UNITED STATES DISTRICT COURT

EASTERN DISTRICT OF TEXAS

TEXARKANA DIVISION

BROADCOM CORPORATION,
a California corporation

Plaintiff,

v.

INTEL CORPORATION,
a Delaware corporation,

Defendant.

Civil Action No. 501CV302

FIRST AMENDED COMPLAINT FOR DAMAGES AND INJUNCTIVE RELIEF

JURY TRIAL DEMANDED

FIRST AMENDED COMPLAINT FOR DAMAGES AND INJUNCTIVE RELIEF

JURY TRIAL DEMANDED

Plaintiff Broadcom Corporation, by its attorneys, complains against Defendant Intel Corporation, and alleges as follows:

Parties

1. Plaintiff Broadcom Corporation ("Broadcom") is a corporation organized under the laws of the State of California with its principal place of business at 16215 Alton Parkway, Irvine, California 92618. Broadcom markets and sells integrated circuit products within this District.

2. On information and belief, Defendant Intel Corporation ("Intel") is a corporation organized under the laws of the State of Delaware, with its principal place of business at 2200 Mission College Boulevard, Santa Clara, California 95052. On information and belief, Intel manufactures, markets and sells integrated circuit products, including but not limited to microprocessors and related chipsets, as well as computer components such as motherboards that incorporate Intel's microprocessors in combination with other integrated

circuits. Intel's products are marketed and sold within this District.

Jurisdiction and Venue

3. This is an action arising under the patent laws of the United States, 35 U.S.C. § 101 et seq. This Court has subject matter jurisdiction under 28 U.S.C. §§ 1331 and 1338(a).

4. Venue is proper in this judicial district under 28 U.S.C. §§ 1391(b) and (c) and 1400(b).

The Patents

5. United States Patent No. 6,189,064 B1, entitled "Graphics Display System with Unified Memory Architecture" (the "'064 Patent"), names Alexander G. MacInnis, Chengfuh Jeffrey Tang, Xiaodong Xie, James T. Patterson and Greg A. Kranawetter as inventors. The '064 patent was duly and legally issued by the United States Patent and Trademark Office on February 13, 2001. A copy of the '064 Patent is attached hereto as Exhibit A.

6. United States Patent No. 5,963,210, entitled "Graphics Processor, System and Method for Generating Screen Pixels in Raster Order Utilizing a Single Interpolator" (the "'210 Patent"), names Michael C. Lewis and Stephen L. Morein as inventors. The '210 patent was duly and legally issued by the United States Patent and Trademark Office on October 5, 1999. A copy of the '210 Patent is attached hereto as Exhibit B.

7. United States Patent No. 6,178,198 B1, entitled "Apparatus For, And Method Of, Processing Signals Transmitted Over A Local Area Network" (the "'198 Patent"), names Henry Samueli, Fang Lu, and Avanindra Madisetti as inventors. The '198 patent was duly and legally issued by the United States Patent and Trademark Office on January 23, 2001. A copy of the '198 Patent is attached hereto as Exhibit C.

8. Broadcom is the owner of all right, title and interest in and to the '064, '210, and '198 Patents (the "Broadcom Patents").

Background

9. Plaintiff Broadcom has been engaged in research and development in many areas of integrated circuit, communications and computer technologies and has received, or has been assigned, a number of patents relating to integrated circuit, communications and computer technologies, including the '064, '210, and '198 patents.

10. On information and belief, the microprocessors designed and sold by Defendant Intel, such as the well-known Pentium III, Pentium 4, and Celeron families of microprocessors, must be combined with other components in order to operate as part of a computer system or to communicate on a local area network.

11. On information and belief, Defendant Intel manufactures and sells integrated circuits other than microprocessors that are specifically designed to be combined and to operate with Defendant Intel's microprocessors as part of a computer system ("Intel Support Chips"). Intel Support Chips include, among others, the Intel 82810, 82815, and 82830 families of chipsets with integrated graphics.

12. On information and belief, Defendant Intel manufactures and sells integrated circuits other than microprocessors that are specifically designed to be used to create and implement a local area network in conjunction with Defendant Intel's microprocessors and other integrated circuits. These integrated circuits include, among others, the Intel 82558 Fast Ethernet PCI Bus Controller with Integrated PHY, and the Intel 82559 Fast Ethernet Multifunction PCI/Cardbus Controller ("Intel Networking Chips").

13. On information and belief, the Intel Support Chips and Intel Networking Chips include inventions that are claimed by the Broadcom Patents.

14. On information and belief, Defendant Intel also makes, uses, offers for sale and sells computer motherboards ("Intel Motherboards") that incorporate the Intel Support Chips and/or Intel Networking Chips and that are designed to be combined with one or more Intel microprocessors as part of a computer system. The Intel Motherboards presently encompass, among others, the Intel D810 and D815 families of motherboards.

15. On information and belief, anyone who purchases an Intel Motherboard must also acquire an Intel microprocessor in order to operate the Intel Motherboard as part of a computer system.

16. On information and belief, Intel manufactures and sells Network Interface Cards that include the Intel Networking Chips and sells the Intel Networking Chips to third parties for incorporation into the third parties' network interface cards.

17. On information and belief, the sale of Intel Motherboards, Intel Support Chips, Intel Networking Chips, and Intel Network Interface Cards supports and/or augments the sale of Intel Microprocessors, in part due to the incorporation of Broadcom's patented technology.

Count I

18. Broadcom repeats and realleges paragraphs 1-17.

19. On information and belief, the Defendant has infringed and is infringing the '064 Patent by making, using, offering for sale, or selling in the United States, and/or by inducing others to make, use, offer for sale, or sell in the United States, and/or by contributing to the infringement by others who make, use, offer for sale, or sell in the United States, products that incorporate the claimed invention of the '064 Patent, including, among others, the Intel 82810, 82815, and 82830 families of chipsets with integrated graphics and Intel Motherboards incorporating one or more of those components, separately and/or in combination with other components, including but not limited to, Intel Microprocessors.

20. On information and belief, the Defendant's infringement of the '064 Patent has

been willful.

21. On information and belief, the Defendant's infringement of the '064 Patent has caused, and will continue to cause, Broadcom irreparable harm unless enjoined by the Court. Broadcom has no adequate remedy at law.

Count II

22. Broadcom repeats and realleges paragraphs 1-17.

23. On information and belief, the Defendant has infringed and is infringing the '210 Patent by making, using, offering for sale, or selling in the United States, and/or by inducing others to make, use, offer for sale, or sell in the United States, and/or by contributing to the infringement by others who make, use, offer for sale, or sell in the United States, products that incorporate the claimed invention of the '210 Patent, including, among others, the Intel 82810, 82815, and 82830 families of chipsets with integrated graphics and Intel Motherboards incorporating one or more of those components, separately and/or in combination with other components, including but not limited to, Intel Microprocessors.

24. On information and belief, the Defendant's infringement of the '210 Patent has been willful.

25. On information and belief, the Defendant's infringement of the '210 Patent has caused, and will continue to cause, Broadcom irreparable harm unless enjoined by the Court. Broadcom has no adequate remedy at law.

Count III

26. Broadcom repeats and realleges paragraphs 1-17.

27. On information and belief, the Defendant has infringed and is infringing the '198 Patent by making, using, offering for sale, or selling in the United States, and/or by inducing others to make, use, offer for sale, or sell in the United States, and/or by contributing to the infringement by others who make, use, offer for sale, or sell in the United States, products that incorporate the claimed invention of the '198 Patent, including, among others, the Intel 82558 and 82559 integrated circuits and motherboards and Network Interface Cards made by Intel and third parties incorporating those components, separately and/or in combination with other components, including but not limited to, Intel Microprocessors.

28. On information and belief, the Defendant's infringement of the '198 Patent has been willful.

29. On information and belief, the Defendant's infringement of the '198 Patent has caused, and will continue to cause, Broadcom irreparable harm unless enjoined by the Court. Broadcom has no adequate remedy at law.

Prayer for Relief

WHEREFORE, Broadcom Corporation respectfully requests that this Court enter judgment in its favor and grant the following relief:

A. Adjudge that the Defendant has infringed and is infringing each of the Broadcom Patents;

B. Enter orders preliminarily and permanently enjoining the Defendant from any further acts of infringement of each of the Broadcom Patents;

C. Award Broadcom damages in an amount adequate to compensate Broadcom for the Defendant's infringement of the Broadcom Patents, including but not limited to the damages attributable to the sale of Intel microprocessors and related components in combination with the infringing devices, pursuant to 35 U.S.C. § 284;

D. Adjudge that the Defendant's infringement of each of the Broadcom Patents is willful;

E. Enter an order awarding Broadcom treble damages pursuant to 35 U.S.C. § 284;

F. Enter an order awarding Broadcom interest on the damages awarded and its costs pursuant to 35 U.S.C. § 284;

G. Enter an order finding that this is an exceptional case and awarding Broadcom its reasonable attorneys' fees pursuant to 35 U.S.C. § 285; and

H. Award such additional relief as the Court may deem appropriate and just under the circumstances.

Dated: December 14, 2001

Nicholas H. Patton w/ permission by Robert W. Schroeder III

Nicholas H. Patton
Patton and Tidwell LLP
4605 Texas Boulevard
P.O. Box 5398
Texarkana, TX 75505-5398
Telephone: (903) 792-7080
Facsimile: (903) 792-8233
Attorney for Plaintiff
BROADCOM CORPORATION

Dated: December 14, 2001

J. Thad Heartfield w/ permission by Robert W. Schroeder III

J. Thad Heartfield
HEARTFIELD & McGINNIS LLP
2196 Dowlen Road
Beaumont, TX 77706
Telephone: (409) 866-3318
Facsimile: (409) 886-5789

Of Counsel:

Terrence P. McMahon
Vera M. Elson
McDERMOTT, WILL & EMERY
2700 Sand Hill Road
Menlo Park, CA 94025
Telephone: (650) 233-5500
Facsimile: (650) 233-5599

Fay E. Morisseau
McDERMOTT, WILL & EMERY
18191 Von Karman Avenue, Suite 500
Irvine, CA 92612-0187
Telephone: (949) 851-0633
Facsimile: (949) 851-9348

Raphael V. Lupo
McDERMOTT, WILL & EMERY
600 13th St. N.W.
12th floor
Washington, D.C. 20005
Telephone: (202) 756-8000
Facsimile: (202) 756-8087

US006189064B1

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

(10) **Patent No.: US 6,189,064 B1**
(45) **Date of Patent: *Feb. 13, 2001**

(54) **GRAPHICS DISPLAY SYSTEM WITH UNIFIED MEMORY ARCHITECTURE**

(75) Inventors: **Alexander G. MacInnis**, Los Altos; **Chengfuh Jeffrey Tang**, Saratoga; **Xiaodong Xie**, San Jose; **James T. Patterson**, Saratoga; **Greg A. Kranawetter**, San Jose, all of CA (US)

(73) Assignee: **Broadcom Corporation**, Irvine, CA (US)

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

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

(21) Appl. No.: **09/437,209**

(22) Filed: **Nov. 9, 1999**

Related U.S. Application Data

(60) Provisional application No. 60/107,875, filed on Nov. 9, 1998.

(51) **Int. Cl.**[7] **G06F 13/18**; G06F 9/46

(52) **U.S. Cl.** **710/244**; 710/111; 709/100; 345/133

(58) **Field of Search** 710/244, 240, 710/41, 40, 113, 36, 116, 241, 111, 107; 711/151, 169, 100; 365/230.03; 709/100; 345/185, 133; 714/3; 348/552

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,155,816		10/1992	Kohn	395/375
5,398,211	*	3/1995	Willenz et al. .	
5,604,514		2/1997	Hancock	345/154
5,758,177		5/1998	Gulick et al.	395/800.01
5,864,345		1/1999	Wickstrom et al.	345/431
5,949,439		9/1999	Ben-Yoseph et al.	345/503
5,951,664	*	9/1999	Lambrecht et al. .	
5,956,041		9/1999	Koyamada et al.	345/420
5,987,555	*	11/1999	Alzien et al. .	
6,006,303	*	12/1999	Barnaby et al. .	

OTHER PUBLICATIONS

Motorola, Inc., MC92100 "Scorpion" Graphics Display Generator, SDRAM Controller, and Digital Video Encoder, 1997, 6 pages.

Power TV, Inc., Eagle™ Graphics/Audio Media Compositor Data Sheet, Version 1.7, Feb. 27, 1997, p. 63.

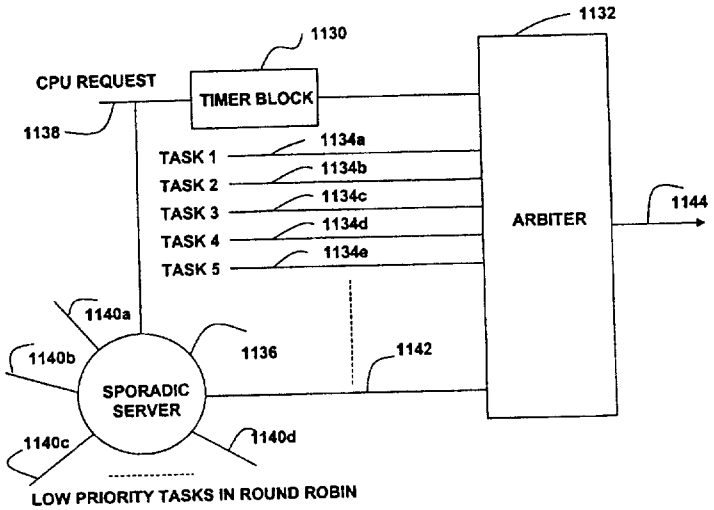
* cited by examiner

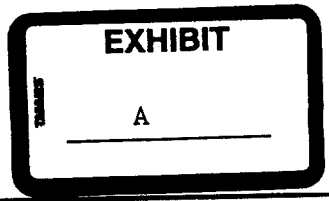
Primary Examiner—Gopal C. Ray
(74) *Attorney, Agent, or Firm*—Christie, Parker & Hale, LLP

(57) **ABSTRACT**

A graphics display system integrated circuit is used in a set-top box for controlling a television display. The graphics display system processes analog video input, digital video input, and graphics input. The system incorporates a unified memory architecture that is shared by the graphics system, a CPU, and other peripherals. The unified memory architecture uses real time scheduling to service tasks. Critical instant analysis is used to find a schedule for memory usage that does not affect memory requirements of real time tasks while at the same time servicing non-real-time tasks as needed.

22 Claims, 37 Drawing Sheets





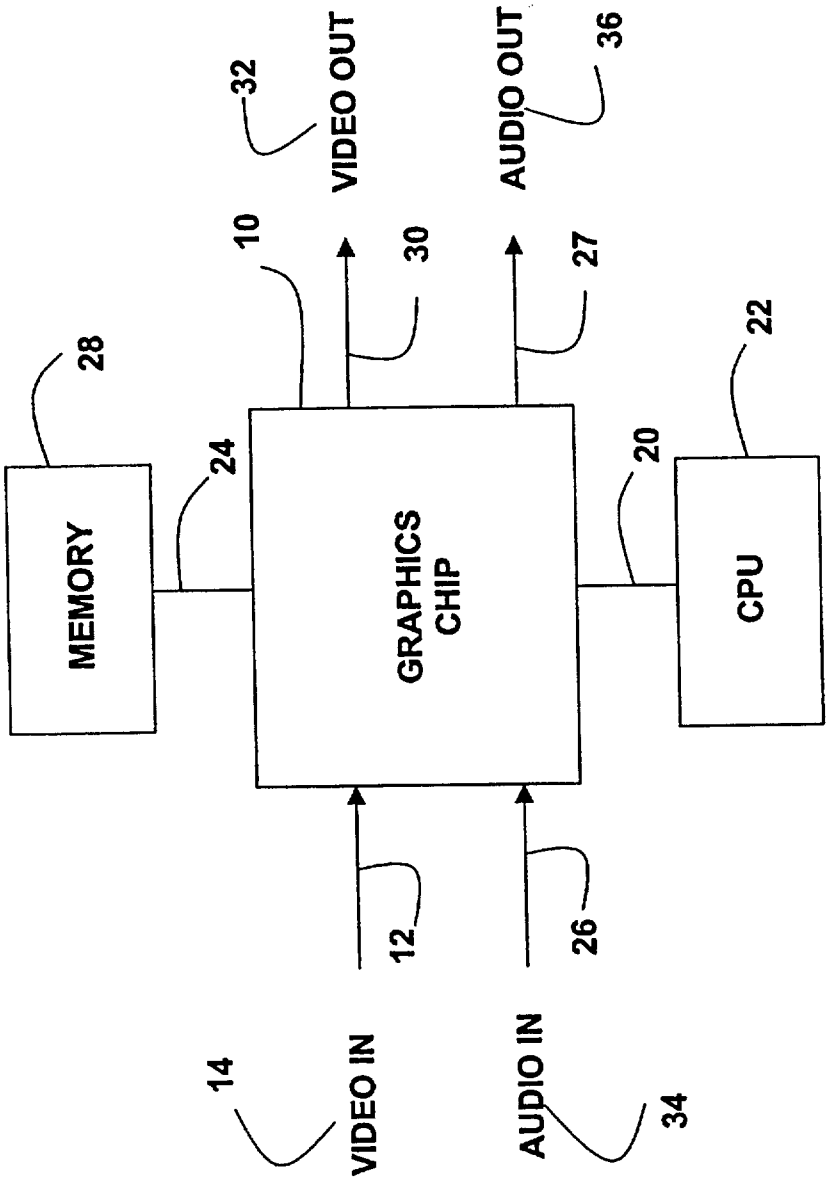

MEMORY
GRAPHICS CHIP
CPU
VIDEO OUT
AUDIO OUT
VIDEO IN
AUDIO IN
10
12
14
20
22
24
26
27
28
30
32
34
36

FIG. 1

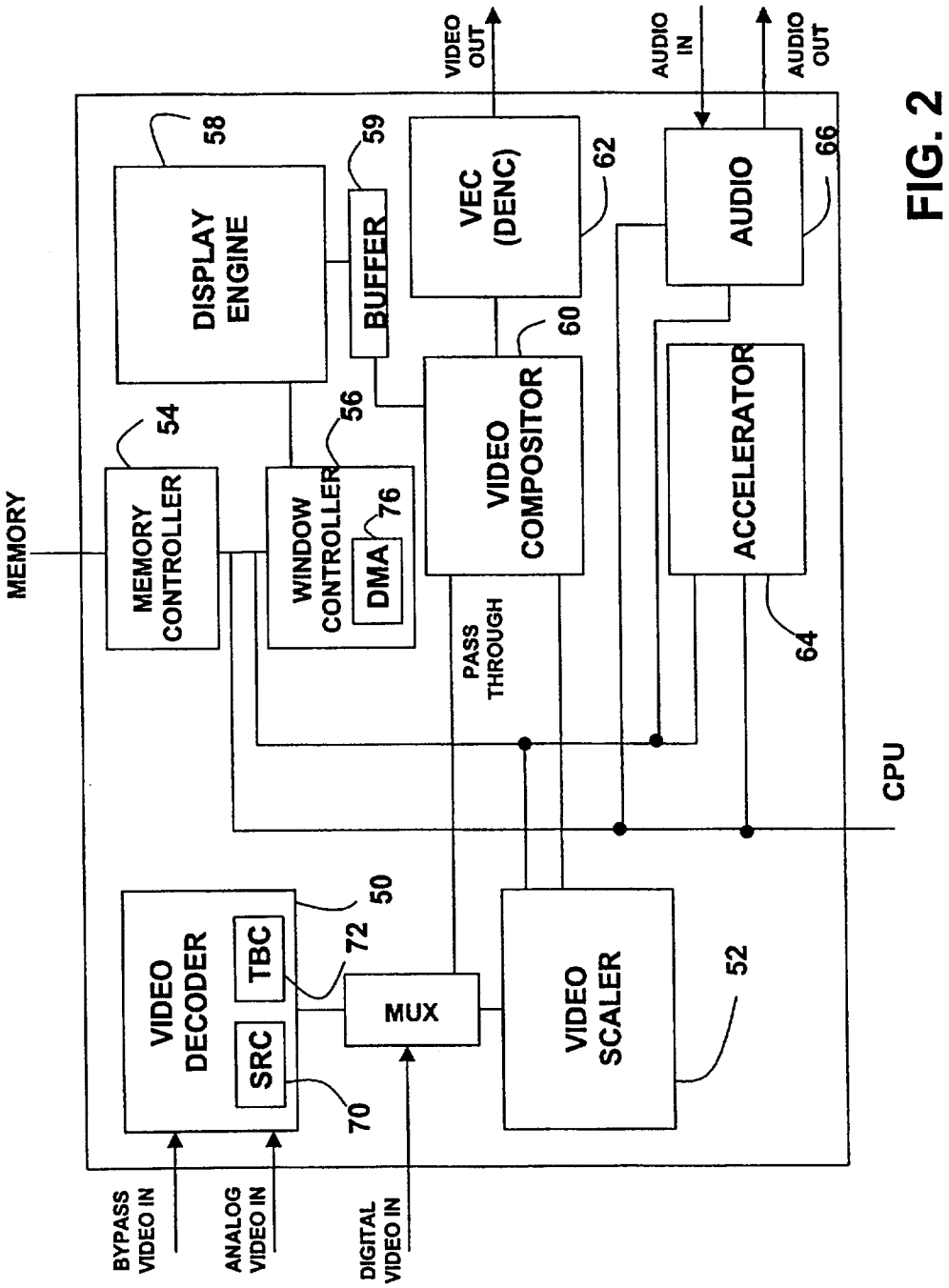

VIDEO OUT
AUDIO IN
AUDIO OUT
DISPLAY ENGINE
58
BUFFER
59
VEC (DENC)
62
AUDIO
66
VIDEO COMPOSITOR
60
ACCELERATOR
64
MEMORY
MEMORY CONTROLLER
54
WINDOW CONTROLLER
56
DMA
76
PASS THROUGH
CPU
VIDEO DECODER
50
TBC
72
SRC
70
MUX
VIDEO SCALER
52
BYPASS VIDEO IN
ANALOG VIDEO IN
DIGITAL VIDEO IN

FIG. 2

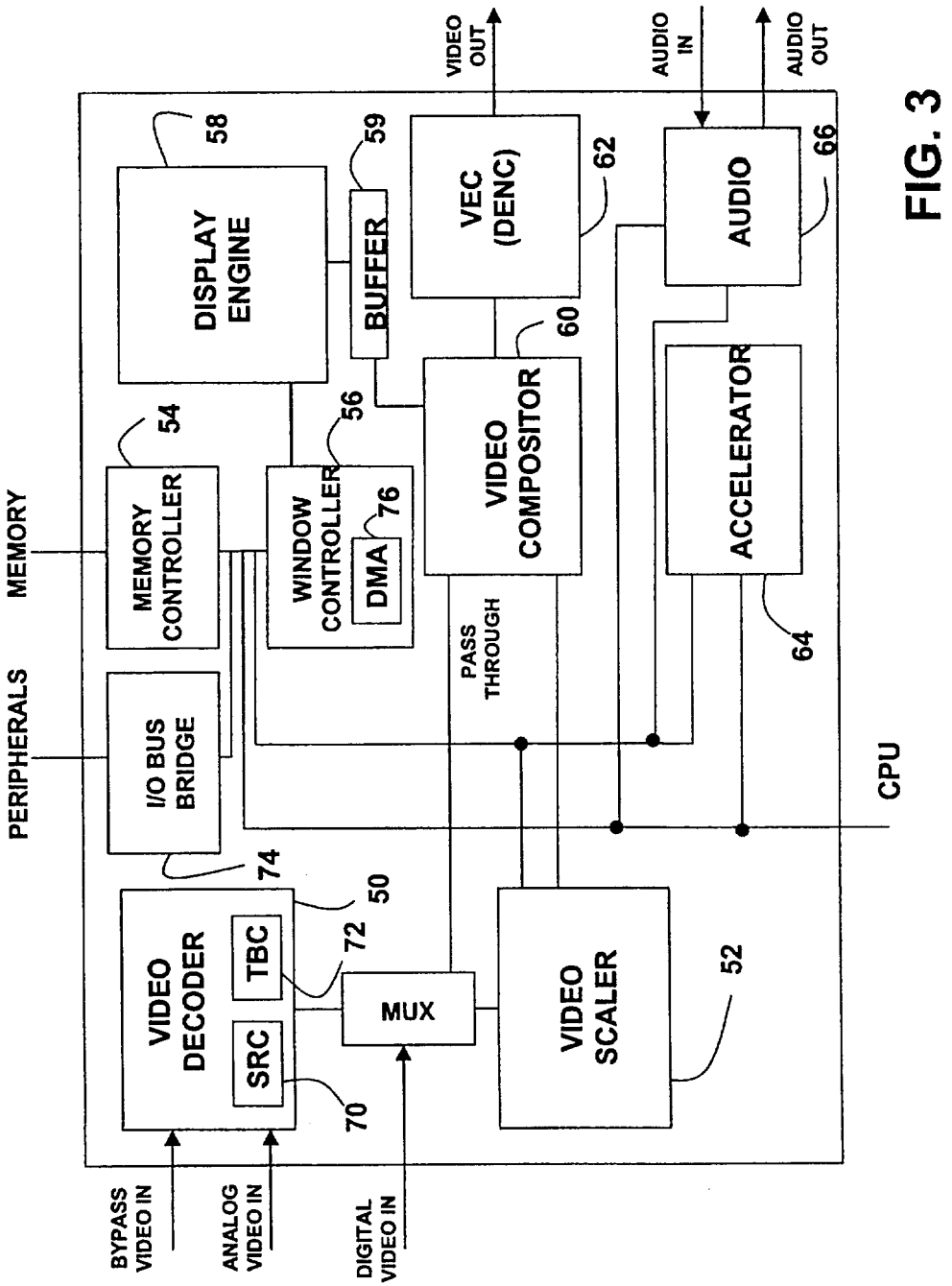

VIDEO OUT
AUDIO IN
AUDIO OUT
DISPLAY ENGINE
58
BUFFER
59
VEC (DENC)
62
AUDIO
66
VIDEO COMPOSITOR
60
ACCELERATOR
64
MEMORY CONTROLLER
54
WINDOW CONTROLLER
56
DMA
76
MEMORY
PERIPHERALS
I/O BUS BRIDGE
74
PASS THROUGH
CPU
VIDEO DECODER
50
TBC
72
SRC
70
MUX
VIDEO SCALER
52
BYPASS VIDEO IN
ANALOG VIDEO IN
DIGITAL VIDEO IN

FIG. 3

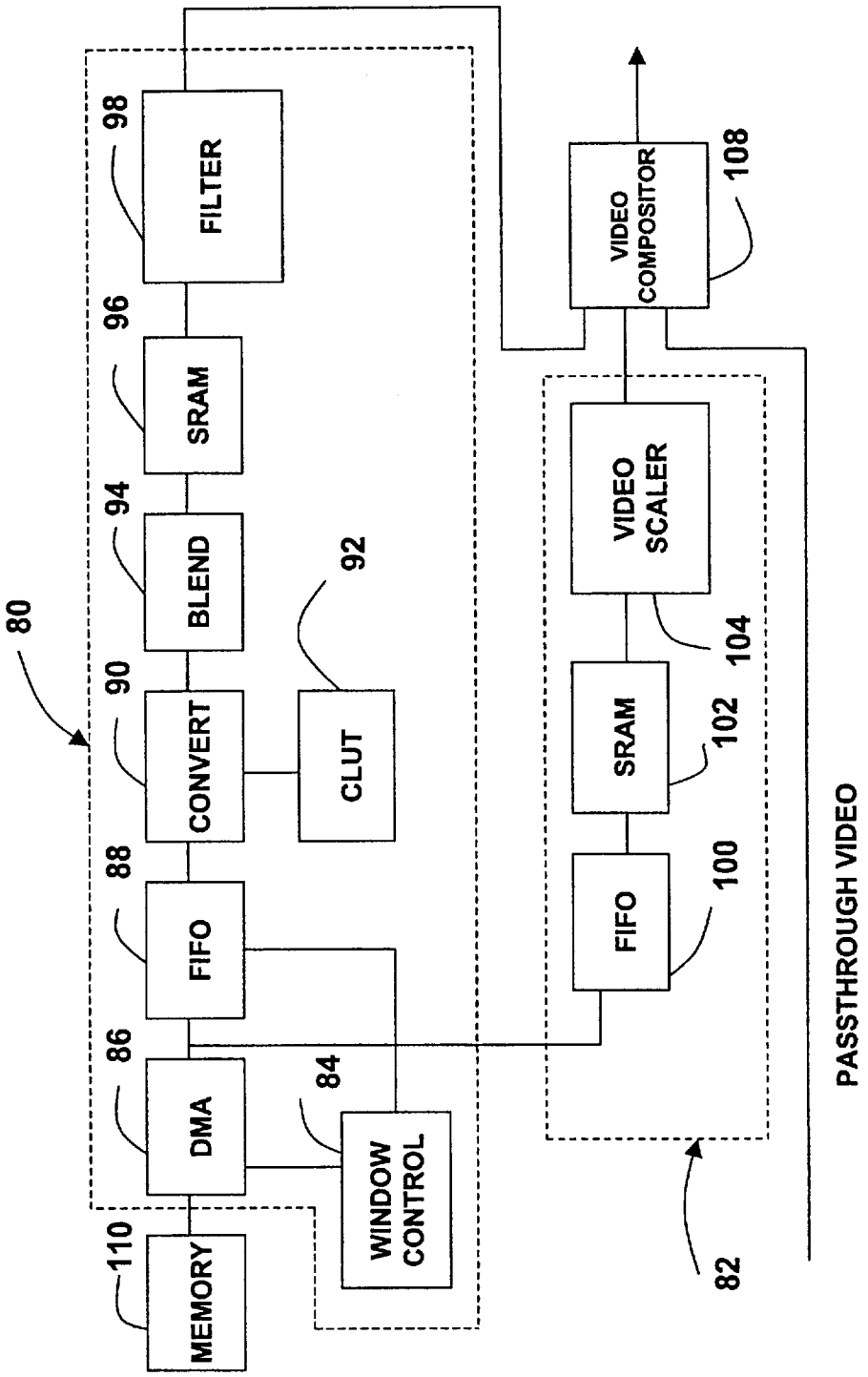

80
82
84
86
88
90
92
94
96
98
100
102
104
108
110
FILTER
SRAM
BLEND
CONVERT
CLUT
FIFO
DMA
WINDOW CONTROL
MEMORY
VIDEO COMPOSITOR
VIDEO SCALER
SRAM
FIFO
PASSTHROUGH VIDEO

FIG. 4

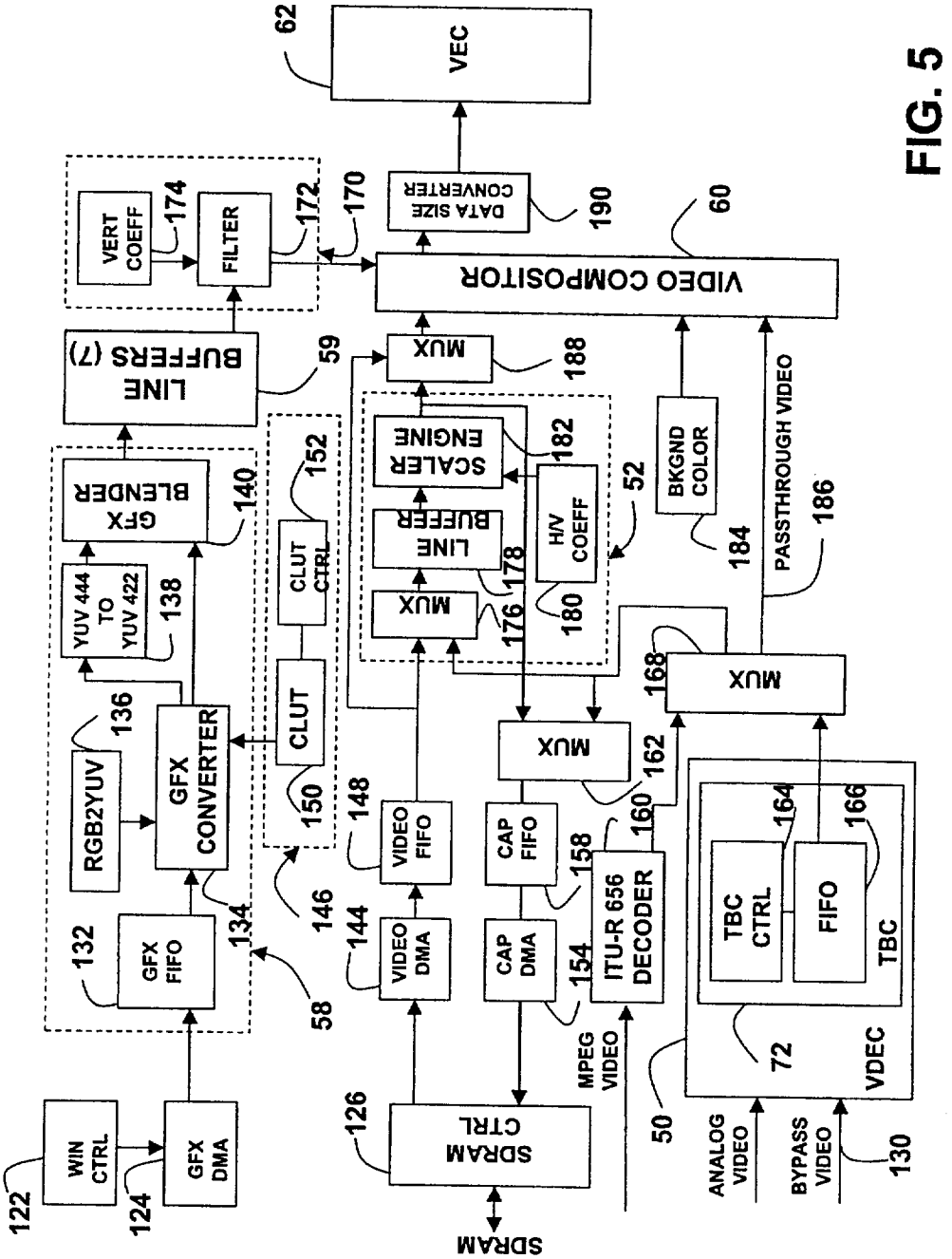

VEC
62
DATA SIZE CONVERTER
190
60
VIDEO COMPOSITOR
VERT COEFF
174
FILTER
172
170
LINE BUFFERS (7)
59
MUX
188
SCALER ENGINE
182
LINE BUFFER
178
MUX
176
H/V COEFF
180
52
BKGND COLOR
184
PASSTHROUGH VIDEO
186
GFX BLENDER
140
CLUT CTRL
152
CLUT
150
YUV 444 TO YUV 422
138
RGB2YUV
136
GFX CONVERTER
GFX FIFO
132
134
58
146
VIDEO FIFO
148
VIDEO DMA
144
MUX
168
MUX
162
CAP FIFO
158
CAP DMA
154
ITU-R 656 DECODER
160
TBC CTRL
164
FIFO
166
TBC
VDEC
72
50
MPEG VIDEO
WIN CTRL
122
GFX DMA
124
SDRAM CTRL
126
SDRAM
ANALOG VIDEO
BYPASS VIDEO
130

FIG. 5

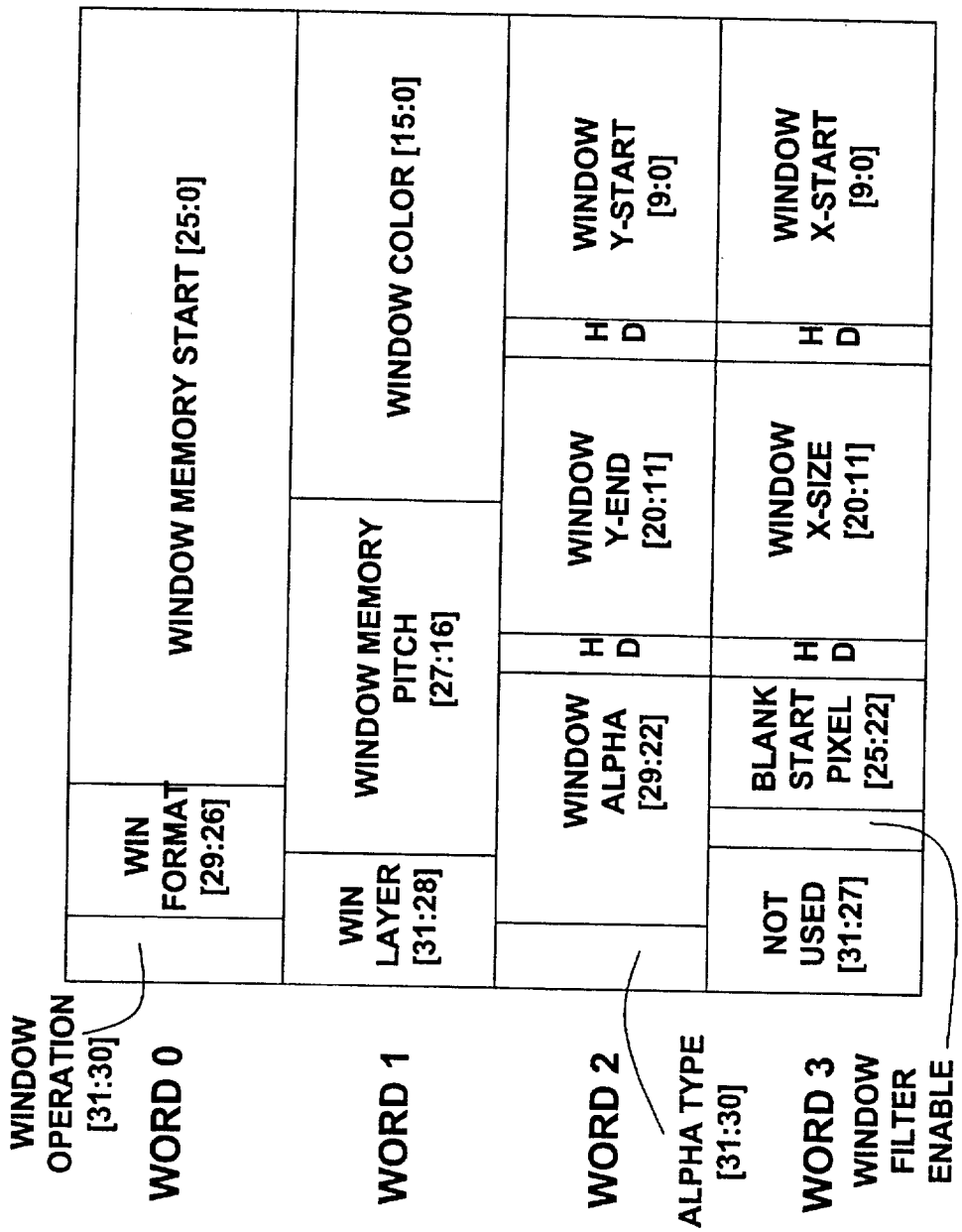

WINDOW OPERATION [31:30]
WORD 0
WIN FORMAT [29:26]
WINDOW MEMORY START [25:0]
WORD 1
WIN LAYER [31:28]
WINDOW MEMORY PITCH [27:16]
WINDOW COLOR [15:0]
WORD 2
ALPHA TYPE [31:30]
WINDOW ALPHA [29:22]
H
D
WINDOW Y-END [20:11]
H
D
WINDOW Y-START [9:0]
H
D
WORD 3
WINDOW FILTER ENABLE [26]
NOT USED [31:27]
BLANK START PIXEL [25:22]
H
D
WINDOW X-SIZE [20:11]
H
D
WINDOW X-START [9:0]

FIG. 6

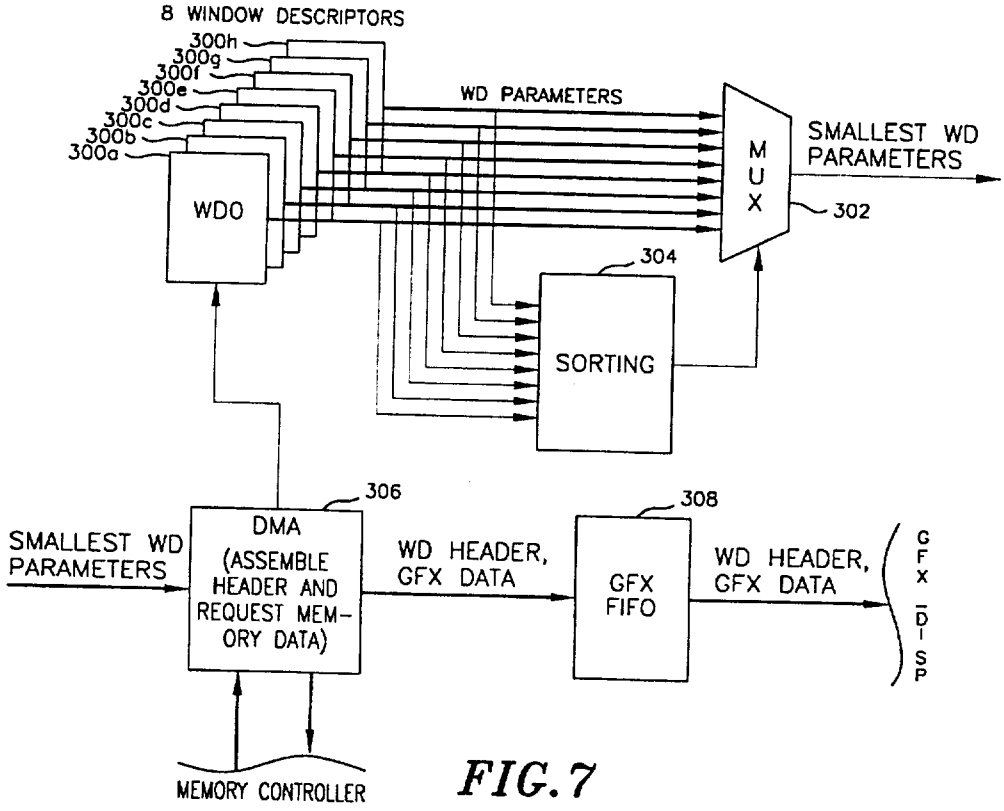

8 WINDOW DESCRIPTORS
300h
300g
300f
300e
300d
300c
300b
300a
WD PARAMETERS
WDO
M
U
X
302
SMALLEST WD PARAMETERS
304
SORTING
306
DMA (ASSEMBLE HEADER AND REQUEST MEMORY DATA)
SMALLEST WD PARAMETERS
WD HEADER, GFX DATA
308
GFX FIFO
WD HEADER, GFX DATA
GFX DISP
MEMORY CONTROLLER

FIG. 7

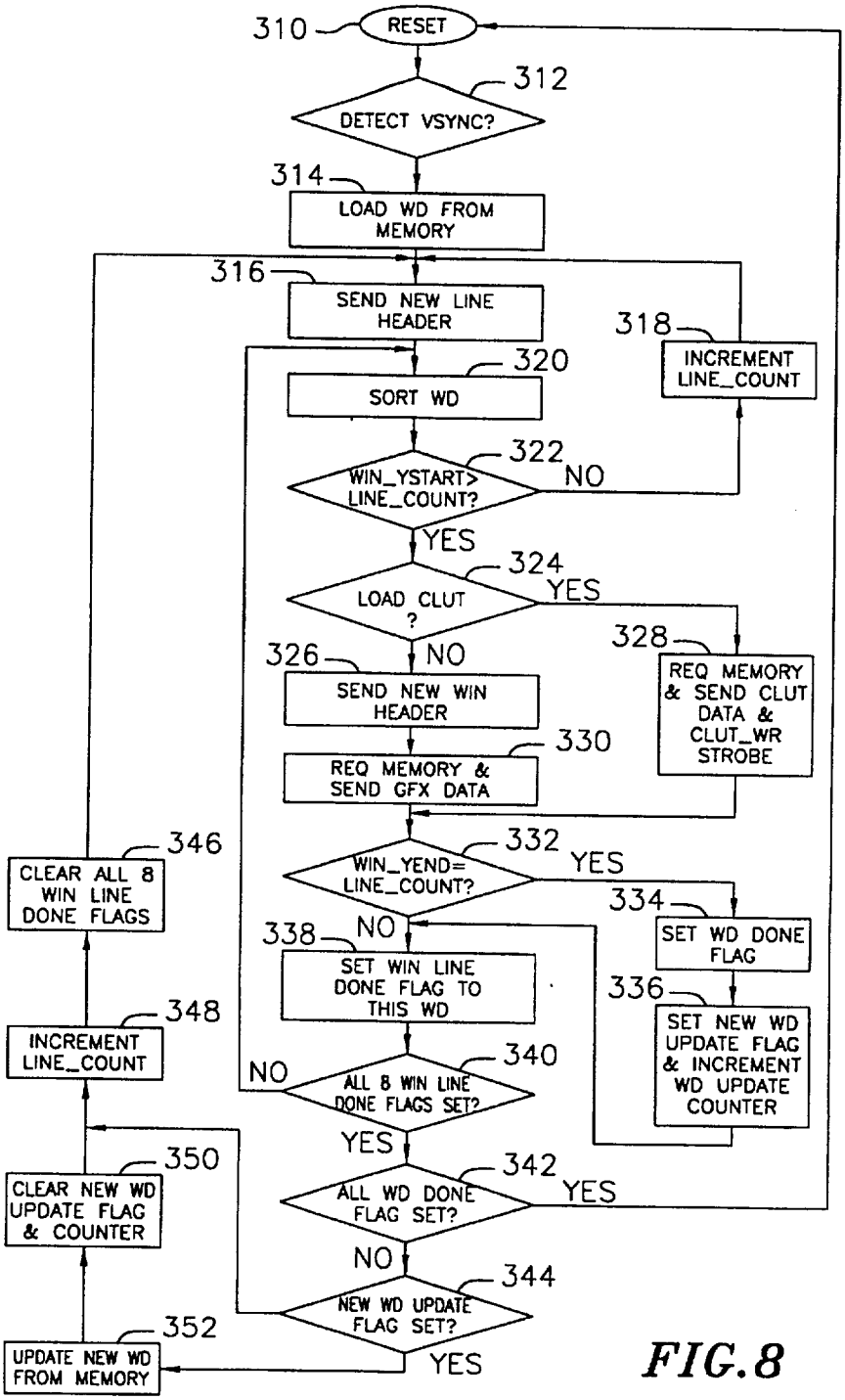

310
RESET
312
DETECT VSYNC?
314
LOAD WD FROM MEMORY
316
SEND NEW LINE HEADER
318
INCREMENT LINE_COUNT
320
SORT WD
322
WIN_YSTART> LINE_COUNT?
NO
YES
324
LOAD CLUT ?
YES
328
REQ MEMORY & SEND CLUT DATA & CLUT_WR STROBE
NO
326
SEND NEW WIN HEADER
330
REQ MEMORY & SEND GFX DATA
332
WIN_YEND= LINE_COUNT?
YES
334
SET WD DONE FLAG
NO
338
SET WIN LINE DONE FLAG TO THIS WD
336
SET NEW WD UPDATE FLAG & INCREMENT WD UPDATE COUNTER
340
ALL 8 WIN LINE DONE FLAGS SET?
NO
YES
342
ALL WD DONE FLAG SET?
YES
NO
344
NEW WD UPDATE FLAG SET?
YES
346
CLEAR ALL 8 WIN LINE DONE FLAGS
348
INCREMENT LINE_COUNT
350
CLEAR NEW WD UPDATE FLAG & COUNTER
352
UPDATE NEW WD FROM MEMORY

FIG. 8

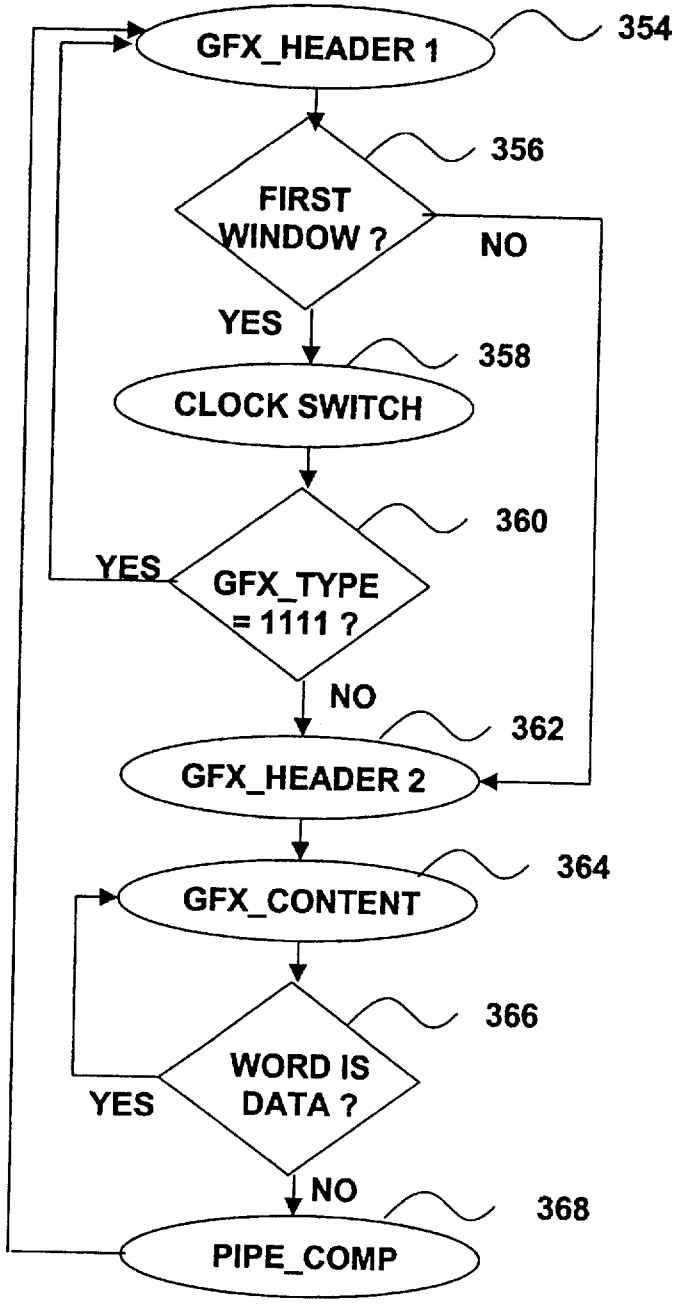

GFX_HEADER 1
354
356
FIRST WINDOW ?
NO
YES
358
CLOCK SWITCH
360
YES
GFX_TYPE = 1111 ?
NO
362
GFX_HEADER 2
364
GFX_CONTENT
366
WORD IS DATA ?
YES
NO
368
PIPE_COMP

FIG. 9

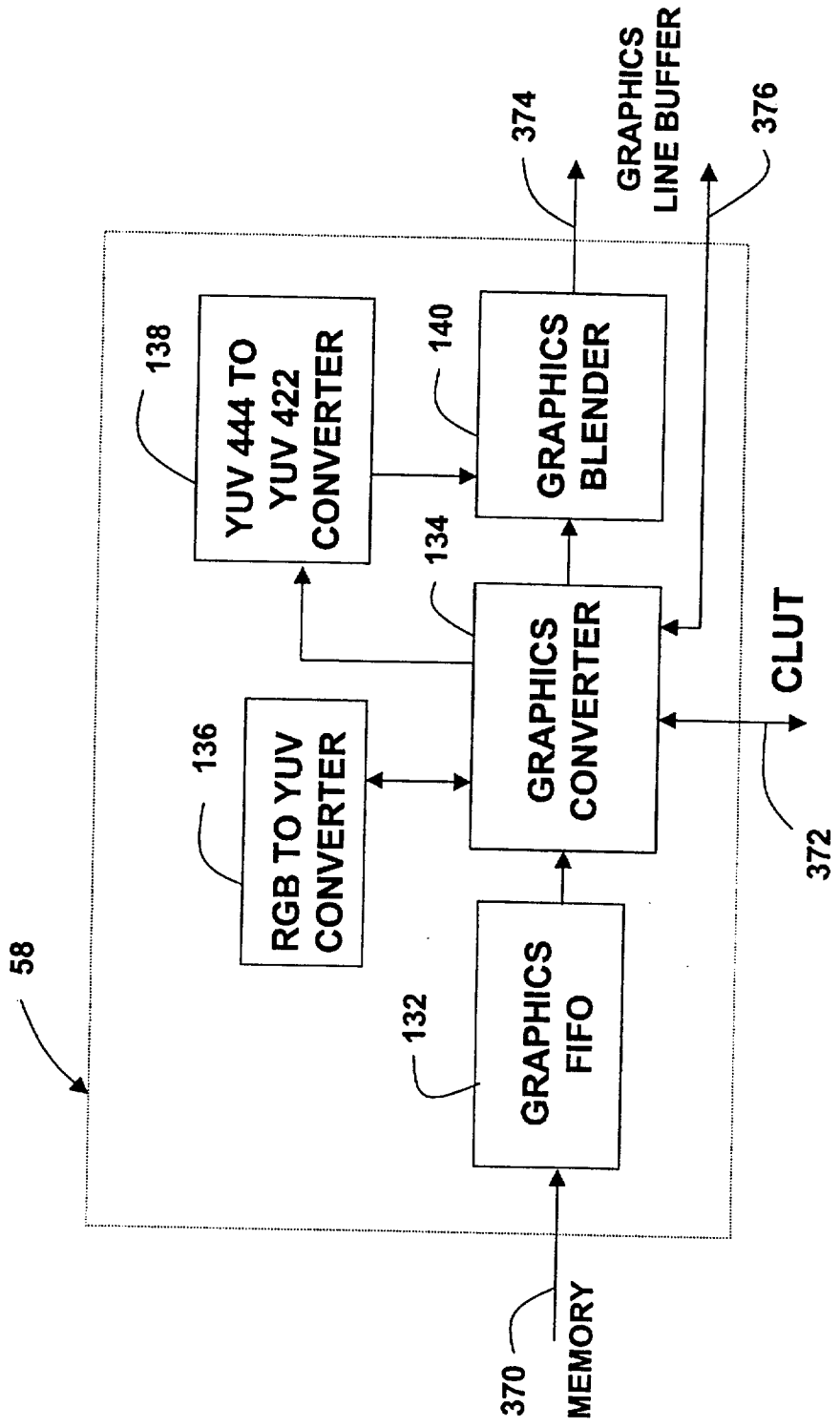

58
370
MEMORY
132
GRAPHICS FIFO
136
RGB TO YUV CONVERTER
138
YUV 444 TO YUV 422 CONVERTER
134
GRAPHICS CONVERTER
140
GRAPHICS BLENDER
372
CLUT
374
376
GRAPHICS LINE BUFFER

FIG. 10

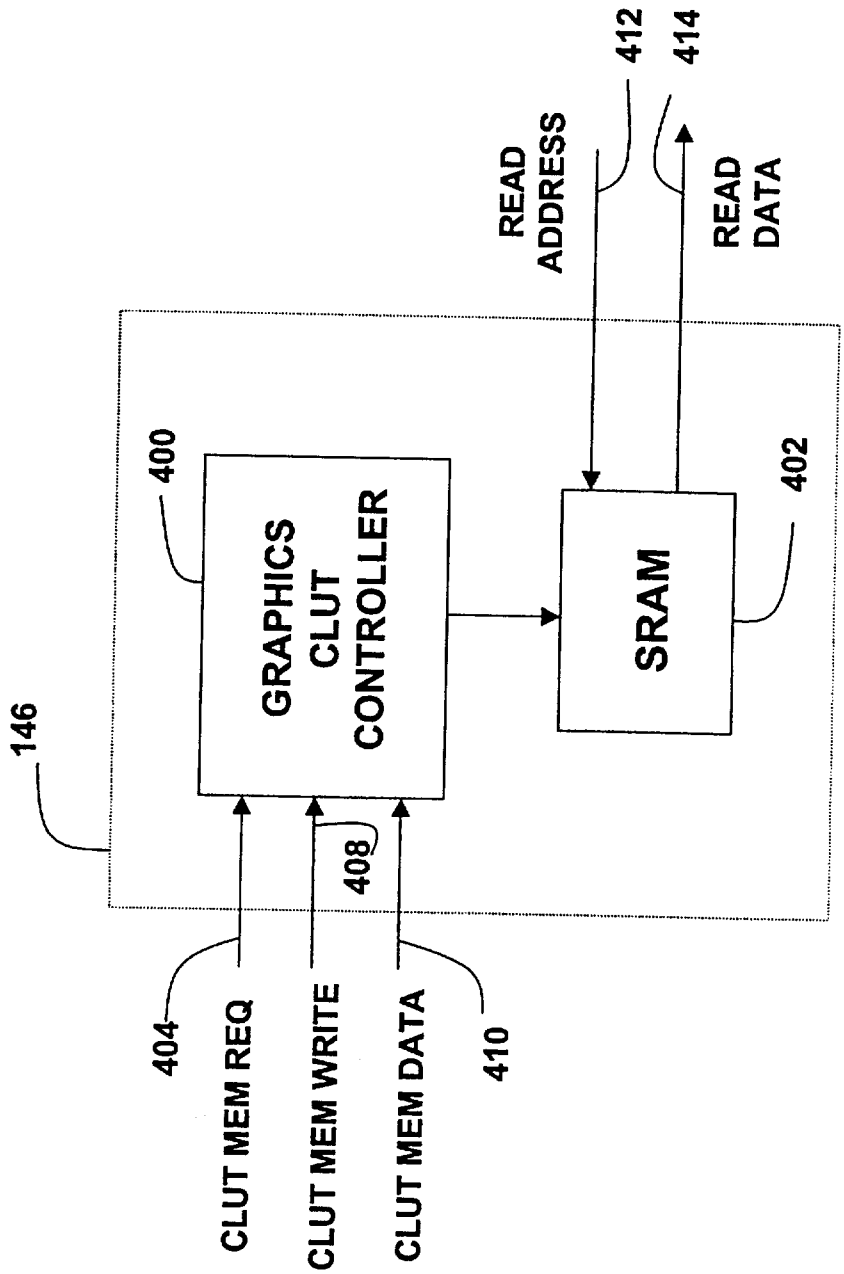

412
414
READ ADDRESS
READ DATA
400
GRAPHICS CLUT CONTROLLER
SRAM
402
146
408
404
CLUT MEM REQ
CLUT MEM WRITE
CLUT MEM DATA
410

FIG. 11

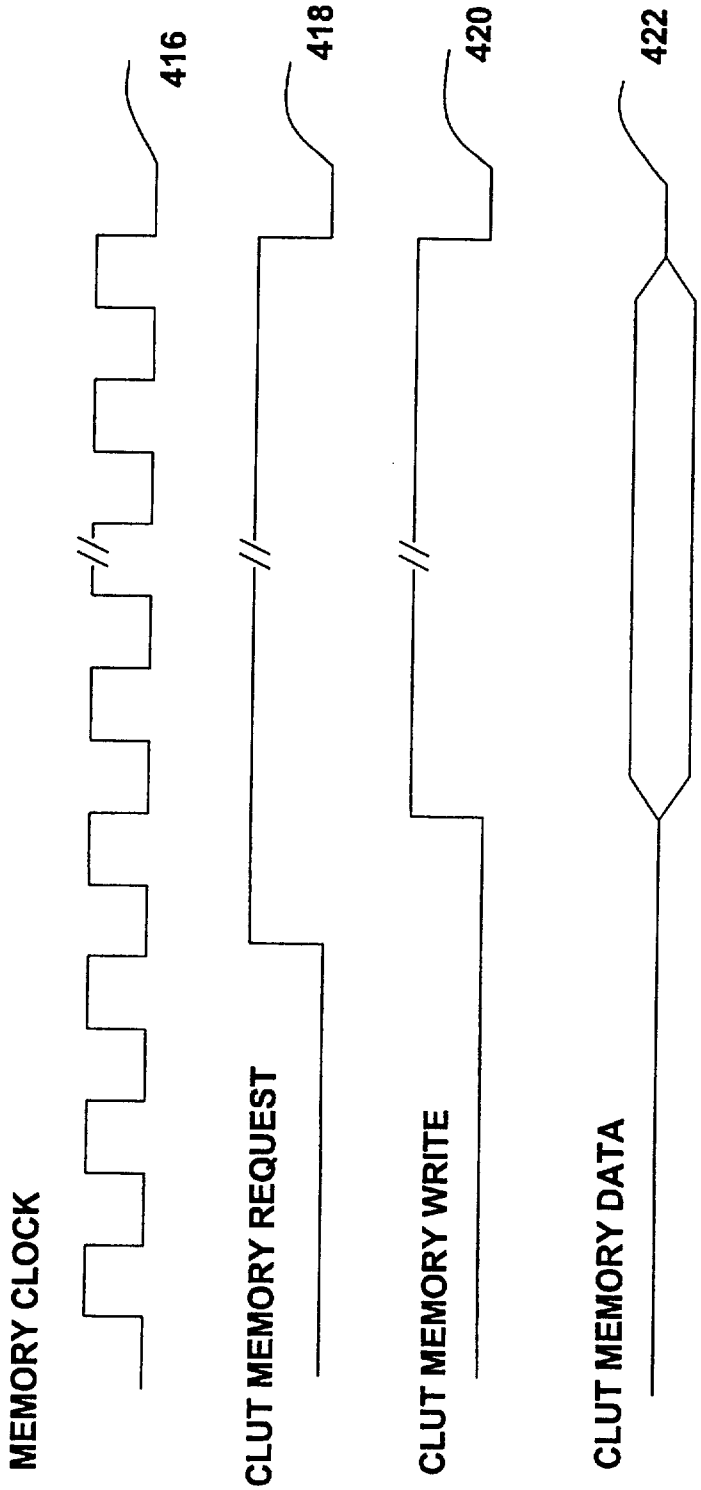

MEMORY CLOCK
CLUT MEMORY REQUEST
CLUT MEMORY WRITE
CLUT MEMORY DATA
416
418
420
422

FIG. 12

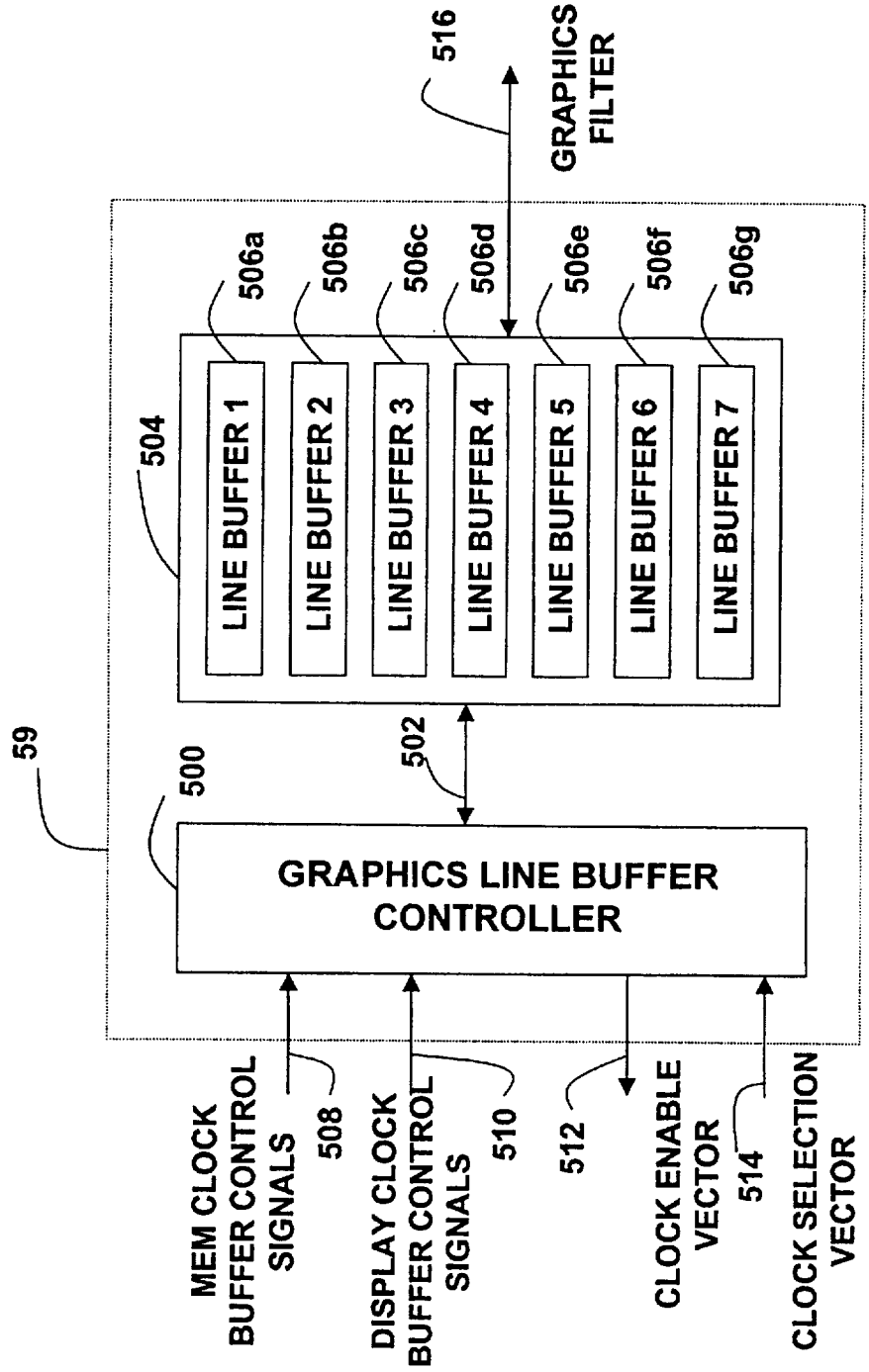

516
GRAPHICS FILTER
506a
506b
506c
506d
506e
506f
506g
504
LINE BUFFER 1
LINE BUFFER 2
LINE BUFFER 3
LINE BUFFER 4
LINE BUFFER 5
LINE BUFFER 6
LINE BUFFER 7
502
59
500
GRAPHICS LINE BUFFER CONTROLLER
MEM CLOCK BUFFER CONTROL SIGNALS
508
DISPLAY CLOCK BUFFER CONTROL SIGNALS
510
512
CLOCK ENABLE VECTOR
514
CLOCK SELECTION VECTOR

FIG. 13

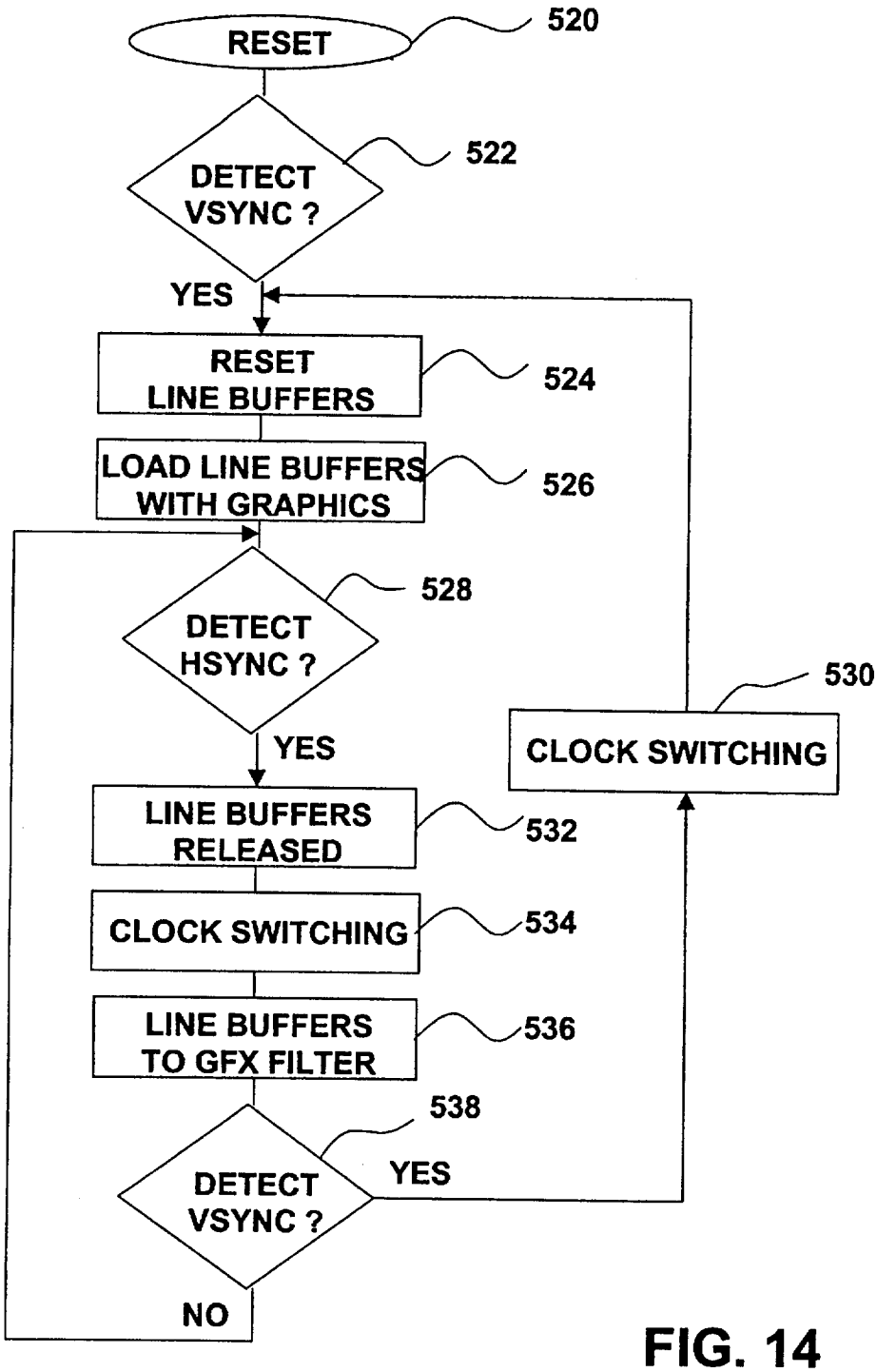

RESET
520
DETECT VSYNC ?
522
YES
RESET LINE BUFFERS
524
LOAD LINE BUFFERS WITH GRAPHICS
526
DETECT HSYNC ?
528
YES
LINE BUFFERS RELEASED
532
CLOCK SWITCHING
534
LINE BUFFERS TO GFX FILTER
536
DETECT VSYNC ?
538
YES
NO
CLOCK SWITCHING
530

FIG. 14

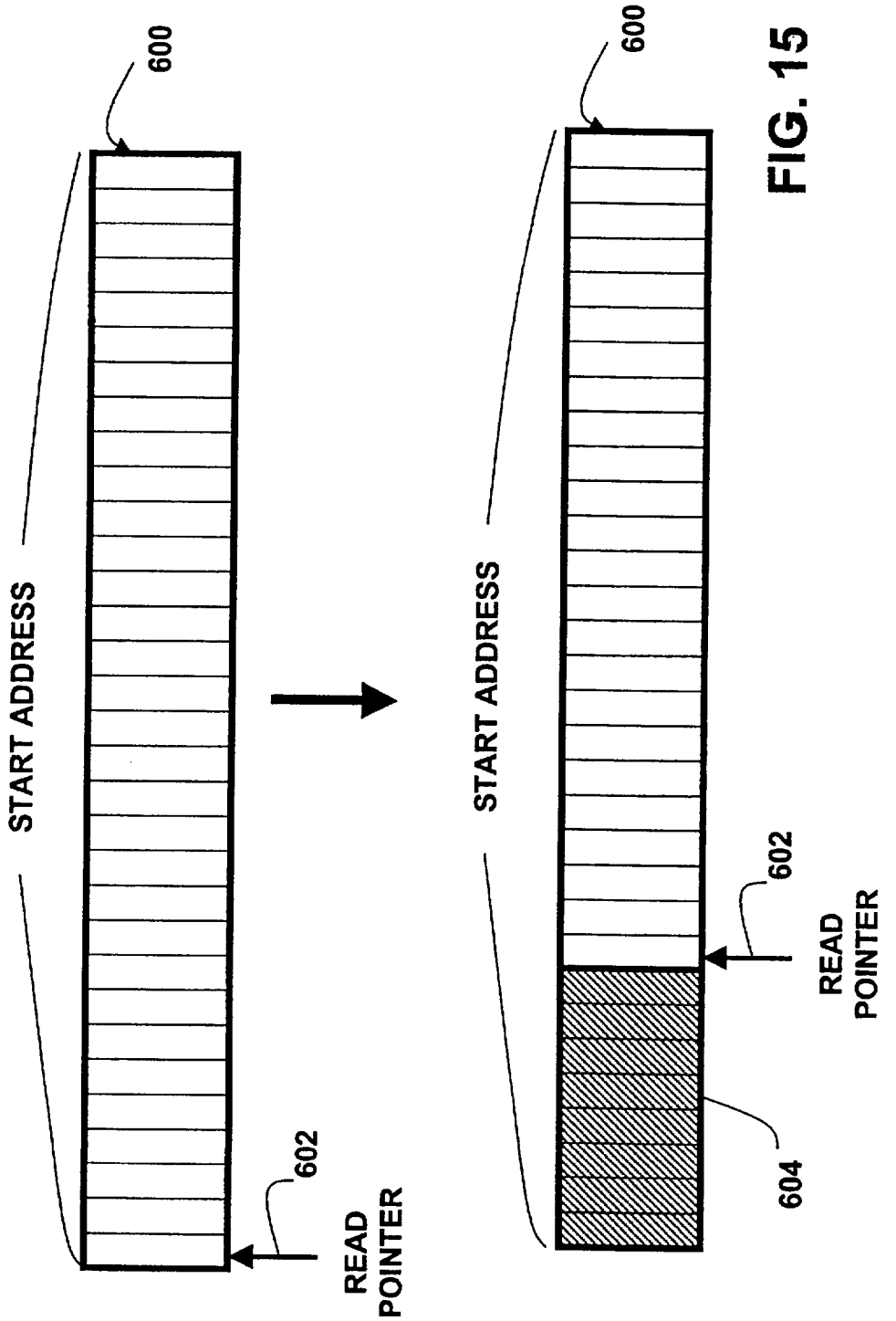

600
START ADDRESS
602
READ POINTER
600
START ADDRESS
602
READ POINTER
604

FIG. 15

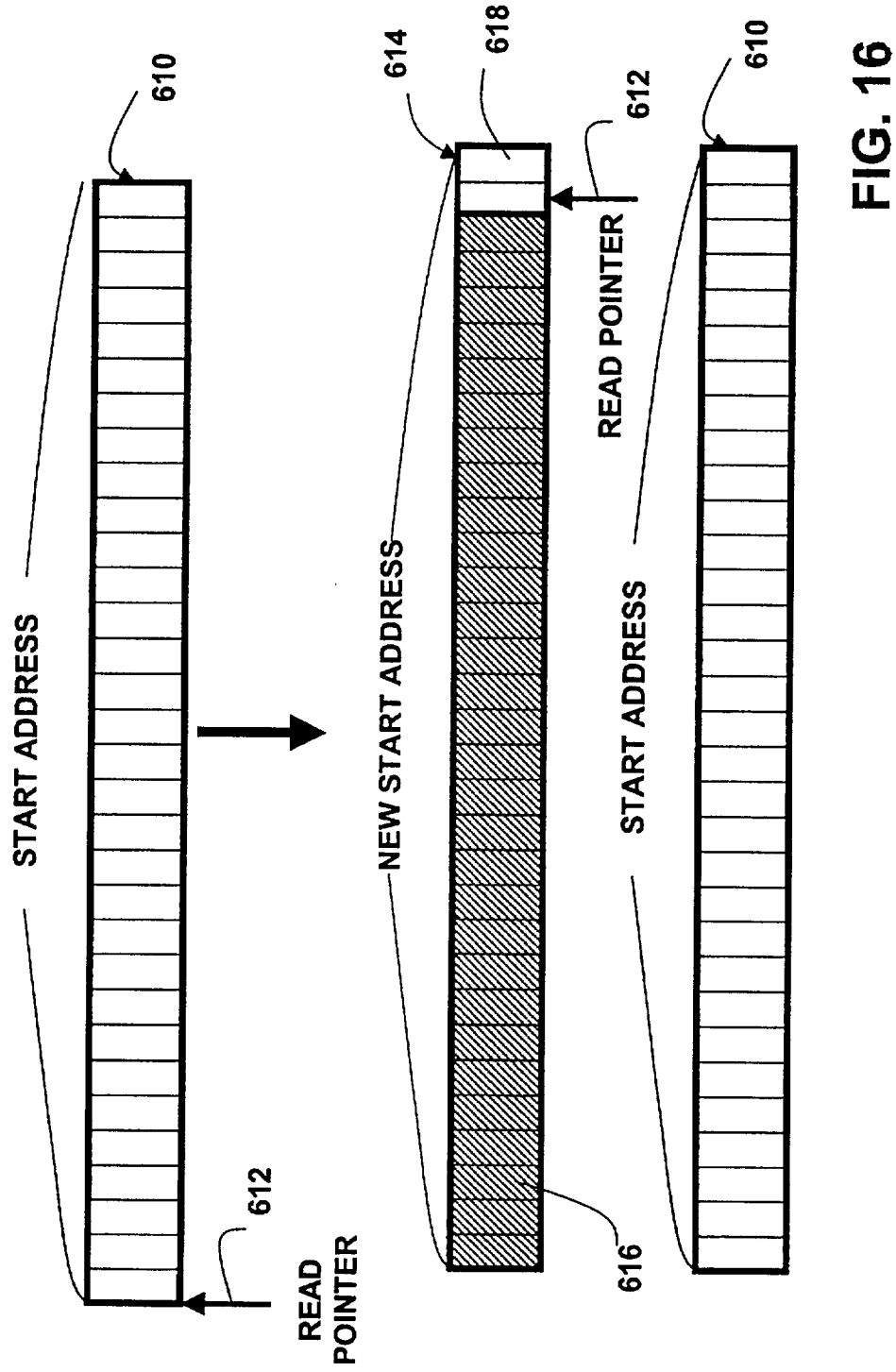

610
START ADDRESS
612
READ POINTER
614
618
NEW START ADDRESS
616
612
READ POINTER
610
START ADDRESS

FIG. 16

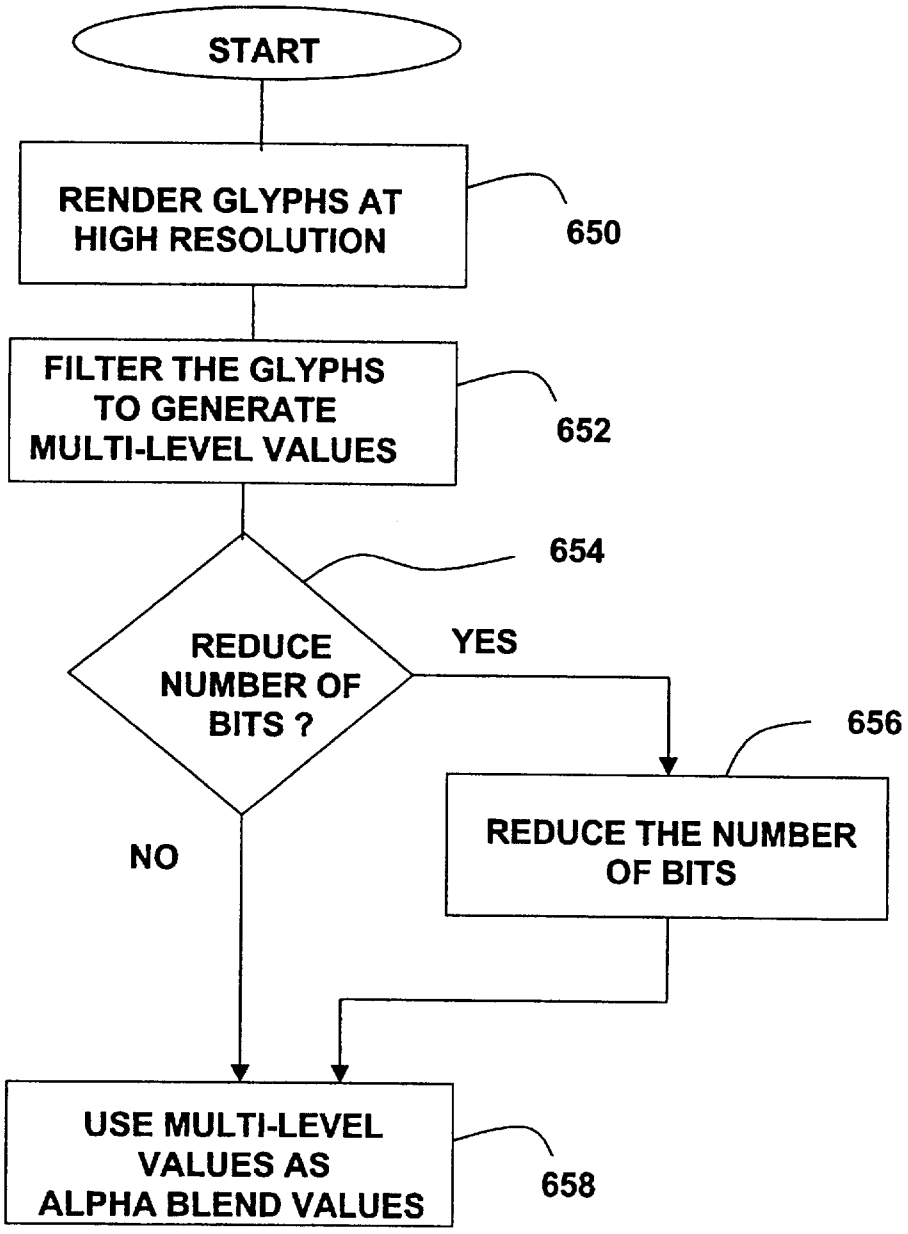

START
RENDER GLYPHS AT HIGH RESOLUTION
650
FILTER THE GLYPHS TO GENERATE MULTI-LEVEL VALUES
652
654
REDUCE NUMBER OF BITS ?
YES
NO
656
REDUCE THE NUMBER OF BITS
USE MULTI-LEVEL VALUES AS ALPHA BLEND VALUES
658

FIG. 17

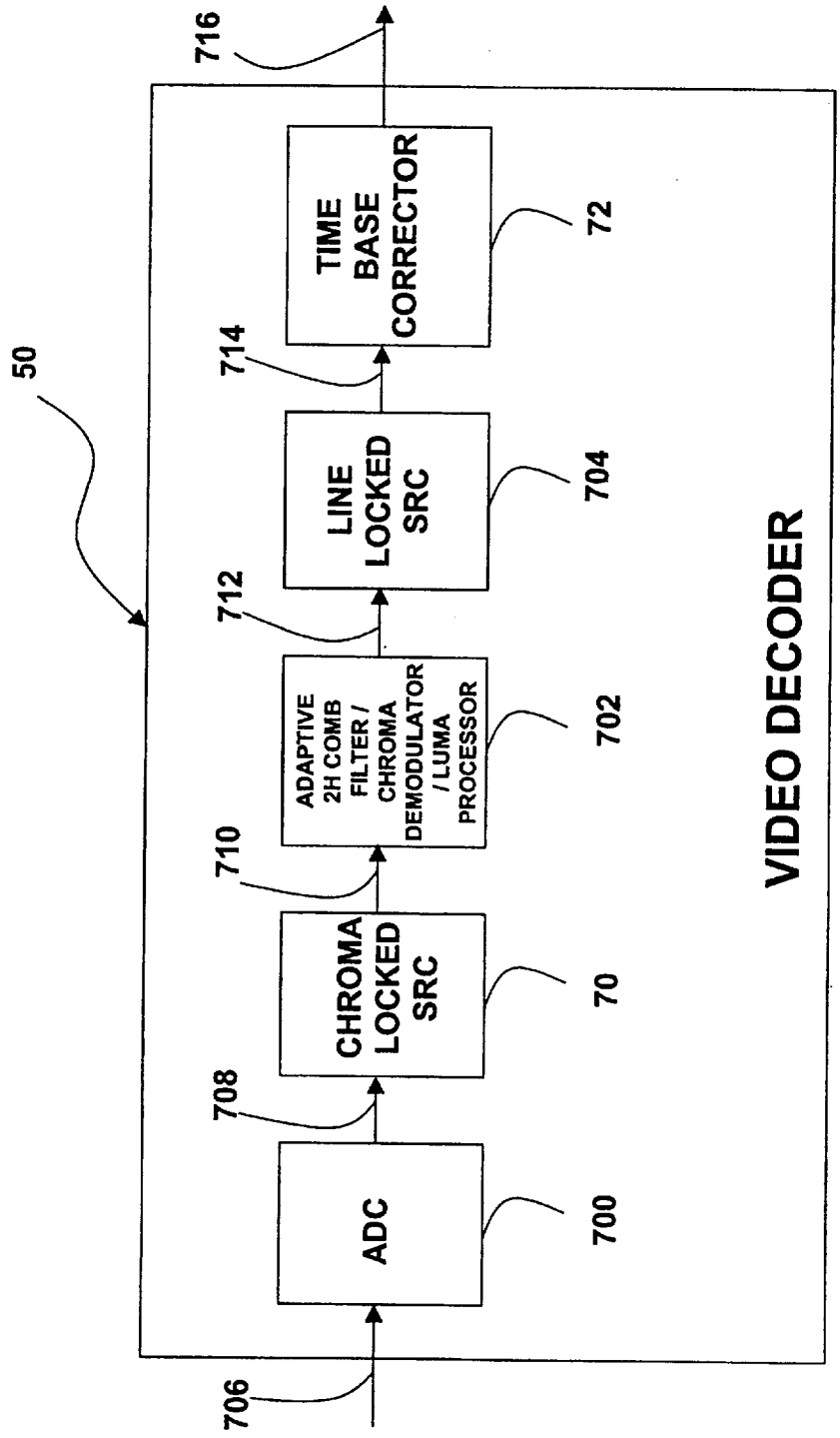

50
VIDEO DECODER
706
ADC
700
708
CHROMA LOCKED SRC
70
710
ADAPTIVE 2H COMB FILTER / CHROMA DEMODULATOR / LUMA PROCESSOR
702
712
LINE LOCKED SRC
704
714
TIME BASE CORRECTOR
72
716

FIG. 18

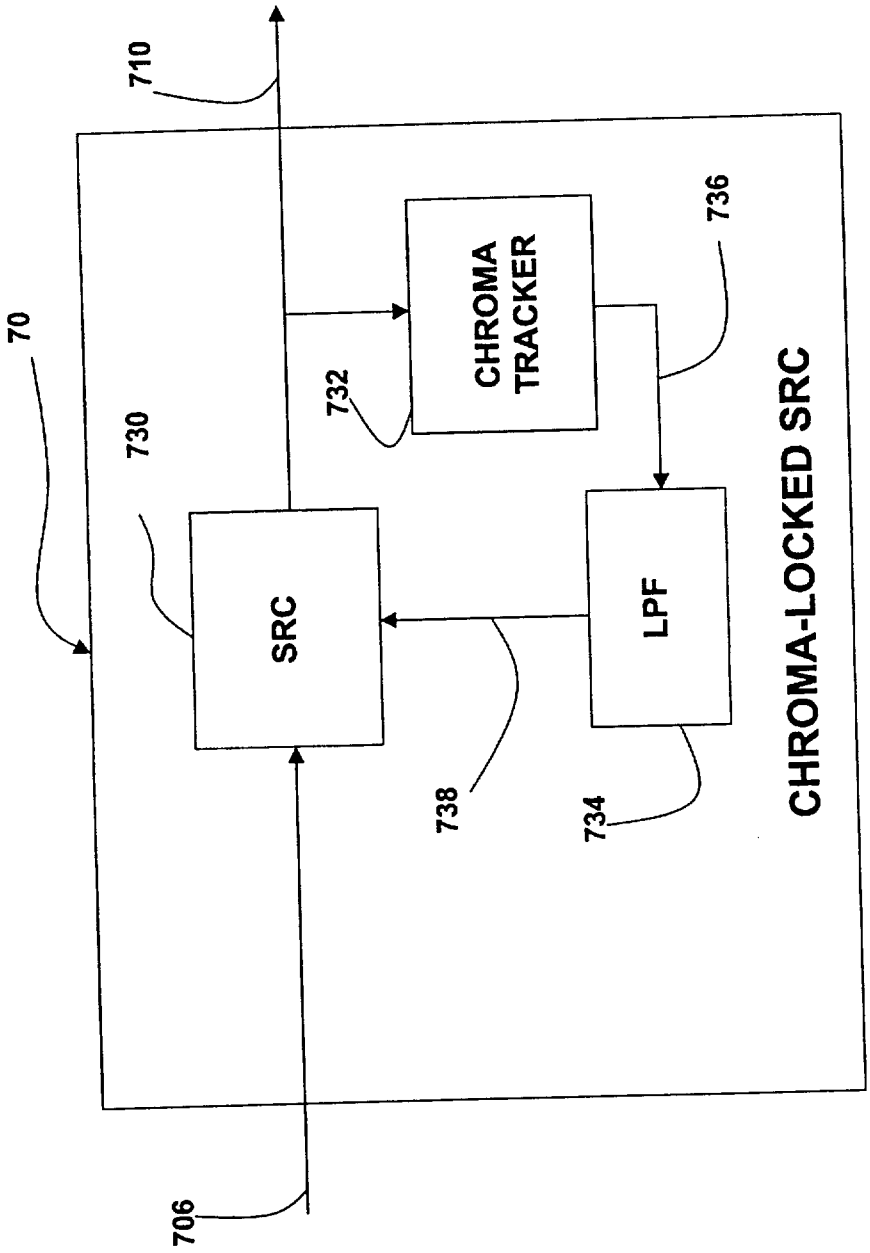

710
70
730
732
736
CHROMA TRACKER
SRC
LPF
738
734
CHROMA-LOCKED SRC
706

FIG. 19

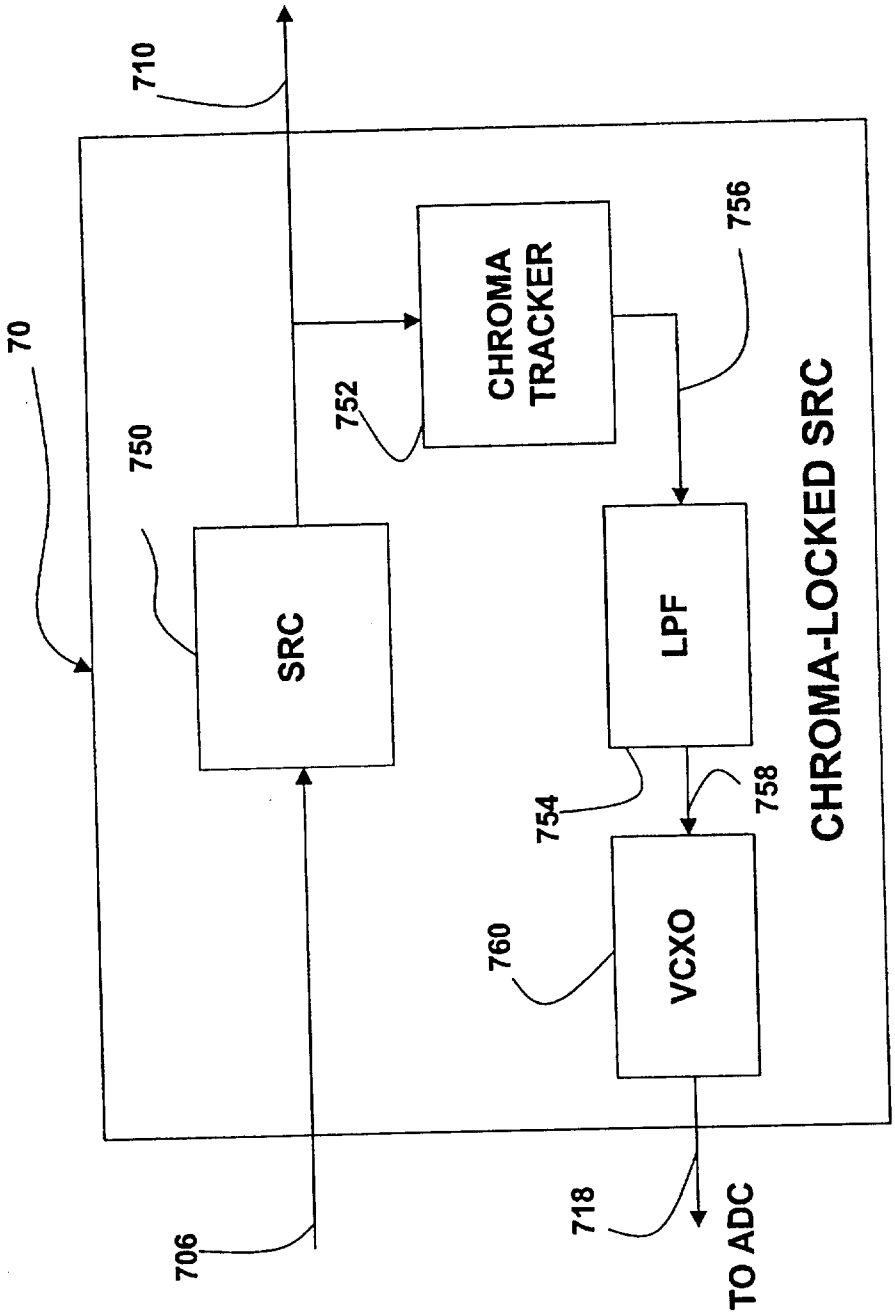

710
70
750
752
CHROMA TRACKER
756
SRC
LPF
754
758
760
VCXO
CHROMA-LOCKED SRC
706
718
TO ADC

FIG. 20

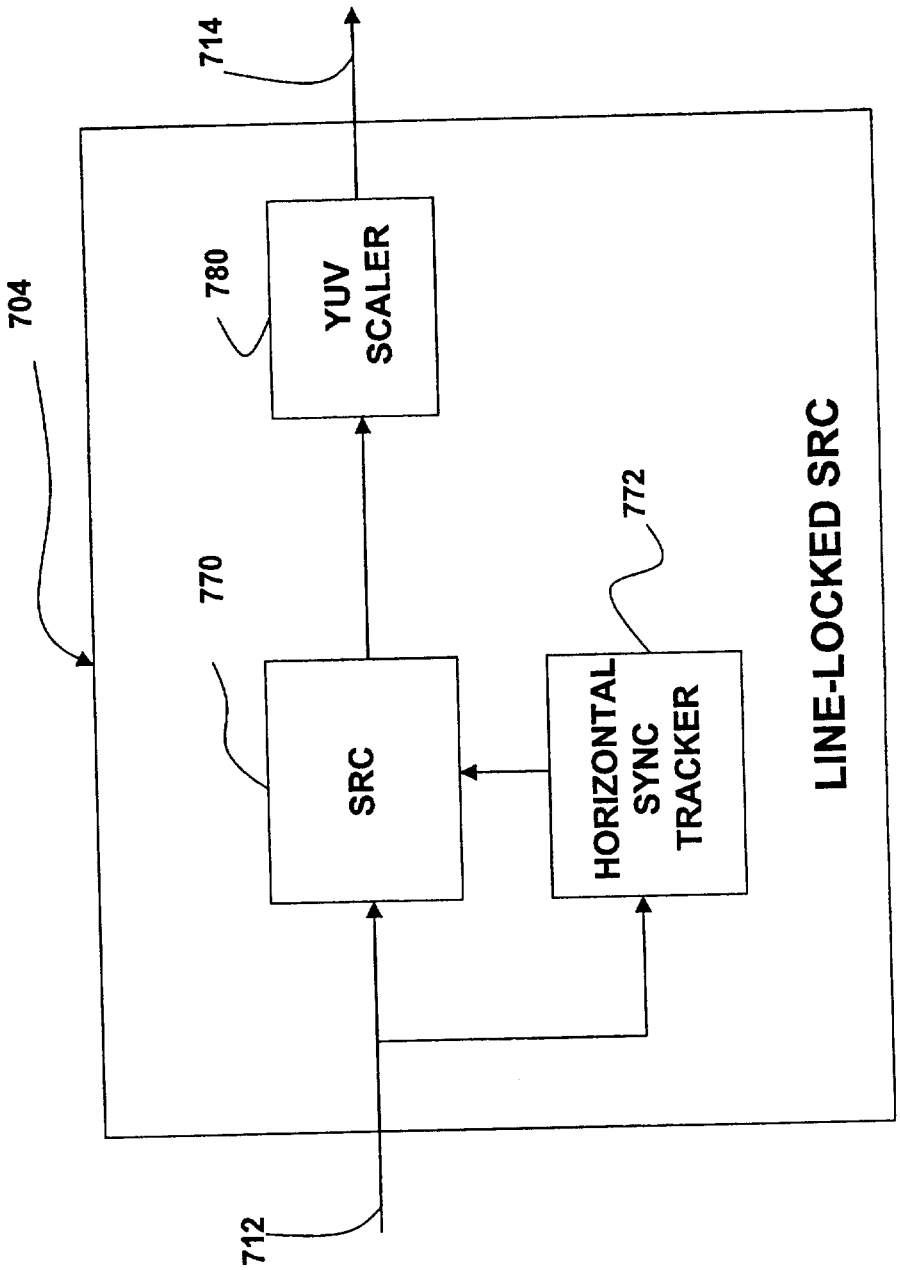

714
704
780
YUV SCALER
770
772
SRC
HORIZONTAL SYNC TRACKER
LINE-LOCKED SRC
712

FIG. 21

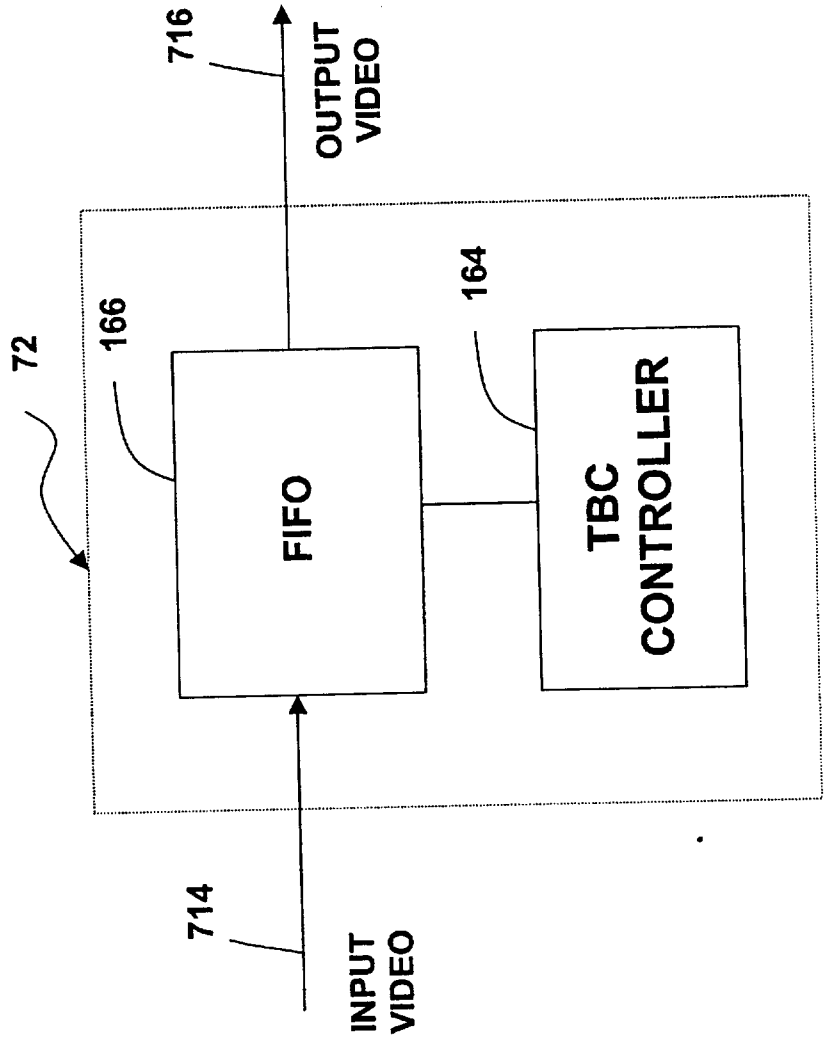

716
OUTPUT VIDEO
72
166
164
FIFO
TBC CONTROLLER
714
INPUT VIDEO

FIG. 22

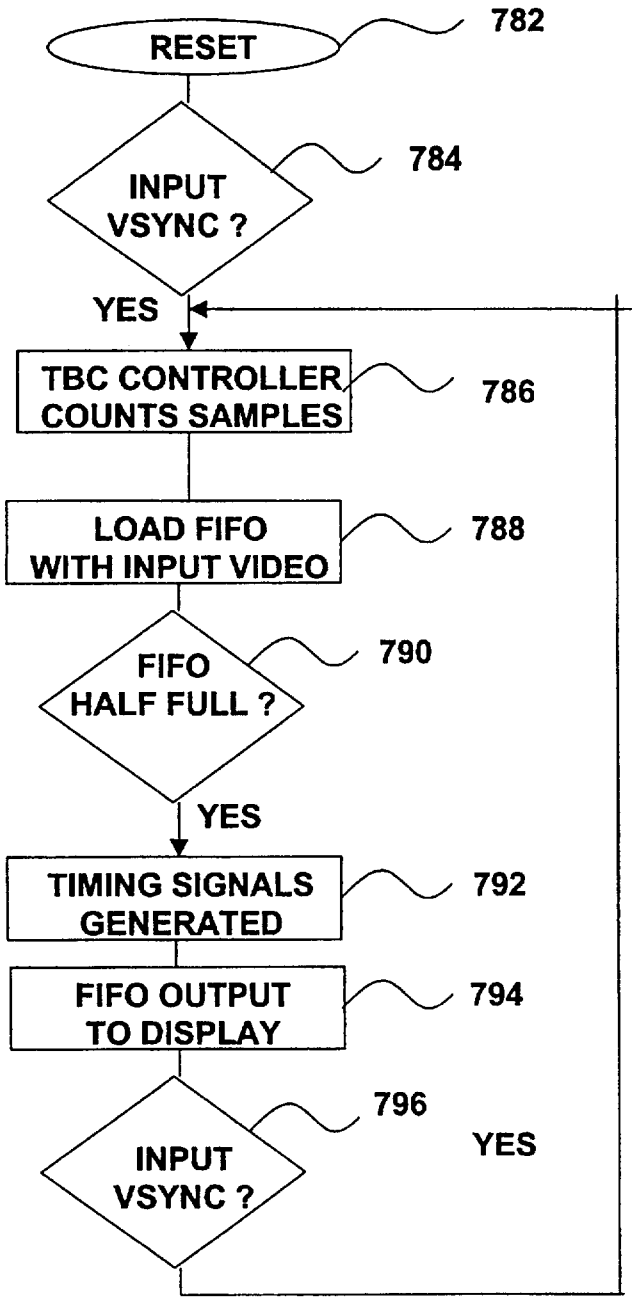

RESET
782
INPUT VSYNC ?
784
YES
TBC CONTROLLER COUNTS SAMPLES
786
LOAD FIFO WITH INPUT VIDEO
788
FIFO HALF FULL ?
790
YES
TIMING SIGNALS GENERATED
792
FIFO OUTPUT TO DISPLAY
794
INPUT VSYNC ?
796
YES

FIG. 23

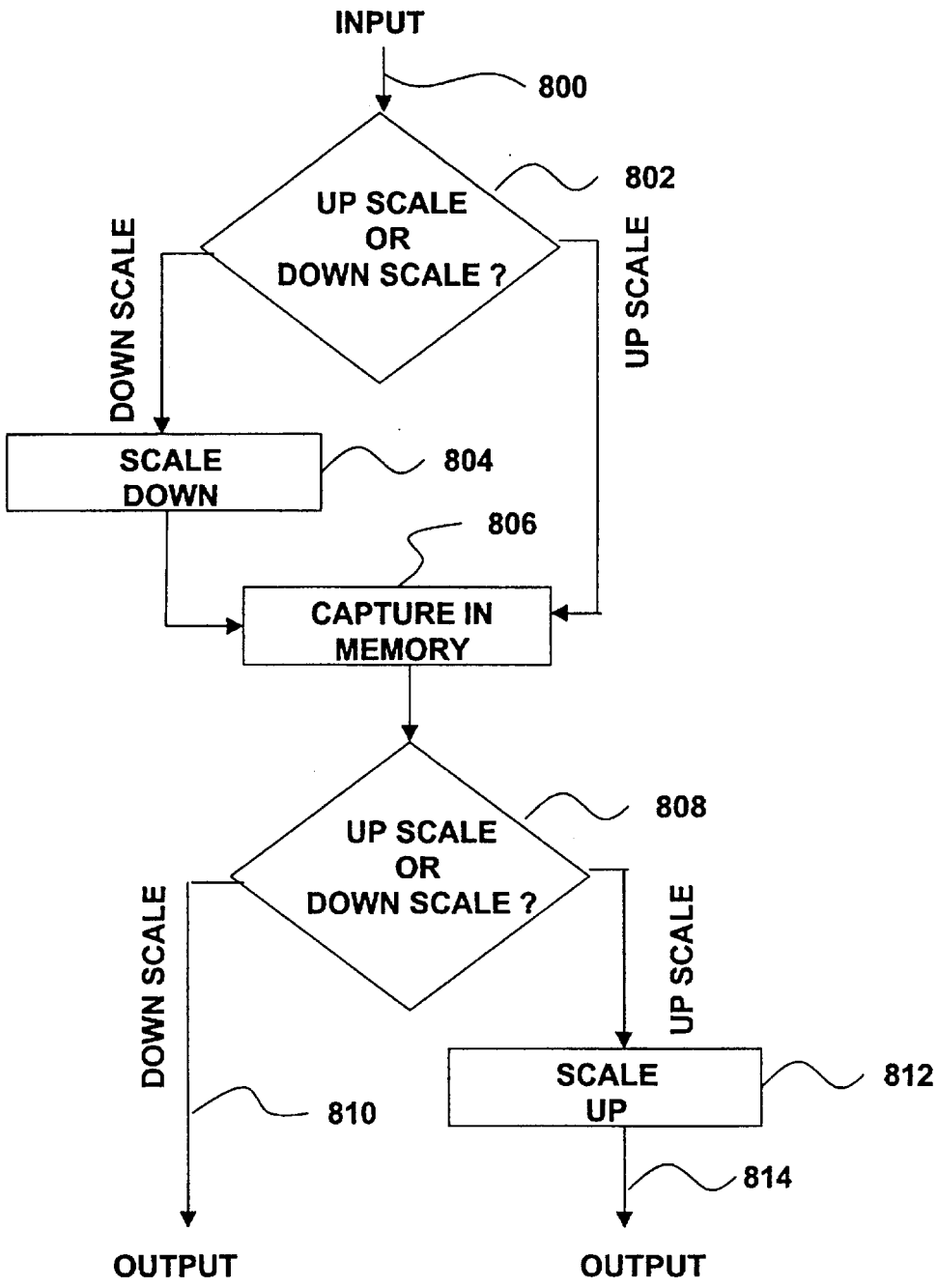

INPUT
800
UP SCALE OR DOWN SCALE ?
802
DOWN SCALE
UP SCALE
SCALE DOWN
804
806
CAPTURE IN MEMORY
UP SCALE OR DOWN SCALE ?
808
DOWN SCALE
UP SCALE
SCALE UP
812
810
814
OUTPUT
OUTPUT

FIG. 24

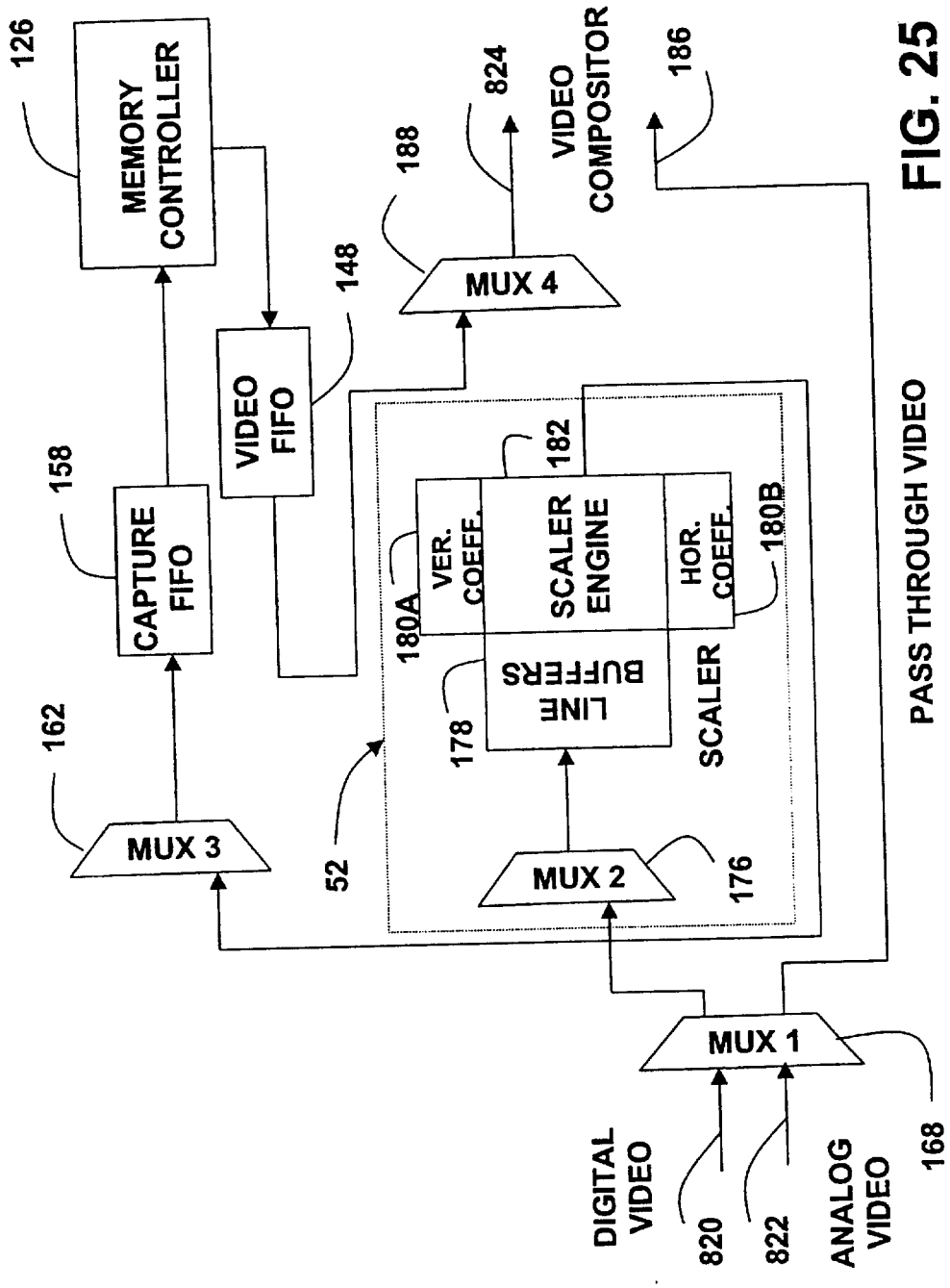

126
MEMORY CONTROLLER
824
VIDEO COMPOSITOR
186
188
MUX 4
148
VIDEO FIFO
158
CAPTURE FIFO
182
VER. COEFF.
SCALER ENGINE
HOR. COEFF.
180B
180A
LINE BUFFERS
178
SCALER
162
MUX 3
52
MUX 2
176
PASS THROUGH VIDEO
MUX 1
DIGITAL VIDEO
820
822
ANALOG VIDEO
168

FIG. 25

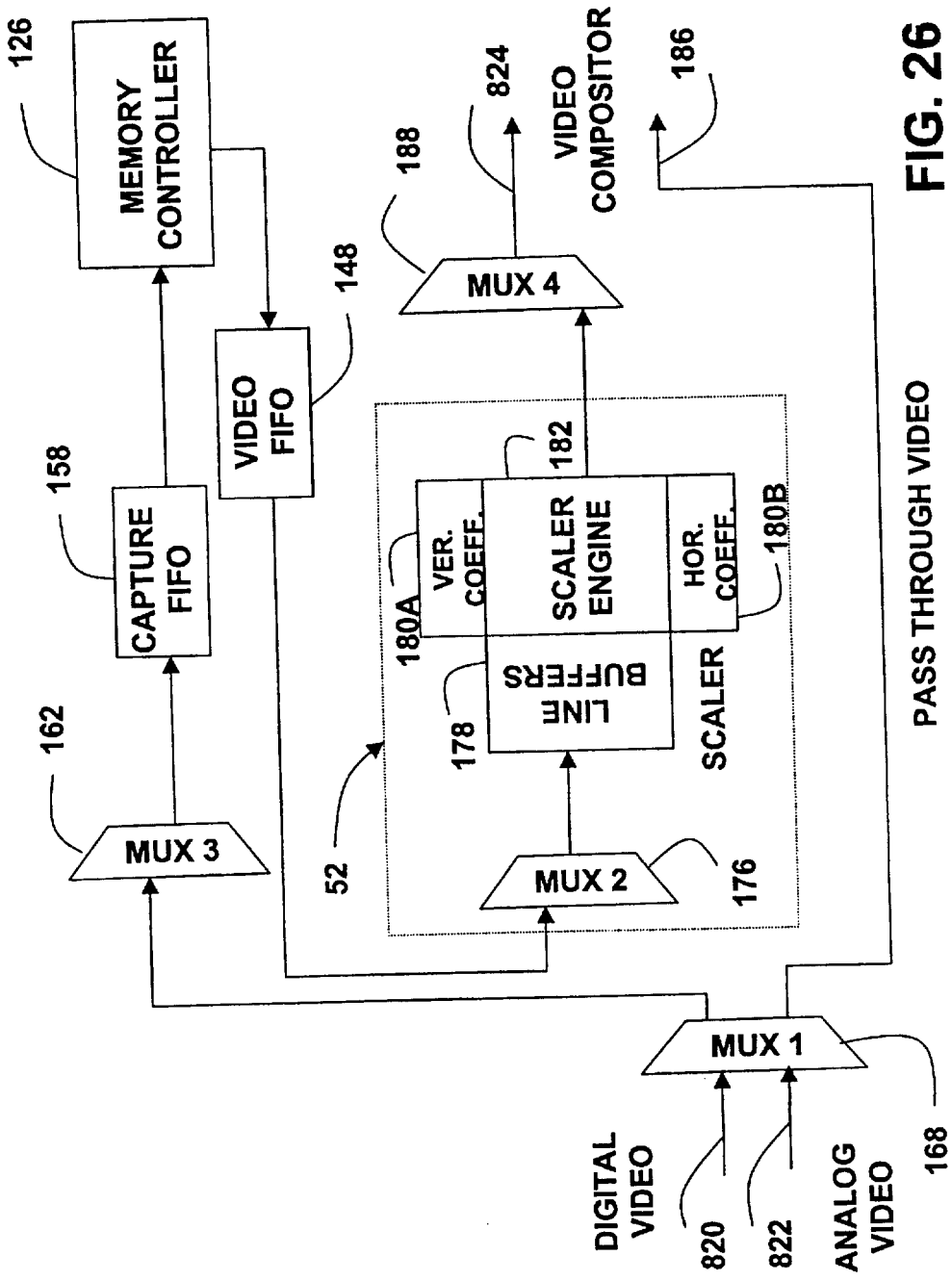

126
MEMORY CONTROLLER
824
VIDEO COMPOSITOR
186
188
MUX 4
148
VIDEO FIFO
158
CAPTURE FIFO
182
VER. COEFF.
SCALER ENGINE
HOR. COEFF.
180B
180A
LINE BUFFERS
178
SCALER
162
MUX 3
52
MUX 2
176
PASS THROUGH VIDEO
MUX 1
DIGITAL VIDEO
820
822
ANALOG VIDEO
168

FIG. 26

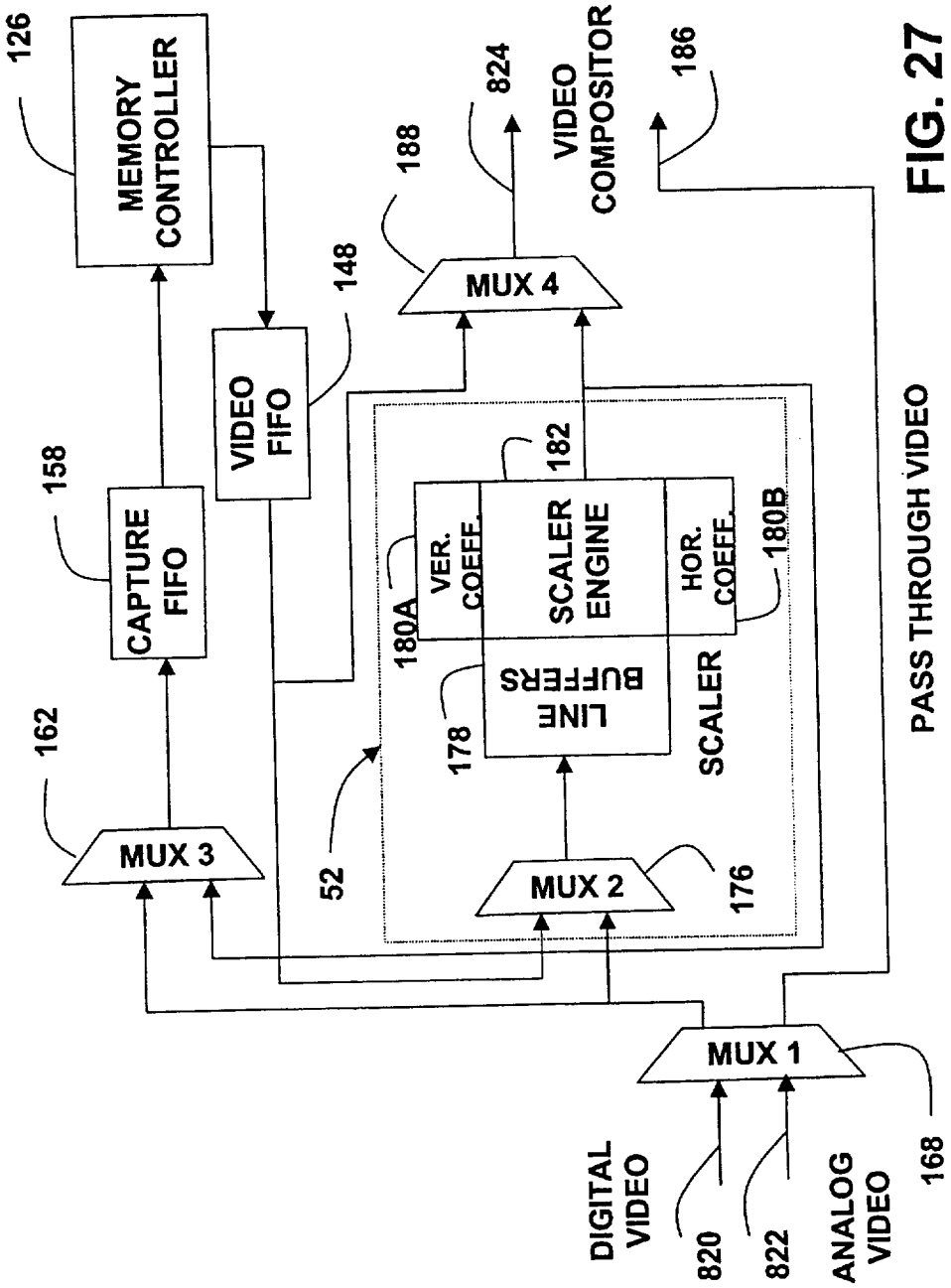

126
MEMORY CONTROLLER
824
VIDEO COMPOSITOR
186
188
148
MUX 4
VIDEO FIFO
158
CAPTURE FIFO
182
VER. COEFF.
SCALER ENGINE
HOR. COEFF.
180B
180A
LINE BUFFERS
178
SCALER
162
MUX 3
52
MUX 2
176
PASS THROUGH VIDEO
MUX 1
DIGITAL VIDEO
820
822
ANALOG VIDEO
168

FIG. 27

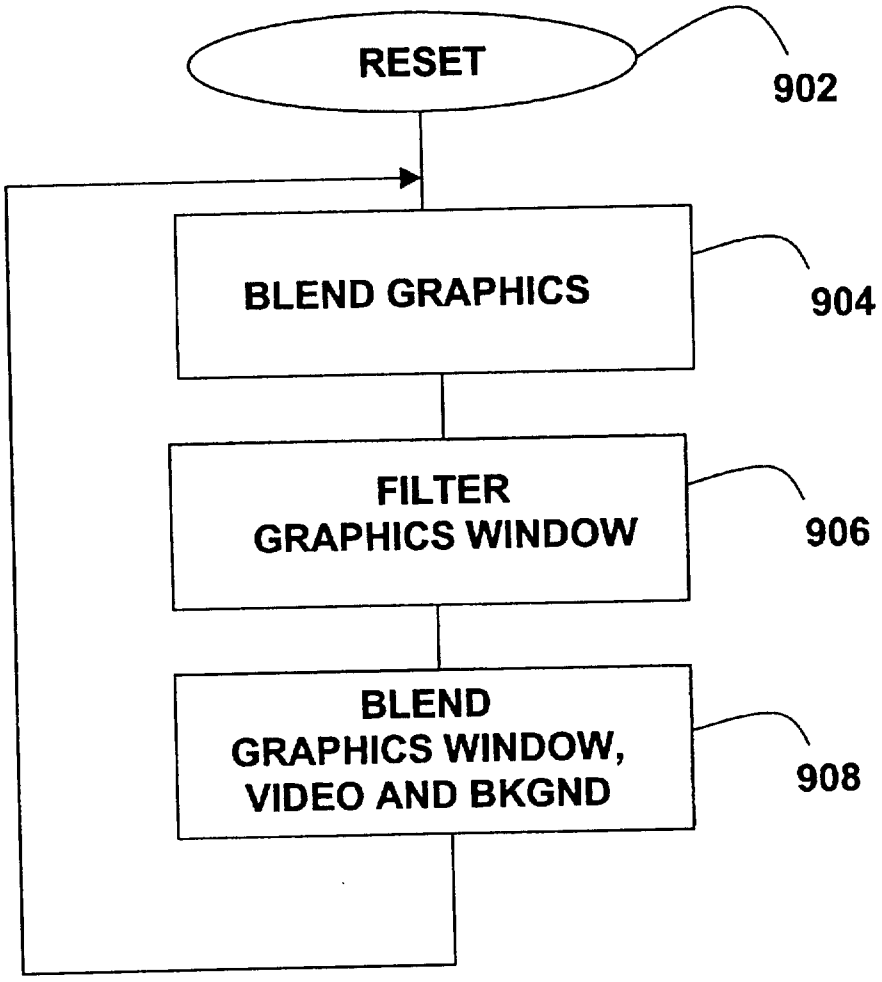

RESET
902
BLEND GRAPHICS
904
FILTER
GRAPHICS WINDOW
906
BLEND
GRAPHICS WINDOW,
VIDEO AND BKGND
908

FIG. 28

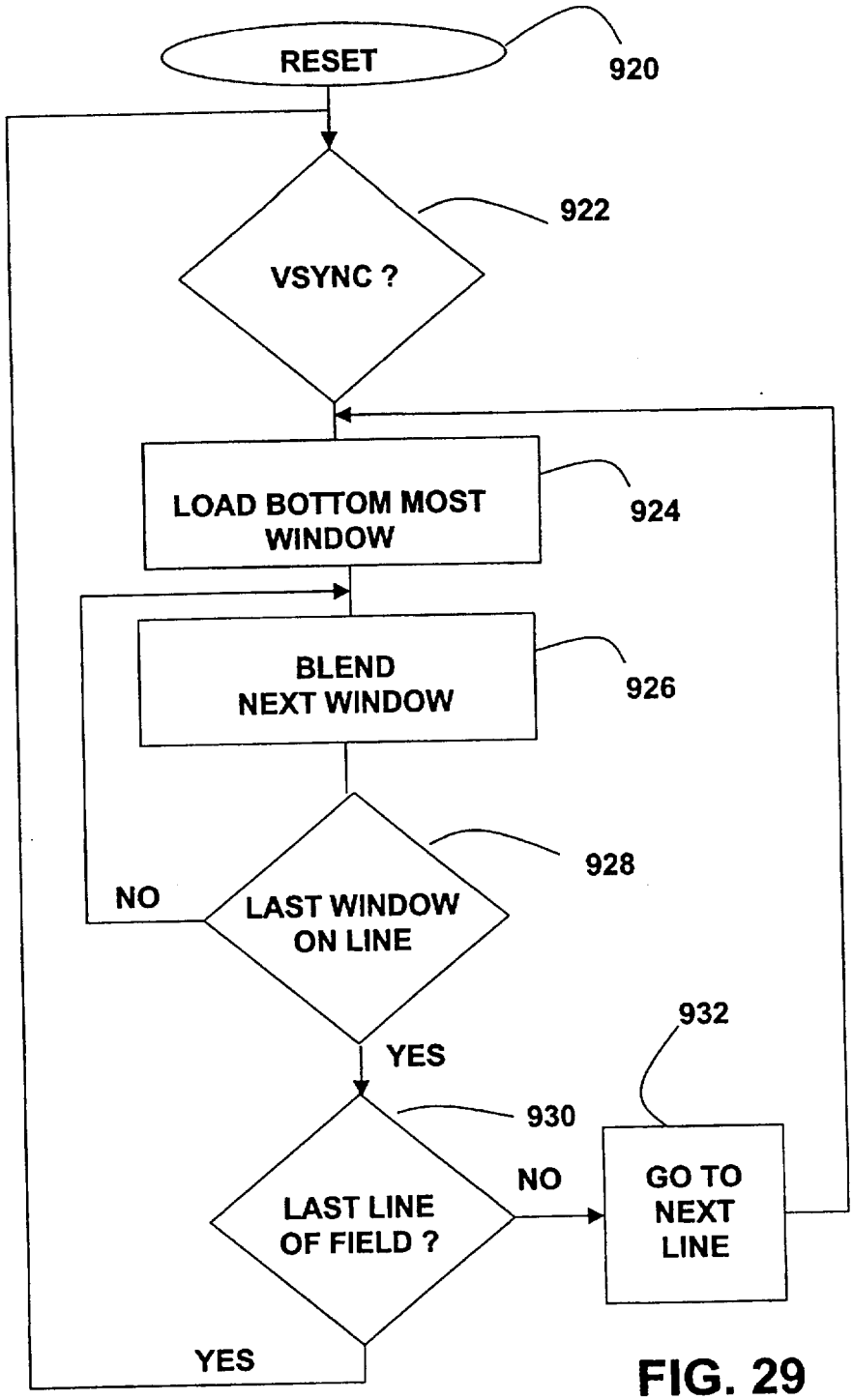

RESET
920
VSYNC ?
922
LOAD BOTTOM MOST WINDOW
924
BLEND NEXT WINDOW
926
LAST WINDOW ON LINE
928
NO
YES
LAST LINE OF FIELD ?
930
NO
GO TO NEXT LINE
932
YES

FIG. 29

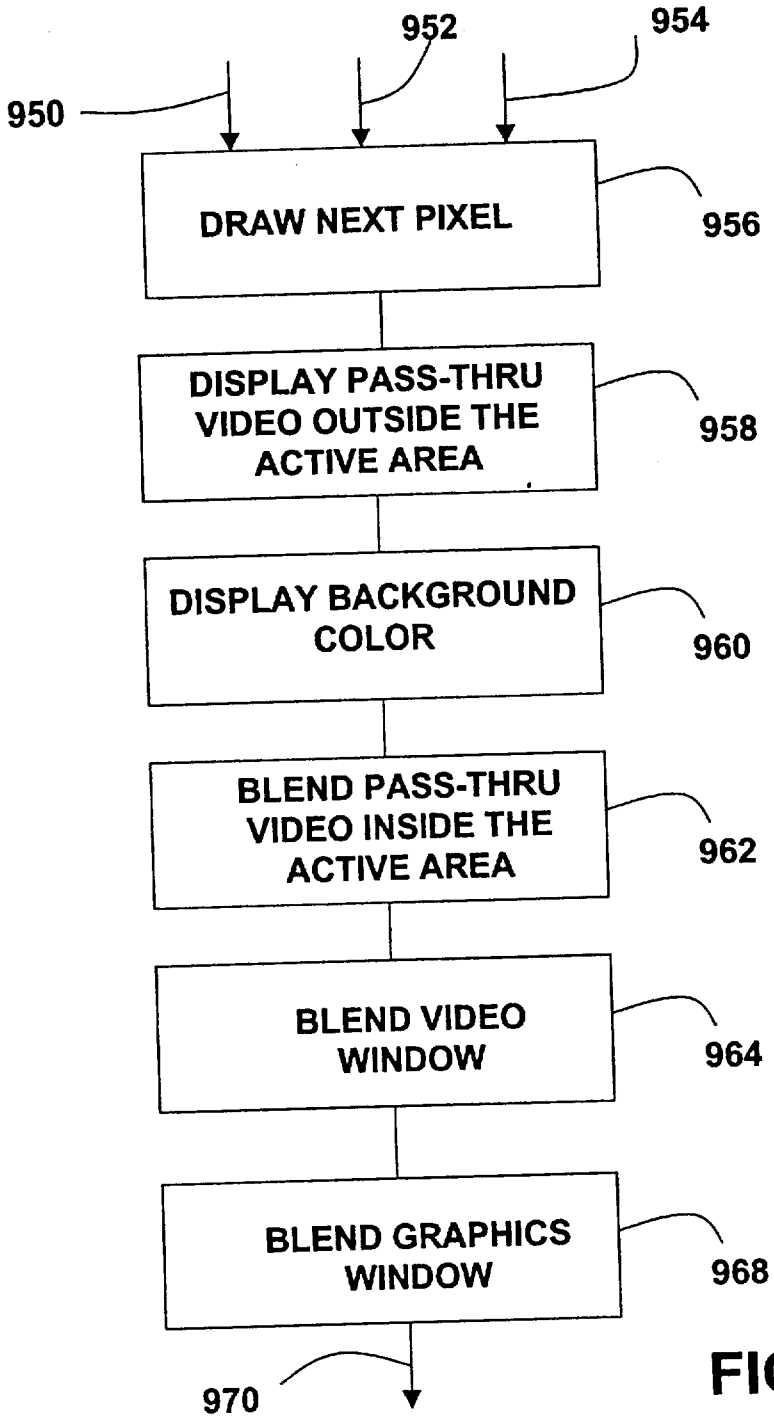

950
952
954
DRAW NEXT PIXEL
956
DISPLAY PASS-THRU VIDEO OUTSIDE THE ACTIVE AREA
958
DISPLAY BACKGROUND COLOR
960
BLEND PASS-THRU VIDEO INSIDE THE ACTIVE AREA
962
BLEND VIDEO WINDOW
964
BLEND GRAPHICS WINDOW
968
970

FIG. 30

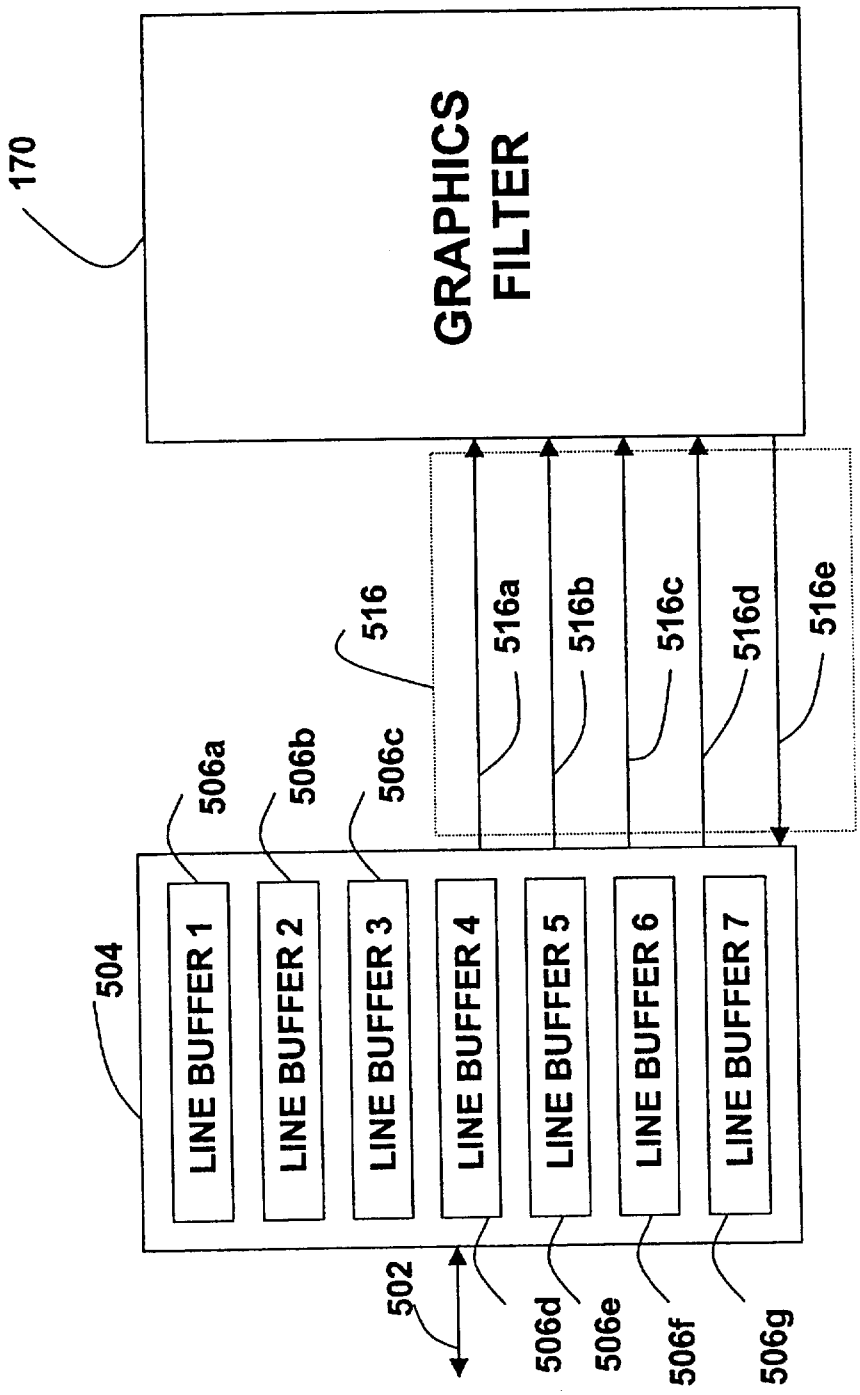

170
GRAPHICS FILTER
516
516a
516b
516c
516d
516e
506a
506b
506c
504
LINE BUFFER 1
LINE BUFFER 2
LINE BUFFER 3
LINE BUFFER 4
LINE BUFFER 5
LINE BUFFER 6
LINE BUFFER 7
502
506d
506e
506f
506g

FIG. 31

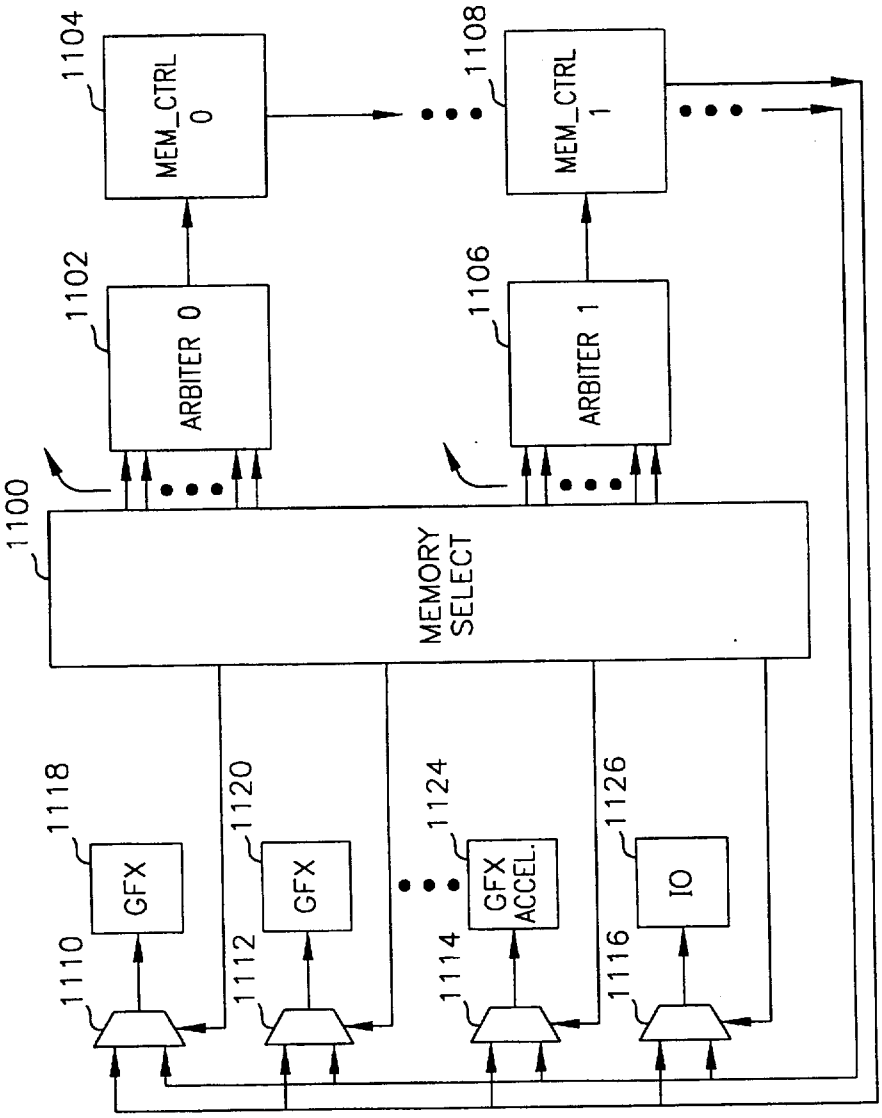

1104
MEM_CTRL 0
1108
MEM_CTRL 1
1102
ARBITER 0
1106
ARBITER 1
1100
MEMORY SELECT
1118
GFX
1120
GFX
1124
GFX ACCEL.
1126
IO
1110
1112
1114
1116

FIG.32

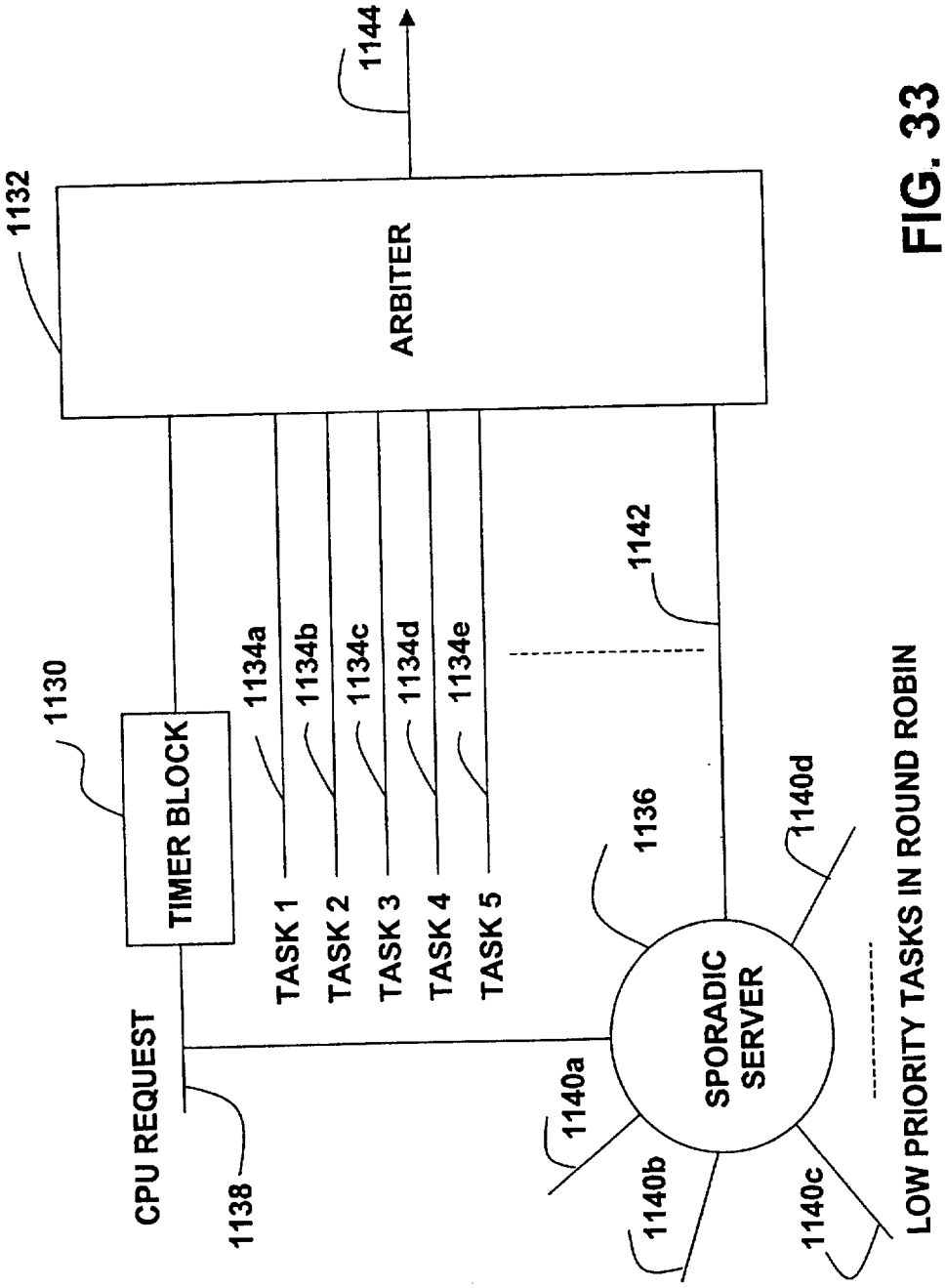

1144
1132
ARBITER
1142
1130
TIMER BLOCK
1134a
1134b
1134c
1134d
1134e
TASK 1
TASK 2
TASK 3
TASK 4
TASK 5
1136
1140d
SPORADIC SERVER
1140a
1140b
1140c
CPU REQUEST
1138
LOW PRIORITY TASKS IN ROUND ROBIN

FIG. 33

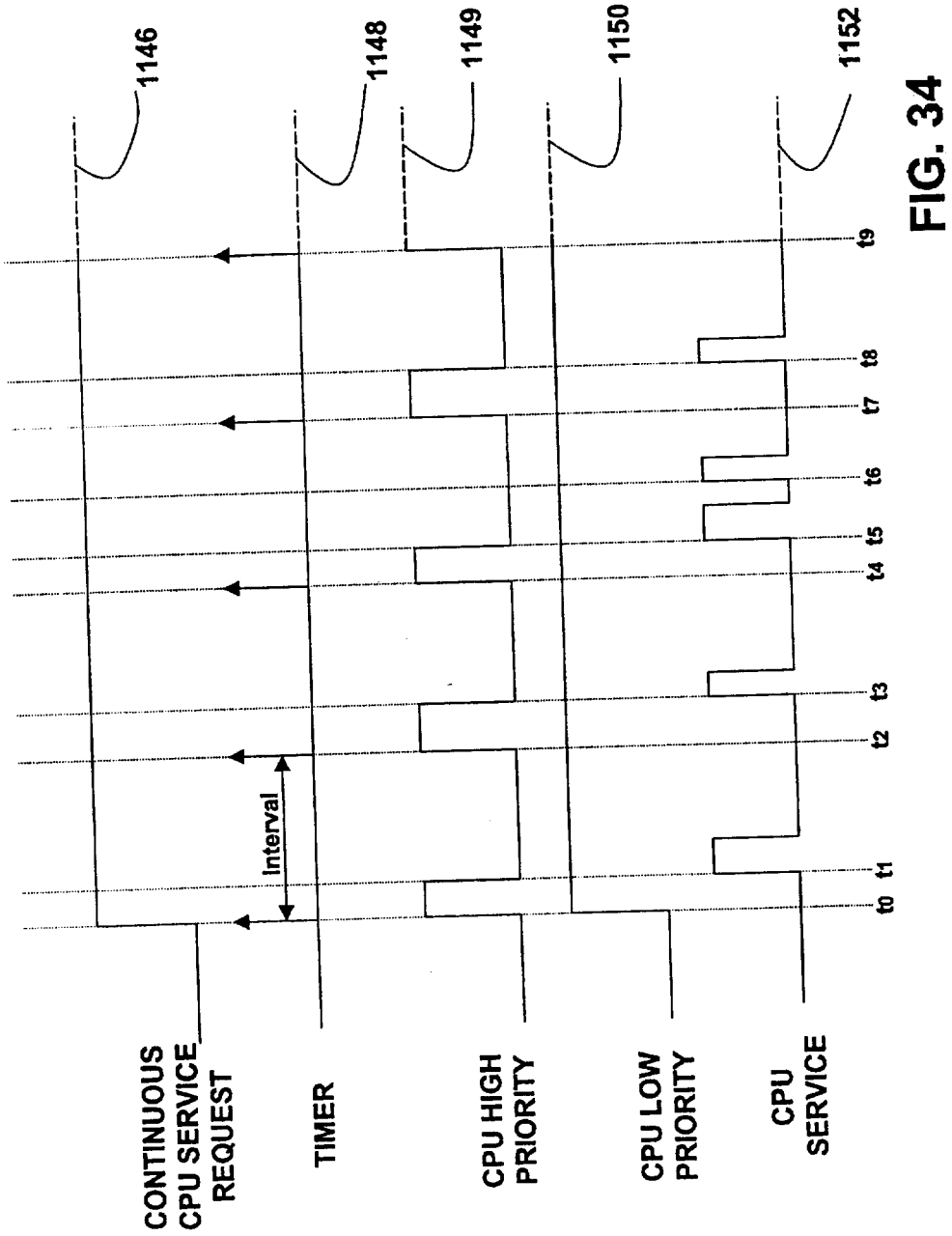

1146
1148
1149
1150
1152
t9
t8
t7
t6
t5
t4
t3
t2
t1
t0
Interval
CONTINUOUS CPU SERVICE REQUEST
TIMER
CPU HIGH PRIORITY
CPU LOW PRIORITY
CPU SERVICE

FIG. 34

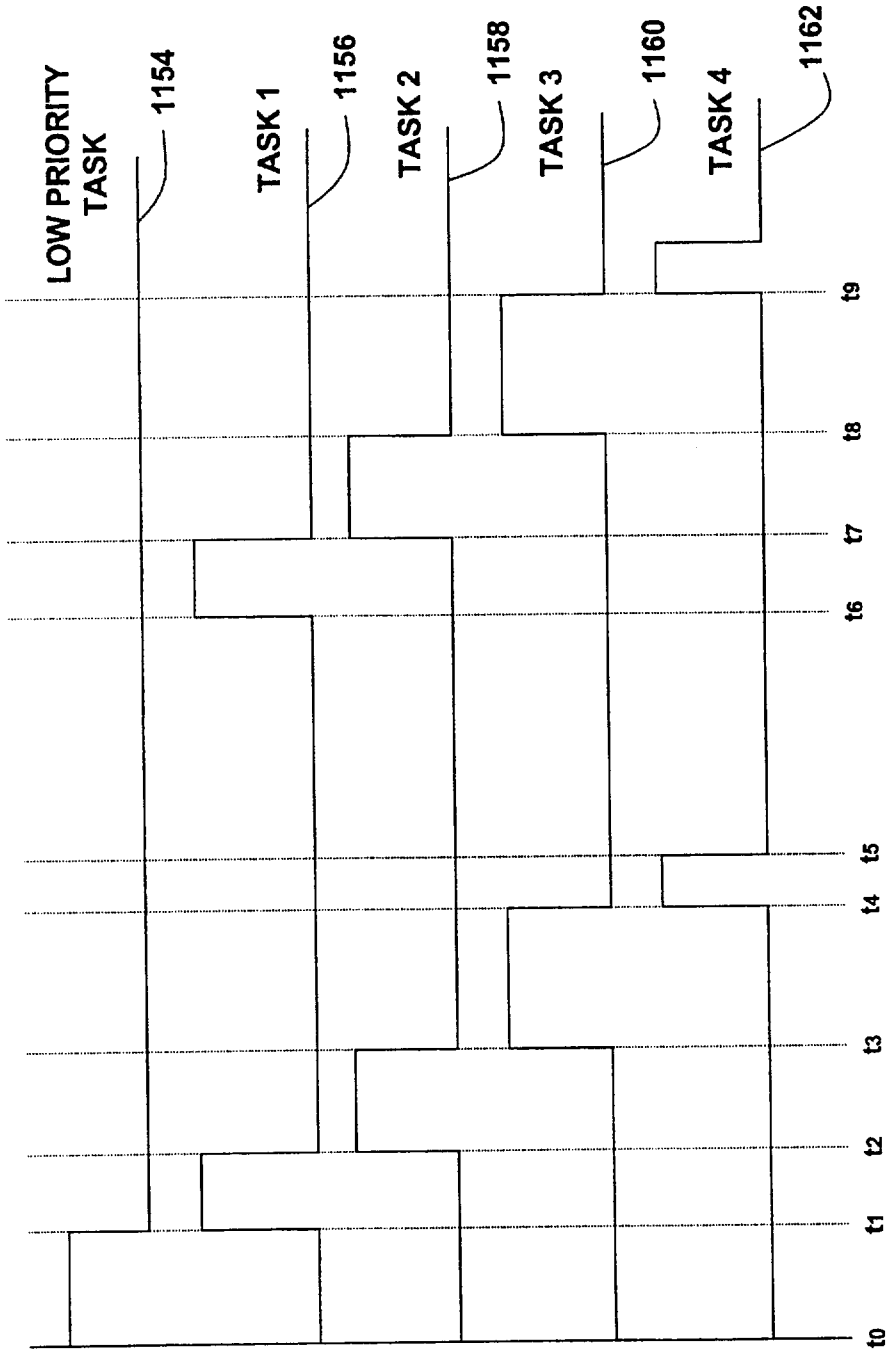

LOW PRIORITY TASK
1154
TASK 1
1156
TASK 2
1158
TASK 3
1160
TASK 4
1162
t9
t8
t7
t6
t5
t4
t3
t2
t1
t0

FIG. 35

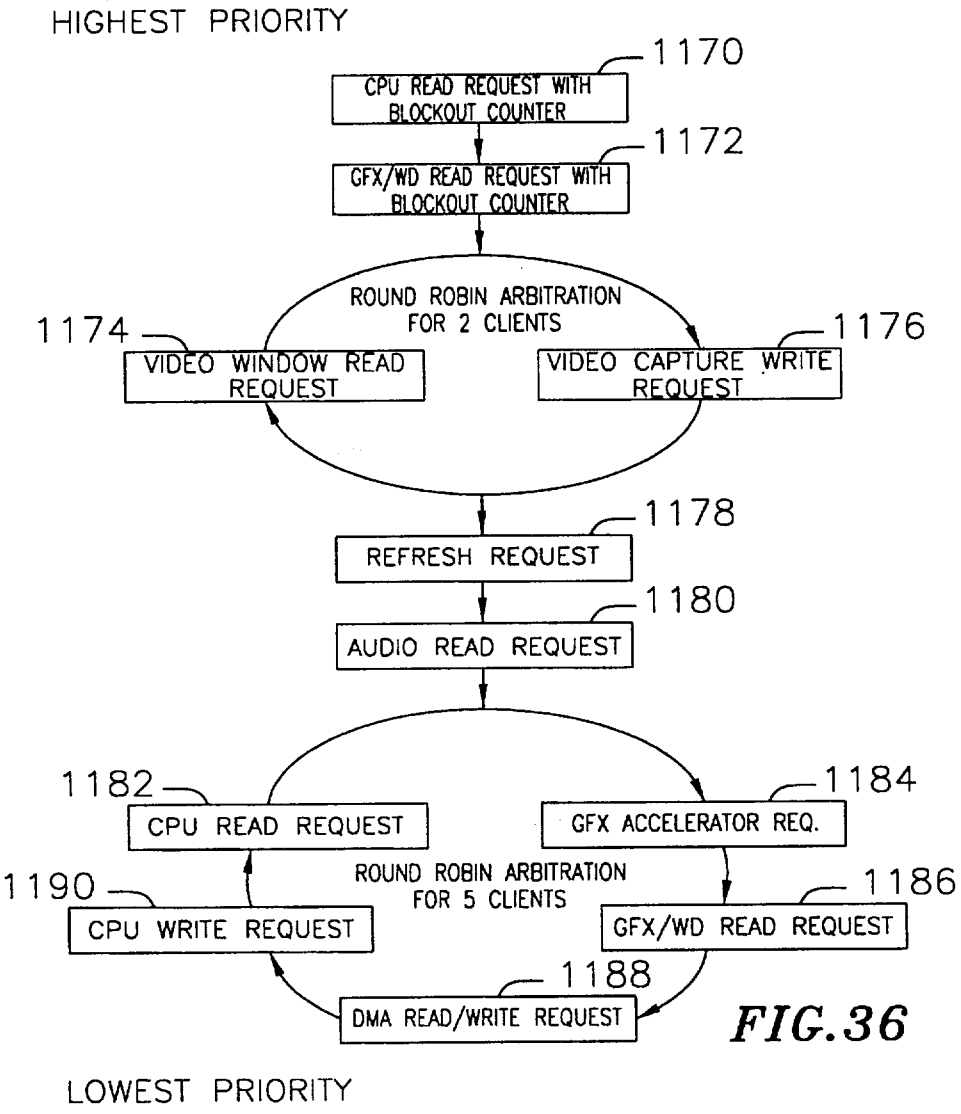

HIGHEST PRIORITY
1170
CPU READ REQUEST WITH BLOCKOUT COUNTER
1172
GFX/WD READ REQUEST WITH BLOCKOUT COUNTER
ROUND ROBIN ARBITRATION FOR 2 CLIENTS
1174
VIDEO WINDOW READ REQUEST
1176
VIDEO CAPTURE WRITE REQUEST
1178
REFRESH REQUEST
1180
AUDIO READ REQUEST
1182
CPU READ REQUEST
1184
GFX ACCELERATOR REQ.
ROUND ROBIN ARBITRATION FOR 5 CLIENTS
1190
CPU WRITE REQUEST
1186
GFX/WD READ REQUEST
1188
DMA READ/WRITE REQUEST
LOWEST PRIORITY

FIG.36

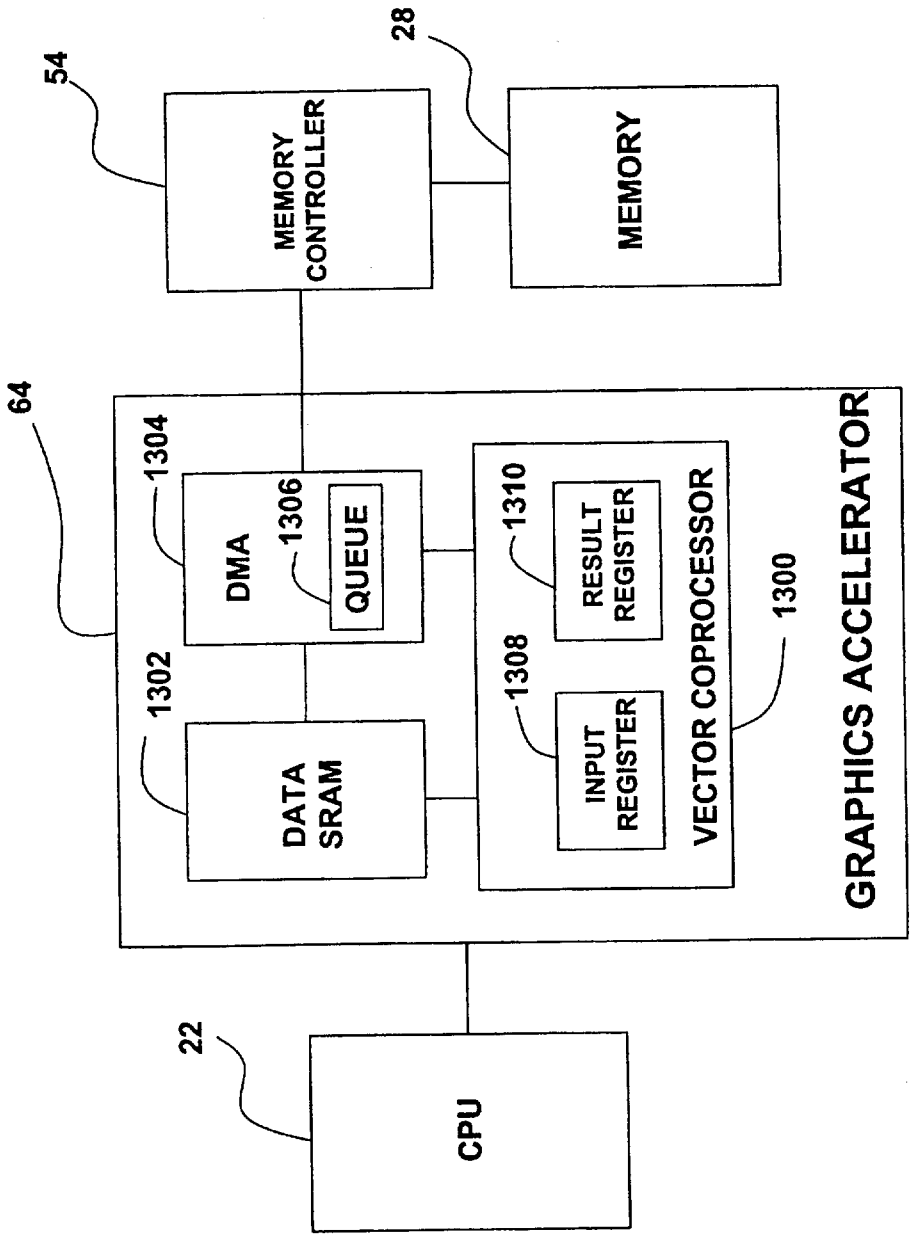

54
MEMORY CONTROLLER
28
MEMORY
64
GRAPHICS ACCELERATOR
1304
DMA
1306
QUEUE
1310
RESULT REGISTER
1308
INPUT REGISTER
VECTOR COPROCESSOR
1300
1302
DATA SRAM
22
CPU

FIG. 37

GRAPHICS DISPLAY SYSTEM WITH UNIFIED MEMORY ARCHITECTURE

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the benefit of the filing date of U.S. provisional patent application number 60/107,875, filed Nov. 9, 1998 and entitled "Graphics Chip Architecture," the contents of which are hereby incorporated by reference. This application is related to U.S. patent application Ser. No. 09/437/208, filed Nov. 9, 1999 and entitled "Graphics Display System," the contents of which are hereby incorporated by reference.

The present application contains subject matter related to the subject matter disclosed in U.S. patent applications entitled "Graphics Display System" (App. Ser. No. 09/437,208), "Graphics Display System with Graphics Window Control Mechanism" (App. Ser. No. 09/437,581), "Graphics Display System with Color Look-Up Table Loading Mechanism" (App. Ser. No. 09/437,206), "Graphics Display System with Line Buffer Control Scheme" (App. Ser. No. 09/437,325), "Graphics Display System with Window Soft Horizontal Scrolling Mechanism" (App. Ser. No. 09/437,580), "Graphics Display System with Window Descriptors" (App. Ser. No. 09/437,716), "Graphics Display System with Anti-Aliased Text and Graphics Feature" (App. Ser. No. 09/437,205), "Graphics Display System with Video Synchronization Feature" (App. Ser. No. 09/437,207), "Graphics Display System with Video Scaler" (App. Ser. No. 09/437,326), "Apparatus and Method for Blending Graphics and Video Surfaces" (App. Ser. No. 09/437,348), "Graphics Display System with Anti-Flutter Filtering and Vertical Scaling Feature" (App. Ser. No. 09/437,327), and "Graphics Accelerator" (App. Ser. No. 09/437,579), all filed Nov. 9, 1999.

FIELD OF THE INVENTION

The present invention relates generally to integrated circuits, and more particularly to an integrated circuit graphics display system.

BACKGROUND OF THE INVENTION

Graphics display systems are typically used in television control electronics, such as set top boxes, integrated digital TVs, and home network computers. Graphics display systems typically include a display engine that may perform display functions. The display engine is the part of the graphics display system that receives display pixel data from any combination of locally attached video and graphics input ports, processes the data in some way, and produces final display pixels as output.

This application includes references to both graphics and video, which reflects in certain ways the structure of the hardware itself. This split does not, however, imply the existence of any fundamental difference between graphics and video, and in fact much of the functionality is common to both. Graphics as used herein may include graphics, text and video.

SUMMARY OF THE INVENTION

The present invention provides a unified memory system including a memory that is shared by a plurality of devices. The system includes a memory request arbiter coupled to the memory. The memory request arbiter performs real time scheduling of memory requests from different devices having different priorities, and assures real time scheduling of tasks, some of which do not inherently have pre-determined periodic behavior. The arbiter provides access to memory by requesters that are sensitive to latency and do not have determinable periodic behavior.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of an integrated circuit graphics display system according to a presently preferred embodiment of the invention;

FIG. **2** is a block diagram of certain functional blocks of the system;

FIG. **3** is a block diagram of an alternate embodiment of the system of FIG. **2** that incorporates an on-chip I/O bus;

FIG. **4** is a functional block diagram of exemplary video and graphics display pipelines;

FIG. **5** is a more detailed block diagram of the graphics and video pipelines of the system;

FIG. **6** is a map of an exemplary window descriptor for describing graphics windows and solid surfaces;

FIG. **7** is a flow diagram of an exemplary process for sorting window descriptors in a window controller;

FIG. **8** is a flow diagram of a graphics window control data passing mechanism and a color look-up table loading mechanism;

FIG. **9** is a state diagram of a state machine in a graphics converter that may be used during processing of header packets;

FIG. **10** is a block diagram of an embodiment of a display engine;

FIG. **11** is a block diagram of an embodiment of a color look-up table (CLUT);

FIG. **12** is a timing diagram of signals that may be used to load a CLUT;

FIG. **13** is a block diagram illustrating exemplary graphics line buffers;

FIG. **14** is a flow diagram of a system for controlling the graphics line buffers of FIG. **13**;

FIG. **15** is a representation of left scrolling using a window soft horizontal scrolling mechanism;

FIG. **16** is a representation of right scrolling using a window soft horizontal scrolling mechanism;

FIG. **17** is a flow diagram illustrating a system that uses graphics elements or glyphs for anti-aliased text and graphics applications;

FIG. **18** is a block diagram of certain functional blocks of a video decoder for performing video synchronization;

FIG. **19** is a block diagram of an embodiment of a chroma-locked sample rate converter (SRC);

FIG. **20** is a block diagram of an alternate embodiment of the chroma-locked SRC of FIG. **19**;

FIG. **21** is a block diagram of an exemplary line-locked SRC;

FIG. **22** is a block diagram of an exemplary time base corrector (TBC);

FIG. **23** is a flow diagram of a process that employs a TBC to synchronize an input video to a display clock;

FIG. **24** is a flow diagram of a process for video scaling in which downscaling is performed prior to capture of video in memory and upscaling is performed after reading video data out of memory;

FIG. **25** is a detailed block diagram of components used during video scaling with signal paths involved in downscaling;

FIG. **26** is a detailed block diagram of components used during video scaling with signal paths involved in upscaling;

FIG. **27** is a detailed block diagram of components that may be used during video scaling with signal paths indicated for both upscaling and downscaling;

FIG. **28** is a flow diagram of an exemplary process for blending graphics and video surfaces;

FIG. **29** is a flow diagram of an exemplary process for blending graphics windows into a combined blended graphics output;

FIG. **30** is a flow diagram of an exemplary process for blending graphics, video and background color;

FIG. **31** is a block diagram of a polyphase filter that performs both anti-flutter filtering and vertical scaling of graphics windows;

FIG. **32** is a functional block diagram of an exemplary memory service request and handling system with dual memory controllers;

FIG. **33** is a functional block diagram of an implementation of a real time scheduling system;

FIG. **34** is a timing diagram of an exemplary CPU servicing mechanism that has been implemented using real time scheduling;

FIG. **35** is a timing diagram that illustrates certain principles of critical instant analysis for an implementation of real time scheduling;

FIG. **36** is a flow diagram illustrating servicing of requests according to the priority of the task; and

FIG. **37** is a block diagram of a graphics accelerator, which may be coupled to a CPU and a memory controller.

DETAILED DESCRIPTION OF A PRESENTLY PREFERRED EMBODIMENT

1. Graphics Display System Architecture

Referring to FIG. **1**, the graphics display system according to the present invention is preferably contained in an integrated circuit **10**. The integrated circuit may include inputs **12** for receiving video signals **14**, a bus **20** for connecting to a CPU **22**, a bus **24** for transferring data to and from memory **28**, and an output **30** for providing a video output signal **32**. The system may further include an input **26** for receiving audio input **34** and an output **27** for providing audio output **36**.

The graphic display system accepts video input signals that may include analog video signals, digital video signals, or both. The analog signals may be, for example, NTSC, PAL and SECAM signals or any other conventional type of analog signal. The digital signals may be in the form of decoded MPEG signals or other format of digital video. In an alternate embodiment, the system includes an on-chip decoder for decoding the MPEG or other digital video signals input to the system. Graphics data for display is produced by any suitable graphics library software, such as Direct Draw marketed by Microsoft Corporation, and is read from the CPU **22** into the memory **28**. The video output signals **32** may be analog signals, such as composite NTSC, PAL, Y/C (S-video), SECAM or other signals that include video and graphics information. In an alternate embodiment, the system provides serial digital video output to an on-chip or off-chip serializer that may encrypt the output.

The graphics display system memory **28** is preferably a unified synchronous dynamic random access memory (SDRAM) that is shared by the system, the CPU **22** and other peripheral components. In the preferred embodiment the CPU uses the unified memory for its code and data while the graphics display system performs all graphics, video and audio functions assigned to it by software. The amount of memory and CPU performance are preferably tunable by the system designer for the desired mix of performance and memory cost. In the preferred embodiment, a set-top box is implemented with SDRAM that supports both the CPU and graphics.

Referring to FIG. **2**, the graphics display system preferably includes a video decoder **50**, video scaler **52**, memory controller **54**, window controller **56**, display engine **58**, video compositor **60**, and video encoder **62**. The system may optionally include a graphics accelerator **64** and an audio engine **66**. The system may display graphics, passthrough video, scaled video or a combination of the different types of video and graphics. Passthrough video includes digital or analog video that is not captured in memory. The passthrough video may be selected from the analog video or the digital video by a multiplexer. Bypass video, which may come into the chip on a separate input, includes analog video that is digitized off-chip into conventional YUV (luma chroma) format by any suitable decoder, such as the BT829 decoder, available from Brooktree Corporation, San Diego, Calif. The YUV format may also be referred to as YCrCb format where Cr and Cb are equivalent to U and V, respectively.

The video decoder (VDEC) **50** preferably digitizes and processes analog input video to produce internal YUV component signals with separated luma and chroma components. In an alternate embodiment, the digitized signals may be processed in another format, such as RGB. The VDEC **50** preferably includes a sample rate converter **70** and a time base corrector **72** that together allow the system to receive non-standard video signals, such as signals from a VCR. The time base corrector **72** enables the video encoder to work in passthrough mode, and corrects digitized analog video in the time domain to reduce or prevent jitter.

The video scaler **52** may perform both downscaling and upscaling of digital video and analog video as needed. In the preferred embodiment, scale factors may be adjusted continuously from a scale factor of much less than one to a scale factor of four. With both analog and digital video input, either one may be scaled while the other is displayed full size at the same time as passthrough video. Any portion of the input may be the source for video scaling. To conserve memory and bandwidth, the video scaler preferably downscales before capturing video frames to memory, and upscales after reading from memory, but preferably does not perform both upscaling and downscaling at the same time.

The memory controller **54** preferably reads and writes video and graphics data to and from memory by using burst accesses with burst lengths that may be assigned to each task. The memory is any suitable memory such as SDRAM. In the preferred embodiment, the memory controller includes two substantially similar SDRAM controllers, one primarily for the CPU and the other primarily for the graphics display system, while either controller may be used for any and all of these functions.

The graphics display system preferably processes graphics data using logical windows, also referred to as viewports, surfaces, sprites, or canvasses, that may overlap or cover one another with arbitrary spatial relationships. Each window is preferably independent of the others. The windows may consist of any combination of image content, including anti-aliased text and graphics, patterns, GIF images, JPEG

images, live video from MPEG or analog video, three dimensional graphics, cursors or pointers, control panels, menus, tickers, or any other content, all or some of which may be animated.

Graphics windows are preferably characterized by window descriptors. Window descriptors are data structures that describe one or more parameters of the graphics window. Window descriptors may include, for example, image pixel format, pixel color type, alpha blend factor, location on the screen, address in memory, depth order on the screen, or other parameters. The system preferably supports a wide variety of pixel formats, including RGB 16, RGB 15, YUV 4:2:2 (ITU-R 601), CLUT2, CLUT4, CLUT8 or others. In addition to each window having its own alpha blend factor, each pixel in the preferred embodiment has its own alpha value. In the preferred embodiment, window descriptors are not used for video windows. Instead, parameters for video windows, such as memory start address and window size are stored in registers associated with the video compositor.

In operation, the window controller **56** preferably manages both the video and graphics display pipelines. The window controller preferably accesses graphics window descriptors in memory through a direct memory access (DMA) engine **76**. The window controller may sort the window descriptors according to the relative depth of their corresponding windows on the display. For graphics windows, the window controller preferably sends header information to the display engine at the beginning of each window on each scan line, and sends window header packets to the display engine as needed to display a window. For video, the window controller preferably coordinates capture of non-passthrough video into memory, and transfer of video between memory and the video compositor.

The display engine **58** preferably takes graphics information from memory and processes it for display. The display engine preferably converts the various formats of graphics data in the graphics windows into YUV component format, and blends the graphics windows to create blended graphics output having a composite alpha value that is based on alpha values for individual graphics windows, alpha values per pixel, or both. In the preferred embodiment, the display engine transfers the processed graphics information to memory buffers that are configured as line buffers. In an alternate embodiment, the buffer may include a frame buffer. In another alternate embodiment, the output of the display engine is transferred directly to a display or output block without being transferred to memory buffers.

The video compositor **60** receives one or more types of data, such as blended graphics data, video window data, passthrough video data and background color data, and produces a blended video output. The video encoder **62** encodes the blended video output from the video compositor into any suitable display format such as composite NTSC, PAL, Y/C (S-video), SECAM or other signals that may include video information, graphics information, or a combination of video and graphics information. In an alternate embodiment, the video encoder converts the blended video output of the video compositor into serial digital video output using an on-chip or off chip serializer that may encrypt the output.

The graphics accelerator **64** preferably performs graphics operations that may require intensive CPU processing, such as operations on three dimensional graphics images. The graphics accelerator may be programmable. The audio engine **66** preferably supports applications that create and play audio locally within a set-top box and allow mixing of the locally created audio with audio from a digital audio source, such as MPEG or Dolby, and with digitized analog audio. The audio engine also preferably supports applications that capture digitized baseband audio via an audio capture port and store sounds in memory for later use, or that store audio to memory for temporary buffering in order to delay the audio for precise lip-syncing when frame-based video time correction is enabled.

Referring to FIG. **3**, in an alternate embodiment of the present invention, the graphics display system further includes an I/O bus **74** connected between the CPU **22**, memory **28** and one or more of a wide variety of peripheral devices, such as flash memory, ROM, MPEG decoders, cable modems or other devices. The on-chip I/O bus **74** of the present invention preferably eliminates the need for a separate interface connection, sometimes referred in the art to as a north bridge. The I/O bus preferably provides high speed access and data transfers between the CPU, the memory and the peripheral devices, and may be used to support the full complement of devices that may be used in a full featured set-top box or digital TV. In the preferred embodiment, the I/O bus is compatible with the 68000 bus definition, including both active DSACK and passive DSACK (e.g., ROM/flash devices), and it supports external bus masters and retry operations as both master and slave. The bus preferably supports any mix of 32-bit, 16-bit and 8-bit devices, and operates at a clock rate of 33 MHz. The clock rate is preferably asynchronous with (not synchronized with) the CPU clock to enable independent optimization of those subsystems.

Referring to FIG. **4**, the graphics display system generally includes a graphics display pipeline **80** and a video display pipeline **82**. The graphics display pipeline preferably contains functional blocks, including window control block **84**, DMA (direct memory access) block **86**, FIFO (first-in-first-out memory) block **88**, graphics converter block **90**, color look up table (CLUT) block **92**, graphics blending block **94**, static random access memory (SRAM) block **96**, and filtering block **98**. The system preferably spatially processes the graphics data independently of the video data prior to blending.

In operation, the window control block **84** obtains and stores graphics window descriptors from memory and uses the window descriptors to control the operation of the other blocks in the graphics display pipeline. The windows may be processed in any order. In the preferred embodiment, on each scan line, the system processes windows one at a time from back to front and from the left edge to the right edge of the window before proceeding to the next window. In an alternate embodiment, two or more graphics windows may be processed in parallel. In the parallel implementation, it is possible for all of the windows to be processed at once, with the entire scan line being processed left to right. Any number of other combinations may also be implemented, such as processing a set of windows at a lower level in parallel, left to right, followed by the processing of another set of windows in parallel at a higher level.

The DMA block **86** retrieves data from memory **110** as needed to construct the various graphics windows according to addressing information provided by the window control block. Once the display of a window begins, the DMA block preferably retains any parameters that may be needed to continue to read required data from memory. Such parameters may include, for example, the current read address, the address of the start of the next lines, the number of bytes to read per line, and the pitch. Since the pipeline preferably includes a vertical filter block for anti-flutter and scaling

purposes, the DMA block preferably accesses a set of adjacent display lines in the same frame, in both fields. If the output of the system is NTSC or other form of interlaced video, the DMA preferably accesses both fields of the interlaced final display under certain conditions, such as when the vertical filter and scaling are enabled. In such a case, all lines, not just those from the current display field, are preferably read from memory and processed during every display field. In this embodiment, the effective rate of reading and processing graphics is equivalent to that of a non-interlaced display with a frame rate equal to the field rate of the interlaced display.

The FIFO block **88** temporarily stores data read from the memory **110** by the DMA block **86**, and provides the data on demand to the graphics converter block **90**. The FIFO may also serve to bridge a boundary between different clock domains in the event that the memory and DMA operate under a clock frequency or phase that differs from the graphics converter block **90** and the graphics blending block **94**. In an alternate embodiment, the FIFO block is not needed. The FIFO block may be unnecessary, for example, if the graphics converter block processes data from memory at the rate that it is read from the memory and the memory and conversion functions are in the same clock domain.

In the preferred embodiment, the graphics converter block **90** takes raw graphics data from the FIFO block and converts it to YUValpha (YUVa) format. Raw graphics data may include graphics data from memory that has not yet been processed by the display engine. One type of YUVa format that the system may use includes YUV 4:2:2 (i.e. two U and V samples for every four Y samples) plus an 8-bit alpha value for every pixel, which occupies overall 24 bits per pixel. Another suitable type of YUVa format includes YUV 4:4:4 plus the 8-bit alpha value per pixel, which occupies 32 bits per pixel. In an alternate embodiment, the graphics converter may convert the raw graphics data into a different format, such as RGBalpha.

The alpha value included in the YUVa output may depend on a number of factors, including alpha from chroma keying in which a transparent pixel has an alpha equal to zero, alpha per CLUT entry, alpha from Y (luma), or alpha per window where one alpha value characterizes all of the contents of a given window.

The graphics converter block **90** preferably accesses the CLUT **92** during conversion of CLUT formatted raw graphics data. In one embodiment of the present invention, there is only one CLUT. In an alternate embodiment, multiple CLUTs are used to process different graphics windows having graphics data with different CLUT formats. The CLUT may be rewritten by retrieving new CLUT data via the DMA block when required. In practice, it typically takes longer to rewrite the CLUT than the time available in a horizontal blanking interval, so the system preferably allows one horizontal line period to change the CLUT. Non-CLUT images may be displayed while the CLUT is being changed. The color space of the entries in the CLUT is preferably in YUV but may also be implemented in RGB.

The graphics blending block **94** receives output from the graphics converter block **90** and preferably blends one window at a time along the entire width of one scan line, with the back-most graphics window being processed first. The blending block uses the output from the converter block to modify the contents of the SRAM **96**. The result of each pixel blend operation is a pixel in the SRAM that consists of the weighted sum of the various graphics layers up to and including the present one, and the appropriate alpha blend value for the video layers, taking into account the graphics layers up to and including the present one.

The SRAM **96** is preferably configured as a set of graphics line buffers, where each line buffer corresponds to a single display line. The blending of graphics windows is preferably performed one graphics window at a time on the display line that is currently being composited into a line buffer. Once the display line in a line buffer has been completely composited so that all the graphics windows on that display line have been blended, the line buffer is made available to the filtering block **98**.

The filtering block **98** preferably performs both anti-flutter filtering (AFF) and vertical sample rate conversion (SRC) using the same filter. This block takes input from the line buffers and performs finite impulse response polyphase filtering on the data. While anti-flutter filtering and vertical axis SRC are done in the vertical axis, there may be different functions, such as horizontal SRC or scaling that are performed in the horizontal axis. In the preferred embodiment, the filter takes input from only vertically adjacent pixels at one time. It multiplies each input pixel times a specified coefficient, and sums the result to produce the output. The polyphase action means that the coefficients, which are samples of an approximately continuous impulse response, may be selected from a different fractional-pixel phase of the impulse response every pixel. In an alternate embodiment, where the filter performs horizontal scaling, appropriate coefficients are selected for a finite impulse response polyphase filter to perform the horizontal scaling. In an alternate embodiment, both horizontal and vertical filtering and scaling can be performed.

The video display pipeline **82** may include a FIFO block **100**, an SRAM block **102**, and a video scaler **104**. The video display pipeline portion of the architecture is similar to that of the graphics display pipeline, and it shares some elements with it. In the preferred embodiment, the video pipeline supports up to one scaled video window per scan line, one passthrough video window, and one background color, all of which are logically behind the set of graphics windows. The order of these windows, from back to front, is preferably fixed as background color, then passthrough video, then scaled video.

The video windows are preferably in YUV format, although they may be in either 4:2:2 or 4:2:0 variants or other variants of YUV, or alternatively in other formats such as RGB. The scaled video window may be scaled up in both directions by the display engine, with a factor that can range up to four in the preferred embodiment. Unlike graphics, the system generally does not have to correct for square pixel aspect ratio with video. The scaled video window may be alpha blended into passthrough video and a background color, preferably using a constant alpha value for each video signal.

The FIFO block **100** temporarily stores captured video windows for transfer to the video scaler **104**. The video scaler preferably includes a filter that performs both upscaling and downscaling. The scaler function may be a set of two polyphase SRC functions, one for each dimension. The vertical SRC may be a four-tap filter with programmable coefficients in a fashion similar to the vertical filter in the graphics pipeline, and the horizontal filter may use an 8-tap SRC, also with programmable coefficients. In an alternate embodiment, a shorter horizontal filter is used, such as a 4-tap horizontal SRC for the video upscaler. Since the same filter is preferably used for downscaling, it may be desirable to use more taps than are strictly needed for upscaling to accommodate low pass filtering for higher quality downscaling.

In the preferred embodiment, the video pipeline uses a separate window controller and DMA. In an alternate embodiment, these elements may be shared. The FIFOs are logically separate but may be implemented in a common SRAM.

The video compositor block **108** blends the output of the graphics display pipeline, the video display pipeline, and passthrough video. The background color is preferably blended as the lowest layer on the display, followed by passthrough video, the video window and blended graphics. In the preferred embodiment, the video compositor composites windows directly to the screen line-by-line at the time the screen is displayed, thereby conserving memory and bandwidth. The video compositor may include, but preferably does not include, display frame buffers, double-buffered displays, off-screen bit maps, or blitters.

Referring to FIG. **5**, the display engine **58** preferably includes graphics FIFO **132**, graphics converter **134**, RGB-to-YUV converter **136**, YUV-**444**-to-YUV-**422** converter **138** and graphics blender **140**. The graphics FIFO **132** receives raw graphics data from memory through a graphics DMA **124** and passes it to the graphics converter **134**, which preferably converts the raw graphics data into YUV 4:4:4 format or other suitable format. A window controller **122** controls the transfer of raw graphics data from memory to the graphics converter **132**. The graphics converter preferably accesses the RGB-to-YUV converter **136** during conversion of RGB formatted data and the graphics CLUT **146** during conversion of CLUT formatted data. The RGB-to-YUV converter is preferably a color space converter that converts raw graphics data in RGB space to graphics data in YUV space. The graphics CLUT **146** preferably includes a CLUT **150**, which stores pixel values for CLUT-formatted graphics data, and a CLUT controller **152**, which controls operation of the CLUT.

The YUV444-to-YUV422 converter **138** converts graphics data from YUV 4:4:4 format to YUV 4:2:2 format. The term YUV 4:4:4 means, as is conventional, that for every four horizontally adjacent samples, there are four Y values, four U values, and four V values; the term YUV 4:2:2 means, as is conventional, that for every four samples, there are four Y values, two U values and two V values. The YUV444-to-YUV422 converter **138** is preferably a UV decimator that sub-samples U and V from four samples per every four samples of Y to two samples per every four samples of Y.

Graphics data in YUV 4:4:4 format and YUV 4:2:2 format preferably also includes four alpha values for every four samples. Graphics data in YUV 4:4:4 format with four alpha values for every four samples may be referred to as being in aYUV 4:4:4:4 format; graphics data in YUV 4:2:2 format with four alpha values for every four samples may be referred to as being in aYUV 4:4:2:2 format.

The YUV444-to-YUV422 converter may also perform low-pass filtering of UV and alpha. For example, if the graphics data with YUV 4:4:4 format has higher than desired frequency content, a low pass filter in the YUV444-to-YUV422 converter may be turned on to filter out high frequency components in the U and V signals, and to perform matched filtering of the alpha values.

The graphics blender **140** blends the YUV 4:2:2 signals together, preferably one line at a time using alpha blending, to create a single line of graphics from all of the graphics windows on the current display line. The filter **170** preferably includes a single 4-tap vertical polyphase graphics filter **172**, and a vertical coefficient memory **174**. The graphics filter may perform both anti-flutter filtering and vertical scaling. The filter preferably receives graphics data from the display engine through a set of seven line buffers **59**, where four of the seven line buffers preferably provide data to the taps of the graphics filter at any given time.

In the preferred embodiment, the system may receive video input that includes one decoded MPEG video in ITU-R 656 format and one analog video signal. The ITU-R 656 decoder **160** processes the decoded MPEG video to extract timing and data information. In one embodiment, an on-chip video decoder (VDEC) **50** converts the analog video signal to a digitized video signal. In an alternate embodiment, an external VDEC such as the Brooktree BT829 decoder converts the analog video into digitized analog video and provides the digitized video to the system as bypass video **130**.

Analog video or MPEG video may be provided to the video compositor as passthrough video. Alternatively, either type of video may be captured into memory and provided to the video compositor as a scaled video window. The digitized analog video signals preferably have a pixel sample rate of 13.5 MHz, contain a 16 bit data stream in YUV 4:2:2 format, and include timing signals such as top field and vertical sync signals.

The VDEC **50** includes a time base corrector (TBC) **72** comprising a TBC controller **164** and a FIFO **166**. To provide passthrough video that is synchronized to a display clock preferably without using a frame buffer, the digitized analog video is corrected in the time domain in the TBC **72** before being blended with other graphics and video sources. During time base correction, the video input which runs nominally at 13.5 MHZ is synchronized with the display clock which runs nominally at 13.5 MHZ at the output; these two frequencies that are both nominally 13.5 MHz are not necessarily exactly the same frequency. In the TBC, the video output is preferably offset from the video input by a half scan line per field.

A capture FIFO **158** and a capture DMA **154** preferably capture the digitized analog video signals and MPEG video. The SDRAM controller **126** provides captured video frames to the external SDRAM. A video DMA **144** transfers the captured video frames to a video FIFO **148** from the external SDRAM.

The digitized analog video signals and MPEG video are preferably scaled down to less than 100% prior to being captured and are scaled up to more than 100% after being captured. The video scaler **52** is shared by both upscale and downscale operations. The video scaler preferably includes a multiplexer **176**, a set of line buffers **178**, a horizontal and vertical coefficient memory **180** and a scaler engine **182**. The scaler engine **182** preferably includes a set of two polyphase filters, one for each of horizontal and vertical dimensions.

The vertical filter preferably includes a four-tap filter with programmable filter coefficients. The horizontal filter preferably includes an eight-tap filter with programmable filter coefficients. In the preferred embodiment, three line buffers **178** supply video signals to the scaler engine **182**. The three line buffers **178** preferably are 720×16 two port SRAM. For vertical filtering, the three line buffers **178** may provide video signals to three of the four taps of the four-tap vertical filter while the video input provides the video signal directly to the fourth tap. For horizontal filtering, a shift register having eight cells in series may be used to provide inputs to the eight taps of the horizontal polyphase filter, each cell providing an input to one of the eight taps.

For downscaling, the multiplexer **168** preferably provides a video signal to the video scaler prior to capture. For

upscaling, the video FIFO **148** provides a video signal to the video scaler after capture. Since the video scaler **52** is shared between downscaling and upscaling filtering, downscaling and upscaling operations are not performed at the same time in this particular embodiment.

In the preferred embodiment, the video compositor **60** blends signals from up to four different sources, which may include blended graphics from the filter **170**, video from a video FIFO **148**, passthrough video from a multiplexer **168**, and background color from a background color module **184**. Alternatively, various numbers of signals may be composited, including, for example, two or more video windows. The video compositor preferably provides final output signal to the data size converter **190**, which serializes the 16-bit word sample into an 8-bit word sample at twice the clock frequency, and provides the 8-bit word sample to the video encoder **62**.

The video encoder **62** encodes the provided YUV 4:2:2 video data and outputs it as an output of the graphics display system in any desired analog or digital format.

II. Window Descriptor and Solid Surface Description

Often in the creation of graphics displays, the artist or application developer has a need to include rectangular objects on the screen, with the objects having a solid color and a uniform alpha blend factor (alpha value). These regions (or objects) may be rendered with other displayed objects on top of them or beneath them. In conventional graphics devices, such solid color objects are rendered using the number of distinct pixels required to fill the region. It may be advantageous in terms of memory size and memory bandwidth to render such objects on the display directly, without expending the memory size or bandwidth required in conventional approaches.

In the preferred embodiment, video and graphics are displayed on regions referred to as windows. Each window is preferably a rectangular area of screen bounded by starting and ending display lines and starting and ending pixels on each display line. Raw graphics data to be processed and displayed on a screen preferably resides in the external memory. In the preferred embodiment, a display engine converts raw graphics data into a pixel map with a format that is suitable for display.

In one embodiment of the present invention, the display engine implements graphics windows of many types directly in hardware. Each of the graphics windows on the screen has its own value of various parameters, such as location on the screen, starting address in memory, depth order on the screen, pixel color type, etc. The graphics windows may be displayed such that they may overlap or cover each other, with arbitrary spatial relationships.

In the preferred embodiment, a data structure called a window descriptor contains parameters that describe and control each graphics window. The window descriptors are preferably data structures for representing graphics images arranged in logical surfaces, or windows, for display. Each data structure preferably includes a field indicating the relative depth of the logical surface on the display, a field indicating the alpha value for the graphics in the surface, a field indicating the location of the logical surface on the display, and a field indicating the location in memory where graphics image data for the logical surface is stored.

All of the elements that make up any given graphics display screen are preferably specified by combining all of the window descriptors of the graphics windows that make up the screen into a window descriptor list. At every display field time or a frame time, the display engine constructs the display image from the current window descriptor list. The display engine composites all of the graphics windows in the current window descriptor list into a complete screen image in accordance with the parameters in the window descriptors and the raw graphics data associated with the graphics windows.

With the introduction of window descriptors and real-time composition of graphics windows, a graphics window with a solid color and fixed translucency may be described entirely in a window descriptor having appropriate parameters. These parameters describe the color and the translucency (alpha) just as if it were a normal graphics window. The only difference is that there is no pixel map associated with this window descriptor. The display engine generates a pixel map accordingly and performs the blending in real time when the graphics window is to be displayed.

For example, a window consisting of a rectangular object having a constant color and a constant alpha value may be created on a screen by including a window descriptor in the window descriptor list. In this case, the window descriptor indicates the color and the alpha value of the window, and a null pixel format, i.e., no pixel values are to be read from memory. Other parameters indicate the window size and location on the screen, allowing the creation of solid color windows with any size and location. Thus, in the preferred embodiment, no pixel map is required, memory bandwidth requirements are reduced and a window of any size may be displayed.

Another type of graphics window that the window descriptors preferably describe is an alpha-only type window. The alpha-only type windows preferably use a constant color and preferably have graphics data with 2, 4 or 8 bits per pixel. For example, an alpha-4 format may be an alpha-only format used in one of the alpha-only type windows. The alpha-4 format specifies the alpha-only type window with alpha blend values having four bits per pixel. The alpha-only type window may be particularly useful for displaying anti-aliased text.

A window controller preferably controls transfer of graphics display information in the window descriptors to the display engine. In one embodiment, the window controller has internal memory to store eight window descriptors. In other embodiments, the window controller may have memory allocated to store more or less window descriptors. The window controller preferably reads the window descriptors from external memory via a direct memory access (DMA) module.

The DMA module may be shared by both paths of the display pipeline as well as some of the control logic, such as the window controller and the CLUT. In order to support the display pipeline, the DMA module preferably has three channels where the graphics pipeline and the video pipeline use separate DMA modules. These may include window descriptor read, graphics data read and CLUT read. Each channel has externally accessible registers to control the start address and the number of words to read.

Once the DMA module has completed a transfer as indicated by its start and length registers, it preferably activates a signal that indicates the transfer is complete. This allows the DMA module that sets up operations for that channel to begin setting up of another transfer. In the case of graphics data reads, the window controller preferably sets up a transfer of one line of graphics pixels and then waits for the DMA controller to indicate that the transfer of that line is

complete before setting up the transfer of the next line, or of a line of another window.

Referring to FIG. 6, each window descriptor preferably includes four 32-bit words (labeled Word 0 through Word 3) containing graphics window display information. Word 0 preferably includes a window operation parameter, a window format parameter and a window memory start address. The window operation parameter preferably is a 2-bit field that indicates which operation is to be performed with the window descriptor. When the window operation parameter is 00b, the window descriptor performs a normal display operation and when it is 01b, the window descriptor performs graphics color look-up table ("CLUT") re-loading. The window operation parameter of 10b is preferably not used. The window operation parameter of 11b preferably indicates that the window descriptor is the last of a sequence of window descriptors in memory.

The window format parameter preferably is a 4-bit field that indicates a data format of the graphics data to be displayed in the graphics window. The data formats corresponding to the window format parameter is described in Table 1 below.

TABLE 1

Graphics Data Formats

win_ format	Data Format	Data Format Description
0000b	RGB16	5-BIT RED, 6-BIT GREEN, 5-BIT BLUE
0001b	RGB15+1	RGB15 plus one bit alpha (keying)
0010b	RGBA4444	4-BIT RED, GREEN, BLUE, ALPHA
0100b	CLUT2	2-bit CLUT with YUV and alpha in table
0101b	CLUT4	4-bit CLUT with YUV and alpha in table
0110b	CLUT8	8-bit CLUT with YUV and alpha in table
0111b	ACLUT16	8-BIT ALPHA, 8-BIT CLUT INDEX
1000b	ALPHA0	Single win_alpha and single RGB win_color
1001b	ALPHA2	2-bit alpha with single RGB win_color
1010b	ALPHA4	4-bit alpha with single RGB win_color
1011b	ALPHA8	8-bit alpha with single RGB win color
1100b	YUV422	U and V are sampled at half the rate of Y
1111b	RESERVED	Special coding for blank line in new header, i.e., indicates an empty line

The window memory start address preferably is a 26-bit data field that indicates a starting memory address of the graphics data of the graphics window to be displayed on the screen. The window memory start address points to the first address in the corresponding external SDRAM which is accessed to display data on the graphics window defined by the window descriptor. When the window operation parameter indicates the graphics CLUT reloading operation, the window memory start address indicates a starting memory address of data to be loaded into the graphics CLUT.

Word 1 in the window descriptor preferably includes a window layer parameter, a window memory pitch value and a window color value. The window layer parameter is preferably a 4-bit data indicating the order of layers of graphics windows. Some of the graphics windows may be partially or completely stacked on top of each other, and the window layer parameter indicates the stacking order. The window layer parameter preferably indicates where in the stack the graphics window defined by the window descriptor should be placed.

In the preferred embodiment, a graphics window with a window layer parameter of 0000b is defined as the bottom most layer, and a graphics window with a window layer parameter of 1111b is defined as the top most layer. Preferably, up to eight graphics windows may be processed in each scan line. The window memory pitch value is preferably a 12-bit data field indicating the pitch of window memory addressing. Pitch refers to the difference in memory address between two pixels that are vertically adjacent within a window.

The window color value preferably is a 16-bit RGB color, which is applied as a single color to the entire graphics window when the window format parameter is 1000b, 1001b, 1010b, or 1011b . Every pixel in the window preferably has the color specified by the window color value, while the alpha value is determined per pixel and per window as specified in the window descriptor and the pixel format. The engine preferably uses the window color value to implement a solid surface.

Word 2 in the window descriptor preferably includes an alpha type, a widow alpha value, a window y-end value and a window y-start value. The word 2 preferably also includes two bits reserved for future definition, such as high definition television (HD) applications. The alpha type is preferably a 2-bit data field that indicates the method of selecting an alpha value for the graphics window. The alpha type of 00b indicates that the alpha value is to be selected from chroma keying. Chroma keying determines whether each pixel is opaque or transparent based on the color of the pixel. Opaque pixels are preferably considered to have an alpha value of 1.0, and transparent pixels have an alpha value of 0, both on a scale of 0 to 1. Chroma keying compares the color of each pixel to a reference color or to a range of possible colors; if the pixel matches the reference color, or if its color falls within the specified range of colors, then the pixel is determined to be transparent. Otherwise it is determined to be opaque.

The alpha type of 01b indicates that the alpha value should be derived from the graphics CLUT, using the alpha value in each entry of the CLUT. The alpha type of 10b indicates that the alpha value is to be derived from the luminance Y. The Y value that results from conversion of the pixel color to the YUV color space, if the pixel color is not already in the YUV color, is used as the alpha value for the pixel. The alpha type of 11b indicates that only a single alpha value is to be applied to the entire graphics window. The single alpha value is preferably included as the window alpha value next.

The window alpha value preferably is an 8-bit alpha value applied to the entire graphics window. The effective alpha value for each pixel in the window is the product of the window alpha and the alpha value determined for each pixel. For example, if the window alpha value is 0.5 on a scale of 0 to 1, coded as 0x80, then the effective alpha value of every pixel in the window is one-half of the value encoded in or for the pixel itself. If the window format parameter is 1000b, i.e., a single alpha value is to be applied to the graphics window, then the per-pixel alpha value is treated as if it is 1.0, and the effective alpha value is equal to the window alpha value.

The window y-end value preferably is a 10-bit data field that indicates the ending display line of the graphics window on the screen. The graphics window defined by the window descriptor ends at the display line indicated by the window y-end value. The window y-start value preferably is a 10-bit data field that indicates a starting display line of the graphics window on a screen. The graphics window defined by the window descriptor begins at the display line indicated in the window y-start value. Thus, a display of a graphics window can start on any display line on the screen based on the window y-start value.

Word 3 in the window descriptor preferably includes a window filter enable parameter, a blank start pixel value, a window x-size value and a window x-start value. In addition, the word 3 includes two bits reserved for future definition, such as HD applications. Five bits of the 32-bit word 3 are not used. The window filter enable parameter is a 1-bit field that indicates whether low pass filtering is to be enabled during YUV 4:4:4 to YUV 4:2:2 conversion.

The blank start pixel value preferably is a 4-bit parameter indicating a number of blank pixels at the beginning of each display line. The blank start pixel value preferably signifies the number of pixels of the first word read from memory, at the beginning of the corresponding graphics window, to be discarded. This field indicates the number of pixels in the first word of data read from memory that are not displayed. For example, if memory words are 32 bits wide and the pixels are 4 bits each, there are 8 possible first pixels in the first word. Using this field, 0 to 7 pixels may be skipped, making the 1^{st} to the 8^{th} pixel in the word appear as the first pixel, respectively. The blank start pixel value allows graphics windows to have any horizontal starting position on the screen, and may be used during soft horizontal scrolling of a graphics window.

The window x-size value preferably is a 10-bit data field that indicates the size of a graphics window in the x direction, i.e., horizontal direction. The window x-size value preferably indicates the number of pixels of a graphics window in a display line.

The window x-start value preferably is a 10-bit data field that indicates a starting pixel of the graphics window on a display line. The graphics window defined by the window descriptor preferably begins at the pixel indicated by the window x-start value of each display line. With the window x-start value, any pixel of a given display line can be chosen to start painting the graphics window. Therefore, there is no need to load pixels on the screen prior to the beginning of the graphics window display area with black.

III. Graphics Window Control Data Passing Mechanism

In one embodiment of the present invention, a FIFO in the graphics display path accepts raw graphics data as the raw graphics data is read from memory, at the full memory data rate using a clock of the memory controller. In this embodiment, the FIFO provides this data, initially stored in an external memory, to subsequent blocks in the graphics pipeline.

In systems such as graphics display systems where multiple types of data may be output from one module, such as a memory controller subsystem, and used in another subsystem, such as a graphics processing subsystem, it typically becomes progressively more difficult to support a combination of dynamically varying data types and data transfer rates and FIFO buffers between the producing and consuming modules. The conventional way to address such problems is to design a logic block that understands the varying parameters of the data types in the first module and controls all of the relevant variables in the second module. This may be difficult due to variable delays between the two modules, due to the use of FIFOs between them and varying data rate, and due to the complexity of supporting a large number of data types.

The system preferably processes graphics images for display by organizing the graphics images into windows in which the graphics images appear on the screen, obtaining data that describes the windows, sorting the data according to the depth of the window on the display, transferring graphics images from memory, and blending the graphics images using alpha values associated with the graphics images.

In the preferred embodiment, a packet of control information called a header packet is passed from the window controller to the display engine. All of the required control information from the window controller preferably is conveyed to the display engine such that all of the relevant variables from the window controller are properly controlled in a timely fashion and such that the control is not dependent on variations in delays or data rates between the window controller and the display engine.

A header packet preferably indicates the start of graphics data for one graphics window. The graphics data for that graphics window continues until it is completed without requiring a transfer of another header packet. A new header packet is preferably placed in the FIFO when another window is to start. The header packets may be transferred according to the order of the corresponding window descriptors in the window descriptor lists.

In a display engine that operates according to lists of window descriptors, windows may be specified to overlap one another. At the same time, windows may start and end on any line, and there may be many windows visible on any one line. There are a large number of possible combinations of window starting and ending locations along vertical and horizontal axes and depth order locations. The system preferably indicates the depth order of all windows in the window descriptor listand implements the depth ordering correctly while accounting for all windows.

Each window descriptor preferably includes a parameter indicating the depth location of the associated window. The range that is allowed for this parameter can be defined to be almost any useful value. In the preferred embodiment there are 16 possible depth values, ranging from 0 to 15, with 0 being the back-most (deepest, or furthest from the viewer), and 15 being the top or front-most depth. The window descriptors are ordered in the window descriptor list in order of the first display scan line where the window appears. For example if window A spans lines 10 to 20, window B spans lines 12 to 18, and window C spans lines 5 to 20, the order of these descriptors in the list would be {C, A, B}.

In the hardware, which is a preferably a VLSI device, there is preferably on-chip memory capable of storing a number of window descriptors. In the preferred implementation, this memory can store up to 8 window descriptors on-chip, however the size of this memory may be made larger or smaller without loss of generality. Window descriptors are read from main memory into the on-chip descriptor memory in order from the start of the list, and stopping when the on-chip memory is full or when the most recently read descriptor describes a window that is not yet visible, i.e., its starting line is on a line that has a higher number than the line currently being constructed. Once a window has been displayed and is no longer visible, it may be cast out of the on-chip memory and the next descriptor in the list may read from main memory. At any given display line, the order of the window descriptors in the on-chip memory bears no particular relation to the depth order of the windows on the screen.

The hardware that controls the compositing of windows builds up the display in layers, starting from the back-most layer. In the preferred embodiment, the back most layer is layer 0. The hardware performs a quick search of the back-most window descriptor that has not yet been

composited, regardless of its location in the on-chip descriptor memory. In the preferred embodiment, this search is performed as follows:

All 8 window descriptors are stored on chip in such a way that the depth order numbers of all of them are available simultaneously. While the depth numbers in the window descriptors are 4 bit numbers, representing 0 to 15, the on-chip memory has storage for 5 bits for the depth number. Initially the 5 bit for each descriptor is set to 0. The depth order values are compared in a hierarchy of pair-wise comparisons, and the lower of the two depth numbers in each comparison wins the comparison. That is, at the first stage of the test descriptor pairs {0, 1 }, {2, 3 }, {4, 5}, and {6, 7} are compared, where (0–7) represent the eight descriptors stored in the on-chip memory. This results in four depth numbers with associated descriptor numbers. At the next stage two pair-wise comparisons compare {(0, 1), (2, 3)} and {(4, 5), (6, 7)}.

Each of these results in a depth number of the lower depth order number and the associated descriptor number. At the third stage, one pair-wise comparison finds the smallest depth number of all, and its associated descriptor number. This number points the descriptor in the on-chip memory with the lowest depth number, and therefore the greatest depth, and this descriptor is used first to render the associated window on the screen. Once this window has been rendered onto the screen for the current scan line, the fifth bit of the depth number in the on-chip memory is set to 1, thereby ensuring that the depth value number is greater than 15, and as a result this depth number will preferably never again be found to be the back-most window until all windows have been rendered on this scan line, preventing rendering this window twice.

Once all the windows have been rendered for a given scan line, the fifth bits of all the on-chip depth numbers are again set to 0; descriptors that describe windows that are no longer visible on the screen are cast out of the on-chip memory; new descriptors are read from memory as required (that is, if all windows in the on-chip memory are visible, the next descriptor is read from memory, and this repeats until the most recently read descriptor is not yet visible on the screen), and the process of finding the back most descriptor and rendering windows onto the screen repeats.

Referring to FIG. **7**, window descriptors are preferably sorted by the window controller and used to transfer graphics data to the display engine. Each of window descriptors, including the window descriptor **0** through the window descriptor **7** 300a–h, preferably contains a window layer parameter. In addition, each window descriptor is preferably associated with a window line done flag indicating that the window descriptor has been processed on a current display line.

The window controller preferably performs window sorting at each display line using the window layer parameters and the window line done flags. The window controller preferably places the graphics window that corresponds to the window descriptor with the smallest window layer parameter at the bottom, while placing the graphics window that corresponds to the window descriptor with the largest window layer parameter at the top.

The window controller preferably transfers the graphics data for the bottom-most graphics window to be processed first. The window parameters of the bottom-most window are composed into a header packet and written to the graphics FIFO. The DMA engine preferably sends a request to the memory controller to read the corresponding graphics data for this window and send the graphics data to the graphics FIFO. The graphics FIFO is then read by the display engine to compose a display line, which is then written to graphics line buffers.

The window line done flag is preferably set true whenever the window surface has been processed on the current display line. The window line done flag and the window layer parameter may be concatenated together for sorting. The window line done flag is added to the window layer parameter as the most significant bit during sorting such that {window line done flag[4], window layer parameter[3:0]} is a five bit binary number, a window layer value, with window line done flag as the most significant bit.

The window controller preferably selects a window descriptor with the smallest window layer value to be processed. Since the window line done flag is preferably the most significant bit of the window layer value, any window descriptor with this flag set, i.e., any window that has been processed on the current display line, will have a higher window layer value than any of the other window descriptors that have not yet been processed on the current display line. When a particular window descriptor is processed, the window line done flag associated with that particular window descriptor is preferably set high, signifying that the particular window descriptor has been processed for the current display line.

A sorter **304** preferably sorts all eight window descriptors after any window descriptor is processed. The sorting may be implemented using binary tree sorting or any other suitable sorting algorithm. In binary tree sorting for eight window descriptors, the window layer value for four pairs of window descriptors are compared at a first level using four comparators to choose the window descriptor that corresponds to a lower window in each pair.

In the second level, two comparators are used to select the window descriptor that corresponds to the bottom most graphics window in each of two pairs. In the third and the last level, the bottom-most graphics windows from each of the two pairs are compared against each other preferably using only one comparator to select the bottom window.

A multiplexer **302** preferably multiplexes parameters from the window descriptors. The output of the sorter, i.e., window selected to be the bottom most, is used to select the window parameters to be sent to a direct memory access ("DMA") module **306** to be packaged in a header packet and sent to a graphics FIFO **308**. The display engine preferably reads the header packet in the graphics FIFO and processes the raw graphics data based on information contained in the header packet.

The header packet preferably includes a first header word and a second header word. Corresponding graphics data is preferably transferred as graphics data words. Each of the first header word, the second header word and the graphics data words preferably includes 32 bits of information plus a data type bit. The first header word preferably includes a 1-bit data type, a 4-bit graphics type, a 1-bit first window parameter, a 1-bit top/bottom parameter, a 2-bit alpha type, an 8-bit window alpha value and a 16-bit window color value. Table 2 shows contents of the first header word.

TABLE 2

First Header Word							
Bit Position	32	31-28	27	26	25-24	23-16	15-0
Data Content	Data type	graphics type	First Window	top/ bottom	alpha type	window alpha	window color

The 1-bit data type preferably indicates whether a 33-bit word in the FIFO is a header word or a graphics data word. A data type of 1 indicates that the associated 33-bit word is a header word while the data type of 0 indicates that the associated 33-bit word is a graphics data word. The graphics type indicates the data format of the graphics data to be displayed in the graphics window similar to the window format parameter in the word 0 of the window descriptor, which is described in Table 1 above. In the preferred embodiment, when the graphics type is 1111, there is no window on the current display line, indicating that the current display line is empty.

The first window parameter of the first header word preferably indicates whether the window associated with that first header word is a first window on a new display line. The top/bottom parameter preferably indicates whether the current display line indicated in the first header word is at the top or the bottom edges of the window. The alpha type preferably indicates a method of selecting an alpha value individually for each pixel in the window similar to the alpha type in the word 2 of the window descriptor.

The window alpha value preferably is an alpha value to be applied to the window as a whole and is similar to the window alpha value in the word 2 of the window descriptor. The window color value preferably is the color of the window in 16-bit RGB format and is similar to the window color value in the word 1 of the window descriptor.

The second header word preferably includes the 1-bit data type, a 4-bit blank pixel count, a 10-bit left edge value, a 1-bit filter enable parameter and a 10-bit window size value. Table 3 shows contents of the second header word in the preferred embodiment.

TABLE 3

Second Header Word					
Bit Position	32	31-28	25-16	10	9-0
Data Content	data type	Blank pixel count	Left edge	filter enabler	window size

Similar to the first header word, the second header word preferably starts with the data type indicating whether the second header word is a header word or a graphics data word. The blank pixel count preferably indicates a number of blank pixels at a left edge of the window and is similar to the blank start pixel value in the word 3 of the window descriptor. The left edge preferably indicates a starting location of the window on a scan line, and is similar to the window x-start value in the word 3 of the window descriptor. The filter enable parameter preferably enables a filter during a conversion of graphics data from a YUV 4:4:4 format to a YUV 4:2:2 format and is similar to the window filter enable parameter in word 3 of the window descriptor. Some YUV 4:4:4 data may contain higher frequency content than others, which may be filtered by enabling a low pass filter during a conversion to the YUV 4:2:2 format. The window size value preferably indicates the actual horizontal size of the window and is similar to the window x-size value in word 3 of the window descriptor.

When the composition of the last window of the last display line is completed, an empty-line header is preferably placed into the FIFO so that the display engine may release the display line for display.

Packetized data structures have been used primarily in the communication world where large amount of data needs to be transferred between hardware using a physical data link (e.g., wires). The idea is not known to have been used in the graphics world where localized and small data control structures need to be transferred between different design entities without requiring a large off-chip memory as a buffer. In one embodiment of the present system, header packets are used, and a general-purpose FIFO is used for routing. Routing may be accomplished in a relatively simple manner in the preferred embodiment because the write port of the FIFO is the only interface.

In the preferred embodiment, the graphics FIFO is a synchronous 32x33 FIFO built with a static dual-port RAM with one read port and one write port. The write port preferably is synchronous to a 81 MHz memory clock while the read port may be asynchronous (not synchronized) to the memory clock. The read port is preferably synchronous to a graphics processing clock, which runs preferably at 81 MHz, but not necessarily synchronized to the memory clock. Two graphics FIFO pointers are preferably generated, one for the read port and one for the write port. In this embodiment, each graphics FIFO pointer is a 6-bit binary counter which ranges from 00000b to 111111b, i.e., from 0 to 63. The graphics FIFO is only 32 words deep and requires only 5 bits to represent each 33-bit word in the graphics FIFO. An extra bit is preferably used to distinguish between FIFO full and FIFO empty states.

The graphics data words preferably include the 1-bit data type and 32-bit graphics data bits. The data type is 0 for the graphics data words. In order to adhere to a common design practice that generally limits the size of a DMA burst into a FIFO to half the size of the FIFO, the number of graphics data words in one DMA burst preferably does not exceed 16.

In an alternate embodiment, a graphics display FIFO is not used. In this embodiment, the graphics converter processes data from memory at the rate that it is read from memory. The memory and conversion functions are in a same clock domain. Other suitable FIFO designs may be used.

Referring to FIG. 8, a flow diagram illustrates a process for loading and processing window descriptors. First the system is preferably reset in step 310. Then the system in step 312 preferably checks for a vertical sync ("VSYNC"). When the VSYNC is received, the system in step 314 preferably proceeds to load window descriptors into the window controller from the external SDRAM or other suitable memory over the DMA channel for window descriptors. The window controller may store up to eight window descriptors in one embodiment of the present invention.

The step in step 316 preferably sends a new line header indicating the start of a new display line. The system in step 320 preferably sorts the window descriptors in accordance with the process described in reference to FIG. 7. Although sorting is indicated as a step in this flow diagram, sorting actually may be a continuous process of selecting the bottom-most window, i.e., the window to be processed. The system in step 322 preferably checks to determine if a

starting display line of the window is greater than the line count of the current display line. If the starting display line of the window is greater than the line count, i.e., if the current display line is above the starting display line of the bottom most window, the current display line is a blank line. Thus, the system in step 318 preferably increments the line count and sends another new line header in step 316. The process of sending a new line header and sorting window descriptor continues as long as the starting display line of the bottom most (in layer order) window is below the current display line.

The display engine and the associated graphics filter preferably operate in one of two modes, a field mode and a frame mode. In both modes, raw graphics data associated with graphics windows is preferably stored in frame format, including lines from both interlaced fields in the case of an interlaced display. In the field mode, the display engine preferably skips every other display line during processing. In the field mode, therefore, the system in step 318 preferably increments the line count by two each time to skip every other line. In the frame mode, the display engine processes every display line sequentially. In the frame mode, therefore, the system in step 318 preferably increments the line count by one each time.

When the system in step 322 determines that the starting display of the window is greater than the line count, the system in step 324 preferably determines from the header packet whether the window descriptor is for displaying a window or re-loading the CLUT. If the window header indicates that the window descriptor is for re-loading CLUT, the system in step 328 preferably sends the CLUT data to the CLUT and turns on the CLUT write strobe to load CLUT.

If the system in step 324 determines that the window descriptor is for displaying a window, the system in step 326 preferably sends a new window header to indicate that graphics data words for a new window on the display line are going to be transferred into the graphics FIFO. Then, the system in step 330 preferably requests the DMA module to send graphics data to the graphics FIFO over the DMA channel for graphics data. In the event the FIFO does not have sufficient space to store graphics data in a new data packet, the system preferably waits until such space is made available.

When graphics data for a display line of a current window is transferred to the FIFO, the system in step 332 preferably determines whether the last line of the current window has been transferred. If the last line has been transferred, a window descriptor done flag associated with the current window is preferably set. The window descriptor done flag indicates that the graphics data associated with the current window descriptor has been completely transferred. When the window descriptor done flag is set, i.e., when the current window descriptor is completely processed, the system sets a window descriptor done flag in step 334. Then the system in step 336 preferably sets a new window descriptor update flag and increments a window descriptor update counter to indicate that a new window descriptor is to be copied from the external memory.

Regardless of whether the last line of the current window has been processed, the system in step 338 preferably sets the window line done flag for the current window descriptor to signify that processing of this window descriptor on the current display line has been completed. The system in step 340 preferably checks the window line done flags associated with all eight window descriptors to determine whether they are all set, which would indicate that all the windows of the current display line have been processed. If not all window line done flags are set, the system preferably proceeds to step 320 to sort the window descriptors and repeat processing of the new bottom-most window descriptor.

If all eight window line done flags are determined to be set in step 340, all window descriptors on the current display line have been processed. In this case, the system in step 342 preferably checks whether an all window descriptor done flag has been set to determine whether all window descriptors have been processed completely. The all window descriptor done flag is set when processing of all window descriptors in the current frame or field have been processed completely. If the all window descriptor done flag is set, the system preferably returns to step 310 to reset and awaits another VSYNC in step 312. If not all window descriptors have been processed, the system in step 344 preferably determines if the new window descriptor update flag has been set. In the preferred embodiment, this flag would have been set in step 334 if the current window descriptor has been completely processed.

When the new window descriptor update flag is set, the system in step 352 preferably sets up the DMA to transfer a new window descriptor from the external memory. Then the system in step 350 preferably clears the new window descriptor update flag. After the system clears the new window descriptor update flag or when the new window descriptor update flag is not set in the first place, the system in step 348 preferably increments a line counter to indicate that the window descriptors for a next display line should be processed. The system in step 346 preferably clears all eight window line done flags to indicate that none of the window descriptors have been processed for the next display line. Then the system in step 316 preferably initiates processing of the new display line by sending a new line header to the FIFO.

In the preferred embodiment, the graphics converter in the display engine converts raw graphics data having various different formats into a common format for subsequent compositing with video and for display. The graphics converter preferably includes a state machine that changes state based on the content of the window data packet. Referring to FIG. 9, the state machine in the graphics converter preferably controls unpacking and processing of the header packets. A first header word processing state 354 is preferably entered wherein a first window parameter of the first header word is checked (step 356) to determine if the window data packet is for a first graphics window of a new line. If the header packet is not for a first window of a new line, after the first header word is processed, the state preferably changes to a second header word processing state 362.

If the header packet is for a first graphics window of a new line, the state machine preferably enters a clock switch state 358. In the clock switch state, the clock for a graphics line buffer which is going to store the new line switches from a display clock to a memory clock, e.g., from a 13.5 MHz clock to a 81 MHz clock. From the clock switch state, a graphics type in the first header word is preferably checked (step 360) to determine if the header packet represents an empty line. A graphics type of 1111b preferably refers to an empty line.

If the graphics type is 1111b, the state machine enters the first header word processing state 354, in which the first header word of the next header packet is processed. If the graphics type is not 1111b, i.e. the display line is not empty, the second header word is processed. Then the state machine

preferably enters a graphics content state **364** wherein words from the FIFO are checked (step 366) one at a time to verify that they are data words. The state machine preferably remains in the graphics content state as long as each word read is a data word. While in the graphics content state, if a word received is not a data word, i.e., it is a first or second header word, then the state machine preferably enters a pipeline complete state **368** and then to the first header processing state **354** where reading and processing of the next window data packet is commenced.

Referring to FIG. **10**, the display engine **58** is preferably coupled to memory over a memory interface **370** and a CLUT over a CLUT interface **372**. The display engine preferably includes the graphics FIFO **132** which receives the header packets and the graphics data from the memory controller over the memory interface. The graphics FIFO preferably provides received raw graphics data to the graphics converter **134** which converts the raw graphics data into the common compositing format. During the conversion of graphics format, the RGB to YUV converter **136** and data from the CLUT over the CLUT interface **372** are used to convert RGB formatted data and CLUT formatted data, respectively.

The graphics converter preferably processes all of the window layers of each scan line in half the time, or less, of an interlaced display line, due to the need to have lines from both fields available in the SRAM for use by the graphics filter when frame mode filtering is enabled. The graphics converter operates at 81 MHz in one embodiment of the present invention, and the graphics converter is able to process up to eight windows on each scan line and up to three full width windows.

For example, with a 13.5 MHz display clock, if the graphics converter processes 81 Mpixels per second, it can convert three windows, each covering the width of the display, in half of the active display time of an interlaced scan line. In one embodiment of the present invention, the graphics converter processes all the window layers of each scan line in half the time of an interlaced display line, due to the need to have lines from both fields available in the SRAM for use by the graphics filter. In practice, there may be some more time available since the active display time leaves out the blanking time, while the graphics converter can operate continuously.

Graphics pixels are preferably read from the FIFO in raw graphics format, using one of the multiple formats allowed in the present invention and specified in the window descriptor. Each pixel may occupy as little as two bits or as much as 16 bits in the preferred embodiment. Each pixel is converted to a YUVa24 format (also referred to as aYUV 4:4:2:2), such as two adjacent pixels sharing a UV pair and having unique Y and alpha values, and each of the Y, U, V and alpha components occupying eight bits. The conversion process is generally dependent on the pixel format type and the alpha specification method, both of which are indicated by the window descriptor for the currently active window. Preferably, the graphics converter uses the CLUT memory to convert CLUT format pixels into RGB or YUV pixels.

Conversions of RGB pixels may require conversion to YUV, and therefore, the graphics converter preferably includes a color space converter. The color space converter preferably is accurate for all coefficients. If the converter is accurate to eight or nine bits it can be used to accurately convert eight bit per component graphics, such as CLUT entries with this level of accuracy or RGB24 images.

The graphics converter preferably produces one converted pixel per clock cycle, even when there are multiple graphics pixels packed into one word of data from the FIFO. Preferably the graphics processing clock, which preferably runs at 81 MHz, is used during the graphics conversion. The graphics converter preferably reads data from the FIFO whenever both conditions are met, including that the converter is ready to receive more data, and the FIFO has data ready. The graphics converter preferably receives an input from a graphics blender, which is the next block in the pipeline, which indicates when the graphics blender is ready to receive more converted graphics data. The graphics converter may stall if the graphics blender is not ready, and as a result, the graphics converter may not be ready to receive graphics data from the FIFO.

The graphics converter preferably converts the graphics data into a YUValpha ("YUVa") format. This YUVa format includes YUV 4:2:2 values plus an 8-bit alpha value for every pixel, and as such it occupies 24 bits per pixel; this format is alternately referred to as aYUV 4:4:2:2. The YUV444-to-YUV422 converter 138 converts graphics data with the aYUV 4:4:4:4 format from the graphics converter into graphics data with the aYUV 4:4:2:2 format and provides the data to the graphics blender **140**. The YUV444-to-YUV422 converter preferably has a capacity of performing low pass filtering to filter out high frequency components when needed. The graphics converter also sends and receives clock synchronization information to and from the graphics line buffers over a clock control interface **376**.

When provided with the converted graphics data, the graphics blender **140** preferably composites graphics windows into graphics line buffers over a graphics line buffer interface **374**. The graphics windows are alpha blended into blended graphics and preferably stored in graphics line buffers.

IV. Color Look-Up Table Loading Mechanism

A color look-up table ("CLUT") is preferably used to supply color and alpha values to the raw graphics data formatted to address information contents of the CLUT. For a window surface based display, there may be multiple graphics windows on the same display screen with different graphics formats. For graphics windows using a color look-up table (CLUT) format, it may be necessary to load specific color look-up table entries from external memory to on-chip memory before the graphics window is displayed.

The system preferably includes a display engine that processes graphics images formatted in a plurality of formats including a color look up table (CLUT) format. The system provides a data structure that describes the graphics in a window, provides a data structure that provides an indicator to load a CLUT, sorts the data structures into a list according to the location of the window on the display, and loads conversion data into a CLUT for converting the CLUT-formatted data into a different data format according to the sequence of data structures on the list.

In the preferred embodiment, each window on the display screen is described with a window descriptor. The same window descriptor is used to control CLUT loading as the window descriptor used to display graphics on screen. The window descriptor preferably defines the memory starting address of the graphics contents, the x position on the display screen, the width of the window, the starting vertical display line and end vertical display line, window layer, etc. The same window structure parameters and corresponding fields may be used to define the CLUT loading. For example, the graphics contents memory starting address may define

CLUT memory starting address; the width of graphics window parameter may define the number of CLUT entries to be loaded; the starting vertical display line and ending vertical display line parameters may be used to define when to load the CLUT; and the window layer parameter may be used to define the priority of CLUT loading if several windows are displayed at the same time, i.e., on the same display line.

In the preferred embodiment, only one CLUT is used. As such, the contents of the CLUT are preferably updated to display graphics windows with CLUT formatted data that is not supported by the current content of the CLUT. One of ordinary skill in the art would appreciate that it is straightforward to use more than one CLUT and switch back and forth between them for different graphics windows.

In the preferred embodiment, the CLUT is closely associated with the graphics converter. In one embodiment of the present invention, the CLUT consists of one SRAM with 256 entries and 32 bits per entry. In other embodiments, the number of entries and bits per entry may vary. Each entry contains three color components; either RGB or YUV format, and an alpha component. For every CLUT-format pixel converted, the pixel data may be used as the address to the CLUT and the resulting value may be used by the converter to produce the YUVa (or alternatively RGBa) pixel value.

The CLUT may be re-loaded by retrieving new CLUT data via the direct memory access module when needed. It generally takes longer to re-load the CLUT than the time available in a horizontal blanking interval. Accordingly, in the preferred embodiment, a whole scan line time is allowed to re-load the CLUT. While the CLUT is being reloaded, graphics images in non-CLUT formats may be displayed. The CLUT reloading is preferably initiated by a window descriptor that contains information regarding CLUT reloading rather than a graphics window display information.

Referring to FIG. **11**, the graphics CLUT **146** preferably includes a graphics CLUT controller **400** and a static dual-port RAM (SRAM) **402**. The SRAM preferably has a size of 256×32 which corresponds to 256 entries in the graphics CLUT. Each entry in the graphics CLUT preferably has 32 bits composed of Y+U+V+alpha from the most significant bit to the least significant bit. The size of each field, including Y, U, V, and alpha, is preferably eight bits.

The graphics CLUT preferably has a write port that is synchronized to a 81 MHz memory clock and a read port that may be asynchronous to the memory clock. The read port is preferably synchronous to the graphics processing clock, which runs preferably at 81 MHz, but not necessarily synchronized to the memory clock. During a read operation, the static dual-port RAM ("SRAM") is preferably addressed by a read address which is provided by graphics data in the CLUT images. During the read operation, the graphics data is preferably output as read data **414** when a memory address in the CLUT containing that graphics data is addressed by a read address **412**.

During write operations, the window controller preferably controls the write port with a CLUT memory request signal **404** and a CLUT memory write signal **408**. CLUT memory data **410** is also preferably provided to the graphics CLUT via the direct memory access module from the external memory. The graphics CLUT controller preferably receives the CLUT memory data and provides the received CLUT memory data to the SRAM for writing.

Referring to FIG. **12**, an exemplary timing diagram shows different signals involved during a writing operation of the CLUT. The CLUT memory request signal **418** is asserted when the CLUT is to be re-loaded. A rising edge of the CLUT memory request signal **418** is used to reset a write pointer associated with the write port. Then the CLUT memory write signal **420** is asserted to indicate the beginning of a CLUT re-loading operation. The CLUT memory data **422** is provided synchronously to the 81 MHz memory clock **416** to be written to the SRAM. The write pointer associated with the write port is updated each time the CLUT is loaded with CLUT memory data.

In the preferred embodiment, the process of reloading a CLUT is associated with the process of processing window descriptors illustrated in FIG. **8** since CLUT re-loading is initiated by a window descriptor. As shown in steps 324 and 328 of FIG. **8**, if the window descriptor is determined to be for reloading CLUT in step **324**, the system in step 328 sends the CLUT data to the CLUT. The window descriptor for the CLUT reloading may appear anywhere in the window descriptor list. Accordingly, the CLUT reloading may take place at any time whenever CLUT data is to be updated.

Using the CLUT loading mechanism in one embodiment of the present invention, more than one window with different CLUT tables may be displayed on the same display line. In this embodiment, only the minimum required entries are preferably loaded into the CLUT, instead of loading all the entries every time. The loading of only the minimum required entries may save memory bandwidth and enables more functionality. The CLUT loading mechanism is preferably relatively flexible and easy to control, making it suitable for various applications. The CLUT loading mechanism of the present invention may also simplify hardware design, as the same state machine for the window controller may be used for CLUT loading. The CLUT preferably also shares the same DMA logic and layer/priority control logic as the window controller.

V. Graphics Line Buffer Control Scheme

In the preferred embodiment of the present invention, the system preferably blends a plurality of graphics images using line buffers. The system initializes a line buffer by loading the line buffer with data that represents transparent black, obtains control of a line buffer for a compositing operation, composites graphics contents into the line buffer by blending the graphics contents with the existing contents of the line buffer, and repeats the step of compositing graphics contents into the line buffer until all of the graphics surfaces for the particular line have been composited.

The graphics line buffer temporarily stores composited graphics images (blended graphics). A graphics filter preferably uses blended graphics in line buffers to perform vertical filtering and scaling operations to generate output graphics images. In the preferred embodiment, the display engine composites graphics images line by line using a clock rate that is faster than the pixel display rate, and graphics filters run at the pixel display rate. In other embodiments, multiple lines of graphics images may be composited in parallel. In still other embodiments, the line buffers may not be needed. Where line buffers are used, the system may incorporate an innovative control scheme for providing the line buffers containing blended graphics to the graphics filter and releasing the line buffers that are used up by the graphics filter.

The line buffers are preferably built with synchronous static dual-port random access memory ("SRAM") and dynamically switch their clocks between a memory clock and a display clock. Each line buffer is preferably loaded

with graphics data using the memory clock and the contents of the line buffer is preferably provided to the graphics filter synchronously to the display clock. In one embodiment of the present invention, the memory clock is an 81 MHz clock used by the graphics converter to process graphics data while the display clock is a 13.5 MHz clock used to display graphics and video signals on a television screen. Other embodiments may use other clock speeds.

Referring to FIG. **13**, the graphics line buffer preferably includes a graphics line buffer controller **500** and line buffers **504**. The graphics line buffer controller **500** preferably receives memory clock buffer control signals **508** as well as display clock buffer control signals **510**. The memory clock control signals and the display clock control signals are used to synchronize the graphics line buffers to the memory clock and the display clock, respectively. The graphics line buffer controller receives a clock selection vector **514** from the display engine to control which graphics line buffers are to operate in which clock domain. The graphics line buffer controller returns a clock enable vector to the display engine to indicate clock synchronization settings in accordance with the clock selection vector.

In the preferred embodiment, the line buffers **504** include seven line buffers **506** a–g. The line buffers temporarily store lines of YUVa24 graphics pixels that are used by a subsequent graphics filter. This allows for four line buffers to be used for filtering and scaling, two are available for progressing by one or two lines at the end of every line, and one for the current compositing operation. Each line buffer may store an entire display line. Therefore, in this embodiment, the total size of the line buffers is (720 pixels/display line) * (3 bytes/pixel) * (7 lines) =15,120 bytes.

Each of the ports to the SRAM including line buffers is 24 bits wide to accommodate graphics data in YUVa24 format in this embodiment of the present invention. The SRAM has one read port and one write port. One read port and one write port are used for the graphics blender interface, which performs a read-modify-write typically once per clock cycle. In another embodiment of the present invention, an SRAM with only one port is used. In yet another embodiment, the data stored in the line buffers may be YUVa32 (4:4:4:4), RGBa32, or other formats. Those skilled in the art would appreciate that it is straightforward to vary the number of graphics line buffers, e.g., to use different number of taps for filter, the format of graphics data or the number of read and write ports for the SRAM.

The line buffers are preferably controlled by the graphics line buffer controller over a line buffer control interface **502**. Over this interface, the graphics line buffer controller transfers graphics data to be loaded to the line buffers. The graphics filter reads contents of the line buffers over a graphics line buffer interface **516** and clears the line buffers by loading them with transparent black pixels prior to releasing them to be loaded with more graphics data for display.

Referring FIG. **14**, a flow diagram of a process of using line buffers to provide composited graphics data from a display engine to a graphics filter is illustrated. After the graphics display system is reset in step **520**, the system in step 522 receives a vertical sync (VSYNC) indicating a field start. Initially, all line buffers preferably operate in the memory clock domain. Accordingly, the line buffers are synchronized to the 81 MHz memory clock in one embodiment of the present invention. In other embodiments, the speed of the memory clock may be different from 81 MHz, or the line buffers may not operate in the clock domain of the main memory. The system in step 524 preferably resets all line buffers by loading them with transparent black pixels.

The system in step 526 preferably stores composited graphics data in the line buffers. Since all buffers are cleared at every field start by the display engine to the equivalent of transparent black pixels, the graphics data may be blended the same way for any graphics window, including the first graphics window to be blended. Regardless of how many windows are composited into a line buffer, including zero windows, the result is preferably always the correct pixel data.

The system in step 528 preferably detects a horizontal sync (HSYNC) which signifies a new display line. At the start of each display line, the graphics blender preferably receives a line buffer release signal from the graphics filter when one or more line buffers are no longer needed by the graphics filter. Since four line buffers are used with the four-tap graphics filter at any given time, one to three line buffers are preferably made available for use by the graphics blender to begin constructing new display lines in them. Once a line buffer release signal is recognized, an internal buffer usage register is updated and then clock switching is performed to enable the display engine to work on the newly released one to three line buffers. In other embodiments, the number of line buffers may be more or less than seven, and more or less than three line buffers may be released at a time.

The system in step 534 preferably performs clock switching. Clock switching is preferably done in the memory clock domain by the display engine using a clock selection vector. Each bit of the clock selection vector preferably corresponds to one of the graphics line buffers. Therefore, in one embodiment of the present invention with seven graphics line buffers, there are seven bits in the clock selection vector. For example, a corresponding bit of logic 1 in the clock selection vector indicates that the line buffer operates in the memory clock domain while a corresponding bit of logic 0 indicates that the line buffer operates in the display clock domain.

Other embodiments may have different numbers of line buffers and the number of bits in the clock selection vector may vary accordingly. Clock switching logic preferably switches between the memory clock and the display clock in accordance with the clock selection vector. The clock selection vector is preferably also used to multiplex the memory clock buffer control signals and the display clock buffer control signals.

Since there is preferably no active graphics data at field and line starts, clock switching preferably is done at the field start and the line start to accommodate the graphics filter to access graphics data in real-time. At the field and line starts, clock switching may be done without causing glitches on the display side. Clock switching typically requires a dead cycle time. A clock enable vector indicates that the graphics line buffers are ready to synchronize to the clocks again. The clock enable vector is preferably the same size at the clock selection vector. The clock enable vector is returned to the display engine to be compared with the clock selection vector.

During clock switching, the clock selection vector is sent by the display engine to the graphics line buffer block. The clocks are preferably disabled to ensure a glitch-free clock switching. The graphics line buffers send the clock enable vector to the display engine with the clock synchronization settings requested in the clock selection vector. The display engine compares contents of the clock selection vector and the clock enable vector. When the contents match, the clock synchronization is preferably turned on again.

After the completion of clock switching during the video inactive region, the system in step 536 preferably provides the graphics data in the line buffers to the graphics filter for anti-flutter filtering, sample rate conversion (SRC) and display. At the end of the current display line, the system looks for a VSYNC in step **538**. If the VSYNC is detected, the current field has been completed, and therefore, the system in step 530 preferably switches clocks for all line buffers to the memory clock and resets the line buffers in step 524 for display of another field. If the VSYNC is not detected in step **538**, the current display line is not the last display line of the current field. The system continues to step 528 to detect another HSYNC for processing and displaying of the next display line of the current field.

VI. Window Soft Horizontal Scrolling Mechanism

Sometimes it is desirable to scroll a graphics window softly, e.g., display text that moves from left to right or from right to left smoothly on a television screen. There are some difficulties that may be encountered in conventional methods that seek to implement horizontal soft scrolling.

Graphics memory buffers are conventionally implemented using low-cost DRAM, SDRAM, for example. Such memory devices are typically slow and may require each burst transfer to be within a page. Smooth (or soft) horizontal scrolling, however, preferably enables the starting address to be set to any arbitrary pixel. This may conflict with the transfer of data in bursts within the well-defined pages of DRAM. In addition, complex control logic may be required to monitor if page boundaries are to be crossed during the transfer of pixel maps for each step during soft horizontal scrolling.

In the preferred embodiment, an implementation of a soft horizontal scrolling mechanism is achieved by incrementally modifying the content of a window descriptor for a particular graphics window. The window soft horizontal scrolling mechanism preferably enables positioning the contents of graphics windows on arbitrary positions on a display line.

In an embodiment of the present invention, the soft horizontal scrolling of graphics windows is implemented based on an architecture in which each graphics window is independently stored in a normal graphics buffer memory device (SDRAM, EDO-DRAM, DRAM) as a separate object. Windows are composed on top of each other in real time as required. To scroll a window to the left or right, a special field is defined in the window descriptor that tells how many pixels are to be shifted to the left or right.

The system according to the present invention provides a method of horizontally scrolling a display window to the left, which includes the steps of blanking out one or more pixels at a beginning of a portion of graphics data, the portion being aligned with a start address; and displaying the graphics data starting at the first non-blanked out pixel in the portion of the graphics data aligned with the start address.

The system according to the present invention also provides a method of horizontally scrolling a display window to the right which includes the steps of moving a read pointer to a new start address that is immediately prior to a current start address, blanking out one or more pixels at a beginning of a portion of graphics data, the portion being aligned to the new start address, and displaying the graphics data starting at the first non-blanked out pixel in the portion of the graphics data aligned with the new start address.

In practice, each graphics window is preferably addressed using an integer word address. For example, if the memory system uses 32 bit words, then the address of the start of a window is defined to be aligned to a multiple of 32 bits, even if the first pixel that is desired to be displayed is not so aligned. Each graphics window also preferably has associated with it a horizontal offset parameter, in units of pixels, that indicates a number of pixels to be ignored, starting at the indicated starting address, before the active display of the window starts. In the preferred embodiment, the horizontal offset parameter is the blank start pixel value in the word 3 of the window descriptor. For example, if the memory system uses 32-bit words and the graphics format of a window uses 8 bits per pixel, each 32-bit word contains four pixels. In this case, the display of the window may ignore one, two or three pixels (8, 16, or 24 bits), causing an effective left shift of one, two, or three pixels.

In the embodiment illustrated by the above example, the memory system uses 32-bit words. In other embodiments, the memory system may use more or less number of bits per word, such as 16 bits per word or 64 bits per word. In addition, pixels in other embodiments may have various different number of bits per pixel, such as 1, 2, 4, 8, 16, 24 and 32.

Referring to FIG. **15**, in the preferred embodiment, a first pixel (e.g., the first 8 bits) 604 of a 32-bit word 600, which is aligned to the start address, is blanked out. The remaining three 8-bit pixels, other than the blanked out first pixel, are effectively shifted to the left by one pixel. Prior to blanking out, a read pointer **602** points to the first bit of the 32-bit word. After blanking out, the read pointer **602** points to the ninth bit of the 32-bit word.

Further, a shift of four pixels is implemented by changing the start address by one to the next 32-bit word. Shifts of any number of pixels are thereby implemented by a combination of adjusting the starting word address and adjusting the pixel shift amount. The same mechanism may be used for any number of bits per pixel (1, 2, 4, etc.) and any memory word size.

To shift a pixel or pixels to the right, the shifting cannot be achieved simply by blanking some of the bits at the start address since any blanking at the start will simply have an effect of shifting pixels to the left. Further, the shifting to the right cannot be achieved by blanking some of the bits at the end of the last data word of a display line since display of a window starts at the start address regardless of the position of the last pixel to be displayed.

Therefore, in one embodiment of the present invention, when the graphics display is to be shifted to the right, a read pointer pointing at the start address is preferably moved to an address that is just before the start address, thereby making that address the new start address. Then, a portion of the data word aligned with the new start address is blanked out. This provides the effect of shifting the graphics display to the right.

For example, a memory system may use 32-bit words and the graphics format of a window may use 2 bits per pixel, e.g., a CLUT 2 format. If the graphics display is to be shifted by a pixel to the right, the read pointer is moved to an address that is just before the start address, and that address becomes a new start address. Then, the first 30 bits of the 32-bit word that is aligned with the new start address are blanked out. In this case, blanking out of a portion of the 32-bit word that is aligned with the new start address has the effect of shifting the graphics display to the right.

Referring to FIG. **16**, a 32-bit word **610** that is aligned with the starting address is shifted to the right by one pixel. The 32-bit word **610** has a CLUT **2** format, and therefore

contains 16 pixels. A read pointer **612** points at the beginning of the 32-bit word **610**. To shift the pixels in the 32-bit word **610** to the right, an address that is just before the start address is made a new start address. A 32-bit data word **618** is aligned with the new start address. Then, the first 30 bits (15 pixels) **616** of the 32-bit data word **618** aligned with the new start address are blanked out. The read pointer **612** points at a new location, which is the 31st bit of the new start address. The 31st bit and the 32nd bit of the new start address may constitute a pixel **618**. Insertion of the pixel **618** in front of 16 pixels of the 32-bit data word **610** effectively shifts those 16 pixels to the right by one pixel.

VII. Anti-Aliased Text and Graphics

TV-based applications, such as interactive program guides, enhanced TV, TV navigators, and web browsing on TV frequently require the display of text and line-oriented graphics on the display. A graphical element or glyph generally represents an image of text or graphics. Graphical element may refer to text glyphs or graphics. In conventional methods of displaying text on TV or computer displays, graphical elements are rendered as arrays of pixels (picture elements) with two states for every pixel, i.e. the foreground and background colors.

In some cases the background color is transparent, allowing video or other graphics to show through. Due to the relatively low resolution of most present day TVs, diagonal and round edges of graphical elements generally show a stair-stepped appearance which may be undesirable; and fine details are constrained to appear as one or more complete pixels (dots), which may not correspond well to the desired appearance. The interlaced nature of TV displays causes horizontal edges of graphical elements, or any portion of graphical elements with a significant vertical gradient, to show a "fluttering" appearance with conventional methods.

Some conventional methods blend the edges of graphical elements with background colors in a frame buffer, by first reading the color in the frame buffer at every pixel where the graphical element will be written, combining that value with the foreground color of the graphical element, and writing the result back to the frame buffer memory. This method requires there to be a frame buffer; it requires the frame buffer to use a color format that supports such blending operations, such as RGB24 or RGB16, and it does not generally support the combination of graphical elements over full motion video, as such functionality may require repeating the read, combine and write back function of all pixels of all graphical elements for every frame or field of the video in a timely manner.

The system preferably displays a graphical element by filtering the graphical element with a low pass filter to generate a multi-level value per pixel at an intended final display resolution and uses the multi-level values as alpha blend values or the graphical element in the subsequent compositing stage.

In one embodiment of the present invention, a method of displaying graphical elements on televisions and other displays is used. A deep color frame buffer with, for example, 16, 24, or 32 bits per pixel, is not required to implement this method since this method is effective with as few as two bits per pixel. Thus, this method may result in a significant reduction in both the memory space and the memory bandwidth required to display text and graphics. The method preferably provides high quality when compared with conventional methods of anti-aliased text, and produces higher display quality than is available with conventional methods that do not support anti-aliased text.

Referring to FIG. **17**, a flow diagram illustrates a process of providing very high quality display of graphical elements in one embodiment of the present invention. First, the bi-level graphical elements are filtered by the system in step **652**. The graphical elements are preferably initially rendered by the system in step 650 at a significantly higher resolution than the intended final display resolution, for example, four times the final resolution in both horizontal and vertical axes. The filter may be any suitable low pass filter, such as a "box" filter. The result of the filtering operation is a multi-level value per pixel at the intended display resolution.

The number of levels may be reduced to fit the number of bits used in the succeeding steps. The system in step 654 determines whether the number of levels are to be reduced by reducing the number of bits used. If the system determines that the number of levels are to be reduced, the system in step **656** preferably reduces the number of bits. For example, the result of box-filtering 4×4 super-sampled graphical elements normally results in 17 possible levels; these may be converted through truncation or other means to 16 levels to match a 4 bit representation, or eight levels to match a 3 bit representation, or four levels to match a 2 bit representation. The filter may provide a required vertical axis low pass filter function to provide anti-flutter filter effect for interlaced display.

In step **658**, the system preferably uses the resulting multi-level values, either with or without reduction in the number of bits, as alpha blend values, which are preferably pixel alpha component values, for the graphical elements in a subsequent compositing stage. The multi-level graphical element pixels are preferably written into a graphics display buffer where the values are used as alpha blend values when the display buffer is composited with other graphics and video images.

In an alternate embodiment, the display buffer is defined to have a constant foreground color consistent with the desired foreground color of the text or graphics, and the value of every pixel in the display buffer is defined to be the alpha blend value for that pixel. For example, an Alpha-4 format specifies four bits per pixel of alpha blend value in a graphics window, where the 4 bits define alpha blend values of 0/16, 1/16, 2/16, . . . , 13/16, 14/16, and 16/16. The value 15/16 is skipped in this example in order to obtain the endpoint values of 0 and 16/16 (1) without requiring the use of an additional bit. In this example format, the display window has a constant foreground color which is specified in the window descriptor.

In another alternate embodiment, the alpha blend value per pixel is specified for every pixel in the graphical element by choosing a CLUT index for every pixel, where the CLUT entry associated with every index contains the desired alpha blend value as part of the CLUT contents. For example, a graphical element with a constant foreground color and 4 bits of alpha per pixel can be encoded in a CLUT 4 format such that every pixel of the display buffer is defined to be a 4 bit CLUT index, and each of the associated 16 CLUT entries has the appropriate alpha blend value (0/16, 1/16, 2/16, . . . , 14/16, 16/16) as well as the (same) constant foreground color in the color portion of the CLUT entries.

In yet another alternate embodiment, the alpha per pixel values are used to form the alpha portion of color+alpha pixels in the display buffer, such as alphaRGB(4,4,4,4) with 4 bits for each of alpha, Red, Green, and Blue, or alphaRGB32 with 8 bits for each component. This format does not require the use of a CLUT.

In still another alternate embodiment, the graphical element may or may not have a constant foreground color. The

various foreground colors are processed using a low-pass filter as described earlier, and the outline of the entire graphical element (including all colors other than the background) is separately filtered also using a low pass filter as described. The filtered foreground color is used as either the direct color value in, e.g., an alphaRGB format (or other color space, such as alphaYUV) or as the color choice in a CLUT format, and the result of filtering the outline is used as the alpha per pixel value in either a direct color format such as alphaRGB or as the choice of alpha value per CLUT entry in a CLUT format.

The graphical elements are displayed on the TV screen by compositing the display buffer containing the graphical elements with optionally other graphics and video contents while blending the subject display buffer with all layers behind it using the alpha per pixel values created in the preceding steps. Additionally, the translucency or opacity of the entire graphical element may be varied by specifying the alpha value of the display buffer via such means as the window alpha value that may be specified in a window descriptor.

VIII. Video Synchronization

When a composite video signal (analog video) is received into the system, it is preferably digitized and separated into YUV (luma and chroma) components for processing. Samples taken for YUV are preferably synchronized to a display clock for compositing with graphics data at the video compositor. Mixing or overlaying of graphics with decoded analog video may require synchronizing the two image sources exactly. Undesirable artifacts such as jitter may be visible on the display unless a synchronization mechanism is implemented to correctly synchronize the samples from the analog video to the display clock. In addition, analog video often does not adhere strictly to the television standards such as NTSC and PAL. For example, analog video which originates in VCRs may have synchronization signals that are not aligned with chroma reference signals and also may have inconsistent line periods. Thus, the synchronization mechanism preferably should correctly synchronize samples from non-standard analog videos as well.

The system, therefore, preferably includes a video synchronizing mechanism that includes a first sample rate converter for converting a sampling rate of a stream of video samples to a first converted rate, a filter for processing at least some of the video samples with the first converted rate, and a second sample rate converter for converting the first converted rate to a second converted rate.

Referring to FIG. **18**, the video decoder **50** preferably samples and synchronizes the analog video input. The video receiver preferably receives an analog video signal **706** into an analog-to-digital converter (ADC) **700** where the analog video is digitized. The digitized analog video **708** is preferably sub-sampled by a chroma-locked sample rate converter (SRC) **708**. A sampled video signal **710** is provided to an adaptive 2H comb filter/chroma demodulator/luma processor **702** to be separated into YUV (luma and chroma) components. In the 2H comb filter/chroma demodulator/luma processor **702**, the chroma components are demodulated. In addition, the luma component is preferably processed by noise reduction, coring and detail enhancement operations. The adaptive 2H comb filter provides the sampled video **712**, which has been separated into luma and chroma components and processed, to a line-locked SRC **704**. The luma and chroma components of the sample video is preferably sub-sampled once again by the line-locked SRC and the sub-sampled video **714** is provided to a time base corrector (TBC) **72**. The time base corrector preferably provides an output video signal **716** that is synchronized to a display clock of the graphics display system. In one embodiment of the present invention, the display clock runs at a nominal 13.5 MHz.

The synchronization mechanism preferably includes the chroma-locked SRC **70**, the line-locked SRC **704** and the TBC **72**. The chroma-locked SRC outputs samples that are locked to chroma subcarrier and its reference bursts while the line-locked SRC outputs samples that are locked to horizontal syncs. In the preferred embodiment, samples of analog video are over-sampled by the ADC **700** and then down-sampled by the chroma-locked SRC to four times the chroma sub-carrier frequency (Fsc). The down-sampled samples are down-sampled once again by the line-locked SRC to line-locked samples with an effective sample rate of nominally 13.5 MHz. The time base corrector is used to align these samples to the display clock, which runs nominally at 13.5 MHz.

Analog composite video has a chroma signal frequency interleaved in frequency with the luma signal. In an NTSC standard video, this chroma signal is modulated on to the Fsc of approximately 3.579545 MHz, or exactly 227.5 times the horizontal line rate. The luma signal covers a frequency span of zero to approximately 4.2 MHz. One method for separating the luma from the chroma is to sample the video at a rate that is a multiple of the chroma sub-carrier frequency, and use a comb filter on the sampled data. This method generally imposes a limitation that the sampling frequency is a multiple of the chroma sub-carrier frequency (Fsc).

Using such a chroma-locked sampling frequency generally imposes significant costs and complications on the implementation, as it may require the creation of a sample clock of the correct frequency, which itself may require a stable, low noise controllable oscillator (e.g. a VCXO) in a control loop that locks the VCXO to the chroma burst frequency. Different sample frequencies are typically required for different video standards with different chroma subcarrier frequencies. Sampling at four times the subcarrier frequency, i.e. 14.318 MHz for NTSC standard and 17.72 MHz for PAL standard, generally requires more anti-alias filtering before digitization than is required when sampling at higher frequencies such as 27 MHz. In addition, such a chroma-locked clock frequency is often unrelated to the other frequencies in a large scale digital device, requiring multiple clock domains and asynchronous internal interfaces.

In the preferred embodiment, however, the samples are not taken at a frequency that is a multiple of Fsc. Rather, in the preferred embodiment, an integrated circuit takes samples of the analog video at a frequency that is essentially arbitrary and that is greater than four times the Fsc (4 Fsc=14.318 MHz) . The sampling frequency preferably is 27 MHz and preferably is not locked to the input video signal in phase or frequency. The sampled video data then goes through the chroma-locked SRC that down-samples the data to an effective sampling rate of 4 Fsc. This and all subsequent operations are preferably performed in digital processing in a single integrated circuit.

The effective sample rate of 4 Fsc does not require a clock frequency that is actually at 4 Fsc, rather the clock frequency can be almost any higher frequency, such as 27 MHz, and valid samples occur on some clock cycles while the overall rate of valid samples is equal to 4 Fsc. The down-sampling (decimation) rate of the SRC is preferably controlled by a

chroma phase and frequency tracking module. The chroma phase and frequency tracking module looks at the output of the SRC during the color burst time interval and continuously adjusts the decimation rate in order to align the color burst phase and frequency. The chroma phase and frequency tracking module is implemented as a logical equivalent of a phase locked loop (PLL), where the chroma burst phase and frequency are compared in a phase detector to the effective sample rate, which is intended to be 4 Fsc, and the phase and frequency error terms are used to control the SRC decimation rate.

The decimation function is applied to the incoming sampled video, and therefore the decimation function controls the chroma burst phase and frequency that is applied to the phase detector. This system is a closed feedback loop (control loop) that functions in much the same way as a conventional PLL, and its operating parameters are readily designed in the same way as those of PLLs.

Referring to FIG. **19**, the chroma-locked SRC **70** preferably includes a sample rate converter (SRC) **730**, a chroma tracker **732** and a low pass filter (LPF). The SRC **730** is preferably a polyphase filter having time-varying coefficients. The SRC is preferably implemented with 35 phases and the conversion ratio of 35/66. The SRC **730** preferably interpolates by exactly 35 and decimates by (66+epsilon), i.e. the decimation rate is preferably adjustable within a range determined by the minimum and maximum values of epsilon, generally a small range. Epsilon is a first adjustment value, which is used to adjust the decimation rate of a first sample rate converter , i.e., the chroma-locked sample rate converter.

Epsilon is preferably generated by the control loop comprising the chroma tracker **732** and the LPF **734**, and it can be negative, positive or zero. When the output samples of the SRC **730** are exactly frequency and phase locked to the color sub-carrier then epsilon is zero. The chroma tracker tracks phase and frequency of the chroma bursts and compares them against an expected pattern.

In one embodiment of the present invention, the conversion rate of the chroma-locked SRC is adjusted so that, in effect, the SRC samples the chroma burst at exactly four times per chroma sub-carrier cycle. The SRC takes the samples at phases 0 degrees, 90 degrees, 180 degrees and 270 degrees of the chroma sub-carrier cycle. This means that a sample is taken at every cycle of the color sub-carrier at a zero crossing, a positive peak, zero crossing and a negative peak, (0, +1, 0, −1) . If the pattern obtained from the samples is different from (0, +1, 0, −1), this difference is detected and the conversion ratio needs to be adjusted inside the control loop.

When the output samples of the chroma-locked SRC are lower in frequency or behind in phase, e.g., the pattern looks like (−1, 0, +1, 0), then the chroma tracker **732** will make epsilon negative. When epsilon is negative, the sample rate conversion ratio is higher than the nominal 35/66, and this has the effect of increasing the frequency or advancing the phase of samples at the output of the chroma-locked SRC. When the output samples of the chroma-locked SRC are higher in frequency or leading in phase, e.g., the pattern looks like (+1, 0, −1, 0), then the chroma tracker **732** will make epsilon positive. When epsilon is positive, the sample rate conversion ratio is lower than the nominal 35/66, and this has the effect of decreasing the frequency or retarding the phase of samples out of the chroma-locked SRC. The chroma tracker provides error signal **736** to the LPF **734** that filters the error signal to filter out high frequency components and provides the filtered error signal to the SRC to complete the control loop.

The sampling clock may run at the system clock frequency or at the clock frequency of the destination of the decoded digital video. If the sampling clock is running at the system clock, the cost of the integrated circuit may be lower than one that has a system clock and a sub-carrier locked video decoder clock. A one clock integrated circuit may also cause less noise or interference to the analog-to-digital converter on the IC. The system is preferably all digital, and does not require an external crystal or a voltage controlled oscillator.

Referring to FIG. **20**, an alternate embodiment of the chroma-locked SRC **70** preferably varies the sampling rate while the conversion rate is held constant. A voltage controlled oscillator (e.g., VCXO) **760** varies the sampling rate by providing a sampling frequency signal **718** to the ADC **700**. The conversion rate in this embodiment is fixed at 35/66 in the SRC **750** which is the ratio between four times the chroma sub-carrier frequency and 27 MHz.

In this embodiment, the chroma burst signal at the output of the chroma-locked SRC is compared with the expected chroma burst signal in a chroma tracker **752**. The error signals **756** from the comparison between the converted chroma burst and the expected chroma burst are passed through a low pass filter **754** and then filtered error signals **758** are provided to the VCXO **760** to control the oscillation frequency of the VCXO. The oscillation frequency of the VCXO changes in response to the voltage level of the provided error signals. Use of input voltage to control the oscillation frequency of a VCXO is well known in the art. The system as described here is a form of a phase locked loop (PLL), the design and use of which is well known in the art.

After the completion of chroma-luma separation and other processing to the chroma and luma components, the samples with the effective sample rate of 4 Fsc (i.e. 4 times the chroma subcarrier frequency) are preferably decimated to samples with a sample rate of nominally 13.5 MHz through the use of a second sample rate converter. Since this sample rate is less than the electrical clock frequency of the digital integrated circuit in the preferred embodiment, only some clock cycles carry valid data. In this embodiment, the sample rate is preferably converted to 13.5 MHz, and is locked to the horizontal line rate through the use of horizontal sync signals. Thus, the second sample rate converter is a line-locked sample rate converter (SRC).

The line-locked sample rate converter converts the current line of video to a constant (Pout) number of pixels. This constant number of pixels Pout is normally 858 for ITU-R BT.601 applications and 780 for NTSC square pixel applications. The current line of video may have a variable number of pixels (Pin). In order to do this conversion from a chroma-locked sample rate, the following steps are performed. The number of input samples Pin of the current line of video is accurately measured. This line measurement is used to calculate the sample rate conversion ratio needed to convert the line to exactly Pout samples. An adjustment value to the sample rate conversion ratio is passed to a sample rate converter module in the line-locked SRC to implement the calculated sample rate conversion ratio for the current line. The sample conversion ratio is calculated only once for each line. Preferably, the line-locked SRC also scales YUV components to the proper amplitudes required by ITU-R BT.601.

The number of samples detected in a horizontal line may be more or less if the input video is a non-standard video.

For example, if the incoming video is from a VCR, and the sampling rate is four times the color sub-carrier frequency (4 Fsc), then the number of samples taken between two horizontal syncs may be more or less than 910, where 910 is the number of samples per line that is obtained when sampling NTSC standard video at a sampling frequency of 4 Fsc. For example, the horizontal line time from a VCR may vary if the video tape has been stretched.

The horizontal line time may be accurately measured by detecting two successive horizontal syncs. Each horizontal sync is preferably detected at the leading edge of the horizontal sync. In other embodiments, the horizontal syncs may be detected by other means. For example, the shape of the entire horizontal sync may be looked at for detection. In the preferred embodiment, the sample rate for each line of video has been converted to four times the color sub-carrier frequency (4 Fsc) by the chroma-locked sample rate converter. The measurement of the horizontal line time is preferably done at two levels of accuracy, an integer pixel accuracy and a sub-sample accuracy.

The integer pixel accuracy is preferably done by counting the integer number of pixels that occur between two successive sync edges. The sync edge is presumed to be detected when the data crosses some threshold value. For example, in one embodiment of the present invention, the analog-to-digital converter (ADC) is a 10-bit ADC, i.e., converts an input analog signal into a digital signal with ($2^{79}10-1=1023$) scale levels. In this embodiment, the threshold value is chosen to represent an appropriate slicing level for horizontal sync in the 10-bit number system of the ADC; a typical value for this threshold is 128. The negative peak (or a sync tip) of the digitized video signal normally occurs during the sync pulses. The threshold level would normally be set such that it occurs at approximately the mid-point of the sync pulses. The threshold level may be automatically adapted by the video decoder, or it may be set explicitly via a register or other means.

The horizontal sync tracker preferably detects the horizontal sync edge to a sub-sample accuracy of (1/16)th of a pixel in order to more accurately calculate the sample rate conversion. The incoming samples generally do not include a sample taken exactly at the threshold value for detecting horizontal sync edges. The horizontal sync tracker preferably detects two successive samples, one of which has a value lower than the threshold value and the other of which has a value higher than the threshold value.

After the integer pixel accuracy is determined (sync edge has been detected) the sub-pixel calculation is preferably started. The sync edge of a horizontal sync is generally not a vertical line, but has a slope. In order to remove noise, the video signal goes through a low pass filter. The low pass filter generally decreases sharpness of the transition, i.e., the low pass filter may make the transition from a low level to a high-level last longer.

The horizontal sync tracker preferably uses a sub-sample interpolation technique to obtain an accurate measurement of sync edge location by drawing a straight line between the two successive samples of the horizontal sync signal just above and just below the presumed threshold value to determine where the threshold value has been crossed.

Three values are preferably used to determine the sub-sample accuracy. The three values are the threshold level (T), the value of the sample that crossed the threshold level (V2) and the value of the previous sample that did not cross the threshold level (V1). The sub-sample value is the ratio of (T−V1)/(V2−V1). In the present embodiment a division is not performed. The difference (V2−V1) is divided by 16 to make a variable called DELTA. V1 is then incremented by DELTA until it exceeds the threshold T. The number of times that DELTA is added to V1 in order to make it exceed the threshold (T) is the sub-pixel accuracy in terms of 1/16th of a pixel.

For example, if the threshold value T is presumed to be 146 scale levels, and if the values V1 and V2 of the two successive samples are 140 and 156, respectively, the DELTA is calculated to be 1, and the crossing of the threshold value is determined through interpolation to be six DELTAs away from the first of the two successive samples. Thus, if the sample with value **140** is the nth sample and the sample with the value **156** is the (n+1)th sample, the (n+(6/16))th sample would have had the threshold value. Since the horizontal sync preferably is presumed to be detected at the threshold value of the sync edge, a fractional sample, i.e., 6/16 sample, is added to the number of samples counted between two successive horizontal syncs.

In order to sample rate convert the current number of input pixels Pin to the desired output pixels Pout, the sample rate converter module has a sample rate conversion ratio of Pin/Pout. The sample rate converter module in the preferred embodiment of the line-locked sample rate converter is a polyphase filter with time-varying coefficients. There is a fixed number of phases (I) in the polyphase filter. In the preferred embodiment, the number of phases (I) is 33. The control for the polyphase filter is the decimation rate (d_act) and a reset phase signal. The line measurement Pin is sent to a module that converts it to a decimation rate d_act such that I/d_act (33/d_act) is equal to Pin/Pout. The decimation rate d_act is calculated as follows: d_act=(I/Pout)*Pin.

If the input video line is the standardized length of time and the four times the color sub-carrier is the standardized frequency then Pin will be exactly 910 samples. This gives a sample rate conversion ratio of (858/910). In the present embodiment the number of phases (the interpolation rate) is 33. Therefore the nominal decimation rate for NTSC is 35 (=(33/858)*910). This decimation rate d_act may then be sent to the sample rate converter module. A reset phase signal is sent to the sample rate converter module after the sub-sample calculation has been done and the sample rate converter module starts processing the current video line. In the preferred embodiment, only the active portion of video is processed and sent on to a time base corrector. This results in a savings of memory needed. Only 720 samples of active video are produced as ITU-R BT.601 output sample rates. In other embodiments, the entire horizontal line may be processed and produced as output.

In the preferred embodiment, the calculation of the decimation rate d_act is done somewhat differently from the equation d_act=(I/Pout)*Pin. The results are the same, but there are savings to hardware. The current line length, Pin, will have a relatively small variance with respect to the nominal line length. Pin is nominally 910. It typically varies by less than 62. For NTSC, this variation is less than 5 microseconds. The following calculation is done: d_act ((I/Pout)*(Pin−Pin_nominal))+d_act_nominal

This preferably results in a hardware savings for the same level of accuracy. The difference (Pin−Pin_nominal) may be represented by fewer bits than are required to represent Pin so a smaller multiplier can be used. For NTSC, d_act_nominal is 35 and Pin_nominal is 910. The value (I/Pout) *(Pin−Pin_nominal) may now be called a delta_dec (delta decimation rate) or a second adjustment value.

Therefore, in order to maintain the output sample rate of 858 samples per horizontal line, the conversion rate applied

preferably is 33/(35+delta_dec) where the samples are interpolated by 33 and decimated by (35+delta_dec). A horizontal sync tracker preferably detects horizontal syncs, accurately counts the number of samples between two successive horizontal syncs and generates delta_dec.

If the number of samples between two successive horizontal syncs is greater than 910, the horizontal sync tracker generates a positive delta_dec to keep the output sample rate at 858 samples per horizontal line. On the other hand, if the number of samples between two successive horizontal syncs is less than 910, the horizontal sync tracker generates a negative delta_dec to keep the output sample rate at 858 samples per horizontal line.

For PAL standard video, the horizontal sync tracker generates the delta_dec to keep the output sample rate at 864 samples per horizontal line.

In summary, the position of each horizontal sync pulse is determined to sub-pixel accuracy by interpolating between two successive samples, one of which being immediately below the threshold value and the other being immediately above the threshold value. The number of samples between the two successive horizontal sync pulses is preferably calculated to sub-sample accuracy by determining the positions of two successive horizontal sync pulses, both to sub-pixel accuracy. When calculating delta_dec, the horizontal sync tracker preferably uses the difference between 910 and the number of samples between two successive horizontal syncs to reduce the amount of hardware needed.

In an alternate embodiment, the decimation rate adjustment value, delta_dec, which is calculated for each line, preferably goes through a low pass filter before going to the sample rate converter module. One of the benefits of this method is filtering of variations in the line lengths of adjacent lines where the variations may be caused by noise that affects the accuracy of the measurement of the sync pulse positions.

In another alternative embodiment, the input sample clock is not free running, but is instead line-locked to the input analog video, preferably 27 MHz. The chroma-locked sample rate converter converts the 27 MHz sampled data to a sample rate of four times the color sub-carrier frequency. The analog video signal is demodulated to luma and chroma component video signals, preferably using a comb filter. The luma and chroma component video signals are then sent to the line-locked sample rate converter where they are preferably converted to a sample rate of 13.5 MHz. In this embodiment the 13.5 MHz sample rate at the output may be exactly one-half of the 27 MHz sample rate at the input. The conversion ratio of the line-locked sample rate converter is preferably exactly one-half of the inverse of the conversion ratio performed by the chroma-locked sample rate converter.

Referring to FIG. **21**, the line-locked SRC **704** preferably includes an SRC **770** which preferably is a polyphase filter with time varying coefficients. The number of phases is preferably fixed at 33 while the nominal decimation rate is 35. In other words, the conversion ratio used is preferably 33/(35+delta_dec) where delta_dec may be positive or negative. The delta_dec is a second adjustment value, which is used to adjust the decimation rate of the second sample rate converter. Preferably, the actual decimation rate and phase are automatically adjusted for each horizontal line so that the number of samples per horizontal line is 858 (720 active Y samples and 360 active U and V samples) and the phase of the active video samples is aligned properly with the horizontal sync signals.

In the preferred embodiment, the decimation (down-sampling) rate of the SRC is preferably controlled by a horizontal sync tracker **772**. Preferably, the horizontal sync tracker adjusts the decimation rate once per horizontal line in order to result in a correct number and phase of samples in the interval between horizontal syncs. The horizontal sync tracker preferably provides the adjusted decimation rate to the SRC **770** to adjust the conversion ratio. The decimation rate is preferably calculated to achieve a sub-sample accuracy of 1/16. Preferably, the line-locked SRC **704** also includes a YUV scaler **780** to scale YUV components to the proper amplitudes required by ITU-R BT.601.

The time base corrector (TBC) preferably synchronizes the samples having the line-locked sample rate of nominally 13.5 MHz to the display clock that runs nominally at 13.5 MHz. Since the samples at the output of the TBC are synchronized to the display clock, passthrough video may be provided to the video compositor without being captured first.

To produce samples at the sample rate of nominally 13.5 MHz, the composite video may be sampled in any conventional way with a clock rate that is generally used in the art. Preferably, the composite video is sampled initially at 27 MHz, down sampled to the sample rate of 14.318 MHz by the chroma-locked SRC, and then down sampled to the sample rate of nominally 13.5 MHz by the line-locked SRC. During conversion of the sample rates, the video decoder uses for timing the 27 MHz clock that was used for input sampling. The 27 MHz clock, being free-running, is not locked to the line rate nor to the chroma frequency of the incoming video.

In the preferred embodiment, the decoded video samples are stored in a FIFO the size of one display line of active video at 13.5 MHz, i.e., 720 samples with 16 bits per sample or 1440 bytes. Thus, the maximum delay amount of this FIFO is one display line time with a normal, nominal delay of one-half a display line time.

In the preferred embodiment, video samples are outputted from the FIFO at the display clock rate that is nominally 13.5 MHz. Except for vertical syncs of the input video, the display clock rate is unrelated to the timing of the input video. In alternate embodiments, larger or smaller FIFOs may be used.

Even though the effective sample rate and the display clock rate are both nominally 13.5 MHz the rate of the sampled video entering the FIFO and the display rate are generally different. This discrepancy is due to differences between the actual frequencies of the effective input sample rate and the display clock. For example, the effective input sample rate is nominally 13.5 MHz but it is locked to operate at 858 times the line rate of the video input, while the display clock operates nominally at 13.5 MHz independently of the line rate of the video input.

Since the rates of data entering and leaving the FIFO are typically different, the FIFO will tend to either fill up or become empty, depending on relative rates of the entering and leaving data. In one embodiment of the present invention, video is displayed with an initial delay of one-half a horizontal line time at the start of every field. This allows the input and output rates to differ up to the point where the input and output horizontal phases may change by up to one-half a horizontal line time without causing any glitches at the display.

The FIFO is preferably filled up to approximately one-half full during the first active video line of every field prior to taking any output video. Thus, the start of each display field follows the start of every input video field by a fixed delay that is approximately equal to one-half the amount of

time for filling the entire FIFO. As such, the initial delay at the start of every field is one-half a horizontal line time in this embodiment, but the initial delay may be different in other embodiments.

Referring to FIG. **22**, the time base corrector (TBC) **72** includes a TBC controller **164** and a FIFO **166**. The FIFO **166** receives an input video **714** at nominally 13.5 MHz locked to the horizontal line rate of the input video and outputs a delayed input video as an output video **716** that is locked to the display clock that runs nominally at 13.5 MHz. The initial delay between the input video and the delayed input video is half a horizontal line period of active video, e.g., 53.5 μs per active video in a horizontal line/2=26.75 μs for NTSC standard video.

The TBC controller **164** preferably generates a vertical sync (VSYNC) for display that is delayed by one-half a horizontal line from an input VSYNC. The TBC controller **164** preferably also generates timing signals such as NTSC or PAL standard timing signals. The timing signals are preferably derived from the VSYNC generated by the TBC controller and preferably include horizontal sinc. The timing signals are not affected by the input video, and the FIFO is read out synchronously to the timing signals. Data is read out of the FIFO according to the timing at the display side while the data is written into the FIFO according to the input timing. A line reset resets the FIFO write pointer to signal a new line. A read pointer controlled by the display side is updated by the display timing.

As long as the accumulated change in FIFO fullness, in either direction, is less than one-half a video line, the FIFO will generally neither underflow nor overflow during the video field. This ensures correct operation when the display clock frequency is anywhere within a fairly broad range centered on the nominal frequency. Since the process is repeated every field, the FIFO fullness changes do not accumulate beyond one field time.

Referring to FIG. **23**, a flow diagram of a process using the TBC **72** is illustrated. The process resets in step **782** at system start up. The system preferably checks for vertical sync (VSYNC) of the input video in step **784**. After receiving the input VSYNC, the system in step **786** preferably starts counting the number of incoming video samples. The system preferably loads the FIFO in step **788** continuously with the incoming video samples. While the FIFO is being loaded, the system in step **790** checks if enough samples have been received to fill the FIFO up to a half full state.

When enough samples have been received to fill the FIFO to the half full state, the system in step **792** preferably generates timing signals including horizontal sync to synchronize the output of the TBC to the display clock. The system in step **794** preferably outputs the content of the FIFO continuously in sync with the display clock. The system in step **796** preferably checks for another input VSYNC. When another input vertical sync is detected, the process starts counting the number of input video samples again and starts outputting output video samples when enough input video samples have been received to make the FIFO half full.

In other embodiments of the present invention, the FIFO size may be smaller or larger. The minimum size acceptable is determined by the maximum expected difference in the video source sample rate and the display sample rate. Larger FIFOs allow for greater variations in sample rate timing, however at greater expense. For any chosen FIFO size, the logic that generates the sync signal that initiates display video fields should incur a delay from the input video timing of one-half the delay of the entire FIFO as described above. However, it is not required that the delay be one-half the delay of the entire FIFO.

IX. Video Scaler

In certain applications of graphics and video display hardware, it may be necessary or desirable to scale the size of a motion video image either upwards or downwards. It may also be desirable to minimize memory usage and memory bandwidth demands. Therefore it is desirable to scale down before writing to memory, and to scale up after reading from memory, rather than the other way around in either case. Conventionally there is either be separate hardware to scale down before writing to memory and to scale up after reading from memory, or else all scaling is done in one location or the other, such as before writing to memory, even if the scaling direction is upwards.

In the preferred embodiment, a video scaler performs both scaling-up and scaling-down of either digital video or digitized analog video. The video scaler is preferably configured such that it can be used for either scaling down the size of video images prior to writing them to memory or for scaling up the size of video images after reading them from memory. The size of the video images are preferably downscaled prior to being written to memory so that the memory usage and the memory bandwidth demands are minimized. For similar reasons, the size of the video images are preferably upscaled after reading them from memory.

In the former case, the video scaler is preferably in the signal path between a video input and a write port of a memory controller. In the latter case, the video scaler is preferably in the signal path between a read port of the memory controller and a video compositor. Therefore, the video scaler may be seen to exist in two distinct logical places in the design, while in fact occupying only one physical implementation.

This function is preferably achieved by arranging a multiplexing function at the input of the scaling engine, with one input to the multiplexer being connected to the video input port and the other connected to the memory read port. The memory write port is arranged with a multiplexer at its input, with one input to the multiplexer connected to the output of the scaling engine and the other connected to the video input port. The display output port is arranged with a multiplexer at its input, with one connected to the output of the scaling engine and the other input connected to the output of the memory read port.

In the preferred embodiment, there are different clock domains associated with the video input and the display output functions of the chip. The video scaling engine uses a clock that is selected between the video input clock and the display output clock (display clock). The clock selection uses a glitch-free clock selection logic, i.e. a circuit that prevents the creation of extremely narrow clock pulses when the clock selection is changed. The read and write interfaces to memory both use asynchronous interfaces using FIFOs, so the memory clock domain may be distinct from both the video input clock domain and the display output clock domain.

Referring to FIG. **24**, a flow diagram illustrates a process of alternatively upscaling or downscaling the video input **800**. The system in step **802** preferably selects between a downscaling operation and an upscaling operation. If the downscaling operation is selected, the system in step **804** preferably downscales the input video prior to capturing the input video in memory in step **806**. If the upscaling opera-

tion is selected in step **802**, the system in step **806** preferably captures the input video in memory without scaling it.

Then the system in step **808** outputs the downscaled video as downscaled output **810**. The system in step **808**, however, sends non-scaled video in the upscale path to be upscaled in step **812**. The system in step **812** upscales the non-scaled video and outputs it as upscaled video output **814**.

The video pipeline preferably supports up to one scaled video window and one passthrough video window, plus one background color, all of which are logically behind the set of graphics windows. The order of these windows, from back to front, is fixed as background, then passthrough, then scaled video. The video windows are preferably always in YUV format, although they can be in either 4:2:2 or 4:2:0 variants of YUV. Alternatively they can be in RGB or other formats.

When digital video, e.g., MPEG is provided to the graphics display system or when analog video is digitized, the digital video or the digitized analog video is provided to a video compositor using one of three signal paths, depending on processing requirements. The digital video and the digitized analog video are provided to the video compositor as passthrough video over a passthrough path, as upscaled video over an upscale path and a downscaled video over a downscale path.

Either of the digital video or the analog video may be provided to the video compositor as the passthrough video while the other of the digital video or the analog video is provided as an upscaled video or a downscaled video. For example, the digital video may be provided to the video compositor over the passthrough path while, at the same time, the digitized analog video is downscaled and provided to the video compositor over the downscale path as a video window. In one embodiment of the present invention where the scaler engine is shared between the upscale path and the downscale path, the scaler engine may upscale video in either the vertical or horizontal axis while downscaling video in the other axis. However, in this embodiment, an upscale operation and a downscale operation on the same axis are not performed at the same time since only one filter is used to perform both upscaling and downscaling for each axis.

Referring to FIG. **24** a single video scaler **52** preferably performs both the downscaling and upscaling operations. In particular, signals of the downscale path only are illustrated. The video scaler **52** includes a scaler engine **182**, a set of line buffers **178**, a vertical coefficient memory **180A** and a horizontal coefficient memory **180B**. The scaler engine **182** is implemented as a set of two polyphase filters, one for each of horizontal and vertical dimensions.

In one embodiment of the present invention, the vertical polyphase filter is a four-tap filter with programmable coefficients from the vertical coefficient memory **180A**. In other embodiments, the number of taps in the vertical polyphase filter may vary. In one embodiment of the present invention, the horizontal polyphase filter is an eight-tap filter with programmable coefficients from the horizontal coefficient memory **180B**. In other embodiments, the number of taps in the horizontal polyphase filter may vary.

The vertical and the horizontal coefficient memories may be implemented in SRAM or any other suitable memory. Depending on the operation to be performed, e.g. a vertical or horizontal axis, and scaling-up or scaling-down, appropriate filter coefficients are used, respectively, from the vertical and horizontal coefficient memories. Selection of filter coefficients for scaling-up and scaling-down operations are well known in the art.

The set of line buffers **178** are used to provide input of video data to the horizontal and vertical polyphase filters. In this embodiment, three line buffers are used, but the number of the line buffers may vary in other embodiments. In this embodiment, each of the three line buffers is used to provide an input to one of the taps of the vertical polyphase filter with four taps. The input video is provided to the fourth tap of the vertical polyphase filter. A shift register having eight cells in series is used to provide inputs to the eight taps of the horizontal polyphase filter, each cell providing an input to one of the eight taps.

In this embodiment, a digital video signal **820** and a digitized analog signal video **822** are provided to a first multiplexer **168** as first and second inputs. The first multiplexer **168** has two outputs. A first output of the first multiplexer is provided to the video compositor as a pass through video **186**. A second output of the first multiplexer is provided to a first input of a second multiplexer **176** in the downscale path.

In the downscale path, the second multiplexer **176** provides either the digital video or the digitized analog video at the second multiplexer's first input to the video scaler **52**. The video scaler provides a downscaled video signal to a second input of a third multiplexer **162**. The third multiplexer provides the downscaled video to a capture FIFO **158** which stores the captured downscaled video. The memory controller **126** takes the captured downscaled video and stores it as a captured downscaled video image into a video FIFO **148**. An output of the video FIFO is coupled to a first input of a fourth multiplexer **188**. The fourth multiplexer provides the output of the video FIFO, which is the captured downscaled video image, as an output **824** to the graphics compositor, and this completes the downscale path. Thus, in the downscale path, either the digital video or the digitized analog video is downscaled first, and then captured.

FIG. **26** is similar to FIG. **25**, but in FIG. **26**, signals of the upscale path are illustrated. In the upscale path, the third multiplexer **162** provides either the digital video **820** or the digitized analog video **822** to the capture FIFO **158** which captures and stores input as a captured video image. This captured video image is provided to the memory controller **126** which takes it and provides to the video FIFO **148** which stores the captured video image.

An output of the video FIFO **148** is provided to a second input of the second multiplexer **176**. The second multiplexer provides the captured video image to the video scaler **52**. The video scaler scales up the captured video image and provides it to a second input of the fourth multiplexer **188** as an upscaled captured video image. The fourth multiplexer provides the upscaled captured video image as the output **824** to the video compositor. Thus, in the upscale path, either the digital video or the digitized analog video is captured first, and then upscaled.

Referring to FIG. **27**, FIG. **27** is similar to FIG. **25** and FIG. **26**, but in FIG. **27**, signals of both the upscale path and the downscale path are illustrated.

X. Blending of Graphics and Video Surfaces

The graphics display system of the present invention is capable of processing an analog video signal, a digital video signal and graphics data simultaneously. In the graphics display system, the analog and digital video signals are processed in the video display pipeline while the graphics data is processed in the graphics display pipeline. After the processing of the video signals and the graphics data have been completed, they are blended together at a video com-

positor. The video compositor receives video and graphics data from the video display pipeline and the graphics display pipeline, respectively, and outputs to the video encoder ("VEC").

The system may employ a method of compositing a plurality of graphics images and video, which includes blending the plurality of graphics images into a. blended graphics image, combining a plurality of alpha values into a plurality of composite alpha values, and blending the blended graphics image and the video using the plurality of composite alpha values.

Referring to FIG. **28**, a flow diagram of a process of blending video and graphics surfaces is illustrated. The graphics display system resets in step **902**. In step **904**, the video compositor blends the passthrough video and the background color with the scaled video window, using the alpha value which is associated with the scaled video window. The result of this blending operation is then blended with the output of the graphics display pipeline. The graphics output has been pre-blended in the graphics blender in step **904** and filtered in step **906**, and blended graphics contain the correct alpha value for multiplication by the video output. The output of the video blend function is multiplied by the video alpha which is obtained from the graphics pipeline and the resulting video and graphics pixel data stream are added together to produce the final blended result.

In general, during blending of different layers of graphics and/or video, every layer {L1, L2, L3 . . . Ln}, where L1 is the back-most layer, each layer is blended with the composition of all of the layers behind it, beginning with L2 being blended on top of L1. The intermediate result R(i) from the blending of pixels P(i) of layer L(i) over the pixels P(i−1) of layer L(i−1) using alpha value A(i) is: R(i)=A(i)*P(i)+(1−A(i))*P(i−1)

The alpha values {A(i)} are in general different for every layer and for every pixel of every layer. However, in some important applications, it is not practical to apply this formula directly, since some layers may need to be processed in spatial dimensions (e.g. 2 dimensional filtering or scaling) before they can be blended with the layer or layers behind them. While it is generally possible to blend the layers first and then perform the spatial processing, that would result in processing the layers that should not be processed if these layers are behind the subject layer that is to be processed. Processing of the layers that are not to be processed may be undesirable.

Processing the subject layer first would generally require a substantial amount of local storage of the pixels in the subject layer, which may be prohibitively expensive. This problem is significantly exacerbated when there are multiple layers to be processed in front of one or more layers that are not to be processed. In order to implement the formula above directly, each of the layers would have to be processed first, i.e. using their own local storage and individual processing, before they could be blended with the layer behind.

In the preferred embodiment, rather than blending all the layers from back to front, all of the layers that are to be processed (e.g. filtered) are layered together first, even if there is one or more layers behind them over which they should be blended, and the combined upper layers are then blended with the other layers that are not to be processed. For example, layers {1, 2 and 3} may be layers that are not to be processed, while layers {4, 5, 6, 7, and 8} may be layers that are to undergo processing, while all 8 layers are to be blended together, using {A(i)} values that are independent for every layer and pixel. The layers that are to be filtered, upper layers, may be the graphics windows. The lower layers may include the video window and passthrough video.

In the preferred embodiment, all of the layers that are to be filtered (referred to as "upper" layers) are blended together from back to front using a partial blending operation. In an alternate embodiment, two or more of the upper layers may be blended together in parallel. The back-most of the upper layers is not in general the back-most layer of the entire operation.

In the preferred embodiment, at each stage of the blending, an intermediate alpha value is maintained for later use for blending with the layers that are not to be filtered (referred to as the "lower" layers).

The formula that represents the preferred blending scheme is:

$$R(i)=A(i)*P(i)+(1-A(i))*P(i-1)$$

and

$$AR(i)=AR(i-1)*(1-A(i))$$

where R(i) represents the color value of the resulting blended pixel, P(i) represents the color value of the current pixel, A(i) represents the alpha value of the current pixel, P(i−1) represents the value at the location of the current pixel of the composition of all of the upper layers behind the current pixel, initially this represents black before any layers are blended, AR(i) is the alpha value resulting from each instance of this operation, and AR(i−1) represents the intermediate alpha value at the location of the current pixel determined from all of the upper layers behind the current pixel, initially this represents transparency before any layers are blended. AR represents the alpha value that will subsequently be multiplied by the lower layers as indicated below, and so an AR value of 1 (assuming alpha ranges from 0 to 1) indicates that the current pixel is transparent and the lower layers will be fully visible when multiplied by 1.

In other words, in the preferred embodiment, at each stage of blending the upper layers, the pixels of the current layer are blended using the current alpha value, and also an intermediate alpha value is calculated as the product (1−A(i))*(AR(i−1)). The key differences between this and the direct evaluation of the conventional formula are: (1) the calculation of the product of the set of {(1−A(i))} for the upper layers, and (2) a virtual transparent black layer is used to initialize the process for blending the upper layers, since the lower layers that would normally be blended with the upper layers are not used at this point in this process.

The calculation of the product of the sets of {(1−A(i)} for the upper layers is implemented, in the preferred embodiment, by repeatedly calculating AR(i)=AR(i−1)*(1−A(i)) at each layer, such that when all layers {i} have been processed, the result is that AR the product of all (1−A(i)) values for all upper layers. Alternatively in other embodiments, the composite alpha value for each pixel of blended graphics may be calculated directly as the product of all (1−alpha value of the corresponding pixel of the graphics image on each layer)'s without generating an intermediate alpha at each stage.

To complete the blending process of the entire series of layers, including the upper and lower layers, once the upper layers have been blended together as described above, they may be processed as desired and then the result of this processing, a composite intermediate image, is blended with the lower layer or layers. In addition, the resulting alpha

values preferably are also processed in essentially the same way as the image components. The lower layers can be blended in the conventional fashion, so at some point there can be a single image representing the lower layers. Therefore two images, one representing the upper layers and one representing the lower layers can be blended together. In this operation, the AR(n) value at each pixel that results from the blending of the upper layers and any subsequent processing is used to be multiplied with the composite lower layer.

Mathematically this latter operation is as follows: let L(u) be the composite upper layer resulting from the process described above and after any processing, let AR(u) be the composite alpha value of the upper layers resulting from the process above and after any processing, let L(1) be the composite lower layer that results from blending all lower layers in the conventional fashion and after any processing, and let Result be the final result of blending all the upper and lower layers, after any processing. Then, Result=L(u)+AR(u)*L(1). L(u) does not need to be multiplied by any additional alpha values, since all such multiplication operations were already performed at an earlier stage.

In the preferred embodiment, a series of images makes up the upper layers. These are created by reading pixels from memory, as in a conventional graphics display device. Each pixel is converted into a common format if it is not already in that format; in this example the YUV format is used. Each pixel also has an alpha value associated with it. The alpha values can come from a variety of sources, including (1) being part of the pixel value read from memory (2) an element in a color look-up table (CLUT) in cases where the pixel format uses a CLUT (3) calculated from the pixel color value, e.g. alpha as a function of Y, (4) calculated using a keying function, i.e. some pixel values are transparent (i.e. alpha=0) and others are opaque (alpha=1) based on a comparison of the pixel value with a set of reference values, (5) an alpha value may be associated with a region of the image as described externally, such as a rectangular region, described by the four corners of the rectangle, may have a single alpha value associated with it, or (6) some combination of these.

The upper layers are preferably composited in memory storage buffers called line buffers. Each line buffer preferably is sized to contain pixels of one scan line. Each line buffer has an element for each pixel on a line, and each pixel in the line buffer has elements for the color components, in this case Y, U and V, and one for the intermediate alpha value AR. Before compositing of each line begins, the appropriate line buffer is initialized to represent a transparent black having already been composited into the buffer; that is, the YUV value is set to the value that represents black (i.e. Y=0, U=V=128) and the alpha value AR is set to represent (1−transparent)=(1−0)=1.

Each pixel of the current layer on the current line is combined with the value pre-existing in the line buffer using the formulas already described, i.e.,

$$R(i)=A(i)*P(i)+(1-A(i)*P(i-1)$$

and

$$AR(i)=AR(i-1)*(1-A(i)).$$

In other words, the color value of the current pixel P(i) is multiplied by its alpha value A(i), and the pixel in the line buffer representing the same location on the line P(i−1) is read from the line buffer, multiplied by (1−A(i)), and added to the previous result, producing the resulting pixel value R(i). Also, the alpha value at the same location in the line buffer (AR(i−1)) is read from the buffer and multiplied by (1−A(i)), producing AR(i). The results R(i) and AR(i) are then written back to the line buffer in the same location.

When multiplying a YUV value by an alpha value between 0 and 1, the offset nature of the U and V values should preferably be accounted for. In other words, U=V=128 represents a lack of color and it is the value that should result from a YUV color value being multiplied by 0. This can be done in at least two ways. In one embodiment of the present invention, 128 is subtracted from the U and V values before multiplying by alpha, and then 128 is added to the result. In another embodiment, U and V values are directly multiplied by alpha, and it is ensured that at the end of the entire compositing process all of the coefficients multiplied by U and V sum to 1, so that the offset 128 value is not distorted significantly.

Each of the layers in the group of upper layers is preferably composited into a line buffer starting with the backmost of the upper layers and progressing towards the front until the front-most of the upper layers has been composited into the line buffer. In this way, a single hardware block, i.e., the display engine, may be used to implement the formula above for all of the upper layers. In this arrangement, the graphics compositor engine preferably operates at a clock frequency that is substantially higher than the pixel display rate. In one embodiment of the present invention, the graphics compositor engine operates at 81 MHz while the pixel display rate is 13.5 MHz.

This process repeats for all of the lines in the entire image, starting at the top scan line and progressing to the bottom. Once the compositing of each scan line into a line buffer has been completed, the scan line becomes available for use in processing such as filtering or scaling. Such processing may be performed while subsequent scan lines are being composited into other line buffers. Various processing operations may be selected such as anti-flutter filtering and vertical scaling.

In alternative embodiments more than one graphics layer may be composited simultaneously, and in some such embodiments it is not necessary to use line buffers as part of the compositing process. If all upper layers are composited simultaneously, the combination of all upper layers can be available immediately without the use of intermediate storage.

Referring to FIG. **29**, a flow diagram of a process of blending graphics windows is illustrated. The system preferably resets in step **920**. In step **922**, the system preferably checks for a vertical sync (VSYNC). If a VSYNC has been received, the system in step **924** preferably loads a line from the bottom most graphics window into a graphics line buffer. Then the system in step **926** preferably blends a line from the next graphics window into the line buffer. Then the system in step **928** preferably determines if the last graphics window visible on a current display line has been blended. If the last graphics window has not been blended, the system continues on with the blending system in step **926**.

If the last window of the current display line has been reached, the system preferably checks in step **930** to determine if the last graphics line of a current display field has been blended. If the last graphics line has been blended, the system awaits another VSYNC in step **922**. If the last graphics line has not been blended, the system goes to the next display line in step **932** and repeats the blending process.

Referring to FIG. **30**, a flow diagram of a process of receiving blended graphics **950**, a video window **952** and a passthrough video **954** and blending them. A background

color preferably is also blended in one embodiment of the present invention. As step **956** indicates, the video compositor preferably displays each pixel as they are composited without saving pixels to a frame buffer or other memory.

When the video signals and graphics data are blended in the video compositor, the system in step **958** preferably displays the passthrough video **954** outside the active window area first. There are 525 scan lines in each frame and 858 pixels in each scan line of NTSC standard television signals, when a sample rate of 13.5 MHz is used, per ITU-R Bt.601. An active window area of the NTSC standard television is inside an NTSC frame. There are 625 scan lines per frame and 864 pixels in each scan line of PAL standard television, when using the ITU-R Bt.601 standard sample rate of 13.5 MHz. An active window area of the PAL standard television is inside a PAL frame.

Within the active window area, the system in step **960** preferably blends the background color first. On top of the background color, the system in step **962** preferably blends the portion of the passthrough video that falls within the active window area. On top of the passthrough window, the system in step **964** preferably blends the video window. Finally, the system in step **968** blends the graphics window on top of the composited video window and outputs composited video **970** for display.

Interlaced displays, such as televisions, have an inherent tendency to display an apparent vertical motion at the horizontal edges of displayed objects, with horizontal lines, and on other points on the display where there is a sharp contrast gradient along the vertical axis. This apparent vertical motion is variously referred to as flutter, flicker, or judder.

While some image elements can be designed specifically for display on interlaced TVs or filtered before they are displayed, when multiple such image objects are combined onto one screen, there are still visible flutter artifacts at the horizontal top and bottom edges of these objects. While it is also possible to include filters in hardware to minimize visible flutter of the display, such filters are costly in that they require higher memory bandwidth from the display memory, since both even and odd fields should preferably be read from memory for every display field, and they tend to require additional logic and memory on-chip.

One embodiment of the present invention includes a method of reducing interlace flutter via automatic blending. This method has been designed for use in graphics displays device that composites visible objects directly onto the screen; for example, the device may use windows, window descriptors and window descriptor lists, or similar mechanisms. The top and bottom edges (first and last scan lines) of each object (or window) are displayed such that the alpha blend value (alpha blend factor) of these edges is adjusted to be one-half of what it would be if these same lines were not the top and bottom lines of the window.

For example, a window may constitute a rectangular shape, and the window may be opaque, i.e. it's alpha blend factor is 1, on a scale of 0 to 1. All lines on this window except the first and last are opaque when the window is rendered. The top and bottom lines are adjusted so that, in this case, the alpha blend value becomes 0.5, thereby causing these lines to be mixed 50% with the images that are behind them. This function occurs automatically in the preferred implementation. Since in the preferred implementation, windows are rectangular objects that are rendered directly onto the screen, the locations of the top and bottom lines of every window are already known.

In one embodiment, the function of dividing the alpha blend values for the top and bottom lines by two is implemented only for the top fields of the interlaced display. In another embodiment, the function of dividing the alpha blend values for the top and bottom lines by two is implemented only for the bottom fields of the interlaced display.

In the preferred embodiment, there exists also the ability to alpha blend each window with the windows behind it, and this alpha value can be adjusted for every pixel, and therefore for every scan line. These characteristics of the application design are used advantageously, as the flutter reduction effect is implemented by controlling the alpha blend function using information that is readily available from the window control logic.

In a specific illustrative example, the window is solid opaque white, and the image behind it is solid opaque black. In the absence of the disclosed method, at the top and bottom edges of the window there would be a sharp contrast between black and white, and when displayed on an interlaced TV, significant flutter would be visible. Using the disclosed method, the top and bottom lines are blended 50% with the background, resulting in a color that is halfway between black and white, or gray. When displayed on an interlaced TV, the apparent visual location of the top and bottom edges of the object is constant, and flutter is not apparent. The same effect applies equally well for other image examples.

The method of reducing interlace flutter of this embodiment does not require any increase in memory bandwidth, as the alternate field (the one not currently being displayed) is not read from memory, and there is no need for vertical filtering, which would have required logic and on-chip memory.

The same function can alternatively be implemented in different graphics hardware designs. For example in designs using a frame buffer (conventional design), graphic objects can be composited into the frame buffer with an alpha blend value that is adjusted to one-half of its normal value at the top and bottom edges of each object. Such blending can be performed in software or in a blitter that has a blending capability.

XI. Anti-Flutter Filtering/Vertical Scaling

In the preferred embodiment, the vertical filtering and anti-flutter filtering are performed on blended graphics by one graphics filter. One function of the graphics filter is low pass filtering in the vertical dimension. The low pass filtering may be performed in order to minimize the "flutter" effect inherent in interlaced displays such as televisions. The vertical downscaling or upscaling operation may be performed in order to change the pixel aspect ratio from the square pixels that are normal for computer, Internet and World Wide Web content into any of the various oblong aspect ratios that are standard for televisions as specified in ITU-R 601B. In order to be able to perform vertical scaling of the upper layers the system preferably includes seven line buffers. This allows for four line buffers to be used for filtering and scaling, two are available for progressing by one or two lines at the end of every line, and one for the current compositing operation.

When scaling or filtering are performed, the alpha values in the line buffers are filtered or scaled in the same way as the YUV values, ensuring that the resulting alpha values correctly represent the desired alpha values at the proper location. Either or both of these operations, or neither, or other processing, may be performed on the contents of the line buffers.

Once the optional processing of the contents of the line buffers has been completed, the result is the completed set

of upper layers with the associated alpha value (product of (1−A(i)). These results are used directly for compositing the upper layers with the lower layers, using the formula: Result=L(u)−AR(u)*L(1) as explained in detail in reference to blending of graphics and video. If the lower layers require any processing independent of processing required for the upper layers or for the resulting image, the lower layers are processed before being combined with the upper layers; however in one embodiment of the present invention, no such processing is required.

Each of the operations described above is preferably implemented digitally using conventional ASIC technology. As part of the normal ASIC technology the logical operations are segmented into pipeline stages, which may require temporary storage of logic values from one clock cycle to the next. The choice of how many pipeline stages are used in each of the operations described above is dependent on the specific ASIC technology used, the clock speed chosen, the design tools used, and the preference of the designer, and may vary without loss of generality. In the preferred embodiment the line buffers are implemented as dual port memories allowing one read and one write cycle to occur simultaneously, facilitating the read and write operations described above while maintaining a clock frequency of 81 MHz. In this embodiment the compositing function is divided into multiple pipeline stages, and therefore the address being read from the memory is different from the address being written to the same memory during the same clock cycle.

Each of the arithmetic operations described above in the preferred embodiment use 8 bit accuracy for each operand; this is generally sufficient for providing an accurate final result. Products are rounded to 8 bits before the result is used in subsequent additions.

Referring to FIG. **31**, a block diagram illustrates an interaction between the line buffers **504** and a graphics filter **172**. The line buffers comprises a set of line buffers **1–7 506** *a–g*. The line buffers are controlled by a graphics line buffer controller over a line buffer control interface **502**. In one embodiment of the present invention, the graphics filter is a four-tap polyphase filter, so that four lines of graphics data **516***a–d* are provided to the graphics filter at a time. The graphics filter **172** sends a line buffer release signal **516***e* to the line buffers to notify that one to three line buffers are available for compositing additional graphics display lines.

In another embodiment, line buffers are not used, but rather all of the upper layers are composited concurrently. In this case, there is one graphics blender for each of the upper layers active at any one pixel, and the clock rate of the graphics blender may be approximately equal to the pixel display rate. The clock rate of the graphics blenders may be somewhat slower or faster, if FIFO buffers are used at the output of the graphics blenders.

The mathematical formulas implemented are the same as in the first embodiment described. The major difference is that instead of performing the compositing function iteratively by reading and writing a line buffer, all layers are composited concurrently and the result of the series of compositor blocks is immediately available for processing, if required, and for blending with the lower layers, and line buffers are not used for purposes of compositing.

Line buffers may still be needed in order to implement vertical filtering or vertical scaling, as those operations typically require more than one line of the group of upper layers to be available simultaneously, although fewer line buffers are generally required here than in the preferred embodiment. Using multiple graphics blenders operating at approximately the pixel rate simplifies the implementation in applications where the pixel rate is relatively fast for the ASIC technology used, for example in HDTV video and graphics systems where the pixel rate is 74.25 MHz.

XII. Unified Memory Architecture/Real Time Scheduling

Recently, improvements to memory fabrication technologies have resulted in denser memory chips. However memory chip bandwidth has not been increasing as rapidly. The bandwidth of a memory chip is a measure of how fast contents of the memory chip can be accessed for reading or writing. As a result of increased memory density without necessarily a commensurate increase in bandwidth, in many conventional system designs multiple memory devices are used for different functions, and memory space in some memory modules may go unused or is wasted. In the preferred embodiment, a unified memory architecture is used. In the unified memory architecture, all the tasks (also referred to as "clients"), including CPU, display engine and IO devices, share the same memory.

The unified memory architecture preferably includes a memory that is shared by a plurality of devices, and a memory request arbiter coupled to the memory, wherein the memory request arbiter performs real time scheduling of memory requests from different devices having different priorities. The unified memory system assures real time scheduling of tasks, some of which do not inherently have pre-determined periodic behavior and provides access to memory by requesters that are sensitive to latency and do not have determinable periodic behavior.

In an alternate embodiment, two memory controllers are used in a dual memory controller system. The memory controllers may be 16-bit memory controllers or 32-bit memory controllers. Each memory controller can support different configuration of SDRAM device types and banks, or other forms of memory besides SDRAM. A first memory space addressed by a first memory controller is preferably adjacent and contiguous to a second memory space addressed by a second memory controller so that software applications view the first and second memory spaces as one continuous memory space. The first and the second memory controllers may be accessed concurrently by different clients. The software applications may be optimized to improve performance.

For example, a graphics memory may be allocated through the first memory controller while a CPU memory is allocated through the second memory controller. While a display engine is accessing the first memory controller, a CPU may access the second memory controller at the same time. Therefore, a memory access latency of the CPU is not adversely affected in this instance by memory being accessed by the display engine and vice versa. In this example, the CPU may also access the first memory controller at approximately the same time that the display engine is accessing the first memory controller, and the display controller can access memory from the second memory controller, thereby allowing sharing of memory across different functions, and avoiding many copy operations that may otherwise be required in conventional designs.

Referring to FIG. **32**, a dual memory controller system services memory requests generated by a display engine **1118**, a CPU **1120**, a graphics accelerator **1124** and an input/output module **1126** are provided to a memory select

block **1100**. The memory select block **1100** preferably routes the memory requests to a first arbiter **1102** or to a second arbiter **1106** based on the address of the requested memory. The first arbiter **1102** sends memory requests to a first memory controller **1104** while the second arbiter **1106** sends memory requests to a second memory controller **1108**. The design of arbiters for handling requests from tasks with different priorities is well known in the art.

The first memory controller preferably sends address and control signals to a first external SDRAM and receives a first data from the first external SDRAM. The second memory controller preferably sends address and control signals to a second external SDRAM and receives a second data from the second external SDRAM.

The first and second memory controllers preferably provide first and second data received, respectively, from the first and second external SDRAMs to a device that requested the received data.

The first and second data from the first and second memory controllers are preferably multiplexed, respectively, by a first multiplexer **1110** at an input of the display engine, by a second multiplexer **1112** at an input of the CPU, by a third multiplexer **1114** at an input of the graphics accelerator and by a fourth multiplexer **1116** at an input of the I/O module. The multiplexers provide either the first or the second data, as selected by memory select signals provided by the memory select block, to a corresponding device that has requested memory.

An arbiter preferably uses an improved form of real time scheduling to meet real-time latency requirements while improving performance for latency-sensitive tasks. First and second arbiters may be used with the flexible real time scheduling. The real time scheduling is preferably implemented on both the first arbiter and the second arbiter independently.

When using a unified memory, memory latencies caused by competing memory requests by different tasks should preferably be addressed. In the preferred embodiment, a real-time scheduling and arbitration scheme for unified memory is implemented, such that all tasks that use the unified memory meet their real-time requirements. With this innovative use of the unified memory architecture and real-time scheduling, a single unified memory is provided to the CPU and other devices of the graphics display system without compromising quality of graphics or other operations and while simultaneously minimizing the latency experienced by the CPU.

The methodology used preferably implements real-time scheduling using Rate Monotonic Scheduling ("RMS") . It is a mathematical approach that allows the construction of provably correct schedules of arbitrary numbers of real-time tasks with arbitrary periods for each of the tasks. This methodology provides for a straight forward means for proof by simulation of the worst case scenario, and this simulation is simple enough that it can be done by hand. RMS, as normally applied, makes a number of simplifying assumptions in the creation of a priority list.

In the normal RMS assumptions, all tasks are assumed to have constant periods, such that a request for service is made by the task with stated period, and all tasks have a latency tolerance that equals that task's period. Latency tolerance is defined as the maximum amount of time that can pass from the moment the task requests service until that task's request has been completely satisfied. During implementation of one embodiment of the present invention, the above assumptions have been modified, as described below.

In the RMS method, all tasks are generally listed along with their periods. They are then ordered by period, from the shortest to the longest, and priorities are assigned in that order. Multiple tasks with identical periods can be in any relative order. In other words, the relative order amongst them can be decided by, for example, flipping a coin.

Proof of correctness, i.e. the guarantee that all tasks meet their deadlines, is constructed by analyzing the behavior of the system when all tasks request service at exactly the same time; this time is called the "critical instant". This is the worst case scenario, which may not occur in even a very large set of simulations of normal operation, or perhaps it may never occur in normal operation, however it is presumed to be possible. As each task is serviced, it uses the shared resource, memory clock cycles in the present invention, in the degree stated by that task. If all tasks meet their deadlines, the system is guaranteed to meet all tasks' deadlines under all conditions, since the critical instant analysis simulates the worst case.

When the lowest priority real-time task meets its deadline, without any higher priority tasks missing their deadlines, then all tasks are proven to meet their deadlines. As soon as any task in this simulation fails to meet its deadline, the test has failed and the task set cannot be guaranteed, and therefore the design should preferably be changed in order to guarantee proper operation under worst case conditions.

In the RMS methodology, real-time tasks are assumed to have periodic requests, and the period and the latency tolerance are assumed to have the same value. Since the requests may not be in fact periodic, it is clearer to speak in terms of "minimum interval" rather than period. That is, any task is assumed to be guaranteed not to make two consecutive requests with an interval between them that is any shorter than the minimum interval.

The deadline, or the latency tolerance, is the maximum amount of time that may pass between the moment a task makes a request for service and the time that the service is completed, without impairing the function of the task. For example, in a data path with a constant rate source (or sink), a FIFO, and memory access from the FIFO, the request may occur as soon as there is enough data in the FIFO that if service is granted immediately the FIFO does not underflow (or overflow in case of a read operation supporting a data sink). If service is not completed before the FIFO overflows (or underflows in the case of a data sink) the task is impaired.

In the RMS methodology, those tasks that do not have specified real-time constraints are preferably grouped together and served with a single master task called the "sporadic server", which itself has the lowest priority in the system. Arbitration within the set of tasks served by the sporadic server is not addressed by the RMS methodology, since it is not a real-time matter. Thus, all non-real-time tasks are served whenever there is resource available, however the latency of serving any one of them is not guaranteed.

To implement real-time scheduling based on the RMS methodology, first, all of the tasks or clients that need to access memory are preferably listed, not necessarily in any particular order. Next, the period of each of the tasks is preferably determined. For those with specific bandwidth requirements (in bytes per second of memory access), the period is preferably calculated from the bandwidth and the burst size. If the deadline is different from the period for any given task, that is listed as well. The resource requirement when a task is serviced is listed along with the task. In this case, the resource requirement is the number of memory

clock cycles required to service the memory access request. The tasks are sorted in order of increasing period, and the result is the set of priorities, from highest to lowest. If there are multiple tasks with the same period, they can be given different, adjacent priorities in any random relative order within the group; or they can be grouped together and served with a single priority, with round-robin arbitration between those tasks at the same priority.

In practice, the tasks sharing the unified memory do not all have true periodic behavior. In one embodiment of the present invention, a block out timer, associated with a task that does not normally have a period, is used in order to force a bounded minimum interval, similar to a period, on that task. For example a block out timer associated with the CPU has been implemented in this embodiment. If left uncontrolled, the CPU can occupy all available memory cycles, for example by causing a never-ending stream of cache misses and memory requests. At the same time, CPU performance is determined largely by "average latency of memory access", and so the CPU performance would be less than optimal if all CPU memory accessed were consigned to a sporadic server, i.e., at the lowest priority.

In this embodiment, the CPU task has been converted into two logical tasks. A first CPU task has a very high priority for low latency, and it also has a block out timer associated with it such that once a request by the CPU is made, it cannot submit a request again until the block out timer has timed out. In this embodiment, the CPU task has the top priority. In other embodiments, the CPU task may have a very high priority but not the top priority. The timer period has been made programmable for system tuning, in order to accommodate different system configurations with different memory widths or other options.

In one embodiment of the present invention, the block out timer is started when the CPU makes a high priority request. In another embodiment, the block out timer is started when the high priority request by the CPU is serviced. In other embodiments, the block out timer may be started at any time in the interval between the time the high priority request is made and the time the high priority request is serviced.

A second CPU task is preferably serviced by a sporadic server in a round-robin manner. Therefore if the CPU makes a long string of memory requests, the first one is served as a high priority task, and subsequent requests are served by the low priority sporadic server whenever none of the real-time tasks have requests pending, until the CPU block out timer times out. In one embodiment of the present invention, the graphics accelerator and the display engine are also capable of requesting more memory cycles than are available, and so they too use similar block out timer.

For example, the CPU read and write functions are grouped together and treated as two tasks. A first task has a theoretical latency bound of 0 and a period that is programmable via a block out timer, as described above. A second task is considered to have no period and no deadline, and it is grouped into the set of tasks served by the sporadic server via a round robin at the lowest priority. The CPU uses a programmable block out timer between high priority requests in this embodiment.

For another example, a graphics display task is considered to have a constant bandwidth of 27 MB/s, i.e., 16 bits per pixel at 13.5 MHz. However, the graphics bandwidth in one embodiment of the present invention can vary widely from much less than 27 MB/s to a much greater figure, but 27 MB/s is a reasonable figure for assuring support of a range of applications. For example, in one embodiment of the present invention, the graphics display task utilizes a block out timer that enforces a period of 2.37 μs between high priority requests, while additional requests are serviced on a best-effort basis by the sporadic server in a low priority round robin manner.

Referring to FIG. **33**, a block diagram illustrates an implementation of a real-time scheduling using an RMS methodology. A CPU service request **1138** is preferably coupled to an input of a block out timer **1130** and a sporadic server **1136**. An output of the block out timer **1130** is preferably coupled to an arbiter **1132** as a high priority service request. Tasks **1–5 1134***a–e* may also be coupled to the arbiter as inputs. An output of the arbiter is a request for service of a task that has the highest priority among all tasks that have a pending memory request.

In FIG. **33**, only the CPU service request **1138** is coupled to a block out timer. In other embodiments, service requests from other tasks may be coupled to their respective block out timers. The block out timers are used to enforce a minimum interval between two successive accesses by any high priority task that is non-periodic but may require expedited servicing. Two or more such high priority tasks may be coupled to their respective block out timers in one embodiment of the present invention. Devices that are coupled to their respective block out timers as high priority tasks may include a graphics accelerator, a display engine, and other devices.

In addition to the CPU request **1138**, low priority tasks **1140***a–d* may be coupled to the sporadic server **1136**. In the sporadic server, these low priority tasks are handled in a round robin manner. The sporadic server sends a memory request **1142** to the arbiter for the next low priority task to be serviced.

Referring to FIG. **34**, a timing diagram illustrates CPU service requests and services in case of a continuous CPU request **1146**. In practice, the CPU request is generally not continuous, but FIG. **34** has been provided for illustrative purposes. In the example represented in FIG. **34**, a block out timer **1148** is started upon a high priority service request **1149** by the CPU. At time to, the CPU starts making the continuous service request **1146**, and a high priority service request **1149** is first made provided that the block out timer **1148** is not running at time to. When the high priority service request is made, the block out timer **1148** is started. Between time to and time t_1, the memory controller finishes servicing a memory request from another task. The CPU is first serviced at time t_1. In the preferred embodiment, the duration of the block out timer is programmable. For example, the duration of the block out timer may be programmed to be 3 μs.

Any additional high priority CPU request **1149** is blocked out until the block out timer times out at time t_2. Instead, the CPU low priority request **1150** is handled by a sporadic server in a round robin manner between time to and time t_2. The low priority request **1150** is active as long as the CPU service request is active. Since the CPU service request **1146** is continuous, another high priority service request **1149** is made by the CPU and the block out timer is started again as soon as the block out timer times out at time t_2. The high priority service request made by the CPU at time t_2 is serviced at time t_3 when the memory controller finishes servicing another task. Until the block out timer times out at time t_4, the CPU low priority request **1150** is handled by the sporadic server while the CPU high priority request **1149** is blocked out.

Another high priority service request is made and the block out timer **1148** is started again when the block out

timer **1148** times out at time t_4. At time t_5, the high priority service request **1149** made by the CPU at time t_4 is serviced. The block out timer does not time out until time t_7. However, the block out timer is not in the path of the CPU low priority service request and, therefore, does not block out the CPU low priority service request. Thus, while the block out timer is still running, a low priority service request made by the CPU is handled by the sporadic server, and serviced at time t_6.

When the block out timer **1148** times out at time t_7, it is started again and yet another high priority service request is made by the CPU, since the CPU service request is continuous. The high priority service request **1149** made by the CPU at time t_7 is serviced at time t_8. When the block out timer times out at time t_9, the high priority service request is once again made by the CPU and the block out timer is started again.

The schedule that results from the task set and priorities above is verified by simulating the system performance starting from the "critical instant", when all tasks request service at the same time and a previously started low priority task is already underway. The system is proven to meet all the real-time deadlines if all of the tasks with real-time deadlines meet their deadlines. Of course, in order to perform this simulation accurately, all tasks make new requests at every repetition of their periods, whether or not previous requests have been satisfied.

Referring to FIG. **35**, a timing diagram illustrates an example of a critical instant analysis. At time to, a task **1** **1156**, a task **2** **1158**, a task **3** **1160** and a task **4** **1162** request service at the same time. Further, at time to, a low priority task **1154** is being serviced. Therefore, the highest priority task, the task **1**, cannot be serviced until servicing of the low priority task has been completed.

When the low priority task is completed at time t_1, the task **1** is serviced. Upon completion of the task **1** at time t_2, the task **2** is serviced. Upon completion of the task **2** at time t_3, the task **3** is serviced. Upon completion of the task **3** at time t_4, the task **4** is serviced. The task **4** completes at time t_5, which is before the start of a next set of tasks: the task **1** at t_6, the task **2** at t_7, the task **3** at t_8, and the task **4** at t_9.

For example, referring to FIG. **36**, a flow diagram illustrates a process of servicing memory requests with different priorities, from the highest to the lowest. The system in step **1170** makes a CPU read request with the highest priority. Since a block out timer is used with the CPU read request in this example, the block out timer is started upon making the highest priority CPU read request. Then the system in step **1172** makes a graphics read request. A block out timer is also used with the graphics read request, and the block out timer is started upon making the graphics read request.

A video window read request in step **1174** and a video capture write request in step **1176** have equal priorities. Therefore, the video window read request and the video capture write request are placed in a round robin arbitration for two tasks (clients). The system in step **1178** and step **1180** services a refresh request and a audio read request, respectively.

While respective block out timers for the CPU read request and the graphics read request are active, the system places the CPU read request and the graphics read request in a round robin arbitration for five tasks (clients), respectively, in step **1182** and step **1186**. The system in steps **1184**, **1188** and **1190** places other lowest priority tasks such as a graphics accelerator read/write request, a DMA read/write request and a CPU write request, respectively, in this round robin arbitration with five clients.

XIII. Graphics Accelerator

Displaying of graphics generally requires a large amount of processing. If all processing of graphics is performed by a CPU, the processing requirements may unduly burden the CPU since the CPU generally also performs many other tasks. Therefore, many systems that perform graphics processing use a dedicated processor, which is typically referred to as a graphics accelerator.

The system according to the present invention may employ a graphics accelerator that includes memory for graphics data, the graphics data including pixels, and a coprocessor for performing vector type operations on a plurality of components of one pixel of the graphics data.

The preferred embodiment of the graphics display system uses a graphics accelerator that is optimized for performing real-time 3D and 2D effects on graphics and video surfaces. The graphics accelerator preferably incorporates specialized graphics vector arithmetic functions for maximum performance with video and real-time graphics. The graphics accelerator performs a range of essential graphics and video operations with performance comparable to hardwired approaches, yet it is programmable so that it can meet new and evolving application requirements with firmware downloads in the field. The graphics accelerator is preferably capable of 3D effects such as real-time video warping and flipping, texture mapping, and Gouraud and Phong polygon shading, as well as 2D and image effects such as blending, scaling, blitting and filling. The graphics accelerator and its caches are preferably completely contained in an integrated circuit chip.

The graphics accelerator of the present invention is preferably based on a conventional RISC-type microprocessor architecture. The graphics accelerator preferably also includes additional features and some special instructions in the instruction set. In the preferred embodiment, the graphics accelerator is based on a MIPS R3000 class processor. In other embodiments, the graphics accelerator may be based on almost any other type of processors.

Referring to FIG. **37**, a graphics accelerator **64** receives commands from a CPU **22** and receives graphics data from main memory **28** through a memory controller **54**. The graphics accelerator preferably includes a coprocessor (vector coprocessor) **1300** that performs vector type operations on pixels. In vector type operations, the R, G, and B components, or the Y, U and V components, of a pixel are processed in parallel as the three elements of a "vector". In alternate embodiments, the graphics accelerator may not include the vector coprocessor, and the vector coprocessor may be coupled to the graphics accelerator instead. The vector coprocessor **1300** obtains pixels (3-tuple vectors) via a specialized LOAD instruction.

The LOAD instruction preferably extracts bits from a 32-bit word in memory that contains the required bits. The LOAD instruction also preferably packages and converts the bits into the input vector format of the coprocessor. The vector coprocessor **1300** writes pixels (3-tuple vectors) to memory via a specialized STORE instruction. The STORE instruction preferably extracts the required bits from the accumulator (output) register of the coprocessor, converts them if required, and packs them into a 32-bit word in memory in a format suitable for other uses within the IC, as explained below.

Formats of the 32-bit word in memory preferably include an RGB16 format and a YUV format. When the pixels are formatted in RGB16 format, R has 5 bits, G has 6 bits, and B has 5 bits. Thus, there are 16 bits in each RGB16 pixel and

there are two RGB16 half-words in every 32-bit word in memory. The two RGB16 half-words are selected, respectively, via VectorLoadRGB16Left instruction and VectorLoadRGB16Right instruction. The 5 or 6 bit elements are expanded through zero expansion into 8 bit components when loaded into the coprocessor input register **1308**.

The YUV format preferably includes YUV 4:2:2 format, which has four bytes representing two pixels packed into every 32-bit word in memory. The U and V elements preferably are shared between the two pixels. A typical packing format used to load two pixels having YUV 4:2:2 format into a 32-bit memory is YUYV, where each of first and second Y's, U and V has eight bits. The left pixel is preferably comprised of the first Y plus the U and V, and the right pixel is preferably comprised of the second Y plus the U and V. Special LOAD instructions, LoadYUVLeft and LoadYUVRight, are preferably used to extract the YUV values for the left pixel and the right pixel, respectively, and put them in the coprocessor input register **1308**.

Special STORE instructions, StoreVectorAccumulatorRGB16, StoreVectorAccumulatorRGB24, StoreVectorAccumulatorYUVLeft, and StoreVectorAccumulatorYUVRight, preferably convert the contents of the accumulator, otherwise referred to as the output register of the coprocessor, into a chosen format for storage in memory. In the case of StoreVectorAccumulatorRGB16, the three components (R, G, and B) in the accumulator typically have 8, 10 or more significant bits each; these are rounded or dithered to create R, G, and B values with 5, 6, and 5 bits respectively, and packed into a 16 bit value. This 16 bit value is stored in memory, selecting either the appropriate 16 bit half word in memory via the store address.

In the case of StoreVectorAccumulatorRGB24, the R, G, and B components in the accumulator are rounded or dithered to create 8 bit values for each of the R, G, and B components, and these are packed into a 24 bit value. The 24 bit RGB value is written into memory at the memory address indicated via the store address. In the cases of StoreVectorAccumulatorYUVLeft and StoreVectorAccumulatorYUVRight, the Y, U and V components in the accumulator are dithered or rounded to create 8 bit values for each of the components.

In the preferred embodiment, the StoreVectorAccumulatorYUVLeft instruction writes the Y, U and V values to the locations in the addressed memory word corresponding to the left YUV pixel, i.e. the word is arranged as YUYV, and the first Y value and the U and V values are over-written. In the preferred embodiment, the StoreVectorAccumulatorYUVRight instruction writes the Y value to the memory location corresponding to the Y component of the right YUV pixel, i.e. the second Y value in the preceding example. In other embodiments the U and V values may be combined with the U and V values already in memory creating a weighted sum of the existing and stored values and storing the result.

The coprocessor instruction set preferably also includes a GreaterThanOREqualTo (GE) instruction. The GE instruction performs a greater-than-or-equal-to comparison between each element of a pair of 3-element vectors. Each element in each of the 3-element vectors has a size of one byte. The results of all three comparisons, one bit per each result, are placed in a result register **1310**, which may subsequently be used for a single conditional branch operation. This saves a lot of instructions (clock cycles) when performing comparisons between all the elements of two pixels.

The graphics accelerator preferably includes a data SRAM **1302**, also called a scratch pad memory, and not a conventional data cache. In other embodiments, the graphics accelerator may not include the data SRAM, and the data SRAM may be coupled to the graphics accelerator instead. The data SRAM **1302** is similar to a cache that is managed in software. The graphics accelerator preferably also includes a DMA engine **1304** with queued commands.

In other embodiments, the graphics accelerator may not include the DMA engine, and the DMA engine may be coupled to the graphics accelerator instead. The DMA engine **1304** is associated with the data SRAM **1302** and preferably moves data between the data SRAM **1302** and main memory **28** at the same time the graphics accelerator **64** is using the data SRAM **1302** for its load and store operations. In the preferred embodiment, the main memory **28** is the unified memory that is shared by the graphics display system, the CPU **22**, and other peripherals.

The DMA engine **1304** preferably transfers data between the memory **28** and the data SDRAM **1302** to carry out load and store instructions. In other embodiments, the DMA engine **1304** may transfer data between the memory **28** and other components of the graphics accelerator without using the data SRAM **1302**. Using data SRAM, however, generally results in faster loading and storing operations.

The DMA engine **1304** preferably has a queue **1306** to hold multiple DMA commands, which are executed sequentially in the order they are received. In the preferred embodiment, the queue **1306** is four instructions deep. This may be valuable because the software (firmware) may be structured so that the loop above the inner loop may instruct the DMA engine **1304** to perform a series of transfers, e.g. to get two sets of operands and write one set of results back, and then the inner loop may execute for a while; when the inner loop is done, the graphics accelerator **64** may check the command queue **1306** in the DMA engine **1304** to see if all of the DMA commands have been completed. The queue includes a mechanism that allows the graphics accelerator to determine when all the DMA commands have been completed. If all of the DMA commands have been completed, the graphics accelerator **64** preferably immediately proceeds to do more work, such as commanding additional DMA operations to be performed and to do processing on the new operands. If not, the graphics accelerator **64** preferably waits for the completion of DMA commands or perform some other tasks for a while.

Typically, the graphics accelerator **64** is working on operands and producing outputs for one set of pixels, while the DMA engine **1304** is bringing in operands for the next (future) set of pixel operations, and also the DMA engine **1304** is writing back to memory the results from the previous set of pixel operations. In this way, the graphics accelerator **64** does not ever have to wait for DMA transfers (if the code is designed well), unlike a conventional data cache, wherein the conventional data cache gets new operands only when there is a cache miss, and it writes back results only when either the cache writes it back automatically because it needs the cache line for new operands or when there is an explicit cache line flush operation performed. Therefore, the graphics accelerator **64** of the present invention preferably reduces or eliminates period of waiting for data, unlike conventional graphics accelerators which may spend a large fraction of their time waiting for data transfer operations between the cache and main memory.

Although this invention has been described in certain specific embodiments, many additional modifications and

variations would be apparent to those skilled in the art. It is therefore to be understood that this invention may be practiced otherwise than as specifically described. Thus, the present embodiments of the invention should be considered in all respects as illustrative and not restrictive, the scope of the invention to be determined by the appended claims and their equivalents.

What is claimed is:

1. A unified memory system comprising:

a memory that is shared by a plurality of devices;

a memory request arbiter coupled to the memory, wherein the memory request arbiter performs real time scheduling of memory requests from different devices having different priorities; and

one or more block out timers, each block out timer being associated with one or more devices and coupled between the one or more associated devices and the memory request arbiter, wherein each block out timer is used to enforce a minimum interval between subsequent accesses by the one or more associated devices,

wherein the unified memory system provides for real time scheduling of tasks, some of which do not inherently have pre-determined periodic behavior, and for access to memory by devices that are sensitive to latency and do not have determinable periodic behavior.

2. The unified memory system of claim 1 wherein the devices associated with the block out timers include a CPU.

3. The unified memory system of claim 1 wherein the devices associated with the block out timers make high priority service requests through their associated block out timers.

4. The unified memory system of claim 3 wherein each block out timer blocks out the high priority service request made by its one or more associated devices while the block out timer is running until the block out timer times out.

5. The unified memory system of claim 4 wherein each block out timer is started when any one of its one or more associated devices makes the high priority service request.

6. The unified memory system of claim 4 wherein each block out timer is started when the high priority service request made by any one of its one or more associated devices is serviced.

7. The unified memory system of claim 4 wherein each block out timer is started some time between the time any one of its one or more associated devices makes the high priority service request and the time when the high priority service request is serviced.

8. The unified memory system of claim 1 wherein a block out period of at least one of the block out timers is programmable.

9. The unified memory system of claim 1 further comprising a round robin server for handling low priority tasks, wherein one or more devices that are associated with the block out timers are also coupled to the round robin server as low priority tasks, and the round robin server requests service to the memory request arbiter for one of the low priority tasks coupled to the round robin server at a time in a round robin manner.

10. The unified memory system of claim 9 wherein the devices that are associated with the block out timers are handled as one of the low priority tasks whenever their associated block out timers are running.

11. The unified memory system of claim 1 wherein each block out timer is associated with one of the devices, is coupled between the associated device and the memory request arbiter, and is used to enforce the minimum time interval between subsequent accesses by the associated device.

12. The unified memory system of claim 1 wherein at least one block out timer is associated with two or more devices, is coupled between the associated devices and the memory request arbiter, and is used to enforce the minimum time interval between subsequent accesses by the associated devices.

13. A method of designing real time scheduling comprising the steps of:

running a critical instant analysis;

ordering periods of different tasks from the shortest to the longest;

assigning priorities based on the order in accordance to the duration of the periods;

selecting tasks that are non-periodic but requires expedited servicing, assigning high priorities to them, and imposing a minimum interval between successive accesses by these tasks; and

assigning low priority to non-periodic tasks, that do not require expedited servicing, to be serviced in a round robin manner.

14. The method of designing real time scheduling of claim 13 wherein the step of running a critical instant analysis comprises the steps of assuming all tasks request service at the critical instant, assuming a task is being serviced at the critical instant, and determining whether or not all tasks can be serviced within a period of each task.

15. The method of designing real time scheduling of claim 13 wherein the design for real time scheduling is re-evaluated if the critical instant analysis fails.

16. The method of designing real time scheduling of claim 13 wherein the minimum interval between successive accesses by the high priority tasks that are non-periodic but requires expedited servicing is imposed by coupling a block out timer between each of these tasks and a memory request arbiter, such that high priority service request by each these tasks is blocked out while the block out timer coupled to the task is running.

17. The method of designing real time scheduling of claim 16 wherein the block out timer is designed to start running upon making of the high priority service request by the coupled task.

18. The method of designing real time scheduling of claim 16 wherein the block out timer is designed to start running upon servicing of the high priority service request made by the coupled task.

19. The method of designing real time scheduling of claim 18 wherein the block out timer is designed to start running some time between the time the high priority service request is made by the coupled task and that high priority service request is serviced.

20. The method of designing real time scheduling of claim 16 wherein the high priority tasks, that are non-periodic but requires expedited servicing, are also coupled to a round robin server that handles service requests from these high priority tasks as low priority requests in a round robin manner while respective block out timer is running.

21. The method of designing real time scheduling of claim 13 wherein real time scheduling is implemented independently to first and second arbiters.

22. The method of designing real time scheduling of claim 21 wherein the first arbiter is coupled to a first memory controller and the second arbiter is coupled to a second memory controller, and wherein the first and second memory controllers control first and second memory spaces that are contiguous to each other.

* * * * *

US005963210A

United States Patent [19]

Lewis et al.

[11] Patent Number: 5,963,210

[45] Date of Patent: Oct. 5, 1999

[54] **GRAPHICS PROCESSOR, SYSTEM AND METHOD FOR GENERATING SCREEN PIXELS IN RASTER ORDER UTILIZING A SINGLE INTERPOLATOR**

[75] Inventors: **Michael C. Lewis**, Redwood Estates; **Stephen L. Morein**, San Jose, both of Calif.

[73] Assignee: **Stellar Semiconductor, Inc.**, San Jose, Calif.

[21] Appl. No.: **08/624,260**

[22] Filed: **Mar. 29, 1996**

[51] **Int. Cl.**[6] **G06T 15/00**

[52] **U.S. Cl.** **345/419**

[58] **Field of Search** 395/119–125; 345/419–425

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,945,500	7/1990	Deering	395/122
5,345,541	9/1994	Kelley et al.	395/126
5,509,110	4/1996	Latham	395/121
5,574,835	11/1996	Duluk, Jr. et al.	395/121
5,579,455	11/1996	Greene et al.	395/122
5,596,686	1/1997	Duluk, Jr.	395/122

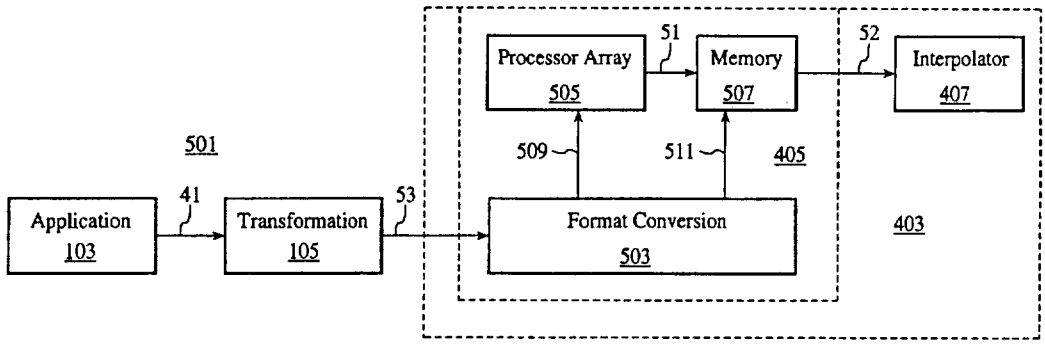
Primary Examiner—Almis R. Jankus

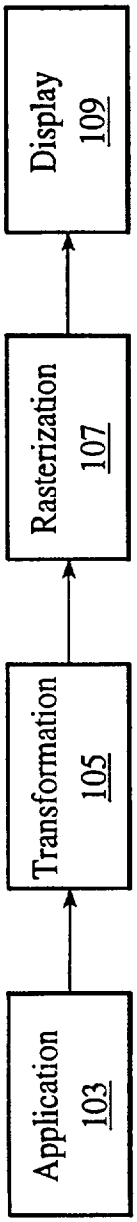
Attorney, Agent, or Firm—Sawyer & Associates

[57] **ABSTRACT**

A method and system for providing a graphics processor is disclosed. The method and system include providing a transformation processor, providing a rasterizer coupled to the transformation processor, and providing an interpolator coupled to the rasterizer. The transformation processor is for producing a set of transformed data according to a set of instructions from a set of raw data. The set of raw data describes at least one three-dimensional object within a bounded space extending from a display screen. The rasterizer is for identifying portions of the transformed data mapping a pre-defined area of the display screen in parallel and for sequentially rendering the identified portions of the transformed data in a pre-determined refresh order. The refresh order is the order that screen data is provided to the display screen to generate a screen image. The rasterizer further includes processor array. The processor array includes a plurality of primitive processors. Each of the plurality of primitive processors processes a corresponding portion of the transformed data and identifies whether the corresponding portion of the transformed data that intersects a selected area of the display screen. The interpolator is for determining a visible portion of the identified portions associated with selected area and characterizing the selected area according to transformed data associated with the visible portion.

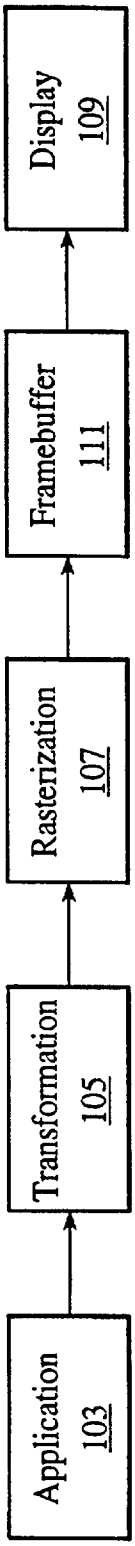
24 Claims, 6 Drawing Sheets





101

FIG. 1A



151

FIG. 1B

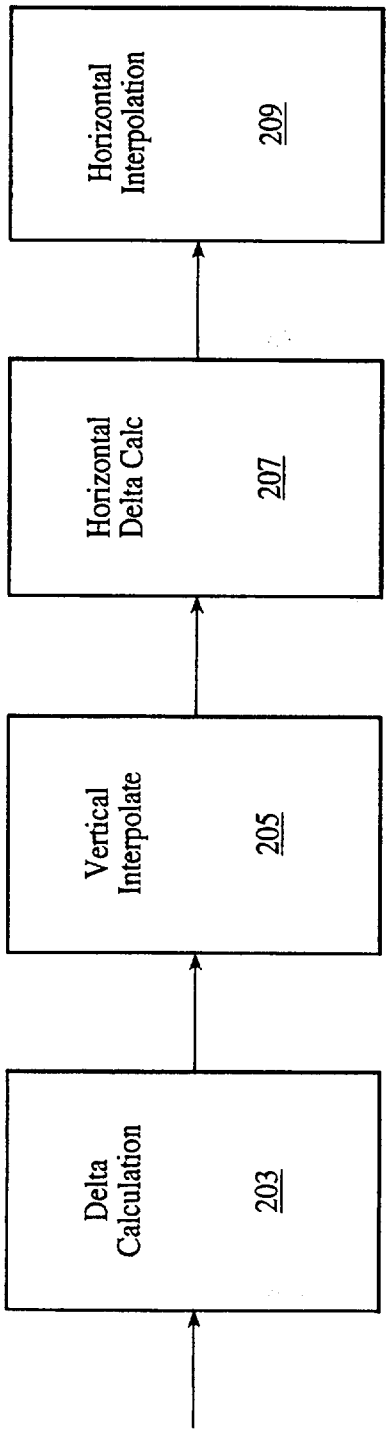


FIG. 2A 107

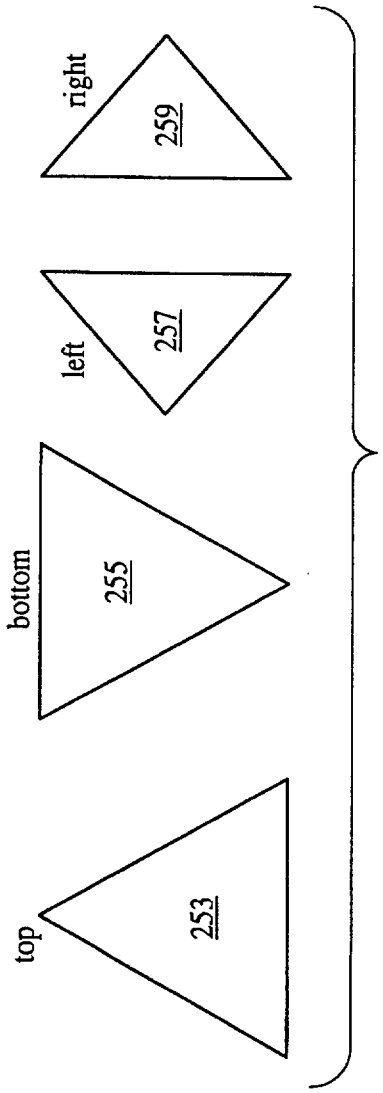


FIG. 2B

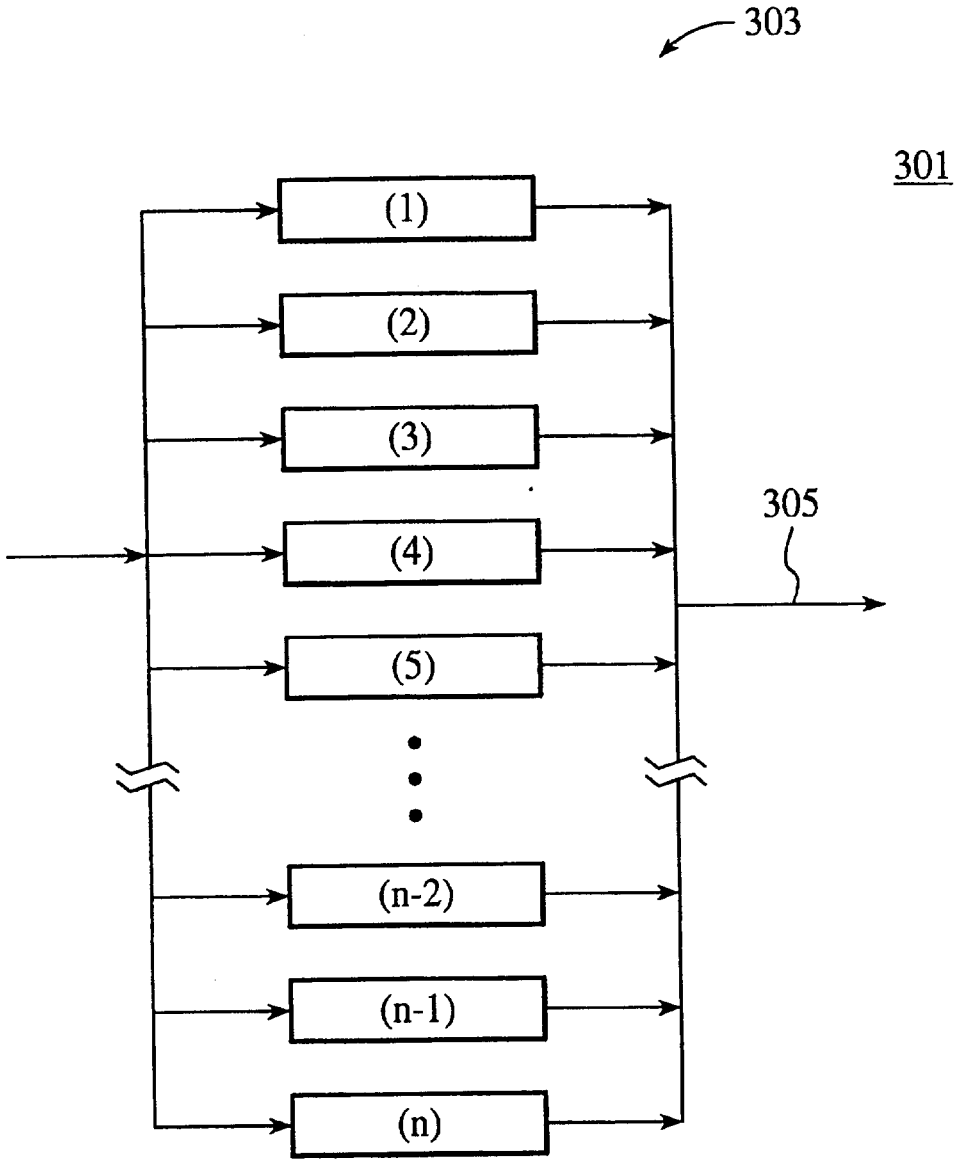
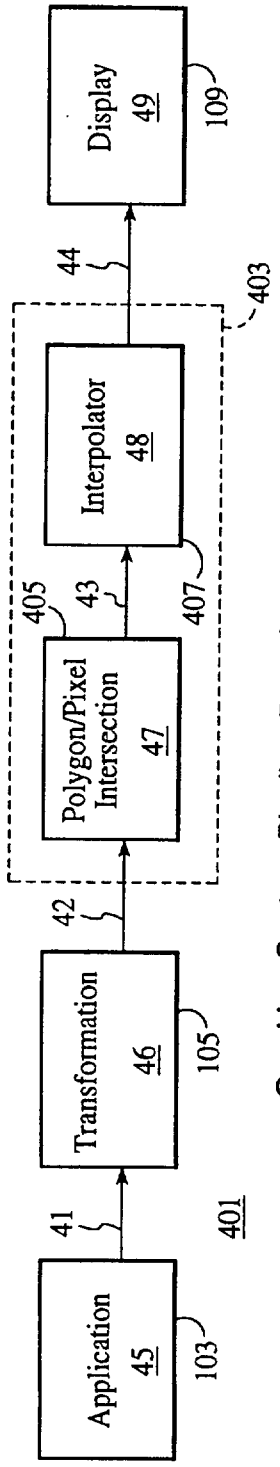


FIG. 3



Graphics System Pipeline Based on Pixelsquirt 2

FIG. 4

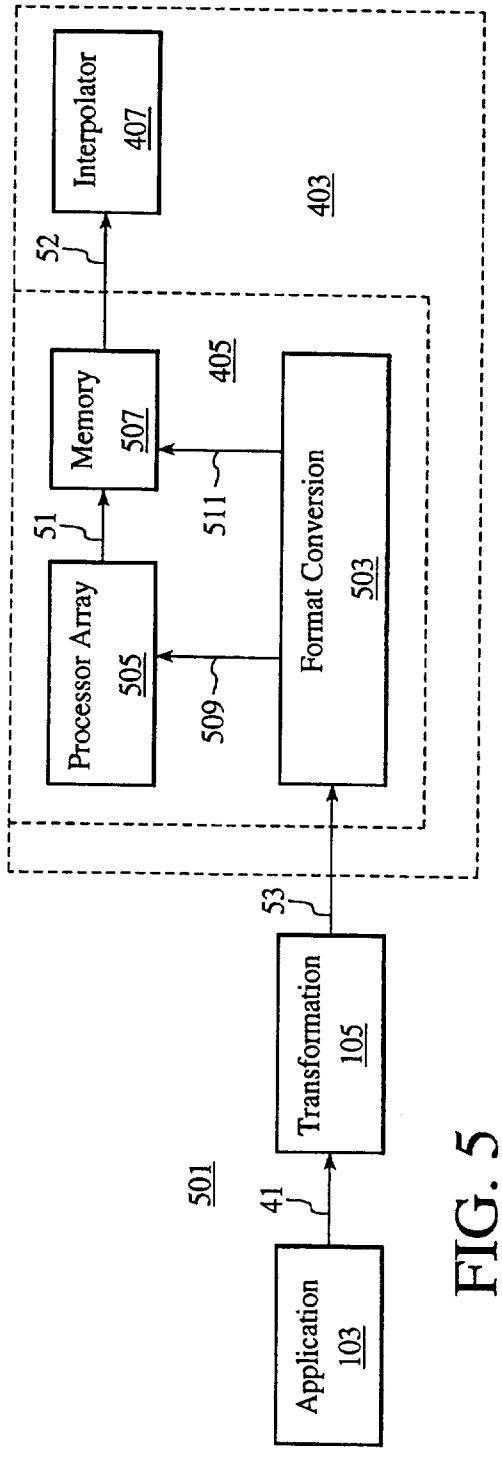


FIG. 5

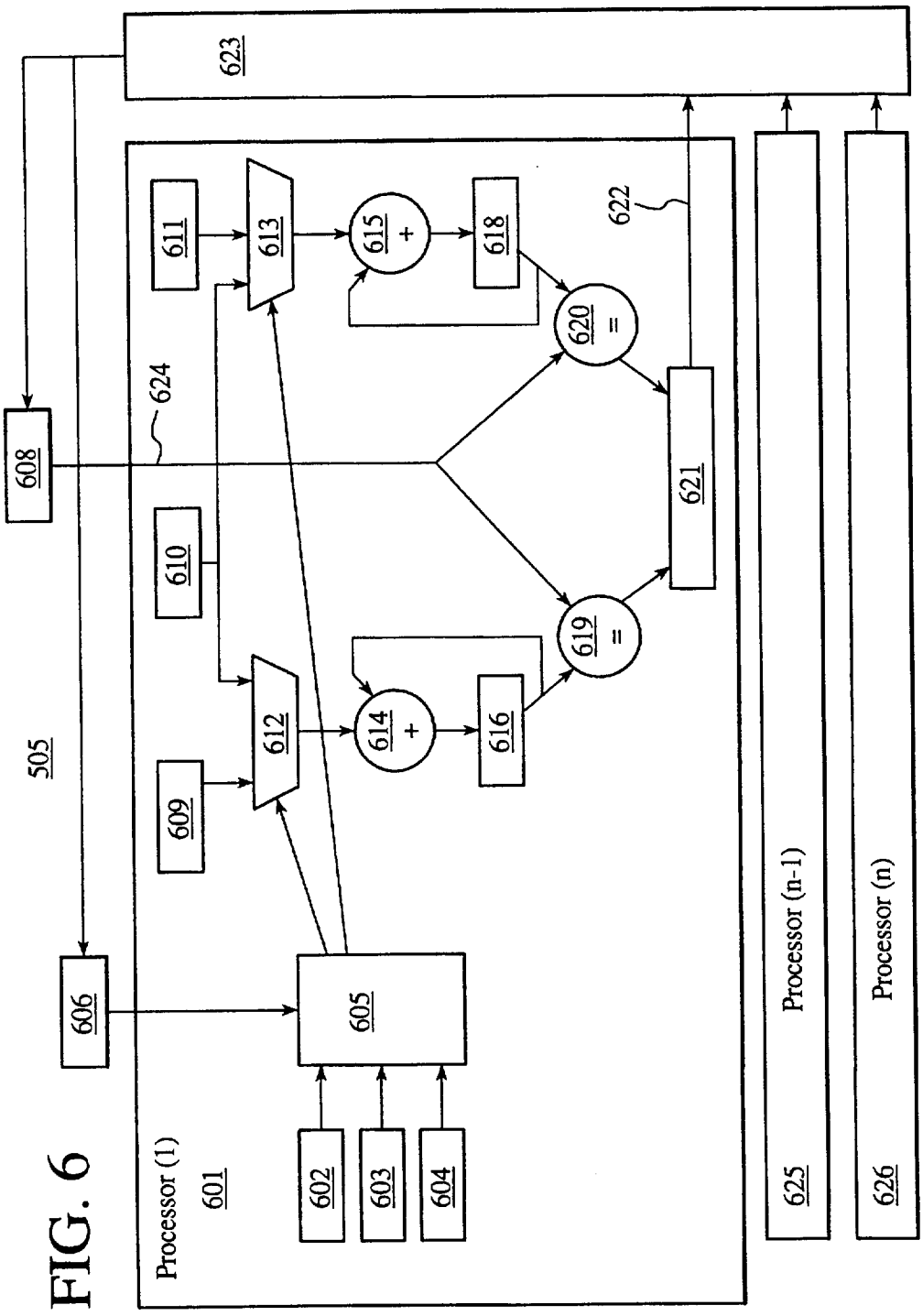

623
611
613
615
+
618
622
620
=
624
608
621
610
619
=
505
612
614
+
616
609
606
605
Processor (1)
601
602
603
604
Processor (n-1)
625
Processor (n)
626

FIG. 6

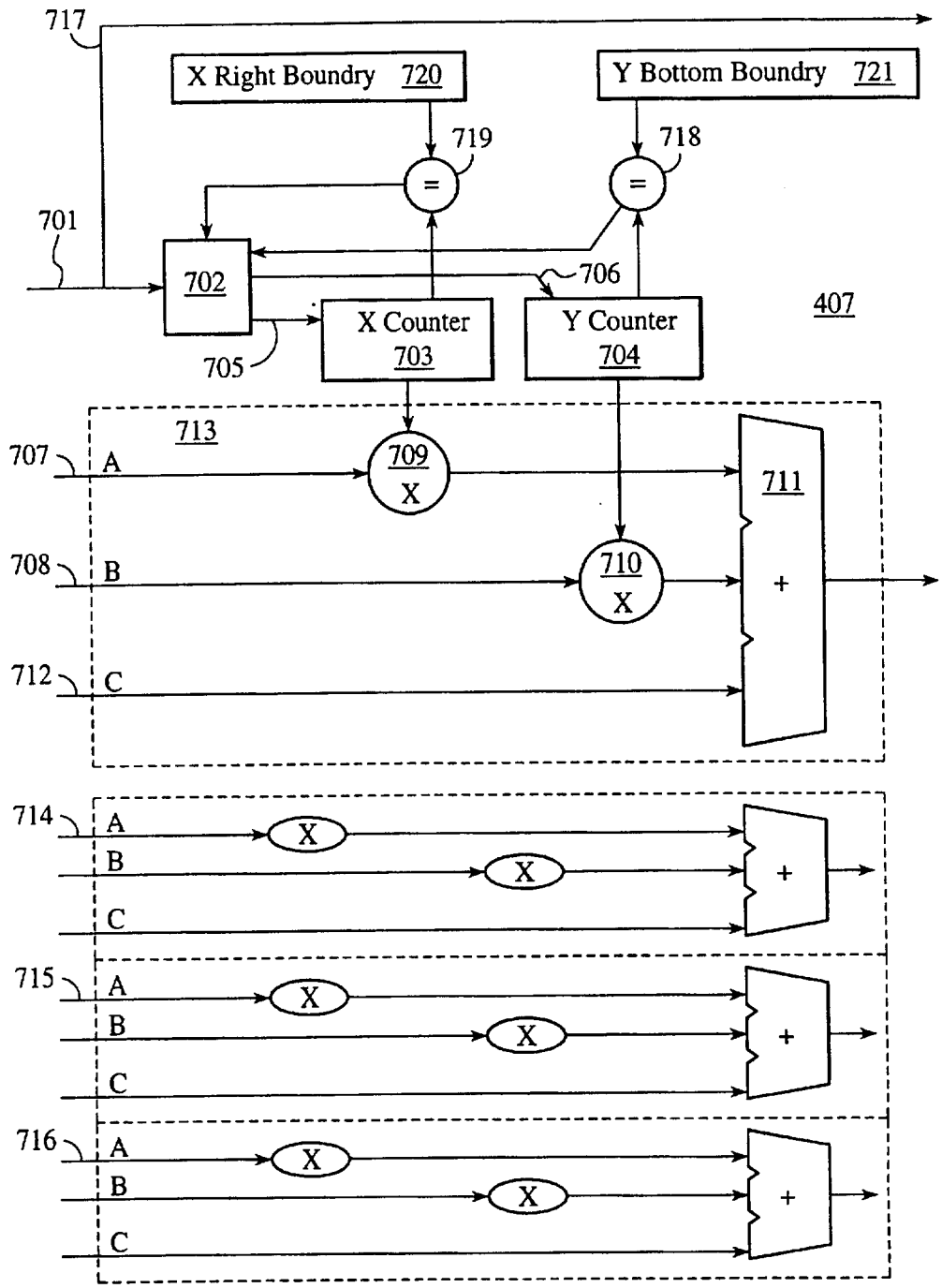

717
X Right Boundry 720
Y Bottom Boundry 721
719
718
=
=
701
702
706
X Counter 703
Y Counter 704
407
705
713
707
A
709 X
711
708
B
710 X
+
712
C
714
A
X
B
X
+
C
715
A
X
B
X
+
C
716
A
X
B
X
+
C

FIG. 7

GRAPHICS PROCESSOR, SYSTEM AND METHOD FOR GENERATING SCREEN PIXELS IN RASTER ORDER UTILIZING A SINGLE INTERPOLATOR

RELATED APPLICATIONS

The subject matter of this application is related to the subject matter of U.S. application Ser. No. 08/624,261 entitled "Method and Apparatus for Identifying and Eliminating Three-Dimensional Objects Visually Obstructed from a Planar Surface" filed on Mar. 29, 1996 by Michael C. Lewis and Stephen L. More in which is hereby incorporated by reference.

FIELD OF THE INVENTION

This invention relates generally to graphics computer systems, and more particularly to a graphics processor, system and method for generating screen pixels in raster order utilizing a single interpolator.

BACKGROUND OF THE INVENTION

In conventional computer systems, images are displayed on a two-dimensional screen. The images are defined by arrays of pixels which are either stored in computer memory or received from sources connected to the computer system.

Many images of physical objects may be defined three-dimensionally and may be stored or received as three-dimensional raw data arrays. In recent years, efforts have been undertaken to utilize three-dimensional raw data to take into account the distance and various characteristics of objects within screen images. One of the problems associated with the generation and display of such screen images is the size and complexity of circuitry and logic currently required to produce a stream of screen image data in the order required by a display. Various techniques have been developed to produce the screen data stream utilizing multi-stage graphics computer systems.

One of the graphics computer systems which have been developed to produce the screen data stream is shown in FIG. 1A. Graphics computer system **101** may be implemented with a conventional X86 (such as the 400 series) IBM-compatible or MacIntosh personal computer or a graphics engine that includes application unit **103** which generates geometries that are to be rendered. The output of application unit **103** is a stream of geometry data characterized in three-dimensional space. Transformation unit **105** transforms the geometry data from three-dimensional spatial coordinates to two-dimensional display coordinates corresponding to the screen plane. Transformation unit **105** also reformats the geometry data into a more unified format. For example, transformation unit **105** may accept as input independent polygons, quad strips, concave polygons and higher and only output triangle strips. The output of transformation unit **105** includes graphics primitives readable by rasterizer **107** in display coordinates. Rasterizer **107** receives the graphics primitives and converts them to pixels which in turn are transmitted to display **109** to generate a screen image.

Another graphics computer systems which has been developed to produce the screen data stream is shown in FIG. 1B. Graphics computer system **151** is shown which includes frame buffer **111**. Frame buffer **111** is utilized by system **151** to decouple the rendering process from a video refresh rate. This permits the image undergoing rendering to be updated at a slower rate than the screen image shown on

display **109** is refreshed. Some implementations of rasterizers **107** require associated frame buffers **111** to reorder pixels into screen refresh order. The output of frame buffer **111**, or rasterizer **107** if no frame buffer **111** exists, is a stream of pixels, where each pixel contains one color associated with one screen pixel.

With reference to FIG. 2A, conventional rasterizer **107** is shown which outputs pixels in polygon order. The conventional rasterizer includes slope, vertical, horizontal slope, and horizontal processing units designated as delta calculation, vertical interpolate, horizontal delta calc, and horizontal interpolation units **203, 205, 207, 209**, respectively.

An example of triangle primitives rasterized by rasterizer **107** are shown in FIG. 2B. The triangles are type classified as: top, bottom, left, and right facing triangles **253, 255, 257, 259**.

Conventionally, rasterizing occurs in three steps. The first rasterizing step converts the three point format of the triangle into three edges. The edges are usually described in the form of By+C. The second rasterizing step evaluates points along the edges of the triangle. There are two interpolators utilized in the second step, one for the left edge and one for the right edge. The output of the interpolator are referred to as spans, which are horizontal lines defined by a y-value, left and right x-values and any other parameters of the polygon defined at the ends of the span. The third rasterizing step also utilizes an interpolator which accepts a span and outputs the pixels that the span defines. For each pixel, the interpolator outputs the y-value and x-value of the current pixel and the value of the parameters of each pixel.

Graphics computer systems **151** utilizing such conventional rasterizers **107** require frame buffer **111** to reorder the pixels into the order needed by display **109** and also require a color unit to determine which pixel is visible and carries the color to be utilized by the associated screen pixel. Software applying the painters' algorithm provide a simple process to perform this task, which follows the rule that the last pixel sent to a pixel in frame buffer **111** replaces the pixel stored within frame buffer **111**. However, in order to perform this operation, the polygon data sent to rasterizer **107** must be sorted from back-to-front.

Conventional hardware, such as SGI-GTX manufactured by Silicon Graphics, Inc., that does not require back-to-front sorting applies a z-buffer algorithm. Z-buffer algorithms utilize an additional buffer, referred to as the z-buffer, that stores range values (z-values) as described by K. Akeley and T. Jermoluk in "High-Performance Polygon Rendering", SIGGRAPH 88, 239–246. The pixel currently stored in frame buffer **111** and z-buffer is read by rasterizer **107** and the z-values of the new pixel and the pixel in frame buffer **111** are compared by rasterizer **107**. If the new pixel is in front of the pixel in frame buffer **111** then the new pixel replaces the pixel in frame buffer **111**, otherwise the new pixel is discarded. Some algorithms that determine the color of a screen pixel require information about more than just the frontmost polygon that intersects a screen pixel. Examples of cases where information about multiple polygons is needed include: anti-aliasing, CSG (constructional solid geometry), and transparency. One solution is to modify frame buffer **111** to hold a list of pixels at each point and after all polygons have been rendered process the list of rendered pixels at each screen pixel into a single color.

Other work has been done to modify graphics computer systems which use z-buffers to provide some of the features of the multiple pixel/screen pixel system without the cost of

the memory needed by the multiple pixel/screen pixel system, for example by drawing the polygons in front-to-back or back-to-front order.

Rasterizers **107** that operate on data in polygon order are efficient, since a single interpolator rasterizes multiple polygons and, as long as there are polygons to be rasterized, the interpolator can be rendering. One of the disadvantages is that the pixels are not output in raster order and need to be reordered by frame buffer **111**. Additionally, since the pixels are output in polygon order, it is impossible to merge pixels from different polygons into a single screen pixel before the pixels are written into frame buffer **111** as a result the bandwidth needed into frame buffer **111** is very high. Another disadvantage is that if the color algorithm requires information about more than one rasterized pixel in each screen pixel either a very large frame buffer must be used or the polygons must be presorted, and presorted does not work in all cases.

A technique used to produce and transmit pixels in raster order is implemented with processor per primitive architecture **301** as partially shown in FIG. **3**. Processor architecture **301** includes array **303** of n processor-interpolator pairs. Each processor-interpolator pair of array **303** renders one polygon over the entire screen. The interpolators used in a processor per primitive are similar to those interpolators used in the polygon order system except that since the pixels are output in raster order there is no need to output the coordinate of each pixel. Instead, there is a need to indicate whether or not a given polygon intersects the current pixel. This can be done by comparing the y-values (vertical coordinate) of the current pixels against the top and bottom of the triangle and the x-values (horizontal coordinate) of the current pixels against the left and right edges of the span at the current y-value. The outputs of the interpolators of array **303** are connected to bus **305**. The output data of the respective interpolators is merged into a single stream by selecting one of the interpolators that has a polygon that intersects the current pixel during each clock cycle and transmitting the pixel that the selected interpolator has generated, where a clock cycle is defined by the processor to coordinate transfers of data. During the following clock cycle, a next interpolator pixel is transmitted, and so on until all the active interpolators for the current screen pixel have transmitted their respectively generated pixels. Once an entire screen of pixels has been generated and transmitted, the interpolators then generate and transmit the associated screen pixels for a next screen and so forth. A method of merging pixels that may be applied with processor per primitive architecture **301** is to use several z-value compare units to determine which interpolator has generated the frontmost screen pixel and to enable the interpolator with the frontmost screen pixel to transmit the screen pixel data. Disadvantages of processor per primitive architecture **301** include the large size of each interpolator processor, the large number of interpolator processors required, and low efficiency. Methods to reduce some of the disadvantages include: presorting the polygons from top-to-bottom, designing architecture **301** with the least number of processors required for the most complex scanline, loading the processors from the top of the list of polygons and, as the current scan line moves below the bottom of a polygon loaded into a processor, removing the polygon from the processor to free up space for a new polygon to be read from the list of polygons. Despite these improvements, efficient utilization of processor resources is low and the number of processors needed is high. Therefore, there continues to be a need for a more efficient and compact architecture.

SUMMARY OF THE INVENTION

In accordance with the present invention, there is provided a graphics processor, graphics processing system, and method for generating a screen image that associates three-dimensional image data of multiple objects with respective screen pixels and determine the screen pixel characteristics in raster order utilizing a single interpolator.

The method of the present invention segments data describing three-dimensional objects within an x-y planar window into polygons, compares the polygons to determine unobstructed polygons, interpolates the polygon data and associates the characteristics of the unobstructed polygons screen pixels, and generates a data stream of screen pixels in raster order which is the order in which the screen pixels are scannable onto a display.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1A** is a block diagram of a graphics pipeline system according to the prior art.

FIG. **1B** is a block diagram of an alternate embodiment of a graphics computer system according to the prior art.

FIG. **2A** is a block diagram of a conventional rasterizer implemented within the embodiments of FIG. **1A**, **1B**.

FIG. **2B** is a block diagram of polygon primitives produced by the embodiments of FIG. **1A**, **1B**.

FIG. **3** is a block diagram of a conventional array of interpolators implemented within the embodiments of FIG. **1A**, **1B**.

FIG. **4** is a block diagram of a graphics computer system according to the present invention.

FIG. **5** is a block diagram of a graphics processor implemented according to the present invention within the graphics computer system of FIG. **4**.

FIG. **6** is a block diagram representation of the processor array implemented within the rasterizer of FIG. **4**.

FIG. **7** is a block diagram representation of the interpolator processor implemented within the processor of FIG. **5**.

Like reference numerals refer to corresponding components throughout the several diagrams. The most significant digit of each reference numeral corresponds to the figure number wherein the reference numeral first appears and the associated component identified.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring to FIG. **4**, a block diagram is shown of graphics pipeline **401** wherein the present invention is implemented. Graphics system **401** may be implemented on a single semiconductor chip and may be optimized for specific applications to produce a three-dimensional graphics image utilizing integrated rasterizer **403**. Graphics system **401** includes conventional application processor **103** which stores and operates conventional application software that generates the description of the scene to be rendered in terms of a list of graphics objects and data describing the graphics objects in a special or three-dimensional coordinate system, conventional transformation processor **105** which converts the data into a list of graphics primitives in two-dimensional screen space and a pre-defined rasterizer format, and rasterizer **403** which rasterizes the list of graphics primitives utilizing polygon/pixel intersection unit **405** and integrated interpolator **407**.

Graphics system **401** may conventionally define a selected graphics primitive as a triangle. Each set of tri-

angles are generated from the raw data describing the surface of the corresponding three-dimensional object. Triangles in screen (two-dimensional) space are defined by three coordinates indicating the three points (vertices) of the triangle. A coordinate in screen space is comprised of two values, which describe the horizontal (x) and vertical (y) position of a given point. Additional descriptive information which is commonly associated with a graphics primitive are parameters used in visibility calculations, color determination, and z-value identification of each point. The z-value indicates the distance from the viewer of that point which may also be referred to as a depth position. With the z-value known at all three points of the triangle the z-value of any point inside the triangle can be determined, since a triangle is, by definition, planar, and the coordinates and the z-value define the plane of the triangle. The z-value is used in visibility calculations since the z-values of two polygons at the same point can be compared to determine which is in front of the other.

Other parameters are used to determine the color of points over the surface of the triangle. A common and popular method is Gouraud shading. This method approximates the behavior of diffuse illumination. The color at each point of the triangle is determined in the transformation stage by using a conventional lighting algorithm, for instance as discussed by Foley—vanDam in Computer Graphics and Practice. A lighting algorithm returns a color based on inputs that include geometry data, such as normals, color and material data for the object that is being generated, and location, color, and other specifications of the lighting sources in the scene. The color is usually specified as a triplet of red, green and blue values. Each color value is treated as an independent parameter and the value can be determined at any point inside the triangle since the color is specified at the three points of the triangle. Other parameters which may be used include coordinates for texture mapping. Texture mapping is a type of mapping that applies an image (the texture) to a triangle in a scene. The texture coordinate parameters define a triangle in the texture image and this triangle image is warped to fit the triangle being rendered as part of the scene. In general a triangle is specified by providing the three coordinates of the points of the triangle and, at each point, providing the value of the parameters that are used in this system.

Rasterizer **403** processes the list of graphics primitives from transformation processor **105** and renders the scene in refresh order. A current screen pixel is defined as the pixel in the scene that is currently being rendered. Rasterizer **403** initially defines the current screen pixel as the top-left most pixel in the output image. As the scene is rendered, the current screen pixel advances to the right of the screen on the same horizontal scan line. Once rasterizer **403** has advanced to the rightmost pixel of the scan line, rasterizer **403** defines the next current screen pixel as the first (leftmost) pixel on the next scan line down. Other alternative refresh (rendering) processes may be applied by rasterizer **403** which do not render horizontal lines sequentially, such as interlaced processing in which the scene is drawn twice and each rendering of a scene comprises every other horizontal scan line, or, simple reordering processing in which the scene is rendered from top-to-bottom with vertical scan lines and rasterizer increments on a columnar basis from left-to-right as each vertical line is drawn.

Polygon/pixel intersection unit **405** includes an array of processors. The processors are similar to the processors in a processor per primitive system. Each processor is loaded with one triangle and processes the respective triangle regardless of whether the triangle intersects with the current pixel. Each processor identifies whether the loaded primitive intersects the current screen pixel. Since only a single shared interpolator **407** is utilized, logic to interpolate parameters in each processor is not needed. Interpolator **407** is needed to generate the interpolated parameters for each processor only when the current screen pixel intersects the area of the triangle loaded in a particular processor.

The polygons (primitives) read from transformation processor **105** are loaded into available processors within polygon/pixel intersect unit **405**. Each processor rasterizes one triangle; however, the utility efficiency of the processors is improved by reusing processors to operate on multiple triangles during a single image rendering. Once the current screen pixel is below the bottom of a triangle, the processor is done rendering that triangle. If a processor is done rendering a triangle, the processor can be loaded with a new triangle as long as the top of the new triangle is below the current screen pixel. The process may be implemented by sorting the triangles from top to bottom using the topmost point in each triangle. This operation is performed after graphics transformations are completed by transformation processor **105**, or alternatively by rasterizer **403**.

The processor array of intersect unit **405** also includes logic utilized to scan through the pixels on the screen. The pixels are scanned in raster order and the output of the scanning logic is the current screen pixel. For each screen pixel, the outputs of all the processors are scanned. The output of each processor indicates if the current screen pixel intersects the area of the triangle. This is done to find which triangles intersect each screen pixel. The output of the processor array is a stream of screen pixels where each pixel is defined by a list of polygon primitives. The polygons in each screen pixel list are those that intersect the screen pixel.

Interpolator **407** reads in the polygon data and calculates the value of the parameters for each screen pixel by direct evaluation of the associated polygon data. The direct evaluation method organizes the polygon data into a format that describes the respective polygons in terms of equations of the form $v=f(x,y)$. This organization is developed by applying conventional mathematical rules, such as $y=mx+b$ and two points to describe a line. The direct evaluation method evaluates a parameter at an arbitrary point within the area of the triangle polygon directly by applying the respective equation $v=f(x,y)$.

After being characterized by interpolator **407**, the respective screen pixels are transmitted by rasterizer **403** in conventional format and in the same order as the current screen pixel is scanned on display **109**. Therefore, the rendered image may be transmitted directly to display **109** from rasterizer **403**. However, frame buffer **111** may be implemented between rasterizer **403** and display **109** to delay transmission of screen pixel data. If so, frame buffer **111** receives and transmits the rendered image in the same sequential raster order.

Referring to FIG. **5**, a flow diagram of an embodiment of graphics computer system processor **501** including integrated rasterizer **403** is shown which implements the process of the invention. Graphics system **501** is preferably implemented in circuitry that is more dense for memory than logic. Logic implementation of graphics system **501** is preferable if the technology used to construct graphics system **501** recommends or is preferably implemented in logic rather than memory.

Integrated rasterizer **403** includes format converter **503** that accepts the polygons from transformation unit **105** and

transmits the geometry of the triangles to processor array **505** and the parameter data to parameter memory **507**. This embodiment of rasterizer **403** separates the processing of geometry data from the processing of parameter data and therefore this data needs to be separated and put into a form by format converter **503** that is readable by the processors within array **505** and interpolator **407**.

The format of the polygon data provided to format converter **503** is machine-dependent. The assumption used in this specification is that the polygons primitives output from transformation unit **105** are triangles which are described by three points, each point including the coordinate of the point and the values of parameters at the point.

If the format of the polygons output from transformation unit **105** into format converter **503** are not triangles, such as squares, rectangles, pentagons, or circles, or are triangles not in a three point format then format converter **503** must be modified accordingly. In such cases, the polygons in the alternative format may be converted into three point triangles by format converter **503** and then processed as described herein. Alternatively, rasterizer **403** may be modified to directly process the alternative primitives or primitive format, such modification may include the application of polar or transform, such as fourier- or z-transform, coordinate systems.

Format converter **503** calculates delta values and performs setup operations, such as with the geometry data to be transferred to processor array **505**. The processors in processor array **505** are designed to perform forward-based differencing operations to calculate the edges of the triangle. Forward differencing requires data describing the edges of the respective polygons to be supplied to the respective processors at a starting value and a delta value. The starting value is the horizontal position of an edge on the first scan line that the edge is active. The delta value is the difference between the position of the edge on the first scan line and the position of the edge on the next scan line, and since the edge is a straight line, the delta value is the difference between the horizontal position of the edge on any two adjacent scan lines. The polygon primitive data supplied to format converter **503** are provided in a three point format. Format converter **503** calculates the delta values for the respective processors. After converting the geometry data, the converted geometry data in the delta format is transmitted by format converter **503** over bus **509** to processor array **505**.

Format converter **503** also uses the triangle data to transmit the parameter data in a format readable by interpolator **407**. Interpolator **407** utilizes the triangle data to calculate the values of the parameters at a point on the triangle. Interpolator **407** preferably uses the plane equation v=Ax+By+C to calculate the value of a parameter of a triangle at a certain point within the area of the triangle. Format converter **503** converts the parameter data associated with each point of the respective triangle into the A, B, C values of the plane equation utilizing the associated geometry data.

The three points of each triangle are referred to as: a, b, and c. Each point is defined by a vector including a coordinate, which is comprised of an x and y value respectively corresponding to horizontal and vertical directions, and the associated parameters. Each parameter is processed individually, and therefore only one parameter v is shown in the following equations.

The conversion is performed utilizing the following steps:

First, the three points a, b, c are re-written into vectors ab and ac utilizing the following calculations:

ab.x=b.x−a.x, ab.y=b.y−a.y, ab.v=b.v−a.v

ac.x=c.x−a.x, ac.y=c.y−a.y, ac.v=c.v−a.v

where a.x is the x component a.y is the y component and a.v is the parameter of point a, such that point a is defined as a.x,a.y,a.v.

A=ic/kc

B=jc/kc

C=(ic*a.x+jc*a.y+kc*a.v)/kc

where ic=(ab.y*ac.v−ab.v*ac.y)

jc=(ab.v*ac.x−ab.x*ac.v)

kc=(ab.x*ac.y−ab.y*ac.x)

Once the parameter data has been converted into A, B, C values, format converter **503** transmits the data over bus **511** into memory array **507**.

The parameter and geometry data need to remain associated so that the interpolator can calculate the correct value for parameters when it outputs the rendered pixels.

The parameter and geometry data of each triangle is associated by assignment of a common index number by format converter **503**. The processors in processor array **505** and portions of parameter memory **507** which respectively receive the parameter and geometry data of a triangle are also respectively assigned with the associated index numbers. A given triangle's data is assigned an index number that is not currently in use by format converter **503**. The geometry data output from format converter **503** is stored in the processor of array **505** identified by the same index number as the index number assigned to the triangle and the triangle's geometry data. The parameter data output from format converter **503** is stored in memory **507** at a location addressed by the same index number as the index number assigned to the original triangle and the triangle's geometry data. In the case when a processor indicates that the triangle contained by the processor intersects the current screen pixel, the index number of the processor may be used to address the portion of memory **507** containing the parameter data and to transfer the data to interpolator **407** which calculates the value of the parameters at the current screen pixel.

Processor array **505** includes n processors where the number n of processors, which should be greater than the number of triangles to be rendered, is defined by a graphics system designer. Processor array **505** is loaded with the triangle geometry data output from format converter **503**, such that each processor in array **505** is loaded with the geometry data from one triangle. Each of the processors determines whether the current screen pixel is within the area defined by the geometry of the triangle loaded into the processor. Array **505** renders one triangle at a time in raster order, where raster order is defined as that order in which the screen pixels in a scene are scanned. Each processor renders one triangle, such that all the pixels that result from intersecting triangles with a screen pixel are output by the respective processors before any pixels in the next screen pixel are output. Thus, the scene to be displayed is rendered by processor array **505** in raster order and is transferred by array **505** to memory **507** as a stream of screen pixels.

Each pixel output from array **505** is defined by a list of tokens produced by the processors of array **505**. Each token is produced by a respective processor to indicate whether the triangle assigned to the processor intersects the screen pixel. A token includes an index number and two flags. The index number is the index number of the triangle from which the geometry data was generated and is used to look up the parameter data for that triangle in memory **507**. The parameter data for that triangle is also associated with the same

index number and is stored in memory **507** at the address indicated by the index number. The two flag bits are defined as notvalid and last. When set, the last flag indicates that the particular token is the last token in a list of tokens associated with a screen pixel and the notvalid flag indicates that the particular token is not valid. In the case where a list of tokens contains a single token that has both the notvalid and last flag set, no triangles intersect the current screen pixel. If the notvalid flag is not set, the token indicates that the associated triangle intersects the current screen pixel.

The four possible token states that may be applied are therefore:

- valid, containing an index that refers to a polygon (triangle) and indicates that the polygon (triangle) intersects the screen pixel and may be visible or contribute to the color of the screen pixel;
- lastvalid, containing an index that refers to a polygon (triangle) and indicates that the polygon intersects with the screen pixel and may be visible or contribute to the color of the screen pixel, and that this is the last polygon associated with the screen pixel;
- last notvalid, indicating that there are no polygons associated with the screen pixel, and the next token will be for the next screen pixel; and,
- notvalid, containing an index that refers to a polygon (triangle) and indicates that the polygon (triangle) does not intersect the screen pixel which further indicates that this token is a null token and will not be generated by the processor array.

Memory **507** has a number of entries equal to the number of processors of array **505**. Each memory address is associated with one processor and contains the parameter data from the triangle whose geometry data is stored with the respective processor. Memory **505** is addressed with the index numbers. Each entry in memory **507** contains the ABC values for all parameters of the polygon. Memory **507** provides the parameter data to interpolator **407**. During each clock cycle, a token is read from processor array **505**. The index number in the token is used by array **505** to address and fetch the parameter data set from memory **507**. The parameter set and the flags from the respective token are then transmitted to interpolator **407**.

Interpolator **407** evaluates the data read from memory **507** to determine the values of the parameters at the current pixel. Interpolator **407** determines the values of the parameters by evaluating the equation Ax+By+C for each parameter and utilizing the A, B, C values read from memory **507**. From these values, interpolator **407** generates the rendered screen pixels in a format conventionally readable by display **109**.

Referring to FIG. **6**, a block diagram is shown of processor array **505**. Processor array **505** includes two counters **606, 608** which indicate the current screen pixel. Processor array **505** includes a plurality of n processors which may be sixteen or more. The 1st, n-1st, and nth processors **601, 625, 626** are shown in FIG. **6** as representative members of processor array **505**. Processor **601** is shown in detail as representative of the various processors in array **505**. The processors of array **505** calculate the intersections of the respective triangles and screen pixels and array **505** transmits the list of triangles that intersect the current screen pixel. Each processor determines if the triangle stored in the processor intersects the current screen pixel and transmits a signal to priority encoder **623** that indicates the result of the intersection determination. Priority encoder **623** converts the intersection determination signal into the index number associated with the triangle that intersects the current pixel. Priority encoder **623** includes the array control logic of processor array **505**. The control logic in priority encoder **623** controls counters **606, 608** so that the current screen pixel remains unchanged for the same number of clock cycles as there are intersections. Thus, priority encoder **623** transmits one index number for each processor with an active output (valid token). Priority encoder **623** also sets the two flags in each token that is output.

For example, if there are sixteen processors in processor array **505** with four processors containing triangles that intersect the current screen pixel and these processors have index numbers **4, 6, 10, 11**, then priority encoder **623** will utilize four clock cycles to transmit the respective tokens. The fourth token will have the last flag set which causes priority encoder **623** to increment the clocks **606, 608** to identify the next current screen pixel to be rendered beginning with the next clock cycle.

Once the last pixel of a frame has been rendered, priority encoder **623** resets counters **606, 608** to 0,0 which identifies the initial current screen pixel for a fame at the top left of the screen. Counters **606, 608** each include a limit register to indicate the maximum value for each counter, and define the resolution of the screen. When horizontal (x) counter **606** reaches a pre-determined maximum value, such as **1024**, a next value will be zero and vertical (y) counter **608** is incremented by one. When horizontal and vertical counters **606, 608** reach pre-determined maximum values, for instance **1024**, the current screen pixel is the last pixel for a frame. After a new display list is provided to rasterizer **403** by transformation processor **105**, the current screen pixel is reset to zero and the process for rendering a frame begins anew.

The processors of array **505** are loaded with the converted triangles from format converter **503** shown in FIG. **5**. If processor array **505** is designed so that the number of processors is greater than the number of triangles in a scene, then the various processors of array **505** are loaded with the triangles before the scene is rendered. Each triangle is described by various start positions and deltas, whose values are loaded into a set of registers within the respective processor.

According to the process implemented by graphics processor **401**, the triangles are transmitted from transformation processor **105** in a defined order, sorted by the topmost point from top-to-bottom. At the beginning of the rendering of a scene, each processor of array **505** is loaded with a triangle. As long as the number of processors is greater than the number of triangles that intersect the first scanline, then all triangles that may be drawn on the first scanline are loaded into processors. Each of the triangles are drawn by the respective processors. Any polygons that do not intersect the current scanline are identified by the respective processor and remain inactive in the processor for the first scanline. If the bottom of a triangle is above the next scanline then the processor is marked as empty and a new triangle is loaded into the processor before the next scanline. In this manner, processor array **505** is kept loaded with triangles that either intersect the current scanline or will intersect a future scanline.

Each processor includes registers **602, 603, 604, 609, 610, 611, 616** and **618** which store the data associated with respective triangles. Register **602** stores the top y value of the triangle. Register **603** stores the middle y value of the polygon, where one edge ends and another begins. Register **604** stores the bottom y of a triangle. Register **609** stores the initial left delta which defines the slope of the left edge. Register **611** stores the initial right delta which defines the

slope of the right edge. Register **610** stores the third delta which defines the second slope of the left or right side of the triangle. Register **616** stores the horizontal position of the left edge of the triangle. Register **618** stores the horizontal position of the right edge of the triangle.

The top y value may be used to indicate where the triangle starts in a scene. The middle value may be used in two of the four classifications of triangles. The classifications of triangles applied by processor array **505** is shown in FIG. **2B**. Top and bottom triangles have one horizontal edge while left and right triangles can be viewed as always having one edge on the right or left side and two edges on the other side. Control unit **605** classifies the respective triangle and stores two identification control bits which indicate the triangle classification. If the triangle is either right or left, then the middle y value indicates on which scanline the side with two edges should switch from the upper edge to the lower edge. The initial deltas are the slopes of the left and right edges of triangle. If the triangle is either left or right, then there is a second left or right edge and the delta stored in the initial register of the side with two edges is the slope of the upper edge. The slope of the lower edge is stored in third delta register **610**. If the triangle has a top or bottom classification, then the contents of middle y register **603** and third delta register **610** are not used. The left and right horizontal registers **616, 618** are loaded with the initial horizontal position of the two starting edges. If the triangle has either left, right or top classifications, then the two values are equal to the horizontal position of the topmost point. If the triangle has a bottom classification, then the triangle has two topmost points which are located at the same vertical position. The horizontal position of the leftmost points is loaded into left edge register **616** and the horizontal position of the rightmost point is loaded into right edge register **618**.

Control unit **605** compares top, middle, and bottom y values stored in registers **602, 603, 604** with the current y value stored in counter **606** to determine which edges of the triangle to use. When the current y value is above the top y value of the triangle, then the current scanline is above the triangle and the triangle is not visible in the scanline. When the current scanline has not yet reached the triangle, the respective processor is off-line and the contents of the respective registers within the processor, specifically the left and right registers **616, 618**, are left unchanged. When the current scanline reaches the top of the triangle, the processor is placed on-line by priority encoder **623** and the stored triangle begins the rendering process.

If the current y value is between the top and middle y values, then the triangle is visible in the scanline and the left and right edge positions must be updated. The left and right registers are updated by adding the delta values, the contents of left delta value register **609** for the left edge and the contents of right delta value register **611** for the right value, to the values stored in left and right registers **616, 618**.

If the triangle is a "left" or "right" triangle, then the comparison of the middle y value to the current y position is used. If the current y value is between the middle y value and the bottom y value, then either the left or right edge switches from the initial edge used by the processor on that side of the triangle to the third edge of the triangle. If the triangle is a left triangle and the y value is between the middle and bottom y values, then the contents of left value register **616** is incremented by the contents of third delta register **610** and the contents of right value register **618** is incremented by the contents of right delta register **611**. If the triangle is a right triangle and the y value is between the middle and bottom y values, then the contents of left value register **616** is incremented by the contents of left delta register **609** and the contents of right value register **618** is incremented by the contents of third delta value register **610**. Since the third delta value is the slope of the third edge and the beginning of the third edge is the location where the previous edge ends, the result of switching to the third delta value is to switch to the third edge.

If the current y value is below the bottom y value, then the triangle is not visible on the current scanline and will not be visible on any scanlines below the current scanline. The triangle data contains no more useful information in the rendering of the screen pixels. Therefore, the processor containing the associated triangle data may be reloaded by writing the data associated with another triangle over the data of the prior triangle.

The contents of left and right value registers **619, 620** describe the left and right boundaries of the area covered by the triangle on the current scanline. This area is generally called a span of the triangle. If the current screen pixel is between the left and right bounds of the triangle then the triangle intersects the current screen pixel and the processor transmits a valid token (corresponding to a logical true state). The current screen pixel is on the current scanline by definition since the current scanline is specified by the vertical position of the current screen pixel. Whether the current screen pixel is within the span of the triangle in the processor is determined by two equality comparators **619, 620** that compare the current horizontal position against the contents of left and right value registers **616, 618**. Each equality comparator **619, 620** transmits a logical true signal when the current screen pixel is on either (or both) the left or/and right edge of the span. State machine **621** indicates if the current screen pixel is inside the span with a one bit state signal. When the left comparison is logically true the state of the state machine is set to inside and when the right comparison is logically true the state is reset to outside.

The truth table of the stage machine is as follows:

lefteq	righteq	current state	new state
0	0	0	0
0	0	1	1
0	1	x	0
1	0	x	1
1	1	x	0

where 'lefteq' and 'righteq' are the outputs of the two comparators **619, 620**. 'lefteq' is the result of comparing the current screen pixel to the left horizontal register and 'righteq' is the result of comparing the current screen pixel to the right horizontal register. The current state is the state of the machine during the current clock cycle. A positive state indicates that the triangle is active (valid) at the current pixel. The new state is the state of the machine during the next clock cycle. The state of state machine **621** is output to priority encoder **623**.

With further reference to FIG. **5**, triangle parameter memory **507** reads in the tokens output from priority encoder **623** in processor array **505** and replaces the index in each token with the contents of parameter memory **507** at the location addressed by the index from the token. The new token, containing the parameter data is then output to interpolator **407**. The parameter is needed by interpolator **407** to calculate the values of the parameters over the surface of the polygon. An alternative to utilizing tokens is to store the parameter data in each processor along with the triangle geometry and to include a multiplexer connecting to priority encoder **623** so that priority encoder **623** outputs the triangle

parameter data instead of the index. The alternative embodiment would require an increased size of the data that flows through processor array **505** and a corresponding increase in the size of the circuitry, since storing data in memory **507** is more area efficient than storing the data in registers within each processor of array **505**.

A flow diagram of interpolator **407** utilized within graphics processor **501** is shown in FIG. **7**. Interpolator **407** calculates the value of the parameters of a triangle at the intersection of the current screen pixel and the surface of the triangle. It calculates the values of the parameters by direct evaluation. Since graphics processor **501** has a pipeline design, processor array **505** generates tokens and then uploads parameter data from parameter memory **507** before transmitting the tokens to interpolator **407**. This procedure delays the arrival of tokens with parameter data at interpolator **407** which may be required for a screen pixel that was before the current screen pixel in processor array **505**. Interpolator **407** therefore determines an interpolator current screen pixel with two counters **703, 704**, counter **703** tracks x values and counter **704** tracks y values.

Counters **703, 704** are controlled by the flags of the incoming tokens. The "last" flag indicates that the list of tokens that make up a screen pixel has ended and that counters should be advanced to the next screen pixel. Interpolator **407** includes two equality comparators **718, 719** to checks for bounds, two registers **720, 721** that store the right and bottom boundaries of the screen, and decoder **702** that controls counters **703, 704** based on the flags from the incoming token and the results of comparators **718, 719**.

When the last flag is set for the incoming triangle, decoder **702** increments horizontal counter **703**. The value stored in register **720** is the right horizontal boundary of the screen and is loaded by a host computer system when the computer system is initialized or is a fixed value if the resolution is fixed. Similarly, the value stored in register **721** contains the bottom of the screen and is also loaded at initialization. If the value stored in horizontal counter **703** is equal to the value stored in x boundary register **720**, then, when the next triangle with the last valid flag set is received, x value counter **703** is reset to zero and y value counter **704** is incremented by one. If the value stored by y counter **704** is equal to the value stored in y boundary register **718**, then, when x counter **703** is reset to zero, y counter **704** is reset to zero. When y counter **704** is zero and x counter **703** is zero then the current interpolator pixel is the top left pixel. When x and y counters **703, 704** are both equal to the boundary registers then the current pixel is the last on the screen and at the bottom right. Vertical counter **704** reset to zero indicates the completion of a scene. Horizontal counter **703** reset to zero indicates the end of a scanline. In this manner, the current screen pixel in the interpolator is updated, scanning from left-to-right and top-to-bottom, keeping in synchronization with the stream of tokens input from processor array **505**.

The flag bits **701** remain in synchronization with the parameter processing and output with the rendering pixel and the outputs of blocks **714, 715, 716** on output **717**.

The data path portion of interpolator **407** includes several individual interpolators **713, 714, 715, 716** of which interpolator **713** is shown in detail. Each of the interpolators **713, 714, 715, 716** evaluates one parameter of the triangle at the current screen pixel. The parameter value is determined by evaluating the equation Ax+By+C, where x and y are coordinates of the current interpolator screen pixel and A,B, and C are the parameter data associated with the triangle that is in the token transmitted from memory **507** into interpolator **407** and was originally generated from the three-point form of the triangle in format converter **503**. For a Gouraud shaded triangle with z depth information, four interpolators **713, 714, 715, 716** are needed. Interpolator **713** is applied to the z-value and interpolators **714, 715, 716** are applied to the colors: red, green, and blue. Each interpolator **713, 714, 715, 716** is identical in structure, but the precision, in terms of bit-width of the inputs, outputs, and internal busses vary based on the precision needed by the parameter. Color commonly utilize an eight bit result while z-values use a sixteen bit result.

Each interpolator **713, 714, 715, 716** is comprised of two multipliers **709, 710** and three operand adder **711**. Multiplier **709** multiplies the current horizontal value stored by counter **703** by A value input **707** associated with the triangle. Multiplier **709** performs the Ax portion of the Ax+By+C calculation of the parameter. Multiplier **710** multiplies the current vertical value stored in counter **704** by B value input **708** associated with the triangle. Multiplier **710** performs the By portion of the Ax+By+C calculation of the parameter. Three operand adder **711** outputs the sum of three inputs **707, 708, 712**. Adder **711** can be constructed from two operand adders by summing two of the inputs in one two operand adder and summing the output of the first adder with the third input. Three operand adder **711** sums the results of multipliers **709, 710** with C value input **712** associated with the triangle. The result is the value of the parameter at the current screen pixel for the input triangle.

In an alternate embodiment of graphics processor **501**, rather than convert the triangles into the Ax+By+C form, the parameter data is directly evaluated from the triangle data. This is accomplished by finding the values of the parameters at the intersection of the left and right edges with the current scanline and then finding the intersection of the line defined by those two endpoints with the current pixel. The alternate embodiment works directly from the three point definition of the triangle to determine the value of the parameter at a point on the screen. It does this by finding the left and right edges that intersect the current scanline. The edges are defined by their endpoints. The intersection of an edge and the scanline is found by solving the following equation:

$$v(y)=(y/(p2.y-p1.y)*(p2.v-p1.v))+p1.v$$

where x,y are the coordinates of a point, v is the parameter, p1.x is the x value of the first point, p1.y is the y value of the first point, p2.x is the x value of the second point, and p2.y is the y value of the second point.

The value of the parameter at both intersections is determined by the above equation and the value of x for both edges is calculated. The value of the parameter is found by the following equation:

$$v(y)=(x/(xright-xleft)*(vright-vleft))+vleft$$

where vleft and vright are the values of the parameter at each endpoint and xlelft and xright are the horizontal position of the intersection.

The combined equation is:

$$v(y) = (x/(((y/(p4\cdot y - p3\cdot y)*(p4\cdot x - p3\cdot x) + p3\cdot x) - (y/(p2\cdot y - p1\cdot y)*(p2\cdot x - p1\cdot x) + p1\cdot x))* ((y/p4\cdot y - p3\cdot y)*(p4\cdot v - p3\cdot v) + p3\cdot v) -$$

-continued

$$(y/p2\cdot y - p1\cdot y)*(p2\cdot v - p1\cdot v) + p1\cdot v))+$$

$$(y/(p2\cdot y - p1\cdot y)*(p2\cdot v - p1\cdot v) + p1\cdot v)$$

where the edges are p1-p2 and p3-p4. The combined equation replaces Ax+By+C in the first embodiment.

Although the invention herein has been fully described, it may be appreciated that various of the structures described herein may be implemented either in hardware, software, or combinations thereof.

What is claimed is:

1. A graphics processor for providing an image from a set of data describing at least one three-dimensional object within a bounded space extending from a display screen, the graphics processor comprising:

a processor array for identifying portions of the data mapping a particular area of the display screen and for rendering the identified portions of the transformed data in an order, the processor array further including a plurality of primitive processors, each of the plurality of primitive processors for processing a corresponding portion of the data and for identifying whether the corresponding portion of the data intersects the particular area; and

a single interpolator coupled to the processor array for determining a visible portion of the identified portions associated with the particular area and characterizing the selected area according to a portion of the data associated with the visible portion.

2. The graphics processor of claim 1 wherein each of the plurality of primitive processors is further for providing an output if the corresponding portion of the data intersects the particular area.

3. The graphics processor of claim 2 wherein the data includes geometric data and parameter data, and wherein:

the plurality of processors are for processing a corresponding portion of the geometric data associated with the corresponding portion of the data.

4. The graphics processor of claim 3 wherein the single interpolator is further for interpolating a parameter data for the visible portion of the data.

5. The graphics processor of claim 4 wherein the refresh order is the order that screen data is provided to the display screen to generate a screen image.

6. A graphics processor comprising:

a transformation processor for producing a set of transformed data according to a set of instructions from a set of raw data describing at least one three-dimensional object within a bounded space extending from a display screen;

a rasterizer coupled to the transformation processor for identifying portions of the transformed data mapping a pre-defined area of the display screen in parallel and for sequentially rendering the identified portions of the transformed data in a pre-determined refresh order, the refresh order being the order that screen data is provided to the display screen to generate a screen image, the processor further including

a processor array including a plurality of primitive processors, each of the plurality of primitive processors for processing a corresponding portion of the transformed data and for identifying whether the corresponding portion of the transformed data intersects the pre-defined area; and

a single interpolator coupled to the rasterizer for determining a visible portion of the identified portions

associated with the pre-defined area and characterizing the pre-defined area according to transformed data associated with the visible portion.

7. The graphics processor as in claim 6 wherein the graphics processor includes;

an application processor including a software application, the software application including the set of instructions for generating the screen image.

8. The graphics processor as in claim 6 wherein the set of transformed data comprises multiple sets of primitive data respectively describing planar geometric shapes and parameter data associated with respective of the sets of primitive data, each of the planar geometric shapes describing a respective surface portion of the three-dimensional object.

9. The graphics processor as in claim 8 wherein the rasterizer includes a memory for storing the associated parameters of the respective multiple sets of primitive data.

10. The graphics processor as in claim 6 wherein the interpolator includes:

a single interpolator processor connected to the processor array for sequentially evaluating and comparing data from each primitive processor associated with the pre-defined area to determine the visible portion, the single interpolator processor determining the visible portion associated with each area of the display in the refresh order.

11. The graphics processor as in claim 6 wherein the predefined area comprises a screen pixel, a set of the screen pixels define a screen image for display on the display screen, the refresh order defining a pre-determined order of a data stream of the screen pixels for transmission to and illumination of the display.

12. The graphics processor of claim 6 wherein each of the plurality of primitive processors is further for providing an output relating to the corresponding portion of transformed data if the corresponding portion of the transformed data intersects the pre-defined area.

13. A graphics processor system comprising:

a graphics processor including:

a transformation processor for producing a set of transformed data according to a set of instructions from a set of raw data describing at least one three-dimensional objects within a bounded space extending from a display screen;

a rasterizer coupled to the transformation processor for identifying portions of the transformed data mapping a pre-defined area of the display screen in parallel and sequentially rendering the identified portions of the transformed data in a pre-determined refresh order, the refresh order being the order that screen data is provided to the display screen to generate a screen image, the rasterizer further including

a processor array including a plurality of primitive processors, each of the plurality of primitive processors processing a corresponding portion of the transformed data and identifying whether the corresponding portion of the transformed data intersects the pre-defined area;

a single interpolator coupled to the rasterizer for determining a visible portion of the identified portions associated with the pre-defined area, characterizing the pre-defined area according to transformed data associated with the visible portion, and providing rasterized data including the visible portions; and

a display producing an image by scanning the rasterized data in the refresh order.

14. The graphics processor system as in claim 13 including:

a memory connecting to the graphics processor and storing the rasterized data.

15. The graphics processor system as in claim 13 including:

a user interface providing access by a user to the graphics processor to accommodate the insertion of instructions and data by the user to the system.

16. The graphics processor system as in claim 13 wherein the graphics processor includes:

an application processor including a software application, the software application including the set of instructions for generating the screen image.

17. The graphics processor system as in claim 13 wherein the set of transformed data comprises multiple sets of primitive data respectively describing planar geometric shapes and parameter data associated with respective of the sets of primitive data, each of the planar geometric shapes describing a respective surface portion of the three-dimensional object.

18. The graphics processor system as in claim 13 wherein the interpolator further includes:

a single interpolator processor connected to the processor array for sequentially evaluating and comparing data from each primitive processor associated with a selected respective area to determine the visible portion, the single interpolator processor determining the visible portion associated with each area of the display in the refresh order.

19. The graphics processor system as in claim 13 wherein the pre-defined area comprises a screen pixel, a set of the screen pixels define a screen image for display on the display screen, the refresh order defining a pre-determined order of a data stream of the screen pixels for transmission to and illumination of the display.

20. The graphics processor system as in claim 19 wherein the rasterizer includes a memory for storing the associated parameters of the respective multiple sets of primitive data.

21. The graphics processor system of claim 13 wherein each of the plurality of primitive processors is further for providing an output relating to the corresponding portion of transformed data if the corresponding portion of the transformed data intersects the pre-defined area.

22. A method for producing a graphics image including the steps of:

obtaining data describing a three-dimensional object within a space bounded in two directions x,y, the space including a reference plane in the x,y directions, the reference plane including an identifiable area, the three-dimensional object having a plurality of surface portions;

transforming the data into a plurality of geometric data blocks describing a plurality of geometric surfaces, the plurality of geometric surfaces describing the plurality of surface portions of the three-dimensional object;

identifying a plurality of geometric data blocks which map a portion of the plurality of geometric surfaces onto the identifiable area of the reference plane, the geometric data blocks being identified in parallel in a plurality of primitive processors, each primitive processor processing a corresponding one of the plurality of geometric data blocks; and

identifying a visible data block defining a geometric surface that is nearest to the identifiable area along an axis extending perpendicularly from the area of those surfaces described by the identified geometric data blocks, the visible data block being identified by a single interpolator.

23. The method as in claim 22 wherein the method includes:

generating the screen image according to a set of instructions.

24. The method as in claim 22 including the steps of:

interpolating the visible data block to determine a point on the perpendicular axis;

characterizing the selected area according to transformed data associated with the point; and

generating a set of screen data characterizing the selected area and successively selected areas obtained by repeating the prior steps, the set of screen data being generated in refresh order for directly scanning a screen image onto a display.

* * * * *

US006178198B1

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

(10) **Patent No.: US 6,178,198 B1**
(45) **Date of Patent: *Jan. 23, 2001**

(54) **APPARATUS FOR, AND METHOD OF, PROCESSING SIGNALS TRANSMITTED OVER A LOCAL AREA NETWORK**

(75) Inventors: **Henry Samueli**, Northridge; **Fang Lu**, Irvine; **Avanindra Madisetti**, Torrance, all of CA (US)

(73) Assignee: **Broadcom Corproation**, Irvine, CA (US)

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

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

(21) Appl. No.: **08/970,557**

(22) Filed: **Nov. 14, 1997**

(51) **Int. Cl.**[7] **H03K 3/11**

(52) **U.S. Cl.** **375/214**; 375/232; 375/233; 375/286; 375/355

(58) **Field of Search** 375/214, 232, 375/233, 348, 355, 222, 229, 286; 708/322, 323

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,962,637	6/1976	Motley et al.	375/42
4,597,089	6/1986	Motley et al.	375/13
4,599,732	7/1986	LeFever	375/13
4,864,590	9/1989	Arnon et al.	375/14
5,003,555	3/1991	Bergmans	375/12
5,031,194 *	7/1991	Crespo et al.	375/230
5,065,412	11/1991	Schenk	375/111
5,230,012	7/1993	Schenk	375/111
5,276,711	1/1994	Rossi	375/106
5,444,712	8/1995	Betts et al.	370/110.1
5,559,840	9/1996	Melas et al.	375/355
5,604,741	2/1997	Samueli et al.	370/402
5,617,450 *	4/1997	Kakuishi et al.	375/230
5,638,409 *	6/1997	Awata et al.	375/355
5,724,397 *	3/1998	Torsti	375/355
5,748,674 *	5/1998	Lim	375/233

OTHER PUBLICATIONS

Joshi, Robindra B. et al., "WP 4.2: A 100 MHz, 5MBaud QAM Decision–Feedback Equalizer for Digital Television Applications" 1994 IEEE International Solid–State Circuits Conference, ISSCC94/Session 4/Video and Communication Signal Processors/Paper WP 4.2 pp. 68–69.

Martin, James et al., "Local Area Networks Architectures and Implementations" Second Edition, 1994, pp. 167–170, 204.

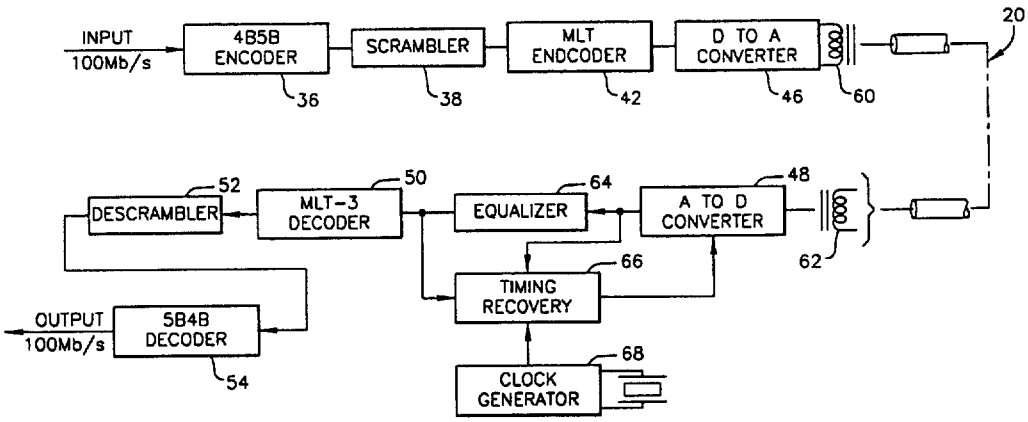
(List continued on next page.)

Primary Examiner—Amanda T. Le
(74) *Attorney, Agent, or Firm*—Christie, Parker & Hale, LLP

(57) **ABSTRACT**

Digital signals provided by a repeater connected to a plurality of clients by unshielded twisted wire pairs, are converted to analog signals which become degraded during transmission through the wires. Clients convert the degraded analog signals to digital signals. Digital signal phases are coarsely adjusted to have assumed zero crossing times coincide in-time with a clock signal zero crossing. Signal polarity, and the polarity of any change, is determined at the assumed zero crossing times of the digital signals. Precursor and post-cursor responses, resulting from signal degradation, are respectively inhibited by a feed forward and a decision feedback equalizer. The time duration of post-cursor response is further inhibited by a high pass filter and a tail canceller. Phase adjustments are made, after response inhibition, by determining the polarity, and the polarity of any change, at the assumed zero crossing times. Before phase adjustments are made, a phase offset is provided in order to compensate for phase degradations introduced by the unshielded twisted wire pairs.

51 Claims, 6 Drawing Sheets



EXHIBIT

C

OTHER PUBLICATIONS

Razavi, James et al., "Monolithic Phase–Locked Loops and Clock Recovery Circuits" IEEE Press, 1996, Tutorial pp. 2–7.

Anderson, John B., "Digital Transmission Engineering" IEEE Press, 1999, Synchronization—Chapter 4, pp. 193–199.

* cited by examiner

FIG. 1
PRIOR ART

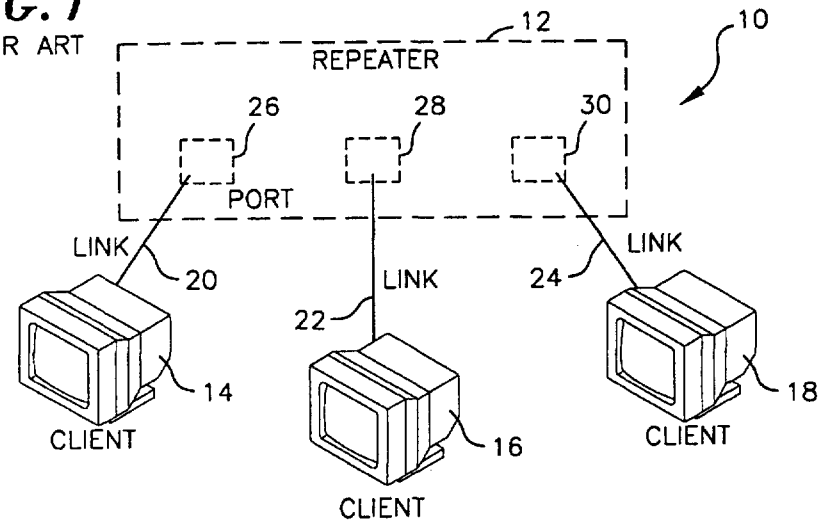


FIG. 2
PRIOR ART

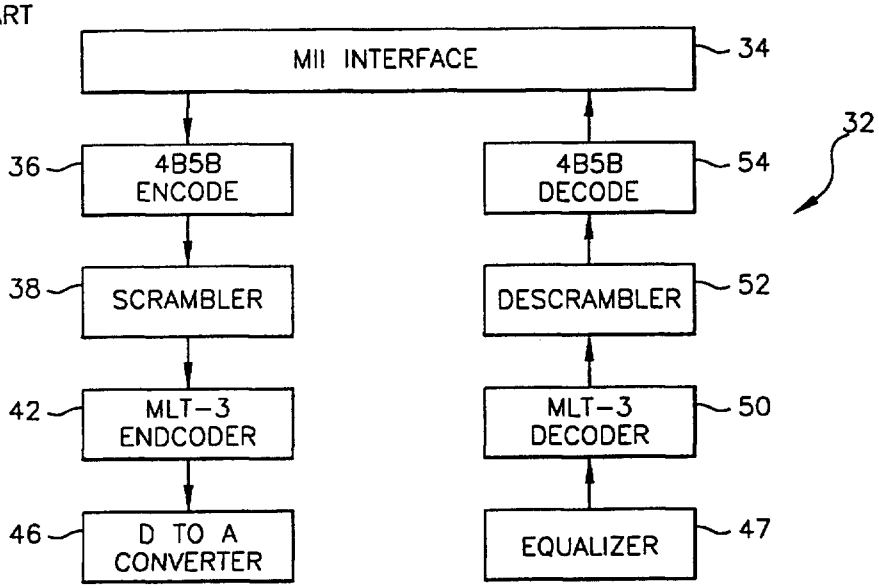
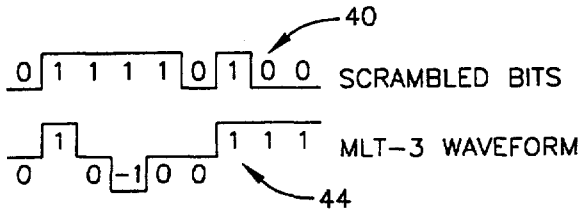


FIG. 3
PRIOR ART



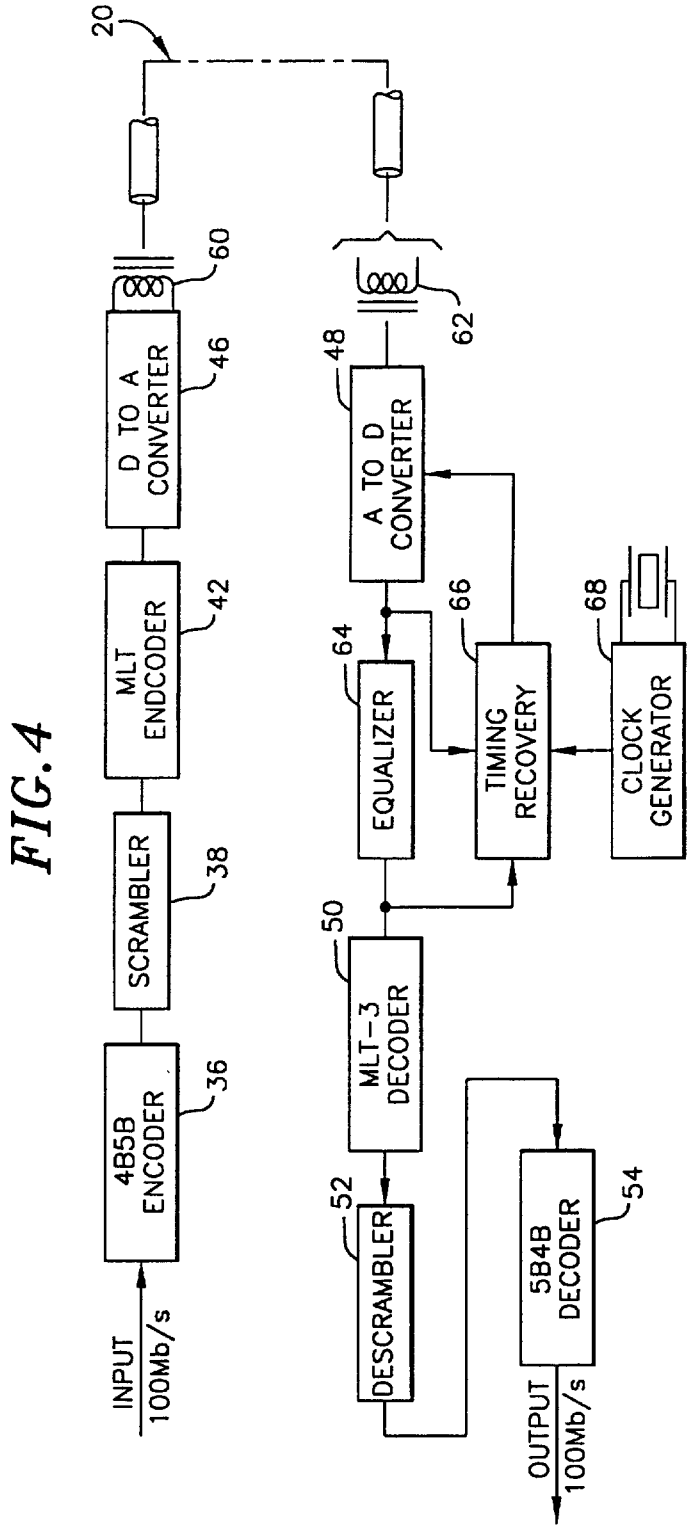

20
60
62
D TO A CONVERTER
46
A TO D CONVERTER
48
MLT ENDCODER
42
EQUALIZER
64
TIMING RECOVERY
66
CLOCK GENERATOR
68
SCRAMBLER
38
MLT-3 DECODER
50
4B5B ENCODER
36
DESCRAMBLER
52
5B4B DECODER
54
INPUT 100Mb/s
OUTPUT 100Mb/s

FIG. 4

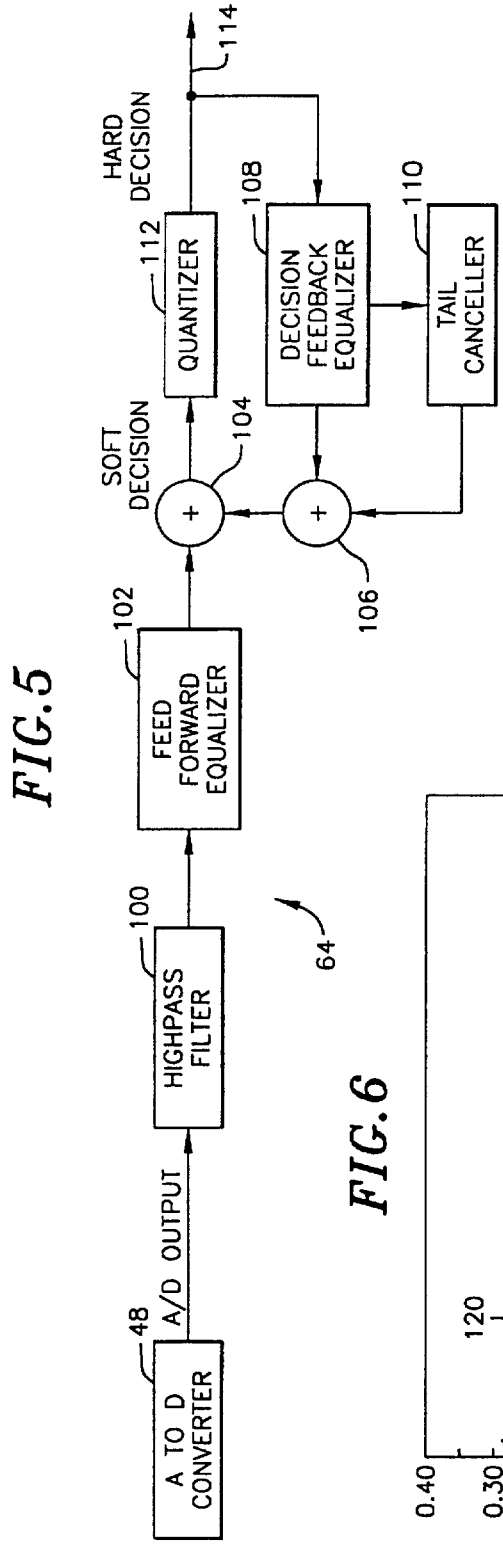

A TO D CONVERTER
48
A/D OUTPUT
HIGHPASS FILTER
100
FEED FORWARD EQUALIZER
102
SOFT DECISION
104
106
QUANTIZER
112
HARD DECISION
114
DECISION FEEDBACK EQUALIZER
108
TAIL CANCELLER
110
64

FIG. 5

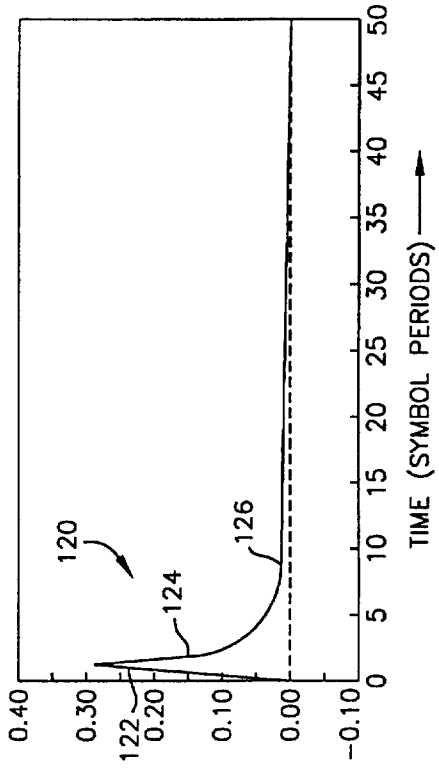

120
122
124
126
0.40
0.30
0.20
0.10
0.00
-0.10
0
5
10
15
20
25
30
35
40
45
50
TIME (SYMBOL PERIODS)

FIG. 6

FIG. 7

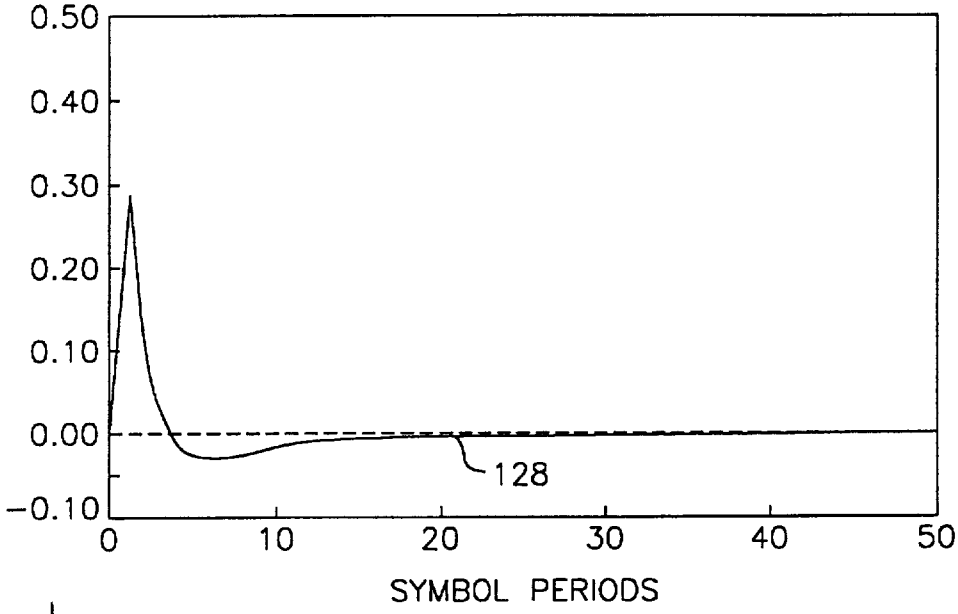
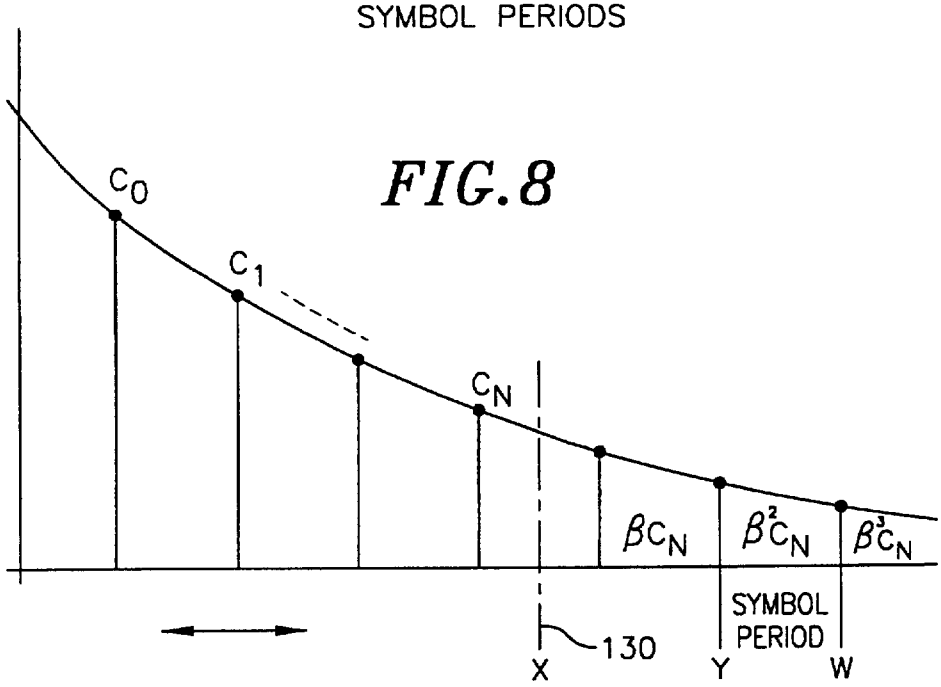

0.50
0.40
0.30
0.20
0.10
0.00
−0.10
0
10
20
30
40
50
128
SYMBOL PERIODS

FIG. 8


C_0
C_1
C_N
βC_N
$\beta^2 C_N$
$\beta^3 C_N$
SYMBOL PERIOD
130
X
Y
W

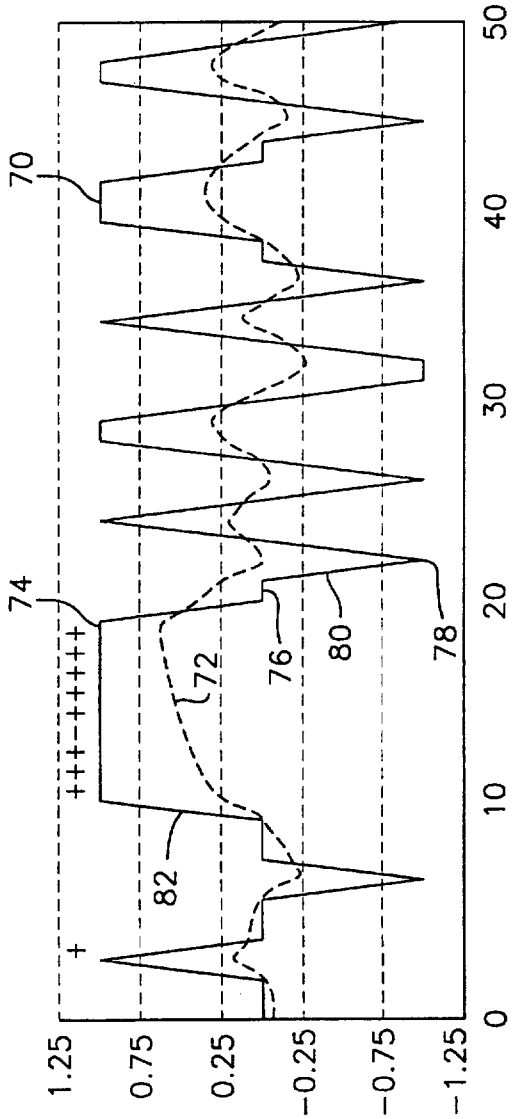

70
74
72
76
80
78
82
+++-++++
+
1.25
0.75
0.25
−0.25
−0.75
−1.25
0
10
20
30
40
50

FIG. 9

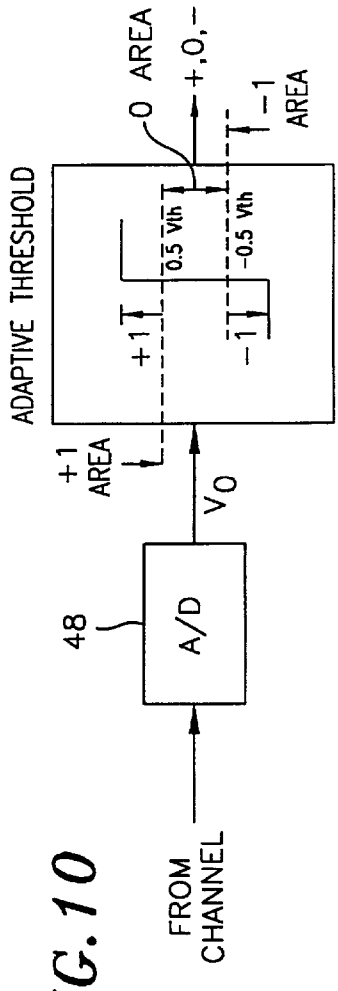

FROM CHANNEL
A/D
48
V_O
+1 AREA
ADAPTIVE THRESHOLD
+1
0.5 Vth
-0.5 Vth
-1
0 AREA
+,0,-
-1 AREA

FIG. 10

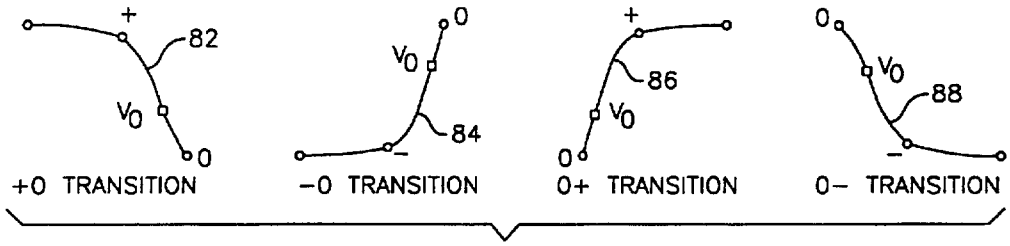


FIG. 11

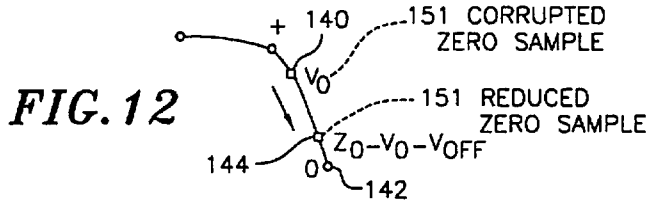


FIG. 12

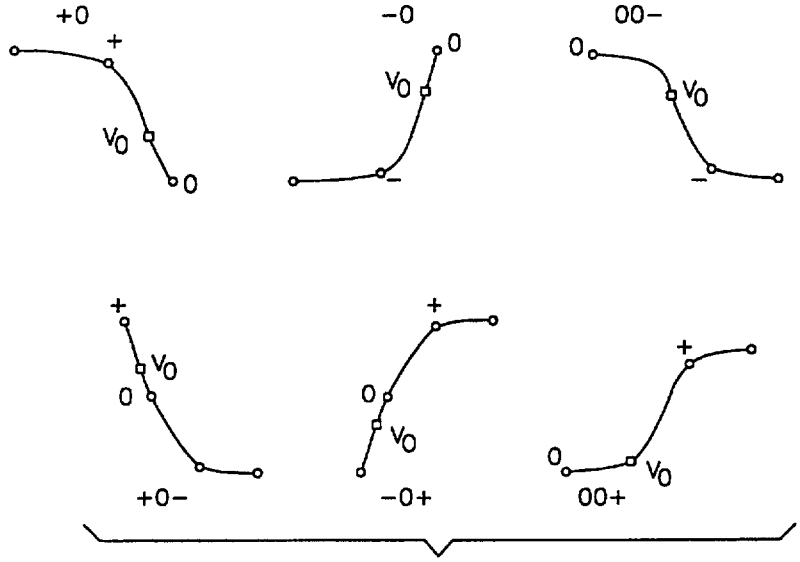


FIG. 13

APPARATUS FOR, AND METHOD OF, PROCESSING SIGNALS TRANSMITTED OVER A LOCAL AREA NETWORK

This invention relates to systems for, and methods of, providing for the transmission and reception of signals through unshielded twisted pairs of wires between a repeater and a plurality of clients. The invention particularly relates to systems for, and methods of, using digital techniques for enhancing the recovery, and the quality of such recovery, of the analog signals passing through the unshielded twisted pairs of wires to the client so that the information represented by such analog signals will be accurately recovered at the client.

BACKGROUND OF THE INVENTION

In a hub-and-spoke network topology, a repeater resides on a hub. The repeater facilitates an exchange of data packets among a number of clients. A client can be a computer, a facsimile machine, another computer, etc. The repeater serves several ports where each port is connected to an individual one of the clients with a separate point-to-point link between the repeater and such client.

In a 100BASE-TX signalling protocol, unshielded twisted pairs of wires constitute the point-to-point link between the repeater and each of the clients. Each link consists of two pairs of unshielded twisted wires. One pair of the unshielded twisted wires provides for a transmission of data from the repeater to an individual one of the clients. The other pair of the unshielded twisted wires provides for a transmission of data from the individual one of the clients to the repeater.

When information is illustratively transmitted from the repeater to an individual one of the clients in a 100BASE-TX system, the information is originally in digital form. The digital information may represent individual ones of a plurality of analog levels. Specifically, in a 100BASE-TX System, the digital signals may represent analog levels of +1, 0 and −1.

The digital information at the repeater may be converted to analog form and then transmitted in analog form through the unshielded twisted pair of wires to the individual one of the clients. The transmitted signals are received in analog form at the individual one of the clients. The received signals are then processed to recover the transmitted information represented by the analog information.

The distance between the repeater and the individual one of the clients may be as great as one hundred meters. The unshielded twisted pair of wires coupling the repeater and the individual one of the clients produces a degradation in the characteristics of the signals as the signals pass through the unshielded twisted pair of wires. The amount of the degradation rapidly increases with increases in the length of the unshielded twisted pair of wires connected between the repeater and the individual one of the clients.

The degradation results in part from Inter Symbol Interference (ISI), signal attenuation, crosstalk, clock jitter and a number of other factors. Such degradation severely distorts the transmitted data signals. The degradation also results in part from the fact that the analog information transmitted from the repeater to the individual one of the clients is also received at the other clients connected to the repeater and is reflected back to the repeater, thereby affecting the characteristics of the signals transmitted from the repeater to the individual one of the clients.

Analog techniques have been used in the prior art to process the analog signals received at the individual one of

the clients. These analog techniques have not been completely effective in eliminating the degradation or distortions in the signals received at the individual one of the clients. This has caused errors to be produced in the information received and processed at the individual one of the clients. This has been true even though the 100BASE-TX system provides substantially greater noise immunity than other types of systems and is able to handle smaller signal levels than other types of systems.

BRIEF DESCRIPTION OF THE INVENTION

This invention relates to a system for, and method of, converting analog signals received at a client from a repeater to corresponding digital signals. The digital signals are processed to shift the times for the production of the digital signals so that the digital signals are produced at the zero crossings of clock signals having a particular frequency. The digital signals are also processed to determine at each instant the magnitude of the digital signals closest to the magnitude representing individual ones of a plurality of amplitude levels such as +1, 0 and −1 and to then convert such magnitude to such closest one of such amplitude levels. In this way, the information represented by the transmitted signals is accurately recovered at the client.

In one embodiment of the invention, digital signals provided by a repeater connected as by unshielded twisted pairs of wires to a plurality of clients are converted to analog signals. The analog signals become degraded during transmission through the wires. At the client, the degraded analog signals are converted to digital signals. Initially, the phases of the digital signals are coarsely adjusted to have the times assumed for a zero crossing of the digital signals coincide in time with the zero crossing of a clock signal. This phase adjustment is made by determining the polarity, and the polarity of any change, in the digital signals at the time assumed to be the zero crossings of the digital signal.

Subsequently the pre-cursor and post-cursor responses (resulting from the signal degradations) in the digital signals are respectively inhibited by a feed forward equalizer and a decision feedback equalizer. A high pass filter and a tail canceller also inhibit the post-cursor response of the digital signals by limiting the time duration of the post-cursor response.

Phase adjustments are made in the resultant digital signals, after the inhibition in the pre-cursor and post-cursor responses, by determining the polarity, and the polarity of any change, in the digital signals at the times assumed to be the zero crossings of the digital signals. However, before any phase adjustments are made, a phase offset is provided in the digital signals to compensate for phase degradations produced in the signals passing through the unshielded twisted pairs of wires.

Although the invention is discussed in this application with reference to the 100BASE-TX system, it will be appreciated that the invention is not limited to the 100BASE-TX system. For example, the invention is applicable to any 100BASE-TX system. The invention is also applicable to other systems.

BRIEF DESCRIPTION OF THE DRAWINGS

In the drawings:

FIG. 1 is a schematic diagram, primarily in block form, of a system known in the prior art and including a repeater, a plurality of clients and a plurality of links (e.g., unshielded twisted pairs of wires) each connected between the repeater and an individual one of the clients;

FIG. 2 is a schematic diagram, primarily in block form, of a system known in the prior art for encoding information in digital form at the repeater, converting the digital information to analog information at the repeater, transmitting the analog information to a client, converting the analog information to digital information at the client and decoding the digital information at the client to recover the transmitted information;

FIG. 3 is a schematic diagram showing how digital bits of information are scrambled at the receiver in the prior art and how the scrambled bits are encoded at the repeater to a sequence of bits having a plurality of amplitude levels such as +1, 0, and −1;

FIG. 4 is a circuit diagram, primarily in block form, of a system known in the prior art for encoding information in digital form and transmitting the information in an analog form to a client and of a system included as one embodiment of the invention for digitally processing the analog signals received at the client to recover the encoded information;

FIG. 5 is a circuit diagram, primarily in block form, of a system, including equalizers, for inclusion in the embodiment shown in FIG. 4 to process digitally the analog signals received at the client and to produce signals representative of individual ones of the plurality of amplitude levels such as +1, 0 and −1;

FIG. 6 is a curve schematically illustrating the pulse response of a link (e.g. unshielded twisted pairs of wires) connected between the repeater and the client in the system shown in FIGS. 4 and 5;

FIG. 7 is a curve similar to that shown in FIG. 6 and illustrates the response of the system after an operation of a high pass filter included in the embodiment shown in FIG. 5 in limiting the length of a tail in the cable response shown in FIG. 6;

FIG. 8 is a curve similar to that shown in FIGS. 5 and 6 and illustrates the response of the system after an operation of a tail canceller included in the embodiment shown in FIG. 5 in limiting the length of the tail in the cable response shown in FIG. 6;

FIG. 9 shows curve illustrating the pattern of digital signals encoded at the repeater at the different amplitude levels such as +1, 0 and −1 and the pattern of the analog signals received at the client as a result of such encoding at the repeater;

FIG. 10 illustrates the adaptive thresholds for controlling whether the digital signals produced at each instant at the client represent individual ones of a plurality of amplitude levels such as +1, 0 and −1;

FIG. 11 shows different timing relationships between (a) a voltage assumed at the client to be at a zero crossing in the production of digital signals at the client and (b) a zero crossing of a clock signal at a particular frequency, these timing relationships being used to adjust the time at which the voltage is assumed to be at the zero crossing;

FIG. 12 illustrates the timing offset, made in the voltage assumed at the client to be at a zero crossing in the production of digital signals at the client, to compensate for the phase degradation produced during the passage of signals through the unshielded twisted pair of wires connected between the repeater and the client; and

FIG. 13 provides timing relationships similar to those shown in FIG. 11 but including the effects of the offset shown in FIG. 12.

DETAILED DESCRIPTION OF THE INVENTION

The discussion in this specification may be considered to relate specifically to a 100BASE-TX system for the purposes of explanation and understanding of the invention. However, it will be understood that the concepts of this invention and the scope of the claims apply to other types of systems than the 100BASE-TX system. For example, the concept of this invention and the scope of the claims apply to any 100BASE-TX system. For example, the concepts of the invention and the scope of the claims also apply to other systems than 100BASE-TX systems.

FIG. 1 illustrates a system, generally indicated at **10**, of the prior art. The system includes a repeater **12** and a plurality of clients **14, 16** and **18**. The repeater **12** facilitates the exchange of data packets between the repeater and the clients **14, 16** and **18** and among the clients. Each of the clients **14, 16** and **18** may be a computer, a facsimile machine, another repeater or a number of other different types of equipment. The clients **14, 16** and **18** may be respectively connected to the repeater **12** as by cable or links **20, 22** and **24**. The cables or links **20, 22** and **24** may be respectively connected to ports **26, 26** and **30** in the repeater **12**.

Although the following discussion relates to the transfer of information from the repeater **12** to individual ones of the clients **14, 16** and **18**, it will be appreciated that the information transfer may be from individual ones of the clients **14, 16** and **18** to the repeater **12** without departing from the scope of the invention. Furthermore, a different number of clients than three (3) may be connected to the repeater **12** without departing from the scope of the invention.

The cables or links **20, 22** and **24** may constitute pairs of unshielded twisted wires. Two pairs of such wires may be provided between the repeater **12** and each individual one of the clients **14, 16** and **18**. One pair of such wires provides for a transmission of information from the repeater **12** to the individual one of the clients **14, 16** and **18**. The other pair of such wires provides for the transmission of information from the individual one of the clients **14, 16** and **18** to the repeater **12**.

In the prior art, each of the links **14, 16** and **18** severely distorts the transmitted data packets. The amount of the degradation rapidly increases with increases in the length of the link. The degradation results from Inter Symbol Interference (ISI), signal attenuation, crosstalk, clock jitter, etc. Therefore, an adaptor is provided to couple data reliably to and from the link. The adaptor provides the interface to a computer on one side (e.g., ISA, EISA, PCI, etc.) of the adaptor and to the links such as the link **14** on the other side of the adaptor. It can also include circuitry such as a transducer to transmit data to, and receive data from, a link such as the links **14, 16** and **18**.

A transceiver generally indicated at **32** is shown in FIG. **2** and is known in the prior art with respect to most of the blocks shown in FIG. **2**. The transceiver **32** includes a standard connector designated as a Media Independent Interface (MII) **34**. The Media Independent Interface **34** may be a four (4)-bit wide data path in both the transmit and receive directions. Clocked at a suitable frequency such as 25 MHz, it results in a net throughput in both directions of data at a suitable rate such as 100 Mb/sec. It provides a symmetrical interface in both the transmit and receive directions and may have a total of forty (40) clock, data and control pins.

The input data passes through the Media Independent Interface **34** in FIG. **2** to a 4B5B Encoder **36**. The input data is grouped into nibbles (or groups of four (4) bits each). Each 4-bit nibble is then encoded to produce a five (5)-bit symbol. The 4B5B encoding was originally provided to (1) maintain dc balanced codes—in other words, equal numbers of 1's

and 0's, (2) introduce redundancy so that control information can be distinguished from data, and (3) provide sufficient transitions to facilitate clock recovery. A consequence of 4B5B encoding is that the data rate increases to a suitable rate such as 125 Mb/sec. and the coding efficiency is reduced to eighty percent (80%) because of this increase in data rate without a corresponding increase in the amount of data processed.

The 5B encoded symbols from the encoder **36** are introduced to a scrambler **38** in FIG. **2**. The 5B encoded symbols are scrambled to ensure that the transmitted spectrum complies with the Federal Communications Commission (FCC) mandates on EMI. The scrambler **38** may be a maximal-length Pseudo Noise (PN) sequence generator with a period of 2047 bits. It is generated by an 11-b linear feedback shift register (LFSR). The output bits from the scrambler **38** are generated recursively as $X(n)=X(n-11)+X(n-9)$. The pseudo-random bit stream produced by the scrambler **38** is exclusive-or'd with the transmit datastream. Scrambling destroys the dc balance and transition properties of the 5B codes.

The scrambled bits are indicated schematically at **40** in FIG. **3**. The scrambled bits **40** are encoded by an MLT-3 encoder **42** to produce bits indicated at **44** in FIG. **3**. The scrambled bits **40** provide a binary 1 when a transition is to be made in the amplitude level between symbol values of +1, 0 and −1. If a scrambled bit is a 0, the amplitude level of the previous bit in the sequence **44** is retained. By controlling the transitions (not allowing a direct transition between states +1 and −1), MLT-3 signalling limits the maximum frequency to 31.25 MHz (Nyquist frequency is 62.5 MHz).

The signals from the MLT-3 encoder **42** are introduced to a digital-to-analog converter **46** and the resultant analog signals are passed through one of the links such as the link in FIG. **1**. The signals at the other end of the link such as the link **20** are then processed by an analog equalizer **47** and the resultant signals are introduced to an MLT-3 decoder **50**. The MLT-3 decoder operates to decode the signals previously encoded by the MLT-3 encoder **42**. The decoded signals then pass to a descrambler **52** which operates to descramble the signals previously scrambled by the scrambler **38**. A 4B5B decoder then operates to decode to four (4) bits the five (5) bit encoding provided by the encoder **36**. The signals in the four (4) bit format then pass to the Media Independent Interface **34**.

The signals passing through the link such as the link **14** in FIG. **1** have not been converted in the prior art to the digital form such as provided as at **48** in FIG. **4** in the embodiment of this invention. Instead, the signals passing through the link such as the link **20** have been processed in the prior art in the analog form. This has prevented the distortions produced in the links such as the link **20** from being eliminated to the extent that they are eliminated when the signals are processed in the digital form as in the embodiment of this invention. Furthermore, as will be seen from the subsequent discussions, applicants use individual techniques in this invention to process the signals in the digital form. These individual techniques have not been provided in the prior art. These individual techniques cause the information represented by the digital signals to be recovered with an enhanced accuracy relative to that obtained in the prior art.

FIG. **4** illustrates a circuit diagram, primarily in block form, of applicants' invention when incorporated in the prior art system shown in FIG. **2**. In FIG. **4**, the blocks common to the blocks shown in FIG. **2** are given the same numerical designations as the corresponding blocks shown in FIG. **2**. However, additional blocks are shown in FIG. **4** and these are given individual identifications in FIG. **4**. These include a transformer **60** between the digital-to-analog converter **46** and the link **20** and a transformer **62** between the link **20** and the analog-to-digital converter **48**.

A block generally indicated at **64** and generically designated as an equalizer receives the output of the analog-to-digital converter **48** in FIG. **4**. The equalizer **64** is shown in detail in FIG. **5** and will be described in detail subsequently. The digital signals from the analog-to-digital converter **48** are also introduced to a timing recovery stage **66**, the output of which passes to the analog-to-digital converter **48** to control the operation of the converter. The operation of the timing recovery stage **66** is controlled by a clock signal generator **68** which generates clock signals at a particular frequency such as approximately 125 MHz. The operation of the clock signal generator **68** may be crystal controlled as at **70**. In addition to receiving inputs from the analog-to-digital converter **48**, the timing recovery stage receives as an input the output from the equalizer **64**. The output of the equalizer **64** also passes to the MLT-3 decoder **50** also shown in FIG. **2**.

As previously indicated, the MLT-3 encoder **42** provides digital signals at a suitable frequency such as approximately 125 MHz. These signals are converted to analog signals by the converter **46**. After being introduced to the transformer **60**, the analog signals are passed through the link such as the link **20** to the transformer **62**, which introduces the signals to the analog-to-digital converter **48**.

FIG. **9** illustrates at **70** the signals produced by the MLT encoder **42**. As will be seen, the signals from the encoder **42** have at each instant one of three (3) amplitude levels such as +1, 0 and −1 to represent information. FIG. **9** also illustrates at **72** the signals received at the analog-to-digital converter **48**. As will be seen, there is a considerable degradation or distortion of the signals **72** relative to the signals **70**. This degradation is produced in the link **20** and is also produced because of the interference provided by the signals in the links **22** and **24**.

It is desirable for the converter **48** to sample the analog signals at the zero crossing and peak amplitude of the waveform **70**. In this way, the converter **48** will provide an indication of the amplitude level of the encoded signals from the encoder **42**. For example, if the converter **48** samples the analog signals in FIG. **9** at the times indicated at **74**, **76** and **78**, the converter will produce digital signals respectively representing the analog levels +1, 0 and −1. However, if the converter **48** samples the signals at a time indicated at **80** or at a time indicated at **82**, the converter will produce digital signals which may not represent the proper one of the analog levels +1, 0 and −1. This may cause errors to be produced in the reproduction of the information represented by the digital signals produced by the converter **48**.

The timing recovery stage **66** operates at a suitable frequency such as approximately 125 MHz to produce digital signals having amplitudes corresponding to the magnitudes of the analog signals **72** in FIG. **9** at the instants of conversion. The timing recovery stage **66** operates to adjust the times that the digital signals are produced by the converter **48** so that these signals occur at the zero crossings and the peak amplitudes of the waveform **70**. In this way, the digital signals will be produced by the converter **48** at times such as the times **74**, **76** and **78** in FIG. **9** rather than at times such as the times **80** and **82** in FIG. **9**.

FIG. **10** illustrates how the timing recovery stage **66** initially operates to determine whether each digital conver-

sion has an amplitude level representing +1, 0 or −1. For an analog voltage between −0.5 volts and +0.5 volts, the amplitude level of the digital conversion of this analog voltage is initially assumed to be 0. For an analog voltage with a positive value greater than +0.5 volts, the amplitude level of the digital conversion of the analog voltage initially is assumed to be +1. When the analog voltage has a negative value with an absolute magnitude greater than 0.5 volts, the amplitude level of the digital conversion of the analog voltage is initially assumed to be −1. These assumptions are made because of the considerable distortion in the characteristics of the signals **72** (FIG. **9**) introduced to the converter **48** relative to the characteristics of the signals **70** produced by the encoder **42**.

FIG. **11** indicates how phase adjustments are initially made for different operating conditions to have the time assumed by the converter **48** for the zero crossing of the digital voltage V_o coincide in time with the time for the zero crossing of the clock signals from the clock generator **68**. FIG. **11** indicates four (4) different conditions in which phase adjustments are made in the time assumed by the converter **48** for the zero crossing of the digital voltage. For each of these four (4) conditions, the indication "0" represents the time at which the clock signal from the clock signal generator **68** crosses the zero line. Furthermore, for each of these four (4) operating conditions, V_o indicates the voltage which is actually produced by the analog-to-digital converter **48** at the time assumed by the converter to constitute the time at which a zero crossing occurs.

As will be seen in FIG. **4**, the digital signals from the analog-to-digital converter **48** are shown as being introduced directly to the timing recovery stage **66**. This occurs before the equalizer **64** becomes operative to determine whether each of the digital signals from the converter **48** has an amplitude level of +1, 0, or −1. The digital signals from the converter **48** are initially processed by the timing recovery stage **66** because no significant information is obtained from the operation of the equalizer **64** until a coarse adjustment has been provided by the timing recovery stage in the times for the production of the voltage V_o.

The first condition in FIG. **11** is designated as "+0 transition." In this condition, the voltage V_o is positive as indicated by a "+" sign above and to the left of the "V_o" designation. Furthermore, the V_o voltage occurs before the "0" voltage indicating the time at which the clock signals from the generator **68** cross the zero line. As shown in the curve at the left in FIG. **11**, the voltage decreases from V_o to the "0" line crossing. Under such conditions, the time for the production of the digital signals by the converter **48** would be moved to the right—or, from a time standpoint, delayed—in FIG. **11** to have the V_o indication coincide in time with the "0" indication.

If the V_o voltage should be negative with the same shape of curve as shown in the "+0" transition in FIG. **11**, the V_o voltage would be below and to the right of the "0" indication. Under such circumstances, the time for the production of the digital signals by the converter **48** would be moved to the left—or, from a time standpoint, advanced—in FIG. **11** to have V_o coincide in time with the "0" indication.

The condition second from the left in FIG. **11** is designated as "−0 transition." In that condition, V_o is below the "0" indication from a voltage standpoint and occurs to the left—or, from a time standpoint, before—the "0" indication. Furthermore, the V_o voltage is negative as indicated by a "−" sign to the left and below the "0" and "V_o" indications. Under such circumstances, the voltage V_o is moved to the right—or, from a time standpoint, delayed—to have the V_o indication coincide in time with the "0" indication.

If the V_o indication should be positive with the same shape of curve as shown in the "−0" transition in FIG. **11**, the V_o voltage would be above and to the right of the "0" indication. Under such circumstances, the production of the voltage V_o would be moved to the left by the converter **48**—or, from a time standpoint, advanced—in FIG. **11** to have V_o coincide in time with the "0" indication.

The third condition in FIG. **11** is designated as a "0+" transition. In that condition, the "0" indication is below and to the left—or, from a time standpoint, before—the V_o indication. In other words, V_o is positive relative to the "0" indication. This is indicated by a "+" sign above and to the right of the V_o indication. Under such circumstances, the production of the V_o indication would be moved to the left—or, from a time standpoint, advanced—in FIG. **11** to have V_o coincide in time with the "0" indication.

If the V_o indication should be negative with the same shape of curve as shown in the "0+" transition in FIG. **11**, the V_o voltage would be below and to the left of the "0" indication. Under such circumstances, the time for the production of the digital signals by the converter **48** would be moved to the right—or, from a time standpoint, delayed—in FIG. **11** to have V_o coincide in time with the "0" indication.

The fourth condition in FIG. **11** is designated as a "0−" transition. In that condition, the "0" indication is above and to the left—or, from a time standpoint, before—the V_o indication. In other words, V_o is negative relative to the "0" indication. This is indicated by a "−" sign below and to the right of the V_o indication. Under such circumstances, the timing of the V_o indication would be moved to the left—or, from a time standpoint, advanced—in FIG. **11** to have V_o coincide in time with the "0" indication.

If the V_o indication should be positive with the same shape of curve as shown in the "0−" transition in FIG. **11**, the V_o voltage should be above and to the left of the "0" indication. Under such circumstances, the production of the digital signals by the converter would be moved to the right—or, from a time standpoint, delayed—in FIG. **11** to have V_o coincide in time with the "0" indication.

After the time of the V_o indication has been adjusted as shown in FIG. **11** and discussed above to have it coincide in time with the "0" indication, the digital signals from the analog-to-digital converter **48** are introduced to the equalizer **64** in FIG. **4**. The equalizer **64** is shown in detail in FIG. **5**. In FIG. **5**, the signals from the analog-to-digital converter **48** are introduced to a high pass filter **100**. The signals from the high pass filter **100** in turn pass to a feed forward equalizer **102**. A feed forward equalizer such as the equalizer **100** is known in the prior art. The signals from the feed forward equalizer **102** are introduced to an adder **104** which also receives signals from an adder **106**.

The adder **106** receives the outputs from a decision feedback equalizer **108** and from a tail canceller **110**. A decision feedback analyzer such as the equalizer **100** is known in the prior art. The signals from the decision feedback equalizer **108** are also introduced to the tail canceller **110**. Signals are introduced to the decision feedback equalizer **108** from a quantizer **112**. The quantizer **112** receives the output from the adder **104**. The quantizer **112** (also known as a slicer) is known in the art.

A feed forward equalizer, a decision feedback equalizer and a slicer are shown in FIG. **7** and are disclosed in U.S. Pat. No. 5,604,741, issued to Henry Samueli, Mark Berman

and Fan Lu on Feb. 18, 1997, for an "Ethernet System" and assigned of record to the assignee of record of this application. Reference is made to U.S. Pat. No. 5,604,741 if any additional disclosure is necessary to complete the disclosure of the feed forward equalizer **102**, the decision feedback equalizer **108**, the quantizer **112** and the adder **104** in this application.

As will be seen in FIG. **6**, a composite signal generally indicated at **120** is shown as being comprised of a left portion **122** and a right portion **124**. Each of the portions **122** and **124** has distortions. The distortions in the left portion **122** may be considered as a pre-cursor response. The distortions in the right portion **124** may be considered as a post-cursor response. The distortions result in part from the fact that the digital signals representing information or data develop tails as they travel through the unshielded twisted pairs of wires defined as the links **20**, **22** and **24**. The distortions also result in part from the reflections from the links **20**, **22** and **24** to the repeater **12** in FIG. **1**.

The feed forward equalizer **102** may be considered to correct for distortions (or pre-cursor responses) in the portion **122** of the composite signal **120**. The decision feedback equalizer **124** may be considered to correct for distortions (or post-cursor responses) in the portion **124** of the composite signal **120**. As will be seen in FIG. **6**, the distortions (or post-cursor response) in the portion **124** of the composite signal **120** result in a tail **126**. This tail extends for a considerable period of time as indicated by the number of taps along the horizontal axis in FIG. **6**. If corrections had to be provided for as many as fifty (50) taps to eliminate or significantly reduce the tail **126**, this would unduly complicate the construction of the decision feedback equalizer **64** in FIG. **4**.

To simplify the construction of the equalizer **64** in FIG. **4**, the high pass filter **100** and the tail canceller **110** are included in the embodiment of the equalizer as shown in FIG. **5**. The high pass filter **100** operates to block the passage of the low frequency signals which constitute a significant portion of the tail **126**. As a result of the operation of the high pass filter **100**, the length of the tail **126** is significantly reduced as indicated at **128** in FIG. **7**. As will be seen schematically by a comparison of FIGS. **6** and **7**, the number of taps is reduced from approximately fifty (50) in FIG. **6** to approximately (twenty) 20 in FIG. **7** because of the inclusion of the high pass filter **100** in FIG. **5**.

The tail canceller **110** reduces the number of taps required in the decision feedback equalizer. This may be seen from FIG. **8**, which illustrates the tail on an enlarged schematic basis. As shown in FIG. **8**, the tail decays substantially on an exponential basis from a position **130** which is the last tap of the decision feedback equalizer. This exponential decay is predictable. The tail canceller **110** accurately predicts the shape of this exponential decay and provides a cancellation of this exponential decay The tail canceller **110** may constitute a first order recursive filter.

The output from the equalizer **64** in FIG. **4** is obtained from the quantizer **112** in FIG. **5**. The quantizer **112** provides a plurality (e.g. 3) of progressive amplitude values and determines the particular one of the three (3) amplitude values closest to the output from the adder **104** for each of the digital signals produced by the converter **48**. The quantizer **112** provides this output on a line **114** for each of the digital signals to indicate the data or information represented by such digital signals. In this way, the equalizer **64** in FIG. **4** restores the analog levels of the digital signals to the analog levels of these digital signals at the repeater **12** even with the distortions produced in these signals as they pass through the unshielded twisted pairs of wires defining the link such as the link **14**.

The signals from the quantizer **112** in FIG. **5** are introduced to the timing recovery stage **66** in FIG. **4**. The timing recovery stage provides a fine regulation of the time at which the analog-to-digital converter **42** produces the voltage V_o. As a first step in this regulation, the timing recovery stage **66** determines the amount of offset produced in the voltage V_o as a result of the distortion produced in the unshielded twisted pairs of wires constituting the link such as the link **20**.

FIG. **12** illustrates the voltage V_o at **140** and illustrates at **142** the "0" indication corresponding to the time at which the clock signal provided by the generator **68** crosses the zero axis. FIG. **12** also illustrates at **144** the shift in phase of the voltage V_o as a result of the offset produced by the unshielded twisted pair of wires constituting the link such as the link **20**. This voltage with the shifted phase is designated as $Z_o=V_o-V_{off}$ where V_{off} is the offset voltage resulting from the phase distortion or degradation produced by the unshielded twisted pair of wires constituting the link such as the link **20**.

FIG. **13** provides a number of schematic representations similar to those shown in FIG. **11** and discussed above. However, many of the representations include a consideration of the offset voltage V_o discussed in the previous paragraph and shown in FIG. **12**. The first condition shown in FIG. **13** is designated as a "+0" transition. In this transition, V_o-V_{off} has a value greater than 0. Furthermore, V_o-V_{off} has a positive value as indicated by the "+" sign above and to the left of V_o. Under such circumstances, V_o is shifted to the right—or, from a time standpoint, is delayed—so that V_o-V_{off} will correspond in time to the zero crossing of the clock signals from the generator **68**.

If V_o-V_{off} should be negative with the same shape of curve as shown in the "+0" transition in FIG. **13**, the V_o-V_{off} indication would be below and to the right of the "0" indication. Under such circumstances, the time for the production of the digital signals by the converter **48** would be moved to the left—or, from a time standpoint, advanced—in FIG. **13** to have V_o coincide in time with the "0" indication.

The second condition in FIG. **13** is designated as a "−0" transition. In this transition, V_o+V_{off} is less than 0. V_{off} is added to V_o in this transition because V_o is negative and the delay represented by V_{off} advances V_o toward a value of 0. In this transition, the "0" indication is above and to the right of the V_o indication. This is indicated by a "−" sign below and to the left of the V_o indication. Under such circumstances, the timing of the V_o indication would be moved to the right—or, from a time standpoint, delayed—in FIG. **13** to have V_o coincide in time with the "0" indication.

If V_o+V_{off} should be greater than 0 with the same shape of curve as shown in the "−0" transition in FIG. **13**, the V_o+V_{off} voltage would be above and to the right of the "0" indication in FIG. **13**. Under such circumstances, the V_o would be moved to the left—or, from a time standpoint, advanced—in FIG. **13** to have V_o+V_{off} coincide in time with the "0" indication.

The third condition in FIG. **13** is designated as a "+0−" transition. In this transition, V_o-V_{off} is greater than 0. Furthermore, the transition is from a+value to a value of 0 and then to a−value. (This is why it is designated as "+0−".) Under such circumstances, V_o is moved to the right—or, from a time standpoint, delayed—in FIG. **13** to have V_o coincide in time with the "0" indication.

if $V_o - V_{off}$ should be less than 0 with the same shape of curve as shown in the "+0–" transition in FIG. **13**, the $V_o - V_{off}$ indication would be below and to the right of the "0" indication. Under such circumstances, the V_o indication would be moved to the left—or, from a time standpoint, advanced—to have V_o coincide in time with the "0" indication.

The fourth condition in FIG. **13** is designated as a "–0+" transition. In this transition, $V_o + V_{off}$ is less than 0. V_{off} is added to V_o in this transition because V_o is negative and the delay represented by V_{off} advances V_o toward a value of 0. Furthermore, the transition is from a–value to a value of 0 and then to a+value. (This is why it is designated as "–0+".) Under such circumstances, V_o is moved to the right—or, from a time standpoint, delayed—to have V_o coincide in time with the "0" indication.

If $V_o + V_{off}$ should be greater than 0 with the same shape of curve as shown in the "–0+" transition in FIG. **13**, the $V_o + V_{off}$ indication would be above and to the right of the "0" indication. Under such circumstances, the V_o indication would be moved to the left—or, from a time standpoint, advanced—to have V_o coincide in time with the "0" indication.

The fifth (5th) condition in FIG. **13** is designated as a "00–" transition. (This results from the fact that the first two (2) positions in this transition have values of 0 or values close to 0 and the third position in this transition is negative). The voltage V_o is between the two (2) zero (0) indications and has a value greater than the two (2) zero (0) indications. Under such circumstances, the V_o voltage is moved to the right—or, from a time standpoint, is delayed—to have the V_o voltage correspond in time with the second of the two zero (0) indications.

If V_o should be less than the two 0 indications with the same shape of curve as shown in the "00–" transition in FIG. **13**, the V_o voltage should be below and to the right of the second of the two zero (0) indications. Under such circumstances, the V_o voltage is moved to the left—or, from a time standpoint, advanced—in FIG. 13 to have the V_o voltage correspond in time with the second of the two zero (0) indications.

The sixth condition in FIG. **13** is designated as a "00+" transition. (This results from the fact that the first two (2) positions in this transition have values of 0 or values close to 0 and the third position in this transition is positive.) The voltage V_o is between the two zero (0) indications and has a value less than the two zero (0) indications. Under such circumstances, the V_o voltage is moved to the right—or, from a time standpoint, delayed—to have the V_o voltage correspond in time with the second of the two zero (0) indications.

If V_o should be greater than the two zero (0) indications, with the same shape of curve as shown in the "00+" transition in FIG. **13**, the V_o voltage would be above and to the right of the second of the two zero (0) indications. Under such circumstances, the V_o voltage is moved to the left—or, from a time standpoint, advanced—to have the V_o voltage correspond in time with the second of the two zero (0) conditions.

The system and method of this invention have certain important advantages. They provide a conversion of the received analog signals to digital signals. They provide for a processing of the digital signals by the timing recovery stage **66** to have the digital conversions occur at the zero crossings of a reference clock signal generated by the generator **68**. In this way, the analog signals can be sampled digitally at the times at which the amplitudes of the analog signals represent individual ones of analog levels +1, 0 and –1. This processing of the digital signals by the timing recovery stage **66** initially provides a coarse regulation of the time for the digital conversions by the converter **48**.

Subsequently the equalizer **64** operates upon the digital signals from the converter **48** to determine whether the amplitudes of the digital signals have analog values of +1, 0 or –1. The operation of the equalizer **64** to determine the amplitudes of the digital signal is facilitated by the inclusion of the high pass filter **100** and the canceller **112** to limit the length of the tail in the digital signals. The timing recovery stage **66** then provides fine regulation of the signals from the equalizer **64** to have the digital processing by the converter **48** occur at the zero crossings of the clock signals from the clock signal generator **68**.

Although this invention has been disclosed and illustrated with reference to particular embodiments, the principles involved are susceptible for use in numerous other embodiments which will be apparent to persons of ordinary skill in the art. The invention is, therefore, to be limited only as indicated by the scope of the appended claims.

What is claimed is:

1. In combination for operating upon digital signals provided at a particular frequency, which digital signals have been scrambled and encoded and then converted to analog signals to recover the information represented by such digital signals,

first means for providing clock signals at the particular frequency,

second means for converting the analog signals to digital signals at the particular frequency,

third means responsive to the clock signals and to the digital signals for initially providing a coarse adjustment in the times at which the analog signals are converted to the digital signals to obtain a zero value for the digital signals at the zero crossings of the clock signals;

an equalizer for providing signals for eliminating the effects, from each individual one of the digital signals, of the digital signals adjacent in time to such individual one of the digital signals,

fourth means responsive to the initial operation of the third means for activating the equalizer, and

fifth means responsive to the activation of the equalizer for providing a fine adjustment in the times at which the analog signals are converted to the digital signals to obtain a zero value for the digital signals at the zero crossings of the clock signals.

2. In a combination as set forth in claim **1** wherein

the equalizer includes a feed forward equalizer for eliminating from each digital signal the effects on such digital signal from previous digital signals and wherein,

each digital signal includes a tail and wherein

the equalizer includes a decision feedback equalizer for eliminating the tail from each digital signal.

3. In combination as set forth in claim **2** wherein

the third means is operative to adjust the time for the conversion of the analog signals to the digital signals to have such time coincide with the time for the production of the zero voltage in the clock signals,

a link extends from the position of the encoding to the position of the second means and wherein

the fifth means compensates for the length of the link in providing the adjustment in the times at which analog

signals are converted to the digital signals to have such time coincide with the zero crossing of the clock signals.

4. In a combination as set forth in claim 1 wherein

the third means is responsive to the clock signals and to the relative timing of the zero voltage in the clock signals and the digital voltage at the time of the conversion of the analog signals to the digital signals and responsive to the polarity of the difference between the zero voltage in the clock signals and the digital voltage at the time of the conversion of the analog signals to the digital signals for adjusting the timing of the conversion of the digital signals in a direction to minimize the time between the zero voltage in the clock signals and the conversion of the analog signals to the digital signals.

5. In a combination as set forth in claim 1 wherein

the third means is operative to adjust the time for the conversion of the analog signals to the digital signals to have such time coincide with the time for the production of the zero voltage in the clock signals.

6. In a combination as set forth in claim 1 wherein

a link extends from the position of the encoding to the position of the second means and wherein

the fifth means compensates for the length of the link in providing the adjustment in the time at which analog signals are converted to the digital signals to obtain the coincidence between such conversion and the zero crossing of the clock signals.

7. In combination for operating upon digital signals provided at a particular frequency and representing information, which digital signals have been scrambled and encoded and then converted to analog signals, to recover the information represented by the digital signals,

first means for providing clock signals at the particular frequency,

second means for converting the analog signals to digital signals at the particular frequency,

third means responsive to the clock signals and the digital signals for providing coarse adjustments in the phase of the digital signals to obtain a correspondence in phase between the digital signals and the clock signals,

fourth means responsive to the digital signals for minimizing, in each individual one of the digital signals, the responses on such individual one of the digital signals of adjacent ones of the digital signals, and

fifth means responsive to the digital signals from the fourth means for providing fine adjustments in the phase of the digital signals to obtain a correspondence in phase between the digital signals and the clock signals.

8. In a combination as set forth in claim 7 wherein

the third means is initially operative without an operation of the fourth and fifth means and wherein the fourth and fifth means are subsequently operative without an operation of the third means.

9. In a combination as set forth in claim 8 wherein

the digital signals from the second means have leading and trailing portions and wherein

the fourth means includes a first equalizer responsive to the digital signals from the second means for minimizing, in the leading portion of each individual one of the digital signals, the responses of the adjacent digital signals and for minimizing, in the trailing portion of each individual one of the digital signals, the responses of the adjacent digital signals and wherein

a high pass filter operates to limit the length of the post-cursor response before the introduction of the digital signals to the second equalizer and wherein

a tail canceller is included to predict the shape of the trailing end of the post-cursor response and to provide a signal with such predicted shape to cancel the trailing end of the post-cursor response and wherein

the third means determines the phase of the digital signals at assumed zero crossings of such digital signals relative to the phase of the zero crossings of the clock signals and adjusts the phase of the assumed zero crossings to coincide with the phase of the zero crossings of the clock signals and wherein

a link is provided between the position of converting the analog signals and the second means and wherein

an offset is provided in the phase of the digital signals to compensate for the effect of the link on the digital signals and wherein

the fifth means determines the offset in the phase of the digital signals relative to the zero crossings of the clock signals and adjusts the phase of the digital signals to coincide with the phase of the zero crossings of the clock signals.

10. In a combination as set forth in claim 7 wherein

the digital signals from the second means have leading and trailing portions and wherein

the fourth means includes a first equalizer responsive to the digital signals from the second means for minimizing, in the leading portion of each individual one of the digital signals, the responses of the adjacent digital signals and for minimizing, in the trailing portion of each individual one of the digital signals, the responses of the adjacent digital signals.

11. In a combination as set forth in claim 7 wherein

each of the digital signals has a relatively short pre-cursor response and a relatively long post-cursor response and wherein

a first equalizer operates to inhibit the short pre-cursor response and a second equalizer operates to inhibit the long post-cursor response and wherein

a high pass filter operates to limit the length of the post-cursor response before the introduction of the digital signals to the second equalizer.

12. In a combination as set forth in claim 11 wherein

a tail canceller is included to predict the shape of the trailing end of the post-cursor response and to provide a signal with such predicted shape to cancel the trailing end of the post-cursor response.

13. In a combination as set forth in claim 7 wherein

the third means determines the phase of the digital signals at assumed zero crossings of such digital signals relative to the phase of the zero crossings of the clock signals and adjusts the phase of the assumed zero crossings to coincide with the phase of the zero crossings of the clock signals.

14. In a combination as set forth in claim 7 wherein

a link is provided between the position of conversion of the digital signals and the second means and wherein

an offset is provided in the phase of the digital signals to compensate for the effect of the link on the digital signals and wherein

the fifth means determines the offset in the phase of the digital signals relative to the zero crossings of the clock

signals and adjusts the phase of the digital signals to coincide with the phase of the zero crossings of the clock signals.

15. In combination for operating upon digital signals provided at a particular frequency, which digital signals have been scrambled and encoded and then converted to analog signals, to recover the information represented by such digital signals,

first means for providing clock signals at the particular frequency,

second means for converting the analog signals to digital signals at the particular frequency,

there being a link between the position of converting to the analog signals and the second means,

the digital signals having pre-cursor and post-cursor responses,

third means for offsetting the phase of the digital signals in accordance with the characteristics of the link,

fourth means responsive to the digital signals with the offset phase for limiting the pre-cursor and post-cursor responses, and

fifth means for adjusting the offset phases of the digital signals to have the times for the zero crossings of the digital signals correspond to the times for the zero crossings of the clock signals.

16. In a combination as set forth in claim **15** wherein

the fourth means includes a first equalizer for inhibiting the pre-cursor response in the digital signals and includes a second equalizer for inhibiting the post-cursor response in the digital signals.

17. In a combination as set forth in claim **16**,

a high pass filter operative to limit the length of the post-cursor response before the operation of the second equalizer.

18. In a combination as set forth in claim **17**,

a tail canceller for limiting the length of the post-cursor response,

a first adder for combining the outputs of the second equalizer and the tail canceller,

a second adder for combining the outputs of the first and second equalizers, and

a quantizer responsive to the output of the second adder at each instant for selecting, for each of the digital signals, a particular one of a plurality of digital values closest to the output of the second adder at that instant and for using the selected one of the digital values as the peak value of the digital signal at that instant,

the fifth means being operative to determine the polarity of the value of the digital signals at times assumed for the zero crossing of the digital signals and to determine the relative times of occurrence of the assumed zero crossings of the digital signals and the zero crossings of the clock signals and to adjust the times assumed for the zero crossings of the digital signals in accordance with such determinations to have the times assumed for the zero crossings of the digital signals coincide with the times for the zero crossings of the clock signals.

19. In a combination as set forth in claim **16**,

a tail canceller for limiting the length of the post-cursor response, and

an adder for combining the outputs of the second equalizer and the tail canceller.

20. In a combination as set forth in claim **16**,

an adder for combining the outputs of the first and second equalizers, and

a quantizer responsive to the output of the adder at each instant for selecting, in each of the digital signals, a particular one of a plurality of digital values closest to the output of the adder at that instant and for using the selected one of the digital values as the peak value of such digital signal at that instant.

21. In a combination as set forth in claim **16**, wherein

the fifth means determines the polarity of the digital signals at times assumed for the zero crossings of the digital signals and determines the relative times of occurrence of the assumed zero crossings of the digital signals and the zero crossings of the clock signals and adjusts the times assumed for the zero crossings of the digital signals in accordance with such determinations to have the times assumed for the zero crossings of the digital signals coincide with the times for the zero crossings of the clock signals.

22. In combination for operating upon digital signals provided at a particular frequency, which digital signals have been scrambled and encoded and then converted to analog signals, to recover the information represented by such digital signals,

first means for providing clock signals at the particular frequency,

second means for converting the analog signals to digital signals at the particular frequency,

third means responsive to the digital signals for providing a value of zero for the digital signals when the digital signals have a value within particular limits and for providing a value of +1 for positive values of the digital signals above the particular limits and for providing a value of −1 for negative values of the digital signals below the particular limits, and

fourth means for determining the times for the changes of the digital signals between the digital values of +1, 0 and −1 relative to the times for the zero crossings of the clock signals and for determining the polarity of the digital signals between such relative times and for changing the times for the production of the digital values of +1, 0 and −1 in accordance with such determinations.

23. In a combination as recited in claim **22** wherein

the analog signals are provided at a particular position and wherein

a link is provided between the particular position and the second means and wherein fifth means are provided for offsetting the time for the occurrence of the digital signals representing +1, 0 and −1 in accordance with the characteristics of the link.

24. In a combination as set forth in claim **23** wherein

a link is provided between the position of providing the analog signals and the second means and wherein means are provided for offsetting the time for the occurrence of the digital signals representing +1, 0 and −1 in accordance with the characteristics of the link and wherein

sixth means are provided for determining the offset times for the changes of the digital signals between the digital values of +1, 0 and −1 relative to the times for the zero crossings of the clock signals and for determining the polarity of the digital signals between such relative times and for changing the times for the production of the digital values between +1, 0 and −1 in accordance with such determinations.

25. In a combination as set forth in claim **24** wherein

the sixth means is operative after the operation of the fourth means and wherein

the digital signals include a pre-cursor response and a post-cursor response and wherein

seventh means are provided for minimizing the pre-cursor response and the post-cursor response after the operation of the fourth means but before the operation of the sixth means.

26. In a combination as set forth in claim **25** wherein

the seventh means includes a first equalizer for minimizing the pre-cursor response and a second equalizer for minimizing the post-cursor response and a quantizer operatively coupled to the first and second equalizers for selecting an individual one of the values of +1, 0 and −1 for each of the digital signals in accordance with the magnitude of the digital signals relative to the particular limits and wherein

the seventh means additionally includes a high pass filter and a tail canceller for limiting the time duration of the post-cursor response for each of the digital signals.

27. In a combination as set forth in claim **22** wherein

the digital signals include a pre-cursor response and a post-cursor response and wherein fifth means are provided for minimizing the pre-cursor response and the post-cursor response after the operation of the fourth means.

28. In a combination as set forth in claim **27** wherein

the digital signals include a pre-cursor response and a post-cursor response and wherein

sixth means are provided for minimizing the pre-cursor response and the post-cursor response and wherein

the sixth means includes a first equalizer for minimizing the pre-cursor response and a second equalizer for minimizing the post-cursor response and a quantizer operatively coupled to the first and second equalizers for selecting an individual one of the values of +1, 0 and −1 for each of the digital signals in accordance with the magnitude of the digital signals relative to the particular limits.

29. In combination for use in a system providing first signals having individual ones of a plurality of analog levels to represent information,

a repeater,

a plurality of clients,

pairs of unshielded twisted wires, each pair being disposed between the repeater and an individual one of the clients to transmit the first signals between the repeater and the individual one of the clients,

first means at each of the clients for receiving from the repeater the first signals having individual ones of the plurality of amplitude levels,

second means at each of the clients for providing clock signals at a particular frequency,

third means at each of the clients for converting to digital signals the first signals having the individual ones of the different amplitude levels, and

fourth means at each of the clients for adjusting the times of the digital conversions of the first signals having the different amplitude levels in accordance with the phase differences between the zero crossings of the clock signals and the digital conversions of the first signals.

30. In a combination as set forth in claim **29** wherein

a plurality of two pairs of unshielded twisted wires are provided and wherein each of the two pairs of the unshielded twisted wires is connected between the receiver and an individual one of the clients, one of the pairs providing for the transmission of information from the receiver to the individual one of the clients and the other of the pairs providing for the transmission of information from the individual one of the clients to the receiver.

31. In a combination as set forth in claim **30**, wherein

the pairs of the unshielded twisted wires between the receiver and the individual ones of the clients have impedance characteristics dependent upon the lengths of such wires and wherein

means are provided at each of the clients for offsetting the times of the digital conversions of the first signals in accordance with the impedance characteristics of the unshielded twisted pairs of wire between the repeaters and such clients.

32. In a combination as set forth in claim **31**, wherein

the pairs of the unshielded twisted wires between the receiver and the individual ones of the clients have impedance characteristics dependent upon the lengths of such wires and wherein

means are provided for offsetting the times of the digital conversions of the first signals at each of the clients in accordance with the impedance characteristics of the unshielded twisted pairs of wires between the repeater and such client.

33. In a combination as set forth in claim **32**,

there being unshielded twisted pairs of wires between the repeater and each of the clients, and

means at each of the clients for offsetting the phase of the digital signals from the quantizer at such client, before the fine adjustments in the phase of these signals at such client, in accordance with the characteristics of the unshielded twisted pairs of wires between the repeater and such client, to have the zero crossings of these signals coincide in phase with the zero crossings of the clock signals at such client.

34. In a combination as set forth in claim **30**, wherein

the digital conversions of the first signals at each of the clients have a pre-cursor response and a post-cursor response and wherein,

first equalizer means are provided at each of the clients for limiting the pre-cursor response at such client and wherein

second equalizer means are provided at each of the clients for limiting the post-cursor response at such client.

35. In a combination as set forth in claim **29**, wherein

the fourth means at each of the clients includes fifth means for initially providing a coarse adjustment in the times of the digital conversions of the first signals in accordance with the phase differences between the zero crossings of the clock signals and the digital conversions of the first signals and wherein

the fourth means at each of the clients includes sixth means for subsequently providing fine adjustments in the times of the digital conversions of the first signals at such client in accordance with the phase differences between the zero crossings of the clock signals and the digital conversions of the first signals.

36. In a combination as recited in claim **35** wherein

a plurality of two pairs of unshielded twisted wires are provided and wherein each of the two pairs of the unshielded twisted wires is connected between the receiver and an individual one of the clients, one of the pairs providing for the transmission of information from the receiver to the individual one of the clients and

the other of the pairs providing for the transmission of information from the individual one of the clients to the receiver and wherein

the pairs of the unshielded twisted wires between the receiver and the individual ones of the clients have impedance characteristics dependent upon the lengths of such wires and wherein

means are provided at each of the clients for offsetting the times of the digital conversions of the first signals in accordance with the impedance characteristics of the unshielded twisted pairs of wire at such client and wherein

the digital conversions of the first signals at each of the clients have a pre-cursor response and a post-cursor response and wherein

first equalizer means are provided at each of the clients for limiting the pre-cursor response at such client and wherein

second equalizer means are provided at each of the clients for limiting the post-cursor response at such client and wherein

an adaptive threshold is provided and wherein

the outputs of the first equalizer means and the second equalizer means at each of the clients are introduced to an adder at such client and wherein

the output of the adder at each of the clients is introduced to a quantizer at such client for selecting the individual one of the analog levels in the plurality for each of the digital signals at such client in accordance with the amplitude and polarity of such digital signal relative to the adaptive threshold and wherein

the information at each of the clients is provided by the sequence of the amplitude levels from the quantizer at such client.

37. In a combination as set forth in claim **36**, including,

a high pass filter responsive at each of the clients to the digital signals provided by the third means for limiting the post-cursor response before the introduction of the digital signals to the first and second equalizer means at such client, and

a tail canceller at each of the clients for attenuating the post-cursor response beyond a particular time for each of the digital signals at such client.

38. In a method of operating upon digital signals provided at a particular frequency, which digital signals have been scrambled and encoded and then converted to analog signals, to recover the information represented by such digital signals, the steps of:

providing clock signals at the particular frequency,

converting the analog signals to digital signals at the particular frequency,

converting the magnitude of each of the digital signals to the individual one of a plurality of amplitude values closest in value to the magnitude of such digital signal, and

determining the times for the changes of the digital signals between the individual one of the plurality of amplitude values relative to the times for the zero crossings of the clock signals to have the times for such changes coincide with the times for the zero crossings of the clock signals.

39. In a method as set forth in claim **38** wherein

the amplitude values in the plurality are +1, 0 and −1 and wherein

the step of converting involves the conversion of the magnitude of each of the digital signals to the individual one of the amplitudes +1, 0 and −1 closest in value to the magnitude of such digital signal.

40. In a method as set forth in claim **38** wherein

the analog signals are provided at a first particular position and wherein

the analog signals are converted to digital signals at a second particular position displaced from the first particular position and wherein

a link is provided between the first and second particular positions and wherein

the time for determining the changes of the digital signals between the individual ones of the plurality of amplitude values relative to the times for the zero crossings of the clock signals are offset by the characteristics of the link.

41. In a method as set forth in claim **40** wherein

the digital signals include a pre-cursor response and a post-cursor response and

wherein the pre-cursor response and the post-cursor response in the digital signals are minimized before the step of determining the times for the changes of the digital signals between the individual ones of the plurality of amplitude values relative to the times for the zero crossings of the clock signals and wherein

the amplitude values in the plurality are +1, 0 and −1 and wherein

the step of converting involves the conversion of the magnitude of each of the digital signals to the individual one of the amplitudes +1, 0 and −1 closest in value to the magnitude of such digital signal.

42. In a method as set forth in claim **38** wherein

the digital signals include a pre-cursor response and a post-cursor response and

wherein the pre-cursor response and the post-cursor response in the digital signals are minimized before the step of determining the times for the changes of the digital signals between the individual ones of the plurality of amplitude values relative to the times for the zero crossings of the clock signals.

43. In a method of operating upon digital signals provided at a particular frequency, which digital signals have been scrambled and encoded and then converted to analog signals, to recover the information represented by such digital signals, the steps of:

providing clock signals at the particular frequency,

converting the analog signals to digital signals at the particular frequency,

initially providing coarse adjustments in the phases at which the digital signals are produced thereby to have the times for the digital crossings of the digital signals coincide with the times for the zero crossings of the clock signals, and

subsequently providing fine adjustments in the phases at which the digital signals are produced thereby to have the times for the digital crossings of the digital signals coincide with the times for the zero crossings of the clock signals.

44. In a method as set forth in claim **43** wherein

the digital signals have a pre-cursor response and a post-cursor response and wherein

the pre-cursor response and the post-cursor response are minimized in the period between the coarse and fine adjustments in the time at which the digital signals are produced.

45. In a method as set forth in claim 44 wherein

the magnitude of each of these digital signals is adjusted, in the period of time between the coarse and fine adjustments in the phases of the digital signals, to provide an individual one of a plurality of amplitudes closest in value to such magnitude of such individual one of the digital signals.

46. In a method as set forth in claim 43 wherein

the magnitude of each of these digital signals is adjusted, in the period of time between the coarse and fine adjustments in the phases of the digital signals, to provide an individual one of a plurality of amplitudes closest in value to such magnitude of such individual one of the digital signals.

47. In combination for operating upon digital signals provided at a particular frequency, which digital signals have been scrambled and encoded and then converted to analog signals, to recover the information represented by such digital signals,

first means for providing clock signals at the particular frequency,

second means for converting the analog signals to digital signals at the particular frequency,

the digital signals having a pre-cursor response and a post-cursor response,

a feed forward equalizer for producing signals inhibiting the pre-cursor response,

a decision feedback equalizer for producing signals inhibiting the post-cursor response,

an adder for combining the signals from the feed forward equalizer and the decision feedback equalizer to provide resultant signals,

a high pass filter for producing signals limiting the time duration of the post-cursor response before the introduction of the digital signals to the feed forward equalizer,

a tail canceller responsive to the digital signal from the feed forward equalizer for producing signals further limiting the time duration of the post-cursor response in the digital signals,

a quantizer responsive to the resultant signals from the adder for providing digital signals representing values of +1, 0 and −1 in accordance with the amplitudes of the resultant signals from the first adder, and

means responsive to the signals from the quantizer for adjusting the phase of such signals to have the zero crossings of these signals coincide in phase with the zero crossings of the clock signals.

48. In combination for operating upon digital signals provided at a particular frequency, which digital signals have been scrambled and encoded and then converted to analog signals, to recover the information represented by such digital signals,

first means for providing clock signals at the particular frequency,

second means for converting the analog signals to digital signals at the particular frequency,

the digital signals having a pre-cursor response and a post-cursor response,

a feed forward equalizer for producing signals inhibiting the pre-cursor response,

a decision feedback equalizer for producing signals inhibiting the post-cursor response,

an adder for combining the signals from the feed forward equalizer and the decision feedback equalizer to provide resultant signals,

a high pass filter for producing signals limiting the time duration of the post-cursor response before the introduction of the digital signals to the feed forward equalizer,

a tail canceller responsive to the digital signal from the feed forward equalizer for producing signals further limiting the time duration of the post-cursor response in the digital signals,

a quantizer responsive to the resultant signals from the adder for providing digital signals representing values of +1, 0 and −1 in accordance with the amplitudes of the resultant signals from the first adder,

the adder constituting a first adder,

a second adder responsive to the output signals from the decision feedback equalizer and the tail canceller for producing signals for introduction to the first adder for combination with the signals from the feed forward equalizer to produce the resultant signals,

means for introducing the signals from the quantizer to the decision feedback equalizer for the production by the decision feedback equalizer of the digital signals inhibiting the post-cursor response,

means initially responsive to the digital signals from the second means before the introduction of such digital signals to the high pass filter for providing a coarse adjustment in the phase of these digital signals to have the zero crossings of these digital signals coincide in phase with the zero crossings of the clock signals, and

means subsequently responsive to the digital signals from the quantizer for providing a fine adjustment in the phase of these signals to have the zero crossings of these digital signals coincide in phase with the zero crossings of the clock signals.

49. In combination for use in a system providing first signals having individual ones of a plurality of analog levels to represent information,

a repeater,

a plurality of clients,

pairs of unshielded twisted wires, each pair being disposed between the repeater and an individual one of the clients to transmit the first signals between the repeater and the individual one of the clients,

first means in each of the clients for receiving the first signals from the repeater,

second means for providing clock signals at a particular frequency,

third means for converting the first signals to digital signals,

the digital signals having a pre-cursor response and a post-cursor response,

a feed forward equalizer at each of the clients for providing an output inhibiting the pre-cursor response in the digital signals from the third means at such client,

a decision feedback equalizer at each of the clients for providing an output inhibiting the post-cursor response in the digital signals from the third means at such client,

a tail canceller at each of the clients for providing an output limiting the duration of the post-cursor response of the digital signals at such client,

a first adder at each of the clients for combining the outputs from the feed forward equalizer and the tail canceller to obtain first resultant signals at such client,

a second adder at each of the clients for combining the output from the feed forward equalizer and the first

resultant signals from the first adder to produce second resultant signals indicative of the information represented by the first signals,

a quantizer responsive at each of the clients to the second resultant signals from the second adder at such client for selecting for each of the second resultant signals a magnitude indicative of an individual one of a plurality of amplitudes closest in magnitude to each of the second resultant signals,

means responsive to the magnitudes selected by the quantizer for each of the digital signals for introducing such magnitudes as the digital signals to the decision feedback equalizer, and

means responsive at each of the clients to the magnitude of the second resultant signals from the second adder for adjusting the phase of the second resultant signals to have the zero crossings of the second resultant signals coincide in phase with the zero crossings of the clock signals at such client.

50. In combination for use in a system providing first signals having individual ones of a plurality of analog levels to represent information,

a repeater,

a plurality of clients,

pairs of unshielded twisted wires, each pair being disposed between the repeater and an individual one of the clients to transmit the first signals between the repeater and the individual one of the clients,

first means in each of the clients for receiving the first signals from the repeater,

second means for providing clock signals at a particular frequency,

third means for converting the first signals to digital signals,

the digital signals having a pre-cursor response and a post-cursor response,

a feed forward equalizer at each of the clients for providing an output inhibiting the pre-cursor response in the digital signals from the third means at such client,

a decision feedback equalizer at each of the clients for providing an output inhibiting the post-cursor response in the digital signals from the third means at such client,

a tail canceller at each of the clients for providing an output limiting the duration of the post-cursor response of the digital signals at such client,

a first adder at each of the clients for combining the outputs from the feed forward equalizer and the tail canceller to obtain first resultant signals at such client,

a second adder at each of the clients for combining the output from the feed forward equalizer and the first resultant signals from the first adder to produce second resultant signals indicative if the information represented by the first signals,

a quantizer responsive at each of the clients to the second resultant signals from the second adder at such client for selecting for each of the second resultant signals magnitude indicative of an individual one of a plurality of amplitudes closest in magnitude to each of the second resultant signals,

means responsive to the magnitudes selected by the quantizer for each of the digital signals for introducing such magnitudes as the digital signals to the decision feedback equalizer,

a high pass filter at each of the clients for receiving the digital signals from the third means at such client and for passing the high frequency components of such digital signals to the feed forward equalizer as inputs to the feed forward equalizer at such client,

means initially responsive at each of the clients to the digital signals from the third means before the introduction of these signals to the high pass filter at such client for providing a coarse adjustment in the phase of these signals to have the zero crossings of these signals coincide in phase with the zero crossings of the clock signals at such client and,

means subsequently responsive at each of the clients to the digital signals from the quantizer at such client for providing a fine adjustment in the phase of these signals to have the zero crossings of these signals coincide in phase with the zero crossings of the clock signals at such client.

51. In a combination as set forth in claim **50**,

there being unshielded twisted pairs of wires between the repeater and each of the clients, and

means at each of the clients for offsetting the phase of the digital signals from the quantizer at such client, before the fine adjustment in phase of these digital signals at such client, in accordance with the characteristics of the unshielded twisted pairs of wires between the repeater and such client to have the zero crossings of these signals coincide in phase with the zero crossings of the clock signals at such client.

* * * * *