

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

INTELLECTUAL VENTURES I LLC and
INTELLECTUAL VENTURES II LLC,

Plaintiffs,

v.

SYMANTEC CORP.,

Defendant.

Civil Action No. 13-cv-440-LPS

JURY TRIAL DEMANDED

FIRST AMENDED COMPLAINT FOR PATENT INFRINGEMENT

Plaintiffs Intellectual Ventures I LLC (“Intellectual Ventures I”) and Intellectual Ventures II LLC (“Intellectual Ventures II”), by their undersigned attorneys, hereby allege as follows:

1. This is an action for patent infringement arising under the patent laws of the United States, Title 35 of the United States Code.

2. Intellectual Ventures I is a corporation organized and existing under the laws of Delaware, with a principal place of business located in Bellevue, Washington.

3. Intellectual Ventures II is a corporation organized and existing under the laws of Delaware, with a principal place of business located in Bellevue, Washington.

4. Defendant Symantec Corporation (“Symantec”) is a corporation organized and existing under the laws of Delaware, with a principal place of business in Mountain View, California. Symantec transacts substantial business, either directly or through its agents, on an ongoing basis in this judicial district and elsewhere in the United States.

5. Unless specifically stated otherwise, the acts complained of herein were committed by, on behalf of, and/or for the benefit of Symantec.

JURISDICTION AND VENUE

6. This Court has subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

7. This Court has personal jurisdiction over Symantec because Symantec is incorporated in the State of Delaware.

8. Venue is proper in this District pursuant to 28 U.S.C. §§ 1391 and 1400(b).

COUNT 1 – INFRINGEMENT OF U.S. PATENT NO. 5,537,533

9. On July 16, 1996, United States Patent No. 5,537,533 (“the ’533 patent”) was duly and legally issued for an invention entitled “System and method for remote mirroring of digital data from a primary network server to a remote network server.” Intellectual Ventures II is the owner of the ’533 patent and holds all rights and interests in the ’533 patent. A true and correct copy of the ’533 patent is attached hereto as Exhibit A.

10. Symantec has infringed and continues to infringe one or more claims of the ’533 patent by its manufacture, use, sale, importation, and/or offer for sale of certain storage software products, including but not limited to Symantec’s Replicator and Veritas Volume Replicator products. Symantec is liable for its infringement of the ’533 patent pursuant to 35 U.S.C. § 271.

11. Symantec actively, knowingly, and intentionally has induced, and continues to actively, knowingly, and intentionally induce, infringement of the ’533 patent by making, using, offering for sale, importing, and selling certain storage software products, including but not limited to Symantec’s Replicator and Veritas Volume Replicator products, as well as by contracting with others to use, market, sell, offer to sell, and import such products, all with knowledge of the ’533 patent and its claims; with knowledge that its customers and end users

will use, market, sell, offer to sell, and import such products; and with the knowledge and the specific intent to encourage and facilitate those infringing sales and uses of such products through the creation and dissemination of promotional and marketing materials, instructional materials, product manuals, and technical materials.

12. Symantec also has contributed to the infringement by others, including the end users of certain storage software products, including but not limited to Symantec's Replicator and Veritas Volume Replicator products, and continues to contribute to infringement by others, by selling, offering to sell, and importing such products into the United States, knowing that those products constitute a material part of the inventions of the '533 patent, knowing those products to be especially made or adapted to infringe the '533 patent, and knowing that those products are not staple articles or commodities of commerce suitable for substantial non-infringing use.

13. Symantec has had knowledge of and notice of the '533 patent and its infringement since at least, and through, the filing and service of the Complaint and despite this knowledge continues to commit the aforementioned infringing acts. In addition, Symantec has had knowledge of the '533 patent at least since December 3, 2009, when Symantec referenced the '533 patent in an information disclosure statement during the prosecution of Symantec's U.S. Patent No. 7,664,983, App. No. 11/215,958.

14. Symantec's acts of infringement have caused damage to Intellectual Ventures II, and Intellectual Ventures II is entitled to recover from Symantec the damages it has sustained as a result of Symantec's wrongful acts in an amount subject to proof at trial. Symantec's infringement of Intellectual Ventures II's exclusive rights under the '533 patent will continue to

damage Intellectual Ventures II, causing irreparable harm for which there is no adequate remedy at law, unless enjoined by this Court.

15. Upon information and belief, Symantec's infringement of the '533 patent is willful and deliberate, entitling Intellectual Ventures II to increased damages under 35 U.S.C. § 284 and to attorneys' fees and costs incurred in prosecuting this action under 35 U.S.C. § 285. Some of Symantec's own patents, such as U.S. Patent Nos. 6,144,992 and 7,664,983, indicate Symantec had actual knowledge of the '533 patent before this suit was filed. Nevertheless, Symantec has infringed and continues to infringe the '533 patent despite an objectively high likelihood that its actions constitute infringement.

COUNT II – INFRINGEMENT OF U.S. PATENT NO. 6,598,131

16. On July 22, 2003, United States Patent No. 6,598,131 ("the '131 patent") was duly and legally issued for an invention entitled "Data image management via emulation of non-volatile storage device." Intellectual Ventures II is the owner of the '131 patent and holds all rights and interests in the '131 patent. A true and correct copy of the '131 patent is attached hereto as Exhibit B.

17. Symantec has infringed and continues to infringe one or more claims of the '131 patent by its manufacture, use, sale, importation, and/or offer for sale of certain storage software products, including but not limited to Symantec's Replicator and Veritas Volume Replicator products. Symantec is liable for its infringement of the '131 patent pursuant to 35 U.S.C. § 271.

18. Symantec actively, knowingly, and intentionally has induced, and continues to actively, knowingly, and intentionally induce, infringement of the '131 patent by making, using, offering for sale, importing, and selling certain storage software products, including but not limited to Symantec's Replicator and Veritas Volume Replicator products, as well as by contracting with others to use, market, sell, offer to sell, and import such products, all with

knowledge of the '131 patent and its claims; with knowledge that its customers and end users will use, market, sell, offer to sell, and import such products; and with the knowledge and the specific intent to encourage and facilitate those infringing sales and uses of such products through the creation and dissemination of promotional and marketing materials, instructional materials, product manuals, and technical materials.

19. Symantec also has contributed to the infringement by others, including the end users of certain storage software products, including but not limited to Symantec's Replicator and Veritas Volume Replicator products, and continues to contribute to infringement by others, by selling, offering to sell, and importing such products into the United States, knowing that those products constitute a material part of the inventions of the '131 patent, knowing those products to be especially made or adapted to infringe the '131 patent, and knowing that those products are not staple articles or commodities of commerce suitable for substantial non-infringing use.

20. Symantec has had knowledge of and notice of the '131 patent and its infringement since at least, and through, the filing and service of the Complaint and despite this knowledge continues to commit the aforementioned infringing acts. In addition, Symantec has had knowledge of the '131 patent at least since September 30, 2005, when Symantec first referenced the '131 patent in an information disclosure statement during the prosecution of Symantec's U.S. Patent No. 7,673,130, App. No. 11/239,922; May 15, 2007, when Symantec first referenced the '131 patent in an information disclosure statement during the prosecution of Symantec's U.S. Patent No. 7,702,892, App. No. 11/243,129; and March 6, 2008, when Symantec first referenced the '131 patent in an information disclosure statement during the prosecution of Symantec's U.S. Patent No. 7,496,920, App. No. 11/767,666.

21. Symantec's acts of infringement have caused damage to Intellectual Ventures II, and Intellectual Ventures II is entitled to recover from Symantec the damages it has sustained as a result of Symantec's wrongful acts in an amount subject to proof at trial. Symantec's infringement of Intellectual Ventures II's exclusive rights under the '131 patent will continue to damage Intellectual Ventures II, causing irreparable harm for which there is no adequate remedy at law, unless enjoined by this Court.

22. Upon information and belief, Symantec's infringement of the '131 patent is willful and deliberate, entitling Intellectual Ventures II to increased damages under 35 U.S.C. § 284 and to attorneys' fees and costs incurred in prosecuting this action under 35 U.S.C. § 285. Many of Symantec's own patents, such as U.S. Patent Nos. 6,986,033, 7,058,797, 7,069,428, 7,222,229, 7,496,920, 7,506,151, 7,577,806, 7,577,807, 7,584,337, 7,631,120, 7,673,130, 7,702,892, 7,725,667, 7,725,760, 7,730,222, 7,792,125, 7,827,362, 7,836,292, 7,895,424, 7,904,428, 7,991,748, 8,037,289, 8,051,028, and 8,095,488, indicate Symantec had actual knowledge of the '131 patent before this suit was filed. Nevertheless, Symantec has infringed and continues to infringe the '131 patent despite an objectively high likelihood that its actions constitute infringement.

COUNT III – INFRINGEMENT OF U.S. PATENT NO. 6,732,359

23. On May 4, 2004, United States Patent No. 6,732,359 ("the '359 patent") was duly and legally issued for an invention entitled "Application process monitor." Intellectual Ventures I is the owner of the '359 patent and holds all rights and interests in the '359 patent. A true and correct copy of the '359 patent is attached hereto as Exhibit C.

24. Symantec has infringed and continues to infringe one or more claims of the '359 patent by its manufacture, use, sale, importation, and/or offer for sale of certain availability and

clustering software products, including but not limited to Symantec's ApplicationHA products. Symantec is liable for its infringement of the '359 patent pursuant to 35 U.S.C. § 271.

25. Symantec actively, knowingly, and intentionally has induced, and continues to actively, knowingly, and intentionally induce, infringement of the '359 patent by making, using, offering for sale, importing, and selling certain availability and clustering software products, including but not limited to Symantec's ApplicationHA products, as well as by contracting with others to use, market, sell, offer to sell, and import such products, all with knowledge of the '359 patent and its claims; with knowledge that its customers and end users will use, market, sell, offer to sell, and import such products; and with the knowledge and the specific intent to encourage and facilitate those infringing sales and uses of such products through the creation and dissemination of promotional and marketing materials, instructional materials, product manuals, and technical materials.

26. Symantec also has contributed to the infringement by others, including the end users of certain availability and clustering software products, including but not limited to Symantec's ApplicationHA products, and continues to contribute to infringement by others, by selling, offering to sell, and importing such products into the United States, knowing that those products constitute a material part of the inventions of the '359 patent, knowing those products to be especially made or adapted to infringe the '359 patent, and knowing that those products are not staple articles or commodities of commerce suitable for substantial non-infringing use.

27. Symantec has had knowledge of and notice of the '359 patent and its infringement since at least, and through, the filing and service of the Complaint and despite this knowledge continues to commit the aforementioned infringing acts.

28. Symantec's acts of infringement have caused damage to Intellectual Ventures I, and Intellectual Ventures I is entitled to recover from Symantec the damages it has sustained as a result of Symantec's wrongful acts in an amount subject to proof at trial. Symantec's infringement of Intellectual Ventures I's exclusive rights under the '359 patent will continue to damage Intellectual Ventures I, causing irreparable harm for which there is no adequate remedy at law, unless enjoined by this Court.

JURY DEMAND

29. Pursuant to Rule 38(b) of the Federal Rules of Civil Procedure, Intellectual Ventures I and Intellectual Ventures II respectfully request a trial by jury on all issues.

PRAYER FOR RELIEF

WHEREFORE, Plaintiffs Intellectual Ventures I and Intellectual Ventures II request entry of judgment in their favor and against Symantec as follows:

- a. Declaring that Symantec has infringed U.S. Patent Nos. 5,537,533, 6,598,131, and 6,732,359;
- b. Awarding the damages arising out of Symantec's infringement of U.S. Patent Nos. 5,537,533, 6,598,131, and 6,732,359, including enhanced damages pursuant to 35 U.S.C. § 284, to Intellectual Ventures I and Intellectual Ventures II, together with prejudgment and post-judgment interest, in an amount according to proof;
- c. Permanently enjoining Symantec and its officers, agents, employees, and those acting in privity with it, from further infringement, including contributory infringement and/or inducing infringement, of U.S. Patent Nos. 5,537,533, 6,598,131, and 6,732,359;
- d. Awarding attorneys' fees pursuant to 35 U.S.C. § 285 or as otherwise permitted by law; and

e. Awarding such other costs and further relief as the Court may deem just and proper.

DATED: March 26, 2013

Respectfully submitted,

FARNAN LLP

By: /s/ Brian E. Farnan

Joseph J. Farnan III (Bar No. 3945)

Brian E. Farnan (Bar No. 4089)

919 North Market Street, 12th Floor

Wilmington, Delaware 19801

Telephone: (302) 777-0300

Facsimile: (302) 777-0301

bfarnan@farnanlaw.com

**ATTORNEYS FOR PLAINTIFFS
INTELLECTUAL VENTURES I LLC and
INTELLECTUAL VENTURES II LLC**

OF COUNSEL:

Parker C. Folse III (WA State Bar No. 24895)

Brooke A.M. Taylor (WA State Bar No. 33190)

Daniel J. Shih (WA State Bar No. 37999)

Jordan Talge (WA State Bar No. 45612)

SUSMAN GODFREY L.L.P.

1201 Third Avenue, Suite 3800

Seattle, Washington 98101-3000

Telephone: (206) 516-3880

Facsimile: (206) 516-3883

pfolse@susmangodfrey.com

btaylor@susmangodfrey.com

dshih@susmangodfrey.com

jtalge@susmangodfrey.com

EXHIBIT A

United States Patent [19]

Patent Number: 5,537,533

Staheli et al.

Date of Patent: Jul. 16, 1996

- [54] SYSTEM AND METHOD FOR REMOTE MIRRORING OF DIGITAL DATA FROM A PRIMARY NETWORK SERVER TO A REMOTE NETWORK SERVER
- [75] Inventors: Vaughn Staheli, Payson; Mike Miller, Pleasant Grove; Sam Francis; Dan Haab, both of Springville; Dan Patten, Orem; Kent Johnson, Spanish Fork, all of Utah
- [73] Assignee: Miralink Corporation, Orem, Utah
- [21] Appl. No.: 289,902
- [22] Filed: Aug. 11, 1994
- [51] Int. Cl.⁶ G06F 12/16
- [52] U.S. Cl. 395/182.03; 395/182.18
- [58] Field of Search 395/575, 200, 395/182.03, 182.04, 182.05, 182.13, 182.18

[56] References Cited

U.S. PATENT DOCUMENTS

4,274,139	6/1981	Hodgkinson et al.	364/200
4,654,819	3/1987	Stiffler et al.	364/900
4,819,159	4/1989	Shipley et al.	364/200
4,959,768	9/1990	Gerhart	364/187
5,051,887	9/1991	Berger et al.	364/200
5,133,065	7/1992	Cheffetz et al.	395/575
5,157,663	10/1992	Major et al.	371/9.1
5,163,131	11/1992	Row et al.	395/200
5,167,032	11/1992	Martin et al.	395/575
5,185,884	2/1993	Martin et al.	395/575
5,212,784	5/1993	Sparks	395/575
5,241,670	8/1993	Eastridge et al.	385/575
5,263,154	11/1993	Eastridge et al.	395/575
5,379,417	1/1995	Lui et al.	395/575

FOREIGN PATENT DOCUMENTS

PCT/US94/
04326 4/2094 WIPO .

OTHER PUBLICATIONS

"Remote StandbyServer", Vinca Corporation, Product Announcement, Sep. 12, 1994.

"Network Server", *Product Depot*, Sep. 1994, pp. 59-60.

"Network Management—Controlling and Optimizing Network Environments", Claudia Graziano, *LAN Times*, Jul. 11, 1994, p. 50.

Emerald Systems U.S.A. & Canada Retail Price List, effective Jul. 1, 1994.

"System Fault Tolerance With No Strings Attached", Bradley F. Shimmin, *LAN Times*, Feb. 28, 1994, vol. 11, Issue 4.

"Playing the Odds", Gary Gunnerson et al., *PC Magazine*, Oct. 26, 1993, v12, n18, pp. 285(28).

"Bulletproofing Your Server", David P. Chernicoff, *PC Week*, Oct. 4, 1993, v10, n39, pN1(3).

"Year-Old NetWare SFT II Still Incomplete: Support for UNIX", Patrick Dryden, *LAN Times*, Oct. 4, 1993, v10, n20, p1(2).

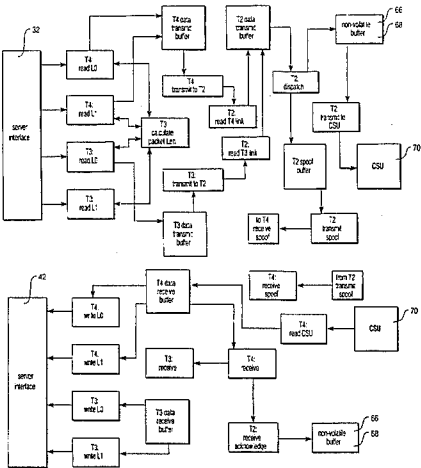
(List continued on next page.)

Primary Examiner—Robert W. Beausoliel, Jr.
Assistant Examiner—Alan M. Fisch
Attorney, Agent, or Firm—Madson & Metcalf

[57] ABSTRACT

A system for remote mirroring of digital data from a primary network server to a remote network server includes a primary data transfer unit and a remote data transfer unit which are connectable with one another by a conventional communication link. The primary data transfer unit sends mirrored data from the primary network server over the link to the remote data transfer unit which is located a safe distance away. Each data transfer unit includes a server interface and a link interface. The server interface is viewed by the network operating system as another disk drive controller. The link interface includes four interconnected parallel processors which perform read and write processes in parallel. The link interface also includes a channel service unit which may be tailored to commercial communications links such as T1, E1, or analog telephone lines connected by modems.

34 Claims, 40 Drawing Sheets



OTHER PUBLICATIONS

- "On The Edge with SFT III", Patrick Dryden, *LAN Times*, Oct. 4, 1993, v10, n20, p1(3).
- "SFT Level III Protection, But Without The Big Price Tag", *LAN Times*, Oct. 4, 1993, v10, n20, p101(7).
- "Third-Party NLM Allows Multiple NIC's In Server", Eric Smalley, *PC Week*, Sep. 27, 1993, v10, n38, p73(1).
- "An SFT Safety Net", Howard Lubert, *LAN Magazine*, Aug. 1993, v8, n8, p99(4).
- "Startup NetGuard Systems Offers Alternative to SFT III", Patrick Dryden, *LAN Times*, Jul. 26, 1993, v10, n14, p7(1).
- "Wired for the Future: 100M-bps Technology to Expand Networking Bandwidth", David P. Chernicoff, *PC Week*, Mar. 15, 1993, v10, n10, p49(3).
- "Research Report—Disk Mirroring with Alternating Deferred Updates", Christos A. Polyzios et al., *Computer Science*, Mar. 4, 1993, pp. 1–25.
- "Pipeline", *InfoWorld*, Mar. 8, 1993, v15, n10, p25(1).
- "Mirrored NetWare Now Sold in Smaller Packs; Novell Ships SFT III", Florence Olson, *Government Computer News*, Feb. 15, 1993, v12, n4, p21(1).
- "Fault-Tolerance App Gains Windows Support: Clone Star's Reflect 4.0 Takes on SFT III", Nico Krohn, *PC Week*, Feb. 15, 1993, v10, n6, p47(2).
- "Finally, Fault-Tolerant LANs: a new Offshoot of NetWare Brings New Reliability, But Only to Limited Applications", Tracey Capen, *Corporate Computing*, Feb. 1993, v2, n2, p45(2).
- "So You Need/Want Fault Tolerance? Don't Jump to Solutions", Paul Merenbloom, *InfoWorld*, Jan. 25, 1993, v15, n4, p38(1).
- "System Fault Tolerance", Rodney Gallie, *InfoWorld*, Jan. 25, 1993, v15, n4, p65(1).
- "Novell Ships Commercial Version of SFT III", Nico Kohn, *PC Week*, Jan. 18, 1993, v10, n2, p18(1).
- "Novell Adds Fault Tolerance to NetWare", *The Seybold Report on Desktop Publishing*, Jan. 6, 1993, v7, n5, p32(1).
- "Tradeoffs in Implementing Primary-Backup Protocols", Navin Budhiraja and Keith Marzullo, Department of Computer Science—Cornell University, Oct. 1992, pp. 1–18.
- "Research Report—Evaluation of Remote Backup Algorithms for Transaction Processing Systems", Christos A. Polyzios, *Computer Science*, Oct. 14, 1992, Department of Computer Science, Stanford University, pp. 1–35.
- "Optimal Primary-Backup Protocols", Navin Budhiraja et al., Department of Computer Science, Cornell University, Aug. 1992, pp. 1–18.
- "Performance of a Parallel Network Backup Manager", James da Silva et al., *Computer Science Technical Report Series*, Apr. 1992, pp. 1–13.
- "A Cached WORM File System", Sean Quinlan, *Software—Practice and Experience*, Dec. 1991, vol. 21(12), pp. 1289–1299.
- "Management of a Remote Backup Copy for Disaster Recovery", Richard P. King et al., *ACM Transactions on Database Systems*, Jun. 1991, vol. 16, No. 2, pp. 338–368.
- "Performance of a Mirrored Disk in a Real-Time Transaction System", Shenze Chen and Don Towsley, *ACM Sigmetrics Performance Evaluation Review*, May 1991, pp. 198–207.
- "A Comparison of Two Approaches to Build File Servers", Anupam Bhide et al., 11th International Conference on Distributed Computing Systems, pp. 616–623.
- "Tandem's Remote Data Facility", Jim Lyon, Computer Conference, 1990, pp. 562–567.
- "Issues in Disaster Recovery", Hector Garcia-Molina and Christos A. Polyzios, Department of Computer Science, Princeton University, Computer Conference 1990, pp. 573–577.
- "Design Approaches for Real-Time Transaction Processing Remote Site Recovery", D. L. Burkes and R. K. Treiber, Data Base Technology Institute, IBM Almaden Research Center, Computer Conference 1990, pp. 568–572.
- "Allocating Primary and Back-up Copies of Databases in Distributed Computer Systems: a Model and Solution Procedures", Hasan Pirkul and Yu-Ping Hou, College of Business, The Ohio State University, *Computers Opns Res.*, 1989, vol. 16, No. 3, pp. 235–245.
- "Message-Optimal Incremental Snapshots", S. Venkatesan, Department of Computer Science, University of Pittsburgh, 9th International Conference on Distributed Computing Systems, 1989, pp. 53–60.
- "PC Network Services for Distributed System Design", Gregory Ennis, Systek, Incorporated, Spring Computer Conference, 1986, pp. 155–160.
- "Multi-Copy Workstation Databases", Fred Maryanski, University of Connecticut, Spring Computer Conference, 1985, pp. 50–53.
- "Tape Backup and Network Storage Management", Emerald Systems Product Overview brochure.
- "XpressSERVE Enterprise Software Highest-Performance, Server-Based Network Backup", Emerald Systems product brochure.
- "Xpress Shadow Software Flexible Server Mirroring", Emerald Systems product brochure.
- "Extra Value XpressSERVE & Xpress Librarian Bundled Solution", Emerald Systems product brochure.
- "4 mm & 8 mm AutoLoaders for Total Automation", Emerald Systems product brochure.
- "PC and LAN Servers", Network Solutions Systems information brochure.
- "Network and Communications Solutions", Network Solutions Systems information brochure.
- Quest Software information brochure.
- "Octopus", P & W Technologies, Inc. product brochure.
- "Octopus Real Time Data Protection", P & W Technologies, Inc. product description.
- "No*Stop-No*Stop-No*Stop—", Nonstop Networks Limited product brochure.
- "Chapter 1: Introduction", *LANshadow 4.0*, pp. 1–37.
- "Netware SFT III 3.11—Disaster Prevention Solution", Networld+Interop94 product brochure.
- "Server Design tailored to SFT III . . .", *InfoWorld*, Feb. 8, 1993, v15, n6, p29(1).

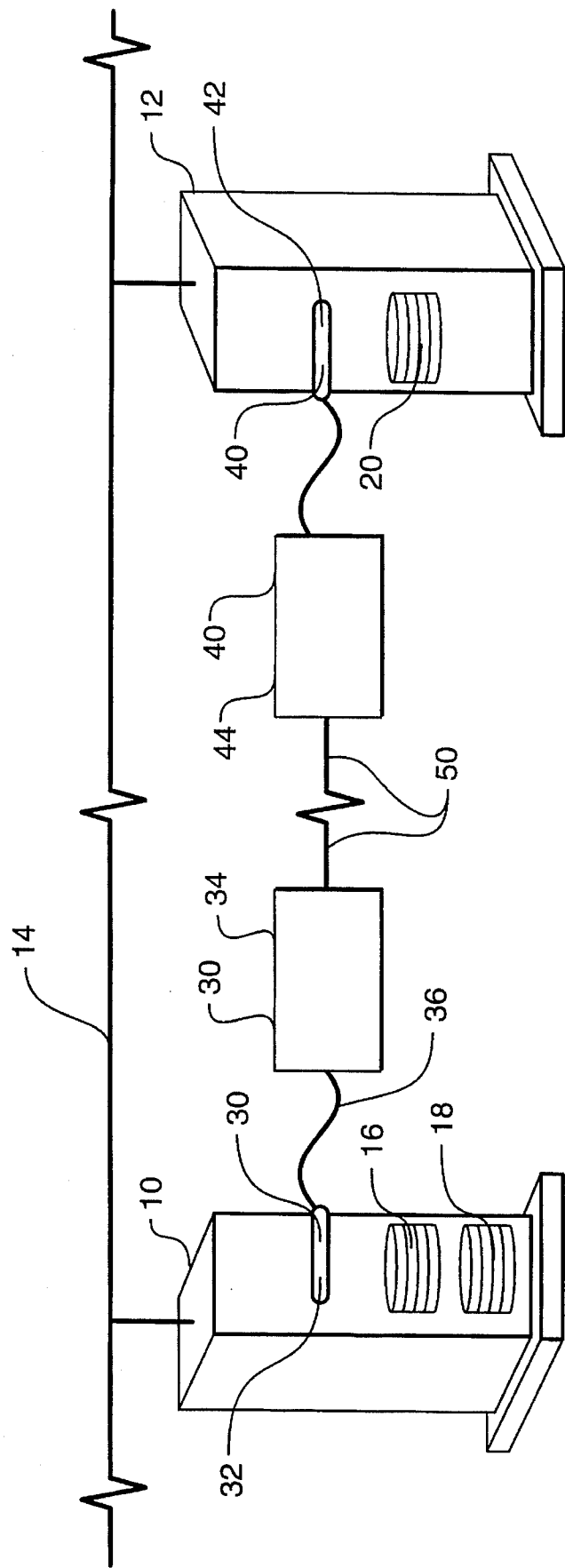


FIG. 1

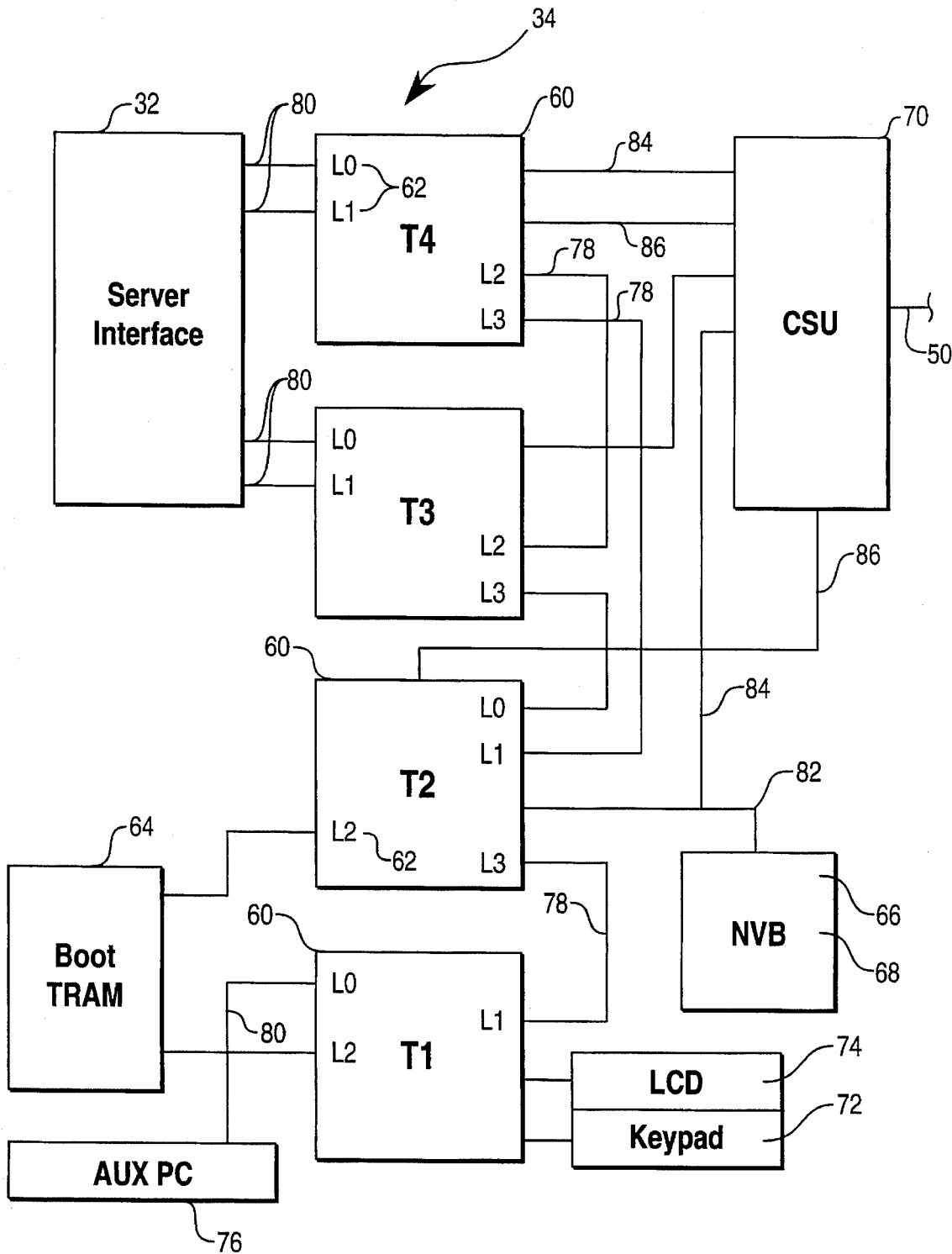


FIG. 2

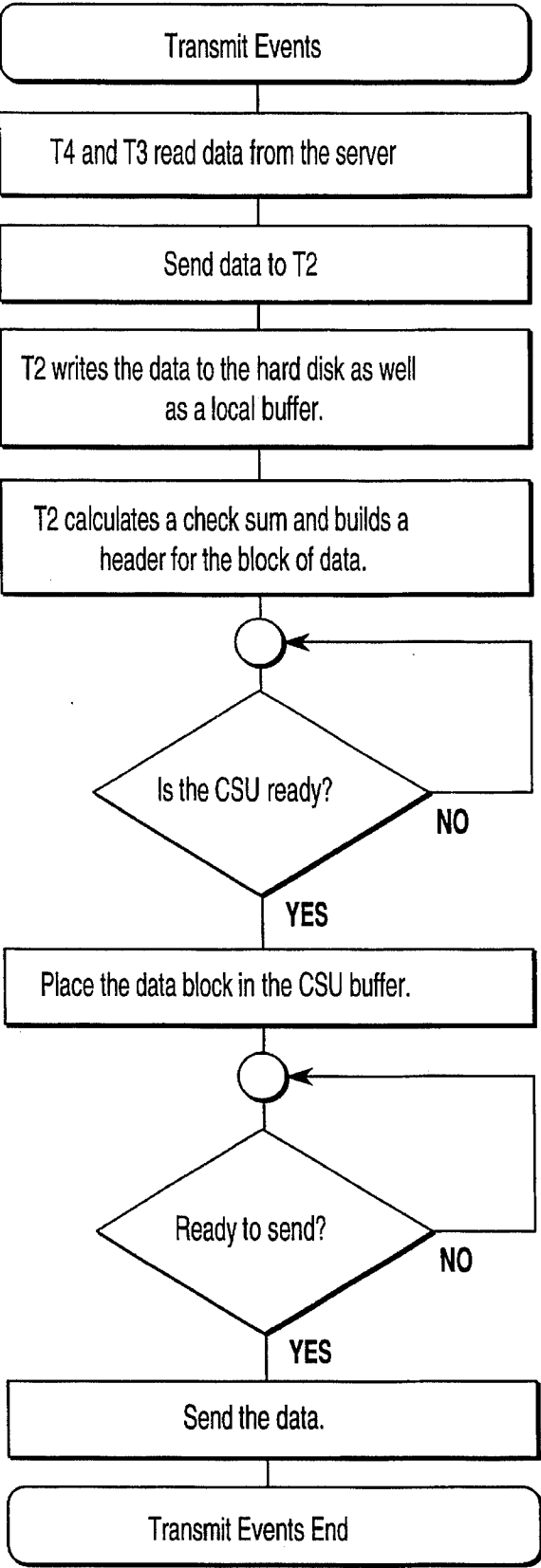


FIG. 3

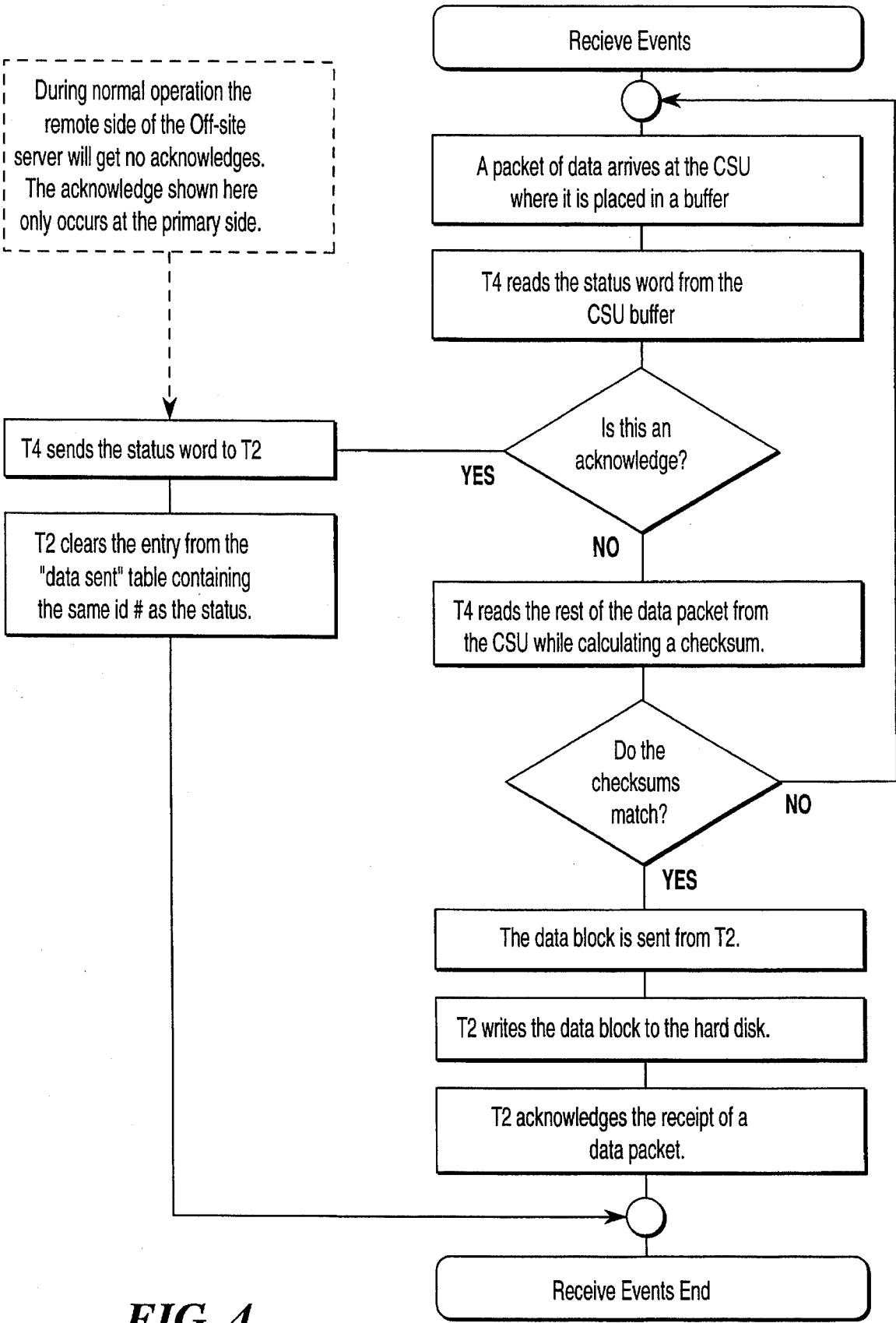


FIG. 4

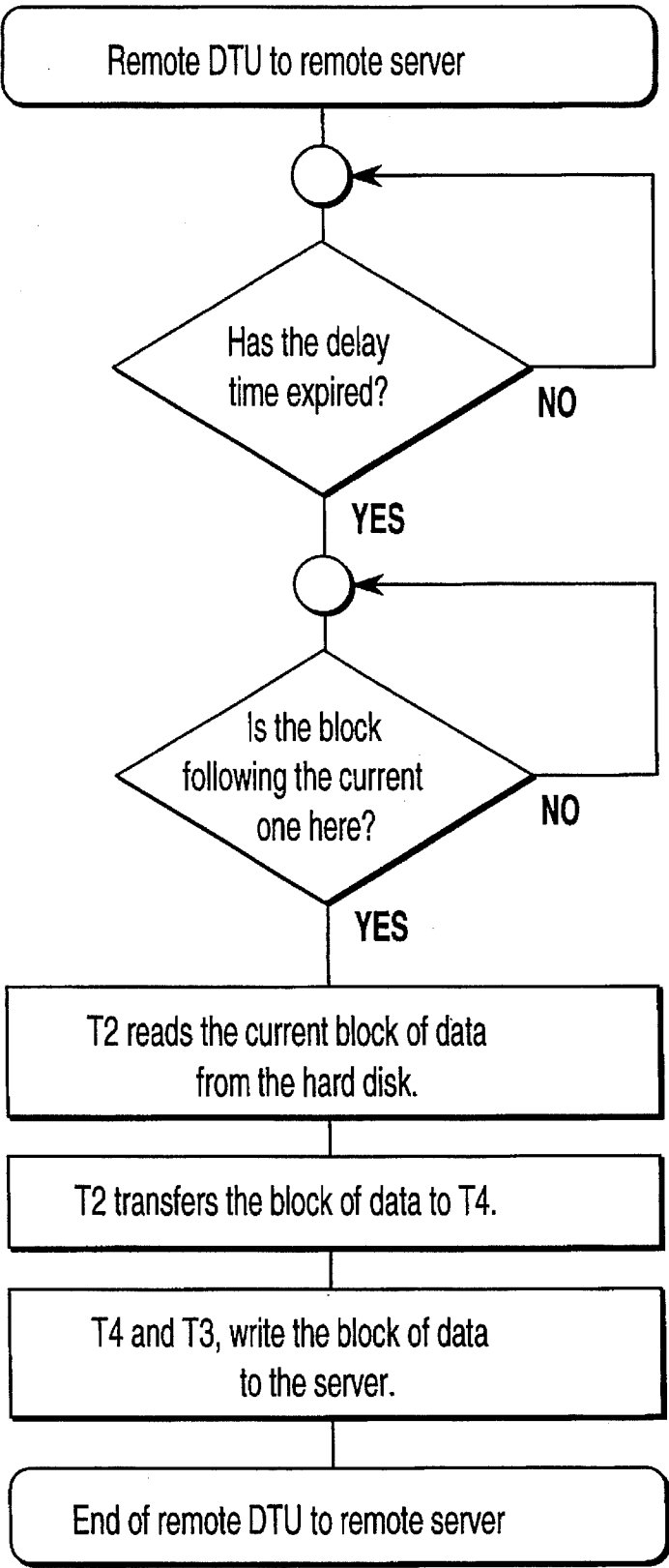


FIG. 5

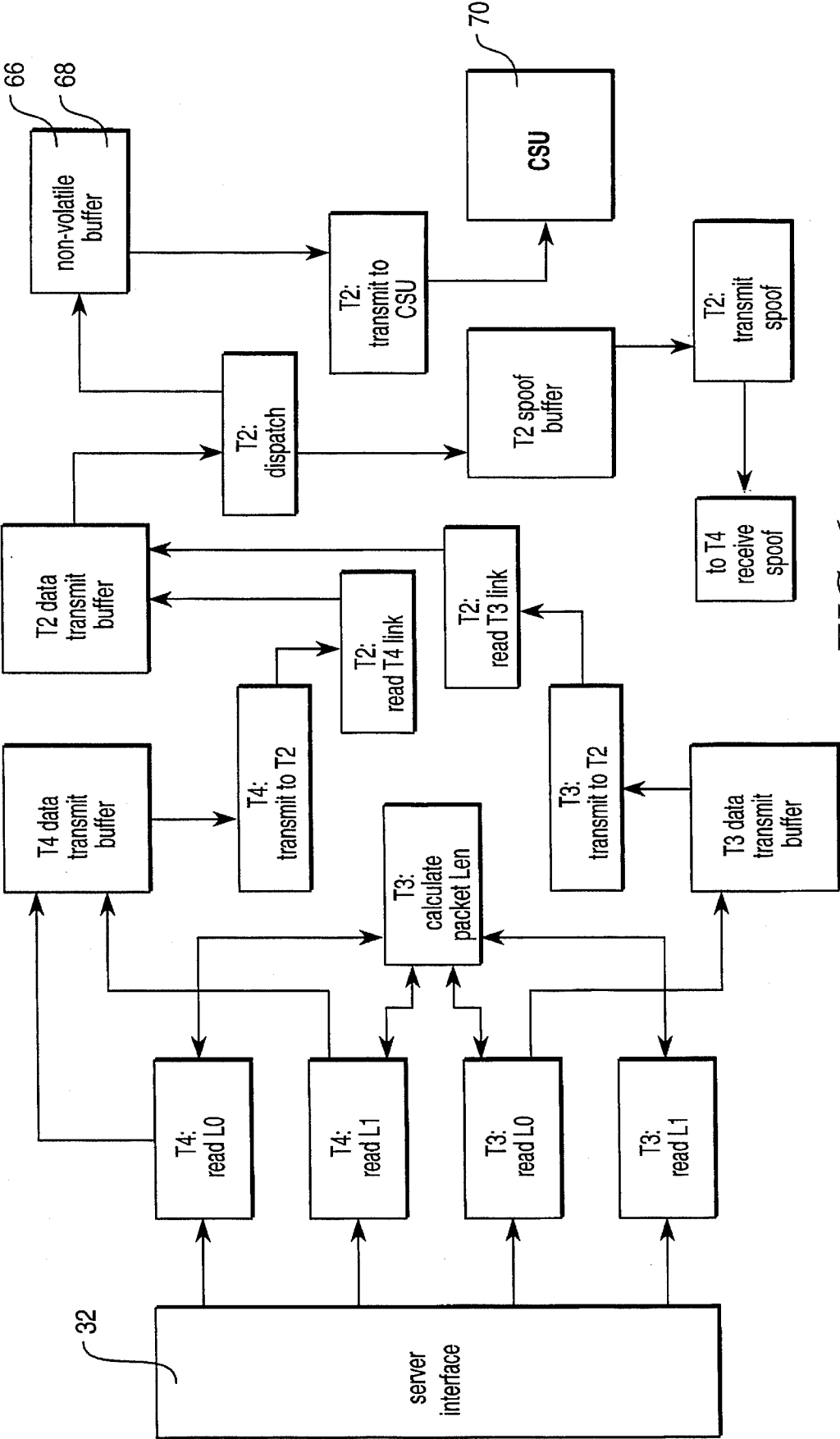


FIG. 6

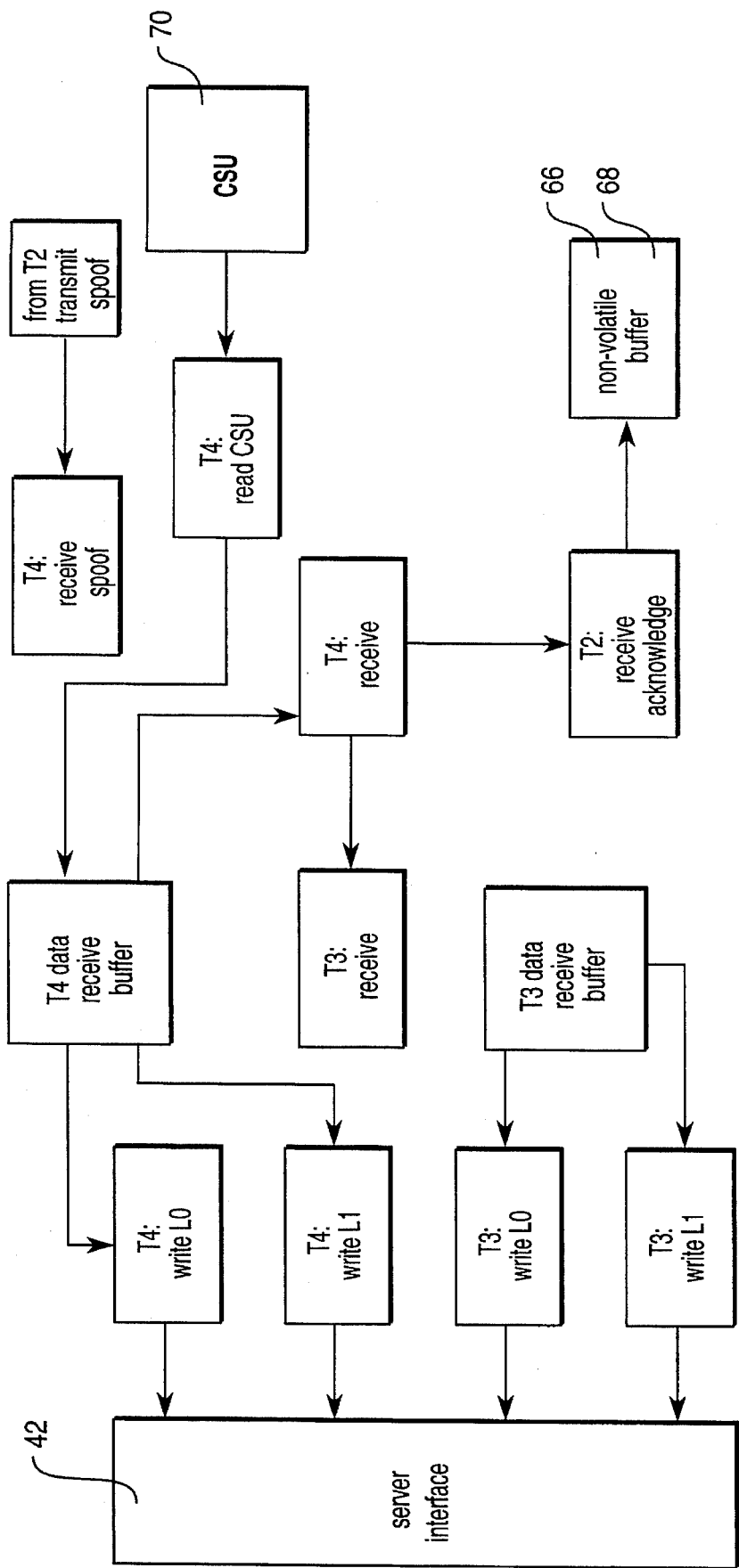


FIG. 7

U.S. Patent

Jul. 16, 1996

Sheet 8 of 40

5,537,533

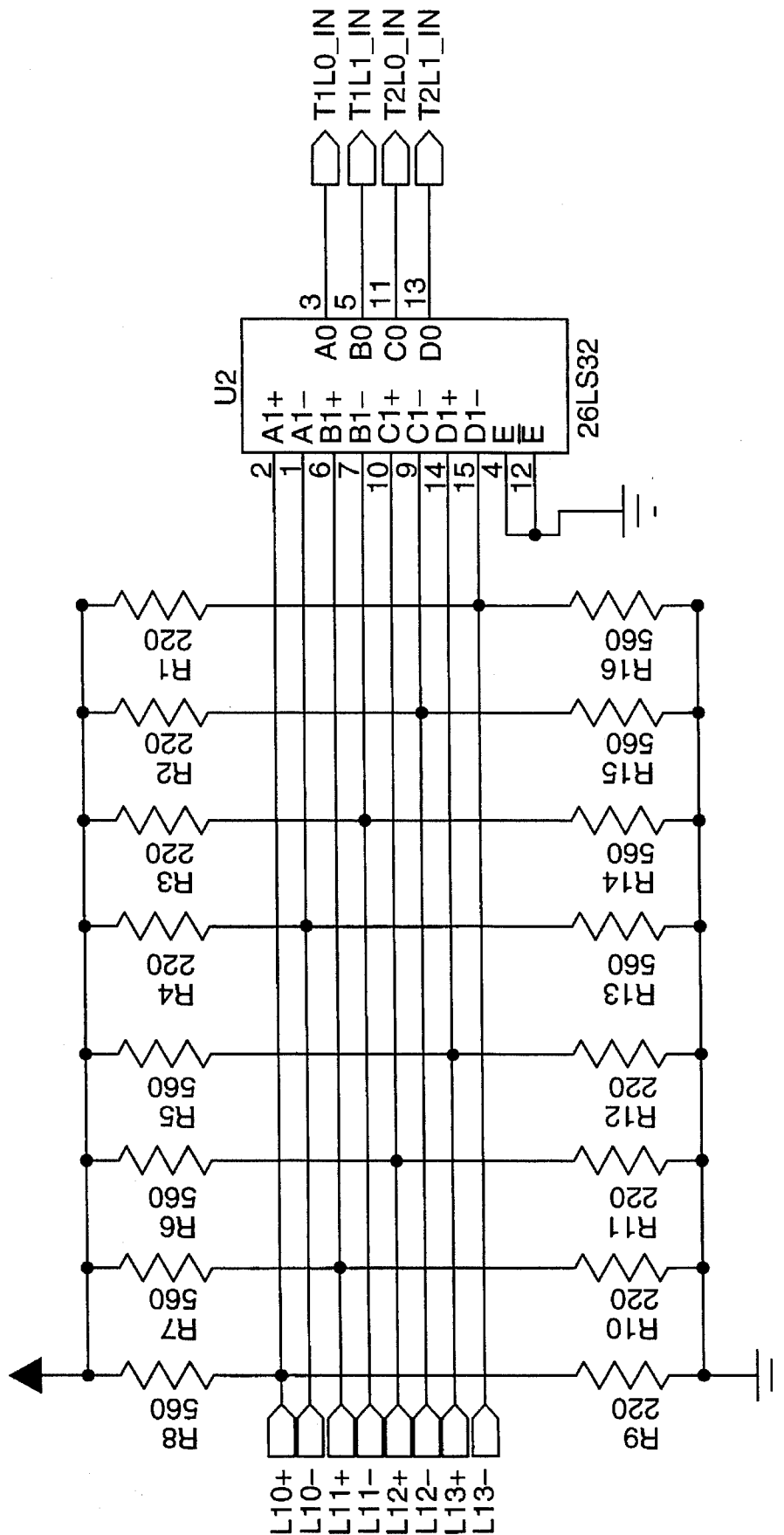


FIG. 8

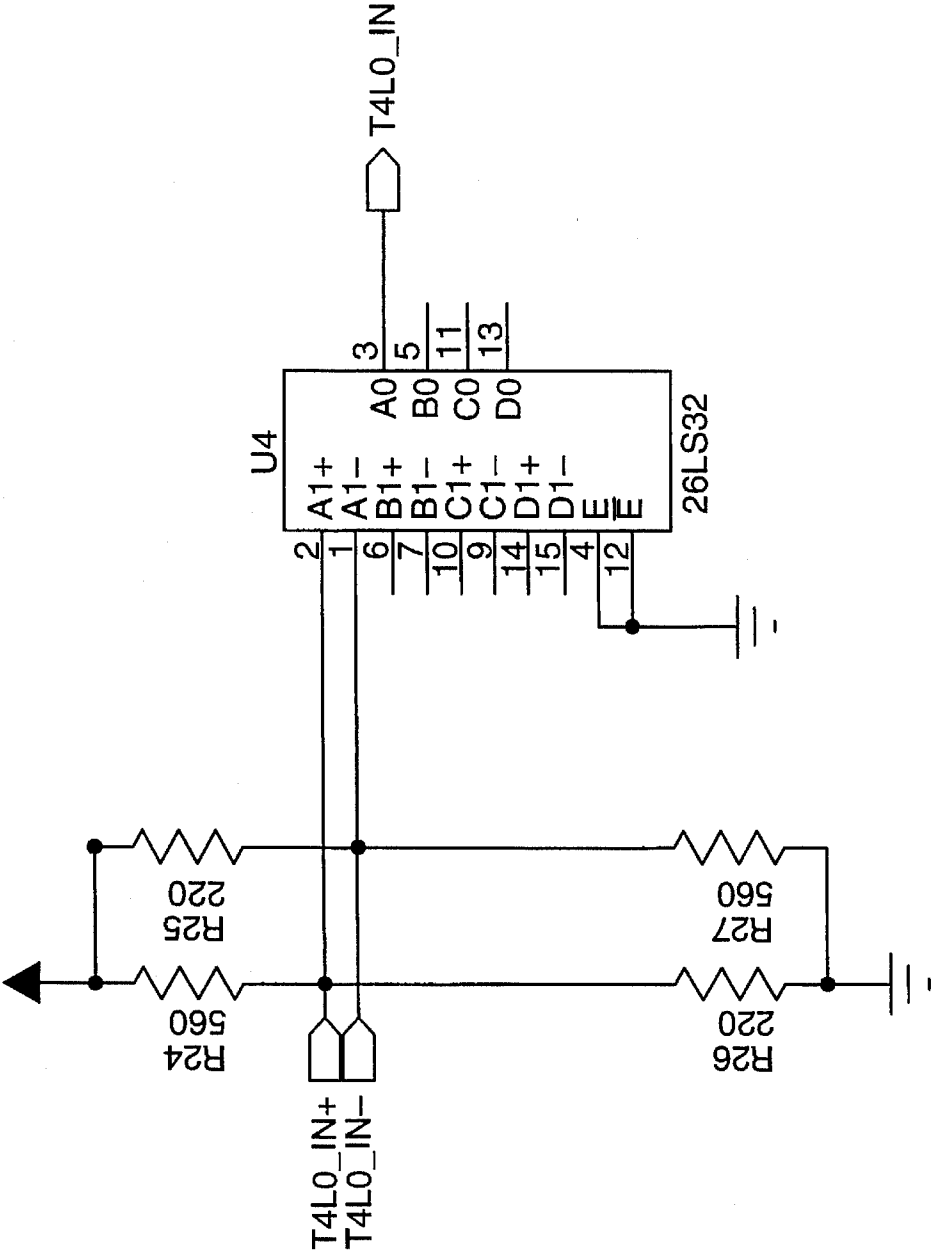


FIG. 9

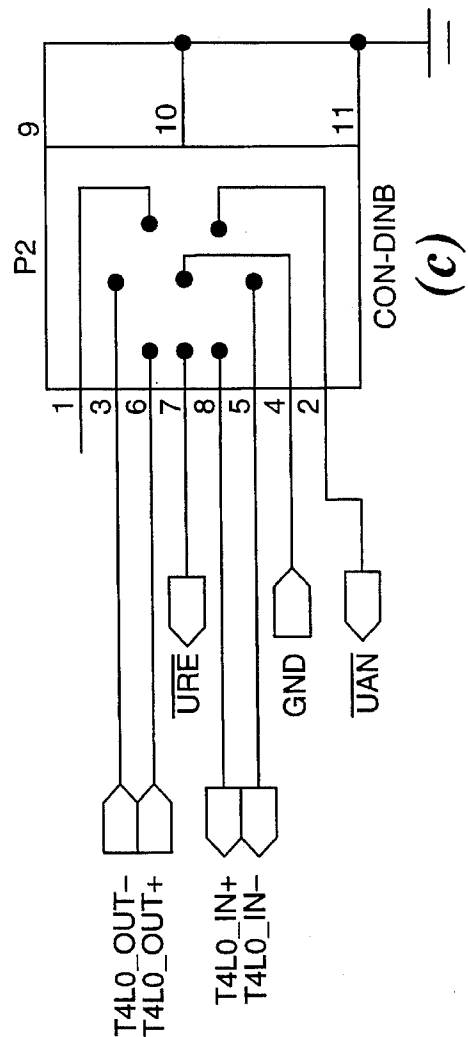
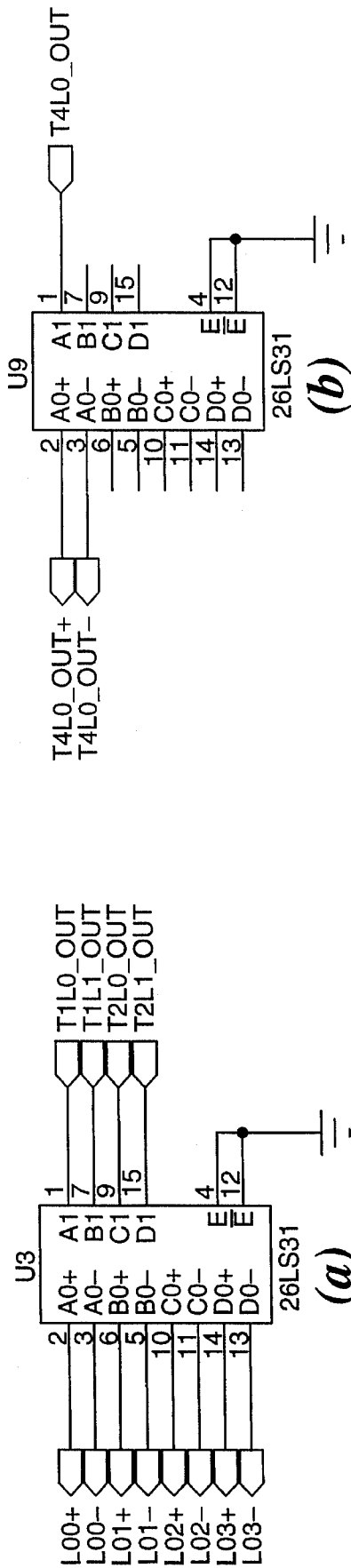


FIG. 10

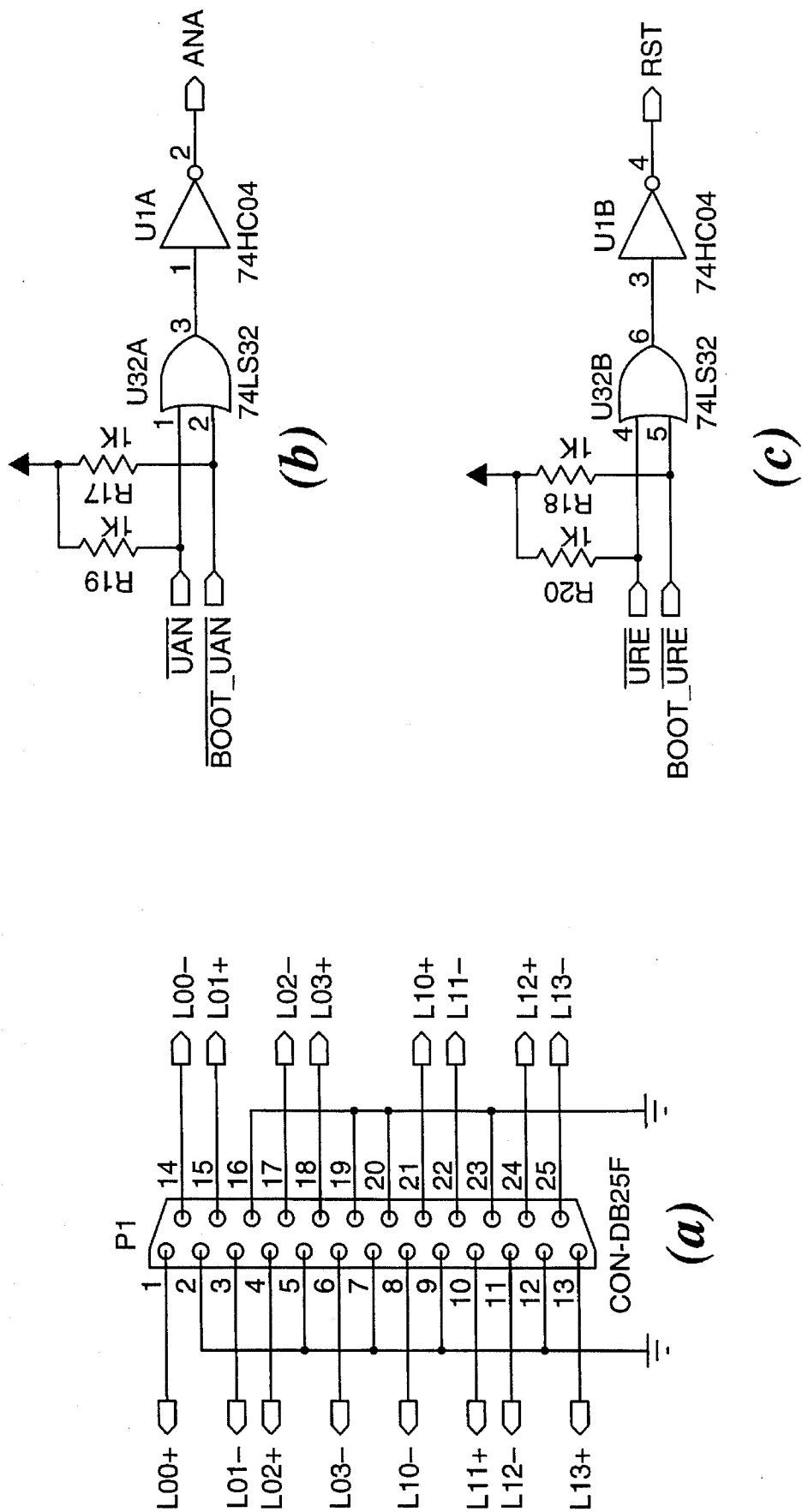


FIG. 11

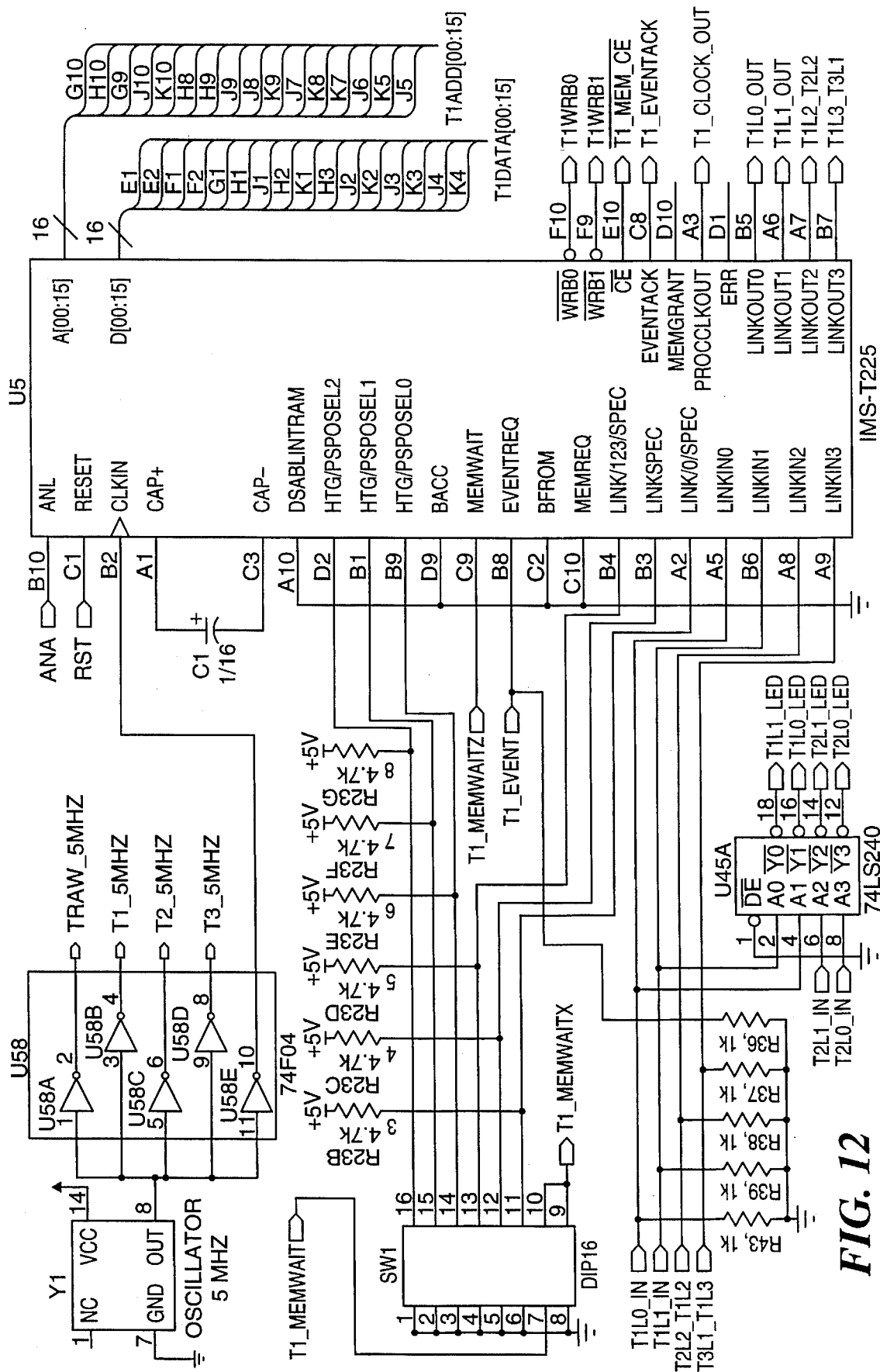


FIG. 12

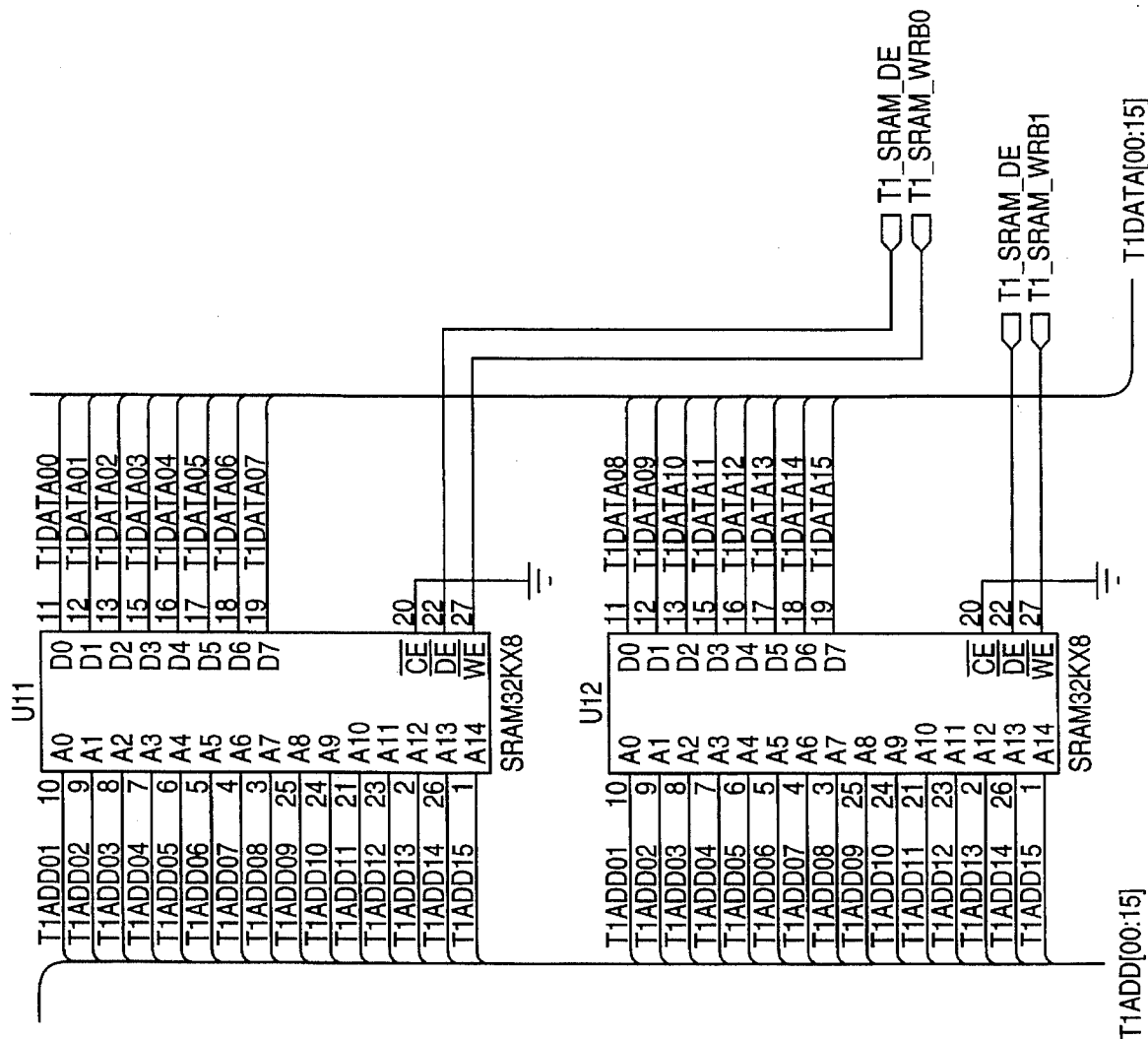


FIG. 13(a)

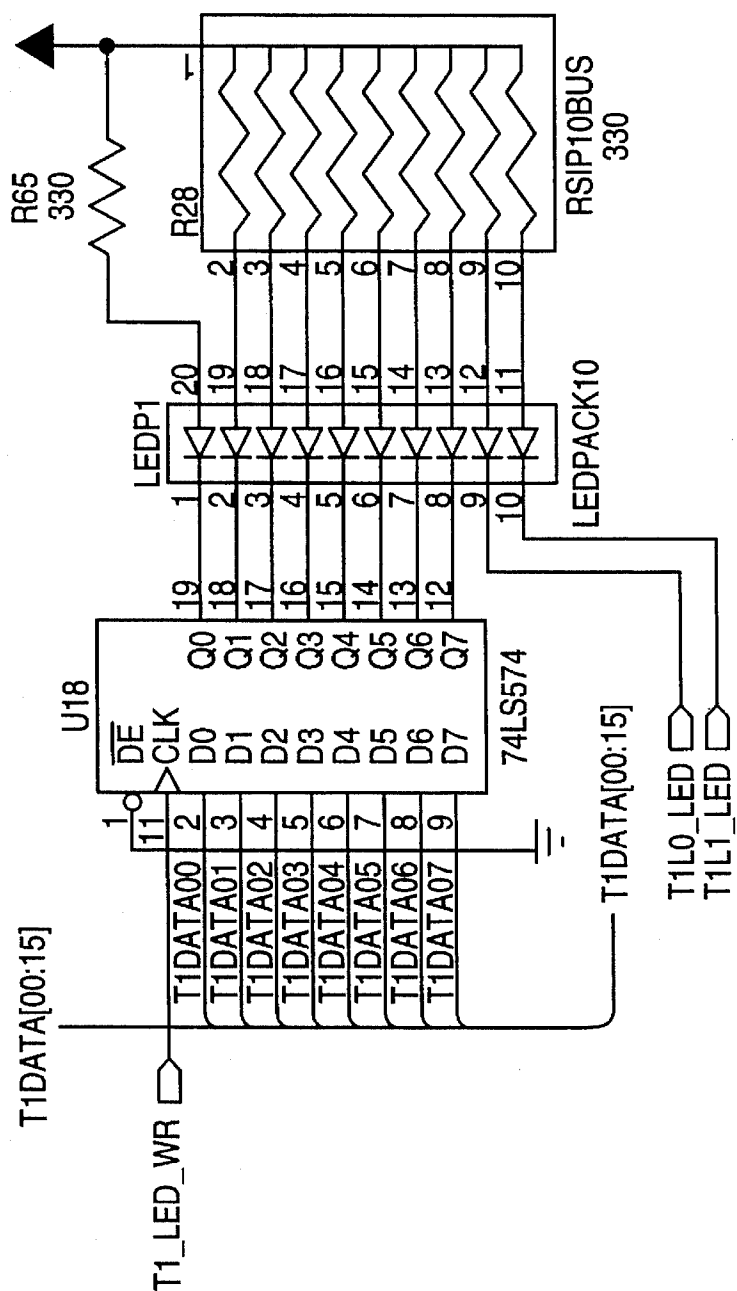
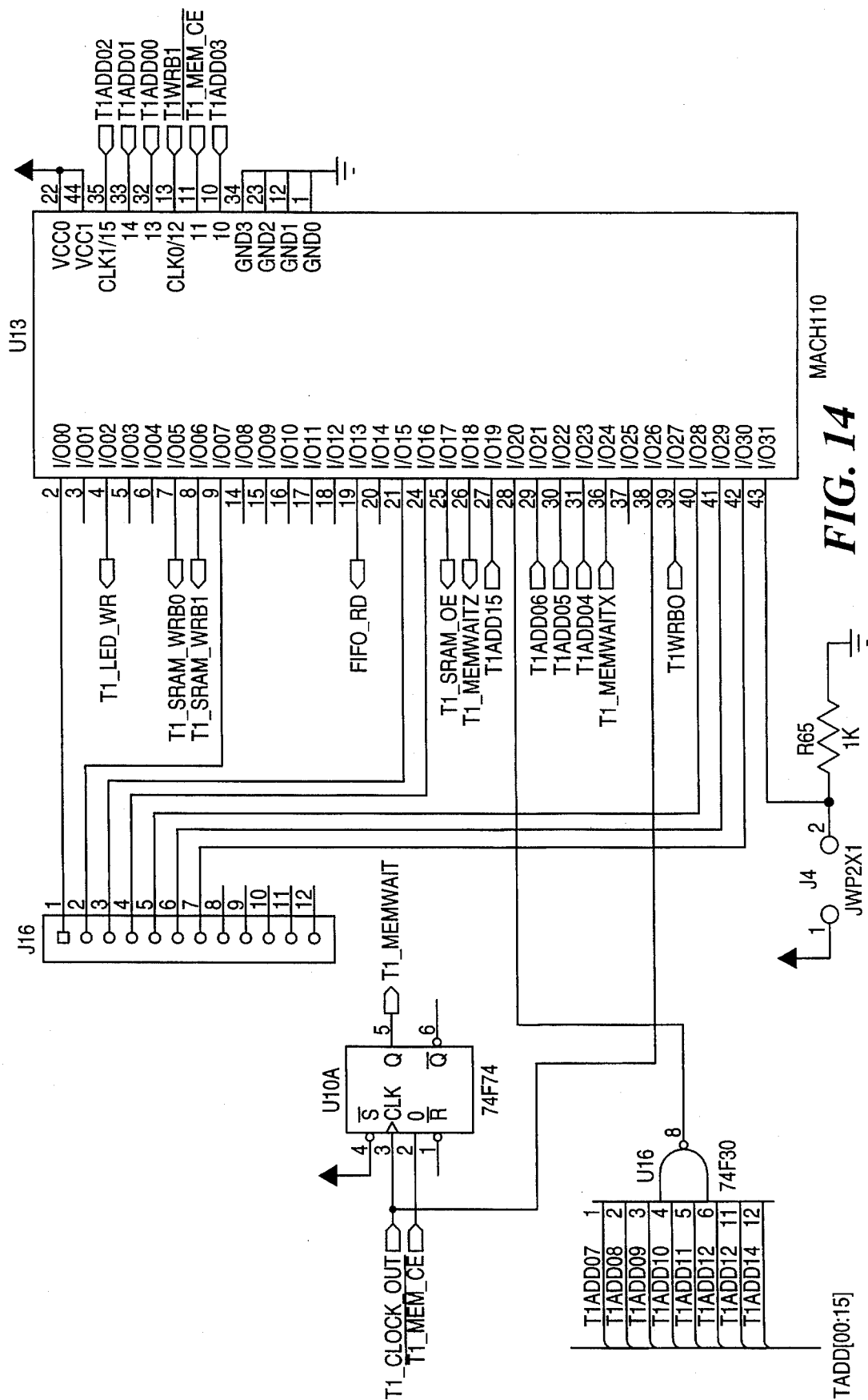


FIG. 13(b)



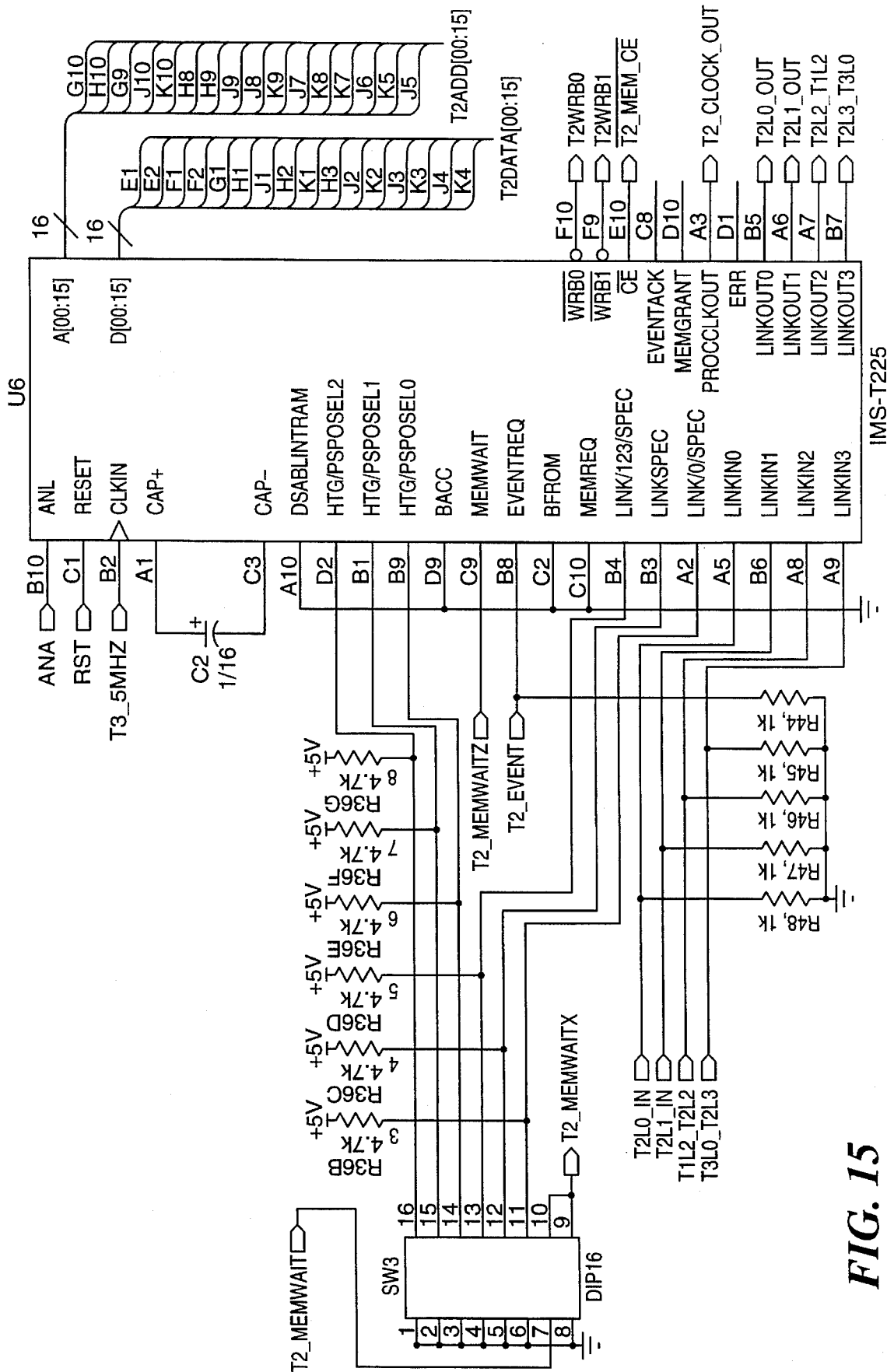


FIG. 15

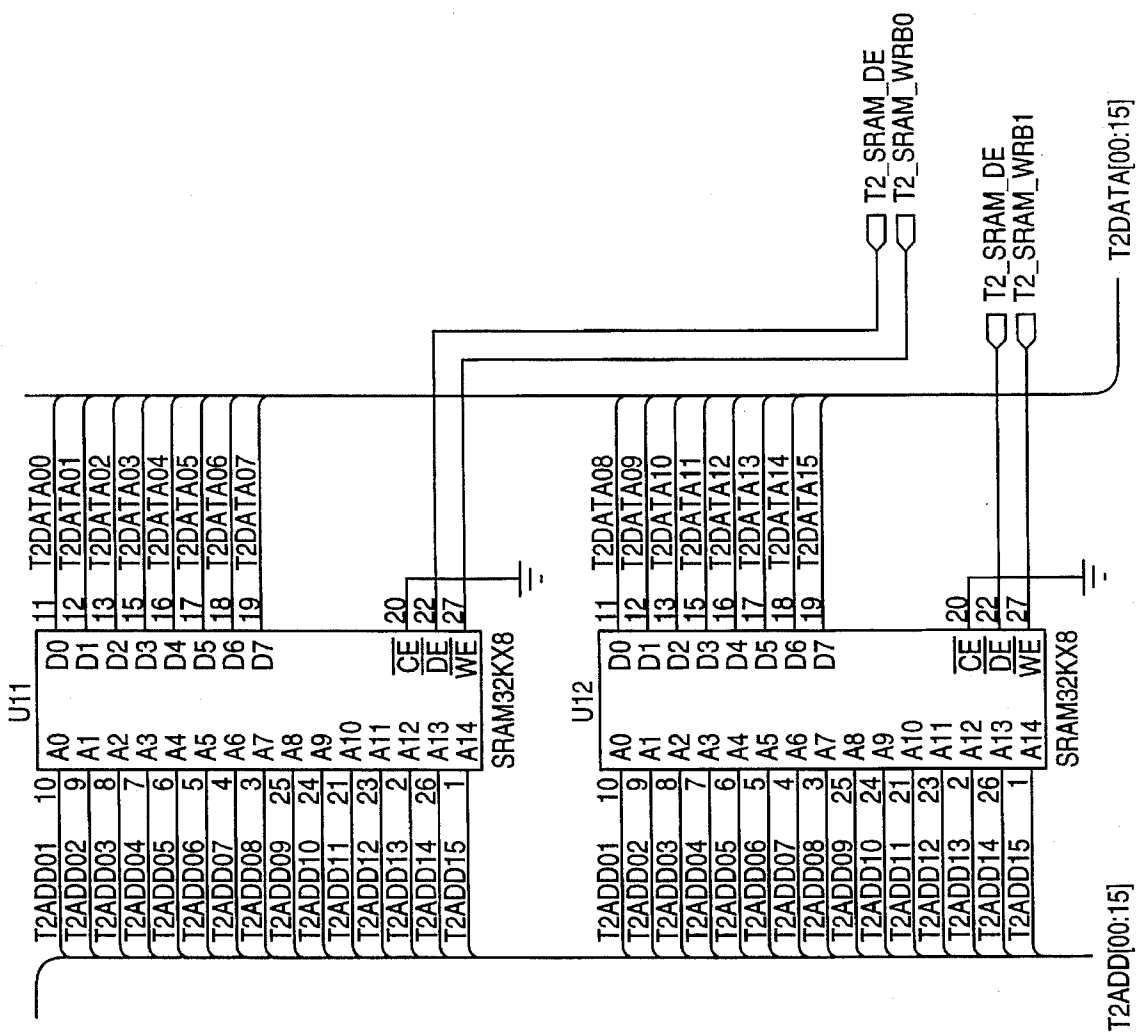


FIG. 16

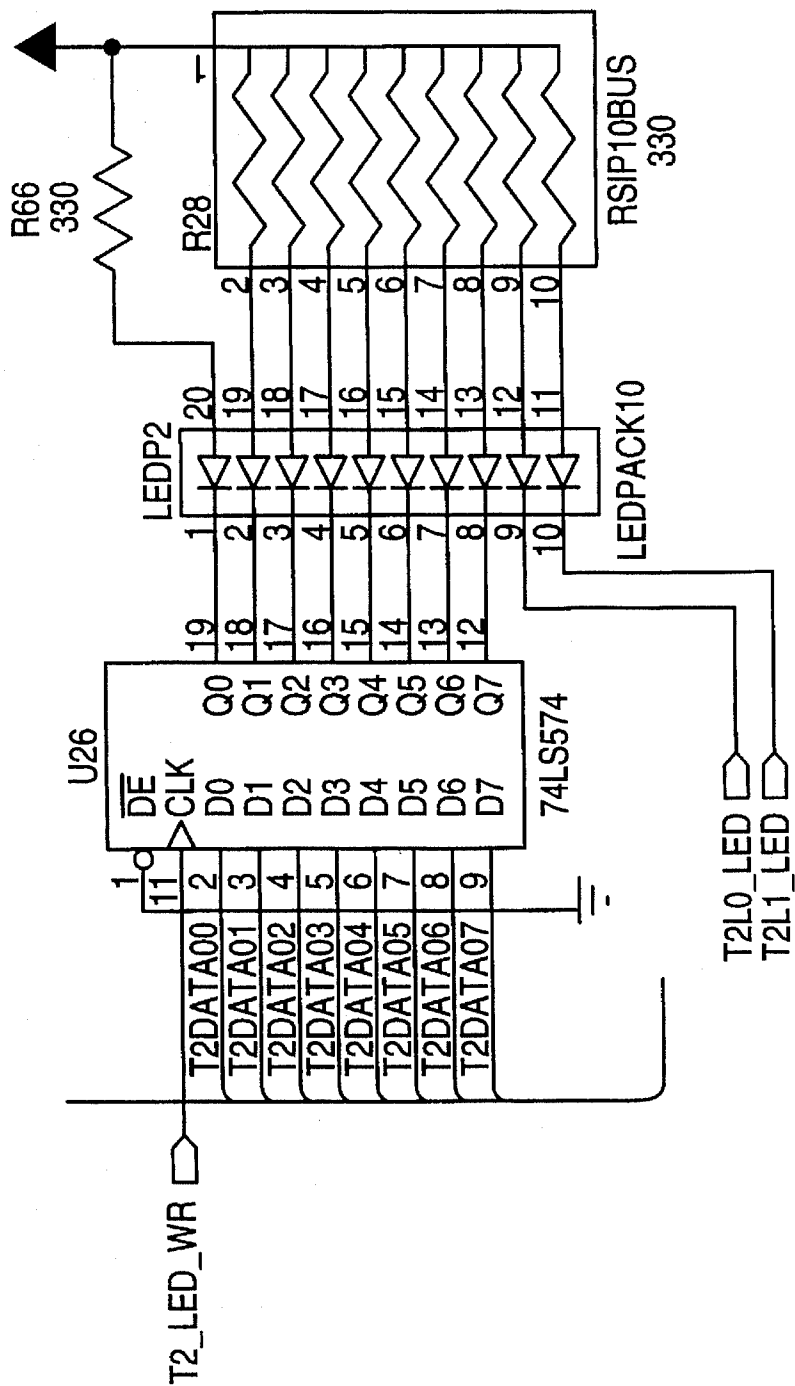
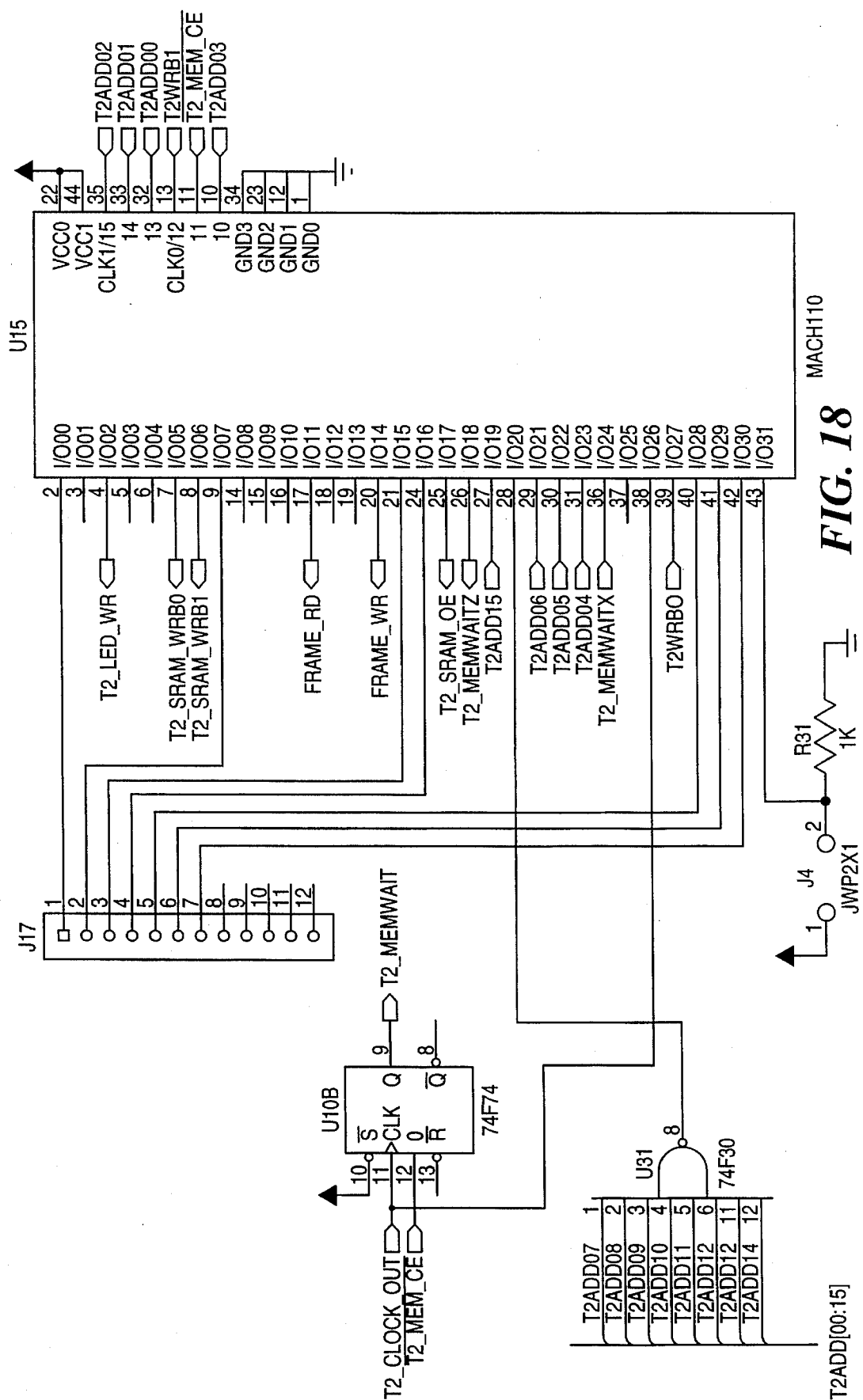


FIG. 17



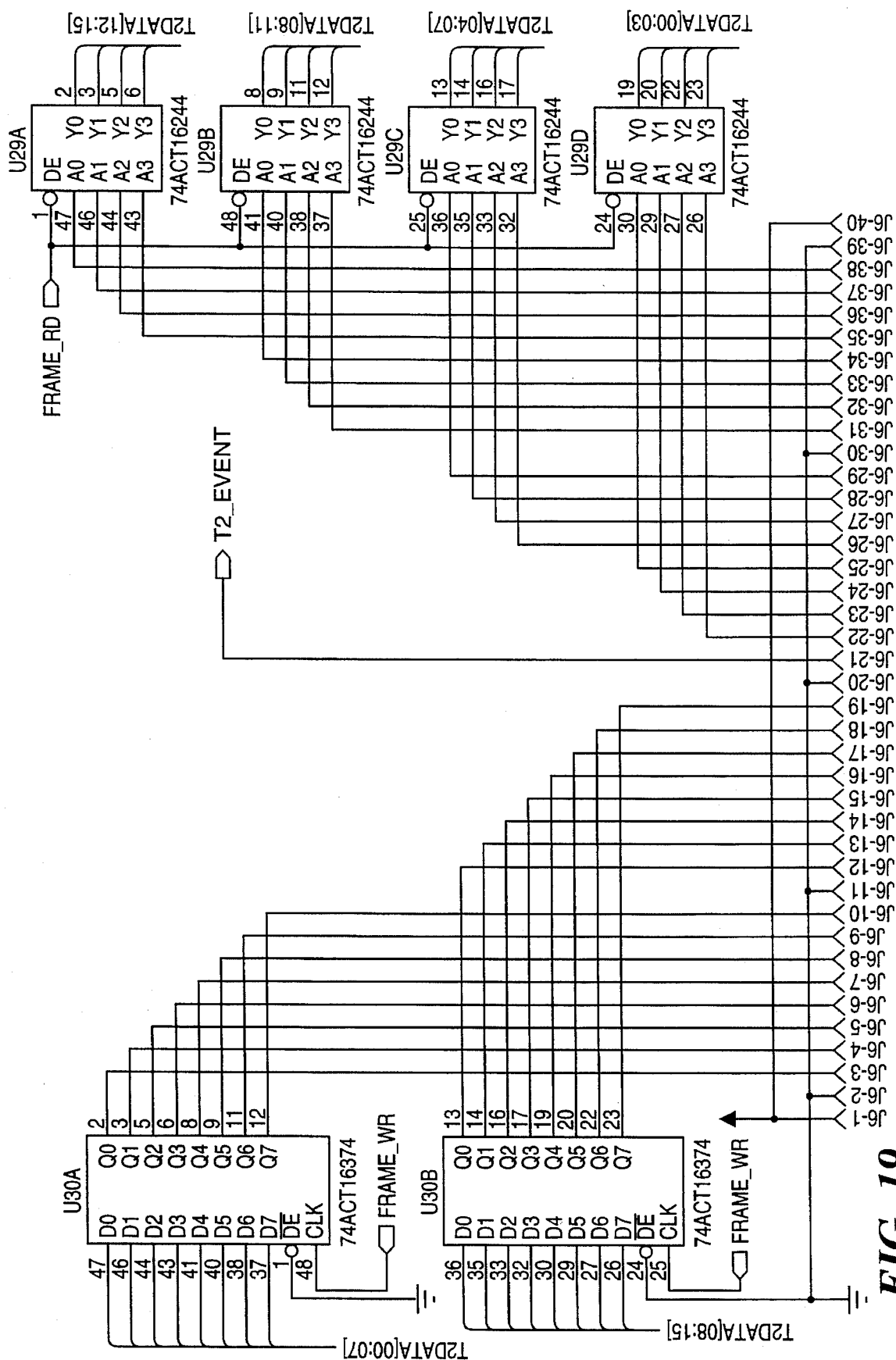


FIG. 19

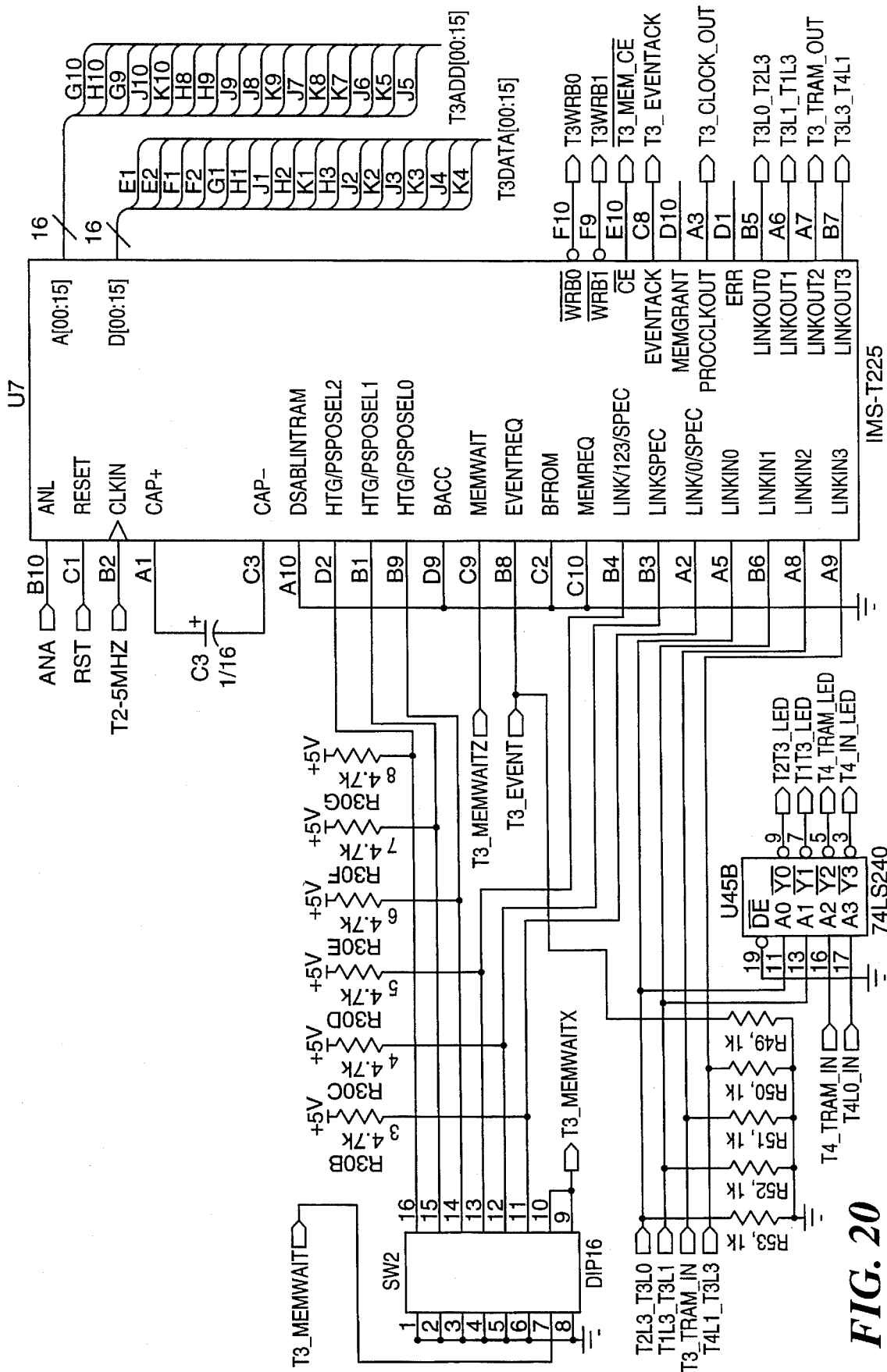


FIG. 20

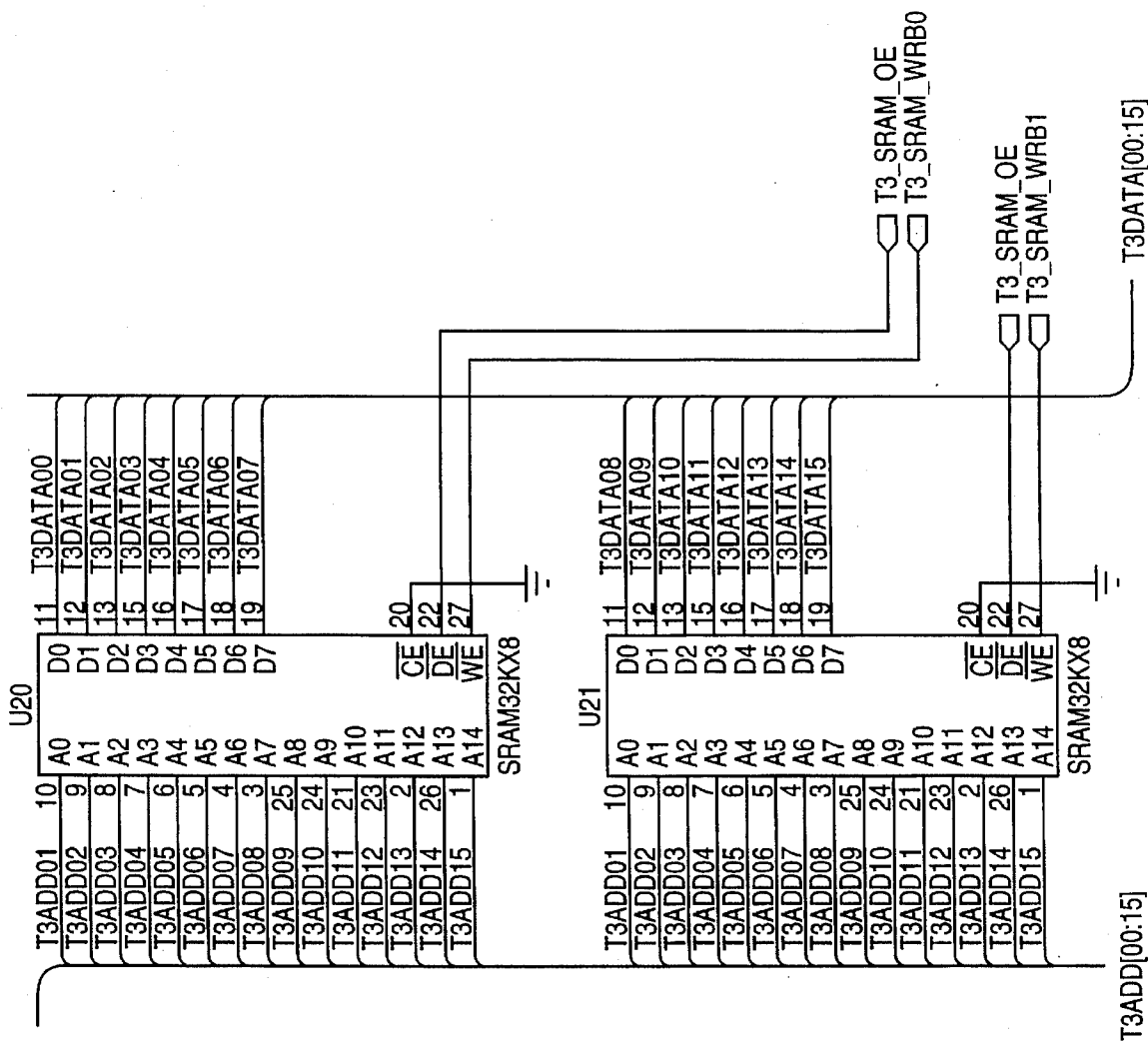


FIG. 21

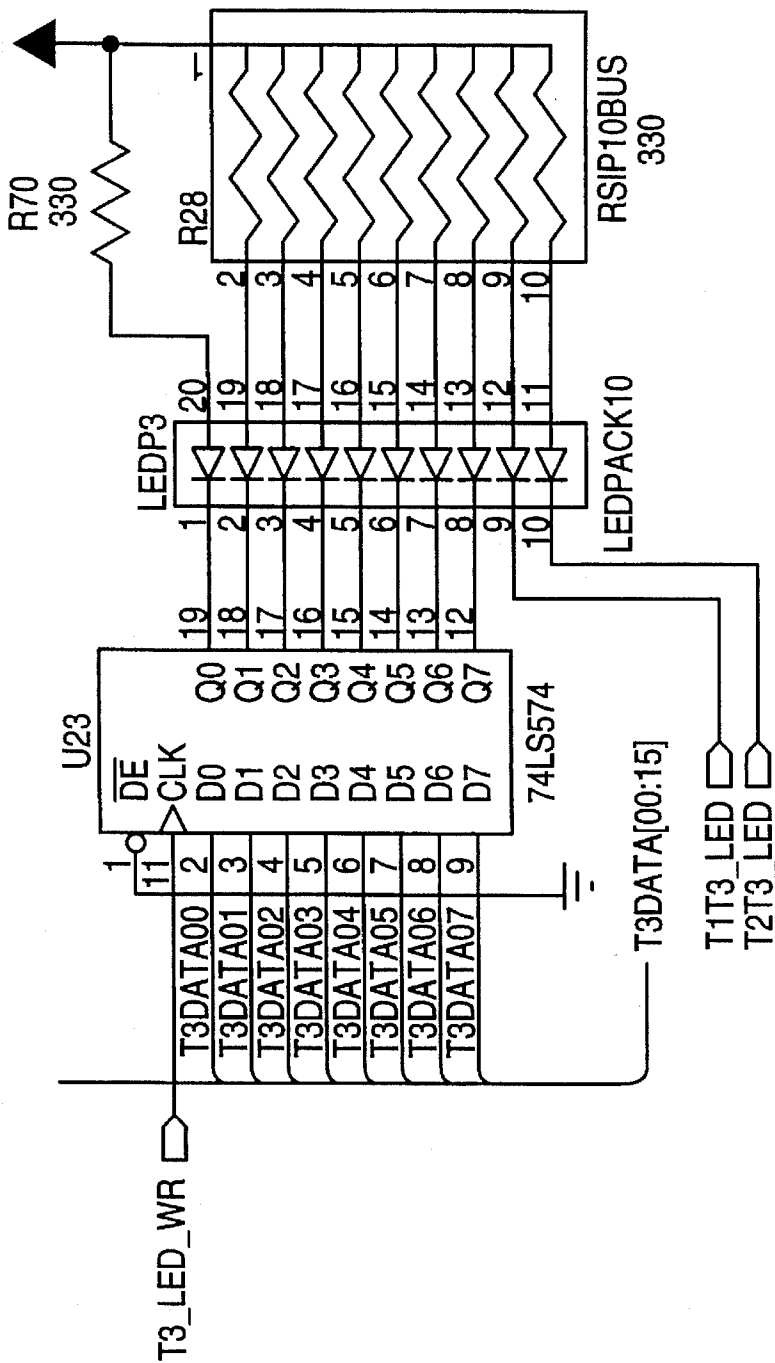
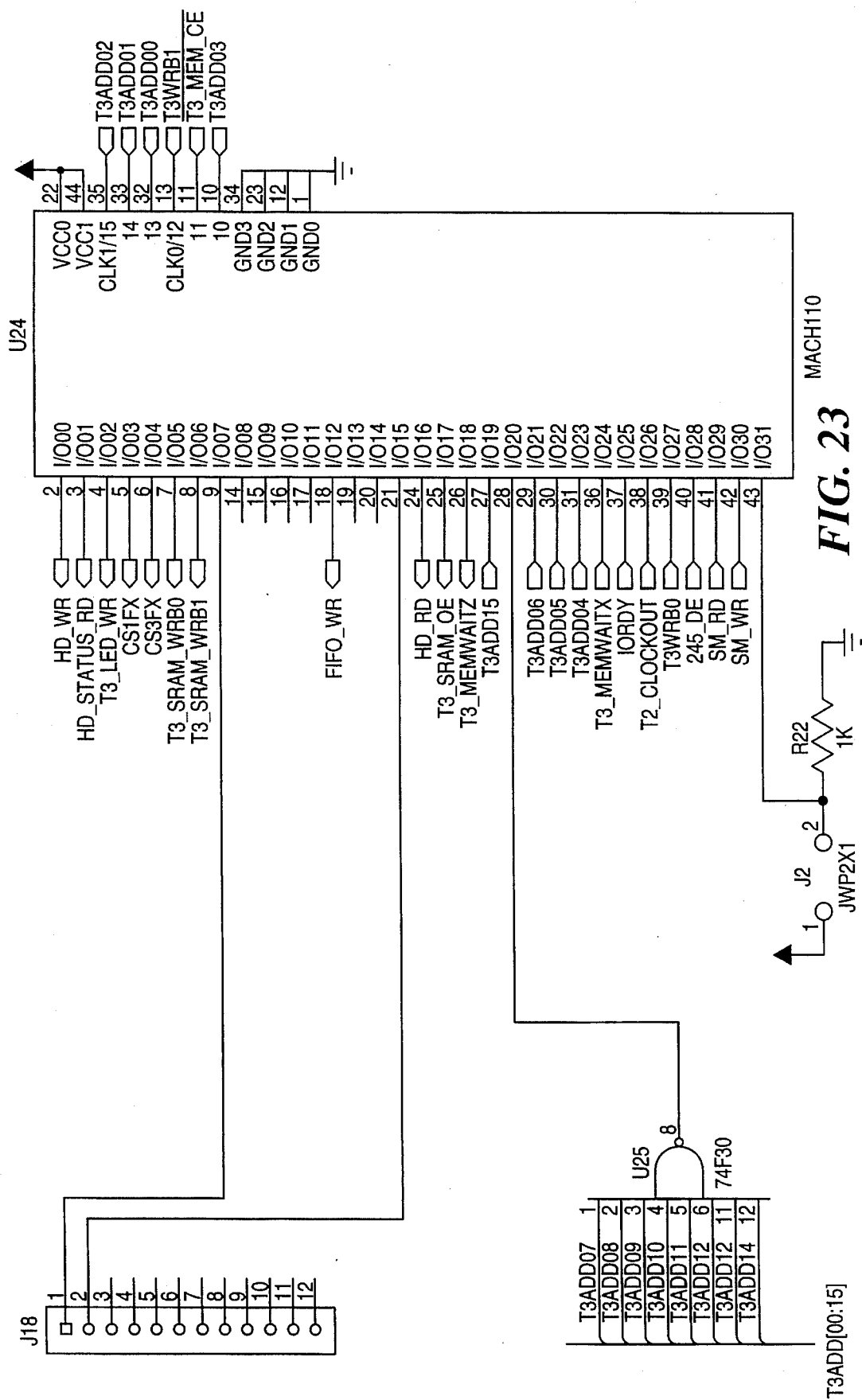


FIG. 22



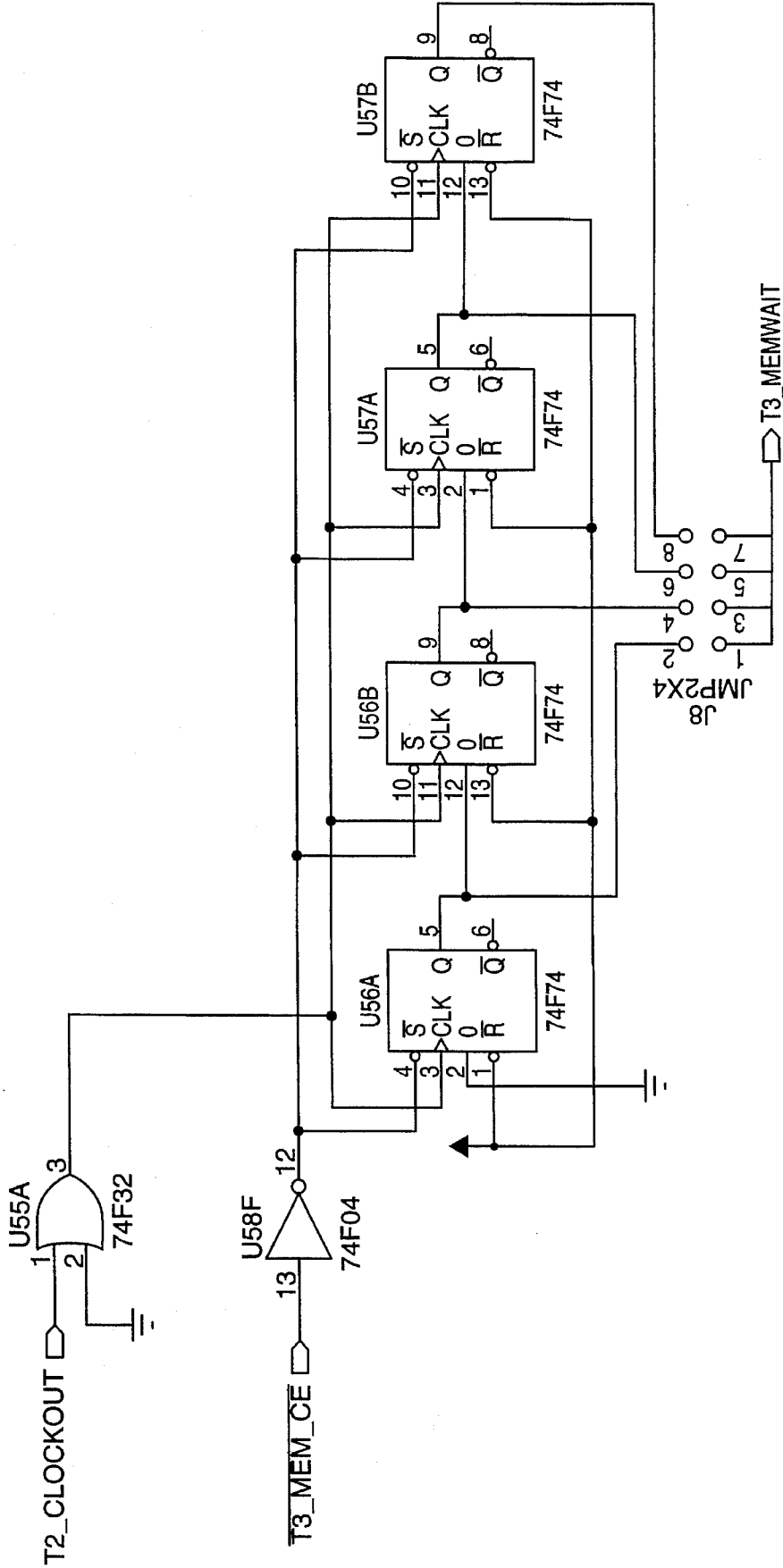


FIG. 24

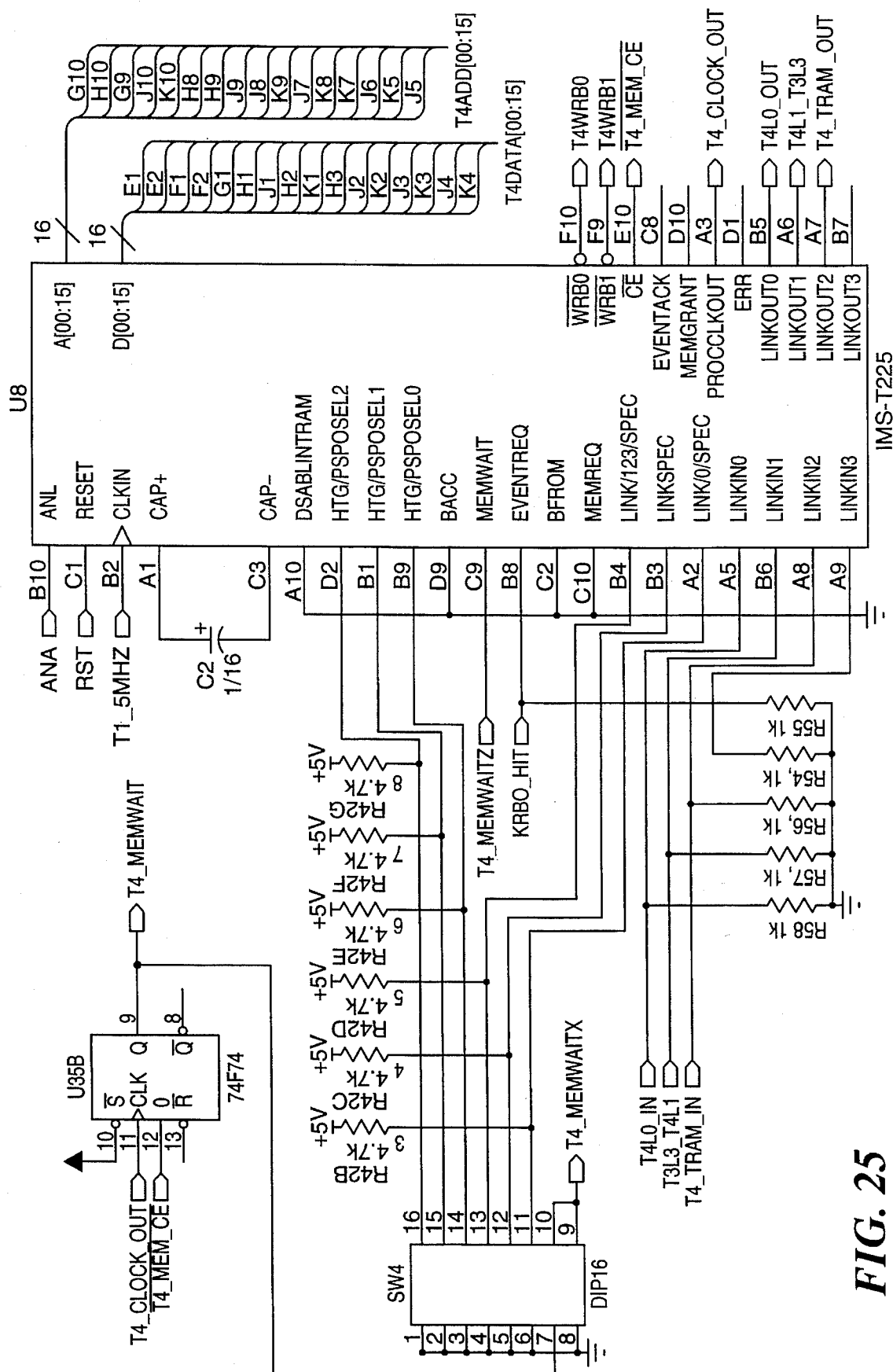


FIG. 25

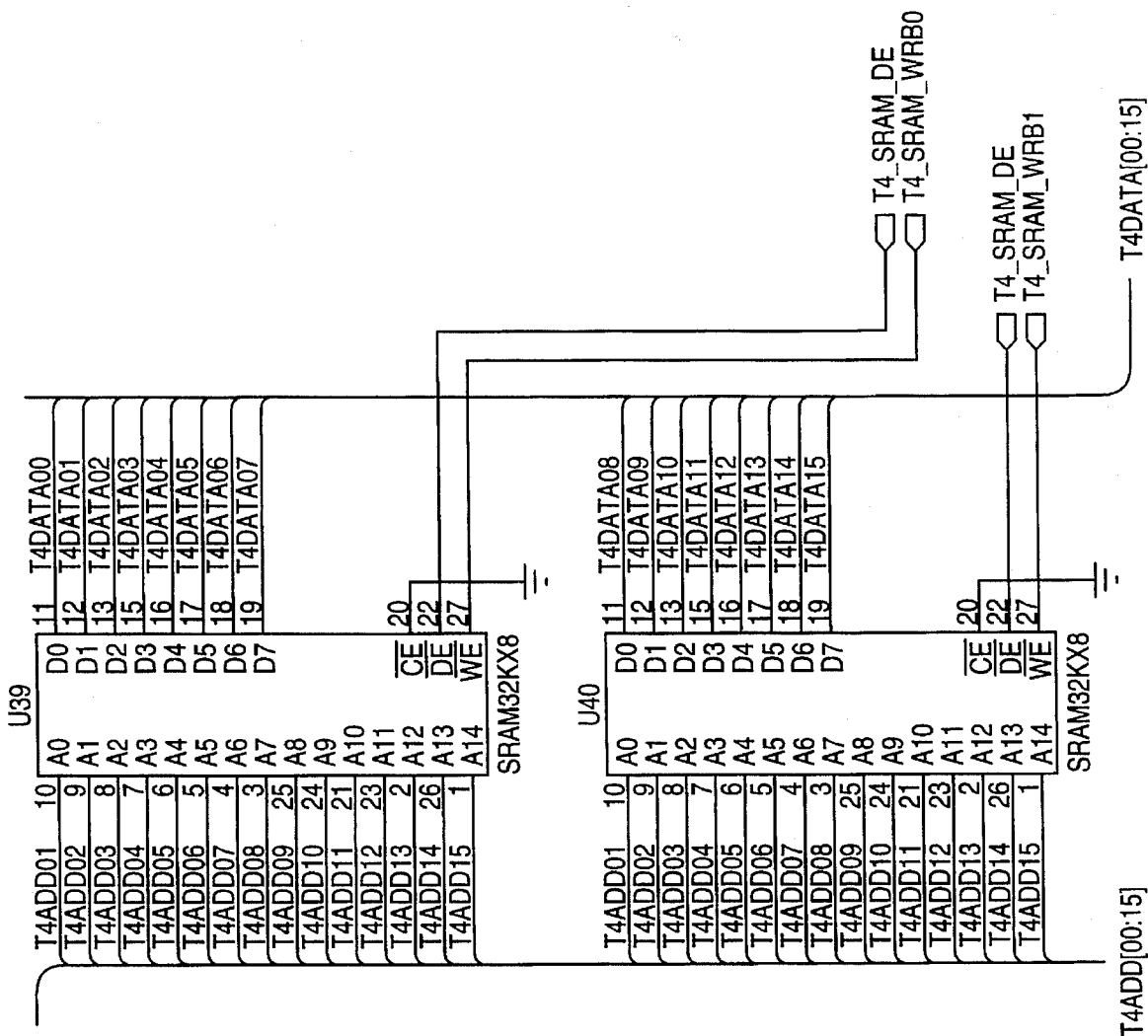


FIG. 26

U.S. Patent

Jul. 16, 1996

Sheet 28 of 40

5,537,533

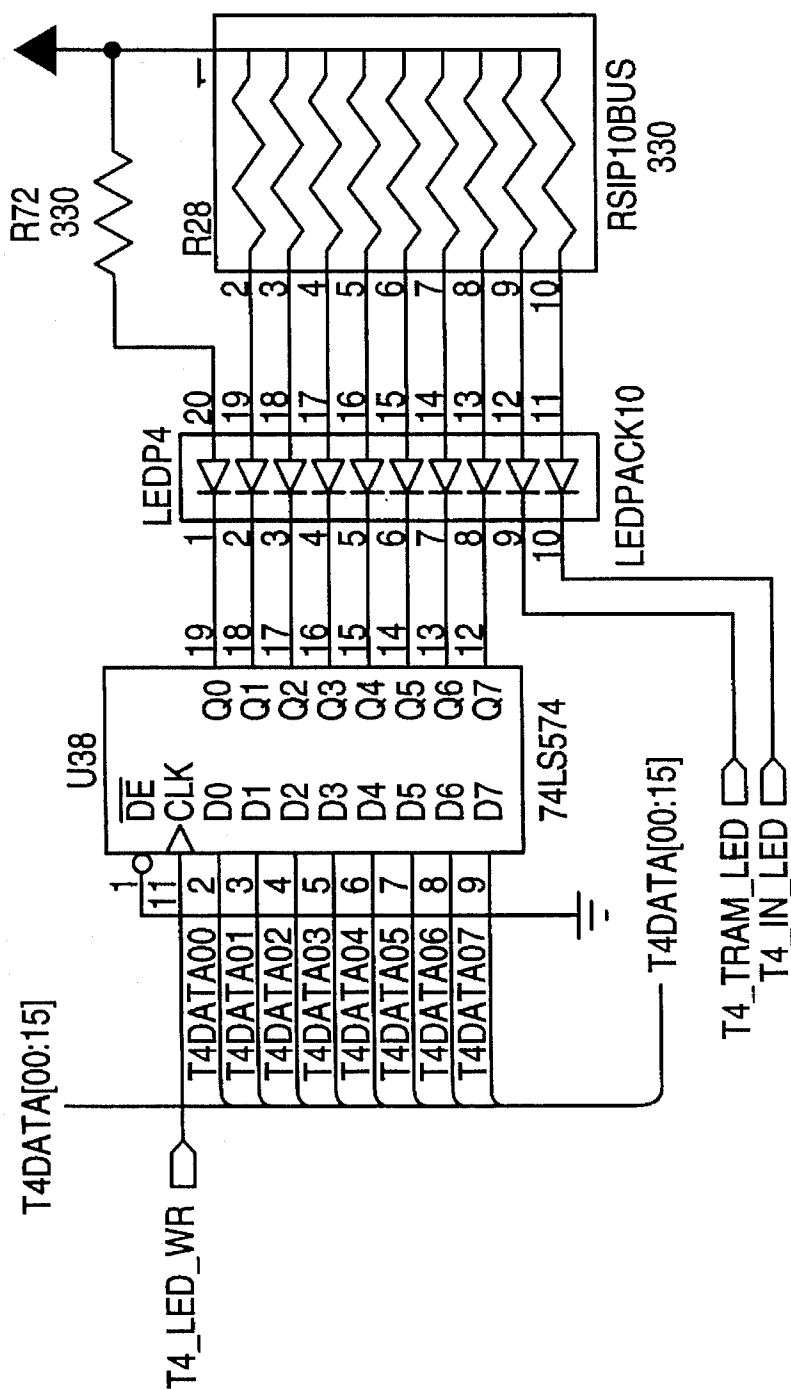


FIG. 27

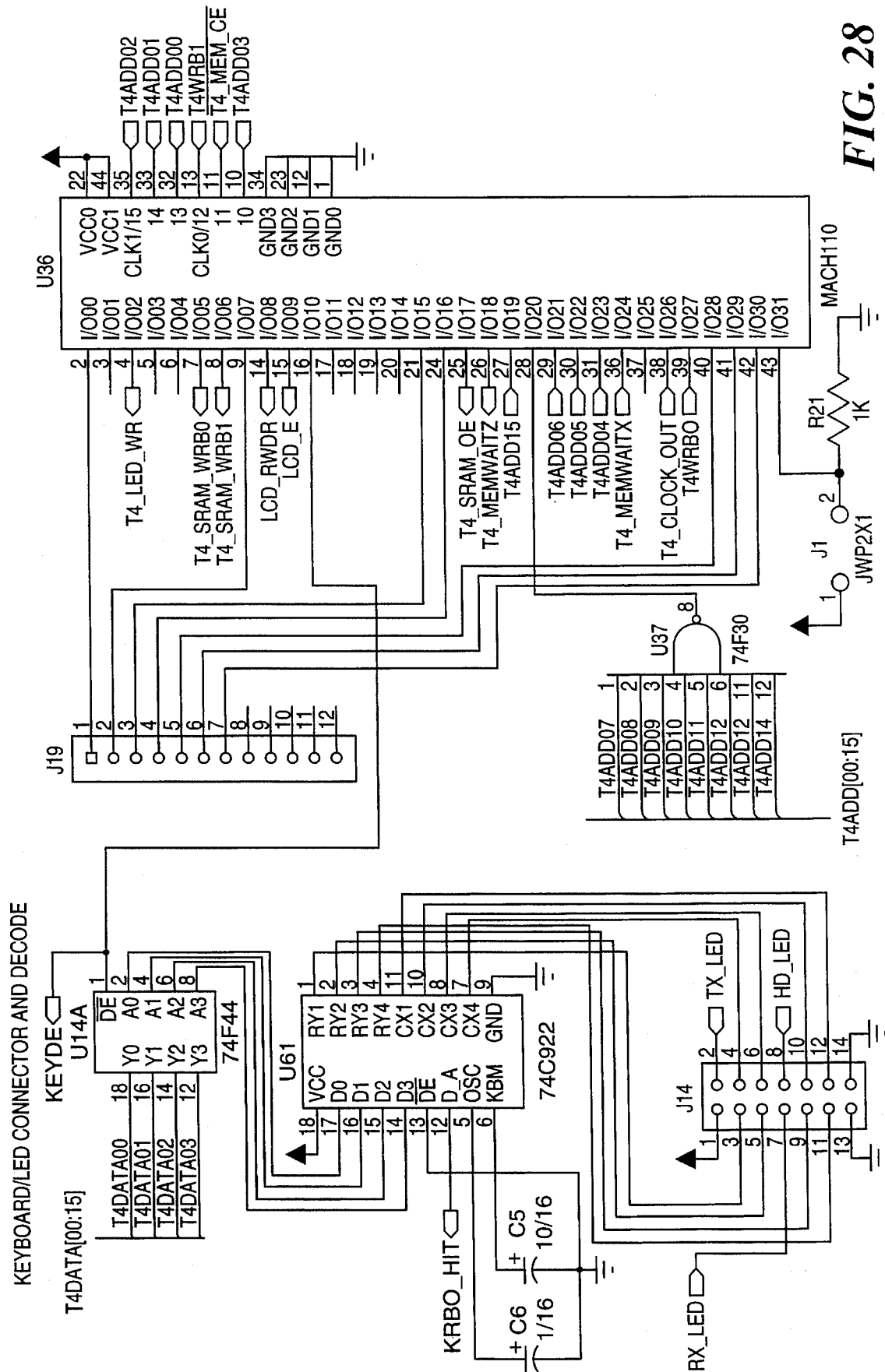


FIG. 28

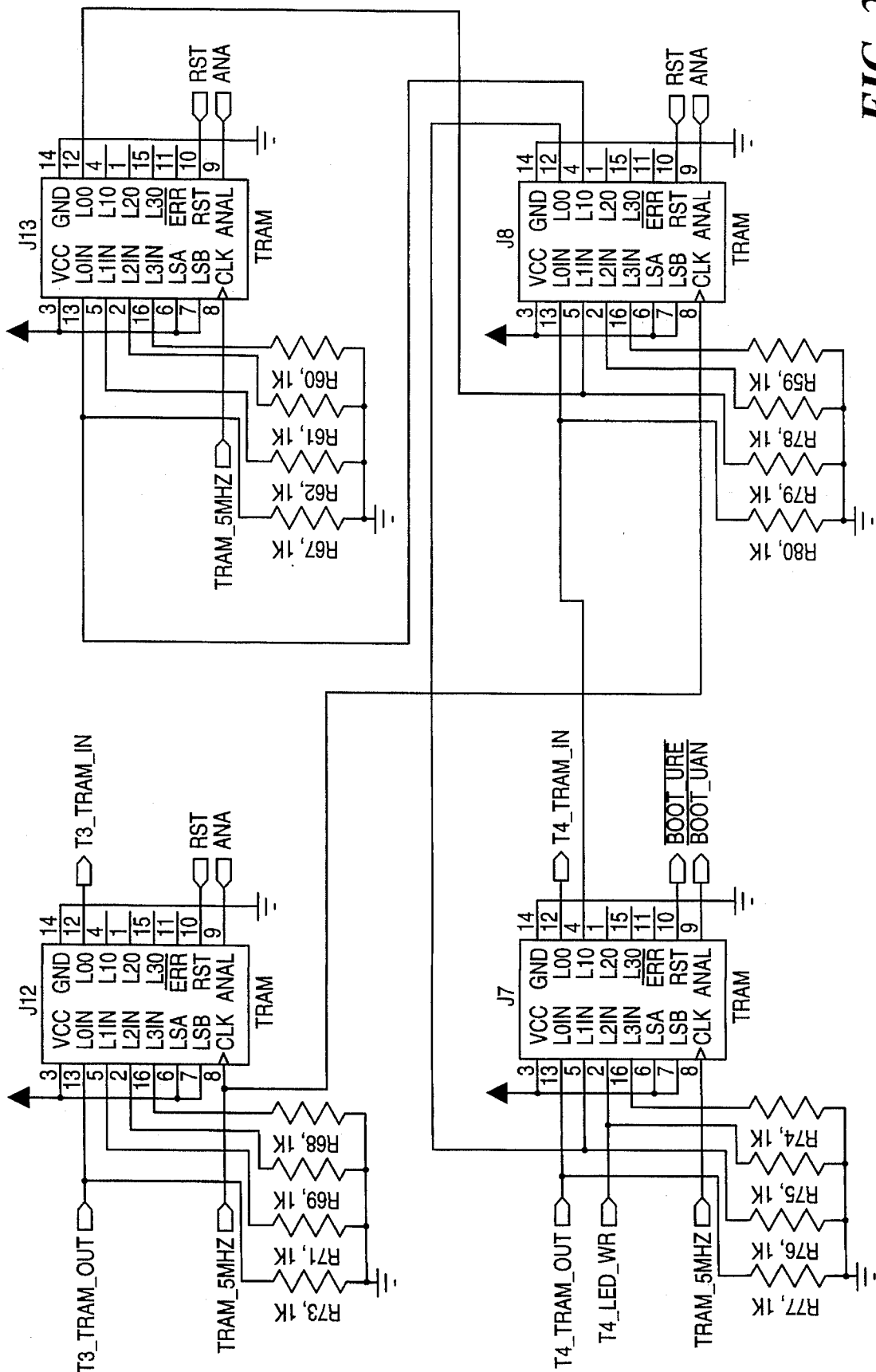


FIG. 29

CSU INTERFACE CONNECTOR

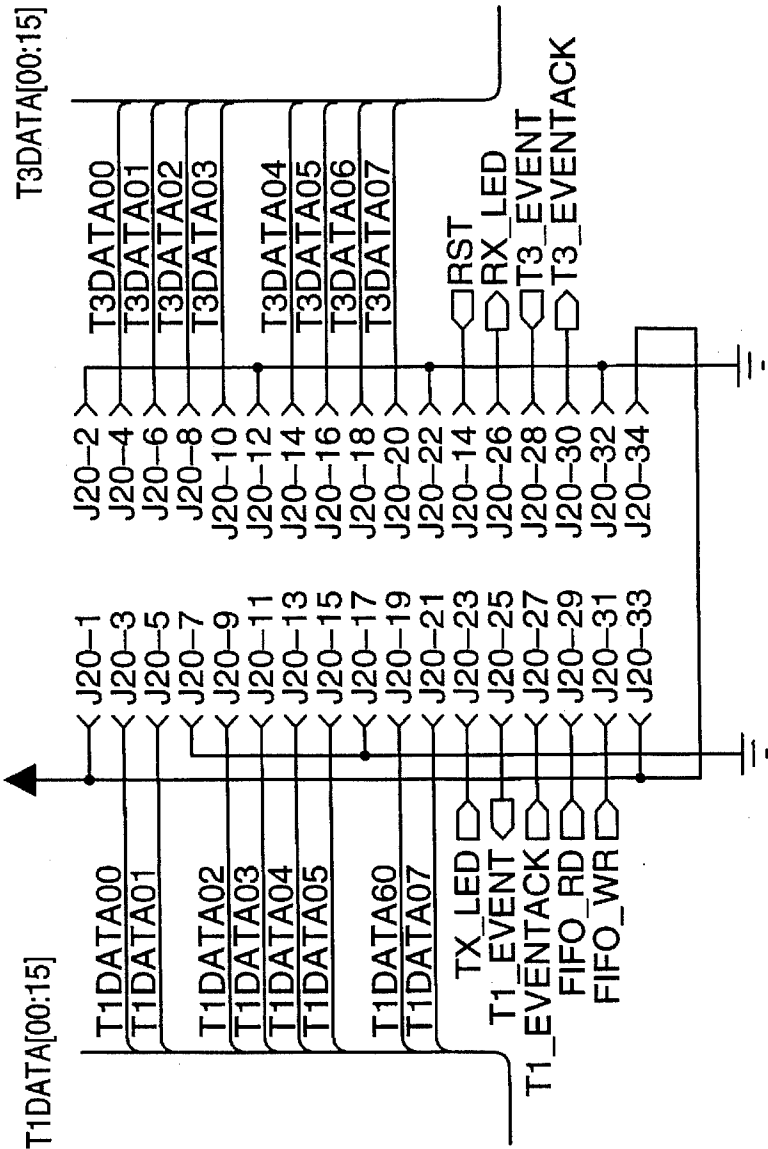


FIG. 30

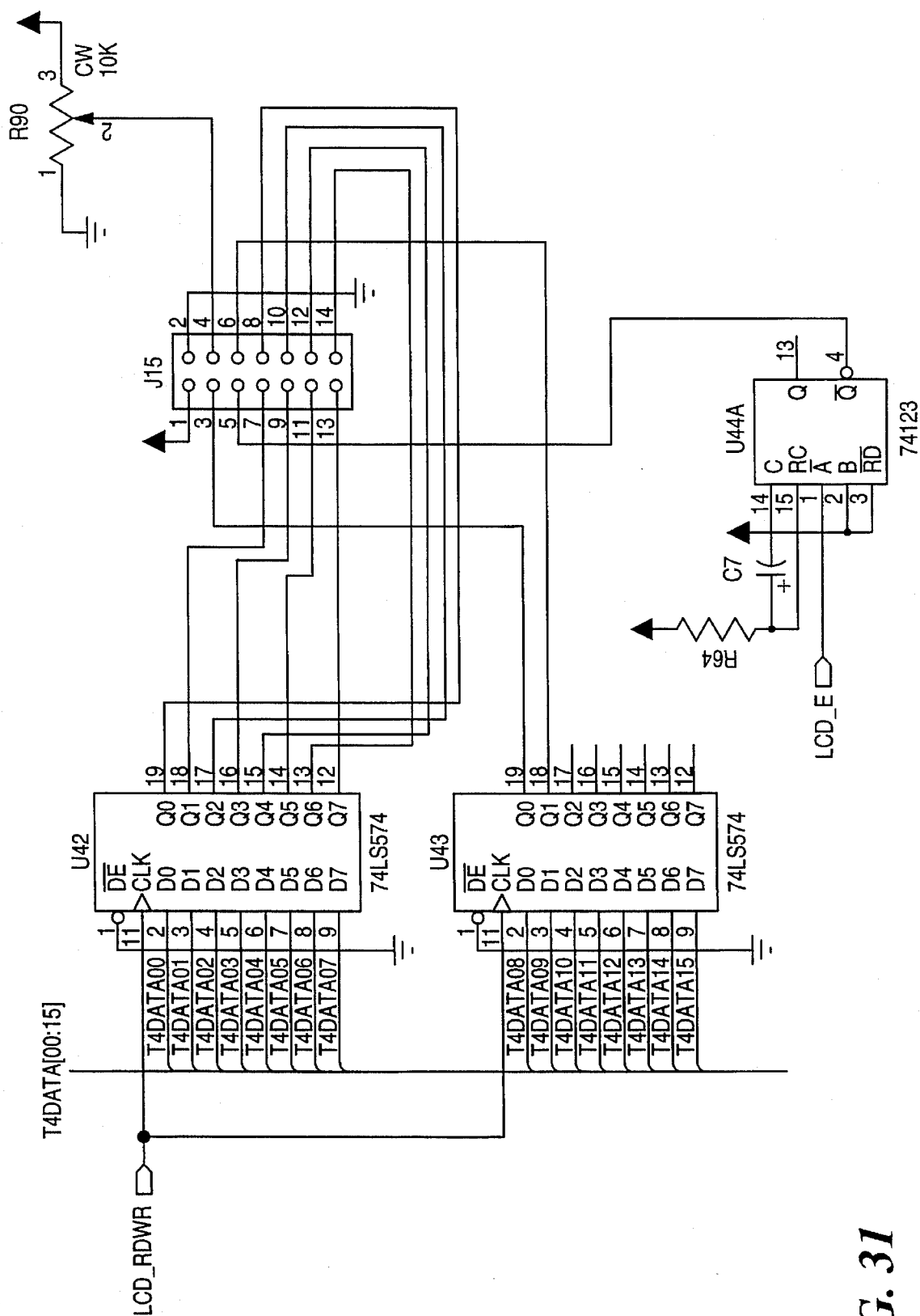


FIG. 31

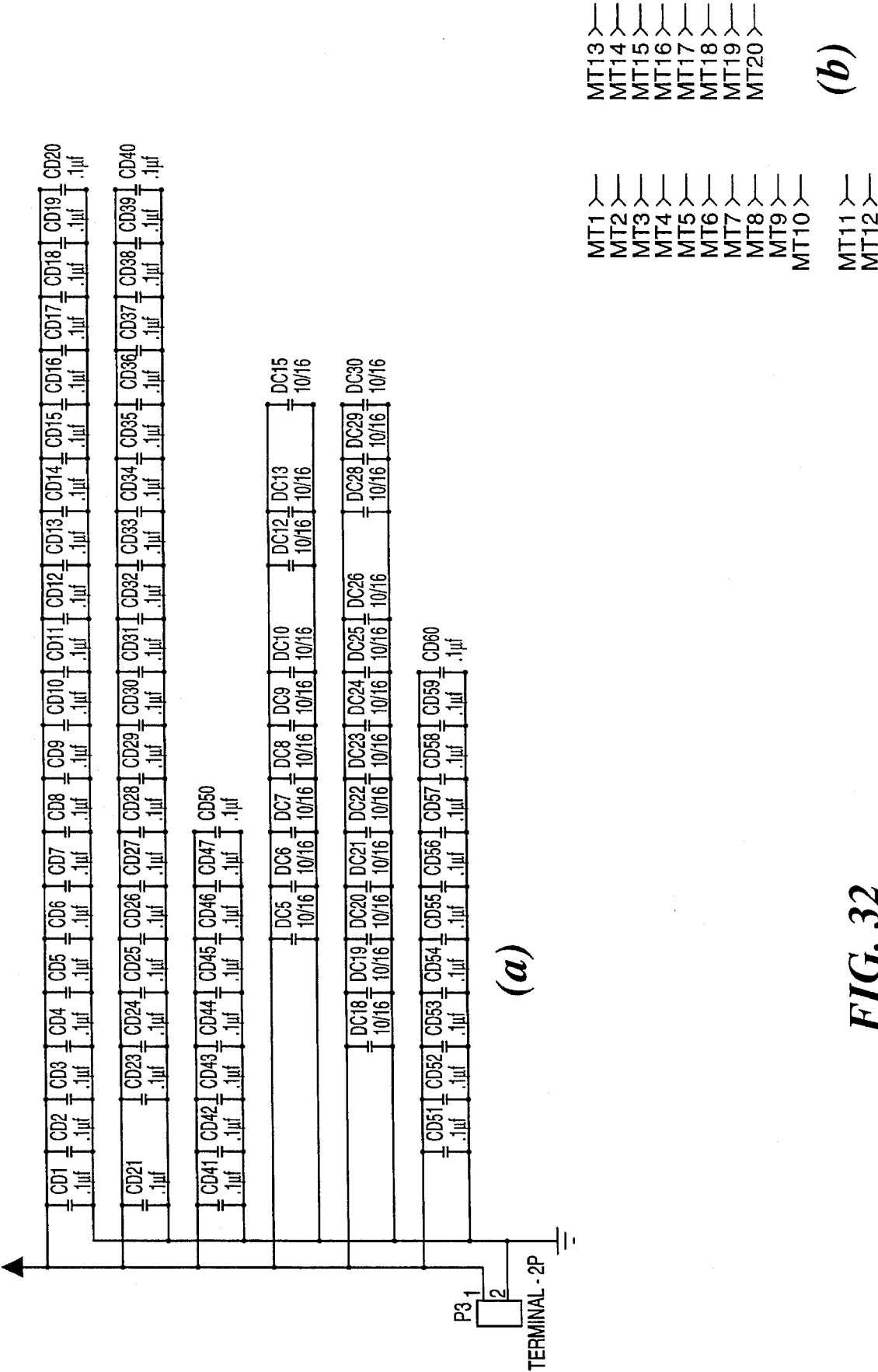


FIG. 32

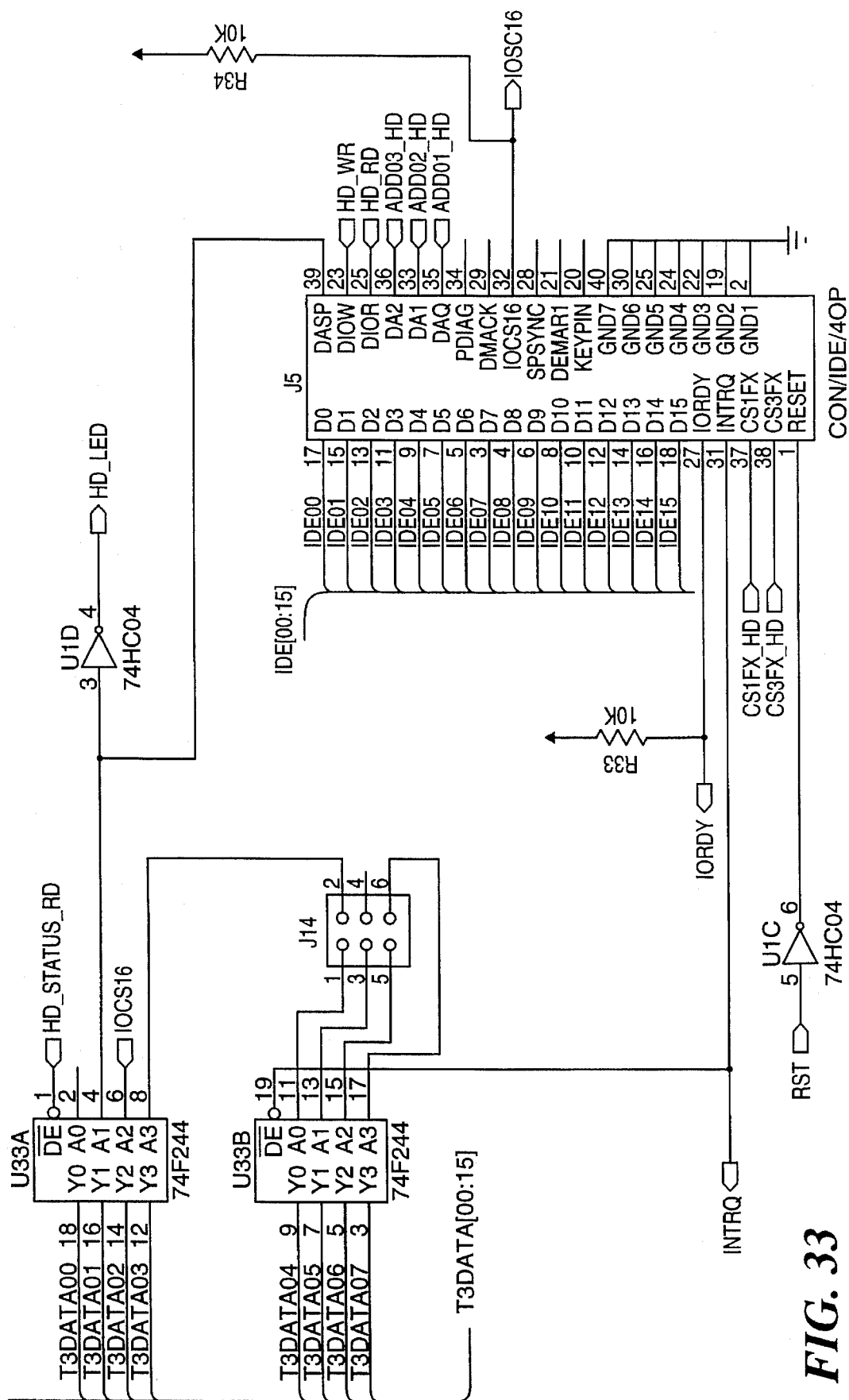


FIG. 33

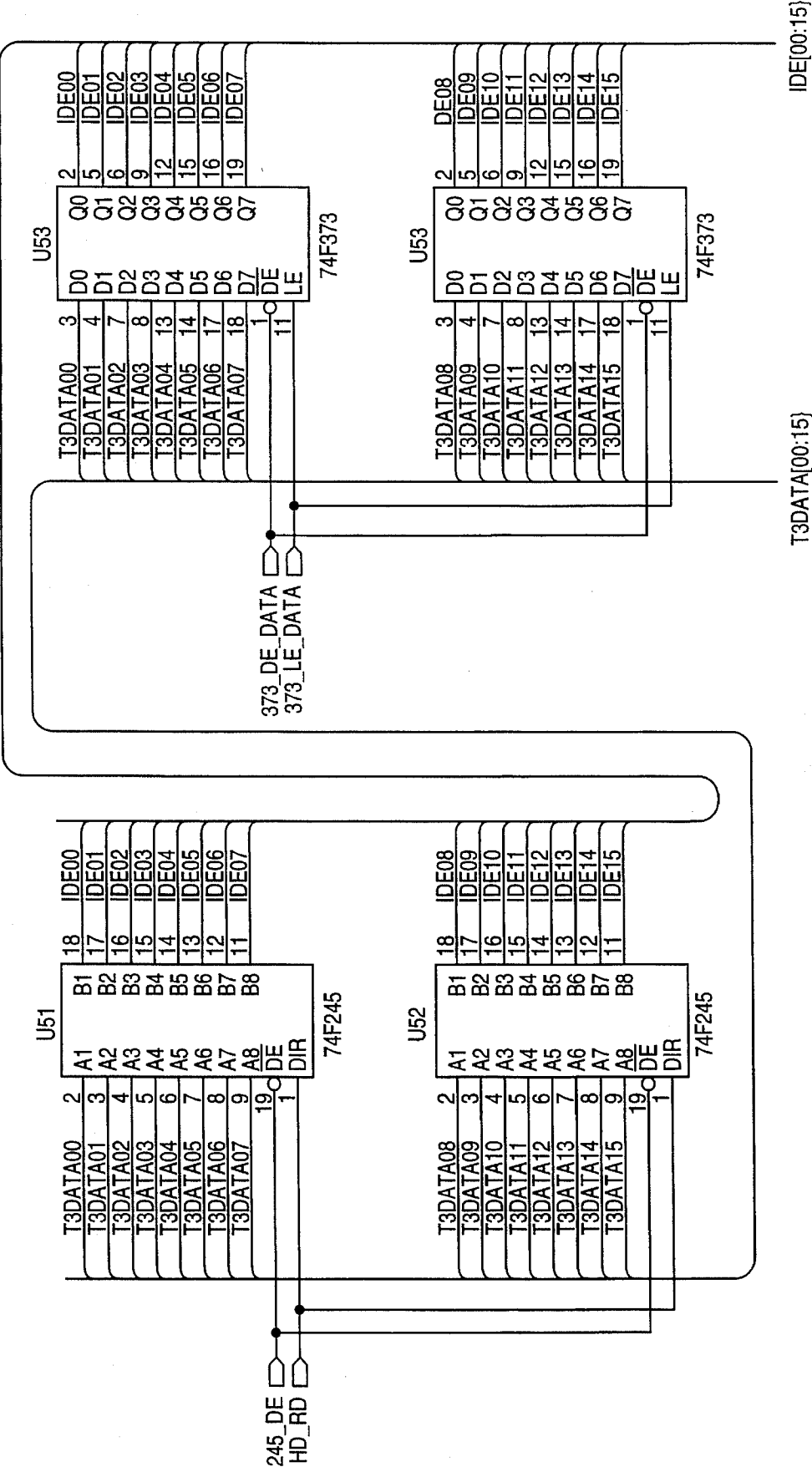


FIG. 34

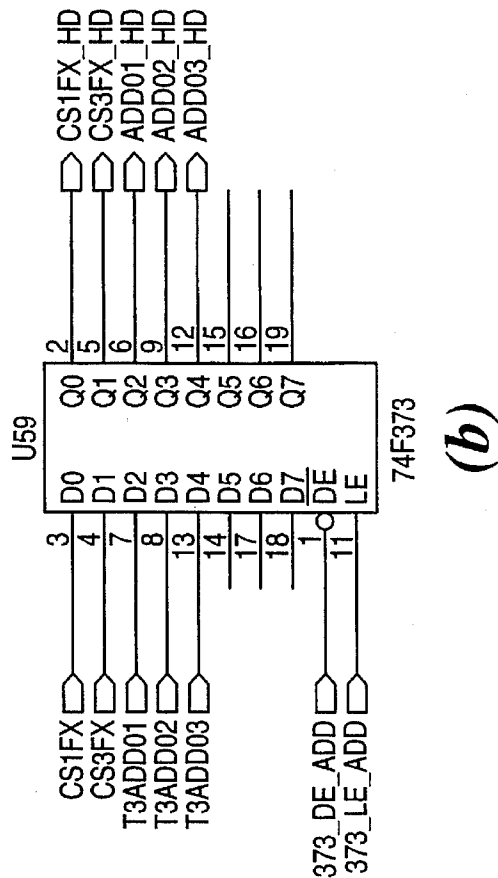
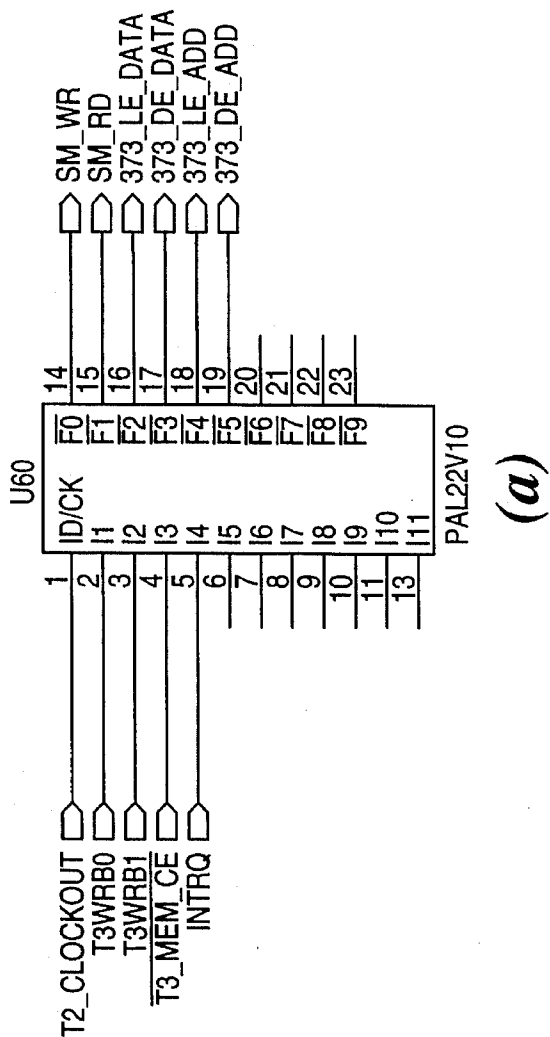


FIG. 35

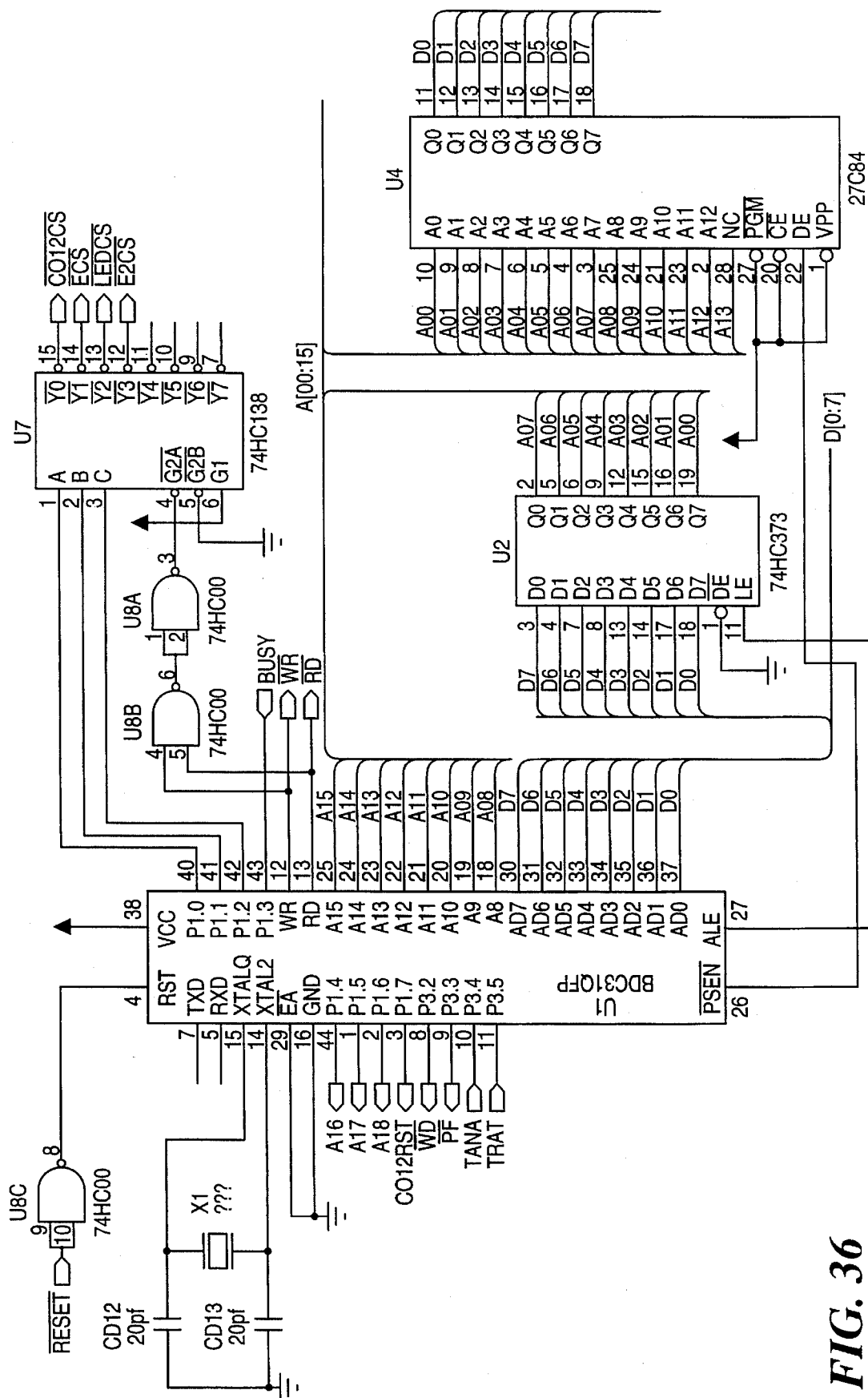


FIG. 36

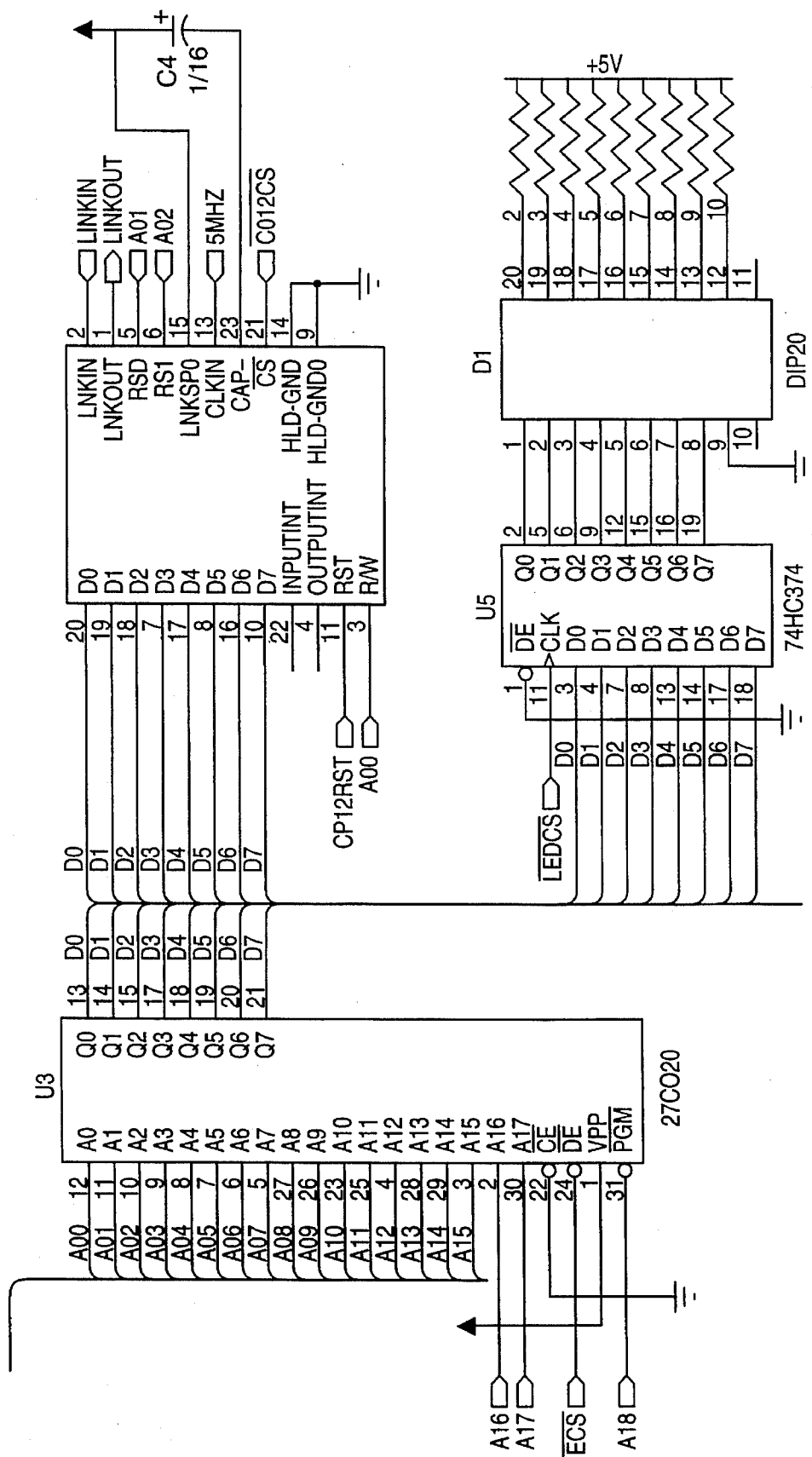


FIG. 37

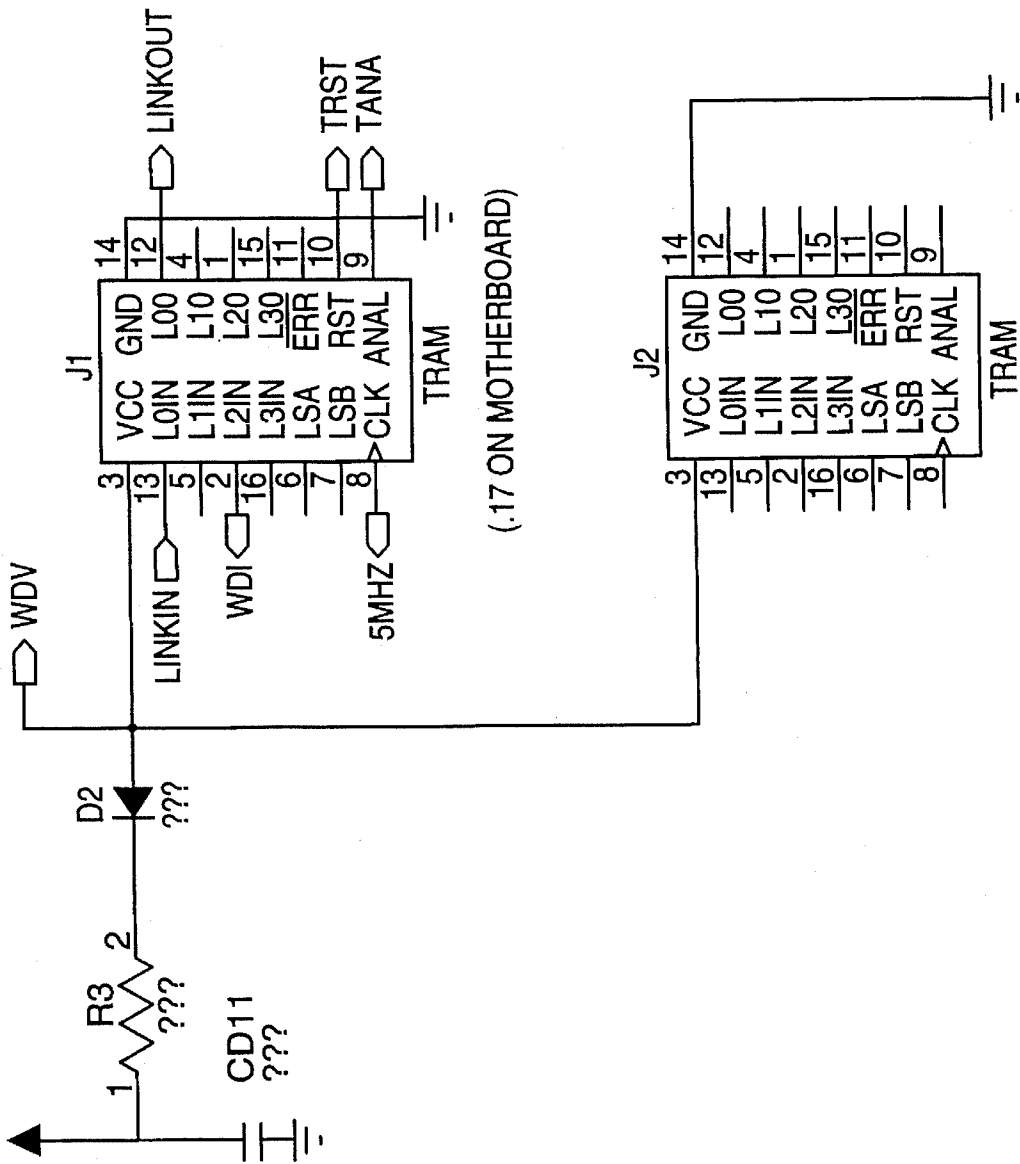


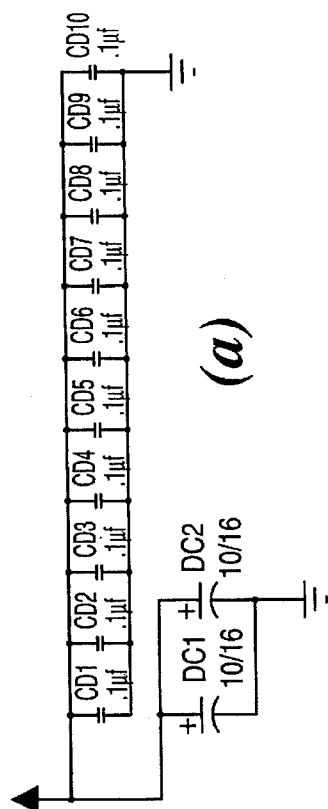
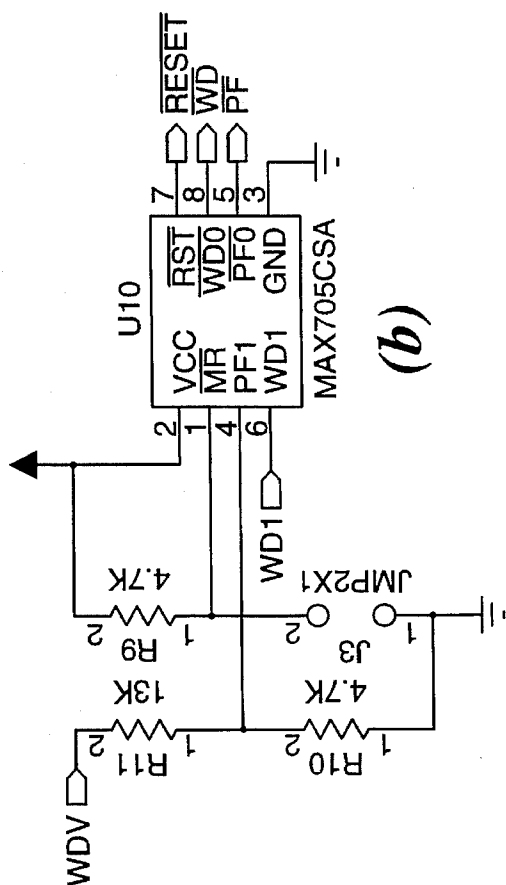
FIG. 38

U.S. Patent

Jul. 16, 1996

Sheet 40 of 40

5,537,533



5,537,533

1

SYSTEM AND METHOD FOR REMOTE MIRRORING OF DIGITAL DATA FROM A PRIMARY NETWORK SERVER TO A REMOTE NETWORK SERVER

FIELD OF THE INVENTION

The present invention relates to the protection of mission-critical data in a client-server computer network as part of a business recovery plan, and more particularly to a system and method for remotely mirroring data such that a substantially concurrent copy of the critical data stored on a primary network server is also continuously available on a replacement network server which is located at a safe distance from the primary network server.

TECHNICAL BACKGROUND OF THE INVENTION

Banks, insurance companies, brokerage firms, financial service providers, and a variety of other businesses rely on client-server computer networks to store, manipulate, and display information that is constantly subject to change. A significant amount of the information stored as digital data in computer networks is mission-critical. For instance, the success or failure of an important transaction may turn on the availability of information which is both accurate and current. In certain cases the credibility of the service provider, or its very existence, depends on the reliability of the information displayed by the network.

Accordingly, many financial firms world-wide recognize the commercial value of their data and are seeking reliable, cost-effective ways to protect the information stored on their client-server computer networks. In the United States, federal banking regulations also require that banks take steps to protect critical data.

Mission-critical digital data may be threatened by natural disasters, by acts of terrorism, or by more mundane events such as computer hardware failures. Although these threats differ in many respects, they are all limited in their geographic extent. Thus, many approaches to protecting data involve creating a copy of the data and placing that copy at a safe geographic distance from the original source of the data. As explained below, geographic separation is an important part of data protection, but does not alone suffice for many network users.

The distance which is deemed safe depends on the expected threats to the data. Storing a copy of the data in the same room with the original data typically provides some protection against hardware failures; storing data in another room in the same building or in a building across the street may provide the copy of the data with sufficient protection against destruction by a fire that destroys the storage medium holding the original data. Acts of terrorism, earthquakes, and floods require greater separation. In some cases, separations of 30 miles or more are required.

In the mainframe computer environment a process known as "remote journaling" or "electronic vaulting" is used to protect data. A mainframe at the original data site is connected by a communications link to a remote mainframe which is located at a safe distance from the original site. Data written to the original mainframe's disks is also sent at essentially the same time to the communications link, and hence to the remote mainframe, where it is stored until needed. Mainframe electronic vaulting thus suggests that, in addition to geographically separating a copy of critical data, data protection for client-server networks should also

2

include some way of updating the geographically separate copy of the data.

Although electronic vaulting provides an abstract model for the protection of client-server computer network data, the extreme differences between mainframe environments and client-server network environments prevent any substantial use of electronic vaulting hardware or software in such networks. At the hardware level, mainframe connectors, bus protocols, and signals are all typically incompatible with those of client-server computer networks. Hardware which connects a mainframe to a communications link will typically not even plug into a networked workstation or personal computer, much less function properly to permit communication.

At the software level, electronic vaulting code may be embedded within the mainframe's operating system, making it difficult or impossible to port the vaulting code to a client-server network environment. Even when the electronic vaulting software is not embedded within the mainframe's operating system, the interface between the proprietary mainframe operating system and the electronic vaulting software generally involves disk accesses, context switching, and other critical low-level operations. Such low-level software is generally very difficult to port to a network, which uses a very different operating system.

In addition, mainframes, unlike networks, do not typically face the prospect of coordinating the activities of numerous users, each of whom is controlling a separate machine having its own local operating system and central processing unit. Thus, mainframe software typically assumes "sole ownership" of files and other system resources. Such assumptions, which may permeate the electronic vaulting code, do not hold in a network.

A different approach to copying data, which is used both with mainframes and with networks, is off-site tape storage. Critical data is copied onto magnetic tapes at the end of each business day. These backup tapes are then taken by truck or plane to a storage site some distance from the original data. Thus, if a disaster of sufficiently limited geographic scope destroys data at the original site, the tapes kept at the storage site may be used to recover important information.

Although off-site tape storage is relatively simple and inexpensive, it has severe limitations. Most importantly, the data on the tapes is only as current as the most recent backup. Thus, assume a business's backup finished at 1:00 AM, the business opened at 8:00 AM, and a disaster occurred at 3:00 PM. Then the business activity in the seven hours from 8:00 AM to 3:00 PM is lost, because it was not stored on the tape. It may be difficult or impossible to accurately reconstruct every transaction that occurred during the lost period. Persuading everyone involved that the reconstruction is accurate may also present problems. In short, merely creating a geographically separate copy of data does not provide adequate protection. The remote copy must also be substantially current.

A continuing disadvantage of off-site tape storage is the time required to create the tape backup. To ensure the integrity of data being stored on the tape, only the backup software typically has access to the network during the backup procedure. If a business closes at the end of each day and leaves its computer network essentially unused at night, the opportunity costs of restricting access during the backup procedure are negligible. However, an increasing number of computer networks are used by businesses that operate world-wide, and hence these networks are needed 24 hours a day, 7 days a week. Shutting down such networks for

5,537,533

3

several hours each day to make a tape backup may have a significant adverse effect on the business.

In addition, hours or days may be needed to restore data from the backup tapes onto hard drives or other immediately useable media. The computer network's performance may be reduced while data is being restored. Indeed, in some instances it is necessary to deny all other users access to the network while data is being restored, in order to ensure the integrity of the data after the restoration.

Another approach to copying data stored on computer networks is known as "data shadowing." A data shadowing program cycles through all the files in a computer network, or through a selected set of critical files, and checks the timestamp of each file. If data has been written to the file since the last time the shadowing program checked the file's status, then a copy of the file is sent over a communications link to another program which is running on a remote computer. The remote program receives the data and stores it at the remote site on tapes or other media. As with off-site tape storage, hours or days may be required to restore shadowed data to a useable form at the original site.

Shadowed data is typically more current than data restored from an off-site tape backup, because at least some information is stored during business hours. However, the shadowed data may nonetheless be outdated and incorrect. For instance, it is not unusual to make a data shadowing program responsible for shadowing changes in any of several thousand files. Nor is it unusual for file activity to occur in bursts, with heavy activity in one or two files for a short time, followed by a burst of activity in another few files, and so on. Thus, a data shadowing program may spend much of its time checking the status of numerous inactive files while a few other files undergo rapid changes. Mission-critical data may be lost because the shadowing program is driven by the list of files and their timestamps rather than directly by file activity.

Many conventional attempts to protect data also share another problem, namely, that open files are not copied. The contents of files which have been "opened" for access by users may change during tape backup, data shadowing, or other procedures that create a copy of the file contents. These changes may lead to internal inconsistencies and lost data because a copy program (e.g., a tape backup or data shadowing program) sees one part of the file before the change and another part of the file after the change.

For instance, suppose that an open file has a length of 10,000 bytes and this length is recorded in the first block of the file. Critical data will be lost if events occur as follows: (1) the copy program notes that the file is 10,000 bytes long; (2) an additional 5,000 bytes of critical new data is added to the end of the open file by the user; and (3) at some later time, the original copy of the file—including the 5,000 new bytes—is destroyed.

The copy program will only have copied the first 10,000 bytes of the file. The additional 5,000 bytes will be lost even if the program had plenty of time to copy that data as well, because the copy program doesn't "know" that the additional data is there until it works its way back to the file in question. Depending on the copy program, the number of files involved, and other factors, minutes or even hours may pass before the program returns to the file in question and notes that additional data needs to be copied.

Accordingly, client-server computer network operating systems typically restrict access to open files, and conventional data copying methods generally do not create copies of open files even when permitted to do so by the network

4

operating system. However, the failure to copy open files also has severe drawbacks. Files may be left open longer than necessary, so that their mission-critical contents are actually stable enough to copy but are nevertheless not copied simply because the file is open. Thus, data may be lost even though it could have been copied to a safe location, merely because a file was left open longer than necessary.

In addition, failure to copy even a single open file in a relational database may lead to the loss of data in many files because such databases depend on files that are interrelated and sequential. For instance, suppose a database must search in sequential order through files A, B, C, and D to obtain the required information. Suppose file C was open during the backup and therefore was not copied. The data restored after a disaster may therefore include copies of files A, B, and D which are more current than the most recent available copy of file C. Such inconsistencies may corrupt the database and render useless the information in all four files.

Perhaps the most common solution to the open file problem is to perform backup procedures at the end of the work day. All users except the backup software are logged off the system, and all files are closed. Thus, a current, complete, and consistent copy of the critical data is obtained. However, this approach to dealing with open files has many of the drawbacks of off-site tape storage. Many networks, such as those used by hotel and airline reservation systems, credit authorization services, and global trading position databases, are in use non-stop. Moreover, critical data added after the backup software finishes is not protected until the next backup or shadowing file copy, which may be minutes or hours later.

Thus, it would be an advancement in the art to provide a system and method for effectively protecting mission-critical data in a client-server computer network.

It would also be an advancement to provide such a system and method which maintain a substantially current copy of critical network data.

It would be a further advancement to provide such a system and method which maintain a substantially current copy of data as that data is committed for storage in open files on disk.

It would also be an advancement to provide such a system and method which do not require limiting or denying access by other users while a copy of the critical data is created.

In addition, it would be an advancement in the art to provide such a system and method which permit storage of the data copy at distances up to 30 miles or more from the original data.

It would also be an advancement to provide such a system and method which make the copied data useable immediately after a disaster.

Such a system and method are disclosed and claimed herein.

BRIEF SUMMARY OF THE INVENTION

The present invention provides a system for remote mirroring of digital data from a primary network server to a remote replacement network server. As used herein, "mirroring" means creating a copy of data as the data travels toward a storage location on nonvolatile media. Mirroring may include copying data whose immediate destination is a dedicated RAM cache if the data's ultimate destination is a nonvolatile medium. Data shadowing or tape backup creation do not involve mirroring, because they typically

5,537,533

5

involve reading data from nonvolatile storage, and also involve copying the data many seconds, minutes, or even hours after the data originally reached the nonvolatile storage.

"Remote mirroring" means transmitting mirrored data to a remote location for storage. Conventional network servers have the ability to locally mirror data by copying data to a disk which is physically located within a few feet of the network server. But known systems lack the remote mirroring capability of the present invention. Moreover, unlike data shadowing or tape backup approaches, remote mirroring captures data early in the data's existence. Remote mirroring thus significantly reduces the risk that the data will be destroyed before a copy of the data is sent to a safe remote storage location.

The primary network server and the remote replacement network server which are used in conjunction with the present invention each have their own nonvolatile server store. "Nonvolatile" stores include data storage media such as magnetic or optical disk drives which preserve data in the absence of any external source of power. By contrast, random access memory ("RAM") is typically a volatile medium because it does not preserve data when power to the server is lost.

A presently preferred embodiment of the invention includes a primary data transfer unit ("primary DTU") and a remote data transfer unit ("remote DTU") which are connectable with one another by a conventional communication link. In operation, the primary DTU sends mirrored data from the primary network server over the link to the remote DTU, which is located at a safe distance from the primary DTU. The remote DTU receives the mirrored data and sends it in turn to the replacement server. The replacement server then stores the remotely mirrored data in a conventional manner.

The replacement server is not active as a network server while the primary server is functioning normally, but stands ready to be brought on-line rapidly as a replacement if the primary server fails. At most one instance of the network operating system runs at any time, even though the primary server and the remote server each have a copy of the operating system. The phrases "remote server," "remote network server," "remote replacement server," "replacement server," and "remote network replacement server" are synonymous herein.

Thus, the invention is useful in creating a copy of data at a safe distance from the original data substantially concurrently with the storage of the original data. Moreover, the copied data is useable almost immediately after a disaster, because it has been copied to a "warm" remote network server which can be up and running as the new primary server within minutes of the disaster.

The primary DTU includes a primary server interface and a primary link interface. The primary server interface is digitally connectable to the primary network server, and has sufficient bandwidth and signal compatibility to receive mirrored data from the primary network server as that data is created. The mirrored data tends to arrive in high bandwidth bursts, as it is a substantially concurrent copy of original data which is destined for storage in the nonvolatile server store of the primary network server.

The primary link interface is digitally connected to the primary server interface, and is capable of receiving the mirrored data from the primary server interface and sending it across a conventional communication link to the remote DTU. A checksum is preferably computed on the data and

6

transmitted with the data so the remote DTU can detect transmission errors. The data may also be compressed and/or encrypted by the primary DTU before it is placed on the link. Communication links to which the link interface is tailored include T1, E1, analog telephone line, and other conventional links.

The remote DTU similarly includes a remote link interface and a remote server interface. In fact, the primary DTU and the remote DTU preferably include identical hardware and software. The remote link interface is connectable to the communication link for receiving the mirrored data sent across the link by the primary DTU. The remote DTU has decompression and decryption capabilities corresponding to the compression and encryption capabilities of the primary DTU.

The remote server interface is digitally connected to the remote link and is capable of receiving the mirrored data from the remote link. The remote server interface is digitally connectable to the remote network server and has sufficient bandwidth and signal compatibility to send the mirrored data to the remote network server. Each DTU is equipped with a link interface diagnostic unit which provides selected status and diagnostic information in a human-readable format.

The DTUs and other system components are configured such that the mirrored data is normally sent to the remote network server by the remote server interface within a measurable delay time, such as 1 or 10 seconds, from the time the mirrored data is received by the primary server interface. The system is preferably configured such that the mirrored data is also stored in the nonvolatile server store of the remote network server within a short time, e.g., 10 seconds, from the time the corresponding original data is stored in the nonvolatile server store of the primary network server.

The primary network server typically provides mirrored data to the primary DTU in bursts whose bandwidth is greater than the bandwidth of the link between the DTUs. In addition, it may be necessary to use the remote server for tasks other than receiving and storing mirrored data. Thus, one or both of the DTUs preferably includes a data buffer, and most preferably includes a data buffer which is non-volatile to reduce the risk of data loss. The presently preferred data buffer includes magnetic hard disks disposed within each DTU.

Each of the DTUs includes at least one microprocessor and a block of RAM which is accessible by the microprocessor. It is presently preferred that each DTU include four digitally interconnected parallel processors. Each parallel processor includes a microprocessor, a block of RAM accessible by the processor, and up to four interprocessor serial communication lines which support communication between parallel processors. In operation, the parallel processors work substantially concurrently to process different selected portions of the mirrored data. For instance, two parallel processors read data from the primary server interface substantially in parallel.

The present invention also provides a method for remote mirroring of digital data. The method begins by using the local disk mirroring utility of a conventional client-server network operating system to copy the data from a primary network server to a primary DTU which is digitally connected to the primary network server. Then the data is copied from the primary DTU to an input end of a communication link, possibly after being compressed and/or encrypted. The output end of the communication link is physically separated from its input end by a distance of at least 100 feet. The data

5,537,533

7

is copied from the output end of the communication link to a remote DTU, any required decompression or decryption is performed by the remote DTU, and the data is sent by the remote DTU to a nonvolatile server store on a remote network server.

The data typically moves between a server and a DTU in chronologically spaced-apart, rapid, high bandwidth bursts. The communication link between the DTUs generally has lower bandwidth, but also permits transmissions that last longer than such bursts. Thus, the data is preferably copied to a nonvolatile data buffer in the primary DTU in order to allow the use of a conventional communication link having lower bandwidth than the bandwidth of the channel between the server and the server interface.

The copying steps which move the data between the servers and the DTUs are preferably implemented by substantially concurrent copying of different selected portions of the data. That is, several computer-implemented parallel copying processes are used to accomplish substantially concurrent copying. The parallel processes communicate with one another to coordinate the copying.

Thus, the present invention provides a system and method for effectively protecting mission-critical data in a computer network. Unlike data shadowing or tape backup approaches, the present invention stores mirrored data and hence maintains a substantially current copy of the data, including data in open files, without limiting access by other users. The mirrored data is moved by the DTUs to safe distances up to 30 miles or more from the original data. The copied data is also useable immediately after a disaster, because it has been copied to the nonvolatile store of a "warm" remote network server which can be rapidly placed in operation as the new primary network server.

These and other features and advantages of the present invention will become more fully apparent through the following description and appended claims taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

In order that the manner in which the above-recited and other advantages and features of the invention are obtained, a more particular description of the invention summarized above will be rendered by reference to the appended drawings. Understanding that these drawings only provide a selected embodiment of the invention and are not therefore to be considered limiting of its scope, the invention will be described and explained with additional specificity and detail through the use of the accompanying drawings in which:

FIG. 1 is a schematic diagram illustrating two data transfer units of the present invention connecting a primary network server with a remote network server.

FIG. 2 is a hardware block diagram illustrating the presently preferred embodiment of the link interface portion of the data transfer unit of the present invention, including four parallel processors labeled T1 through T4, a channel service unit, a boot module, keypad and display I/O devices, and a nonvolatile buffer.

FIG. 3 is a flowchart illustrating the events involved in transmitting a packet of mirrored data from the server interface portion of a primary data transfer unit to a communication link which connects the primary data transfer unit with a remote data transfer unit.

FIG. 4 is a flowchart illustrating the events involved in receiving a packet of mirrored data sent from the primary

8

data transfer unit over the communication link to the remote data transfer unit.

FIG. 5 is a flowchart illustrating the events involved in transmitting a packet of mirrored data from the link interface of the remote data transfer unit to a remote server interface.

FIG. 6 is a schematic block diagram illustrating software processes and hardware resources involved in sending data from the link interface.

FIG. 7 is a schematic block diagram illustrating software processes and hardware resources involved in receiving data at the link interface.

FIGS. 8 through 11 are hardware schematic diagrams illustrating a presently preferred embodiment of a serial interface to parallel processors of the link interface.

FIGS. 12 through 14 are hardware schematic diagrams illustrating a presently preferred embodiment of the parallel processor T4 and a data receiving portion of the channel service unit.

FIGS. 15 through 19 are hardware schematic diagrams illustrating a presently preferred embodiment of the parallel processor T3 and a control portion of the channel service unit.

FIGS. 20 through 24 are hardware schematic diagrams illustrating a presently preferred embodiment of the parallel processor T2 and transmit and IDE portions of the channel service unit.

FIGS. 25 through 29 are hardware schematic diagrams illustrating a presently preferred embodiment of the parallel processor T1, I/O circuits, and boot module.

FIGS. 30 and 31 are hardware schematic diagrams illustrating a presently preferred embodiment of a UART/T1 interface for connecting the link interface to a conventional T1 communication link.

FIG. 32 is a hardware schematic diagram illustrating a presently preferred embodiment of decoupling and filter capacitors for use in the data transfer unit.

FIGS. 33 through 35 are hardware schematic diagrams illustrating a presently preferred embodiment of an IDE interface portion of the link interface.

FIGS. 36 through 39 are hardware schematic diagrams further illustrating a presently preferred embodiment of the boot module.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Reference is now made to the figures wherein like parts are referred to by like numerals. With reference to FIG. 1, the present invention relates to a system and method for remote mirroring of digital data from a primary network server 10 to a remote network server 12. The servers 10, 12 are conventional servers for a local area network ("LAN") and/or a wide area network ("WAN").

It is presently preferred that the servers 10, 12 be IBM PC-compatible servers capable of running the Novell NetWare® operating system. The primary server 10 actually runs the operating system, while the remote server 12 preferably runs a simple DOS data-capturing program that transfers data from the remote DTU 40 to the remote disk 20. Although only one copy of the client-server network operating system is in use at any time, the remote server is preferably a "warm" server in that it is equipped with a copy of the network operating system and may be brought on-line as the new primary network server if the original primary network server 10 is disabled.

5,537,533

9

Those of skill in the art will appreciate that other hardware platforms such as 680x0-, PowerPC®, or RISC-based architectures, possibly in combination with other operating systems such as UNIX® or NT® operating systems, may also be advantageously employed in conjunction with the present invention. POWERPC is a trademark of International Business Machines Corp.; UNIX is a registered trademark of Unix System Laboratories, Inc.; NT is a registered trademark of Microsoft Corp. Also, the servers **10**, **12** may be connected by a conventional LAN or WAN **14** in addition to the remote mirroring connection provided by the present invention.

As noted above, "mirroring" means creating a copy of data as the data travels toward a storage location on non-volatile media. In the configuration shown in FIG. 1, the primary server **10** includes two magnetic hard disks **16**, **18**, and the remote server **12** includes one magnetic hard disk **20**, as nonvolatile server stores. The hard disk **20** preferably has a storage capacity of at least about 500 Mbytes, and preferably follows a write-read-verify-acknowledge protocol to detect and subsequently avoid bad hard disk sectors. However, those of skill in the art will appreciate that other types of nonvolatile media, such as optical disks or RAM powered by an uninterruptible power supply, may also be used in either or both of the servers **10**, **12**.

A presently preferred system embodying the invention includes a primary data transfer unit ("primary DTU") **30** and a remote data transfer unit ("remote DTU") **40**. The primary DTU **30** includes a primary server interface **32** and a primary link interface **34**. The remote DTU **40** similarly includes a remote server interface **42** and a remote link interface **44**.

Except as otherwise noted herein, the two link interfaces **34**, **44** preferably comprise the same hardware and software. The presently preferred link interface software can identify attributes of data packets and, based on these attributes and the direction of packet movement, the software can determine whether it resides on the primary DTU **30** or on the remote DTU **40**. Appropriate portions of the software may thus be invoked on each DTU **30**, **40** without requiring the user to manually set a hardware switch or take similar action. The two server interfaces **32**, **42** likewise preferably comprise identical hardware controlled by appropriate portions of identical software. As used herein, "software" includes firmware, object code, and other program instructions capable of controlling microprocessors or other computer hardware.

It is presently preferred that the network operating system on the primary network server **10** mirror the data and then transmit the mirrored data to the primary DTU **30**. Conventional operating systems such as Novell NetWare® provide suitable local mirroring or duplexing capability. Conventional hardware and software provide mirroring and local transmission of the mirrored data for distances of about fifty feet.

According to the teachings herein, the network operating system is made to see the remote disk **20** as simply another drive available to the primary server **10**, and so is able to use its mirroring utility and the present invention to remotely mirror data to that disk **20** even when the disk **20** is miles away from the primary server **10**. Moreover, the mirrored data includes a substantially concurrent copy of original critical data, including data which is being sent by the operating system to the local disk **16** for storage in a presently open file of the primary network server **10**.

Advantageously, the present invention does not require that the primary server **10** and the remote server **12** utilize

10

identical hardware or that their hard disks **16**, **20** operate in lockstep with each other. The present invention also does not require transmission of substantial amounts of information other than the user's critical data. Unlike conventional remote duplication of an entire file server, such as with Novell SFT III, the remote mirroring of file data according to the present invention does not require transmission of CPU registers, cache, operating system process contexts, and the like between the primary and remote servers **10**, **12**.

Thus, the bandwidth of the link **50** between the DTUs **30**, **40** which is required for effective performance of the present invention is substantially lower than the bandwidth that would be required to remotely mirror an entire file server. A link **50** having a bandwidth of about 1.5 Mbits/sec is adequate for many applications of the present invention, whereas full server mirroring could require a communication link **50** bandwidth of at least about 100 Mbits/sec. The present invention will operate over existing communication links **50**, such as existing telephone and data lines, without requiring the user to lay a new dedicated fiber optic cable or other high-bandwidth signal carrier.

The server interface **32** is preferably configured such that it emulates a conventional I/O device controller in communicating with the server **10**. In particular, it is presently preferred that the server interface **32** emulate a hard disk drive controller. Thus, the hardware and operating system of the server **10** communicate with the server interface **32** as they would communicate with the controllers of conventional hard disk drives **16**, **18**.

Hard disk controller emulation by the server interface **32** allows utilization of the built-in data duplication capability of the operating system of the primary server **10**. The operating system is instructed in a conventional manner to duplicate critical data as that data is sent toward the main disk **16**. The duplication may utilize one or more I/O channels. In the configuration illustrated in FIG. 1, the duplicated data is copied to the local disk **18** to guard against data loss from failure of the main disk **16**. More importantly, a duplicate of the data is also routed to the server interface **32** for remote mirroring by way of the present invention.

The server interface **32** passes the mirrored data it receives from the server **10** to the link interface **34**. The server interface **32** shown in FIG. 1 communicates with the link interface **34** by way of a high-speed serial link **36**, but those of skill in the art will appreciate that parallel links may be employed in other embodiments. One combination of hardware and software which is suitable for use as either the primary server interface **32** or the remote server interface **42** is the StandbyServer product available from Vinca Corporation of Orem, Utah.

FIG. 2 illustrates a presently preferred embodiment of the link interface **34**. The link interface **34** includes four parallel processors **60**, which are individually labelled T1 through T4 in FIG. 2. Parallel processors are microprocessors designed specifically for interprocess communication and hence are well-suited for use in parallel processing applications. Each parallel processor **60** includes an arithmetic and logical processing unit and local RAM accessible by that processing unit. Each parallel processor **60** also includes four serial communication ports **62**, which are individually labelled L0 through L3.

It is presently preferred that each of the parallel processors **60** include a Transputer® microprocessor; Transputer is a registered trademark of Inmos Ltd. Presently preferred embodiments of the parallel processors **60** and serial ports **62** are further illustrated in the hardware schematic diagrams

5,537,533

11

of FIGS. 8 through 29, but those of skill in the art will appreciate that other embodiments may also be utilized according to the present invention.

With reference once more to FIG. 2, the link interface 34 also includes a boot module 64 ("boot TRAM"). As further illustrated in FIGS. 36 through 39, the boot TRAM 64 preferably contains an EPROM which holds the boot program that loads the operating software which controls the link interface 34 to perform the operations described herein.

As shown in FIG. 2, the preferred embodiment of the link interface 34 also contains a nonvolatile buffer 66. The presently preferred nonvolatile buffer 66 is a conventional magnetic hard disk drive 68. However, optical drives, RAM powered by an uninterruptible power supply, or other non-volatile storage media may also be utilized according to the present invention.

A channel service unit ("CSU") 70 acts as the direct interface between the rest of the link interface 34 and the conventional communication link 50. A presently preferred embodiment of the CSU 70 contains circuits for receiving data from the parallel processors 60 (FIGS. 12 through 14), control circuits (FIGS. 15 through 19), and circuits for transmitting data to the communication link 50 (FIGS. 20-24, 30-31, and 33-35).

With reference to FIGS. 2 and 25-31, the link interface 34 preferably also includes a keypad 72, a display such as a liquid crystal display ("LCD") 74, and supporting circuitry. Thus, the link interface 34 is capable of providing performance and diagnostic information in human-readable format. Additional information and control are preferably available via an auxiliary computer 76 which may be placed in serial communication with the link interface 34.

With reference to FIG. 2, the parallel processors T1 through T4 are interconnected by serial communication lines 78. Additional serial communication lines 80 connect one or more of the parallel processors 60 with the server interface 32, the boot TRAM 64, and the optional auxiliary computer 76. A data bus 82, which is preferably 16 bits wide, connects the parallel processor T2 to the nonvolatile buffer 66. Other data buses 84, which are preferably 8 bits wide, connect the CSU 70 to the parallel processors T2 and T4. Control lines 86 connect the CSU 70 with parallel processors T2, T3, and T4.

Presently preferred embodiments of the communication lines, data buses, and control buses, and their associated buffers and processors, are illustrated in FIGS. 8 through 39, but those of skill in the art will appreciate that other embodiments may also be utilized according to the present invention. Moreover, although particular assignments of the serial ports 62 are illustrated, other port assignments may also be utilized according to the present invention.

As a very simple example, the assignments of port L0 and port L1 of parallel processor T1 could be swapped in the hardware and software of the link interface 34, such that port L0 would connect to the boot TRAM 64 and port L1 would connect to the auxiliary computer 76. Moreover, the labeling of the parallel processors may also vary. Indeed, the processors labelled T1 through T4 in FIGS. 2 through 7 are respectively denoted T4 through T1 in the hardware schematics of FIGS. 8 through 39.

The link interface 34 is further illustrated by the flowcharts shown in FIGS. 3 through 5 and the process block diagrams in FIGS. 6 and 7. Those of skill in the art will appreciate that certain portions of the data flow steps shown in FIGS. 3 through 5 are preferably accomplished substantially in parallel. For instance, steps accomplished by data

12

reading and data writing processes are performed substantially concurrently by at least two parallel processors.

With reference to FIGS. 1 and 2, during remote mirroring several tasks are generally performed by the data transfer units 30, 40. Mirrored data is moved from the server 10 to the primary DTU 30. The data is moved from the primary DTU 30 to the communication link 50. The data is moved from the communication link 50 to the remote DTU 40, and from the remote DTU 40 to the remote disk 20. In addition, an acknowledgement may be generated and sent to the primary server 10 indicating that the data has been received.

With reference now to FIGS. 1, 2, 3, and 6, the task of moving data from the server 10 to the primary DTU 30 is presently accomplished by moving each packet of data sent from the server 10 as follows. A process in T4 reads half of the waiting packet's header from the server 10, while a substantially parallel (i.e., concurrent) process in T3 reads the other half of the header. T4 then transmits at least a portion of its half of the packet header to T3.

Next, T3 parses the two halves of the packet header and determines the length of the data portion of the waiting packet. This length is transmitted back to T4. T4 and T3 then concurrently read the data, with half of the data being read by a process on each parallel processor. As shown in FIG. 6, T3 and T4 are each equipped with a data transmit buffer to hold the data. T4 and T3 concurrently transmit their respective halves of the data to T2, where the data resides in the T2 data transmit buffer. The data transmit buffers are preferably implemented with static RAM (FIGS. 13, 16, and 21).

With reference now to FIGS. 1, 2, 3, and 6, the task of moving the data from T2 to the communication link 50 is presently accomplished as follows. If the packet is an acknowledgement generated outside the DTUs 30, 40, the packet may be thrown away if the DTUs are configured to generate their own acknowledgements as described herein. Otherwise, T2 generates a packet header which uniquely identifies the data, and writes ("dispatches") the header and data to the nonvolatile buffer 66. T2 also sends a copy of the data to the CSU 70, which prepares the data for transmission and places it on the communication link 50. The preparation preferably includes computing a checksum on the data so that CRC error checking or another conventional algorithm can be used to detect transmission errors and cause retransmission of the data to the remote DTU 40. The data may be compressed and/or encrypted using conventional algorithms and hardware before it is placed on the communication link 50.

The communication link 50 may be a telecommunications T1 (also known as "DS1"), T2 ("DS2"), T3 ("DS3"), E1, E2, or E3 link, a fractional T1 link, an Asynchronous Transfer Mode ("ATM") link, a Frame Relay link, a so-called "10 megabit per second bridged service" link having a bandwidth of approximately 10 Mbits/second, an analog telephone line connected to modems, an Ethernet, an ISDN link, a 56-switched link, or another conventional communication link. The link 50 may include physical connections in the form of cables, twisted pair wires, analog telephone lines, microwave transmitters and receivers, optical fibers, or other conventional signal carriers. The output end of the communication link 50 is preferably physically separated from its input end by a distance of at least 100 feet, and may be separated by thousands of feet or even by many miles.

The link interface 34 is preferably implemented at least in part with a communication-link-specific daughter board which snaps into a connector on an interface motherboard. The CSU 70 (FIG. 2) is preferably located on such a

5,537,533

13

daughter board, with the parallel processors 60 being located on the motherboard. Thus, the hardware changes needed to tailor the link interface 34 to different communication links are easily effected in the field.

With reference now to FIGS. 1, 2, 4, and 7, the task of moving data from the communication link 50 to the remote disk 20 is accomplished by the remote DTU 40 as follows. T4 reads a data packet from the CSU 70. Recalling that the DTUs 30 and 40 preferably comprise identical hardware, note that the CSU 70 now referred to is part of the remote DTU 40, not part of the primary DTU 30.

T4 then examines the data packet. If the packet is an acknowledge generated by the DTUs, T4 sends the acknowledge to T2, and T2 clears the appropriate entry in a packet allocation table, thereby freeing space on the nonvolatile buffer 66. If the packet is a data packet, T4 sends half of the packet to T3. Then T4 and T3 concurrently write their respective halves of the data packet to the remote server 12.

With reference to FIGS. 1, 2, 5 and 7, the present invention optionally includes the task of sending a "spoof packet" as an acknowledgement to the primary server 10 indicating that the data has been received. The spoof packet is in the format of an acknowledgement that would be sent by a conventional hard disk drive controller when the operating system writes data to the hard disk. As described below, the user may specify which of several events should trigger generation of a spoof packet, and may enable or disable spoofing to balance server 10 performance against the risk of lost data.

In a presently preferred embodiment, the invention generates a spoof packet and sends the spoof packet to the operating system of the primary network server 10 in response to any of several user-selectable trigger events. For instance, users may choose to generate and send a spoof packet after the primary DTU 30 receives the mirrored data, after the data is stored in the nonvolatile buffer 66 of the primary DTU 30, after the data has been transferred to the communication link 50 by the primary DTU 30, after the data has been received from the communication link 50 by the remote DTU 40, or after the data is transferred to the remote server 12. The spoof packet may also be generated and sent a given time after one of these trigger events. Alternatively, the invention may be configured to generate no spoof packets but to instead simply pass along to the primary server 10 the acknowledgements generated by the nonvolatile store 20 of the remote server 12.

The spoof packet is preferably generated by the DTU involved in the triggering event. For instance, if the trigger is storage of the mirrored data in the nonvolatile buffer 66 of the primary DTU 30, T2 prepares a spoof packet acknowledging receipt of the data. Before sending a spoof packet, T2 first determines that the packet being transmitted is a data packet that needs to be spoofed. T2 then sends the spoof packet to T4. T4 divides the spoof packet in half and sends half to T3. Then T3 and T4 concurrently write their respective halves of the spoof packet to the primary server 10. Note that the spoof packet may be received by the primary server 10 before the data reaches the remote hard disk 20. The acknowledgement packet generated by the hard disk 20 or by its controller will be thrown away and thus will never reach the primary server 10, which receives the spoof packet instead.

In addition to specifying a triggering event, it is presently preferred that the user be permitted to specify a cache buffer threshold which determines whether or not spoofing is enabled. Data destined for eventual storage in a nonvolatile

14

store is typically written first to a volatile cache buffer (not shown) in the primary network server 10. Cache buffers which hold data that has not yet been written to a nonvolatile store, i.e., data which is held pending a write, are conventionally termed "dirty cache buffers."

The total storage capacity of the primary server's cache buffers is limited. If the cache buffers fill up, performance of the primary server 10 may decrease perceptibly because the network operating system waits for an acknowledgement after each disk write before proceeding. Without spoofing, such acknowledgements may be long in coming because the "oldest" data, i.e. data written first to the cache buffers, has priority over other data for disk writes. Thus, if burst traffic occurs from the primary server 10 to the cache buffers associated with the server interface 32, and the magnitude of that burst traffic exceeds the bandwidth of the link 50, then the percentage of cache buffers which are dirty may increase to a point where overall server 10 disk access performance becomes very poor.

Enabling spoofing generally helps clean out the dirty cache buffers because the network operating system on the primary server 10 generally receives spoofing packets sooner than acknowledgements that are generated by the remote nonvolatile store 20. However, spoofing is preferably disabled when the percentage of cache buffers that are dirty falls below a user-specified threshold, e.g., 75 percent. Depending on the trigger event which causes generation of the spoof packet, the data being acknowledged has various degrees of vulnerability to loss in the event of a disaster. Some users may elect to disable spoofing permanently to avoid possible data loss. Dirty cache information is readily obtained by software, such as a NetWare Loadable Module or the equivalent, and relayed from the primary server 10 to the link interface 32 for use in enabling or disabling spoofing.

With reference to FIG. 1, the typical general operation of the present invention thus proceeds as follows. The network operating system on the primary server 10 copies critical data which is destined for the local hard disk 16. The operating system sends the mirrored data to a device it sees as simply another hard drive controller, but which is actually the server interface 32.

The server interface 32 transmits the mirrored data to the link interface 34, which stores the data on a local disk drive 68 (FIG. 2) and sends the primary server's operating system a spoof packet acknowledging receipt of the data. The link interface 34 then sends the mirrored data across the conventional communication link 50 to the remote DTU 40, which is located at a safe distance from the primary DTU 30. The remote DTU 40 sends the mirrored data in turn to the remote network server 12, where it is stored on a remote hard disk 20.

Although a configuration involving only two network servers 10, 12 is illustrated, those of skill in the art will appreciate that additional servers may also be employed according to the present invention. For instance, a primary and secondary server may be located in the same room while a tertiary server is located miles away. The primary server would then locally mirror data to the secondary server by conventional means, and would remotely mirror critical data to the tertiary server according to the present invention.

Thus, the present invention provides a system and method for effectively protecting mission-critical data in a computer network. Unlike data shadowing or tape backup approaches, the present invention stores mirrored data and hence maintains a substantially current copy of the data, including data

5,537,533

15

which is being written to open files, without limiting access by other users. The mirrored data is moved by the DTUs to safe distances that may be many miles from the original data. The copied data is also useable immediately after a disaster, because it has been copied to the hard disk of a "warm" remote network server which can be rapidly installed as the new primary network server.

The invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. Any explanations provided herein of the scientific principles employed in the present invention are illustrative only. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

What is claimed and desired to be secured by patent is:

1. A system for remote mirroring of digital data from a primary network server to a remote network server, each network server having its own nonvolatile server store, said system comprising a primary data transfer unit and a remote data transfer unit, wherein:

said primary data transfer unit comprises:

a primary server interface which is digitally connectable to the primary network server, said primary server interface having sufficient bandwidth and signal compatibility to receive mirrored digital data from the primary network server, the mirrored data being a substantially concurrent copy of original data which is destined for storage in the nonvolatile server store of the primary network server; and

a primary link interface digitally connected to said primary server interface and capable of receiving the mirrored data from said primary server interface, said primary link interface connectable to a communication link and capable of sending the mirrored data across the link to said remote data transfer unit, said primary link interface further comprising;

a nonvolatile data buffer for temporarily storing the mirrored data; and

spoof packet generator capable of generating a pre-acknowledgement for transmission to the primary network server by said primary link interface after the mirrored data has been stored on said nonvolatile data buffer and before an acknowledgement arrives indicating that the mirrored data has been stored by the remote network server;

and said remote data transfer unit comprises:

a remote link interface connectable to the communication link for receiving the mirrored data sent across the link by said primary data transfer unit; and

a remote server interface which is digitally connected to said remote link and capable of receiving the mirrored data from said remote link, said remote server interface being digitally connectable to the remote network server and having sufficient bandwidth and signal compatibility to send the mirrored data to the remote network server.

2. The system of claim 1, wherein said nonvolatile data buffer comprises a magnetic hard disk.

3. The system of claim 1, wherein said primary server interface receives from the primary network server a substantially concurrent copy of original data which is being written to an open file in the nonvolatile server store of the primary network server.

4. The system of claim 1, wherein said primary server interface receives from the primary network server a sub-

16

stantially concurrent copy of substantially all original data which is being stored in the nonvolatile server store of the primary network server.

5. The system of claim 1, wherein each of said data transfer units comprises a microprocessor and a block of volatile random access memory which is accessible by the microprocessor.

6. The system of claim 1, wherein each of said data transfer units comprises a plurality of digitally interconnected parallel processors, and each of said parallel processors comprises:

a processor;

a block of volatile random access memory which is accessible by the processor; and

at least one interprocessor communication line which supports communication between at least two of said parallel processors.

7. The system of claim 6, wherein said interprocessor communication line is a serial communication line.

8. The system of claim 6, wherein each of said data transfer units comprises a plurality of said parallel processors which are capable of substantially concurrently processing different selected portions of the mirrored data.

9. The system of claim 1, wherein said primary data transfer unit and said remote data transfer unit comprise substantially identical hardware.

10. The system of claim 1, wherein said primary data transfer unit and said remote data transfer unit comprise substantially identical hardware and substantially identical software.

11. The system of claim 1, wherein said primary link interface is physically separated from said remote link interface by a distance of at least 100 feet.

12. The system of claim 1, wherein said primary link interface is physically separated from said remote link interface by a distance of at least 10,000 feet.

13. The system of claim 1, wherein said primary link interface is physically separated from said remote link interface by a distance of at least 30 miles.

14. The system of claim 1, wherein said system is configured such that the mirrored data is sent to the remote network server by the remote server interface within a delay time from the time the mirrored data is received by the primary server interface.

15. The system of claim 14, wherein said data transfer units are configured such that said delay time is less than one second.

16. The system of claim 14, wherein said data transfer units are configured such that said delay time is less than ten seconds.

17. The system of claim 1, wherein each of said link interfaces is connectable to a T1 communication link and said link interfaces are capable of transferring the mirrored data across the T1 communication link.

18. The system of claim 1, wherein each of said link interfaces is connectable to an E1 communication link and said link interfaces are capable of transferring the mirrored data across the E1 communication link.

19. The system of claim 1, wherein each of said link interfaces is connectable to an ISDN communication link and said link interfaces are capable of transferring the mirrored data across the ISDN communication link.

20. The system of claim 1, wherein each of said link interfaces is connectable to an analog telephone line communication link and said link interfaces are capable of transferring the mirrored data across the analog telephone line communication link.

5,537,533

17

21. The system of claim 1, wherein said primary link interface further comprises means for generating a checksum corresponding to given data, and said remote link interface further comprises corresponding means for using said checksum to detect transmission errors and for requesting retransmission of the given data by said primary link interface.

22. The system of claim 1, wherein said primary link interface further comprises a data compression means for compressing data prior to sending the data across the communication link, and said remote link interface further comprises a corresponding data decompression means for decompressing data that was compressed by said data compression means.

23. The system of claim 1, wherein said primary link interface further comprises a data encryption means for encrypting data prior to sending the data across the communication link, and said remote link interface further comprises a corresponding data decryption means for decrypting data that was encrypted by said data encryption means.

24. The system of claim 1, wherein each of said data transfer units further comprises a diagnostic unit which is digitally connected to said link interface and which is capable of providing selected status and diagnostic information about said link interface in a human-readable format.

25. A method for remote mirroring of digital data, said method comprising the steps of:

copying the data from a primary network server to a nonvolatile data buffer in a data transfer unit which is digitally connected to the primary network server, the primary network server including an operating system which is capable of accessing a nonvolatile server store, the data copied to the data transfer unit being a substantially concurrent copy of data which is being stored by the operating system in the nonvolatile server store of the primary network server;

copying the data from the data transfer unit to an input end of a communication link which has an output end physically separated from its input end;

generating and sending a spoof packet to the operating system of the primary network server; and

copying the data from the output end of the communication link to a nonvolatile server store on a remote network server.

26. The method of claim 25, wherein said step of generating and sending a spoof packet precedes said step of copying the data from the output end of the communication link.

27. The method of claim 23, wherein said step of generating and sending a spoof packet occurs in response to a triggering event selected by a user from the group consisting of:

receipt of the data by the data transfer unit which is digitally connected to the primary network server;

storage of the data in a nonvolatile buffer in the data transfer unit which is digitally connected to the primary network server;

copying of the data from the data transfer unit to the input end of the communication link;

copying of the data from the output end of the communication link to a second data transfer unit which is digitally connected to the remote network server; and

copying of the data to the remote network server.

18

28. The method of claim 25, wherein said step of copying the data from a primary network server to a data transfer unit comprises substantially concurrent copying of different selected portions of the data, said substantially concurrent copying being performed by a plurality of computer-implemented parallel copying processes which communicate with one another to coordinate said substantially concurrent copying.

29. The method of claim 25, wherein data which is being stored by the operating system in the nonvolatile server store of the primary network server travels along an internal communication path having a predetermined internal bandwidth, and said step of copying the data from the data transfer unit to the input end of the communication link comprises:

buffering at least some of the data within the data transfer unit; and

copying the data to a communication link which has a lower bandwidth than the internal bandwidth of the primary network server.

30. The method of claim 25, wherein said step of copying the data from a primary network server to a data transfer unit comprises the steps of:

copying a data packet header from the primary network server to the data transfer unit, the header including an indication of the length of the packet; and

copying additional data from the primary network server to the data transfer unit based on the indication of the length of the packet.

31. The method of claim 30, wherein said step of copying a data packet header comprises substantially concurrent copying of different selected portions of the data packet header, said substantially concurrent copying being performed by a plurality of computer-implemented parallel copying processes which communicate with one another to coordinate said substantially concurrent copying.

32. The method of claim 31, wherein said step of copying additional data comprises substantially concurrent copying of different selected portions of additional data, said substantially concurrent copying being performed by a plurality of computer-implemented parallel copying processes which communicate with one another to coordinate said substantially concurrent copying.

33. The method of claim 25, further comprising the steps of:

compressing the data prior to said step of copying the data from the data transfer unit to the input end of the communication link; and

decompressing the data after said step of copying the data from the output end of the communication link.

34. The method of claim 25, further comprising the steps of:

encrypting the data prior to said step of copying the data from the data transfer unit to the input end of the communication link; and

decrypting the data after said step of copying the data from the output end of the communication link.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

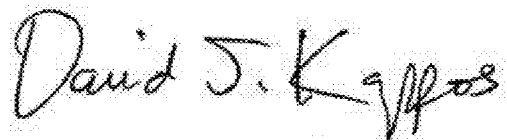
PATENT NO. : 5,537,533
APPLICATION NO. : 08/289902
DATED : July 16, 1996
INVENTOR(S) : Vaughn Staheli et al.

Page 1 of 1

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

On the Title page, item [21], delete “289,902” and insert --08/289,102--.

Signed and Sealed this
Twenty-fifth Day of October, 2011

A handwritten signature in black ink that reads "David J. Kappos". The signature is written in a cursive, flowing style with a large initial 'D' and 'K'.

David J. Kappos
Director of the United States Patent and Trademark Office

EXHIBIT B

(12) **United States Patent**
Kedem et al.

(10) **Patent No.:** **US 6,598,131 B2**
(45) **Date of Patent:** **Jul. 22, 2003**

(54) **DATA IMAGE MANAGEMENT VIA EMULATION OF NON-VOLATILE STORAGE DEVICE**

(75) Inventors: **Zvi M. Kedem**, New York, NY (US);
Davi Geiger, New York, NY (US);
Salvatore Paxia, New York, NY (US);
Arash Baratloo, New York, NY (US);
Peter Wyckoff, New York, NY (US)

5,390,331 A 2/1995 Yui
5,452,454 A 9/1995 Basu
5,471,674 A 11/1995 Stewart et al.
5,727,170 A 3/1998 Mitchell et al.
5,802,297 A 9/1998 Engquist
5,815,706 A 9/1998 Stewart et al.
5,819,065 A * 10/1998 Chilton et al. 395/500

(List continued on next page.)

(73) Assignee: **Ondotek, Inc.**, New York, NY (US)

(74) *Attorney, Agent, or Firm*—**Rothwell, Figg, Ernest & Manbeck**

(*) Notice: 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.: **10/268,898**

(22) Filed: **Oct. 11, 2002**

(65) **Prior Publication Data**

US 2003/0037202 A1 Feb. 20, 2003

Related U.S. Application Data

(63) Continuation of application No. 09/690,058, filed on Oct. 16, 2000, now Pat. No. 6,477,624.

(60) Provisional application No. 60/163,954, filed on Nov. 8, 1999, provisional application No. 60/211,291, filed on Jun. 13, 2000, and provisional application No. 60/240,138, filed on Oct. 13, 2000.

(51) **Int. Cl.**⁷ **G06F 13/368**; G06F 13/16

(52) **U.S. Cl.** **711/147**; 711/130; 709/216; 714/29

(58) **Field of Search** 709/216, 219; 711/1, 6, 118, 124, 130, 147, 162, 165, 202; 714/29

(56) **References Cited**

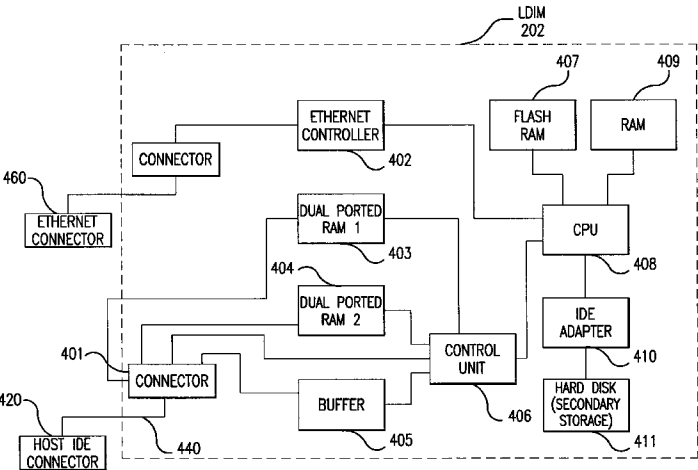
U.S. PATENT DOCUMENTS

5,146,568 A 9/1992 Flaherty et al.
5,230,052 A 7/1993 Dayan et al.
5,280,627 A * 1/1994 Flaherty et al. 713/2

(57) **ABSTRACT**

A data image management system (DIMS) that includes a local data image manager (LDIM), a remote data image manager (RDIM), and a remote persistent storage device (RPSD). The LDIM communicates with the RDIM through a direct communication link or through a communication network. The RDIM can store data on and retrieve data from the RPSD. In an environment where an LDIM has been installed in a computer having a “local” persistent storage device (LPSD), the DIMS allows for the storing of the LPSD’s data image on the RPSD, with the LPSD serving as a persistent, consistent cache of the data image. The data image stored on the RPSD is referred to as the “master data image” and the data image cached on the LPSD is referred to as the “local data image” or “cached data image.” The LDIM functions to intercept read/write requests that are intended to be received by the LPSD. The read/write requests specify an address of the LPSD. Upon intercepting a read request, the LDIM is programmed to determine whether the portion of the cached data image that is stored at the specified address is up-to-date. If it is up-to-date, the LDIM retrieves the requested data from the LPSD and passes the data back to the component or device from which it received the request. If it is not up-to-date, the LDIM transmits the read request to the RDIM. Upon receiving the read request, the RDIM locates and reads the requested data from the master data image stored on the RPSD and then transmits the data back to the LDIM.

44 Claims, 9 Drawing Sheets



US 6,598,131 B2

Page 2

U.S. PATENT DOCUMENTS			
5,828,887	A	10/1998	Yeager et al.
5,896,322	A *	4/1999	Ishii 365/189.04
5,918,229	A	6/1999	Davis et al.
5,963,941	A	10/1999	Hirakawa
5,963,971	A *	10/1999	Fosler et al. 711/114
5,987,506	A *	11/1999	Carter et al. 709/213
5,991,542	A *	11/1999	Han et al. 395/712
5,991,875	A	11/1999	Paul
5,996,028	A	11/1999	Niimi et al.
6,185,580	B1 *	2/2001	Day, III et al. 707/205
6,434,695	B1 *	8/2002	Esfahani et al. 713/2
* cited by examiner			

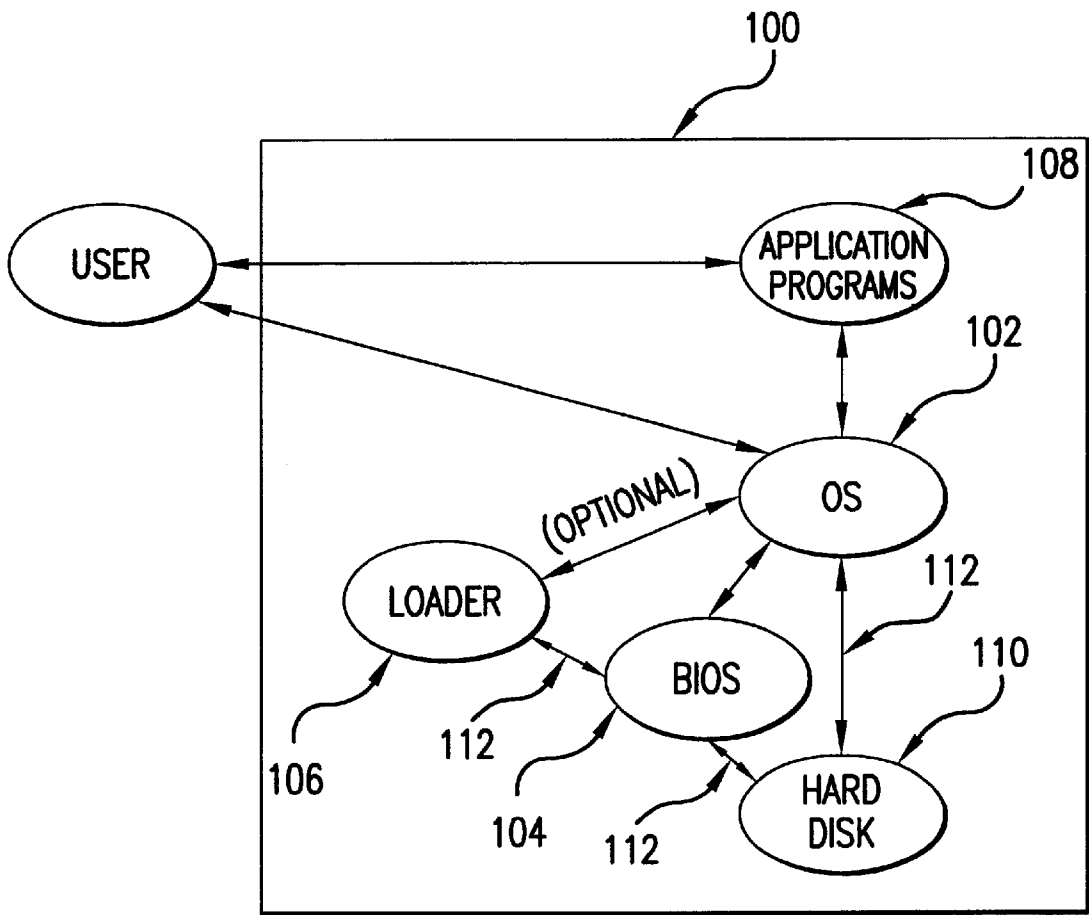


FIG.1

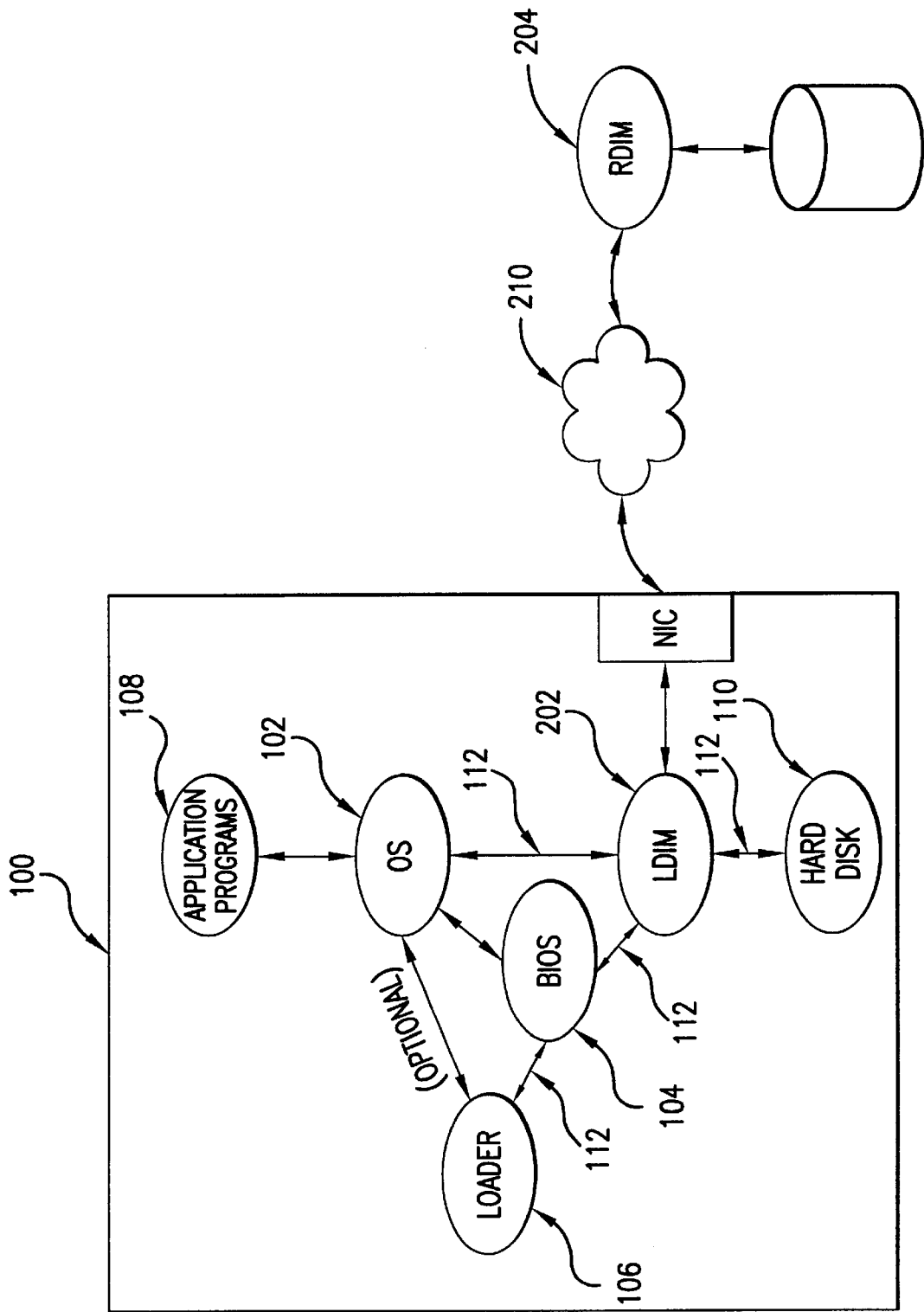


FIG. 2

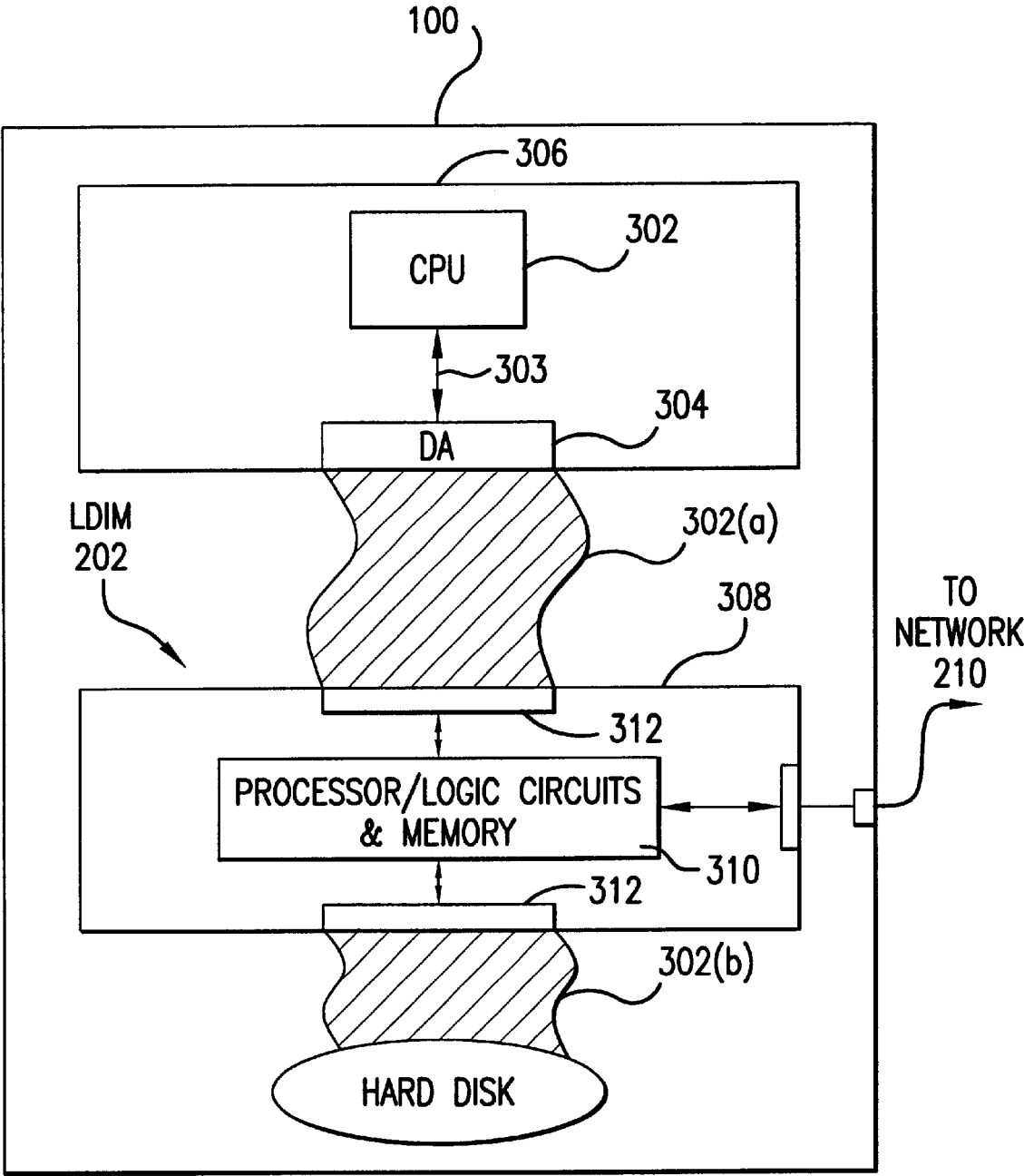


FIG.3

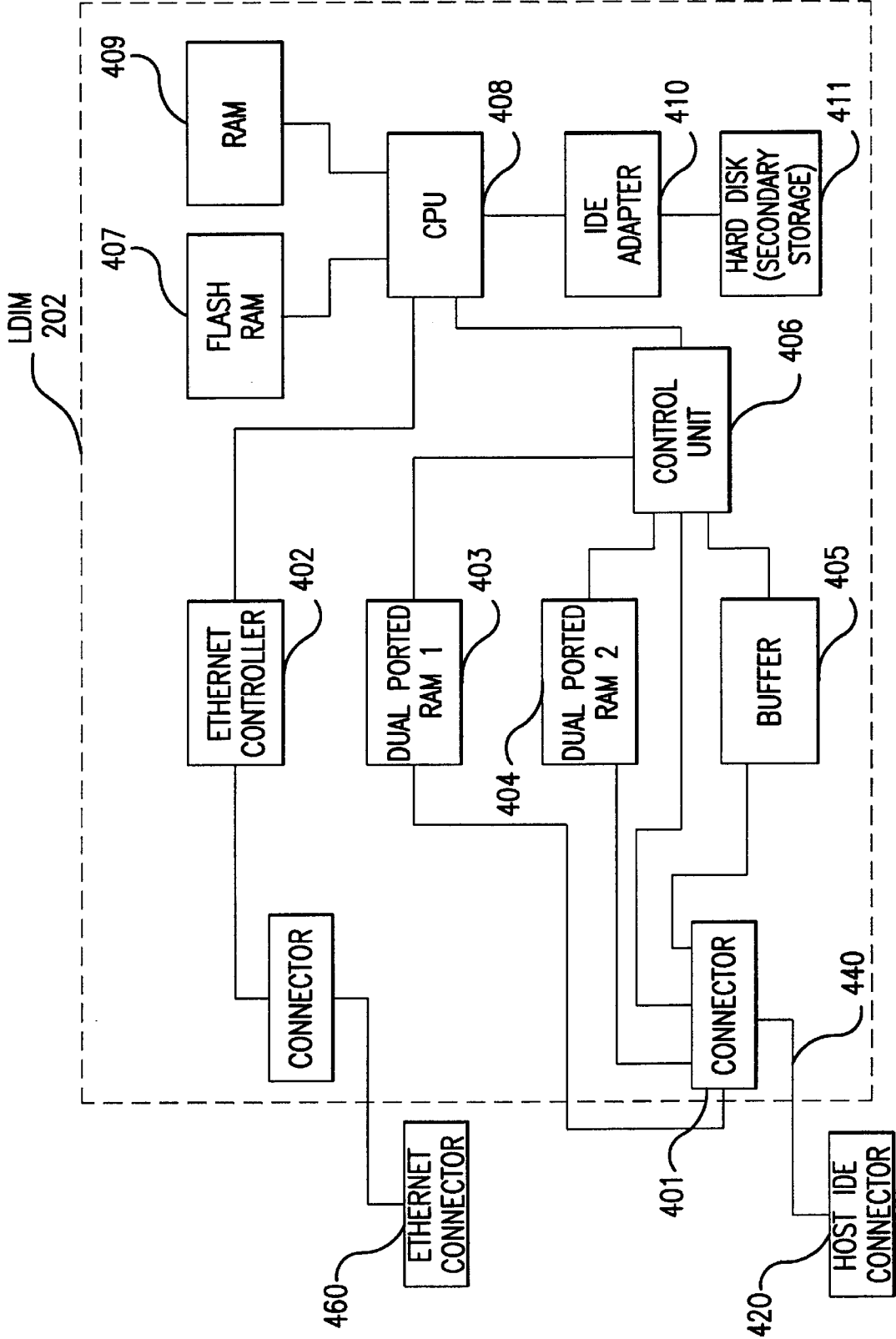
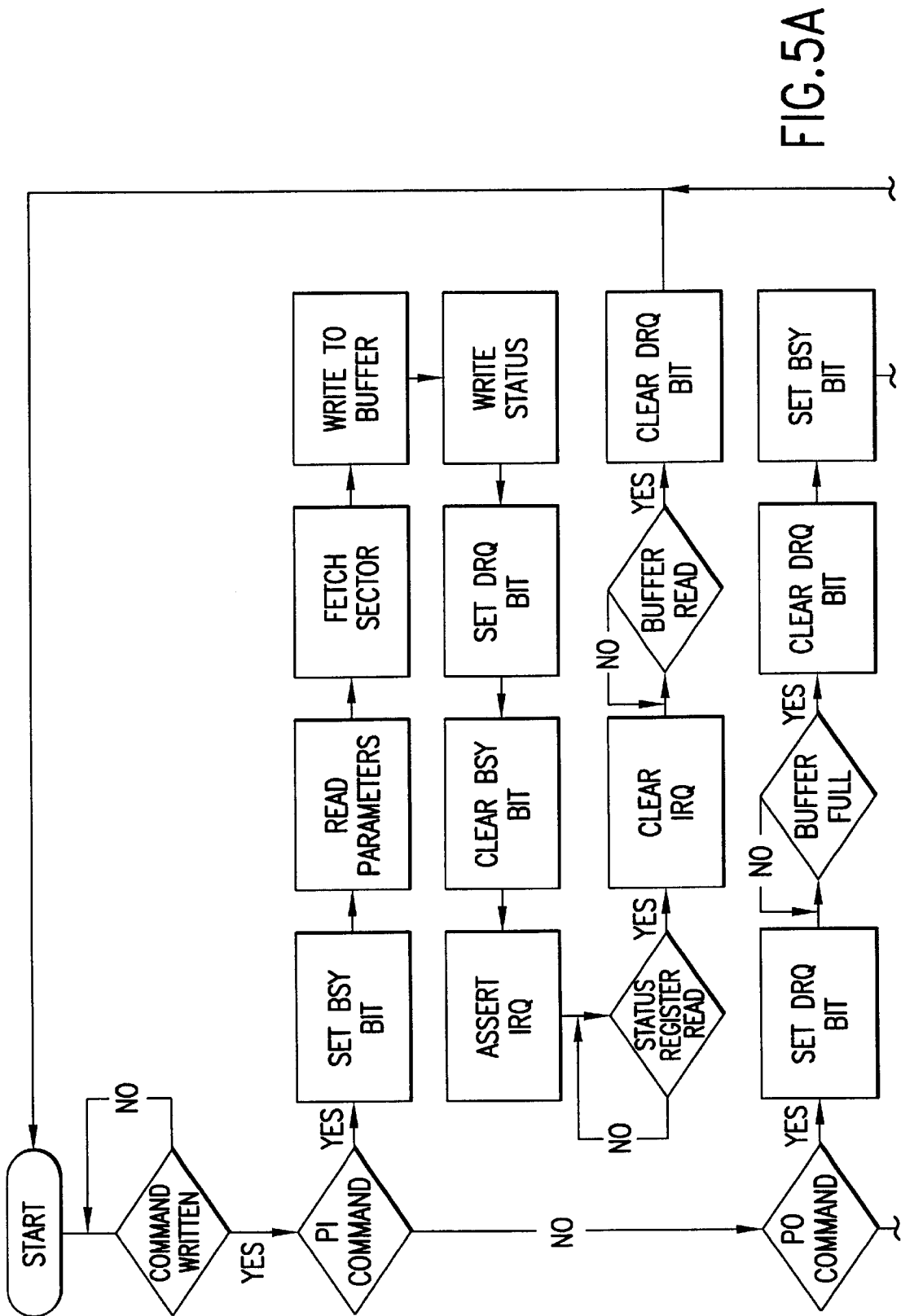


FIG. 4



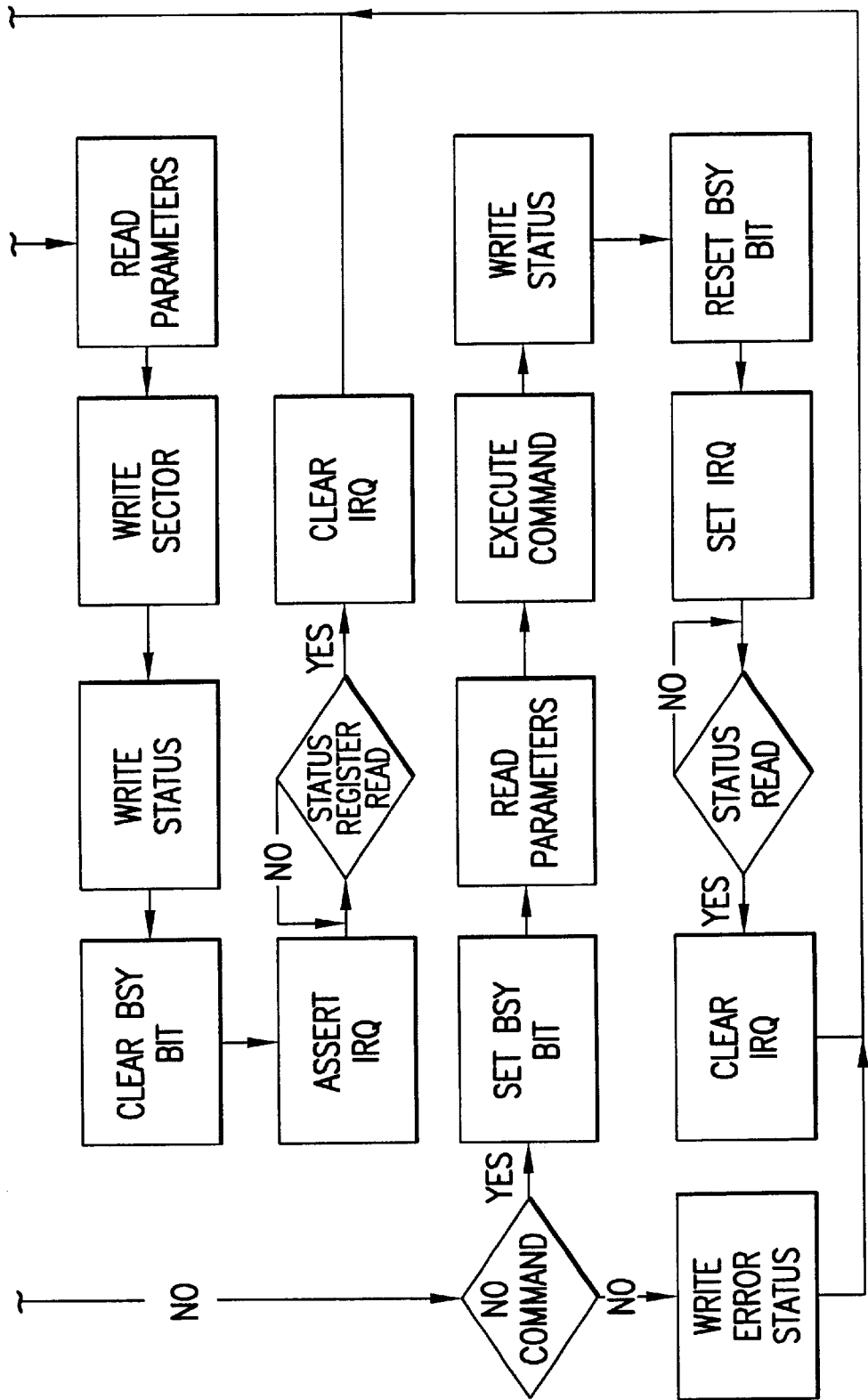


FIG. 5B

HOST COMPUTER PI COMMANDS

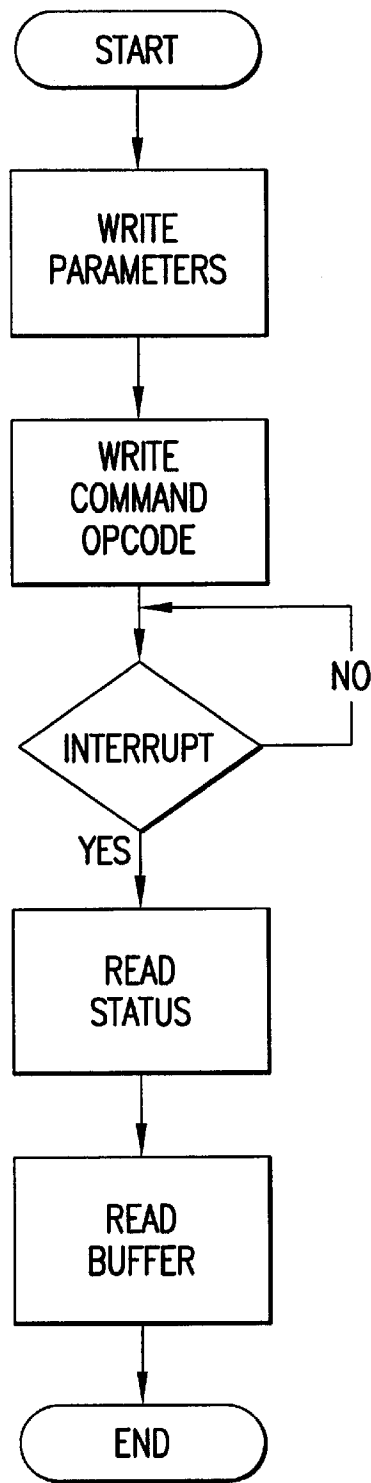


FIG.6

HOST COMPUTER PO COMMANDS

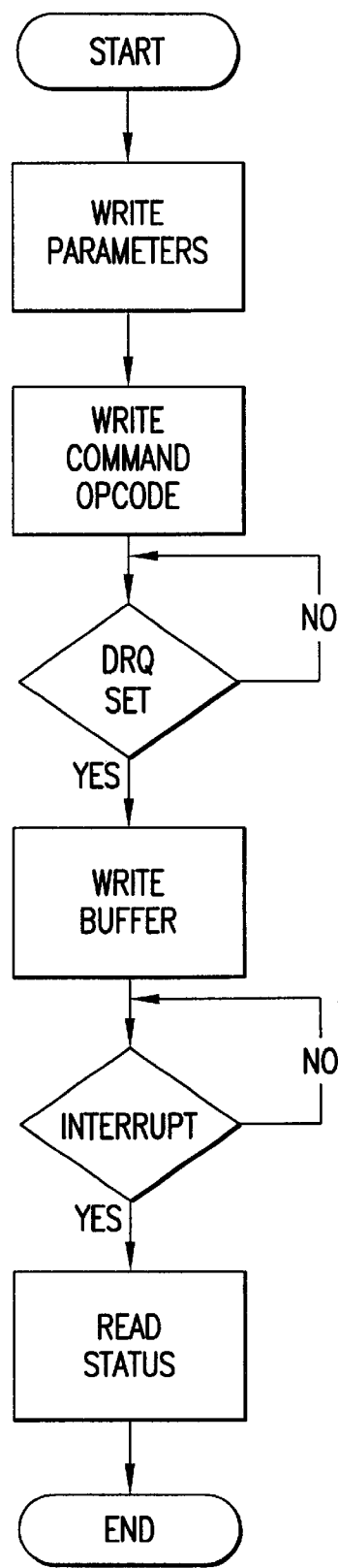


FIG.7

HOST COMPUTER ND COMMANDS

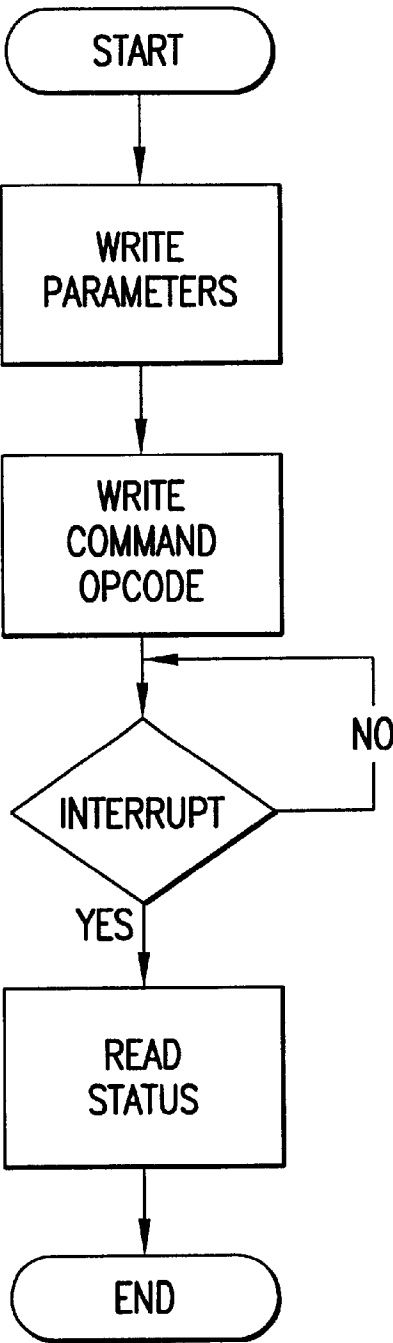


FIG.8

US 6,598,131 B2

1

DATA IMAGE MANAGEMENT VIA EMULATION OF NON-VOLATILE STORAGE DEVICE

This application is a continuation of U.S. patent application Ser. No. 09/690,058 filed on Oct. 16, 2000, now U.S. Pat. No. 6,477,624, which claims the benefit of the following three U.S. Provisional Patent Applications: (1) U.S. Provisional Patent Application No. 60/163,954, filed, Nov. 8, 1999; (2) U.S. Provisional Patent Application No. 60/211,291, filed, Jun. 13, 2000; and (3) U.S. Provisional Patent Application No. 60/240,138, filed, Oct. 13, 2000. All of the above mentioned patent applications are incorporated herein in their entirety by this reference.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention is generally related to persistent storage devices, and, more specifically, to a system and method for enabling the centralized storage and maintenance of persistent storage device data images.

2. Discussion of the Background

In general, the ability to store and access data is critical for computers. For example, when turned on, a computer (e.g., a personal computer ("PC")) accesses, and prepares (or "boots") the operating system from its local persistent storage device (e.g., "hard disk"). Once the booting is finished, the contents of the hard disk are accessible and available to the user. The contents of the hard disk (also referred to as the hard disk's "disk image" or "data image") define the user's personalized environment: the operating system (such as Windows 98 SR-2, Linux, etc.), the software applications (such as word processors, spreadsheet programs, web browsers, etc.), the data files (such as documents, spreadsheets, images, or cookies), and any additional customization (such as whether a particular web browser, such as Netscape Navigator or Internet Explorer, is automatically launched when an HTML file is accessed).

A hard disk is but one example of a persistent storage device. A persistent storage device can be defined as follows:

- (a) it is a physical device that is physically attached to a computer using a standard physical interface (e.g., a hard disk attached with an IDE cable). This physical interface provides the link between the persistent storage device and the computer. (If the persistent storage device under consideration is a hard disk, the physical interface is frequently called a connector and is typically attached to a hardware component on the computer called the disk adapter, which itself provides the logical link between the persistent storage device and the computer);
- (b) it contains a local permanent medium (e.g., magnetic media) for storing a sequence of bits, (i.e., data), typically organized according to a particular file structure. The bits are collectively called the persistent storage device data image (PSDDI), or data image (DI) for short. When the persistent storage device is a hard disk, the persistent storage device data image will frequently be called a "disk image." Typically, the local permanent medium is capable of storing a large amount of data (e.g., more than 10 Megabytes);
- (c) it has the ability to selectively read and write any part of the data image; and
- (d) it allows the computer to which the device is attached to selectively read and write any part of the data image through a standard set of interface protocols.

2

The scope of persistent storage devices includes all hard disk drives implementing interfaces such as ST506/412, ESDI, SCSI, IDE, ATA, ATAPI, ATA-E and EIDE, read/write CD ROM drives, ZIP drives, JAZ drives, floppy drives and the like. In addition, the present invention applies to embedded systems' persistent storage devices, such as, Flash, and DiskOnChip.

Any two "hardware-similar" PCs having the same data image would appear the same to the user. In contrast, if a user's data image is replaced by a significantly different data image, the user will most likely see an unfamiliar desktop displayed on the PC's display screen. What would be even more disturbing and likely to make the PC unusable to the user, is the fact that the new data image would have different software and data files from the original data image. Thus, it is the data image that makes a user's PC the user's "Personal Computer," and it is the most valuable and essentially the only irreplaceable component of the PC.

The conventional PC is "governed" by the contents of its hard disk, and therefore the limits of the installed software become the limits of the user. Once the user's needs change, or grow beyond the capabilities of the installed software, the user has to deal with upgrading or installing a new OS or application software, a costly, time consuming, and frequently aggravating process even for a professional. Moreover, in environments such as offices within large companies or firms, this problem is compounded because the hard drive on each individual PC needs to be accessed in order to perform an upgrade. In addition, such upgrades may cause some existing software not to work properly, in effect corrupting the previously stable data image.

There are several computer architecture models that attempt to solve the above problem. These architecture models and their respective disadvantages are described below.

Network Computer: A network computer (NC) is a lightweight computer with a simple built-in operating system. After booting, it connects to a remote computer for file system access. Software programs reside on the remote computer. Once invoked, they are downloaded to the NC where they execute. The applications are typically based on Java or JavaScript. The problems with an NC are that existing applications have to be reengineered for this platform, and an NC has limited capability to perform computing operations when not connected to the network. If the software on the NC is badly corrupted, it may not be able to boot or access the network and therefore the NC will not be functional. Thus well functioning local software is required for operation. Further, NCs have no notion of providing a remote image to a local computer transparently to the operating system executing on the local computer.

Thin Client: The local computer, termed the thin client, is used mainly for display and user input. Applications execute on a server and the thin client opens a window to the server to interact with the applications. For the thin client to work, a continuous connection from the thin client to the server is needed. A thin client is typically running on a standard computer; however, the thin client technology does not provide any means for remotely administering or upgrading of the computer's software. In addition, thin client technology requires that the data files (such as Word documents) be manipulated on the server, which requires that they not be encrypted during such manipulation. Also, well functioning local software is required for operation. Thin clients are also operating system specific.

Remote booting and Disk-less computers: Some operating systems, such as Unix, MacOS and Windows 95 allow

US 6,598,131 B2

3

computers to boot from an image on a remote computer. This feature is typically used for disk-less computers. However, even if the computers have a disk drive or other persistent storage device, it is only used as a swap space (runtime operating system scratch space), and the contents do not persist across boot sessions. Remote booting and diskless computers do not work off line.

Remote File System Technologies: They allow mounting of a remote file system to a local computer (e.g., NFS). Remote file systems can be provided by a remote computer or by a remote network disk. These technologies allow a computer to access data and programs stored on remote server(s). However, system software built into the operating system is required. Remote file technologies do not allow remote administration of the computer. They also require functioning software on the computer. In addition, remote file system technologies do not work off line whilst the present invention does work off line.

Automatic file propagation: Software tools such as Unix's rdist, allow files to be synchronized across networked computers; however, such tools are operating system and file system specific, and require a functioning operating system for them to work.

What is desired is a system and/or method that overcomes these and other disadvantages of conventional computers and computer architectures.

SUMMARY OF THE INVENTION

The present invention provides a persistent storage device data image management system, or data image management system (DIMS) for short, that is able to solve the above described problems that users encounter when upgrading and/or maintaining their computers.

According to the present invention, the DIMS completely de-couples a persistent storage device data image "seen" by the computer from a persistent storage device attached to the computer (also referred to as the local persistent storage device (LPSD)). The DIMS includes a local data image manager (LDIM), which is required to be installed (either by the manufacturer, the distributor, a user, or a technician) on the user's computer, a remote data image manager (RDIM), and a remote persistent storage device (RPSD). The LDIM communicates with the RDIM through a direct communication link or through a communication network (e.g., a local area network, a wide area network, the Internet, the public switched telephone network, a wireless network, etc). The RDIM can store data on and retrieve data from the RPSD.

In an environment where an LDIM has been installed in a computer having a "local" persistent storage device (LPSD), the DIMS allows for the storing of the LPSD's data image on the RPSD, with the LPSD serving as a persistent, consistent cache of the data image. The data image stored on the RPSD is referred to as the "master data image" and the data image cached on the LPSD is referred to as the "local data image" or "cached data image." In general, there is no requirement that the LPSD and the RPSD be of the same type. For instance, the LPSD could be a DiskOnChip and the RPSD could be a hard disk. Also, the RPSD may be a specialized smart device rather than being installed in a general purpose computer.

The purpose of the LDIM is to imitate the LPSD. That is, the LDIM, from the computer's perspective, appears exactly like the LPSD. More specifically, the LDIM functions to intercept and process requests that are intended to be received by the LPSD, which may not be in fact installed in

4

the computer. Common are read/write requests specifying an address (for example, in the case where the LPSD includes a hard disk, the read/write requests specify a sector of the hard disk).

Upon intercepting a read request, which specifies an address, the LDIM is programmed to determine whether the portion of the cached data image that is stored at the specified address is up-to-date (i.e., whether the portion of the cached data image reflects the latest changes made to the master data image). In one embodiment, this feature is implemented by having the LDIM request a "modified-list" from the RDIM each time the LDIM is powered on and (optionally) to have the RDIM provide to the LDIM updates to the modified list whenever a modification to the master data image occurs. The "modified-list" is a list of all the "parts" or "portions" of the master data image that have been modified since the last time the LDIM was informed of modifications to the master data image. (For example, if the master data image is a data image from a hard disk, the list of parts could be a list of the disk's sectors.) Thus, if the LDIM receives a read request specifying an address that is on the modified list, the LDIM will know that the portion of the cached data image stored at the specified address is not up-to-date.

If the LDIM determines that the cached data image has the most up to date version of the requested data, then the LDIM (1) retrieves the requested data from the LPSD by issuing a read request to the LPSD and (2) passes the retrieved data back to the component or device from which it received the request. If the cached data image does not have the most update version of the requested data, then it must be stored on the RPSD (i.e., the master data image). In this case, the LDIM transmits to the RDIM a read request message, which may include the address specified in the intercepted read request. Upon receiving the read request message, the RDIM locates and reads the requested data from the master data image stored on the RPSD and then transmits the data back to the LDIM.

Upon intercepting a write request, the LDIM may write the data to the LPSD, if there is one, and transmits the data to the RDIM so that the RDIM can update the master data image thereby ensuring that the master data image is up to date. The LDIM may either transmit the data to the RDIM substantially concurrently with writing the data to the LPSD or wait until some later time (e.g., if the computer is not currently connected to the network or if the network is heavily loaded).

On requests other than read or write request, such as PND (Program Non Data request for IDE hard disks), the LDIM returns a response as required by the standard protocol for communicating with the LPSD.

It is contemplated that in some embodiments there will be no LPSD. In this case, there is no cache as described above. Instead, all read/write requests for data that are received by the LDIM are transmitted to the RDIM. In the case of a read request, the RDIM retrieves the requested data and transmits the data back to the LDIM. In this manner, a user of the computer has access to his or her personalized data image even when the computer is not equipped with a local hard disk or other persistent storage device. It is also contemplated that to gain the greatest benefit from the invention the computer in which an LDIM is installed should, as often as is possible, be connected to a network so that the LDIM can communicate with an RDIM.

From now, and without limiting the scope of the invention, the invention and its benefits will be described

US 6,598,131 B2

5

with respect to the particular embodiment where the LPSD is a hard disk. Once the DIMS is in place, the user need not concern him or herself with the task of upgrading his or her operating systems, application programs, data files, etc., following setting the appropriate agreements with the organization in charge of managing the master data images on RPSDs. This is because software patches and upgrades can be first performed on the master data image by an experienced system administrator. After the system administrator performs the upgrade on the master data image, the DIMS transparently ensures that these patches and upgrades are propagated to the local hard disk, as described above. Similarly, as described above, the DIMS automatically backs up all data files that are stored on the local hard disk that have been modified. This is accomplished by the LDIM transmitting the modified files (or sectors) to the RDIM so that the RDIM can update the master data image. In this manner, the master data image is kept up to date.

Additionally, the DIMS can cache multiple master data images on the local hard disk. This is advantageous where the computer has more than one user and each user has his or her own personalized data image. The DIMS uses a standard coherent caching algorithm (such as described in the standard textbook: Almasi and Gottlieb, *Parallel Computing*, 2nd edition, 1994, the entire contents of which are incorporated herein by this reference.) and implementation to store the cached data images and maintain their coherency with the corresponding master data image. When the LDIM is unable to communicate with the RDIM, the computer in which it is installed can still operate to the extent that the required software and data is cached on the local on the local hard disk.

Preferably, the DIMS provides this functionality below the operating system and all of its components (including device drivers) and BIOS routines specific to the hardware of the computer. Thus, the DIMS is completely transparent to the operating system and all applications of the computer. This allows the DIMS to be independent of any operating system or software installed on the computer. At the same time, it has the ability to provide the computer with any type of operating system compatible with the computer's hardware, software, and other data.

This enabling technology provides a rich gamut of functionality that completely changes the way computers are utilized. A user can use any "hardware compatible" computer as their "Personal Computer," as the new computer transparently, without the user's intervention, obtains over a network only those parts of the user's data image needed for the current execution, with the other parts following later. For instance if the user wants to start execution by editing a document and then at a later point in time create and send an e-mail, a word processor will be downloaded before the e-mail program. A user's computer can be replaced by a new one with the ease of "plug and play," retaining all the user's desired previous software, data, and settings. The user is presented with practically unlimited disk space, as the size of the master data image is not constrained by the size of a local disk.

The software and data cached on the local disk provide instantaneously for the normal needs of the user, thus minimizing the network traffic between the location where the master copy is stored and an individual computer. As the user's software does not execute remotely, data files are kept private through encryption—which can be done even on the local hard disk, with LDIM encrypting and decrypting the data as needed.

The DIMS is easy to integrate into existing platforms, provides, for the first time, a complete automated manage-

6

ment and administration capability in LANs and over the Internet, while maintaining fully the rich computer functionality. The DIMS creates the following benefits: increased user satisfaction and productivity, removal of the need for users to spend time on administration or wasting time waiting for somebody to repair the damage they may have caused to their local disk contents, and in general tremendous savings in both the explicit and implicit parts of total cost of ownership.

In today's computers, to access a hard disk, the following sequence of steps are performed:

1. An application wishing to read or write a file issues a request to an operating system API for such action.
2. The operating system checks if the request can be serviced from its file cache, if such cache is maintained.
3. On a miss, or write through, the operating system directs the request to an appropriate device driver for the physical device to which the request was made.
4. Optionally, the device driver may issue the request to the computer's BIOS.
5. The device driver (or BIOS) issues the request to a disk adapter, which is a component typically installed on the motherboard of the PC.
6. The disk adapter uses a physical connection to send the request to the controller of the local hard disk.

It is an object of the present invention to implement the LDIM to intercept the request from the disk adapter so that the controller of the local persistent storage device will not receive it. Alternatively, it is an object of the invention to implement the LDIM to intercept the request from the device driver (or BIOS) so that the disk adapter does not receive it.

There are a number of ways in which this interception can be done including system management mode, a PC card, building it with new chips on the motherboard or the persistent storage device, or building the functionality into the adapter chip. The Alternative Embodiments section of this document elaborates on these embodiments.

It is another object of the present invention to allow the DIMS to encrypt writes and decrypt reads using a password, a pass-phrase, or a key supplied by system software or the user. As commonly done, the key itself could be kept encrypted and only be decrypted after a password or a pass-phrase are supplied. The obvious advantage of encryption is to secure the local data, the remote data, and the data transferred over the network. Furthermore, this functionality is transparent to the user, the operating system, and software applications.

It is another object of the present invention to allow the DIMS to work with any operating system presently available on the market (including DOS, Windows 98/NT/2000/ME, Unix, Novell, OS/2, BeOS) or any future operating system that supports any of the standard persistent storage device interfaces.

It is another object of the present invention to allow the DIMS to work with any standard hard drive interface.

It is another object of the present invention to make the above functionalities available for any existing or future computer hardware that uses any of the current or future persistent storage device interfaces.

It is another object of the present invention to provide a transparent mechanism for upgrading software and hardware drivers by means of pulling changes as needed when the computer is connected to the network. This is as opposed to current operating system specific push mechanisms, such as Unix's rdist.

It is another object of the present invention to provide a mechanism for allowing users to roam from computer to computer and receive their personal operating system, personalization customizations, applications, and data files, no matter what operating system, applications, and data files were previously being used on that computer.

It is another object of the present invention to provide a mechanism allowing the DIMS to transparently present a large amount of storage space (bound only by the addressing size limitations), regardless of the amount of space available on the local physical persistent storage device.

Still other objects and advantages of the invention will in part be obvious and will in part be apparent from the specification.

Further features and advantages of the present invention, as well as the structure and operation of various embodiments of the present invention, are described in detail below with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated herein and form part of the specification, illustrate various embodiments of the present invention and, together with the description, further serve to explain the principles of the invention and to enable a person skilled in the pertinent art to make and use the invention. In the drawings, like reference numbers indicate identical or functionally similar elements. Additionally, the left-most digit(s) of a reference number identifies the drawing in which the reference number first appears.

FIG. 1 depicts a functional block diagram of a standard PC indicating the connection and interaction of a typical persistent storage device;

FIG. 2 depicts a computer and a data image management system (DIMS), according to one embodiment, for improving the operation of the computer.

FIG. 3 depicts a schematic of a PC with an installed local data image manager (LDIM).

FIG. 4 depicts a block diagram of the components of an exemplary LDIM in accordance with the present invention;

FIG. 5 depicts a flowchart of the LDIM control program;

FIG. 6 depicts a flowchart of the PI (PIO In) commands executed by a computer;

FIG. 7 depicts a flowchart of the PO (PIO Out) commands executed by a computer; and

FIG. 8 depicts a flowchart of the ND (Non Data) commands executed by a computer.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The following detailed description of an embodiment of the invention is of the best presently contemplated mode of carrying out the invention. This description is not to be taken in a limiting sense, but is made merely for illustrating the general principles of the invention.

FIG. 1 depicts a functional block diagram of a conventional computer 100 (also referred to as "PC"). As shown, computer 100 includes an operating system (OS) 102, a basic input/output system (BIOS) 104; a loader 106, a variety of software programs 108 and a local persistent storage device (e.g., hard disk) 110. As the figure exemplifies, operating system 102, BIOS 104, and loader 106 (through the BIOS) access storage device 110 through a standard interface/protocol 112 (e.g., the ATA/IDE interface/protocol). Conventionally, applications programs 108 do not

access storage device 100 directly; application programs 108 rely on services provided by operating system 102 to access data stored on storage device 110.

FIG. 2 depicts computer 100 and a data image management system (DIMS), according to one embodiment, for improving the operation of computer 100. The DIMS is a client/server system, and thus includes a client 202 and a server 204. Client 202 is referred to as the "local data image manager" (LDIM) and server 204 is referred to as the "remote data image manager" (RDIM). The DIMS also includes a persistent storage device (PSD) 206 that can be read from and written to by RDIM 204. PSD 206 is referred to herein as "remote PSD 206" or "RPSD 206" because it is "remotely" located from computer 100. That is, RPSD 206 is not directly coupled to computer 100.

As shown in FIG. 2, LDIM 202 is installed in computer 100. More specifically, from a functional point of view, LDIM 202 is installed between storage device 110 and OS 102, and BIOS 104. LDIM 202, as shown in FIG. 2, communicates with OS 102 and BIOS 104 using standard interface 112; the same interface used by OS 102 and BIOS 104 to communicate with storage device 110. Additionally, LDIM 202 may be connected to storage device 110 and can read data from and write data to storage device 110 using standard interface 112. To OS 102 and BIOS 104, LDIM 202 "pretends" that it is storage device 110. Thus, LDIM 202 is completely transparent to OS 102 and BIOS 104.

LDIM 202 and RDIM 204 communicate with each other through a direct communication link or network 210 (e.g., a local area network, a wide area network, the Internet, the public switched telephone network, a wireless network, etc.).

As described in the Summary of Invention section, the DIMS allows for the storing of a data image on RPSD 206 with storage device 100 serving as a persistent, consistent cache of the data image. A data image stored on RPSD 206 is referred to as a "master data image" and a data image stored on storage device 110 is referred to as a "cached data image." RPSD 206 can be configured to store more than one master data image and storage device 110 can be used to cache more than one master data image.

Optionally, in one embodiment, LDIM 202 includes a "mini-booter" software program (not shown). In this embodiment, when computer 100 is powered on, LDIM 202 "forces" the "mini-booter" into computer 100's CPU (See FIG. 3). This is done by emulating a disk with the mini-booter installed as a loader. The mini-booter functions to authenticate the user (i.e., it may perform username/password verification). Additionally, if the DIMS has more than one RPSD 206, the user is prompted to select one. Preferably, the user will select the RPSD 206 that stores the master data image(s) that "belong(s)" to the user. After the mini-booter receives the selection from the user, the mini-booter communicates with LDIM 202 (this communication can use custom IDE commands, or reads and writes to reserved sectors) to request that LDIM 202 contact the RDIM 204 associated with the selected RPSD 206 to determine the list of available master data images for this user. (If the server cannot be contacted, some number of most recent user names, passwords, and images are cached on the LDIM 202, assuming a cache exists, and only those users can gain access). The mini-booter displays the list of available master data images and prompts the user to select one. The mini-booter communicates the selection to LDIM 202 and then either reboots the computer or resets the BIOS's disk geometry table to the geometry of the selected

master data image. The latter case is preferred if the BIOS supports such a feature. In the former case, LDIM 202 emulates the selected image including the geometry of the image.

The mini-booter may also provide an interface for configuring and maintaining LDIM 202. Options such as whether to use Dynamic Host Configuration Protocol (DHCP) or a static IP address can be set from this interface.

As is evident from FIG. 2, LDIM 202 functions to intercept requests (for example, read/write requests) that are intended to be received by storage device 110. After a master data image is selected, upon intercepting a read request, LDIM 202 is programmed to determine whether the cached data image or the selected master data image has the most up to date version of the requested data. In one embodiment, this feature is implemented by having LDIM 202 request a “modified list” from RDIM 204 each time LDIM 202 is powered on and (optionally) to have RDIM 204 provide to LDIM 202 updates to the modified list whenever a modification to the master data image occurs. The “modified list” is a list of all the sectors of the master data image that have been modified since the last time LDIM 202 was informed of modifications to the master data image. Thus, if LDIM 202 receives a read request for a sector that is on the modified list, LDIM 202 will know that the cached data image (i.e., storage device 110) does not have the most up to date version of the data.

If LDIM 202 determines that the cached data image has the most up to date version of the requested data, then LDIM 202 retrieves the requested data from storage device 110 and passes the data back to the component or device from which it received the request. If the cached data image does not have the most update version, then it must be stored on the RPSD 206 (i.e., the selected master data image). In this case, LDIM 202 transmits a read request to RDIM 204. Upon receiving the read request, RDIM 204 locates and reads the requested data from the selected master data image stored on RPSD 206 and then transmits the data back to LDIM 202. Upon intercepting a write request, LDIM 202 is programmed to write the data to storage device 110 and to transmit the data to RDIM 204 so that RDIM 204 can update the selected master data image thereby ensuring that the master data image is up to date. LDIM 202 may either transmit the data to RDIM 204 substantially concurrently with writing the data to storage device 110 or at some later time (e.g., when network traffic is low).

It is envisioned that a user who uses the DIMS would initially copy the data image stored on storage device 110 onto RPSD 206 and then always select that image as the master data image. Thus, in most cases the selected master data image on RPSD 206 would initially be identical to the cached data image on storage device 110. Of course, a small amount of space would need to be set aside on storage device 110 for cache data and any other data needed, such as encrypted keys. It is further envisioned that the user would then allow a professional system administrator to make updates to the master data image using RDIM 204.

The various data images stored on RPSD 206 may not require each its own fully dedicated storage space. For instance, if two data images contain the same word processor, that word processor may be stored only once. RDIM 204 will deliver the word processor as needed to both of the cached data images.

RDIM 204 preferably includes image manipulation tools. The image manipulation tools allow system administrators to manipulate master data images stored on RPSD 206. For

example, an administrator could create new images, copy images and delete images. In addition, it is advantageous to allow administrators to manipulate the contents of a master data image stored on RPSD 206. For example, software installation, registry manipulation, and operating system upgrades or restructuring can be performed on a master data image. The image manipulation tools allow centralized administration and upgrading and/or restructuring of operating systems, applications, and user data.

These updates to the master data image would be automatically propagated to the cached data image by virtue of the fact that LDIM 202 is made aware of the updates. As explained above, when LDIM 202 is powered on it requests the modified list from RDIM 204 and RDIM 204 sends to LDIM 202 updated information to be added to the modified list.

The DIMS can provide functionality to computer 100 even when storage device 110 is not present in computer 100. In this situation, LDIM 202 merely forwards all read/write requests to the appropriate RDIM 204. However, to minimize the network traffic and delays, using storage device 110 as a cache is necessary. When storage device 110 is used as a cache, the vast majority of requested blocks (this is likely to include the OS, recently used application, and data files) will most likely be stored in the cache, thus, no network traffic is needed. It should also be noted that the propagation of write requests from LDIM 202 to RDIM 204 for the purpose of updating the master data image can be timed to best utilize the network bandwidth. As an additional feature, since LDIM 202 receives all read requests, LDIM 202 can monitor software usage. That is, LDIM 202 can “meter” the use of any particular software program or data file and gather statistics on the usage patterns of various software programs.

Referring now to FIG. 3, there is shown a hardware diagram of computer 100 and LDIM 202 according to one embodiment of the invention. In a typical computer, a ribbon connects a disk adapter 304 on the computer’s motherboard 306 to a storage device (e.g., hard disk). However, in this embodiment of the invention, LDIM 202 comprises an LDIM card 308 (LDIM card 308 is a standard form card similar to an Ethernet card) that is connected to disk adapter 304 by a ribbon cable 302(a) and connected to storage device 110 by another ribbon cable 302(b), which is similar to ribbon cable 302(a). That is, in this embodiment of the invention, storage device 110 is not connected to disk adapter 304 as is the case in conventional computers. LDIM card 308 is equipped with an embedded processor, logic circuits, and memory 310 for enabling LDIM 202 to perform its functions.

To disk adapter 304 and storage device 110, LDIM 202 presents standard connectors and hardware/software interfaces 312, and therefore, LDIM 202 is completely interoperable with standard computer hardware and interfaces. To disk adapter 304, LDIM 202 “pretends” that it is storage device 110; to storage device 110, LDIM 202 “pretends” it is disk adapter 304. As the CPU 302 and software running on the computer 100 (including the OS) are not “aware” of this replacement, reads and writes that are transmitted by the disk adapter 304 are intercepted and processed by LDIM 202 as described above.

There is no need to modify the existing motherboard bus technology or the CPU 302 to disk drive adapter interface 303 or the adapter to the disk drive’s controller interface: disk adapter 304 can use any protocol to interface with storage device 110 and/or LDIM 202, for example including ST506, SCSI, IDE, ATA, ATA-2-5, EIDE, ESDI.

Furthermore, disk adapter **304** can be coupled to the CPU **302** using a variety of different buses, for example, PCI, ISA, VESA LB, EISA, and MCA.

The particular embodiment of LDIM **202** that is described below implements the IDE protocol and consists of PIO modes 0 through 2. It is straightforward to implement other modes of operation by implementing their respective specifications. This embodiment is concerned with the method of re-directing the host's reads and writes. As such, the embodiment does not implement multi sector reads and writes. It is straightforward to implement the remainder of the specification and implementing PIO modes 0 through 2 does conform to the specification. Note that PI and PO commands include (i.e., specify) a sector.

FIG. 4 depicts a schematic of LDIM **202**, which is for connecting to a computer's primary IDE interface **420**, via a standard 40 pin IDE connector **440**, and to a network interface **460**. Element **401** is a 40 pin IDE flat cable connector. Element **402** is a network interface, which enables software executed by CPU **408** to communicate with other programs (e.g., RDIM **204** programs) over a network or communication link.

Element **403** is a dual ported RAM. In the illustrated embodiment, the dual ported RAM captures the data written by CPU **302** via its adapter **304** to the IDE interface. In particular, dual ported RAM **403** stores the data written onto the IDE Command register block (Feature Register, Sector Count, Sector/Cylinder/Drive-Head Registers and Command register), and Control Register Block (Control Register). Since element **403** is a dual ported RAM, CPU **408** can access its contents at any time thereby reading the parameters written by CPU **302** to the IDE drive.

Element **404** is a second dual ported RAM. It stores the Error and Status registers. It is written by CPU **408** and can be accessed by CPU **302** via adapter **304**.

Element **405** is a buffer that stores data exchanged between adapter **304** and LDIM **202**.

A Control Unit **406** is provided for controlling the dual ported RAMs and the buffer to implement the IDE protocol.

LDIM **202** also includes internal RAM **409** that can potentially be used by the software module that implements the standard caching algorithm as a fast buffer cache and for storing that software module and other modules. A flash RAM **407** stores an initial control program loaded by the CPU **408**, as well as the control program for control unit **406**.

Finally, an IDE adapter **410** is connected to the CPU **408** and to the (optional) secondary storage device **411**.

The elements **403**, **404**, **405**, **406** can be implemented with a Xilinx XC4000 FPGA, or with a specialized ASIC. Element **408** can be any CPU, for example an Intel 80486.

In a preferred embodiment of the present invention, the objectives are achieved by connecting the aforementioned LDIM **202** to the standard IDE connector **420** of the primary IDE interface of computer **100**. Preferably, a new Ethernet connector **460** is added in an empty PCI slot of computer **100**, and the Ethernet connector **460** is linked with a cable to network interface **402**.

Because IDE adapter **304** reads from the disk through the IDE connector **420**, and since LDIM **202** is connected to IDE connector **420**, if LDIM **202** correctly follows the IDE protocol, it appears as a standard IDE disk drive. Thus the present invention achieves its objectives without requiring the modification, removal or replacement of the system IDE adapter or the use of specialized device drivers or BIOS, or additional software.

Attaching LDIM **202** in accordance with the present invention to IDE connector **420** allows LDIM **202** to moni-

tor all the information being transmitted by CPU **302** to the disk drive **110**. This includes all the commands because the set of registers implemented with the dual ported RAMs is identical to the set of registers of disk drive **110**.

The flowcharts depicted in FIGS. 5-8 illustrate an example of disk activity that occurs when an LDIM **202** in accordance with the present invention is installed in a computer. The flowchart depicted in FIG. 5 illustrates the process performed by the LDIM **202** software (not shown) that is stored in flash RAM **407** and that functions to simulate any IDE command.

As depicted in FIG. 5, no action occurs until CPU **302** writes a command to the command register (not shown) that resides in dual port RAM **403**. When this occurs, control unit **406** generates an interrupt to CPU **408**. An interrupt service routine immediately reads the command from the command register, and selects either a PI, PO, or ND handling module depending on the command class of the command written into the command register (PI, PO, or ND).

When a PI command is executed (i.e., CPU **302** is seeking to retrieve a data sector), the PI handling module sets the BSY bit, reads the parameters from the dual ported RAM **403**, and then fetches the sector that is being requested by CPU **302**. The sector is either fetched from the cache **110** or from an RPSD **206**. As described above, the sector is fetched from cache **110** if the data stored there is up-to-date, otherwise the sector is fetched from RPSD **206** (i.e., the PI handling module sends a read request, which indicates the requested sector, to RDIM **204**, RDIM **204** retrieves the requested sector from RPSD **206**, and RDIM **204** sends the requested data sector back to the PI handling module). After the data sector is fetched from either local disk **110** or an RPSD **206**, it is stored in buffer **405**, dual ported RAM **404** is updated with the new status information, the DRQ bit is set, the BSY bit is cleared and, if interrupts are enabled, an interrupt to CPU **302** is generated. The PI handling module then waits until CPU **302** reads the status register. When this happens, the IRQ line is de-asserted, and the PI handling module waits until the contents of buffer **405** have been completely read by CPU **302**. Then the PI handling module clears the DRQ bit and control passes back to the beginning.

When a PO command is executed (i.e., CPU **302** is seeking to write data to a sector), the PO handling module first sets the DRQ bit and then waits until buffer **405** is full. The handling module then clears the DRQ bit and sets the BSY bit, then reads the parameters from the dual ported RAM **403**, and finally writes the sector specified in the PO command. The actual writing is handled by the software module that implements the above mentioned caching algorithm, which may perform the write to the fast buffer cache, the local persistent storage device **110**, or to RDIM **204** or to any combination thereof. After the sector is written, the status is written and the BSY bit is cleared. At this point the IRQ line is asserted, and the handling module waits until CPU **302** reads the status register. Then the IRQ line is cleared and the process goes back to the beginning of the control program main loop.

When an ND command is executed, the ND handling module sets the BSY bit, reads the parameters from the dual ported RAM **403**, and then executes the command. The status is then written, the BSY bit is reset, and the IRQ line is asserted. The handling module waits until the status register is read by CPU **302**, and then clears the IRQ line. The process then goes back to the beginning. When a non-valid command is detected, the handling module writes the correct error value to the error register and the process returns to the beginning of the main loop.

US 6,598,131 B2

13

FIG. 6 depicts the flowchart of the PI commands executed by CPU 302. The parameters are written onto the command block registers and the command op-code is written onto the command register. The computer's operating system then waits for an interrupt. When LDIM 202 asserts the IRQ line, CPU 302 reads the status register (which makes LDIM 202 de-assert the IRQ line), and reads the buffer.

FIG. 7 depicts the flowchart of the PO commands executed by CPU 302. The parameters are written to the command block registers and the command op-code is written to the command register. Then the operating system 102 waits for the DRQ bit on the status register to be set. When LDIM 202 does this, the operating system 102 writes the buffer and waits for an interrupt. When LDIM 202 asserts the IRQ line, CPU 302 reads the status register (which makes LDIM 202 de-assert the IRQ line).

FIG. 8 depicts the flowchart of the ND commands executed by CPU 302. The parameters are written to the command block registers and the command op-code is written to the command register. Then the operating system waits for an interrupt. When LDIM 202 asserts the IRQ line, CPU 302 reads the status register (which makes LDIM 202 de-assert the IRQ line).

Although product names of several specific circuit elements used in preferred embodiments of the present invention have been identified herein, it is recognized that any circuit element chip which performs functions similar or equivalent to the components described here may be substituted with no change in the functionality of the invention. The present invention is therefore not limited to the specific circuit elements identified herein, and changes in such circuit elements may be made to accommodate such consideration as cost reduction or other reasons.

Alternative Embodiments of the Invention:

While the invention has been particularly shown and described with reference to preferred embodiments thereof, it will be understood by those skilled in the art that the foregoing and other changes in form or detail may be made therein without departing from the scope of the invention.

There are a number of ways to intercept the read/write requests that are intended to be received by a persistent storage device and to implement LDIM 202. For example, on Intel 386 and above processors, the system management mode can be employed to redirect communication to/from disk adapter 304 to a specialized program running on CPU 302. In this case, either the ports for accessing adapter 304 need to generate system management interrupts or adapter 304 needs to be modified to generate system management interrupts after some quantum of the request has been made (e.g., for a write of a single sector, after the 512 bytes of data have been collected by the adapter). In the latter case, adapter 304 must be able to handle requests for disk accesses normally, for example when the system management mode code needs to access the persistent storage device 110.

Another approach is to add the interception and LDIM implementation on the motherboard 306 itself; the motherboard 306 would be augmented to route requests for the adapter to a physical unit (not shown) on the motherboard that would handle the interception and implement the LDIM functionality.

Another approach is to add the interception and implementation of the LDIM onto the physical persistent storage device 110. This functionality would be added before the device's controller (not shown) handles requests.

Another approach is to augment the adapter 304 on the motherboard 306 to perform the interception.

The approach employed in the embodiment described in detail above is to insert a card in between the cable connecting an x86 motherboard 306 to an IDE hard disk 110. This embodiment should in no way be seen as limiting the scope of the present invention to this type of embodiment or to these types of components.

14

It should be noted that when the computer 100 already has a network interface card (NIC), the LDIM need not have its own NIC. That is, the LDIM can be connected to the existing NIC and provide routing and Network Address Translation (NAT) for the computer. The advantage of this configuration is that the computer does not require an additional IP (Internet Protocol) address. It is also possible to use a single network connection and single IP address for both the LDIM and the computer. In this case, internal logic would determine whether a network packet was directed to the LDIM, or whether to pass the packet through to the computer.

Benefits of the Invention:

Because the DIMS is capable of updating the cached data image "at its convenience," that is in a "pull" rather than a "push" mode, the DIMS allows all of a computer's software (operating system, software applications, and other data) to be installed and maintained at a remote site, by modifying the data image stored there. This gives users an unprecedented benefit as it requires no support from the local operating system or application software.

By performing the updates pro-actively and below the operating system level, substantial performance and maintainability benefits are achieved in a platform independent manner. This includes maintaining the software so its most current version is available, and easy installation of additional software.

Frequently, the contents of various data images partially overlaps, this is true for instance for operating system files for images in which the same operating system is installed. In this case, the common parts of a set of images may be stored only once, and then upgrades and modification to the common parts may be more efficient.

In contrast to existing vendor centric solutions (thus of limited capability solutions) the present invention provides a comprehensive solution to remotely install and administer any operating system on any computer (as long as the operating system and the host hardware are compatible). This results in a dramatic reduction in the total cost of computer ownership.

A computer can boot a different, up-to-date, or customized version of the operating system and software each time it boots. For instance, a user following an agreement with his Internet Service Provider (ISP), may at each boot time get a different version of trial software, updated operating systems components, or new advertising images as wallpaper.

The user may also transparently use Application Service Providers (ASPs). In this case, his or her master data image will include the ASP's application software, which he will access as if it were resident on a local persistent storage device. The ASP can also "unlock" and "lock" access to such application software at will, adding the software to the master data image, or removing the software from it. In this way, the ASP can easily rent and sell software applications.

The physical media used to remotely store the master data images may be stored at highly available, reliable servers where disk mirroring techniques such as RAID can be used to make the data fault tolerant.

If a computer becomes non-functional (including the case when it cannot even boot) because of corruption of its operating system files, a correct image can be reloaded from a master data image, relying on the direct network connection between the LDIM and the RDIM.

When a user has difficulties or does not understand the behavior of the operating system or applications, technical support can evaluate the situation remotely and help the user (and if necessary repair the configuration), as it has access to the master data image.

Because the physical data image is stored remotely, operating systems, applications and other data may be updated on the remote persistent storage device(s) by Information Technology (IT) professionals (through high-level interfaces such as NTFS).

US 6,598,131 B2

15

Also, because the physical data image is stored remotely, remote servers hosting the remote persistent storage devices may automatically update software, drivers, etc., by interacting with the appropriate software vendors.

A computer including a LDIM can be used by multiple users at different times, with each user's master data image being transparently provided to him. Also, a user can choose which master data image to use.

As the LDIM can include encryption/decryption capability, the images can be stored encrypted on the remote persistent storage device and on the local persistent device, if any. That is, before the LDIM performs a write, the LDIM encrypts the data to be written and then writes the encrypted data. Similarly, when the LDIM process a read request, it first reads the data (either from the cached data image or from the master data image) and then decrypts the data before passing the data to the CPU of the computer.

It will thus be seen that the objects set forth above, among those made apparent from the preceding description, are efficiently attained and, since certain changes may be made in carrying out the above methods and in the devices as set forth without departing from the spirit and scope of the invention, it is intended that all matter contained in the above description and shown in the accompanying drawings shall be interpreted as illustrative and not in a limiting sense. Thus, the breadth and scope of the present invention should be defined only in accordance with the following claims and their equivalents.

What is claimed is:

1. A computer, comprising:

a processing unit;

a local persistent storage device (LPSD) comprising a non-volatile data storage medium; and

a local data image manager (LDIM) that emulates the local persistent storage device, wherein the LDIM is coupled between the processing unit and the LPSD,

the LDIM comprises a network interface that enables the LDIM to send messages to and receive messages from a remote data image manager (RDIM) that is located remotely from the computer and that can access a remote persistent storage device (RPSD), the processing unit is operable to request that data be stored in a non-volatile storage medium, the LDIM is configured to receive the data from the processing unit, and

the LDIM is operable to (a) issue a write request and transmit the data to the LPSD so that the LPSD will store the data on the non-volatile storage medium and (b) transmit the data to the RDIM using the network interface after receiving the data from the processing unit.

2. The computer of claim 1, wherein the LDIM transmits the data to the RDIM substantially concurrently with writing the data to the LPSD.

3. The computer of claim 1, wherein the LDIM is further operable to store the data or a pointer to the data if the LDIM cannot successfully transmit the data to the RDIM, thereby enabling the LDIM to send the data to the RDIM at a later time when the LDIM can successfully transmit the data to the RDIM.

4. The computer of claim 1, wherein the LPSD is a hard disk drive, the computer further comprises a disk adapter, and the LDIM is coupled to the processing unit through the disk adapter.

5. The computer of claim 4, wherein the LDIM comprises a connector and the LPSD is coupled to the LDIM through the LDIP's connector.

6. The computer of claim 4, wherein the LDIM receives the data directly from the disk adapter and transmits the data

16

to the LPSD through the LDIM connector to which the LPSD is connected.

7. The computer of claim 1, wherein the LDIM is coupled to the processing unit through a standard persistent storage device interface.

8. The computer of claim 1, wherein the standard persistent storage device interface is one of the ATA interface, SCSI interface, ST-506/412 interface, ESDI interface, IDE interface, ATAPI interface, ATA-E and EIDE interface.

9. The computer of claim 1, wherein, when the processing unit issues a non-data request, the non-data request is received by the LDIM and, in response to receiving the non-data request, the LDIM returns a response to the processing unit.

10. The computer of claim 1, further comprising a motherboard, wherein the processing unit is installed on the motherboard and the LDIM is connected to the motherboard by a cable.

11. In a computer having an operating system that receives from an application a request to store data on a local persistent storage device (LPSD) and that is configured to send the request and the data to a device driver for the LPSD, with the device driver being operable to issue the request and the data to a disk adapter or to a basic input/output system (BIOS), a method comprising the steps of:

receiving the request and the data directly from the device driver; and

in response to receiving the request and the data from the device driver, performing the steps of:

issuing a write request to the LPSD; and

transmitting a write request message to a system located remotely from the computer, wherein the write request message includes the data and an indication that the data should be written to a data storage medium.

12. The method of claim 11, wherein the request issued from the device driver includes a first address and the message transmitted to the system located remotely from the computer includes a second address, wherein the second address is a function of the first address.

13. The method of claim 12, wherein the second address is the same as the first address.

14. The method of claim 11, wherein the steps of issuing the write request to the LPSD and transmitting the write request message are performed substantially concurrently.

15. The method of claim 11, further comprising the steps of:

receiving a non-data request directly from the device driver; and

in response to receiving the non-data request, sending a response directly back to the device driver.

16. In a computer having a device driver, wherein the device driver receives a request to store data on a local persistent storage device (LPSD), and wherein the device driver is configured to issue the request to a disk adapter, and the disk adapter is configured to send the request to a controller of the LPSD, a method comprising the steps of:

receiving a request directly from the disk adapter, wherein the request requests that data be written; and

in response to receiving the request, performing the steps of:

issuing a write request to the LPSD; and

transmitting a write request message to a system located remotely from the computer, wherein the write request message includes the data and an indication that the data should be written to a data storage medium.

17. The method of claim 16, wherein the request issued from the device driver includes a first address and the message transmitted to the system located remotely from the

US 6,598,131 B2

17

computer includes a second address, wherein the second address is a function of the first address.

18. The method of claim 17, wherein the second address is the same as the first address.

19. The method of claim 16, wherein the steps of issuing the write request to the LPSD and transmitting the write request message are performed substantially concurrently.

20. The method of claim 16, further comprising the steps of:

receiving a non-data request directly from the device driver; and
in response to receiving the non-data request, sending a response directly back to the disk adapter.

21. In an environment where a device comprises a processor and a local persistent storage device (LPSD), a method for managing the LPSD's data image, comprising the steps of:

copying at least a portion of the data image;
storing the copied portion of the data image on a remote persistent storage device (RPSD) located remotely from the device;

receiving a request to write data to the LPSD, wherein the request was transmitted by the processor; and
in response to receiving the request, performing the steps of:

issuing a write request to the LPSD; and
transmitting a write request message to a system located remotely from the computer, wherein the message includes the data and wherein the system is operable to write the data the RPSD.

22. The system of claim 21, wherein the LPSD is a hard disk drive that comprises a hard disk.

23. The method of claim 21, further comprising the steps of:

receiving a non-data request transmitted from the processor; and
in response to receiving the non-data request, sending a response back to the processor.

24. The method of claim 21, wherein the steps of issuing the write request to the LPSD and transmitting the write request message are performed substantially concurrently.

25. A local data image manager, comprising:
interface means for interfacing to a persistent storage device (PSD);

receiving means for receiving commands and data, wherein the commands comprise read commands, write commands, and non-data commands;

means for issuing a write command to the PSD after the receiving means receives data and a write command; and

means for transmitting a write request message to a remote system located remotely from the local data image manager after the receiving means receives data and a write command.

26. The local data image manager of claim 25, wherein said means for receiving said commands receives said commands from a disk adapter.

27. The local data image manager of claim 25, wherein said PSD is a hard disk drive that comprises a hard disk.

28. The local data image manager of claim 27, wherein the receiving means comprises interface means for interfacing with a disk adapter.

29. The local data image manager of claim 28, wherein the interface means comprises a 40 pin IDE flat cable connector.

30. A system, comprising:

a first circuit board; first processing means disposed on the first circuit board;

a first local persistent storage device (LPSD) connector operable to connect to an LPSD, the LPSD connector being disposed on the first circuit board;

18

a second circuit board;

second processing means disposed on the second circuit board, the second processing means being programmed to emulate an LPSD;

a connector, disposed on the second circuit board, being connected to the LPSD connector disposed on the first circuit board;

transmitting means disposed on the second circuit board for enabling the second processing means to send messages to and receive messages from a remote system that is located remotely from the system and that can access a remote persistent storage device (RPSD), and

a second LPSD connector, the second LPSD connector being disposed on said second circuit board.

31. The system of claim 30, further comprising an LPSD, wherein the LPSD is connected to the second LPSD connector.

32. The system of claim 31, wherein the LPSD connected to the second LPSD connector is a hard disk drive.

33. The system of claim 31, wherein the LPSD connected to the second LPSD connector is a drive for writing data to compact disc.

34. The system of claim 33, wherein the second processing means receives read, write, and non-data commands issued from the first processing unit.

35. The system of claim 34, wherein the second processing means issues a write command to the LPSD connected to the second LPSD connector in response to receiving a write command issued from the first processing unit and also transmits a write request message to the remote system in response to receiving the write command.

36. The system of claim 35, wherein the second processing means issues the write command to the LPSD and transmits the write request message substantially concurrently.

37. In a system having an local data image manager (LDIM) and a local persistent storage device (LPSD) connected to the LDIM, a method, comprising:

receiving, at the LDIM, data, a request to store the data, and a first value for identifying a segment of a data storage medium; and

in response to receiving the data, the request to store the data, and the value, performing the steps of:

sending to the LPSD from the LDIM, the data, a second value for identifying a segment of a data storage medium, and a request to store the data; and

transmitting, via a network, a message to a remote system, wherein the message includes the data and a request for the remote system to store the data.

38. The method of claim 37, wherein the first value is a sector value for identifying a sector.

39. The method of claim 37, wherein the LPSD is a hard disk drive.

40. The method of claim 37, wherein the sending and transmitting steps are performed substantially concurrently.

41. The method of claim 37, further wherein the second value is not equal to the first value.

42. The method of claim 41, further comprising the step of determining the second value based on the first value.

43. The method of claim 37, wherein the LPSD comprises one of a hard disk, a read/write CD, a ZIP disk, a JAZ disk, and a floppy disk.

44. The method of claim 37, wherein the data included in the message transmitted to the remote system is encrypted data.

* * * * *

EXHIBIT C

(12) **United States Patent**
Kirkpatrick et al.

(10) **Patent No.:** US 6,732,359 B1
(45) **Date of Patent:** May 4, 2004

(54) **APPLICATION PROCESS MONITOR**

(75) Inventors: **Mark Kirkpatrick**, Conyers, GA (US);
Darin J. Morrow, Acworth, GA (US)

(73) Assignee: **BellSouth Intellectual Property Corporation**, Wilmington, DE (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.

(21) Appl. No.: **09/468,446**

(22) Filed: **Dec. 21, 1999**

(51) **Int. Cl.**⁷ **G06F 9/00**

(52) **U.S. Cl.** **718/102; 718/104; 718/107**

(58) **Field of Search** 709/104, 107,
709/100, 101, 102, 103; 717/27, 127; 718/100,
101, 102, 103, 104, 105, 106, 107

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,715,389 A * 2/1998 Komori 714/47
5,748,468 A * 5/1998 Notenboom et al. 700/3
5,835,765 A * 11/1998 Matsumoto 709/102

* cited by examiner

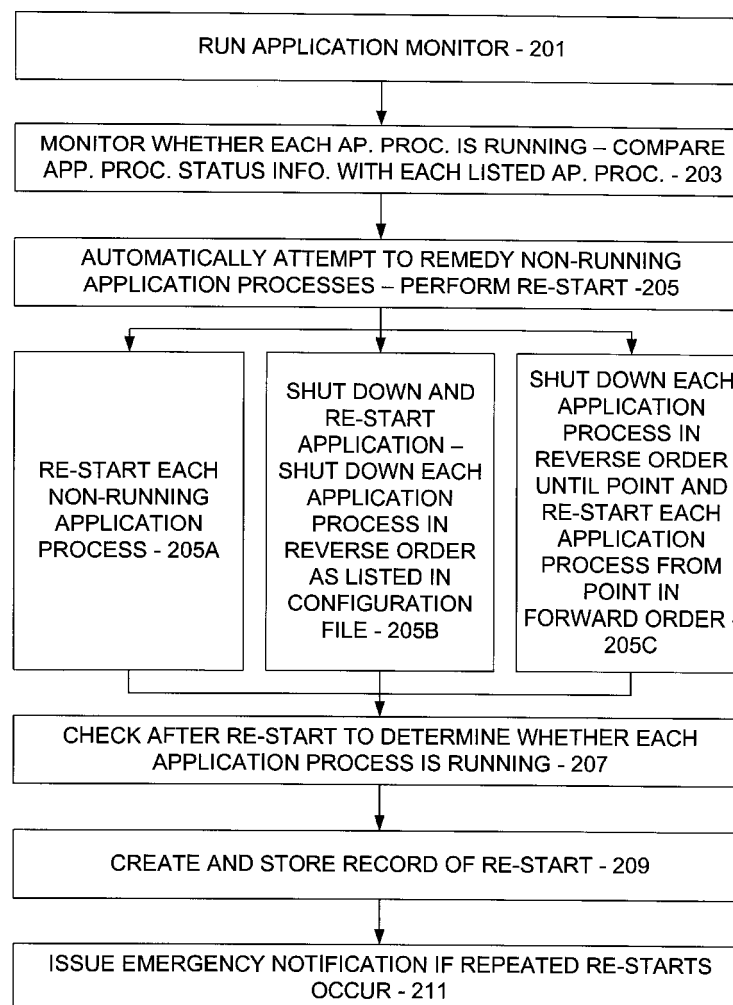
Primary Examiner—Majid A. Banankhah

(74) *Attorney, Agent, or Firm*—Woodcock Washburn LLP

(57) **ABSTRACT**

A computer system has a memory, an operating system, and a computer application instantiated in a work space in the memory as managed by the operating system. The application includes a plurality of application processes running in the work space. An application monitor monitors whether each of the plurality of application processes is in fact running, and automatically attempts to remedy an occurrence where any of the plurality of application processes is not in fact running.

59 Claims, 2 Drawing Sheets



U.S. Patent

May 4, 2004

Sheet 1 of 2

US 6,732,359 B1

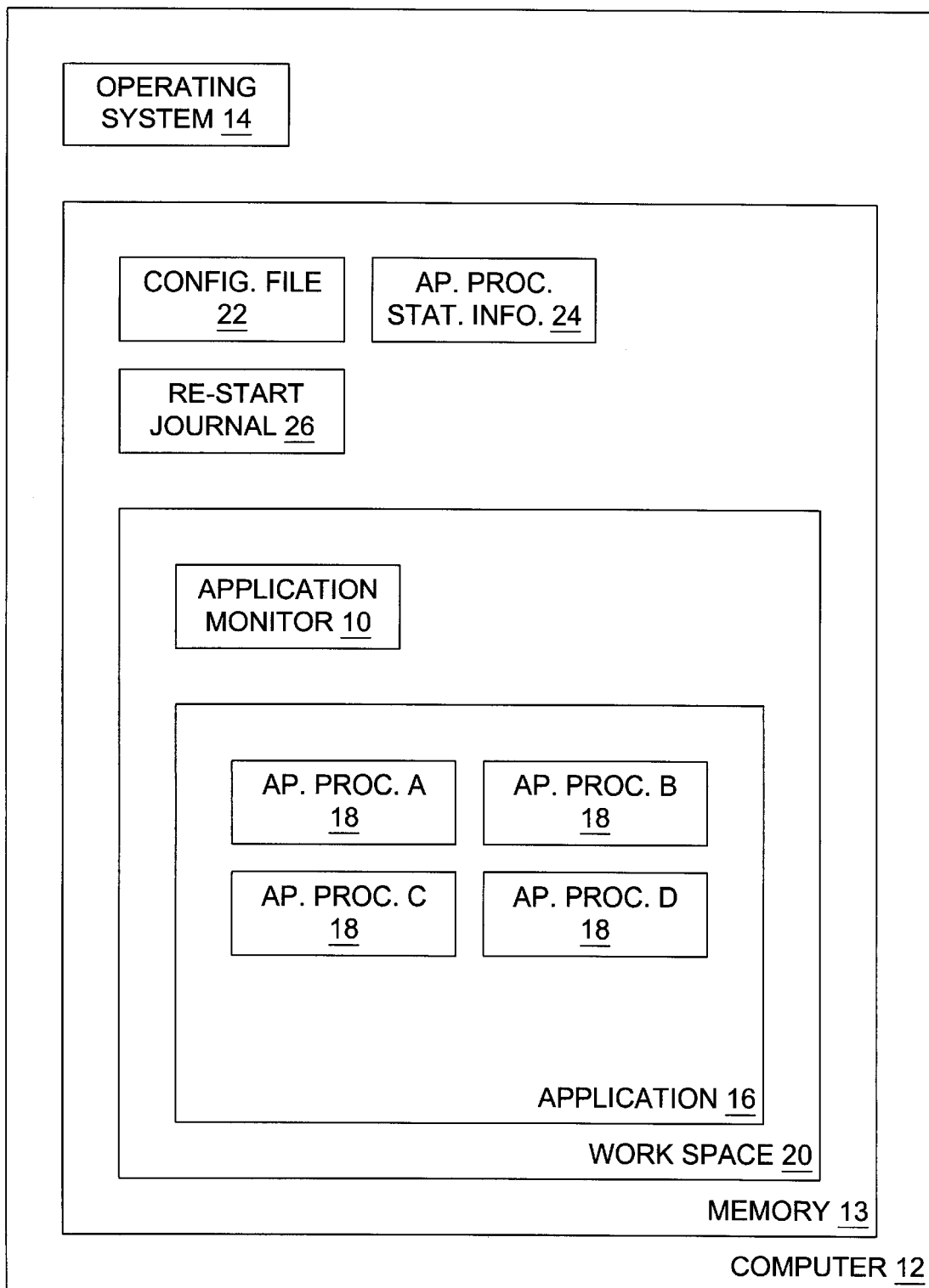


FIG. 1

U.S. Patent

May 4, 2004

Sheet 2 of 2

US 6,732,359 B1

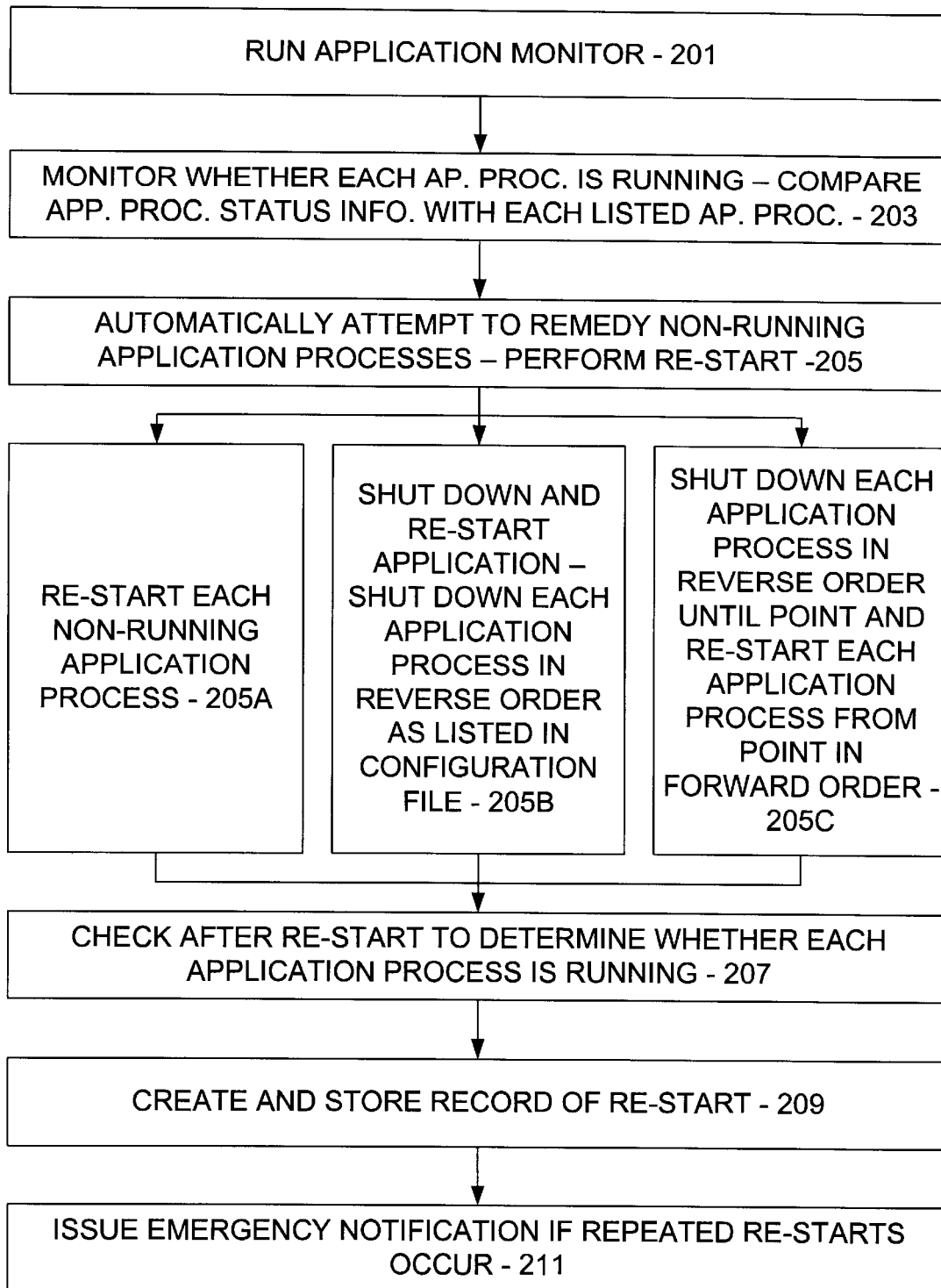


FIG. 2

US 6,732,359 B1

1

APPLICATION PROCESS MONITOR**FIELD OF THE INVENTION**

The present invention relates to a method and apparatus for monitoring an application process. In particular, the present invention relates to monitoring the constituent components of an application and responding to failures in the operation thereof.

BACKGROUND OF THE INVENTION

Typically, a computer system/computer has an operating system, and a computer application is instantiated on the computer in the form of one or more application processes running in a work space managed by the operating system. This is especially true with regards to large and/or complex applications, such as an application for managing one or more aspects of a factory, for controlling environmental conditions in a large building, for controlling power generation in a power facility, etc.

For any of a variety of reasons, an application process can fail. For example, an application process can fail if a needed resource is not available, if an expected piece of information is missing, or if an impermissible operation is performed, among other things. While such a failure does not necessarily cause the entire application to fail immediately, such failure of such entire application is likely to be inevitable.

As should be evident, failure of an entire application can be annoying to a system operator to say the least, and can cause critical and even fatal damage to say the most. In the most benign situation, a computer operator must somehow be informed that the application has failed and then must re-start the application on the computer. Of course, if the operator must be summoned at an inopportune hour, and/or if the operator must travel a relatively long distance to an appropriate location to command the re-start, even the most benign situation can become very costly and/or highly troublesome. In the most ominous situation, the failure of the application can lead to loss of life, if for example the application controls medical equipment in a hospital; loss of property, if for example the application controls environmental equipment in a sensitive location; and/or other dangerous situations.

Accordingly, a need exists for a method and apparatus for monitoring the application processes that comprise an application running on an operating system, and for automatically attempting to address the failure of an application process before such failure causes the failure of the entire application.

SUMMARY OF THE INVENTION

In the present invention, a computer system has a memory, an operating system, and a computer application instantiated in a work space in the memory as managed by the operating system. The application includes a plurality of application processes running in the work space. An application monitor monitors whether each of the plurality of application processes is in fact running, and automatically attempts to remedy an occurrence where any of the plurality of application processes is not in fact running.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing summary, as well as the following detailed description of preferred embodiments of the present invention, will be better understood when read in conjunc-

2

tion with the appended drawings. For the purpose of illustrating the invention, there are shown in the drawings embodiments which are presently preferred. As should be understood, however, the invention is not limited to the precise arrangements and instrumentalities shown. In the drawings:

FIG. 1 is a block diagram showing an application monitor operating in conjunction with a plurality of application processes constituting an application on a computer in accordance with one embodiment of the present invention; and

FIG. 2 is a flow chart depicting steps employed by the application monitor of FIG. 1 in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

Referring now to FIG. 1, an application monitor 10 is shown in accordance with one embodiment of the present invention. As seen, the application monitor 10 runs on a computer system/computer 12 or the like, and therefore may be embodied in the form of appropriate computer software. Of course, the application monitor 10 may also be embodied in the form of appropriate hardware or a combination of appropriate hardware and software without departing from the spirit and scope of the present invention.

As was discussed above, the computer 12 typically has a memory 13 and an operating system 14, and a computer application 16 is instantiated on the computer 12 in the form of one or more application processes 18 running in a work space 20 in the memory 13 as managed by the operating system 14 and set aside specifically for the application 16. Notably, any appropriate computer 12, operating system 14, and application 16 may be employed without departing from the spirit and scope of the present invention. For example, the computer 12 may be a mainframe computer, a mini-computer, a desktop- or laptop-based personal computer, or the like; the operating system may be a LINUX-based operating system, a WINDOWS-based operating system, a UNIX-based operating system, or some other operating system; and the application may be a banking system an environmental control system, a physical plant control system, a factory operation system, a medical facility operation system, or another application.

As shown, the operating system 14 may be operating separately from the memory 13, or may be operated in the memory 13. Such memory 13 may be any type of computer memory, such as RAM, ROM, a hard disk drive, a removable disk drive, a CD-ROM drive, or combinations thereof, without departing from the spirit and scope of the present invention.

Typically, when an operator commands the operating system 14 of a computer 12 to instantiate an application 16, such operating system 14 performs a number of procedures. In one of such procedures, the operating system 14 locates a configuration file 22 for the application 16 in the memory 13, and then processes the located configuration file 22. In particular, the configuration file includes a list of the application processes 18 that are to be run as part of instantiating the application 16. For example, the configuration file 22 for a particular application 16 may include the following commands:

```

. . .
RUN APPLICATION PROCESS A
RUN APPLICATION PROCESS B

```

US 6,732,359 B1

3

RUN APPLICATION PROCESS C
 RUN APPLICATION PROCESS D

As should be apparent from FIG. 1, the processing of the aforementioned configuration file 22 creates such application processes A–D in the work space 20 set aside for the application 16 by the operating system 14 of the computer 12. Of course, any particular configuration file 22 may be employed without departing from the spirit and scope of the present invention, and such configuration file 22 may contain other commands, again without departing from the spirit and scope of the present invention.

Depending on the application 16, the constituent application processes 18 thereof may work independently of each other. That is, the operation of each application process 18 does not affect the operation of any other application process 18. More likely, though, at least some of the constituent application processes 18 of the application 16 are dependent on other application processes 18. That is, for at least some of the application processes 18, the operation of each such application process 18 relies at least in part on the operation of at least one other application process 18. Accordingly, if a first application process 18 fails and therefore does not perform a particular function, a second application process that is depending on the first application process 18 to perform the particular function will likely in turn fail. Since a third application process 18 may be dependent on the second application process 18, a fourth application process 18 may be dependent on the third application process 18, etc., such third, fourth, and further application processes will likely eventually fail, too, until the underlying application 16 collapses and grinds to a halt. As may be appreciated, such a cascade of failures can occur over a relatively short period (a millisecond, for example) or a relatively long period (days, for example).

In one embodiment of the present invention, in addition to running each application process 18 as specified in the configuration file 22 to create such application process 18 in the work space 20 of the application 16, the operating system 14 of the computer 12 also runs the aforementioned application monitor 10 to create such application monitor 10 (step 201, FIG. 2). As should be understood, the application monitor 10 is for monitoring the application 16, and in particular is for monitoring whether each constituent application process 18 is running (step 203, FIG. 2).

As shown, the application monitor 10 for the application 16 may be created by the operating system 14 in the work space 20 of the application 16. However, such application monitor 10 may also be created elsewhere without departing from the spirit and scope of the present invention. Further, the running of the application monitor 10 may take place in the course of processing the configuration file 22, or may take place separately. If separately, an application script or the like associated with the application 16 may include commands such as:

...
 RUN CONFIGURATION FILE
 RUN APPLICATION MONITOR

Such application script thus causes the operating system 14 to locate and process the configuration file 22 and also to locate and run the application monitor 10.

In one embodiment of the present application, the application monitor 10 periodically checks with the operating system 14 to determine whether each application process 18 is still running. The actual frequency of the checks may of course vary without departing from the spirit and scope of

4

the present invention, and may be a function of such factors as the degree of reliability of the application 16, the criticality of the application 16, system resources available, the mean time for a failure of an application process 18 to cause a failure of the entire application 16, and the like.

In operation, the application monitor 10 is aware of each application process 18 listed in the configuration file 22 for the application 16, and refers to application process status information 24 maintained by the operating system 14 in the memory 13 (and perhaps even in the work space 20) to ascertain whether each such application process 18 is still running. The application monitor 10 may be made aware of each application process 18 listed in the configuration file 22 in any of several ways without departing from the spirit and scope of the present application. For example, the application monitor 10 may be specifically programmed with the name of each such application process 18, or may be provided with the capability to read the name of each application process from the configuration file 22, among other things.

The application process status information 24 maintained by the operating system 14 includes the name of each application process 18 currently running or the equivalent, and perhaps further information as to the status of each such application process 18 (i.e., ‘running’, ‘standby’, etc.). Such information 24 is likely organized in a table and stored by the operating system 14 in an appropriate location such as the memory 13, as shown. If an application process 18 has failed, it may be listed in the information 24 as such, or as some variation thereof (‘failed’, ‘not responding’, e.g.). Alternatively, such a failed application process 18 may not be listed in the information 24 at all. Thus, to ascertain whether each application process 18 of the application 16 is still running, the application monitor 10 in essence checks that each application process 18 as listed in the configuration file 22 for the application 16 is also listed in the application process status information 24 as running or the equivalent (i.e., ‘running’, ‘standby’, etc., and not ‘failed’, ‘not responding’, etc.) (step 203, FIG. 2).

If each such application process 18 of the application 16 is in fact still running or the equivalent, the application monitor 10 need take no action except to remind itself to perform another check after the specified period has passed. However, if one or more of the application processes 18 of the application 16 have stopped, either due to failure or otherwise, the application monitor 10 attempts to address and remedy the situation, as will be explained in more detail as follows (step 205, FIG. 2).

In one embodiment of the present invention, addressing and remedying the situation involves the application monitor 10 executing a command in cooperation with the operating system 14 to re-start each failed application process 18 (step 205A, FIG. 2). Notably, though, application processes 18 often must be started in a particular sequence, such as that specified in the configuration file 22, owing to their dependent nature. Accordingly, re-starting a particular application process 18 out of sequence may not be successful, and in fact could cause other application processes to fail. Accordingly, such re-starting of a particular application process 18 is preferably immediately followed by the application monitor 10 re-checking with the operating system 14 to determine whether each application process 18 is still running (step 207, FIG. 2).

In one embodiment of the present invention, if the aforementioned re-start of particular application processes 18 is unsuccessful, or as an alternative thereto, addressing and remedying the situation involves the application monitor 10

US 6,732,359 B1

5

causing the application 16 to shut down by causing each application process 18 thereof to shut down, and then causing the application to re-start in the normal manner, i.e., according to a command to the operating system 14 of the computer 12 to again instantiate the application 16 (step 205B, FIG. 2). Accordingly, such operating system 14 would then perform normal start-up procedures for the application, including locating and processing the configuration file 22 for the application 16, and if necessary or desirable again running the application monitor 10.

Such shut-down may be as simple as immediately killing all of the application processes 18 of the application 16 in no particular order or sequence and without any attempt to save any data. However, such a 'kill' shut-down may be quite radical and needlessly harsh in most circumstances. In one embodiment of the present invention, the shut-down is controlled as much as possible and is done in an orderly manner in an attempt to save as much data as possible. Accordingly, such a 'controlled' shut-down is in a manner similar to if not identical with a normal shut-down of the application 16, excepting of course the fact that one or more constituent application processes 18 are already de facto shut down. Such controlled shut-down may be directed by the application 16 itself or by the application monitor 10 if the application 16 is unable or willing to shut itself down. Accordingly, in such a situation, the application monitor 10 includes appropriate controlled shut-down procedures and is capable of executing such procedures in cooperation with the operating system 14.

Such shut-down procedures may comprise shutting down each application process in the reverse order/sequence as listed in the configuration file 22 (i.e., Application Process D, Application Process C, Application Process B, etc.). In such a situation, it is preferable that the application monitor 10 not be run in the course of processing the configuration file 22. Otherwise, the application monitor 10 could shut itself down prematurely. Once the application 16 is fully shut down, the application monitor 10, which should still be running, then can execute an appropriate start-up command in cooperation with the operating system 14. The application monitor 10 may then shut itself down in anticipation of being re-started by the operating system 14 in the course of re-starting the application 16, or may leave itself running.

In one embodiment of the present invention, if the aforementioned re-start of particular application processes 18 is unsuccessful, or as an alternative thereto, addressing and remedying the situation involves the application monitor 10 executing a command in cooperation with the operating system 14 to partially shut down the application 16 to the point of the failed application process 18, and then re-starting from such point (step 205C, FIG. 2). That is, remembering that the application processes 18 were started in a particular order/sequence as specified in the configuration file 22 (i.e., Application Process A, Application Process B, Application Process C, etc.), such application processes 18 are shut down in the reverse order/sequence (i.e., Application Process D, Application Process C, Application Process B, etc.) until the point where all of the failed application processes 18 would have been shut down. Thereafter, the application monitor 10 causes the application 16 to re-start from that point according to the particular order/sequence as specified in the configuration file 22.

Owing to the fact that such a 'partial' shut-down and re-start may not be successful for any of a variety of reasons, such partial re-start of the application 16 is preferably immediately followed by the application monitor 10 re-checking with the operating system 14 to determine

6

whether each application process 18 is still running. If in fact the partial re-start was unsuccessful, a controlled shut-down and re-start should be performed (step 207, FIG. 2).

In one embodiment of the present invention, a record of each re-start/shut-down, including all appropriate information, is created and stored in a re-start/shut-down journal 26 located in the memory 13 of the computer 12 or elsewhere (step 209, FIG. 2). Thus, an operator and/or programmer may review the journal 26 to diagnose the cause of any repeated application shut-downs. If a repeated shut-downs occur, or if re-starts are repeatedly unsuccessful, the application monitor 10 may cause an emergency notification or the like to be issued to appropriate personnel by way of an electronic mail message, an electronic telephone message, a telephone call to a beeper number, a radio message, a warning buzzer, etc. (step 211, FIG. 2).

The programming necessary to effectuate the present invention, such as the programming run by the application monitor 10, the operating system 14, and the application 16 and application processes 18 thereof, is known or is readily apparent to the relevant public. Accordingly, further details as to the specifics of such programming are not believed to be necessary herein.

As should now be understood, in the present invention, a method and apparatus are provided to monitor the application processes 18 that comprise an application 16 running on an operating system 14 of a computer 12, and for automatically attempting to address the failure of an application process 18 before such failure causes the failure of the entire application 16. Changes could be made to the embodiments described above without departing from the broad inventive concepts thereof. It is understood, therefore, that this invention is not limited to the particular embodiments disclosed, but it is intended to cover modifications within the spirit and scope of the present invention as defined by the appended claims.

What is claimed is:

1. A computer system having a memory, an operating system, a computer application instantiated in a work space in the memory as managed by the operating system, the application including a plurality of application processes running in the work space, and an application monitor monitoring whether each of the plurality of application processes is in fact running and automatically attempting to remedy an occurrence where any of the plurality of application processes is not in fact running.

2. The computer system of claim 1 wherein the application monitor is created by the operating system in the work space of the application.

3. The computer system of claim 1 wherein the operating system instantiates the application by processing a configuration file which includes a sequential list of the plurality of application processes to be run, and wherein the application monitor is run separately from the processing of the configuration file.

4. The computer system of claim 1 wherein the application monitor periodically checks with the operating system to determine whether each of the plurality of application processes is in fact running.

5. The computer system of claim 1 wherein if the application monitor finds that any of the plurality of application processes is not in fact running, such application monitor re-start each non-running application process.

6. The computer system of claim 5 wherein the re-start is followed by the application monitor checking to determine whether each application process is in fact running.

7. The computer system of claim 5 wherein the application monitor creates and stores a record of the re-start.

US 6,732,359 B1

7

8. The computer system of claim 5 wherein the application monitor issues an emergency notification if repeated re-starts occur.

9. The computer system of claim 8 wherein the application monitor issues the emergency notification by way of a member of a group consisting of an electronic mail message, an electronic telephone message, a telephone call to a beeper number, a radio message, and a warning buzzer.

10. The computer system of claim 1 wherein if the application monitor finds that any of the plurality of application processes is not in fact running, such application monitor shuts down and re-starts the application.

11. The computer system of claim 10 wherein the re-start is followed by the application monitor checking to determine whether each application process is in fact running.

12. The computer system of claim 10 wherein the operating system instantiates the application by processing a configuration file which includes an ordered list of the plurality of application processes to be run, and wherein the shut down comprises shutting down each of the plurality of application processes in a reverse order as listed in the configuration file.

13. The computer system of claim 10 wherein the application monitor creates and stores a record of the re-start.

14. The computer system of claim 10 wherein the application monitor issues an emergency notification if repeated re-starts occur.

15. The computer system of claim 14 wherein the application monitor issues the emergency notification by way of a member of a group consisting of an electronic mail message, an electronic telephone message, a telephone call to a beeper number, a radio message, and a warning buzzer.

16. The computer system of claim 1 wherein the operating system instantiates the application by processing a configuration file which includes an ordered list of the plurality of application processes to be run, and wherein if the application monitor finds that any of the plurality of application processes is not in fact running, such application monitor shuts down each of the plurality of application processes in a reverse order as listed in the configuration file until a point where all of the non-running application processes would have been shut down, and then re-starts each of the plurality of application processes from the point in a forward order as listed in the configuration file.

17. The computer system of claim 16 wherein the re-start is followed by the application monitor checking to determine whether each application process is in fact running.

18. The computer system of claim 16 wherein the application monitor creates and stores a record of the re-start.

19. The computer system of claim 16 wherein the application monitor issues an emergency notification if repeated re-starts occur.

20. The computer system of claim 19 wherein the application monitor issues the emergency notification by way of a member of a group consisting of an electronic mail message, an electronic telephone message, a telephone call to a beeper number, a radio message, and a warning buzzer.

21. An application monitor employed in connection with a computer system having a memory, an operating system, and a computer application instantiated in a work space in the memory as managed by the operating system, the application including a plurality of application processes running in the work space, the application monitor monitoring whether each of the plurality of application processes is in fact running and automatically attempting to remedy an occurrence where any of the plurality of application processes is not in fact running.

8

22. The application monitor of claim 21 wherein the application monitor is created by the operating system in the work space of the application.

23. The application monitor of claim 21 wherein the operating system instantiates the application by processing a configuration file which includes a sequential list of the plurality of application processes to be run, and wherein the application monitor is run separately from the processing of the configuration file.

24. The application monitor of claim 21 wherein the application monitor periodically checks with the operating system to determine whether each of the plurality of application processes is in fact running.

25. The application monitor of claim 21 wherein if the application monitor finds that any of the plurality of application processes is not in fact running, such application monitor re-starts each non-running application process.

26. The application monitor of claim 25 wherein the re-start is followed by the application monitor checking to determine whether each application process is in fact running.

27. The application monitor of claim 25 wherein the application monitor creates and stores a record of the re-start.

28. The application monitor of claim 25 wherein the application monitor issues an emergency notification if repeated re-starts occur.

29. The application monitor of claim 28 wherein the application monitor issues the emergency notification by way of a member of a group consisting of an electronic mail message, an electronic telephone message, a telephone call to a beeper number, a radio message, and a warning buzzer.

30. The application monitor of claim 21 wherein if the application monitor finds that any of the plurality of application processes is not in fact running, such application monitor shuts down and re-starts the application.

31. The application monitor of claim 30 wherein the re-start is followed by the application monitor checking to determine whether each application process is in fact running.

32. The application monitor of claim 30 wherein the operating system instantiates the application by processing a configuration file which includes an ordered list of the plurality of application processes to be run, and wherein the shut down comprises shutting down each of the plurality of application processes in a reverse order as listed in the configuration file.

33. The application monitor of claim 30 wherein the application monitor creates and stores a record of the re-start.

34. The application monitor of claim 30 wherein the application monitor issues an emergency notification if repeated re-starts occur.

35. The application monitor of claim 34 wherein the application monitor issues the emergency notification by way of a member of a group consisting of an electronic mail message, an electronic telephone message, a telephone call to a beeper number, a radio message, and a warning buzzer.

36. The application monitor of claim 21 wherein the operating system instantiates the application by processing a configuration file which includes an ordered list of the plurality of application processes to be run, and wherein if the application monitor finds that any of the plurality of application processes is not in fact running, such application monitor shuts down each of the plurality of application processes in a reverse order as listed in the configuration file until a point where all of the non-running application

US 6,732,359 B1

9

processes would have been shut down, and then re-starts each of the plurality of application processes from the point in a forward order as listed in the configuration file.

37. The application monitor of claim 36 wherein the re-start is followed by the application monitor checking to determine whether each application process is in fact running.

38. The application monitor of claim 36 wherein the application monitor creates and stores a record of the re-start.

39. The application monitor of claim 36 wherein the application monitor issues an emergency notification if repeated re-starts occur.

40. The application monitor of claim 39 wherein the application monitor issues the emergency notification by way of a member of a group consisting of an electronic mail message, an electronic telephone message, a telephone call to a beeper number, a radio message, and a warning buzzer.

41. In a computer system having a memory, an operating system, and a computer application instantiated in a work space in the memory as managed by the operating system, the application including a plurality of application processes running in the work space, a method comprising:

monitoring whether each of the plurality of application processes is in fact running; and

automatically attempting to remedy an occurrence where any of the plurality of application processes is not in fact running.

42. The method of claim 41 comprising creating an application monitor in the work space of the application to perform the monitoring and attempting steps.

43. The method of claim 41 comprising periodically checking with the operating system to determine whether each of the plurality of application processes is in fact running.

44. The method of claim 41 comprising, if any of the plurality of application processes is not in fact running, re-starting each non-running application process.

45. The method of claim 44 comprising checking after the re-start to determine whether each application process is in fact running.

46. The method of claim 44 comprising creating and storing a record of the re-start.

47. The method of claim 44 comprising issuing an emergency notification if repeated re-starts occur.

48. The method of claim 47 comprising issuing the emergency notification by way of a member of a group consisting of an electronic mail message, an electronic

10

telephone message, a telephone call to a beeper number, a radio message, and a warning buzzer.

49. The method of claim 41 comprising, if any of the plurality of application processes is not in fact running, shutting down and re-starting the application.

50. The method of claim 49 comprising checking after the re-start to determine whether each application process is in fact running.

51. The method of claim 49 wherein the operating system instantiates the application by processing a configuration file which includes an ordered list of the plurality of application processes to be run, the method comprising shutting down each of the plurality of application processes in a reverse order as listed in the configuration file.

52. The method of claim 49 comprising creating and storing a record of the re-start.

53. The method of claim 49 comprising issuing an emergency notification if repeated re-starts occur.

54. The method of claim 53 comprising issuing the emergency notification by way of a member of a group consisting of an electronic mail message, an electronic telephone message, a telephone call to a beeper number, a radio message, and a warning buzzer.

55. The method of claim 41 wherein the operating system instantiates the application by processing a configuration file which includes an ordered list of the plurality of application processes to be run, the method comprising, if any of the plurality of application processes is not in fact running, shutting down each of the plurality of application processes in a reverse order as listed in the configuration file until a point where all of the non-running application processes would have been shut down, and then re-starting each of the plurality of application processes from the point in a forward order as listed in the configuration file.

56. The method of claim 55 comprising checking after the re-start to determine whether each application process is in fact running.

57. The method of claim 55 comprising creating and storing a record of the re-start.

58. The method of claim 55 comprising issuing an emergency notification if repeated re-starts occur.

59. The method of claim 58 comprising issuing the emergency notification by way of a member of a group consisting of an electronic mail message, an electronic telephone message, a telephone call to a beeper number, a radio message, and a warning buzzer.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,732,359 B1
APPLICATION NO. : 09/468446
DATED : May 4, 2004
INVENTOR(S) : Mark Kirkpatrick and Darin J. Morrow

Page 1 of 2

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Col. 6, lines 37-45 (Claim 1) should read:

1. A computer system having a memory, an operating system, a computer application instantiated in a work space in the memory as managed by the operating system, the application including a plurality of application processes running in the work space, and an application monitor monitoring whether each of the plurality of application processes is in fact running and automatically attempting to remedy an occurrence where any of the plurality of application processes is not in fact running, wherein the application monitor is aware of each of the plurality of application processes and refers to application process status information as maintained by the operating system to ascertain whether each such application process is in fact running, wherein the operating system instantiates the application by processing a configuration file which includes a sequential list of the plurality of application processes to be run, and wherein the application monitor is aware of each application process listed in the configuration file, and wherein the application process status information maintained by the operating system includes each application process currently running.

Col. 7, lines 58-67 (Claim 21) should read as follows:

21. An application monitor employed in connection with a computer system having a memory, an operating system, and a computer application instantiated in a work space in the memory as managed by the operating system, the application including a plurality of application processes running in the work space, the application monitor monitoring whether each of the plurality of application processes is in fact running and automatically attempting to remedy an occurrence where any of the plurality of application processes is not in fact running, wherein the application monitor is aware of each of the plurality of application processes and refers to application process status information as maintained by the operating system to ascertain whether each such application process is in fact running, wherein the operating system instantiates the application by processing a configuration file which includes a sequential list of the plurality of application processes to be run, and wherein the application monitor is aware of each application process listed in the configuration file, and wherein the application process status information maintained by the operating system includes each application process currently running.

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,732,359 B1
APPLICATION NO. : 09/468446
DATED : May 4, 2004
INVENTOR(S) : Mark Kirkpatrick and Darin J. Morrow

Page 2 of 2

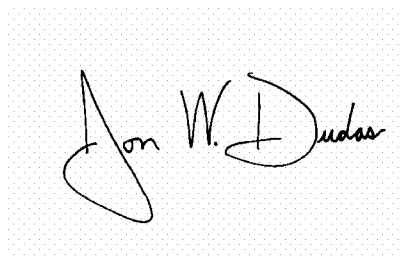
It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Col. 9, lines 19-28 (Claim 41) should read as follows:

41. In a computer system having a memory, an operating system, and a computer application instantiated in a work space in the memory as managed by the operating system, the application including a plurality of application processes running in the work space, a method comprising:
monitoring whether each of the plurality of application processes is in fact running; and
automatically attempting to remedy an occurrence where any of the plurality of application processes is not in fact running,
the method comprising referring to application process status information as maintained by the operating system to ascertain whether each such application process is in fact running,
wherein the operating system instantiates the application by processing a configuration file which includes a sequential list of the plurality of application processes to be run, the method comprising comparing the application process status information and each application process listed in the configuration file.

Signed and Sealed this

Seventeenth Day of October, 2006

A handwritten signature in black ink on a light gray dotted background. The signature is written in a cursive style and appears to read "Jon W. Dudas".

JON W. DUDAS

Director of the United States Patent and Trademark Office