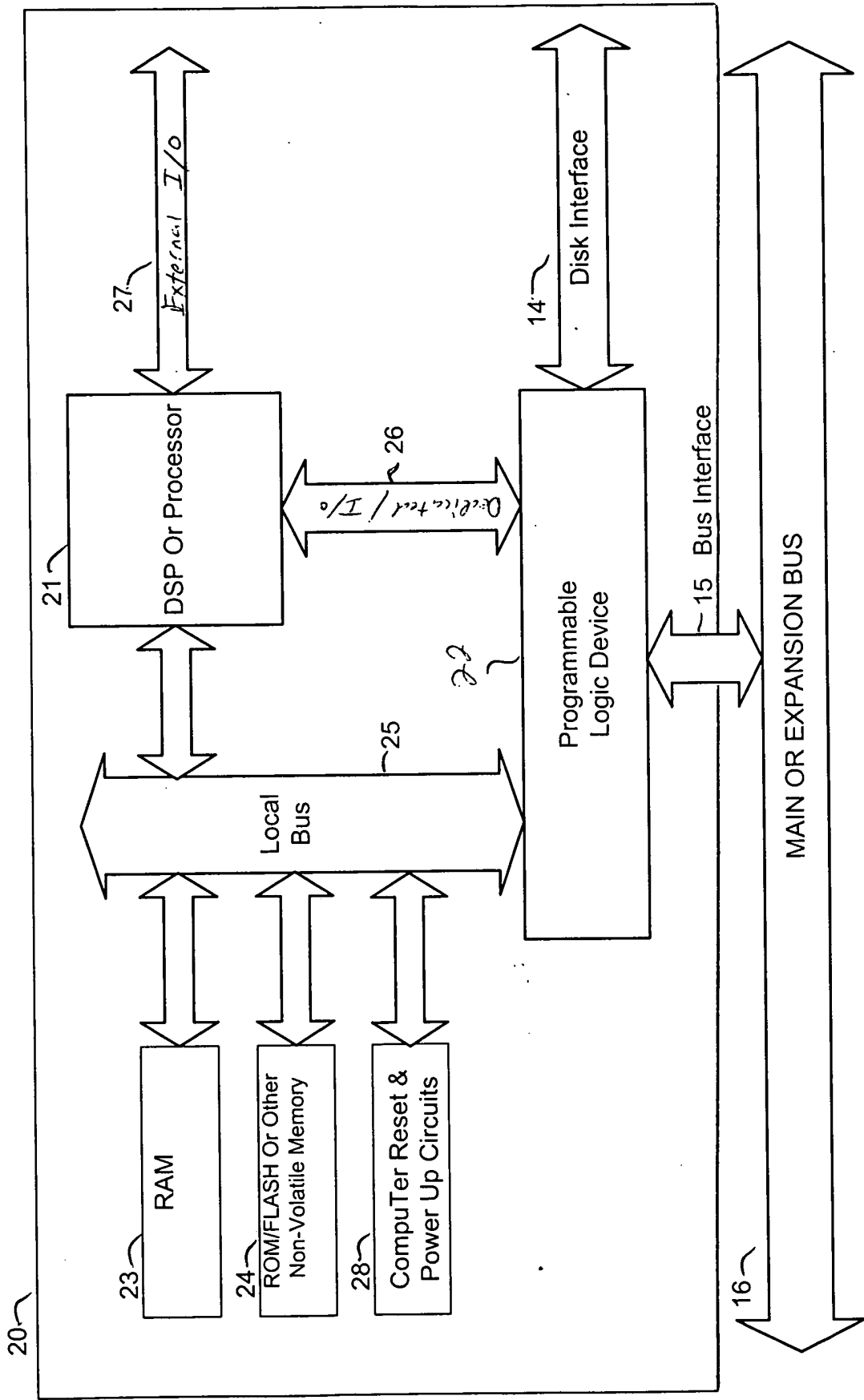


FIGURE 2

35

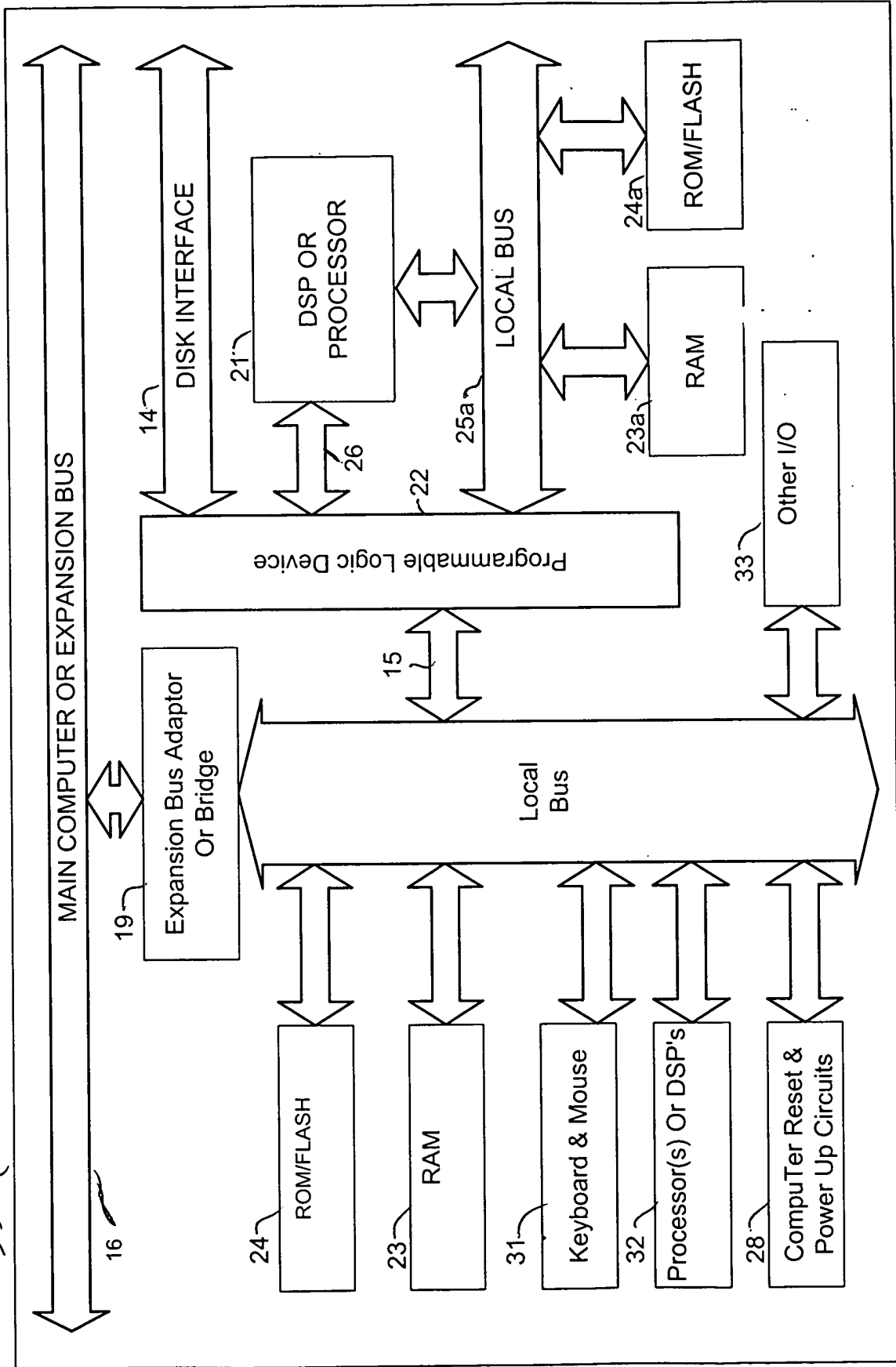


FIGURE 3

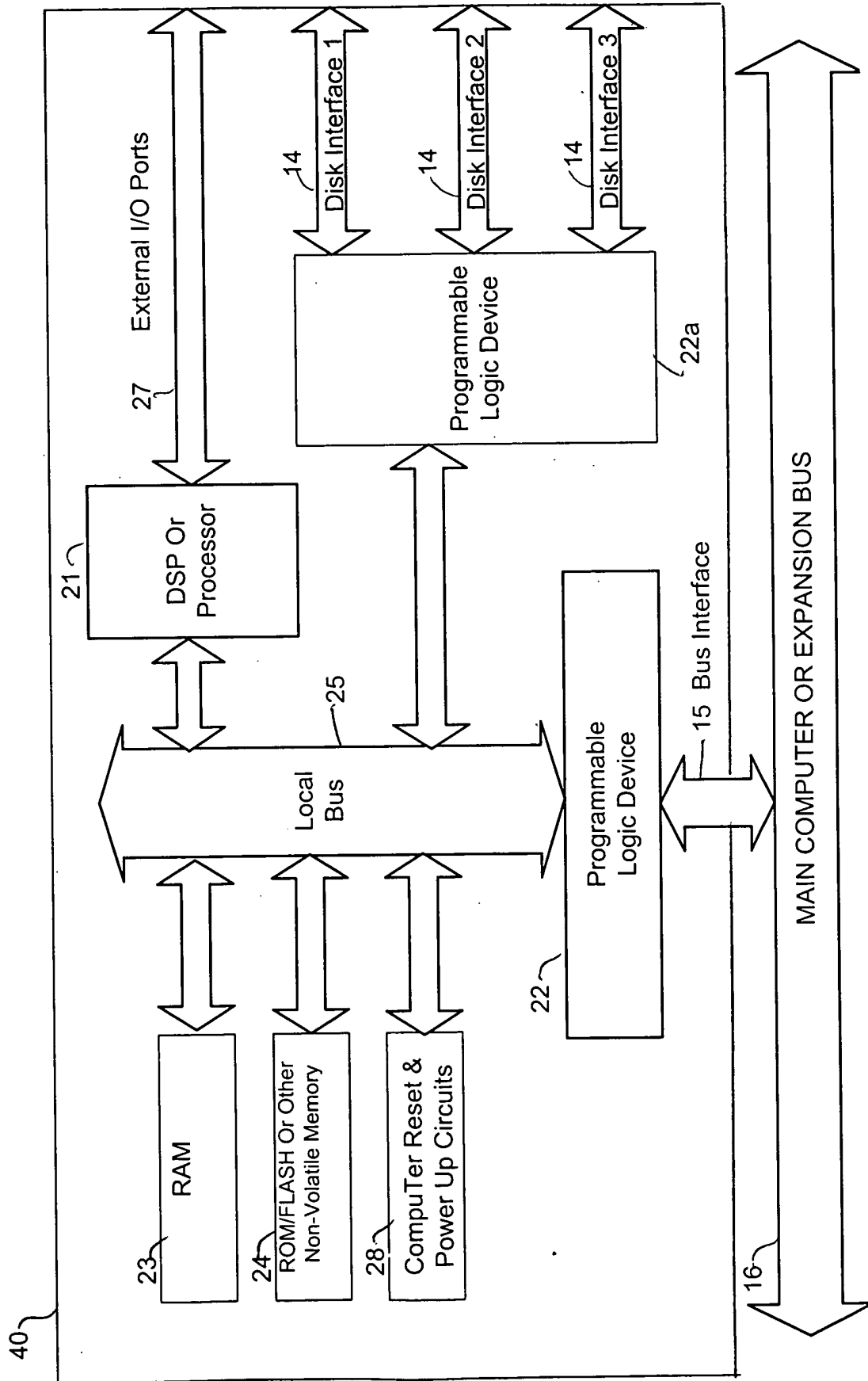


FIGURE 4

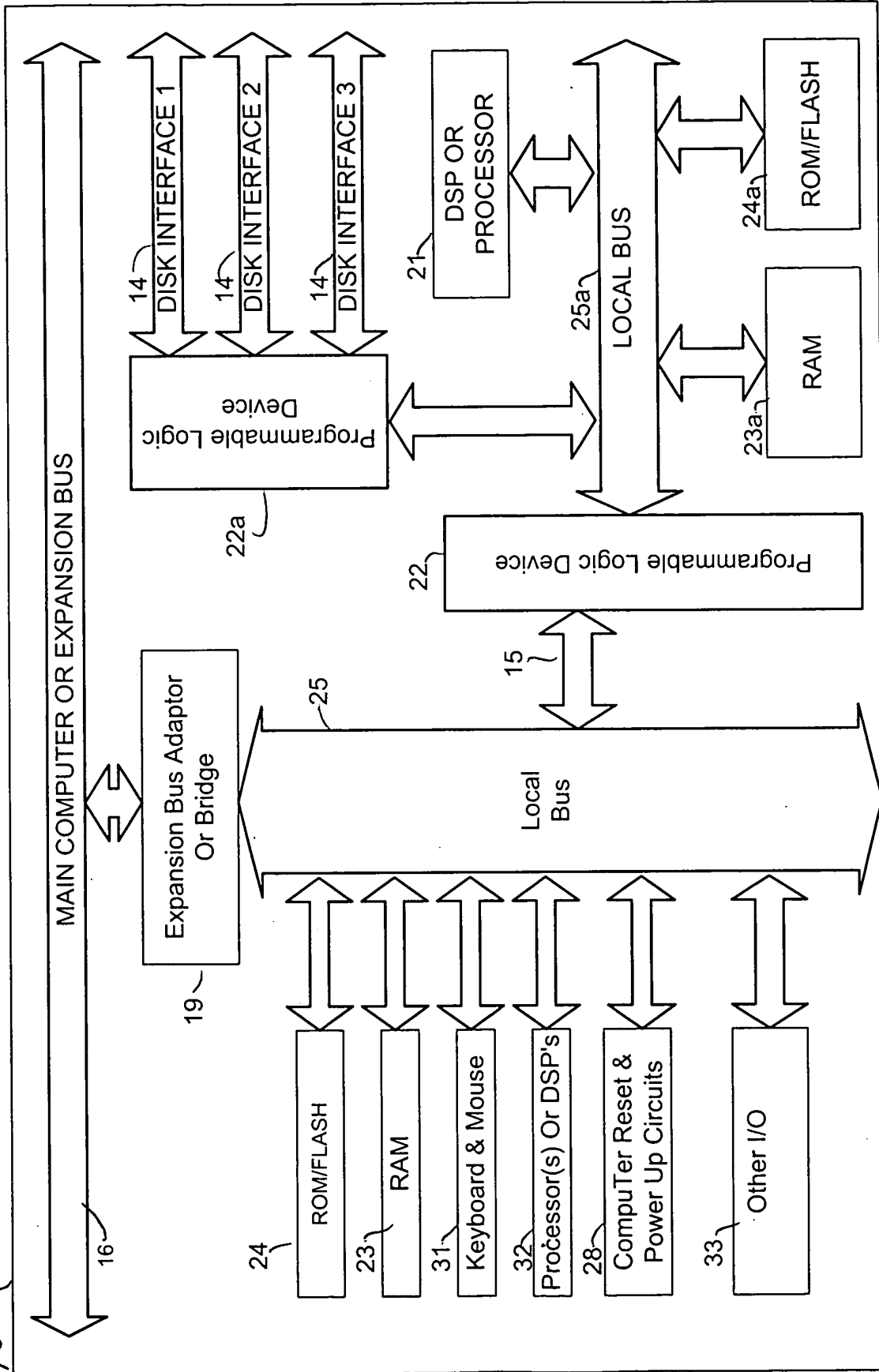


FIGURE 5

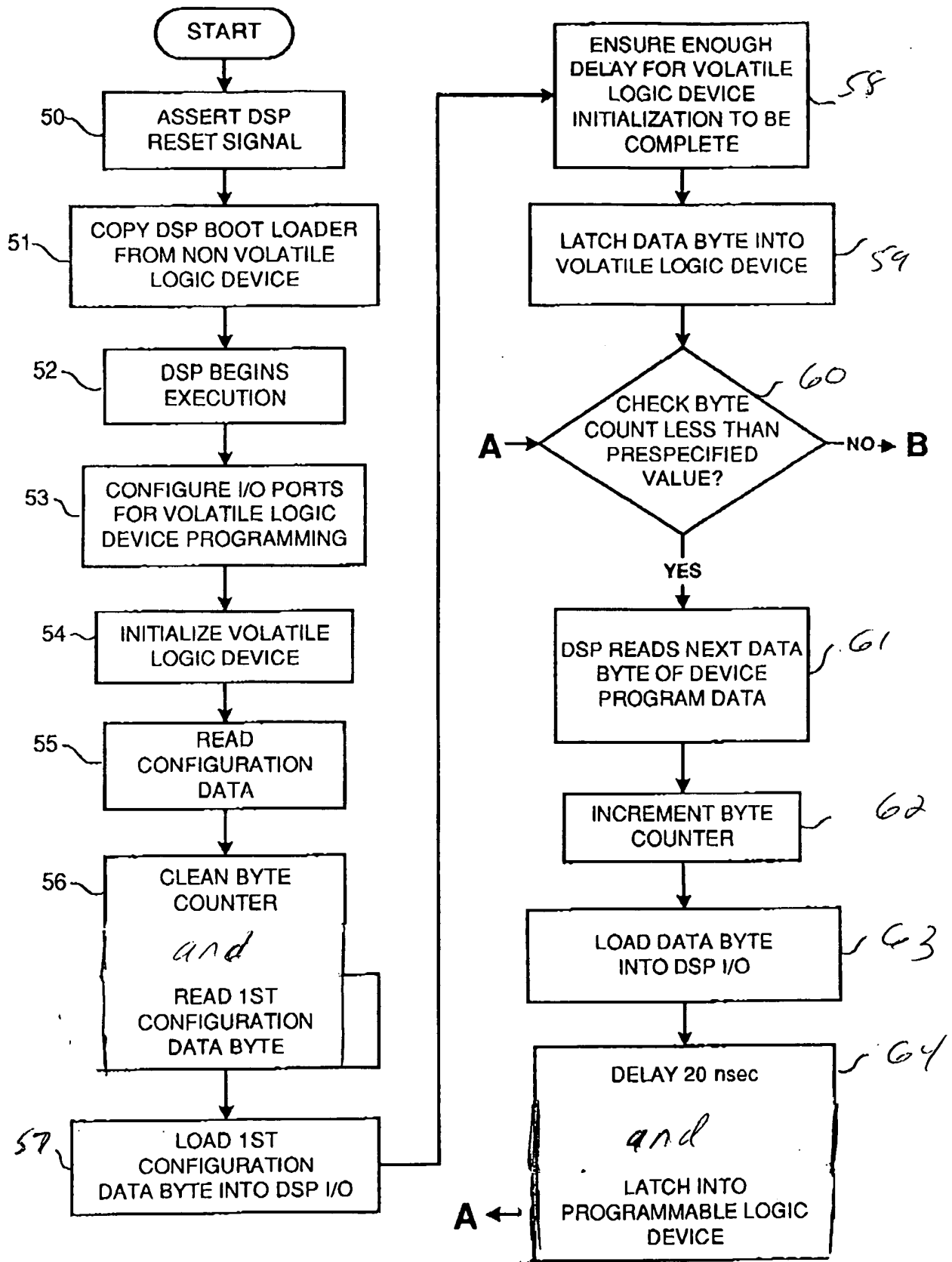


FIG. 6a

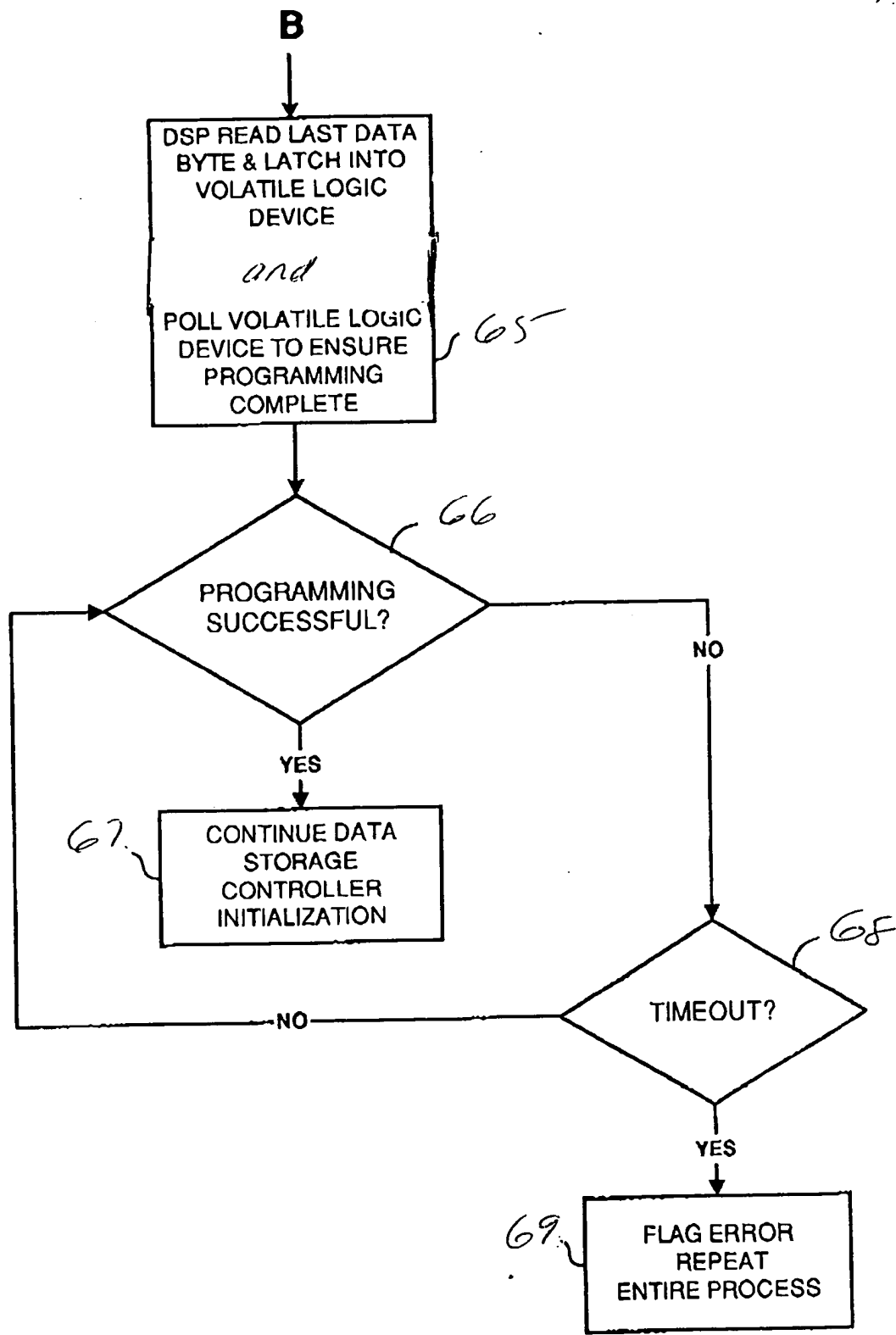


FIG. 6b

FIG. 7a

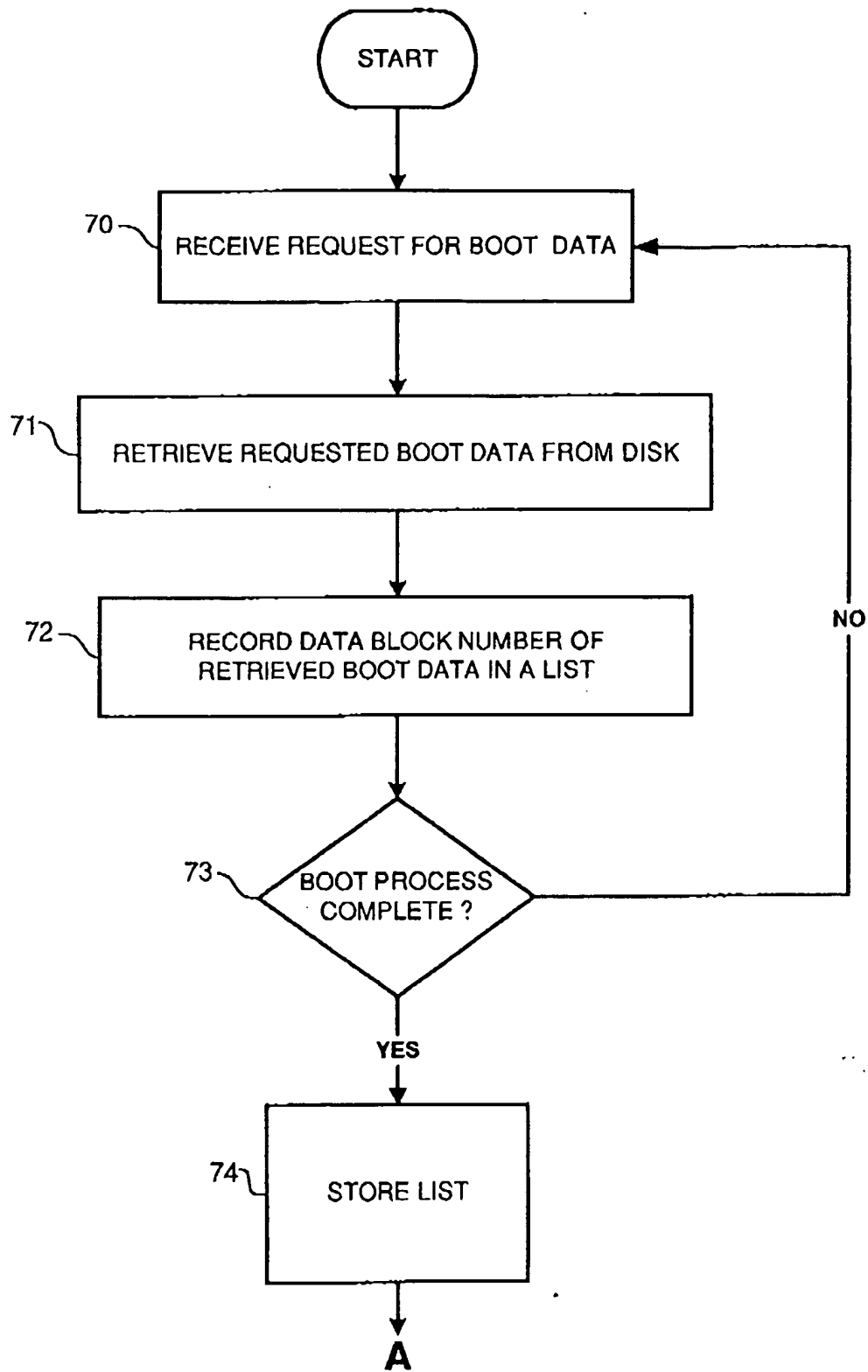


FIG. 7a

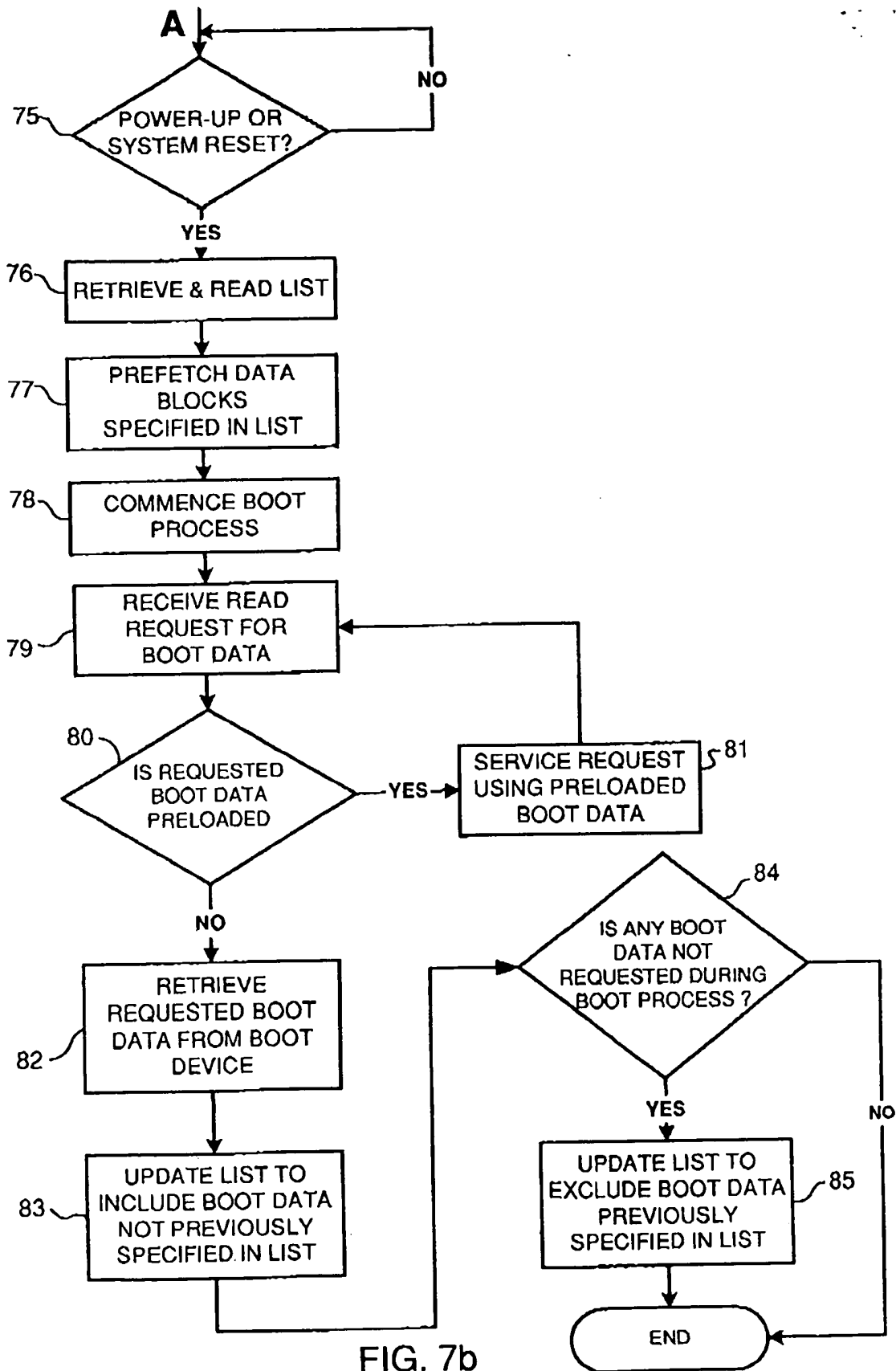


FIG. 7b

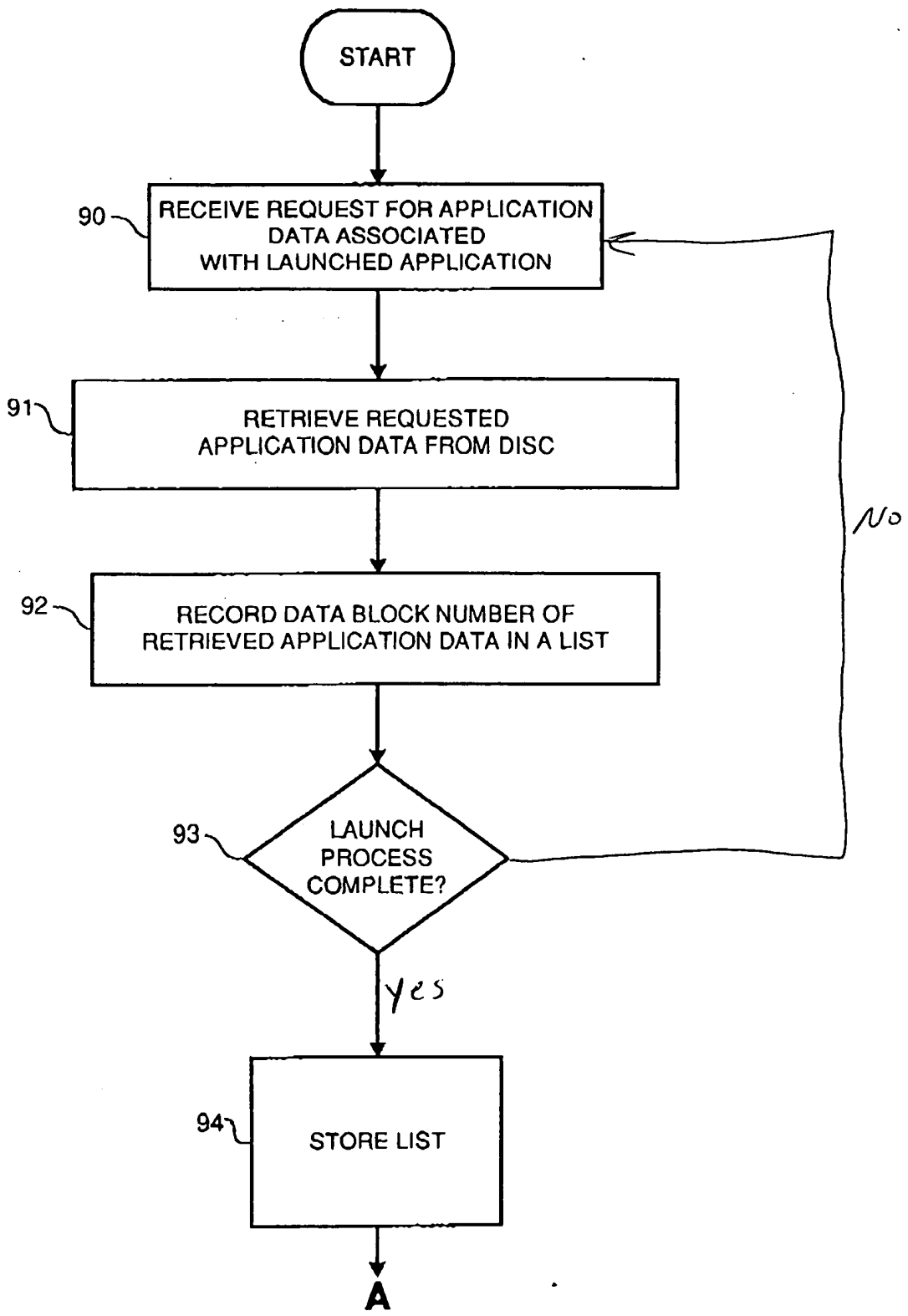


FIG. 8a

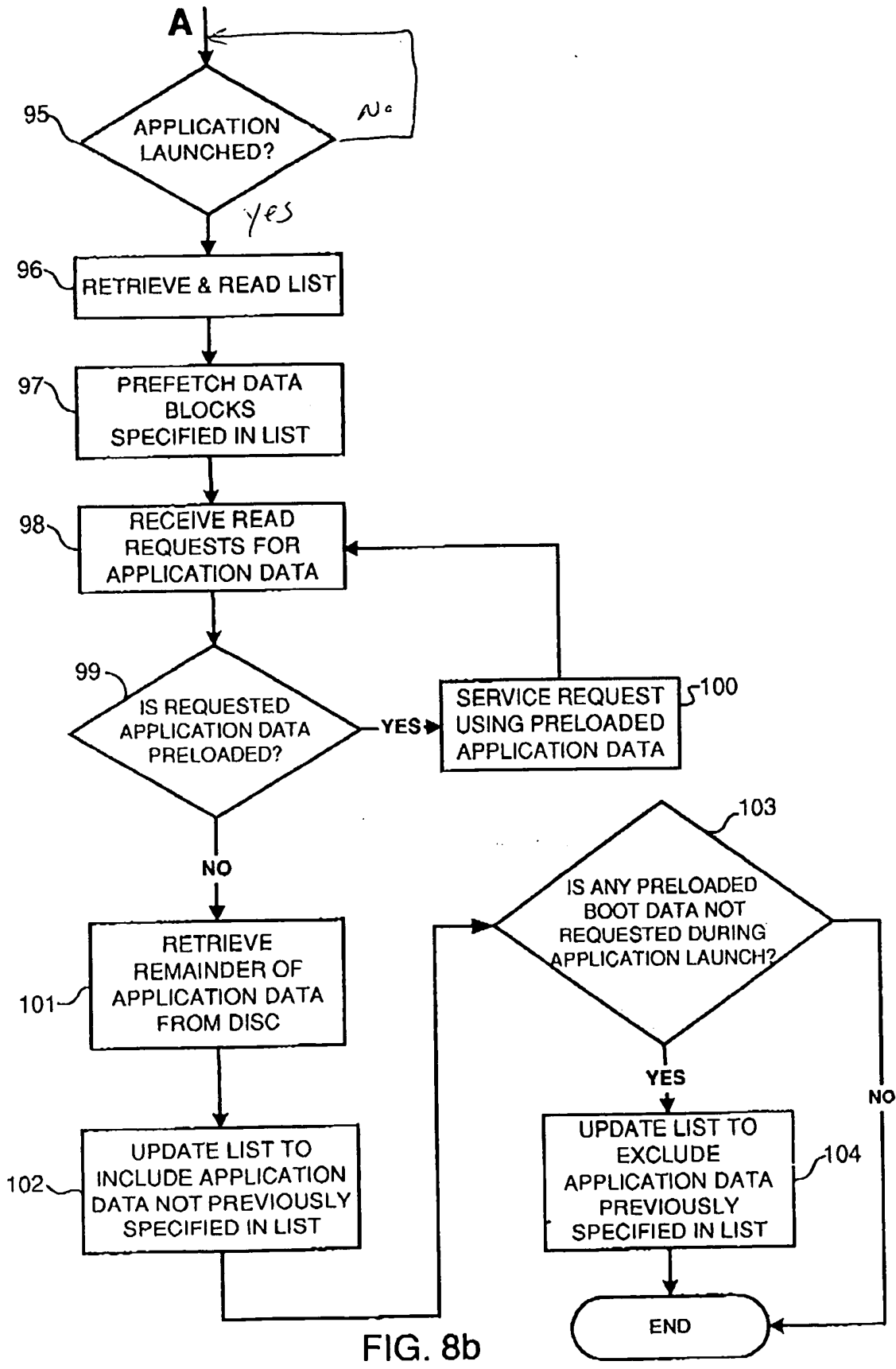


FIG. 8b

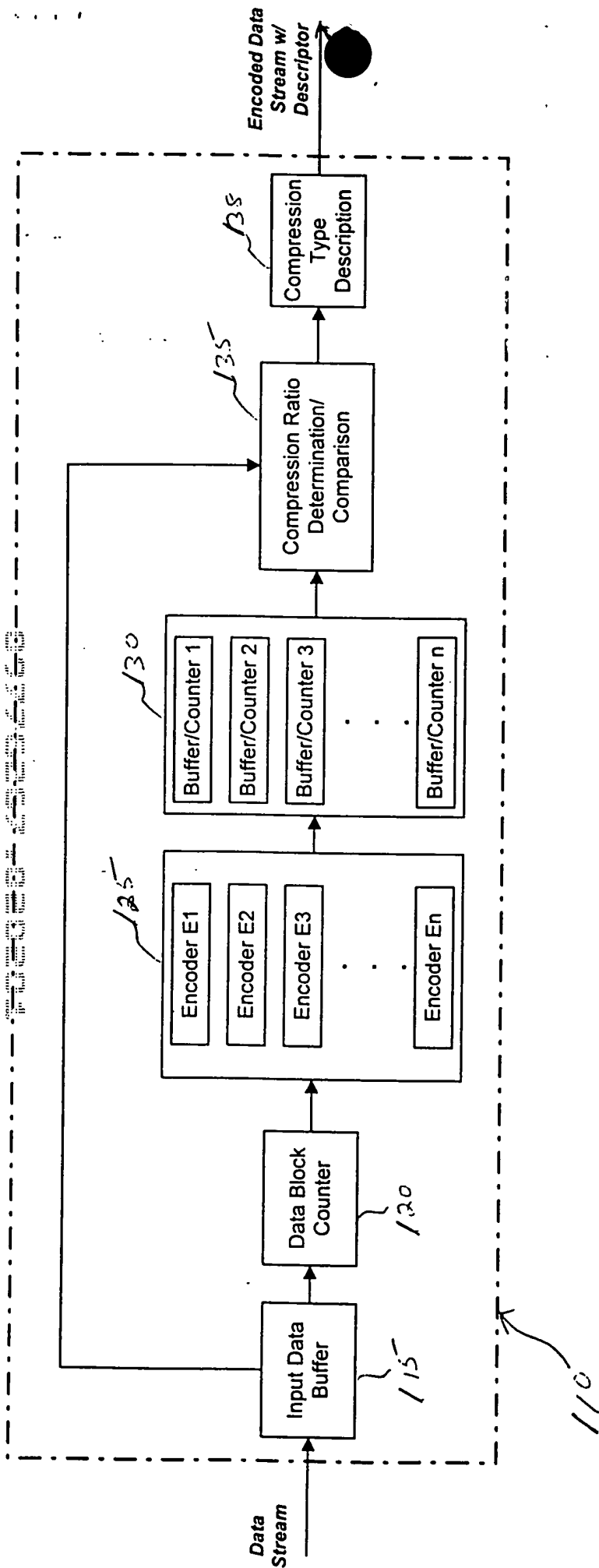


FIGURE 9

FIG. 10

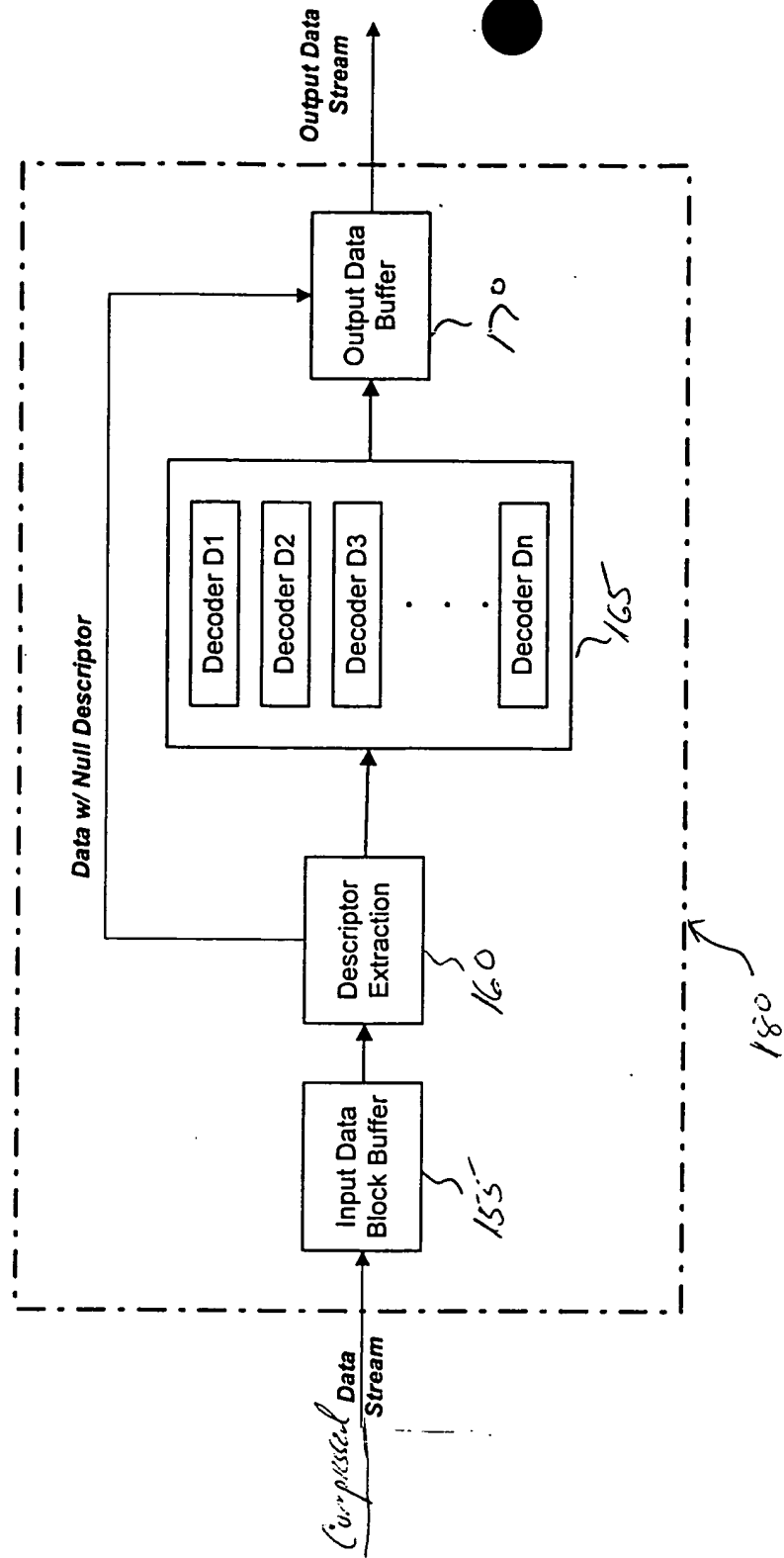


FIG. 10

U.S. Patent Application:

Title: SYSTEMS AND METHODS FOR ACCELERATED LOADING
 OF OPERATING SYSTEMS AND APPLICATION PROGRAMS

Inventor(s): James J. Fallon, 11 Wampus Close, Armonk, New York, 10504;
 John Buck, 362 Christopher Street, Oceanside, New York, 11572;
 Paul F. Pickel, 225 Stewart Avenue, Bethpage, New York, 11714; and
 Stephen J. McErlain, 325 East 17th Street, New York, New York 10003.

Filed: February 2, 2001

Assignee: Realtime Data LLC

20030202

F. Chau & Associates, LLP
1900 Hempstead Turnpike, Suite 501
East Meadow, NY 11554
Tel: (516) 357-0091
Fax: (516) 357-0092

**SYSTEMS AND METHODS FOR ACCELERATED LOADING OF
OPERATING SYSTEMS AND APPLICATION PROGRAMS**

CROSS-REFERENCE TO RELATED APPLICATION

5 This application is based on a United States provisional application Serial No. 60/180,114, filed on February 3, 2000, which is fully incorporated herein by reference.

BACKGROUND

1. Technical Field

10 The present invention relates generally to systems and methods for providing accelerated loading of operating system and application programs upon system boot or application launch and, more particularly, to data storage controllers employing lossless and/or lossy data compression and decompression to provide accelerated loading of operating systems and application programs.

15 **2. Description of the Related Art**

 Modern computers utilize a hierarchy of memory devices. To achieve maximum performance levels, modern processors utilize onboard memory and on board cache to obtain high bandwidth access to both program and data. Limitations in process technologies currently prohibit placing a sufficient quantity of onboard memory for most applications. Thus, in order to offer sufficient memory for the operating system(s), application programs, and user data, computers often use various forms of popular off-processor high speed memory including static random access memory (SRAM), synchronous dynamic random access memory (SDRAM), synchronous burst static ram (SBSRAM). Due to the prohibitive cost of the high-speed random access memory, coupled with their power volatility, a third lower level of the hierarchy exists for non-

20
25

volatile mass storage devices.

Furthermore, mass storage devices offer increased capacity and fairly economical data storage. Mass storage devices (such as a “hard disk”) typically store the operating system of a computer system, as well as applications and data and rapid access to such data is critical to system performance. The data storage and retrieval bandwidth of mass storage devices, however, is typically much less as compared with the bandwidth of other elements of a computing system. Indeed, over the last decade, although computer processor performance has improved by at least a factor of 50, magnetic disk storage performance has only improved by a factor of 5. Consequently, memory storage devices severely limit the performance of consumer, entertainment, office, workstation, servers, and mainframe computers for all disk and memory intensive operations.

The ubiquitous Internet combined with new multimedia applications has put tremendous emphasis on storage volumetric density, storage mass density, storewidth, and power consumption. Specifically, storage density is limited by the number of bits that are encoded in a mass storage device per unit volume. Similarly mass density is defined as storage bits per unit mass. Storewidth is the data rate at which the data may be accessed. There are various ways of categorizing storewidth in terms, several of the more prevalent metrics include sustained continuous storewidth, burst storewidth, and random access storewidth, all typically measured in megabytes/sec. Power consumption is canonically defined in terms of power consumption per bit and may be specified under a number of operating modes including active (while data is being accessed and transmitted) and standby mode. Hence one fairly obvious limitation within the current art is the need for even more volume, mass, and power efficient data storage.

402733220

Another problem with most modern mass storage devices is their inherent unreliability. Many modern mass storage devices utilize rotating assemblies and other types of electromechanical components that possess failure rates one or more orders of magnitude higher than equivalent solid-state devices. RAID systems employ data redundancy distributed across multiple disks to enhance data storage and retrieval reliability. In the simplest case, data may be explicitly repeated on multiple places on a single disk drive, on multiple places on two or more independent disk drives. More complex techniques are also employed that support various trade-offs between data bandwidth and data reliability.

Standard types of RAID systems currently available include RAID Levels 0, 1, and 5. The configuration selected depends on the goals to be achieved. Specifically data reliability, data validation, data storage /retrieval bandwidth, and cost all play a role in defining the appropriate RAID data storage solution. RAID level 0 entails pure data striping across multiple disk drives. This increases data bandwidth at best linearly with the number of disk drives utilized. Data reliability and validation capability are decreased. A failure of a single drive results in a complete loss of all data. Thus another problem with RAID systems is that low cost improved bandwidth requires a significant decrease in reliability.

RAID Level 1 utilizes disk mirroring where data is duplicated on an independent disk subsystem. Validation of data amongst the two independent drives is possible if the data is simultaneously accessed on both disks and subsequently compared. This tends to decrease data bandwidth from even that of a single comparable disk drive. In systems that offer hot swap capability, the failed drive is removed and a replacement drive is

inserted. The data on the failed drive is then copied in the background while the entire system continues to operate in a performance degraded but fully operational mode. Once the data rebuild is complete, normal operation resumes. Hence, another problem with RAID systems is the high cost of increased reliability and associated decrease in performance.

RAID Level 5 employs disk data striping and parity error detection to increase both data bandwidth and reliability simultaneously. A minimum of three disk drives is required for this technique. In the event of a single disk drive failure, that drive may be rebuilt from parity and other data encoded on disk remaining disk drives. In systems that offer hot swap capability, the failed drive is removed and a replacement drive is inserted. The data on the failed drive is then rebuilt in the background while the entire system continues to operate in a performance degraded but fully operational mode. Once the data rebuild is complete, normal operation resumes.

Thus another problem with redundant modern mass storage devices is the degradation of data bandwidth when a storage device fails. Additional problems with bandwidth limitations and reliability similarly occur within the art by all other forms of sequential, pseudo-random, and random access mass storage devices. These and other limitations within the current art are addressed by the present invention.

SUMMARY OF THE INVENTION

The present invention is directed to systems and methods for providing accelerated loading of operating system and application programs upon system boot or application launch and, more particularly, to data storage controllers employing lossless and/or lossy data compression and decompression to provide accelerated loading of

operating systems and application programs.

In one aspect of the present invention, a method for providing accelerated loading of an operating system comprises the steps of: maintaining a list of boot data used for booting a computer system; preloading the boot data upon initialization of the computer system; and servicing requests for boot data from the computer system using the preloaded boot data. The boot data may comprise program code associated with an operating system of the computer system, an application program, and a combination thereof. In a preferred embodiment, the boot data is retrieved from a boot device and stored in a cache memory device.

In another aspect, the method for accelerated loading of an operating system comprises updating the list of boot data during the boot process. The step of updating comprises adding to the list any boot data requested by the computer system not previously stored in the list and/or removing from the list any boot data previously stored in the list and not requested by the computer system.

In yet another aspect, the boot data is stored in a compressed format on the boot device and the preloaded boot data is decompressed prior to transmitting the preloaded boot data to the requesting system.

In another aspect, a method for providing accelerated launching of an application program comprises the steps of: maintaining a list of application data associated with an application program; preloading the application data upon launching the application program; and servicing requests for application data from a computer system using the preloaded application data.

In yet another aspect, a boot device controller for providing accelerated loading of

an operating system of a host system comprises: a digital signal processor (DSP); a programmable logic device, wherein the programmable logic device is programmed by the digital signal processor to (i) instantiate a first interface for operatively interfacing the boot device controller to a boot device and to (ii) instantiate a second interface for
5 operatively interfacing the boot device controller to the host system; and a non-volatile memory device, for storing logic code associated with the DSP, the first interface and the second interface, wherein the logic code comprises instructions executable by the DSP for maintaining a list of boot data used for booting the host system, preloading the boot data upon initialization of the host system, and servicing requests for boot data from the host
10 system using the preloaded boot data. The boot device controller further includes a cache memory device for storing the preloaded boot data.

The present invention is realized due to recent improvements in processing speed, inclusive of dedicated analog and digital hardware circuits, central processing units, (and any hybrid combinations thereof), that, coupled with advanced data compression and
15 decompression algorithms are enabling of ultra high bandwidth data compression and decompression methods that enable improved data storage and retrieval bandwidth

These and other aspects, features and advantages, of the present invention will become apparent from the following detailed description of preferred embodiments that is to be read in connection with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of a data storage controller according to one embodiment of the present invention;

Fig. 2 is a block diagram of a data storage controller according to another embodiment of the present invention;

Fig. 3 is a block diagram of a data storage controller according to another embodiment of the present invention;

5 Fig. 4 is a block diagram of a data storage controller according to another embodiment of the present invention;

Fig. 5 is a block diagram of a data storage controller according to another embodiment of the present invention;

10 Figs. 6a and 6b comprise a flow diagram of a method for initializing a data storage controller according to one aspect of the present invention;

Figs. 7a and 7b comprise a flow diagram of a method for providing accelerated loading of an operating system and/or application programs upon system boot, according to one aspect of the present invention;

15 Figs. 8a and 8b comprise a flow diagram of a method for providing accelerated loading of application programs according to one aspect of the present invention;

Fig. 9 is a diagram of an exemplary data compression system that may be employed in a data storage controller according to the present invention; and

Fig. 10 is a diagram of an exemplary data decompression system that may be employed in a data storage controller according to the present invention.

20

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

In the following description, it is to be understood that system elements having equivalent or similar functionality are designated with the same reference numerals in the Figures. It is to be further understood that the present invention may be implemented in various forms of hardware, software, firmware, or a combination thereof. Preferably, the present invention is implemented on a computer platform including hardware such as one or more central processing units (CPU) or digital signal processors (DSP), a random access memory (RAM), and input/output (I/O) interface(s). The computer platform may also include an operating system, microinstruction code, and dedicated processing hardware utilizing combinatorial logic or finite state machines. The various processes and functions described herein may be either part of the hardware, microinstruction code or application programs that are executed via the operating system, or any combination thereof.

It is to be further understood that, because some of the constituent system components described herein are preferably implemented as software modules, the actual system connections shown in the Figures may differ depending upon the manner in that the systems are programmed. It is to be appreciated that special purpose microprocessors, dedicated hardware, or and combination thereof may be employed to implement the present invention. Given the teachings herein, one of ordinary skill in the related art will be able to contemplate these and similar implementations or configurations of the present invention.

I. System Architectures

The present invention is directed to data storage controllers that provide increased

data storage/retrieval rates that are not otherwise achievable using conventional disk controller systems and protocols to store/retrieve data to/from mass storage devices. The concept of “accelerated” data storage and retrieval was introduced in copending U.S. Patent Application Serial No. 09/266,394, filed March 11, 1999, entitled “System and Methods For Accelerated Data Storage and Retrieval” and copending U.S. Patent Application Serial No. 09/481,243, filed January 11, 2000, entitled “System and Methods For Accelerated Data Storage and Retrieval,” both of which are commonly assigned and incorporated herein by reference. In general, as described in the above-incorporated applications, “accelerated” data storage comprises receiving a digital data stream at a data transmission rate which is greater than the data storage rate of a target storage device, compressing the input stream at a compression rate that increases the effective data storage rate of the target storage device and storing the compressed data in the target storage device. For instance, assume that a mass storage device (such as a hard disk) has a data storage rate of 20 megabytes per second. If a storage controller for the mass storage device is capable of compressing an input data stream with an average compression rate of 3:1, then data can be stored in the mass storage device at a rate of 60 megabytes per second, thereby effectively increasing the storage bandwidth (“storewidth”) of the mass storage device by a factor of three. Similarly, accelerated data retrieval comprises retrieving a compressed digital data stream from a target storage device at the rate equal to, e.g., the data access rate of the target storage device and then decompressing the compressed data at a rate that increases the effective data access rate of the target storage device. Advantageously, accelerated data storage/retrieval mitigates the traditional bottleneck associated with, e.g., local and network disk accesses.

Referring now to Fig. 1, a high-level block diagram illustrates a data storage controller 10 according to one embodiment of the present invention. The data storage controller 10 comprises a data compression engine 12 for compressing/decompressing data (preferably in real-time or psuedo real-time) stored/retrieved from a hard disk 11 (or
5 any other type of mass storage device) to provide accelerated data storage/retrieval. The DCE 12 preferably employs the data compression/decompression techniques disclosed in U.S. Serial No. 09/210,491 entitled "Content Independent Data Compression Method and System," filed on December 11, 1998, which is commonly assigned and which is fully incorporated herein by reference. It is to be appreciated that the compression and
10 decompression systems and methods disclosed in U.S. Serial No. 09/210,491 are suitable for compressing and decompressing data at rates, which provide accelerated data storage and retrieval. A detailed discussion of a preferred "content independent" data compression process will be provided below.

The data storage controller 10 further comprises a cache 13, a disk interface (or
15 disk controller) 14 and a bus interface 15. The storage controller 10 is operatively connected to the hard disk 12 via the disk controller 14 and operatively connected to an expansion bus (or main bus) 16 of a computer system via the bus interface 15. The disk interface 14 may employ a known disk interface standard such as UltraDMA, SCSI, Serial Storage Architecture, FibreChannel or any other interface that provides suitable
20 disk access data rates. In addition, the storage controller 10 preferably utilizes the American National Standard for Information Systems (ANSI) AT Attachment Interface (ATA/ATAPI-4) to connect the data storage controller 10 to the hard disk 12. As is known in the art, this standard defines the connectors and cables for the physical

interconnects between the data storage controller and the storage devices, along with the electrical and logical characteristics of the interconnecting signals.

Further, the bus interface 15 may employ a known standard such as the PCI (Peripheral Component Interconnect) bus interface for interfacing with a computer system. The use of industry standard interfaces and protocols is preferable, as it allows the storage controller 10 to be backwards compatible and seamlessly integrated with current systems. However in new designs the present invention may be utilize any suitable computer interface or combination thereof.

It is to be understood that although Fig. 1 illustrates a hard disk 12, the storage controller 10 may be employed with any form of memory device including all forms of sequential, pseudo-random, and random access storage devices. Storage devices as known within the current art include all forms of random access memory, magnetic and optical tape, magnetic and optical disks, along with various other forms of solid-state mass storage devices. The current invention applies to all forms and manners of memory devices including, but not limited to, storage devices utilizing magnetic, optical, and chemical techniques, or any combination thereof. In addition, the cache 13 may comprise volatile or non-volatile memory, or any combination thereof. Preferably, the cache 13 is implemented in SDRAM (static dynamic random access memory).

The system of Fig. 1 generally operates as follows. When data is read from disk by the host computer, data flows from the disk 11 through the data storage controller 10 to the host computer. Data is stored in one of several proprietary compression formats on the disk 11 (e.g., "content independent" data compression). Data blocks are pre-specified in length, comprised of single or multiple sectors, and are typically handled in fractional

or whole equivalents of tracks, e.g. ½ track, whole track, multiple tracks, etc. To read disk data, a DMA transfer is setup from the disk interface 14 to the onboard cache memory 13. The disk interface 14 comprises integral DMA control to allow transfer of data from the disk 11 directly to the onboard cache 13 without intervention by the DCE 12. It should be noted that the DCE 12 acts as a system level controller and sets-up specific registers within both the disk interface 14 and bus interface 15 to facilitate DMA transfers to and from the cache memory 13. To initiate a transfer from the disk 11 to the cache 13, the DMA transfer is setup via specifying the appropriate command (read disk), the source address (disk logical block number), amount of data to be transferred (number of disk logical blocks), and destination address within the onboard cache memory 13. Then, a disk data interrupt signal (“DISKINT#”) is cleared (if previously set and not cleared) and the command is initiated by writing to the appropriate address space. Once data has been read from disk 11 and placed into onboard cache memory 13, the DISKINT# interrupt is asserted notifying the DCE 12 that requested data is now available in the cache memory 13. Data is then read by the DMA controller within the DCE 12 and placed into local memory for subsequent decompression. The decompressed data is then DMA transferred from the local memory of the DCE 12 back to the cache memory 13. Finally, data is DMA transferred via the bus interface controller 15 from the cache memory 13 to the bus 16. It is to be understood that in the read mode, the data storage controller acts as a bus master. A bus DMA transfer is then setup via specifying the appropriate command (write to host computer), the source address within the cache memory 13, the quantity of data words to be transferred (transfers are preferably in 4 byte increments), and the destination address on the host computer. When a bus 16 read or

write transaction has completed, the appropriate interrupt signals (respectively referred to as PCIRDINT# and PCIWRINT#) are asserted to the DCE 12. Either of these interrupts are cleared by a corresponding interrupt service routines through a read or write to the appropriate address of the DCE 12.

5 Similarly, when data is written to the disk 11 from the host computer, data flows from the host computer through the data storage controller 10 and onto disk 11. Data is normally received from the host computer in uncompressed (raw) format and is compressed by the DCE 12 and stored on the disk 11. Data blocks from the host are pre-specified in length and are typically handled in blocks that are a fixed multiplier higher than fractional or whole equivalents of tracks, e.g. ½ track, whole track, multiple tracks, etc. This multiplier is preferably derived from the expected average compression ratio that is selected when the disk is formatted with the virtual file management system. To read host computer data, a bus DMA transfer is setup from the host bus 16 to the onboard cache memory 13. The bus interface controller 15 comprises integral DMA control that allows large block transfers from the host computer directly to the onboard cache 13 without intervention by the DCE 12. The bus interface controller 15 acts as a host computer Bus Master when executing such transfer. Once data has been read from the host and placed into onboard cache memory 13, the data is read by the onboard DMA controller (residing on the DCE 12) and placed into local memory for subsequent compression. The compressed data is then DMA transferred from the local memory of the DCE 12 back to the cache memory 13. Finally, data is DMA transferred via the disk controller 14 from the cache 13 to the disk 11.

As discussed in greater detail below, upon host computer power-up or external

user reset, the data storage controller 10 initializes the onboard interfaces 14, 5 prior to release of the external host bus 16 from reset. The processor of the host computer then requests initial data from the disk 11 to facilitate the computer's boot-up sequence. The host computer requests disk data over the Bus 16 via a command packet issued from the host computer. Command packets are preferably eight words long (in a preferred embodiment, each word comprises 32 bits). Commands are written from the host computer to the data storage controller 10 with the host computer as the Bus Master and the data storage controller 10 as the slave. The data storage controller 10 includes at least one Base Address Register (BAR) for decoding the address of a command queue of the data storage controller 10. The command queue resides within the cache 13 or within onboard memory of the DCE 12.

When a command is received from the host computer, an interrupt (referred to herein as PCICMDINT#) is generated to the DCE processor. The eight-word command is read by the DCE 12 and placed into the command queue. Because the commands occupy a very small amount of memory, the location of the command queue is at the discretion of software and the associated system level performance considerations. Commands may be moved from the bus interface 16 to the command queue by wither explicit reads and writes by the DCE processor or, as explained below, by utilizing programmed DMA from an Enhanced DMA Controller (EDMA) residing on the DCE 12. This second technique may better facilitate system throughput by allowing the EDMA to automatically load commands while the highly pipelined data compression and decompression processing in the DCE is executed fully undisturbed.

The DCE 12, disk interface 14 and bus interface 15 commonly share the cache 13.

As explained in detail below, the storage controller 10 preferably provides maximum system bandwidth by allowing simultaneous data transfers between the disk 12 and cache 13, the DCE 12 and the cache 13, and the expansion bus 16 and the cache 13. This is realized by employing an integral DMA (direct memory access) protocol that allows the DCE 12, disk interface 14 and bus interface 15 to transfer data without interrupting or interfering with other ongoing processes. In particular, as explained in detail below, an integral bandwidth allocation controller (or arbitrator) is preferably employed to allow the DCE 12, disk controller 14, and bus interface 15 to access the onboard cache with a bandwidth proportional to the overall bandwidth of the respective interface or processing element. The bandwidth arbitration occurs transparently and does not introduce latency in memory accesses. Bandwidth division is preferably performed with a high degree of granularity to minimize the size of requisite onboard buffers to synchronize data from the disk interface 14 and bus interface 15.

It is to be appreciated that the implementation of a storage controller according to the present invention significantly accelerates the performance of a computer system and significantly increases hard disk data storage capacity. For instance, depending on the compression rate, for personal computers running standard Microsoft Windows® based business application software, the storage controller provides: (1) an increase of n:1 in disk storage capacity (for example, assuming a compression ration of 3:1, a 20 gigabyte hard drive effectively becomes a 60 gigabyte hard drive) (2) a significant decrease in the computer boot-up time (turn-on and operating system load) and the time for loading application software and (3) User data storage and retrieval is increased by a factor of n:1.

Referring now to Fig. 2, a block diagram illustrates a data storage controller 20

according to another embodiment of the present invention. More specifically, Fig. 2 illustrates a PCB (printed circuit board) implementation of the data storage controller 10 of Fig. 1. The storage controller 20 comprises a DSP (digital signal processor) 21 (or any other micro-processor device) that implements the DCE 12 of Fig. 1. The storage controller 21 further comprises at least one programmable logic device 22 (or volatile logic device). The programmable logic device 22 preferably implements the logic (program code) for instantiating and driving both the disk interface 14 and the bus interface 15 and for providing full DMA capability for the disk and bus interfaces 14, 15. Further, as explained in detail below, upon host computer power-up and/or assertion of a system-level "reset" (e.g., PCI Bus reset), the DSP 21 initializes and programs the programmable logic device 22 before of the completion of initialization of the host computer. This advantageously allows the data storage controller 20 to be ready to accept and process commands from the host computer (via the bus 16) and retrieve boot data from the disk (assuming the data storage controller 20 is implemented as the boot device and the hard disk stores the boot data (e.g., operating system, etc.)).

The data storage controller 20 further comprises a plurality of memory devices including a RAM (random access memory) device 23 and a ROM (read only memory) device 24 (or FLASH memory or other types of non-volatile memory). The RAM device 23 is utilized as on-board cache and is preferably implemented as SDRAM (preferably, 32 megabytes minimum). The ROM device 24 is utilized for non-volatile storage of logic code associated with the DSP 21 and configuration data used by the DSP 21 to program the programmable logic device 22. The ROM device 24 preferably comprises a one time (erasable) programmable memory (OTP-EPROM) device.

The DSP 21 is operatively connected to the memory devices 23, 24 and the programmable logic device 22 via a local bus 25. The DSP 21 is also operatively connected to the programmable logic device 22 via an independent control bus 26. The programmable logic device 22 provides data flow control between the DSP 21 and the host computer system attached to the bus 16, as well as data flow control between the DSP 21 and the storage device. A plurality of external I/O ports 27 are included for data transmission and/or loading of one programmable logic devices. Preferably, the disk interface 14 driven by the programmable logic device 22 supports a plurality of hard drives.

The storage controller 20 further comprises computer reset and power up circuitry 28 (or "boot configuration circuit") for controlling initialization (either cold or warm boots) of the host computer system and storage controller 20. A preferred boot configuration circuit and preferred computer initialization systems and protocols are described in U.S. Patent Application Serial No. _____ (Attorney Docket No. 8011-10), filed concurrently herewith (Express Mail Label No. EL679454245US), which is commonly assigned and incorporated herein by reference. Preferably, the boot configuration circuit 28 is employed for controlling the initializing and programming the programmable logic device 22 during configuration of the host computer system (i.e., while the CPU of the host is held in reset). The boot configuration circuit 28 ensures that the programmable logic device 22 (and possibly other volatile or partially volatile logic devices) is initialized and programmed before the bus 16 (such as a PCI bus) is fully reset.

In particular, when power is first applied to the boot configuration circuit 28, the

boot configuration circuit 28 generates a control signal to reset the local system (e.g., storage controller 20) devices such as a DSP, memory, and I/O interfaces. Once the local system is powered-up and reset, the controlling device (such as the DSP 21) will then proceed to automatically determine the system environment and configure the local system to work within that environment. By way of example, the DSP 21 of the disk storage controller 20 would sense that the data storage controller 20 is on a PCI computer bus (expansion bus) and has attached to it a hard disk on an IDE interface. The DSP 21 would then load the appropriate PCI and IDE interfaces into the programmable logic device 22 prior to completion of the host system reset. It is to be appreciated that this can be done for all computer busses and boot device interfaces including: PCI, NuBus, ISA, Fiber Channel, SCSI, Ethernet, DSL, ADSL, IDE, DMA, Ultra DMA, and SONET. Once the programmable logic device 22 is configured for its environment, the boot device controller is reset and ready to accept commands over the computer/expansion bus 16. Details of the boot process using a boot device comprising a programmable logic device will be provided below.

It is to be understood that the data storage controller 20 may be utilized as a controller for transmitting data (compressed or uncompressed) to and from remote locations over the DSP I/O ports 27 or system bus 16, for example. Indeed, the I/O ports 27 of the DSP 21 may be used for transmitting data (compressed or uncompressed) that is either retrieved from the disk 11 or received from the host system via the bus 16, to remote locations for processing and/or storage. Indeed, the I/O ports may be operatively connected to other data storage controllers or to a network communication channels. Likewise, the data storage controller 20 may receive data (compressed or uncompressed)

over the I/O ports 27 of the DSP 21 from remote systems that are connected to the I/O ports 27 of the DSP, for local processing by the data storage controller 20. For instance, a remote system may remotely access the data storage controller (via the I/O ports of the DSP or system bus 16) to utilize the data compression, in which case the data storage controller would transmit the compressed data back to the system that requested compression.

The DSP 21 may comprise any suitable commercially available DSP or processor. Preferably, the data storage controller 20 utilizes a DSP from Texas Instruments' 320 series, C62x family, of DSPs (such as TMS320C6211GFN-150), although any other DSP or processor comprising a similar architecture and providing similar functionalities may be employed. The preferred DSP is capable of up to 1.2 billion instructions per second. Additional features of the preferred DSP include a highly parallel eight processor single cycle instruction execution, onboard 4K byte L1P Program Cache, 4K L1D Data Cache, and 64K byte Unified L2 Program/Data Cache. The preferred DSP further comprises a 32 bit External Memory Interface (EMIF) that provides for a glueless interface to the RAM 23 and the non-volatile memory 24 (ROM). The DSP further comprises two multi-channel buffered serial ports (McBSPs) and two 32 bit general purpose timers. Preferably, the storage controller disables the I/O capability of these devices and utilizes the I/O ports of the DSP as general purpose I/O for both programming the programmable logic device 22 using a strobed eight bit interface and signaling via a Light Emitting Diode (LED). Ancillary DSP features include a 16 bit Host Port Interface and full JTAG emulation capability for development support.

The programmable logic device 22 may comprise any form of volatile or non-

volatile memory. Preferably, the programmable logic device 22 comprises a dynamically reprogrammable FPGA (field programmable gate array) such as the commercially available Xilinx Spartan Series XCS40XL-PQ240-5 FPGA. As discussed in detail herein, the FPGA instantiates and drives the disk and bus interfaces 14, 15.

5 The non-volatile memory device 24 preferably comprises a 128 Kbyte M27W101-80K one time (erasable) programmable read only memory, although other suitable non-volatile storage devices may be employed. The non-volatile memory device 24 is decoded at a designated memory space in the DSP 21. The non-volatile memory device 24 stores the logic for the DSP 21 and configuration data for the programmable logic
10 device 22. More specifically, in a preferred embodiment, the lower 80 Kbytes of the non-volatile memory device 24 are utilized for storing DSP program code, wherein the first 1k bytes are utilized for the DSP's boot loader. Upon reset of the DSP 21 (via boot configuration circuit 28), the first 1K of memory of the non-volatile memory device 24 is copied into an internal RAM of the DSP 21 by e.g., the DSP's Enhanced DMA Controller
15 (EDMA). Although the boot process begins when the CPU of the host system is released from external reset, the transfer of the boot code into the DSP and the DSP's initialization of the programmable logic device actually occurs while the CPU of the host system is held in reset. After completion of the 1K block transfer, the DSP executes the boot loader code and continues thereafter with executing the remainder of the code in non-volatile
20 memory device to program the programmable logic device 22.

More specifically, in a preferred embodiment, the upper 48K bytes of the non-volatile memory device 24 are utilized for storing configuration data associated with the programmable logic device 22. If the data storage controller 20 is employed as the

primary boot storage device for the host computer, the logic for instantiating and driving the disk and bus interfaces 14, 15 should be stored on the data storage controller 20 (although such code may be stored in remotely accessible memory locations) and loaded prior to release of the host system bus 16 from “reset”. For instance, revision 2.2 of the PCI Local Bus specification calls for a typical delay of 100msec from power-stable before release of PCI Reset. In practice this delay is currently 200msec although this varies amongst computer manufacturers. A detailed discussion of the power-on sequencing and boot operation of the data storage controller 20 will be provided below.

Fig. 3 illustrates another embodiment of a data storage controller 30 wherein the data storage controller 35 is embedded within the motherboard of the host computer system. This architecture provides the same functionality as the system of Fig. 2, and also adds the cost advantage of being embedded on the host motherboard. The system comprises additional RAM and ROM memory devices 23a, 24a, operatively connected to the DSP 21 via a local bus 25a.

Fig. 4 illustrates another embodiment of a data storage controller. The data storage controller 40 comprises a PCB implementation that is capable of supporting RAID levels 0,1 and 5. This architecture is similar to those of Fig. 1 and 2, except that a plurality of programmable logic devices 22, 22a are utilized. The programmable logic device 22 is dedicated to controlling the bus interface 15. The programmable logic device 22a is dedicated to controlling a plurality of disk interfaces 14, preferably three interfaces. Each disk interface 14 can connect up to two drives. The DSP in conjunction with the programmable logic device 22a can operate at RAID level 0, 1 or 5. At RAID level 0, which is disk striping, two interfaces are required. This is also true for RAID

level 1, which is disk mirroring. At RAID level 5, all three interfaces are required.

Fig. 5 illustrates another embodiment of a data storage controller according to the present invention. The data storage controller 45 provides the same functionality as that of Figure 4, and has the cost advantage of being embedded within the computer system motherboard.

II. Initializing A Programmable Logic Device

As discussed above with reference to Fig. 2, for example, the data storage controller 20 preferably employs an onboard Texas Instruments TMS320C6211 Digital Signal Processor (DSP) to program the onboard Xilinx Spartan Series XCS40XL FPGA upon power-up or system level PCI reset. The onboard boot configuration circuit 28 ensures that from system power-up and/or the assertion of a bus reset (e.g., PCI reset), the DSP 21 is allotted a predetermined amount of time (preferably a minimum of 10msec) to boot the DSP 21 and load the programmable logic device 22. Because of a potential race condition between either the host computer power-up or assertion of PCI Bus reset and configuration of the programmable logic device 20 (which is used for controlling the boot device and accepting PCI Commands), an "Express Mode" programming mode for configuring the SpartanXL family XCS40XL device is preferably employed. The XCS40XL is factory set to byte-wide Express-Mode programming by setting both the M1/M0 bits of the XCS40XL to 0x0. Further, to accommodate express mode programming of the programmable logic device 22, the DSP 21 is programmed to utilize its serial ports reconfigured as general purpose I/O. However, after the logic device 22 is programmed, the DSP 21 may then reconfigure its serial ports for use with other devices. Advantageously, using the same DSP ports for multiple purposes affords greater

flexibility while minimizing hardware resources and thus reducing product cost.

The volatile nature of the logic device 22 effectively affords the ability to have an unlimited number of hardware interfaces. Any number of programs for execution by the programmable logic device 22 can be kept in an accessible memory location (EPROM, 5 hard disk, or other storage device). Each program can contain new disk interfaces, interface modes or subsets thereof. When necessary, the DSP 21 can clear the interface currently residing in the logic device 22 and reprogram it with a new interface. This feature allows the data storage controller 20 to have compatibility with a large number of interfaces while minimizing hardware resources and thus reducing product cost.

10 A preferred protocol for programming the programmable logic device can be summarized in the following steps: (1) Clearing the configuration memory; (2) Initialization; (3) Configuration; and (4) Start-Up. When either of three events occur: the host computer is first powered-up or a power failure and subsequent recovery occurs (cold boot), or a front panel computer reset is initiated (warm boot), the host computer 15 asserts RST# (reset) on the PCI Bus. As noted above, the data storage controller 20 preferably comprises a boot configuration circuit 28 that senses initial host computer power turn-on and/or assertion of a PCI Bus Reset ("PCI RST#"). It is important to note that assuming the data storage controller 20 is utilized in the computer boot-up sequence, it should be available exactly 5 clock cycles after the PCI RST# is deasserted, as per PCI 20 Bus Specification Revision 2.2. While exact timings vary from computer to computer, the typical PCI bus reset is asserted for approximately 200msec from initial power turn-on.

In general, PCI RST# is asserted as soon as the computer's power exceeds a nominal threshold of about 1 volt (although this varies) and remains asserted for 200msec

thereafter. Power failure detection of the 5volt or 3.3 volt bus typically resets the entire computer as if it is an initial power-up event (i.e., cold boot). Front panel resets (warm boots) are more troublesome and are derived from a debounced push-button switch input.

Typical front panel reset times are a minimum of 20msec, although again the only
5 governing specification limit is 1msec reset pulse width.

As discussed in detail below, it may not be necessary to reload the programmable logic device 22 each time the DSP is reset. The boot configuration circuit 20 preferably comprises a state machine output signal that is readable by the DSP 21 to ascertain the type of boot process requested. For example, with a front-panel reset (warm boot), the
10 power remains stable on the PCI Bus, thus the programmable logic device 22 should not require reloading.

Referring now to Fig. 6, a flow diagram illustrates a method for initializing the programmable logic device 22 according to one aspect of the invention. In the following discussion, it is assumed that the programmable logic device 22 is always reloaded,
15 regardless of the type of boot process. Initially, in Fig. 6a, the DSP 21 is reset by asserting a DSP reset signal (step 50). Preferably, the DSP reset signal is generated by the boot circuit configuration circuit 28 (as described in the above-incorporated U.S. Serial No. _____ (Attorney Docket No. 8011-10). While the DSP reset signal is asserted (e.g., active low), the DSP is held in reset and is initialized to a prescribed state.

20 Upon deassertion of the DSP Reset signal, the logic code for the DSP (referred to as the "boot loader") is copied from the non-volatile logic device 24 into memory residing in the DSP 21 (step 51). This allows the DSP to execute the initialization of the programmable logic device 22. In a preferred embodiment, the lower 1K bytes of EPROM memory is

copied to the first 1k bytes of DSP's low memory (0x0000 0000 through 0x0000 03FF).

As noted above, the memory mapping of the DSP 21 maps the CE1 memory space

located at 0x9000 0000 through 0x9001 FFFF with the OTP EPROM. In a preferred

embodiment using the Texas Instrument DSP TMS320c6211GFN-150, this ROM boot

5 process is executed by the EDMA controller of the DSP. It is to be understood, however,

that the EDMA controller may be instantiated in the programmable logic device (Xilinx),

or shared between the DSP and programmable logic device.

After the logic is loaded in the DSP 21, the DSP 21 begins execution out of the

lower 1K bytes of memory (step 52). In a preferred embodiment, the DSP 21 initializes

10 with at least the functionality to read EPROM Memory (CE1) space. Then, as described

above, the DSP preferably configures its serial ports as general purpose I/O (step 53).

Next, the DSP 21 will initialize the programmable logic device 22 using one or more

suitable control signals. (step 54). After initialization, the DSP 21 begins reading the

configuration data of the programmable logic device 22 from the non-volatile memory 24

15 (step 55). This process begins with clearing a Data Byte Counter and then reading the

first data byte beginning at a prespecified memory location in the non-volatile memory 24

(step 56). Then, the first output byte is loaded into the DSP's I/O locations with LSB at

D0 and MSB at D7 (step 57). Before the first byte is loaded to the logic device 22, a

prespecified time delay (e.g., 5usec) is provided to ensure that the logic device 22 has

20 been initialized (step 58). In particular, this time delay should be of a duration at least

equal to the internal setup time of the programmable logic device 22 from completion of

initialization. Once this time delay has expired, the first data byte in the I/O bus 26 of the

DSP 21 is latched into the programmable logic device 22 (step 59).

Next, a determination is made as to whether the Data Byte Counter is less than a prespecified value (step 60). If the Data Byte Counter is less than the prespecified value (affirmative determination in step 60), the next successive data byte for the programmable logic device 22 is read from the non-volatile memory 24 (step 61) and the Data Byte Counter is incremented (step 62).

Next, the read data byte is loaded into the I/O of the DSP (step 63). A time delay of, e.g., 20 nsec is allowed to expire before the data byte is latched to the programmable logic device to ensure that a minimum data set-up time to the programmable logic device 21 is observed (step 64) and the process is repeated (return to step 60). It is to be appreciated that steps 60-64 may be performed while the current data byte is being latched to the programmable logic device. This provides "pipeline" programming of the logic device 22 and minimizes programming duration.

When the Data Byte Counter is not less than the prespecified count value (negative determination in step 60), as shown in Fig. 6b, the last data byte is read from the non-volatile memory and latched to the programmable logic device 22, and the DSP 21 will then poll a control signal generated by the programmable logic device 22 to ensure that the programming of the logic device 22 is successful (step 65). If programming is complete (affirmative determination in step 66), the process continues with the remainder of the data storage controller initialization (step 67). Otherwise, a timeout occurs (step 68) and upon expiration of the timeout, an error signal is provided and the programming process is repeated (step 69)..

III. Data Storage and Retrieval Protocols

A detailed discussion of operational modes of a data storage controller will now

be provided with reference to the embodiment of Fig 2 (although it is to be understood that the following discussion is applicable to all the above-described embodiments). The data storage controller 20 utilizes a plurality of commands to implement the data storage, retrieval, and disk maintenance functions described herein. Each command preferably
 5 comprises eight thirty-two bit data words stored and transmitted in little endian format. The commands include: Read Disk Data; Write Disk Data; and Copy Disk Data, for example. For example, a preferred format for the “Read Disk Data” command is:

31	16	15	8	7	0	
Command Packet Number 0000h to FFFFh		Command Type 00h		Command Parameters (00h)		00h
Starting Block Address (Least Significant Word)						04h
Starting Block Address (Most Significant Word)						08h
Number of Blocks (Least Significant Word)						0Ch
Number of Blocks (Most Significant Word)						10h
Destination Address (Least Significant Word)						14h
Destination Address (Most Significant Word)						18h
Checksum			Reserved			1Ch

The host computer commands the data storage controller 20 over the PCI Bus 16,
 10 for example. Upon computer power-up or reset, the host computer issues a PCI Bus Reset with a minimum pulse width of 100msec (in accordance with PCI Bus Specification Revision 2.2). Upon completion of the PCI Bus reset, the data storage controller 20 is fully initialized and waiting for completion of the PCI configuration cycle. Upon completion of the PCI configuration cycles, the data storage controller will
 15 wait in an idle state for the first disk command.

During operation, the host operating system may issue a command to the data storage controller 20 to store, retrieve, or copy specific logical data blocks. Each command is transmitted over the PCI Bus 16 at the Address assigned to the Base Address Register (BAR) of the data storage controller 20.

5 The commands issued by the host system to the data storage controller and the data transmitted to and from the data storage controller are preferably communicated via a 32 bit, 33MHz, PCI Data Bus. As noted above, the PCI Interface is preferably housed within the onboard Xilinx Spartan XCS40XL-5 40,000 field programmable gate array which instantiates a PCI 32, 32 Bit, 33MHz PCI Bus Interface (as per PCI Bus Revision
10 2.2).

The PCI Bus interface operates in Slave Mode when receiving commands and as a Bus Master when reading or writing data. The source and destination for all data is specified within each command packet. When setting up data transfers, the Enhanced Direct Memory Access (EDMA) Controller of the DSP (or the Xilinx) utilizes two
15 Control Registers, a 16 Word Data Write to PCI Bus FIFO, a 16 Word Data Read From PCI Bus FIFO, and a PCI Data Interrupt (PCIDATINT). The 32 Bit PCI Address Register holds either the starting Source Address for data storage controller Disk Writes where data is read from the PCI Bus, or the starting Destination Address for data storage controller Disk Reads where data is written to the PCI Bus. The second control register is
20 a PCI Count Register that specifies the direction of the data transfer along with the number of 32 bit Data words to be written to or from the PCI bus.

Data is written to the PCI Bus from the DSP via a 16 Word PCI Data Write FIFO located within a prespecified address range. Data writes from the DSP to anywhere

within the address range place that data word in the next available location within the FIFO. Data is read from the PCI Bus to the DSP via a 16 Word PCI Data Read FIFO located within a prespecified address range and data read by the DSP from anywhere within this address range provides the next data word from the FIFO.

5 After completion of the Xilinx initialization by the DSP and subsequent negation of the PCI Bus Reset signal (RST#) by the host computer's PCI Bridge, the data storage controller is ready to accept commands from the host computer via the PCI Bus. When accepting commands it should be noted that the data storage controller is a PCI Target (Slave) Device. Commands are preferably fixed in length at exactly 8 (thirty-two bit) words long. Commands are written from the host computer to the data storage controller via the PCI Bus utilizing the data storage controller's Base Address Register 0 (BAR0). The PCI Bus Reset initially sets the Command FIFO's Counter to zero and also signals the Xilinx's PCI Bus State Controller that the Command FIFO is empty and enable to accept a command.

10
15 Whenever a data write occurs within the valid data range of BAR0, the data word is accepted from PCI Bus and placed in the next available memory position within the Command FIFO. When the last of the 8 thirty-two bit data words is accepted by the PCI Bus (thus completing the command, i.e. last word for the command FIFO to be full), the PCI Bus State Controller is automatically set to Target Abort (within same PCI Transaction) or Disconnect Without Data for all subsequent PCI transactions that try to writes to BAR0. This automatic setting is the responsibility of the Xilinx PCI Data Interface.

The PCI Command FIFO State Controller then asserts the Command Available

Interrupt to the DSP. The DSP services the Command Available Interrupt by reading the command data from a prespecified address range. It should be noted that the command FIFO is read sequentially from any data access that reads data within such address range. It is the responsibility of the DSP to understand that the data is read sequentially from any order of accesses within the data range and should thus be stored accordingly.

Upon completion of the Command Available Interrupt Service Routine the DSP executes a memory read or write to desired location within the PCI Control Register Space mapped into the DSP's CE3 (Xilinx) memory space. This resets the Command FIFO Counter back to zero. Next, the DSP executes a memory read or write to location in the DSP Memory Space that clears the Command Available Interrupt. Nested interrupts are not possible since the PCI Bus State Machine is not yet able to accept any Command Data at BAR0. Once the Command Available Interrupt routine has cleared the interrupt and exited, the DSP may then enable the PCI State Machine to accept a new command by reading or writing to PCI Command Enable location within the PCI Command FIFO Control Register Space.

A preferred architecture has been selected to enable the data storage controller to operate on one command at a time or to accept multiple prioritized commands in future implementations. Specifically, the decoupling of the Command Available Interrupt Service Routine from the PCI State Machine that accepts Commands at BAR0 enables the DSP's "operating system kernel" to accept additional commands at any time by software command. In single command operation, a command is accepted, the Command Available Interrupt Cleared, and the Command executed by the data storage controller in PCI Master Mode prior to the enabling of the PCI State machine to accept new

commands.

In a prioritized multi-command implementation, the “operating system kernel” may elect to immediately accept new commands or defer the acceptance of new commands based upon any software implemented decision criteria. In one embodiment, the O/S code might only allow a pre-specified number of commands to be queued. In another embodiment, commands might only be accepted during processor idle time or when the DSP is not executing time critical (i.e. highly pipelined) compress/decompress routines. In yet another embodiment, various processes are enabled based upon a pre-emptive prioritized based scheduling system.

As previously stated, the data storage controller retrieves commands from the input command FIFO in 8 thirty-two bit word packets. Prior to command interpretation and execution, a command’s checksum value is computed to verify the integrity of the data command and associated parameters. If the checksum fails, the host computer is notified of the command packet that failed utilizing the Command Protocol Error Handler. Once the checksum is verified the command type and associated parameters are utilized as an offset into the command “pointer” table or may other suitable command/data structure that transfers control to the appropriate command execution routine.

Commands are executed by the data storage controller with the data storage controller acting as a PCI Master. This is in direct contrast to command acceptance where the data storage controller acts as a PCI Slave. When acting as a PCI Bus Master, the data storage controller reads or writes data to the PCI Bus utilizing a separate PCI Bus Data FIFO (distinct & apart from the Command FIFO). The PCI Data FIFO is 64 (thirty-two bit) words deep and may be utilized for either data reads or data writes from the DSP

to the PCI Bus, but not both simultaneously.

For data to be written from the data storage controller to the Host Computer, the DSP must first write the output data to the PCI Bus Data FIFO. The Data FIFO is commanded to PCI Bus Data Write Mode by writing to a desired location within the Xilinx (CE3) PCI Control Register Space. Upon PCI Bus Reset the default state for the PCI Data FIFO is write mode and the PCI Data FIFO Available Interrupt is cleared. The PCI Data FIFO Available Interrupt should also be software cleared by writing to a prespecified location. Preferably, the first task for the data storage controller is for system boot-up or application code to be downloaded from disk. For reference, PCI Data Read Mode is commanded by writing to location BFF0 0104. The PCI Bus Reset initializes the Data FIFO Pointer to the first data of the 64 data words within the FIFO. However this pointer should always be explicitly initialized by a memory write to location BFF0 0108. This ensures that the first data word written to the FIFO by the DSP performing the data write anywhere in address range B000 0000 to B000 01FF is placed at the beginning of the FIFO. Each subsequent write to any location within this address range then places one thirty-two bit data word into the next available location within the PCI Data FIFO. The FIFO accepts up to 64 thirty-two bit data words although it should be clearly understood that not all data transfers to and from the PCI Bus will consist of a full FIFO. Counting the number of thirty-two bit data words written to the PCI Data FIFO is the responsibility of the DSP Code. It is envisioned that the DSP will, in general, use 64 word DMA data transfers, thus alleviating any additional processor overhead.

When the data has been transferred from the DSP to the PCI Data FIFO, the PCI Bus Controller also needs the address of the PCI Target along with the number of data

words to be transmitted. In the current data storage controller implementation, the PCI Bus Address is thirty-two bits wide, although future PCI bus implementations may utilize multiword addressing and/or significantly larger (64 bit & up) address widths. The single thirty-two bit address word is written by the DSP to memory location `aaaa+0x10` in the
5 PCI Control Register Space.

Finally, the PCI Bus Data Write transaction is initiated by writing the PCI Data FIFO word count to a prespecified memory address. The word count value is always decimal 64 or less (`0x3F`). When the count register is written the value is automatically transferred to the PCI Controller for executing the PCI Bus Master writes.

10 When the PCI Bus has completed the transfer of all data words within the PCI Data FIFO the PCI Data FIFO Available Interrupt is set. The DSP PCI Data FIFO Available Interrupt handler will then check to see if additional data is waiting or expected to be written to the PCI Data Bus. If additional data is required the interrupt is cleared and the data transfer process repeats. If no additional data is required to be transferred
15 then the interrupt is cleared and the routine must exit to a system state controller. For example, if the command is complete then master mode must be disabled and then slave mode (command mode) enabled – assuming a single command by command execution data storage controller.

For data to be read by the data storage controller from the Host Computer, the
20 DSP must command the PCI Bus with the address and quantity of data to be received.

The PCI Data FIFO is commanded to PCI Bus Data Read Mode by writing to a desired location within the Xilinx (CE3) PCI Control Register Space. Upon PCI Bus Reset the default state for the PCI Data FIFO is Write Mode and the PCI Data FIFO Full

Interrupt is cleared. The PCI Data FIFO Full Interrupt should also be cleared via software by writing to such location. The PCI Bus Reset also initializes the PCI Data FIFO Pointer to the first data word of the available 64 data words within the FIFO. However this pointer should always be explicitly initialized by a memory write to prespecified location.

5 For data to be read from the PCI Bus by the data storage controller, the Xilinx PCI Bus Controller requires the address of the PCI Target along with the number of data words to be received. In the current data storage controller implementation, the PCI Bus Address is thirty-two bits wide, although future PCI bus implementations may utilize multiword addressing and/or significantly larger (64 bit & up) address widths. The single
10 thirty-two bit address word is written by the DSP to prespecified memory location in the PCI Control Register Space.

Finally, the PCI Bus Data Read transaction is initiated by writing the PCI Data FIFO word count to prespecified memory address. The word count value is always decimal 64 or less (0x3F). When the count register is written the value is automatically
15 transferred to the PCI Controller for executing the PCI Bus Master Read.

When the PCI Bus has received all the requested data words PCI Data FIFO Full Interrupt is set. The DSP PCI Data FIFO Full Interrupt handler will then check to see if additional data is waiting or expected to be read from the PCI Data Bus. If additional data is required the interrupt is cleared and the data receipt process repeats. If no
20 additional data is required to be transferred, then the interrupt is cleared and the routine exits to a system state controller. For example, if the command is complete then master mode must be disabled and then slave mode (command mode) enabled – assuming a single command by command execution data storage controller.

It is clearly understood that there are other techniques for handling the PCI Data transfers. The current methodology has been selected to minimize the complexity and resource utilization of the Xilinx Gate Array. It should also be understood that the utilization of asynchronous memory reads and writes to initialize system states and synchronize events at a software level aids in both hardware and system level debug at the expense of increase software overhead. Subsequent embodiments of the gate array may automate resource intensive tasks if system level performance mandates.

IV. Memory Bandwidth Allocation

The onboard cache of the data storage controller is shared by the DSP, Disk Interface, and PCI Bus. The best case, maximum bandwidth for the SDRAM memory is 70 megawords per second, or equivalently, 280 megabytes per second. The 32 bit PCI Bus interface has a best case bandwidth of 132 megabytes per second, or equivalently 33 megawords per second. In current practice, this bandwidth is only achieved in short bursts. The granularity of PCI data bursts to/from the data storage controller is governed by the PCI Bus interface data buffer depth of sixteen words (64 bytes). The time division multiplexing nature of the current PCI Data Transfer Buffering methodology cuts the sustained PCI bandwidth down to 66 megabytes/second.

Data is transferred across the ultraDMA disk interface at a maximum burst rate of 66 megabytes/second. It should be noted that the burst rate is only achieved with disks that contain onboard cache memory. Currently this is becoming more and more popular within the industry. However assuming a disk cache miss, the maximum transfer rates from current disk drives is approximately six megabytes per second. Allotting for

technology improvements over time, the data storage controller has been designed for a maximum sustained disk data rate of 20 megabytes second (5 megawords/second). A design challenge is created by the need for continuous access to the SDRAM memory. Disks are physical devices and it is necessary to continuously read data from disk and place it into memory, otherwise the disk will incur a full rotational latency prior to continuing the read transaction. The maximum SDRAM access latency that can be incurred is the depth of the each of the two disk FIFO s or sixteen data. Assuming the FIFO is sixteen words deep the maximum latency time for emptying the other disk FIFO and restoring it to the disk interface is sixteen words at 5 megawords per second or (16 x 3.2usec) = 1usec. Each EMIF clock cycle is 14.2857nsec, thus the maximum latency translates to 224 clock cycles. It should be noted that transfers across the disk interface are 16 bits wide, thus the FPGA is required to translate 32 bit memory transfers to 16 bit disk transfers, and vice-versa.

The DSP services request for its external bus from two requestors, the Enhanced Direct Memory Access (EDMA) Controller and an external shared memory device controller. The DSP can typically utilize the full 280 megabytes of bus bandwidth on an 8k through 64K byte (2k word through 16k word) burst basis. It should be noted that the DSRA does not utilize the SDRAM memory for interim processing storage, and as such only utilizes bandwidth in direct proportion to disk read and write commands.

For a single read from disk transaction data is transferred from and DMA transfer into SDRAM memory. This data is then DMA transferred by the DSP into onboard DSP memory, processed, and re transferred back to SDRAM in decompressed format (3 words

for every one word in). Finally the data is read from SDRAM by the PCI Bus Controller and placed into host computer memory. This equates to eight SDRAM accesses, one write from disk, one read by the DSP, three writes by the DSP and three by the PCI Bus. Disk write transactions similarly require eight SDRAM accesses, three from the PCI, 5 three DSP reads, one DSP write, and one to the disk.

Neglecting overhead for setting up DMA transfers, arbitration latencies, and memory wait states for setting up SDRAM transactions, the maximum DSRA theoretical SDRAM bandwidth limit for disk reads or writes is 280/8 megabytes second or 35 megabytes second. It should be noted that the best case allocation of SDRAM bandwidth 10 would be dynamic dependent upon the data compression and decompression ratios. Future enhancements to the data storage controller will utilize a programmable timeslice system to allocate SDRAM bandwidth, however this first embodiment will utilize a fixed allocation ratio as follows:

If all three requestors require SDRAM simultaneously:

15	PCI Bus Interface	3/8
	DSP Accesses	4/8
	UltraDMA Disk Interface	1/8

If only the PCI Bus and DSP require SDRAM:

	PCI Bus Interface	4/8
20	DSP Accesses	4/8

If only the DSP and Disk require SDRAM:

	DSP Accesses	6/8
--	--------------	-----

UltraDMA Disk Interface 2/8

If only the PCI Bus and Disk require SDRAM:

PCI Bus Interface 6/8

UltraDMA Disk Interface 2/8

5

If only one device requires SDRAM it receives the full SDRAM bandwidth. It should be noted that different ratios may be applied based upon the anticipated or actual compression and/or decompression ratios. For example in the case of all three requestors active the following equation applies. Assume that data storage accelerator achieves a compression ratio A:B for example 3:1. The Numerator and denominators of the various allocations are defined as follows:

10

PCI Bus Interface A/K

DSP Accesses (A+B)/K

UltraDMA Disk Interface B/K

15

Where Further define a sum K equal to the sum of the numerators of the PCI Bus interface fraction, the DSP Access fraction, and the UltraDMA Disk Interfaces, i.e. $K = 2(A+B)$. Similarly:

If only the PCI Bus and DSP require SDRAM:

PCI Bus Interface (A+B)/K

DSP Accesses (A+B)/K

20

If only the DSP and Disk require SDRAM:

DSP Accesses	2A/K
UltraDMA Disk Interface	2B/K

If only the PCI Bus and Disk require SDRAM:

5	PCI Bus Interface	2A/K
	UltraDMA Disk Interface	2B/K

It should be noted that the resultant ratios may all be scaled by a constant in order to most effectively utilize the bandwidths of the internal busses and external interfaces. In addition each ratio can be scale by an adjustment factor based upon the time required to complete individual cycles. For example if PCI Bus interface takes 20% longer than all other cycles, the PCI time slice should be adjusted longer accordingly.

V. Instant Boot Device For Operating System, Application Program and Loading

Typically, with conventional boot device controllers, after reset, the boot device controller will wait for a command over the computer bus (such as PCI). Since the boot device controller will typically be reset prior to bus reset and before the computer bus starts sending commands, this wait period is unproductive time. The initial bus commands inevitably instruct the boot device controller to retrieve data from the boot device (such as a disk) for the operating system. Since most boot devices are relatively slow compared to the speed of most computer busses, a long delay is seen by the computer user. This is evident in the time it takes for a typical computer to boot.

It is to be appreciated that a data storage controller (having an architecture as described herein) may employ a technique of data preloading to decrease the computer system boot time. Upon host system power-up or reset, the data storage controller will

perform a self-diagnostic and program the programmable logic device (as discussed above) prior to completion of the host system reset (e.g., PCI bus reset) so that the logic device can accept PCI Bus commands after system reset. Further, prior to host system reset, the data storage controller can proceed to pre-load the portions of the computer operating system from the boot device (e.g., hard disk) into the on-board cache memory. The data storage controller preloads the needed sectors of data in the order in which they will be needed. Since the same portions of the operating system must be loaded upon each boot process, it is advantageous for the boot device controller to preload such portions and not wait until it is commanded to load the operating system. Preferably, the data storage controller employs a dedicated IO channel of the DSP (with or without data compression) to pre-load computer operating systems and applications.

Once the data is preloaded, when the computer system bus issues its first read commands to the data storage controller seeking operating system data, the data will already be available in the cache memory of the data storage controller. The data storage controller will then be able to instantly start transmitting the data to the system bus. Before transmission to the bus, if the was stored in compressed format on the boot device, the data will be decompressed. The process of preloading required (compressed) portions of the operating system significantly reduces the computer boot process time.

In addition to preloading operating system data, the data storage controller could also preload other data that the user would likely want to use at startup. An example of this would be a frequently used application such as a word processor and any number of document files.

There are several techniques that may be employed in accordance with the present

invention that would allow the data storage controller to know what data to preload from the boot device. One technique utilizes a custom utility program that would allow the user to specify what applications/data should be preloaded.

Another technique (illustrated by the flow diagram of Figs. 7a and 7b) that may be employed comprises an automatic process that requires no input from the user. With this technique, the data storage controller maintain a list comprising the data associated with the first series of data requests received by the data storage controller by the host system after a power-on / reset. In particular, referring to Fig. 7a, during the computer boot process, the data storage controller will receive requests for the boot data (step 70). In response, the data storage controller will retrieve the requested boot data from the boot device (e.g., hard disk) in the local cache memory (step 71). For each requested data block, the data storage controller will record the requested data block number in a list (step 72). The data storage controller will record the data block number of each data block requested by the host computer during the boot process (repeat steps 70-72). When the boot process is complete (affirmative determination in step 73), the data storage controller will store the data list on the boot device (or other storage device) (step 74).

Then, upon each subsequent power-on / reset (affirmative result in step 75), the data storage controller would retrieve and read the stored list (step 76) and proceed to preload the boot data specified on the list (i.e., the data associated with the expected data requests) into the onboard cache memory (step 77). It is to be understood that the depending on the resources of the given system (e.g., memory, etc.), the preloading process may be completed prior to commencement of the boot process, or continued after the boot process begins (in which case booting and preloading are performed

simultaneously).

When the boot process begins (step 78) (i.e., the storage controller is initialized and the system bus reset is deasserted), the data storage controller will receive requests for boot data (step 79). If the host computer issues a request for boot data that is pre-
5 loaded in the local memory of the data storage controller (affirmative result in step 80), the request is immediately serviced using the preloaded boot data (step 81). If the host computer issues a request for boot data that is not preloaded in the local memory of the data storage controller (negative determination in step 80), the controller will retrieve the requested data from the boot device, store the data in the local memory, and then deliver
10 the requested boot data to the computer bus (step 82). In addition, the data storage controller would update the boot data list by recording any changes in the actual data requests as compared to the expected data requests already stored in the list (step 83). Then, upon the next boot sequence, the boot device controller would pre-load that data into the local cache memory along with the other boot data previously on the list.

15 Further, during the boot process, if no request is made by the host computer for a data block that was pre-loaded into the local memory of the data storage controller (affirmative result in step 84), then the boot data list will be updated by removing the non-requested data block from the list (step 85). Thereafter, upon the next boot sequence, the data storage controller will not pre-load that data into local memory.

20 **VI. Quick Launch for Operating System, Application Program, and Loading**

It is to be appreciated that the data storage controller (having an architecture as described herein) may employ a technique of data preloading to decrease the time to load application programs (referred to as “quick launch”). Conventionally, when a user

launches an application, the file system reads the first few blocks of the file off the disk, and then the portion of the loaded software will request via the file system what additional data it needs from the disk. For example, a user may open a spreadsheet program, and the program may be configured to always load a company spreadsheet each
5 time the program is started. In addition, the company spreadsheet may require data from other spreadsheet files.

In accordance with the present invention, the data storage controller may be configured to “remember” what data is typically loaded following the launch of the spreadsheet program, for example. The data storage controller may then proceed to
10 preload the company spreadsheet and all the necessary data in the order in which such data is needed. Once this is accomplished, the data storage controller can service read commands using the preloaded data. Before transmission to the bus, if the preloaded data was stored in compressed format, the data will be decompressed. The process of preloading (compressed) program data significantly reduces the time for launching an
15 application.

Preferably, a custom utility program is employed that would allow the user to specify what applications should be made ready for quick launch.

Figs. 8a and 8b comprise a flow diagram of a quick launch method according to one aspect of the present invention. With this technique, the data storage controller
20 maintains a list comprising the data associated with launching an application. In particular, when an application is first launched, the data storage controller will receive requests for the application data (step 90). In response, the data storage controller will retrieve the requested application data from memory (e.g., hard disk) and store it in the

local cache memory (step 91). The data storage controller will record the data block number of each data block requested by the host computer during the launch process (step 92). When the launch process is complete (affirmative determination in step 93), the data storage controller will store the data list in a designated memory location (step 94).

5 Then, referring to Fig. 8b, upon each subsequent launch of the application (affirmative result in step 95), the data storage controller would retrieve and read the stored list (step 96) and then proceed to preload the application data specified on the list (i.e., the data associated with the expected data requests) into the onboard cache memory (step 97). During the application launch process, the data storage controller will receive
10 requests for application data (step 98). If the host computer issues a request for application data that is pre-loaded in the local memory of the data storage controller (affirmative result in step 99), the request is immediately serviced using the preloaded data (step 100). If the host computer issues a request for application data that is not preloaded in the local memory of the data storage controller (negative result in step 99),
15 the controller will retrieve the requested data from the hard disk memory, store the data in the local memory, and then deliver the requested application data to the computer bus (step 101). In addition, the data storage controller would update the application data list by recording any changes in the actual data requests as compared to the expected data requests already stored in the list (step 102).

20 Further, during the launch process, if no request is made by the host computer for a data block that was pre-loaded into the local memory of the data storage controller (affirmative result in step 103), then the application data list will be updated by removing the non-requested data block from the list (step 104). Thereafter, upon the next launch

sequence for the given application, the data storage controller will not pre-load that data into local memory.

It is to be understood that the quick boot and quick launch methods described above are preferably implemented by a storage controller according to the present invention and may or may not utilize data compression/decompression by the DSP. However, it is to be understood that the quick boot and quick launch methods may be implemented by a separate device, processor, or system, or implemented in software.

VII. Content Independent Data Compression

It is to be understood that any conventional compression/decompression system and method (which comply with the above mentioned constraints) may be employed in the data storage controller for providing accelerated data storage and retrieval in accordance with the present invention. Preferably, the present invention employs the data compression/decompression techniques disclosed in the above-incorporated U.S. Serial No. 09/210,491.

Referring to FIG. 9, a detailed block diagram illustrates an exemplary data compression system 110 that may be employed herein. Details of this data compression system are provided in U.S. Serial No. 09/210,491. In this embodiment, the data compression system 110 accepts data blocks from an input data stream and stores the input data block in an input buffer or cache 115. It is to be understood that the system processes the input data stream in data blocks that may range in size from individual bits through complete files or collections of multiple files. Additionally, the input data block size may be fixed or variable. A counter 120 counts or otherwise enumerates the size of input data block in any convenient units including bits, bytes, words, and double words.

It should be noted that the input buffer 115 and counter 120 are not required elements of the present invention. The input data buffer 115 may be provided for buffering the input data stream in order to output an uncompressed data stream in the event that, as discussed in further detail below, every encoder fails to achieve a level of compression that exceeds
5 an *a priori* specified minimum compression ratio threshold.

Data compression is performed by an encoder module 125 which may comprise a set of encoders E1, E2, E3 ... En. The encoder set E1, E2, E3 ... En may include any number "n" (where n may =1) of those lossless encoding techniques currently well known within the art such as run length, Huffman, Lempel-Ziv Dictionary Compression,
10 arithmetic coding, data compaction, and data null suppression. It is to be understood that the encoding techniques are selected based upon their ability to effectively encode different types of input data. It is to be appreciated that a full complement of encoders are preferably selected to provide a broad coverage of existing and future data types.

The encoder module 125 successively receives as input each of the buffered input
15 data blocks (or unbuffered input data blocks from the counter module 120). Data compression is performed by the encoder module 125 wherein each of the encoders E1 En processes a given input data block and outputs a corresponding set of encoded data blocks. It is to be appreciated that the system affords a user the option to enable/disable any one or more of the encoders E1.... En prior to operation. As is understood by those
20 skilled in the art, such feature allows the user to tailor the operation of the data compression system for specific applications. It is to be further appreciated that the encoding process may be performed either in parallel or sequentially. In particular, the encoders E1 through En of encoder module 125 may operate in parallel (i.e.,

simultaneously processing a given input data block by utilizing task multiplexing on a single central processor, via dedicated hardware, by executing on a plurality of processor or dedicated hardware systems, or any combination thereof). In addition, encoders E1 through En may operate sequentially on a given unbuffered or buffered input data block.

5 This process is intended to eliminate the complexity and additional processing overhead associated with multiplexing concurrent encoding techniques on a single central processor and/or dedicated hardware, set of central processors and/or dedicated hardware, or any achievable combination. It is to be further appreciated that encoders of the identical type may be applied in parallel to enhance encoding speed. For instance,
10 encoder E1 may comprise two parallel Huffman encoders for parallel processing of an input data block.

A buffer/counter module 130 is operatively connected to the encoder module 125 for buffering and counting the size of each of the encoded data blocks output from encoder module 125. Specifically, the buffer/counter 130 comprises a plurality of
15 buffer/counters BC1, BC2, BC3 ...BCn, each operatively associated with a corresponding one of the encoders E1...En. A compression ratio module 135, operatively connected to the output buffer/counter 130, determines the compression ratio obtained for each of the enabled encoders E1...En by taking the ratio of the size of the input data block to the size of the output data block stored in the corresponding buffer/counters BC1 ... BCn. In
20 addition, the compression ratio module 135 compares each compression ratio with an *a priori*-specified compression ratio threshold limit to determine if at least one of the encoded data blocks output from the enabled encoders E1...En achieves a compression that exceeds an *a priori*-specified threshold. As is understood by those skilled in the art,

the threshold limit may be specified as any value inclusive of data expansion, no data compression or expansion, or any arbitrarily desired compression limit. A description module 138, operatively coupled to the compression ratio module 135, appends a corresponding compression type descriptor to each encoded data block which is selected for output so as to indicate the type of compression format of the encoded data block. A data compression type descriptor is defined as any recognizable data token or descriptor that indicates which data encoding technique has been applied to the data. It is to be understood that, since encoders of the identical type may be applied in parallel to enhance encoding speed (as discussed above), the data compression type descriptor identifies the corresponding encoding technique applied to the encoded data block, not necessarily the specific encoder. The encoded data block having the greatest compression ratio along with its corresponding data compression type descriptor is then output for subsequent data processing, storage, or transmittal. If there are no encoded data blocks having a compression ratio that exceeds the compression ratio threshold limit, then the original unencoded input data block is selected for output and a null data compression type descriptor is appended thereto. A null data compression type descriptor is defined as any recognizable data token or descriptor that indicates no data encoding has been applied to the input data block. Accordingly, the unencoded input data block with its corresponding null data compression type descriptor is then output for subsequent data processing, storage, or transmittal.

Again, it is to be understood that the embodiment of the data compression engine of Fig. 9 is exemplary of a preferred compression system which may be implemented in the present invention, and that other compression systems and methods known to those

skilled in the art may be employed for providing accelerated data storage in accordance with the teachings herein. Indeed, in another embodiment of the compression system disclosed in the above-incorporated U.S. Serial No. 09/210,491, a timer is included to measure the time elapsed during the encoding process against an *a priori*-specified time limit. When the time limit expires, only the data output from those encoders (in the encoder module 125) that have completed the present encoding cycle are compared to determine the encoded data with the highest compression ratio. The time limit ensures that the real-time or pseudo real-time nature of the data encoding is preserved. In addition, the results from each encoder in the encoder module 125 may be buffered to allow additional encoders to be sequentially applied to the output of the previous encoder, yielding a more optimal lossless data compression ratio. Such techniques are discussed in greater detail in the above-incorporated U.S. Serial No. 09/210,491.

Referring now to FIG. 10, a detailed block diagram illustrates an exemplary decompression system that may be employed herein or accelerated data retrieval as disclosed in the above-incorporated U.S. Serial No. 09/210,491. In this embodiment, the data compression engine 180 retrieves or otherwise accepts compressed data blocks from one or more data storage devices and inputs the data via a data storage interface. It is to be understood that the system processes the input data stream in data blocks that may range in size from individual bits through complete files or collections of multiple files. Additionally, the input data block size may be fixed or variable.

The data decompression engine 180 comprises an input buffer 155 that receives as input an uncompressed or compressed data stream comprising one or more data blocks. The data blocks may range in size from individual bits through complete files or

collections of multiple files. Additionally, the data block size may be fixed or variable. The input data buffer 55 is preferably included (not required) to provide storage of input data for various hardware implementations. A descriptor extraction module 160 receives the buffered (or unbuffered) input data block and then parses, lexically, syntactically, or otherwise analyzes the input data block using methods known by those skilled in the art to extract the data compression type descriptor associated with the data block. The data compression type descriptor may possess values corresponding to null (no encoding applied), a single applied encoding technique, or multiple encoding techniques applied in a specific or random order (in accordance with the data compression system embodiments and methods discussed above).

A decoder module 165 includes one or more decoders D1...Dn for decoding the input data block using a decoder, set of decoders, or a sequential set of decoders corresponding to the extracted compression type descriptor. The decoders D1...Dn may include those lossless encoding techniques currently well known within the art, including: run length, Huffman, Lempel-Ziv Dictionary Compression, arithmetic coding, data compaction, and data null suppression. Decoding techniques are selected based upon their ability to effectively decode the various different types of encoded input data generated by the data compression systems described above or originating from any other desired source.

As with the data compression systems discussed in U.S. Application Serial No. 09/210,491, the decoder module 165 may include multiple decoders of the same type applied in parallel so as to reduce the data decoding time. An output data buffer or cache 170 may be included for buffering the decoded data block output from the decoder

module 165. The output buffer 70 then provides data to the output data stream. It is to be appreciated by those skilled in the art that the data compression system 180 may also include an input data counter and output data counter operatively coupled to the input and output, respectively, of the decoder module 165. In this manner, the compressed and
5 corresponding decompressed data block may be counted to ensure that sufficient decompression is obtained for the input data block.

Again, it is to be understood that the embodiment of the data decompression system 180 of FIG. 10 is exemplary of a preferred decompression system and method which may be implemented in the present invention, and that other data decompression
10 systems and methods known to those skilled in the art may be employed for providing accelerated data retrieval in accordance with the teachings herein.

Although illustrative embodiments have been described herein with reference to the accompanying drawings, it is to be understood that the present invention is not limited to those precise embodiments, and that various other changes and modifications may be
15 affected therein by one skilled in the art without departing from the scope or spirit of the invention. All such changes and modifications are intended to be included within the scope of the invention as defined by the appended claims.

WHAT IS CLAIMED IS:

1. A method for providing accelerated loading of an operating system, comprising the steps of:

- maintaining a list of boot data used for booting a computer system;
- 5 preloading the boot data upon initialization of the computer system; and
- servicing requests for boot data from the computer system using the preloaded boot data.

2. The method of claim 1, wherein the boot data comprises program code
10 associated with one of an operating system of the computer system, an application program, and a combination thereof.

3. The method of claim 1, wherein the step of preloading the boot data comprises
15 retrieving boot data from a boot device and storing the retrieved data in a cache memory.

4. The method of claim 3, wherein the method steps are performed by a data
storage controller connected to the boot device.

5. The method of claim 1, further comprising the step of updating the list of boot
20 data during the boot process.

6. The method of claim 5, wherein the step of updating comprises adding to the
list any boot data requested by the computer system not previously stored in the list.

7. The method of claim 5, wherein the step of updating comprises removing from the list any boot data previously stored in the list and not requested by the computer system.

5 8. The method of claim 1, wherein the boot data is compressed and further comprising the step of decompressing the preloaded boot data.

9. The method of claim 1, wherein the method steps are program instructions that are tangibly embodied on a program storage device and readable by a machine to execute
10 the method steps.

10. A method for providing accelerated launching of an application program, comprising the steps of:

maintaining a list of application data associated with an application program;
15 preloading the application data upon launching the application program; and
servicing requests for application data from a computer system using the preloaded application data.

11. The method of claim 1, wherein the application data is compressed and
20 further comprising the step of decompressing the preloaded application data.

12. The method of claim 10, wherein the method steps are program instructions that are tangibly embodied on a program storage device and readable by a machine to

execute the method steps.

13. A boot device controller for providing accelerated loading of an operating system of a host system, the boot device controller comprising:

5 a digital signal processor (DSP);

a programmable logic device, wherein the programmable logic device is programmed by the digital signal processor to (i) instantiate a first interface for operatively interfacing the boot device controller to a boot device and to (ii) instantiate a second interface for operatively interfacing the boot device controller to the host system;

10 and

a non-volatile memory device, for storing logic code associated with the DSP, the first interface and the second interface, wherein the logic code comprises instructions executable by the DSP for maintaining a list of boot data used for booting the host system, preloading the boot data upon initialization of the host system, and servicing requests for boot data from the host system using the preloaded boot data.

15

14. The boot device controller of claim 13, further comprising a cache memory device for storing the preloaded boot data.

20

15. The boot device controller of claim 13, wherein the logic code in the non-volatile memory device further comprises program instructions executable by the DSP for maintaining a list of application data associated with an application program; preloading the application data upon launching the application program, and servicing requests for

the application data from the host system using the preloaded application data.

16. The boot device controller of claim 12, wherein the boot data is compressed and wherein the DSP comprises a data compression engine (DCE) for decompressing the preloaded boot data.

5

8011-15

ABSTRACT OF THE DISCLOSURE

Systems and methods for providing accelerated loading of operating system and application programs upon system boot or application launch. In one aspect, a method for providing accelerated loading of an operating system comprises the steps of: maintaining
5 a list of boot data used for booting a computer system; preloading the boot data upon initialization of the computer system; and servicing requests for boot data from the computer system using the preloaded boot data. The boot data may comprise program code associated with an operating system of the computer system, an application program, and a combination thereof. In a preferred embodiment, the boot data is
10 retrieved from a boot device and stored in a cache memory device. In another aspect, the method for accelerated loading of an operating system comprises updating the list of boot data during the boot process, wherein updating comprises adding to the list any boot data requested by the computer system not previously stored in the list and/or removing from the list any boot data previously stored in the list and not requested by the computer
15 system. In yet another aspect, the boot data is stored in a compressed format on the boot device and the preloaded boot data is decompressed prior to transmitting the preloaded boot data to the requesting system. In another aspect, a method for providing accelerated launching of an application program comprises the steps of: maintaining a list of application data associated with an application program; preloading the application data
20 upon launching the application program; and servicing requests for application data from a computer system using the preloaded application data.



UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
 UNITED STATES PATENT AND TRADEMARK OFFICE
 WASHINGTON, D.C. 20231
 www.uspto.gov



Bib. Data Sheet

CONFIRMATION NO. 9730

SERIAL NUMBER 09/776,267	FILING DATE 02/02/2001 RULE	CLASS 713	GROUP ART UNIT 2182	ATTORNEY DOCKET NO. 8011-15
------------------------------------	---	---------------------	-------------------------------	---------------------------------------

APPLICANTS

James J. Fallon, Armonk, NY;
 John Buck, Oceanside, NY;
 Paul F. Pickel, Bethpage, NY;
 Stephen J. McErlain, New York, NY;

**** CONTINUING DATA *******

THIS APPLN CLAIMS BENEFIT OF 60/180,114 02/03/2000
yes yes

**** FOREIGN APPLICATIONS *******

None yes

IF REQUIRED, FOREIGN FILING LICENSE
 GRANTED ** 03/09/2001

**** SMALL ENTITY ****

Foreign Priority claimed <input type="checkbox"/> yes <input checked="" type="checkbox"/> no	STATE OR COUNTRY NY	SHEETS DRAWING 13	TOTAL CLAIMS 16	INDEPENDENT CLAIMS 3
35 USC 119 (a-d) conditions met <input type="checkbox"/> yes <input type="checkbox"/> no <input type="checkbox"/> Met after Allowance				
Verified and Acknowledged <i>Yes</i> Examiner's Signature _____ Initials _____				

ADDRESS

Frank Chau, Esq.
 F. CHAU & ASSOCIATES, LLP
 Suite 501
 1900 Hempstead Turnpike
 East Meadow ,NY 11554

TITLE

Systems and methods for accelerated loading of operating systems and application programs

FILING FEE RECEIVED 420	FEES: Authority has been given in Paper No. _____ to charge/credit DEPOSIT ACCOUNT No. _____ for following:	<input type="checkbox"/> All Fees
		<input type="checkbox"/> 1.16 Fees (Filing)
		<input type="checkbox"/> 1.17 Fees (Processing Ext. of time)
		<input type="checkbox"/> 1.18 Fees (Issue)
		<input type="checkbox"/> Other _____
		<input type="checkbox"/> Credit

PATENT APPLICATION SERIAL NO. _____

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

02/12/2001 NPRASASO 00000005 09776267

01 FC:201

355.00 OP

PTO-1556
(5/87)

*U.S. GPO: 2000-468-987/39595

PATENT APPLICATION FEE DETERMINATION RECORD

Effective October 1, 2000

Application or Docket Number

CLAIMS AS FILED - PART I

	(Column 1)	(Column 2)
TOTAL CLAIMS	16	
FOR	NUMBER FILED	NUMBER EXTRA
TOTAL CHARGEABLE CLAIMS	16 minus 20 =	* 0
INDEPENDENT CLAIMS	3 minus 3 =	* 0
MULTIPLE DEPENDENT CLAIM PRESENT <input type="checkbox"/>		

* If the difference in column 1 is less than zero, enter "0" in column 2

SMALL ENTITY TYPE

OR OTHER THAN SMALL ENTITY

RATE	FEE		RATE	FEE
BASIC FEE	355.00	OR	BASIC FEE	710.00
X\$ 9=		OR	X\$18=	
X40=		OR	X80=	
+135=		OR	+270=	
TOTAL	355.00	OR	TOTAL	

CLAIMS AS AMENDED - PART II

	(Column 1)	(Column 2)	(Column 3)
AMENDMENT A	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
	Total	Minus **	=
	Independent	Minus ***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM <input type="checkbox"/>			

SMALL ENTITY

OR OTHER THAN SMALL ENTITY

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X\$ 9=		OR	X\$18=	
X40=		OR	X80=	
+135=		OR	+270=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

	(Column 1)	(Column 2)	(Column 3)
AMENDMENT B	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
	Total	Minus **	=
	Independent	Minus ***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM <input type="checkbox"/>			

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X\$ 9=		OR	X\$18=	
X40=		OR	X80=	
+135=		OR	+270=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

	(Column 1)	(Column 2)	(Column 3)
AMENDMENT C	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
	Total	Minus **	=
	Independent	Minus ***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM <input type="checkbox"/>			

RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X\$ 9=		OR	X\$18=	
X40=		OR	X80=	
+135=		OR	+270=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.

** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."

*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."

The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

Best Available Copy

CLAIMS ONLY

SERIAL NO.	FILING DATE
APPLICANT(S)	

CLAIMS

	AS FILED		AFTER 1st AMENDMENT		AFTER 2nd AMENDMENT		*	*	*	IND.	DEP.	IND.	DEP.	IND.	DEP.
	IND.	DEP.	IND.	DEP.	IND.	DEP.									
1															
2															
3															
4															
5															
6															
7															
8															
9															
10															
11															
12															
13															
14															
15															
16															
17															
18															
19															
20															
21															
22															
23															
24															
25															
26															
27															
28															
29															
30															
31															
32															
33															
34															
35															
36															
37															
38															
39															
40															
41															
42															
43															
44															
45															
46															
47															
48															
49															
50															
TOTAL IND.	3	↓		↓		↓									
TOTAL DEP.	13	←		←		←									
TOTAL CLAIMS	16														
51															
52															
53															
54															
55															
56															
57															
58															
59															
60															
61															
62															
63															
64															
65															
66															
67															
68															
69															
70															
71															
72															
73															
74															
75															
76															
77															
78															
79															
80															
81															
82															
83															
84															
85															
86															
87															
88															
89															
90															
91															
92															
93															
94															
95															
96															
97															
98															
99															
100															
TOTAL IND.		↓		↓		↓									
TOTAL DEP.		←		←		←									
TOTAL CLAIMS															

* MAY BE USED FOR ADDITIONAL CLAIMS OR ADMENDMENTS

Best Available Copy