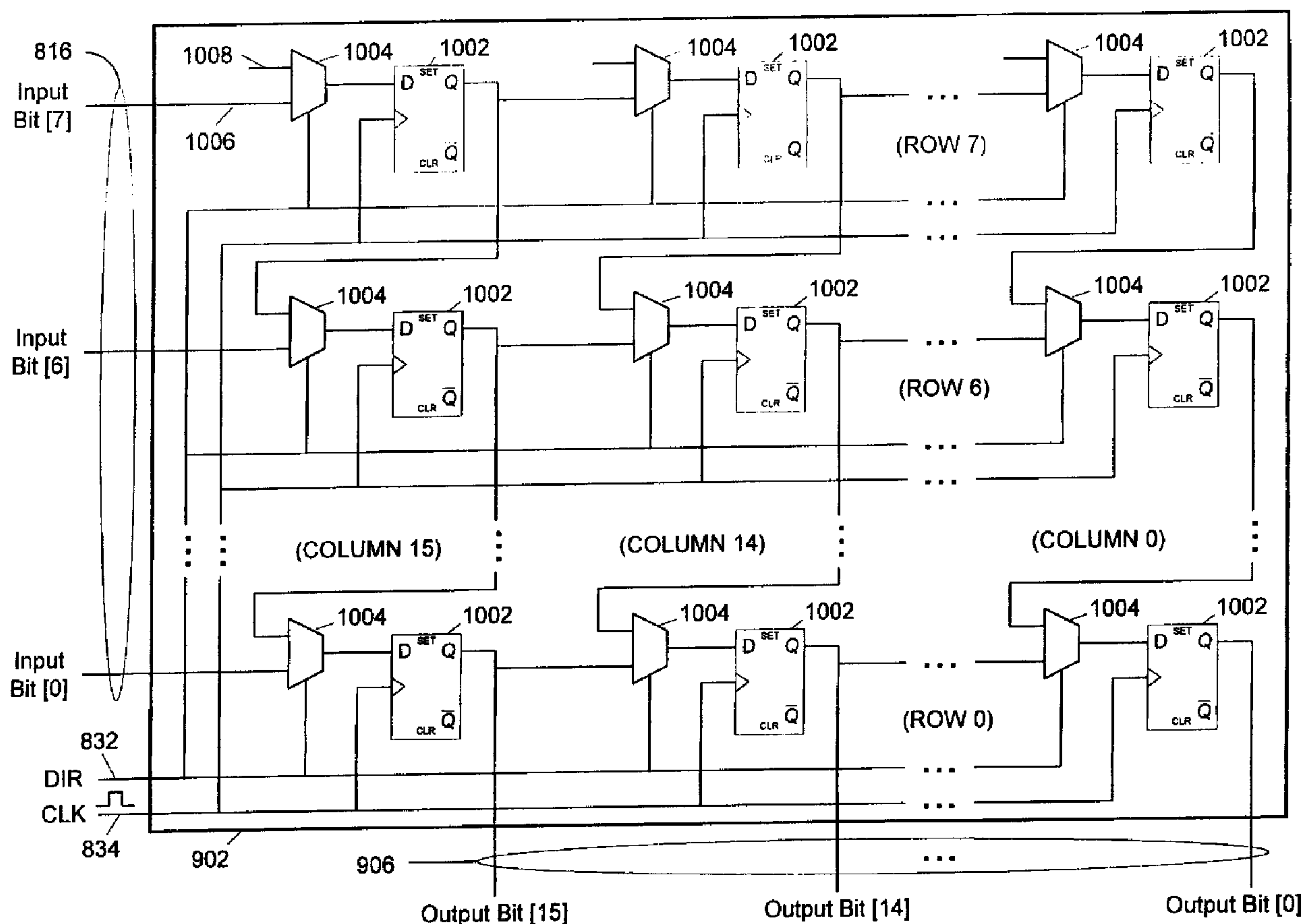




(86) Date de dépôt PCT/PCT Filing Date: 1998/11/13
(87) Date publication PCT/PCT Publication Date: 1999/05/27
(45) Date de délivrance/Issue Date: 2008/05/20
(85) Entrée phase nationale/National Entry: 2000/05/12
(86) N° demande PCT/PCT Application No.: US 1998/024215
(87) N° publication PCT/PCT Publication No.: 1999/026225
(30) Priorité/Priority: 1997/11/14 (US08/970,307)

(51) Cl.Int./Int.Cl. *G09G 3/36* (2006.01)
(72) Inventeurs/Inventors:
WEATHERFORD, WILLIAM THOMAS, US;
WORLEY, W. SPENCER III, US;
CHOW, WING HONG, US
(73) Propriétaire/Owner:
AURORA SYSTEMS, INC., US
(74) Agent: GOWLING LAFLEUR HENDERSON LLP

(54) Titre : SYSTEME ET PROCEDE DE PLANARISATION DE DONNEES
(54) Title: SYSTEM AND METHOD FOR DATA PLANARIZATION



(57) Abrégé/Abstract:

A system and method for planarizing n-bit display data into m-bit data. The system includes a plurality of input terminals, a plurality of output terminals, and a first storage bank having a bit-depth less than m. In one embodiment, the first storage bank is a bi-



(57) **Abrégé(suite)/Abstract(continued):**

directional shift register including a clock input terminal, for receiving a data shift signal, and a direction terminal for receiving a direction control signal. In a particular embodiment, the bi-directional shift register includes a plurality of flip-flops and a plurality of multiplexers, both arranged in a rectangular array of columns and rows. Optionally, the system includes a second storage bank, and data is shifted into one storage bank while data is being shifted out of the other storage bank.

ABSTRACT OF THE DISCLOSURE

A system and method for planarizing n-bit display data into m-bit data. The system includes a plurality of input terminals, a plurality of output terminals, and a first storage bank
5 having a bit-depth less than m. In one embodiment, the first storage bank is a bi-directional shift register including a clock input terminal, for receiving a data shift signal, and a direction terminal for receiving a direction control signal. In a particular embodiment, the bi-directional shift register includes a plurality of flip-flops and a plurality of multiplexers, both arranged in a rectangular array of columns and rows. Optionally, the system includes a second storage bank,
10 and data is shifted into one storage bank while data is being shifted out of the other storage bank.

SPECIFICATION
SYSTEM AND METHOD FOR DATA PLANARIZATION

5

10

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates generally to electronic display drivers, and more particularly to an improved system and method for display data planarization.

Description of the Background Art

15 FIG. 1 shows a single pixel cell 100 of a typical liquid crystal display. Pixel cell 100 includes a liquid crystal layer 102, contained between a transparent common electrode 104 and pixel storage electrode 106, a storage element 108, and a switching transistor 110. Storage element 108 is coupled at node 112 to pixel storage electrode 106 and, via switching transistor 110, to a data input line 114. Storage element 108 is also coupled, as is common electrode 104
20 to a common voltage supply terminal 116 (e.g., ground). Responsive to a select signal on select line 118, which is coupled to the control terminal of switching transistor 110, storage element 108 reads a data signal in from data line 114, stores the signal, and asserts the signal on node 112, even after the select signal is no longer present.

25 Liquid crystal layer 102 rotates the polarization of light passing through it, the degree of rotation depending on the root-mean-square (RMS) voltage across liquid crystal layer 102. The ability to rotate the polarization is exploited to modulate the intensity of reflected light as follows. An incident light beam 120 is polarized by polarizer 122. The polarized beam then passes through liquid crystal layer 102, is reflected off of pixel electrode 106, and passes again through liquid crystal layer 102. During this double pass through liquid crystal layer 102, the
30 beam's polarization is rotated by an amount which depends on the data signal being asserted on pixel storage electrode 106. The beam then passes through polarizer 124, which passes only that

WO 99/26225

PCT/US98/24215

portion of the beam having a specified polarity. Thus, the intensity of the reflected beam passing through polarizer 124 depends on the amount of polarization rotation induced in liquid crystal layer 102, which in turn depends on the data signal being asserted on pixel storage electrode 106.

5 Storage element 108 can be either an analog storage element (e.g. capacitive) or a digital storage element (e.g., SRAM latch). In the case of a digital storage element, a common way to drive pixel storage electrode 106 is via pulse-width-modulation (PWM). In PWM, different gray scale levels are represented by multi-bit words (i.e., binary numbers). The multi-bit words are converted to a series of pulses, whose time-averaged root-mean-square (RMS)
10 voltage corresponds to the analog voltage necessary to attain the desired gray scale value.

For example, in a 4-bit PWM scheme, the frame time (time in which a gray scale value is written to every pixel) is divided into 15 time intervals. During each interval, a signal (high, e.g., 5V or low, e.g., 0V) is asserted on the pixel storage electrode 106. There are, therefore, 16 (0-15) different gray scale values possible, depending on the number of "high" pulses asserted
15 during the frame time. The assertion of 0 high pulses corresponds to a gray scale value of 0 (RMS 0V), whereas the assertion of 15 high pulses corresponds to a gray scale value of 16 (RMS 5V). Intermediate numbers of high pulses correspond to intermediate gray scale levels.

A particular signal being applied during a time interval is referred to as a "state". For example, a high signal being asserted during one time interval is an "on" state. Similarly, a low
20 signal being asserted during one time interval is referred to as an "off" state.

FIG. 2 shows a series of pulses corresponding to the 4-bit gray scale value (1010), where the most significant bit is the far left bit. The pulses are grouped to correspond to the bits of the binary gray scale value. Specifically, the first group B3 includes 8 intervals (23), and corresponds to the most significant bit of the value (1010). Similarly, group B2 includes 4
25 intervals (22) corresponding to the next most significant bit, group B1 includes 2 intervals (21) corresponding to the next most significant bit, and group B0 includes 1 interval (20) corresponding to the least significant bit. This grouping reduces the number of pulses required from 15 to 4, one for each bit of the binary gray scale value, with the width of each pulse corresponding to the significance of its associated bit. Thus, for the value (1010), the first pulse
30 B3 (8 intervals wide) is high, the second pulse B2 (4 intervals wide) is low, the third pulse B1 (2 intervals wide) is high, and the last pulse B0 (1 interval wide) is low. This series of pulses

WO 99/26225

PCT/US98/24215

results in an RMS voltage that is approximately $\sqrt{\frac{2}{3}}$ (10 of 15 intervals) of the full value (5V), or approximately 4.1V.

The resolution of the gray scale can be improved by adding additional bits to the binary gray scale value. For example, if 8 bits are used, the frame time is divided into 255 intervals, providing 256 possible gray scale values. In general, for (n) bits, the frame time is divided into (2ⁿ - 1) intervals, yielding (2ⁿ) possible gray scale values.

Because the liquid crystal cells are susceptible to deterioration due to ionic migration resulting from a DC voltage being applied across them, the above described PWM scheme is modified as shown in FIG. 3. The frame time is divided in half. During the first half, the PWM data is asserted on the pixel storage electrode, while the common electrode is held low. During the second half of the frame time, the complement of the PWM data is asserted on the pixel storage electrode, while the common electrode is held high. This results in a net DC component of 0V, avoiding deterioration of the liquid crystal cell, without changing the RMS voltage across the cell, as is well known to those skilled in the art.

FIG. 4 shows a response curve of an electrically controlled, birefringent liquid crystal cell. The vertical axis 402 indicates the percent of full brightness (i.e., maximum light reflection) of the cell, and the horizontal axis 404 indicates the RMS voltage across the cell. As shown, the minimum brightness (a dark pixel) is achieved at an RMS voltage V_{tt}. For some wavelengths of light, an RMS voltage less than V_{tt} results in a pixel that is not completely dark, as shown in FIG.4. For other wavelengths, all RMS voltages less than V_{tt} result in a dark pixel. In the portion of the curve between V_{tt} and V_{sat}, the percent brightness increases as the RMS voltage increases, until 100% full brightness is reached at V_{sat}. Once the RMS voltage exceeds V_{sat}, however, the percent brightness decreases as the RMS voltage increases.

FIG. 5 shows an RMS voltage versus gray scale value curve, for an 8-bit (256 gray scale values) gray scale system. The RMS voltage for each gray scale value ("Gray-Value") is given by the following formula, where V_{on} is the digital "on" value, typically V_{dd}:

$$V_{rms} = \sqrt{(1/255)(\text{GrayValue})(V_{on})^2} \text{ (Eq. 1)}$$

Gray scale value (x) corresponds to an RMS voltage equal to V_{tt} and, referring back to FIG. 4, to 0% brightness. Thus, the gray scale values less than value (x) are unusable, because

for some wavelengths of light, they result in a brighter rather than a darker pixel, and for other wavelengths, the values result in 0% brightness and are, therefore, redundant. Similarly, value (y) corresponds to an RMS voltage equal to V_{sat} and, referring back to FIG. 4, to 100% full brightness. Thus, the gray scale values greater than value (y) are also unusable, because they
 5 result in a darker rather than a brighter pixel. The result of these wasted values is that true 8-bit gray scale resolution is not obtained.

In order to avoid gray scale distortions, all gray scale values must be confined to the useful portion of the liquid crystal response curve (FIG. 4) between V_{tt} and V_{sat} . One way to accomplish this is to add an additional bit to the gray scale code (e.g., use a 9-bit gray scale
 10 system) and then map the 8-bit values to the values of the 9-bit system corresponding to the useful portion of the response curve. The addition of a single bit, however, increases the bandwidth requirements of the data interface by 100%, and is, therefore, undesirable. What is needed is a system and method for confining all of the available gray scale states to the useful portion of the liquid crystal response curve.

15 Furthermore, because a display data stream is generally received pixel by pixel (one multi-bit gray scale value at a time) and is written to a display one bit (for every pixel) at a time, the incoming data must be reformatted (planarized). One method of reformatting the data is described in U.S. Patent No. 5,255,100, issued to Urbanus on October 19, 1993.

Urbanus describes a system whereby an entire frame worth of
 20 data is stored in a random access memory (RAM), and then transferred to a frame memory one bit at a time. The entire frame worth of memory must be output from the RAM to the frame buffer before additional data can be received by the RAM. The system of Urbanus is, therefore, memory intensive and slow. What is needed is a fast, simple, and memory efficient system and method for planarizing a display data stream.

25 SUMMARY

A novel data planarizer is described. The data planarizer receives data words having a first number of bits (n-bit data) and outputs data words having a second number of bits (m-bit data). The planarizer includes a plurality of data input terminals for receiving the n-bit data words and a plurality of data output terminals for outputting the m-bit data words. The
 30 planarizer further includes a first storage bank having an input terminal set coupled to the plurality of data input terminals, an output terminal set coupled to the plurality of data output terminals, and a bit-depth less than the bit-width (m) of the reformatted data.

WO 99/26225

PCT/US98/24215

In one embodiment, the storage bank is a bi-directional shift register, including a clock input terminal, for receiving a data shift signal, and a direction terminal, for receiving a direction control signal. In a particular embodiment, the bi-directional shift register includes a plurality of flip-flops and a plurality of multiplexers, each arranged in a rectangular array of columns and rows. Each of the flip-flops has a control terminal coupled to the clock terminal of the bi-directional shift register, an input terminal, and an output terminal. Each multiplexer has a first input terminal, a second input terminal, a control terminal coupled to the direction terminal of the bi-directional shift register, and an output terminal coupled to the input terminal of an associated one of the flip-flops. Each multiplexer in a first column of the array has its first input terminal coupled to an associated one of the data input terminals, and each flip-flop in a bottom row has its output terminal coupled to an associated one of the data output terminals. Additionally, each multiplexer of the other columns has its first input terminal coupled to the output terminal of an associated one of the flip-flops located in an adjacent column, and each multiplexer in the rows other than a top row has its second input terminal coupled to the output terminal of an associated one of the flip-flops located in an adjacent row.

In a particular embodiment, the data planarizer includes a first storage bank and a second storage bank. In a more particular embodiment, the combined capacity of the first and the second storage banks is less than one frame of data.

A novel method for planarizing n -bit data into m -bit planarized data is also described. The method includes the steps of receiving a first group of (p) n -bit data words (where $p < m$), storing the first group of (p) data words in a first storage bank, and transferring the first group of (p) data words into a device having (n) different memory locations, by transferring a p -bit data word including one bit from each of the first group of (p) n -bit data words into a first portion of each of the (n) different memory locations. The method further includes the steps of receiving a second group of (p) n -bit data words, storing the second group of (p) data words in the first storage bank, and transferring the second group of (p) data words into the memory device by transferring a p -bit data word including one bit from each of the second group of (p) n -bit data words into a second portion of each of the (n) different memory locations. Optionally, the second group of (p) n -bit data words are stored in and transferred from a second storage bank.

In a particular method, the step of storing a group of (p) data words in a storage bank is performed by shifting the (p) data words into the storage bank along a first direction, (n) bits at a time, and the step of transferring the group of data words into the device comprising (n) different

16527-0014PCT

memory locations is performed by shifting the (p) data words out of the storage bank along a second direction, (p) bits at a time.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 shows a single pixel cell of a typical liquid crystal display;

5 FIG. 2 shows one frame of 4-bit pulse-width modulation data;

FIG. 3 shows a split frame of 4-bit pulse-width modulation data;

FIG. 4 shows a brightness versus RMS voltage curve for a typical liquid crystal cell;

FIG. 5 shows a gray scale value versus RMS voltage curve;

10 FIG. 6 shows a split frame of pulse-width modulation data including additional forced on and forced off states, in accordance with the present invention;

FIG. 7 shows a gray scale value versus RMS voltage curve, as modified by the addition of forced on and forced off states in accordance with the present invention;

FIG. 8a shows detail of an alternate embodiment of a display driver in accordance with the current invention, including a selective inverter.

FIG. 8b shows detail of an alternate embodiment of a display driver in accordance with the current invention, including a selective inverter.

FIG. 9 illustrates a data path through a data planarizer and a frame buffer in accordance with the present invention;

FIG. 10 shows a bi-directional shift register of the data planarizer shown in FIG.9;

FIG. 11 shows a forced state controller of the driver shown in FIG. 8; and

FIG. 12 shows a timing diagram for data and certain control signals communicated between the driver and the uLCD shown in FIG. 8.

16527-0014PCT

DETAILED DESCRIPTION

This patent application is related to the following US Patents associated with the corresponding U. S. patent applications, filed on even date herewith and assigned to a common assignee:

De-Centered Lens Group For Use In An Off-Axis Projector, U. S. Patent No. 6,076,931, Matthew F. Bone and Donald Griffin. Koch;

System And Method For Reducing Peak Current And Bandwidth Requirements In A Display Driver Circuit, U. S. Patent No. 6,288,712, Raymond Pinkham, W. Spencer Worley, III, Edwin Lyle Hudson, and John Gray Campbell;

System And Method For Using Forced State To Improve Gray Scale Performance Of A Display, U. S. Patent No. 6,072,452, W. Spencer Worley, III and Raymond Pinkham.

This patent application is also related to U. S. Patent No. 6,518,945, entitled Replacing Defective Circuit Elements By Column And Row Shifting In A Flat Panel Display, by Raymond Pinkham, assigned to a common assignee.

The present invention overcomes the problems associated with the prior art, by confining the gray scale values to the useful portion of the liquid crystal response curve, without the use of additional data bits. Specifically, the present invention describes a system and method for planarizing n-bit data words, corresponding to a particular pixel, into m-bit, planarized data words, corresponding to a group of m pixels. In the following description, numerous specific details are set forth (e.g., the bit-width of the gray scale code and the bit width of the planarized data) in order to provide a thorough understanding of the invention. Those skilled in the art will recognize, however, that the invention may be practiced apart from these specific details. In other instances, details of well known display circuits and driving methods have been omitted, so as not to unnecessarily obscure the present invention.

As shown in FIG. 6, additional forced "on" states 602 and additional forced "off" states 604 are added to the PWM data 606 during the first half 608 of the frame time 610, to produce a modified display data stream. The extra "on" states 602 and "off" states 604 correspond to extra

WO 99/26225

PCT/US98/24215

time intervals in frame time 610, which are added in one of two different ways. In the first way, the time intervals of the PWM data 606 are shortened to make room for the additional states, without increasing the frame time. Alternatively, the frame time may be increased to make room for the new states, without changing the length of the individual time intervals of the PWM data.

5 During the first half of the frame time, the common electrode of a driven liquid crystal cell (not shown) is held low (0V). In the second half 612 of frame time 610, however, the common electrode is held at 5V, and the PWM data as well as the added "on" and "off" states are inverted. This inversion is necessary to maintain a net DC voltage of 0 V across the driven cell, and thus avoid degradation of the cell.

10 Note that in FIG. 6, the forced on states 602 and forced off states 604 are added to the display data 606 contiguously (i.e., sequentially in a single block of time intervals). This advantageously reduces the number of times the signal to the display must be switched. Those skilled in the art will recognize, however, that the forced states may be intermingled amongst themselves, or with the display data, as long as the resultant RMS voltage is not altered.

15 FIG. 7 shows the effect of the added forced states on the gray scale value versus RMS voltage curve. Taking the added forced states into account, the modified RMS value is given by:

$$V_{rms} = \sqrt{\frac{(\text{ForcedOn} + \text{GrayValue})(V_{on})^2}{(255 + \text{ForcedOn} + \text{ForcedOff})}} \quad (\text{Eq. 2})$$

20 The added "on" states create a minimum RMS voltage floor 702, and the added "off" states create a maximum RMS voltage ceiling 704. RMS voltage floor 702 is raised and lowered by increasing or decreasing the number of additional "on" states, respectively. Similarly, RMS voltage ceiling 704 is lowered or raised by increasing or decreasing the number of additional "off" states, respectively. The number of added "on" and "off" states is selected such that RMS
25 voltage floor 702 is equal to V_{tt} , the RMS voltage corresponding to 0% brightness (FIG. 4), and RMS voltage ceiling 704 is equal to V_{sat} , the RMS voltage corresponding to 100% brightness.

The optimum number of extra "on" and "off" states may be determined mathematically, using the following variables:

n = the number of bits of gray scale resolution;

30 G = the gray scale value;

V_{on} = the digital "on" voltage;

WO 99/26225

PCT/US98/24215

Voff = the digital "off" voltage;

Vtt = the liquid crystal minimum brightness RMS voltage;

Vsat = the liquid crystal saturation RMS voltage;

Fon = the optimum number of fixed "on" states; and

5 Foff = the optimum number of fixed "off" states.

In general, the modified RMS voltage ("Vrms") is given by:

$$V_{rms} = \sqrt{\frac{(G + F_{on}) (V_{on})^2}{(2^n - 1) + F_{on} + F_{off}}} \quad (\text{Eq. 3})$$

It is desirable that $V_{rms} = V_{tt}$, at $G = 0$. Substituting these values into Eq. 3 yields:

10

$$V_{tt} = \sqrt{\frac{F_{on}(V_{on})^2}{(2^n - 1) + F_{on} + F_{off}}} \quad (\text{Eq. 4})$$

Similarly, it is desirable that $V_{rms} = V_{sat}$, at $G = 2^n - 1$. Substituting these values into Eq. 3 yields:

15

$$V_{tt} = \sqrt{\frac{F_{on}(V_{on})^2}{(2^n - 1) + F_{on} + F_{off}}} \quad (\text{Eq. 5})$$

Solving Eq. 4 for Foff yields:

20

$$F_{off} = \frac{F_{on}(V_{on})^2}{(V_{tt})^2} - F_{on} - (2^n - 1) \quad (\text{Eq. 6})$$

Substituting Eq. 6 into Eq. 5, and solving for Fon yields:

$$F_{on} = \frac{(2^n - 1) (V_{tt})^2}{(V_{sat})^2 - (V_{tt})^2} \quad (\text{Eq. 7})$$

25

Substituting Eq. 7 into Eq. 6, and solving for Foff yields:

WO 99/26225

PCT/US98/24215

$$F_{off} = (2^n - 1) \left[\frac{(V_{on})^2 - (V_{tt})^2}{(V_{sat})^2 - (V_{tt})^2} - 1 \right] \quad (\text{Eq. 8})$$

Finally, for given values of n , V_{tt} , V_{sat} , and V_{on} , the optimum numbers of added “on” and “off” states are calculated from Eq. 7 and Eq. 8, respectively.

As indicated above, the required numbers of forced “on” and forced “off” states can be determined empirically, by iterating F_{on} and F_{off} until a gray value of 0 results in an RMS value near V_{tt} , and a gray value of $2n-1$ results in an RMS value near V_{sat} . In one particular system, where $n=8$, $V_{tt}=1.5V$, $V_{sat}=3.0V$, and $V_{dd}=3.3V$, the iterating process yielded the following results:

10	normal PWM states:	255
	fixed “on” states:	86
	fixed “off” states:	82
	total states:	423
	% increase in states	66%

Note that this number may differ from the calculated optimum number of forced states, and still provide an adequately modified LCD data stream.

Clearly, the overhead for the addition of extra states (66%) is significantly less than the overhead required to adding a single bit to the gray scale code (100%), and extremely less than adding two additional bits (200%). Furthermore, because the voltages of the forced “on” and “off” states are known, the forced states can be written to the pixel storage electrode apart from receiving the LCD gray scale data stream, thus reducing the bandwidth requirements for the system interface.

FIG. 8 shows an LCD driver 800 in accordance with the present invention. Driver 800 includes input controller 802, control selector 804, data planarizer 806, frame buffer A 808, frame buffer B 810, phase locked loop 812, and forced state generator 814. Driver 800 receives an 8-bit gray scale display data stream via data input bus 816, and receives horizontal synchronization (Hsync), vertical synchronization (Vsync), and pixel dot clock signals via input terminals 818, 820, and 822, respectively. After inserting forced states into the display data stream to improve gray scale performance, driver 800 transfers the modified display data, via 32-bit data output bus 824, along with control signals, via LCD control bus 826, to a micro-LCD

WO 99/26225

PCT/US98/24215

828, which includes an array of liquid crystal pixel cells, similar to the pixel cell shown in FIG.1.

Input controller 802 uses the Hsync and Vsync signals to coordinate the transfer of data from data input bus 816 into data planarizer 806 and the transfer of data from data planarizer 806, via 32-bit data bus 830 into frame buffers A 808 and B 810. Responsive to the Vsync and Hsync signals indicating valid data on data input bus 816, input controller 802 asserts signals on control lines DIR 832 and CLK 834, causing data to be clocked into and out of data planarizer 806, as will be more fully described in conjunction with FIG. 10 below.

Data planarizer 806 receives the gray scale data, via data input bus 816, in 8-bit data words, each 8-bits (Pm[0-7]) corresponding to a gray scale value to be written to a particular pixel (m) of micro-LCD 828. Data planarizer 806 accumulates the 8-bit gray scale data for 32 pixels and reformats the data into 32-bit data words, each 32-bit word containing one bit from each of the group of 32 8-bit gray scale data words. For example, the 32-bit word formed by bits P0[0] - P31[0] includes the least significant bits of the gray scale values of pixels 0-31. This reformatting is necessary because each bit of gray scale data is written to micro-LCD 828 32 pixels at a time. The reformatting of data by data planarizer 806 is discussed in greater detail in conjunction with FIG. 9 below.

Frame buffer A 808 and frame buffer B 810 are each 32-bit wide synchronous graphics random access memories (SGRAMs). Each of frame buffers 808 and 810 receives data, via 32-bit data bus 830, and stores the data in a memory location associated with a particular bit significance and a particular group of pixels of micro-LCD 828. Further, each of frame buffers 808 and 810 are of sufficient capacity to store 8 bits of gray scale data for each pixel in micro-LCD 828 (i.e., one frame worth of display data). For example, because micro-LCD 828 has 786,432 pixels (1024 X 768), frame buffers 808 and 810 each store 6,291,456 bits (one display screen worth) of data, or 196,608 32-bit words.

The transfer of data from data bus 830 into frame buffers 808 and 810 is also controlled by input controller 802 in cooperation with control selector 804. Input controller 802 asserts frame buffer control signals on input control bus 836 and a frame buffer select signal on select (SEL) line 838. Input control bus 836 includes a write enable line and address lines for indicating the memory location into which data is to be written.

Control selector 804 includes a first multiplexer 840 and a second multiplexer 842. First multiplexer 840 has two sets of input terminals, the first set being coupled to the lines of input

WO 99/26225

PCT/US98/24215

control bus 836. Second multiplexer 842 also has two sets of input terminals, the second set being coupled to the lines of input control bus 836. The output of first multiplexer 840 is asserted on frame buffer A control bus 844, and the output of second multiplexer 842 is asserted on frame buffer B control bus 846.

5

First multiplexer 840 and second multiplexer 842 are both controlled by the SEL signal being asserted on select line 838 by input controller 802. Responsive to a first (e.g. high) SEL signal being asserted on select line 838, first multiplexer 840 couples input control bus 836 with frame buffer A control bus 844, thus allowing input controller 802 to load data from data bus 830 into frame buffer A 808. The first SEL signal also causes second multiplexer 842 to decouple input control bus 836 from frame buffer B control bus 846, so that no data is loaded into frame buffer B 810 while frame buffer A is being loaded. Responsive to a second (e.g., low) SEL signal being asserted on select line 838, first multiplexer decouples input control bus 836 from frame buffer A control bus 844 and couples input control bus 836 with frame buffer B control bus 846, thus allowing input controller 802 to load data from data bus 830 into frame buffer B 810. Input controller 802 toggles the SEL signal each time a Vsync signal is received, such that one display screen worth of data is written into each frame buffer 808 and 810 in alternating order.

Forced state generator 814 controls the output of data from frame buffer A 808 and frame buffer B 810, receives the display data via data bus 848, selectively inserts forced states into the display data stream, and outputs the modified display data stream via data output bus 824 to micro-LCD 828. Forced state generator 814 includes a forced state controller 850 and a multiplexer 852.

Multiplexer 852 receives data from data bus 848, from 32-bit "force-on" bus 854, and from 32-bit "force-off" bus 856. Each line of force-on bus 854 is maintained at a voltage (V_{on}) corresponding to an "on" state, and each line of force-off bus 856 is maintained at a voltage (V_{off}) corresponding to an "off" state. In one embodiment, the sources of the forced state data asserted on force-on bus 854 and force-off bus 856 are system voltage reference terminals (e.g., V_{dd} and Ground). Those skilled in the art will understand, however, that alternate sources of forced state data, for example registers, may be employed. Responsive to control signals received via 2-bit control bus 858, multiplexer 852 selectively couples either data bus 848, force-on bus 854, or force-off bus 856 to data output bus 824. As used herein, the term "multiplexer"

30

WO 99/26225

PCT/US98/24215

is understood to include all selective coupling devices, including, but not limited to, shared bus structures.

Forced state controller 850 receives the Vsync signal via line 860, and receives a clock input signal via line 862 from phase-locked loop 812. Phase-locked loop 812 is well known in the art, and serves to synchronize the pixel dot clock with an internal machine clock (not shown).

Forced state controller 850 controls the output of data from frame buffer A 808 and frame buffer B 810 by asserting control signals on an output control bus 864, which is coupled to the second set of input terminals of first multiplexer 840 and to the first set of input terminals of second multiplexer 842. Thus, when the second SEL signal is asserted on select line 838 by input controller 802, first multiplexer 840 decouples input control bus 836 from and couples output control bus 864 to frame buffer A control bus 844, thus allowing forced state controller 850 to cause frame buffer A 808 to assert data onto data bus 848. On the other hand, when the first SEL signal is asserted on select line 838, second multiplexer 842 decouples input control bus 836 from and couples output control bus 864 to frame buffer B control bus 846, allowing forced state controller 850 to cause frame buffer B 810 to assert data onto data bus 848. Thus, while pixel data for one frame is being loaded into frame buffer A 808 by input controller 802, pixel data for the previous frame is being outputted from frame buffer B 810 by forced state controller 850, and vice versa.

Forced state generator 814 inserts forced states (as shown in FIG. 6) into the display data stream as follows. First, a Vsync signal on line 860 indicates the start of a frame. Forced state controller 850 asserts a first control signal on 2-bit control bus 858, causing multiplexer 852 to couple force-on bus 854 to data output bus 824, thus asserting forced "on" states on data output bus 824. Then, forced state controller 850 asserts control signals on LCD control bus 826 causing forced on states to be loaded from data output bus 824 into micro-LCD 828. After the desired number of forced on states are loaded into micro-LCD 828, forced state controller 850 asserts a second control signal on 2-bit control bus 858, causing multiplexer 852 to couple data bus 848 to data output bus 824. Then, forced state controller 850 asserts frame buffer control signals on output control bus 864 and LCD control signals on LCD control bus 826, causing data to be transferred out of either frame buffer A 808 or frame buffer B 810 (depending on the current SEL signal), through multiplexer 852, and into micro-LCD 828. Data continues to be transferred from frame buffer A 808 or B 810 until an entire frame of display data has been transferred. Then, forced state controller 850 asserts a third control signal on 2-bit data bus 858,

WO 99/26225

PCT/US98/24215

causing multiplexer 852 to couple force-off bus 856 to data output bus 824, and asserts LCD control signals on LCD control bus 826 causing the desired number of forced off states to be transferred into micro-LCD 828, thus completing the first half 608 of frame 610 (FIG. 6).

The second half 612 of frame 610 (FIG. 6) is written to micro-LCD 828 substantially as described above, except that the forced on states, the data, the forced off states, and the common electrode are inverted. The inversion occurs within micro-LCD 828 under the control of forced state controller 850 as follows. LCD control bus 826 includes address lines, op code lines for communicating instructions (e.g., read, write, etc.), a data invert line, a common electrode signal line, and a clock signal line. At the beginning of the second half 612 of frame 610, forced state controller 850 switches the signal being asserted on the common electrode signal line from low to high, and asserts a control signal on the invert line causing micro-LCD 828 to invert all incoming data. The data and the forced states are then transferred out of driver 800 under the control of forced state controller 850 as described above.

Alternatively, the inversion of the forced on states, the data, and the forced off states may be implemented within driver 800. For example, FIG. 8a shows an alternate embodiment of controller 800, including a selective inverter 870 interposed between data bus 848 and data input terminals 872 of multiplexer 852, to selectively invert the data stream. The invert line 874 is redirected from LCD control bus 826 to selective inverter 870. Responsive to a first signal on invert line 874, selective inverter 870 asserts the data received via data bus 848 onto data input terminals 872 of multiplexer 852. Responsive to an invert signal on invert line 874, selective inverter 870 inverts the data received via data bus 848, and asserts the inverted data onto data input terminals 872 of multiplexer 852. In this embodiment, forced state controller 850 controls the inversion of the forced on and forced off states, by controlling multiplexer 852 as described above, but reversing temporally, the couplings of bus 854 and bus 856 to bus 824. FIG. 8b shows another alternate embodiment of driver 800, wherein selective inverter 870 is coupled to receive and selectively invert the modified data stream, which includes the forced states.

FIG. 9 shows an example of data flow through data planarizer 806 and into frame buffer A 808. Data planarizer 806 includes a first bi-directional shift register 902 and a second bi-directional shift register 904, each serving as a temporary storage bank. Each register 902 and 904 is 16 bits deep (16 columns) and 8 bits wide (8 rows). The bit depth corresponds to the number of incoming data words each register has the capacity to store, and the bit-width

WO 99/26225

PCT/US98/24215

corresponds to the number of bits in each incoming data word, which in this particular embodiment is the number of bits per pixel.

Data enters shift registers 902 and 904 via 8-bit data input bus 816, as previously described, and is organized within registers 902 and 904 as follows. First, the 8 bits of gray scale data for pixel 0 (P0[0-7]) enter, via data input bus 816, and are stored in the right most column of register 902. The next 8-bits P1[0-7] enter and are stored in the column to the left of bits P0[0-7]. The data continues to be loaded in this fashion until the bits P15[0-7] are loaded in the left most column of register 902. The 8-bit gray scale data for pixels P16 - P31 is then loaded into register 904 in like fashion, such that bits P16[0-7] are loaded in the right most column and bits P31[0-7] are loaded in the left most column.

Thus loaded, each row of register 902 and register 904 contains a 16-bit word, including one bit of similar significance from 16 sequential pixels. For example, the bottom row of register 904 includes the least significant bits from the gray scale data for pixels P16-P31. Furthermore, like numbered rows of registers 902 and 904 combine to form 32-bit words, each including one bit of similar significance from 32 sequential pixels. For example, the top rows of registers 902 and 904 include the most significant bits from the gray scale data for pixels P0-P31. These 32-bit words are written, via data bus 830, into frame buffers A 808 and B 810, to be stored for subsequent transfer to micro-LCD 828.

Frame buffers A 808 and B 810 are able to read one-half of data bus 830 at a time. Additionally, an inverter 907, having an input terminal coupled to direction control line 832, provides register 902 with an inverted direction control signal. This allows planarizer 806 to write the contents of register 902 to frame buffer A 808 or frame buffer B 810 while register 904 is being loaded. For example, as data is being clocked into register 904 via data input bus 816, data is being clocked out of register 902 via a first half 906 of data bus 830. Similarly, when data is being clocked into register 902 via data input bus 816, data is being clocked out of register 904 via a second half 908 of data bus 830.

Accordingly, a frame worth of data is formatted by data planarizer 806 and stored in either frame buffer A 808 or B 810 as follows. Each memory location in frame buffers A 808 and B 810 is divided into a first half 910 and a second half 912. The gray scale data for pixels P0-P15 is clocked into register 902, as described above. Then, as the next block of data for pixels P16-P31 is clocked into register 904, the data for pixels P0-P15 is clocked into first half 910 of a first memory block 914 (each block contains 8 32-bit memory locations). Next, as the

WO 99/26225

PCT/US98/24215

data for pixels P32-P47 is clocked into register 902, the data for pixels P16-P31 is clocked into second half 912 of first memory block 914. Then, as the data for pixels P48-P63 is clocked into register 904, the data for pixels P32-P47 is clocked into first half 910 of a second memory block 916. This sequence continues until the data for the last pixels (P786,416-P786,431) is clocked
 5 from register 904 into second half 912 of a last memory block 918.

Because registers 902 and 904 are able to planarize the data for groups of less than 32 pixels, data planarizer 806 need not receive an entire frame worth of data before the data can be output in its planarized form. Additionally, registers 902 and 904 are smaller and less expensive than the RAM employed in the prior art. Furthermore, those skilled in the art will recognize that
 10 additional bi-directional shift registers may be employed. For example, four registers, each 8-bits deep, could be used to write 8 bits to each of four different portions of each memory location, thus writing a 32-bit word to each memory location.

Fig. 10 shows first bi-directional shift register 902 in greater detail. Second bi-directional shift register 904 is substantially identical. Register 902 includes 128 D-type flip-flops 1002 arranged in a rectangular array of 16 columns (0-15) and 8 rows (0-7), and an associated array of multiplexers 1004 also arranged in a rectangular array of 16 columns and 8 rows. To facilitate clear explanation, the notation (r,c) refers to the row and column location of a given device. For example, multiplexer 1004(6,14) refers to the multiplexer 1004 located in row 6 and column 14. The rows and columns are labeled in FIG. 10 to correspond to the bit numbers
 15 of data input bus 816 and first half 906 of data bus 830, respectively.
 20

All flip-flops 1002(r,c) receive a clock signal from input controller 802 (FIG. 8) via CLK line 834, and all the multiplexers 1004(r,c) receive control input from input controller 802 via DIR line 832. The input (D) of each flip-flop 1002(r,c) is coupled to the output of an associated multiplexer 1004(r,c) located in the same row and column.

Each multiplexer 1004(r,c) has a first input terminal 1006 and a second input terminal 1008, which are selectively coupled to the input (D) of associated flip-flop 1002(r,c), depending on the control signal being asserted on DIR line 832. First input terminals 1006 of multiplexers 1004(r,15) (column 15) are coupled to associated bit lines of data input bus 816. In the remaining columns (c<15), first input terminals 1006 of multiplexers 1004(r,c) are coupled to
 25 the non-inverting output (Q) of associated flip-flops 1002(r,c+1) (left neighbors). Second input terminals 1008 of multiplexers 1004(7,c) (row 7) are not used. In the remaining rows (r<7), second input terminals 1008 of multiplexers 1004(r,c) are coupled to the non-inverting output
 30

WO 99/26225

PCT/US98/24215

(Q) of associated flip-flops 1002(r+1,c) (upper neighbors). Finally, the non-inverting (Q) output of flip-flops 1002(0,c) (row 0) are coupled to corresponding bit lines of first half 906 of data bus 830.

Bi-directional shift register 902 operates as follows. When input controller 802 asserts a first signal on DIR control line 832, all multiplexers 1004(r,c) couple their first input terminals 1006 with the inputs (D) of flip-flops 1002(r,c). Then, when the first clock signal is received via CLK line 834, flip-flops 1002(r,15) latch the 8-bit data word present on data input bus 816 onto their non-inverting (Q) outputs. When the next clock signal is received, the first 8-bit word stored by flip-flops 1002(r,15) is shifted to the non-inverting outputs (Q) of flip-flops 1002(r,14), and flip-flops 1002(r,15) latch the next 8-bit data word present on data input bus 816 onto their non-inverting (Q) outputs. Upon receiving each subsequent clock signal, new data is received and the previously received data is shifted to the right. This continues until register 902 has loaded 16 8-bit words (one on each column of flip-flops).

Input controller 802 shifts data out of register 902 by asserting a second signal on DIR line 832. Responsive to the second signal on DIR line 832, each multiplexer couples its second input terminal with its output terminal, thus changing the shift direction. Until the next clock signal is received, flip-flops 1002(0,c) are asserting bit 0 of each of the 16 stored 8-bit words on bit lines 0-15 of bus 906. When the next clock signal is received, the bit stored on the non-inverting outputs (Q) of each flip-flop 1002(r,c) is latched onto the non-inverting output of flip-flop 1002(r-1,c) (lower neighbor), thus asserting bit 1 of each of the 16 stored 8-bit words on bit lines 0-15 of bus 906. This process continues as each clock signal is received until all 8 bits (0-7) of each of the 16 stored words have been sequentially asserted on bus 906. After register 904 is loaded (it takes longer to load than to unload registers 902 and 904, i.e., 16 cycles versus 8 cycles), input controller 802 reasserts the first signal on DIR line 832 so that register 902 can be reloaded.

FIG. 11 shows forced state controller 850, in greater detail, to include a memory 1102, a processing unit 1104, a prescale 1106, and a transfer state machine 1108. Memory 1102 is a program storage device, which stores data and commands for access and execution by processing unit 1104. Prescale 1106 receives the dot clock signal via line 862, generates a lower frequency timing signal (e.g., 1/2 the frequency of the dot clock), and communicates the timing signal, via line 1110 to processing unit 1104. The lower frequency timing signal enables processing unit 1104 to employ smaller scale components, for example, smaller counters.

WO 99/26225

PCT/US98/24215

Processing unit 1104 controls transfer state machine 1108 via a transfer request line 1112, transfer select bus 1114, force-on line 1116, and force-off line 1118. Responsive to the signals received from processing unit 1104, transfer state machine 1108 asserts control signals on LCD control bus 826 (FIG. 8), 2-bit control bus 858, and output control bus 864, as follows.

5 Responsive to a signal on transfer request line 1112, transfer state machine 1108 asserts a control signal on 2-bit control bus 858 causing multiplexer 852 to couple data output bus 824, via data bus 848, to frame buffers A 808 and B 810. Transfer select line 1114 is a multi-bit line used to communicate the address of the memory block to be transferred out of frame buffer A 808 or frame buffer B 810. Transfer state machine 1108 uses the block address to initialize the
10 memory address asserted on output control bus 864, and then sequentially increments the memory address while asserting a write signal on LCD control bus 826.

 Responsive to processing unit 1104 asserting a signal on force-on line 1116, transfer state machine 1108 asserts a signal on 2-bit control bus 858, causing multiplexer 852 to couple force-on bus 854 with data output bus 824. Then, transfer state machine 1108 asserts a write
15 signal on LCD control bus 826, thus transferring forced on states into micro-LCD 828. Similarly, responsive to processing unit 1104 asserting a signal on force-off line 1118, transfer state machine 1108 asserts a signal on 2-bit control bus 858, causing data selector 852 to couple force-off bus 856 with data output bus 824, and asserts a write signal on LCD control bus 826, to transfer forced off states into micro-LCD 828.

20 In one embodiment, forced state controller 850 is implemented with a programmable logic device part number EPF10K50 BC356-3, manufactured by Altera Corporation of Santa Clara, California. The verilog code for programming this device is provided at the end of this disclosure.

 FIG. 12 shows a timing diagram 1200 detailing the relationship, during one frame time
25 1201, between the Vsync signal 1202, the common micro-LCD electrode signal 1204, the data invert signal 1206, the pixel data 1208, the first pixel value 1210, the last pixel value 1212, the magnitude and direction of the voltage drop across the first pixel 1214, and the magnitude and direction of the voltage drop across the last pixel 1216. Timing diagram 1200 is useful to illustrate practical considerations which must be taken into account when implementing driver
30 800.

 Recall from the discussion of FIG. 3 that the net dc voltage over time across each liquid crystal cell must be zero, in order to avoid damage to the cell caused by ionic migration.

WO 99/26225

PCT/US98/24215

Therefore, the common signal 1204 and the pixel data 1208 are inverted during a first half 1218 of frame 1201 and are not inverted during a second half 1220 of frame 1201. It does not matter whether the common signal 1204 and the data 1206 are inverted during the first 1218 or second 1220 half of frame 1201, as long as the net dc voltage across each cell is zero.

5 Practical limitations arise because, while the single common electrode signal may be switched very rapidly at the mid point 1222 of frame 1201, it takes some finite amount of time (X) to write the first bit of LCD data to each of the pixels in micro-LCD display 828, as shown by the angled edges of micro-LCD data curve 1208. It is, therefore, desirable to switch the common electrode signal at a time when a forced on or a forced off state is being asserted on
10 every pixel. This results in a very rapid, simultaneous switch of the signals on all of the pixels. Specifically, all forced on states become forced off states, and all forced off states become forced on states, as soon as the common electrode is switched.

The time (X) that it takes to write data to all the pixels also effects the writing of forced states. Comparing first pixel value curve 1210 to last pixel value curve 1212, it is apparent that
15 there is a time delay (X) between writing the first forced on state to the first pixel at 1224 and writing the first forced on state to the last pixel at 1226. In general, this delay is offset, because of the time delay (X) between writing data to the first pixel at 1228 and writing data to the last pixel at 1230. There must be, however, an adequate number of forced states to accommodate the offset. In particular, it is sufficient if the minimum forced-on time and the minimum forced-off
20 time equals 2X, where X is the time required to write to each pixel once, as described above.

Certain timing relationships are also necessary to maintain equal brightness across the display and to maintain a debiased (net dc voltage = 0) condition. The following timing relationships are sufficient, and refer to the voltage drops labeled in voltage drop magnitude/direction curves 1214 and 1216. First, in order to debias the first pixel, voltage drops
25 A and B must be equal in magnitude and opposite in direction to voltage drop C, while the sum of the time intervals that voltage drops A and B are applied must equal the time interval voltage drop C is applied. Similarly, voltage drops D and E must be equal in magnitude and opposite in direction to voltage drop F, while the sum of the time intervals that voltage drops D and E are applied must equal the time interval voltage drop F is applied. The pixel data (Pixel Data) is, by
30 definition, equal in magnitude and opposite in direction to the complementary pixel data (!Pixel Data). Next, in order to maintain equal brightness (RMS voltage offset), the sum of voltage drops A and B must be equal in magnitude and direction to the sum of voltage drops D and E,

WO 99/26225

PCT/US98/24215

and voltage drop C must be equal in magnitude and direction to voltage drop F. Finally, the first half 1218 of frame 1201 should be equal in time duration to the second half 1220 of frame 1201. Those skilled in the art will understand, that the sequence of data and forced states, the division of the frame, and the switching of the common electrode may be altered without deviating from the scope of the invention, however the fundamental debias and RMS voltage offset conditions must be maintained. The above described timing diagram 1200 illustrates only one of many possible ways of accomplishing this.

The description of particular embodiments of the present invention is now complete. Many of the described features may be substituted, altered or omitted without departing from the scope of the invention. For example, the invention may be used in a multi-color system by using a separate driver and display for each color, or by time multiplexing a single driver and display for more than one color. Additionally, the invention may employed with a wide variety of pulse modulation schemes including, but not limited to, pulse-amplitude modulation, pulse-width modulation, pulse-position modulation, and pulse-code modulation. Further, data planarizer 806 may include additional, smaller bi-directional shift registers to enable data reformatting in even smaller increments.

VERILOG CODE

```

20 / fileindata.v
    // target device Altera 10Kx0
    // Copyright 1997 sVISION Inc.

    // Revision history
25 // 6/10/97 in progress

    module format (reset_l, clk, pClk, red, green, blue, hSync, vSync, dEn,
        vAddr, vData, ras_l, cas_l, dsf, we_l, sc, dtoe, se_l, overlay, qsf,
        redOut, greenOut, blueOut, vSyncOut, valid, uAddr, uData, uWr_l, uRd_l,
30 memCS_l, ioCS_l, ale, ready);

```

WO 99/26225

PCT/US98/24215

// Module I/O Definitions

```

input      clk;    // 40 MHz clock from CPU

5  // Inputs from gmZ1
input      reset_l;
input      pClk;    //pixel clock
input [7:0] red;    //red data from gmZ1
input [7:0] green;  //green data from gmZ1
10 input [7:0] blue; //blue data from gmZ1
input      hSync;   //horizontal Sync from gmZ1
input      vSync;   //vertical sync from gmZ1
input      dEn;     //data enable from gmZ1

15 //VRAM signals

output [8:0] vAddr;  //VRAM address
inout [15:0] vData;  // RAM data
output      ras_l;   //RAS
20 output [1:0] cas_l; //CAS (one for each bank)
output      dsf;     //DSF
output [1:0] dtoe;   //DT/OE
output [1:0] we_l;   //Write Enables (High and low bytes)
output [1:0] sc;     //serial clock (one for each bank)
25 output [1:0] se_l; //serial port output enables
input [15:0] overlay; //serial data (used for overlay planes)
input [1:0] qsf;     //serial ports status

//microProcessor signals
30
inout [7:0] uData;   //uP data and address buss
input      uWr_l;    //uP write strobe

```

WO 99/26225

PCT/US98/24215

```

input      uRd_l;      //uP read strobe
input [7:0] uAddr;      //uP Address lines [15:8]
input      ioCS_l;     //cs for IO cycle
input      memCS_l;    //cs for memory (VRAM) access
5  input    ale;        //address latch strobe
output     ready;      //cycle end for uP

```

//pixel outputs

```

10 output [7:0] redOut;    //red pixel output
output [7:0] greenOut;    //green pixel output
output [7:0] blueOut;     //blue pixel output
output      vSyncOut;
output      valid;

```

15

//Register definitions

```

reg [7:0] redAddr;
20 reg [7:0] greenAddr;
reg [7:0] blueAddr;

```

```

wire [7:0] redLUT;
wire [7:0] blueLUT;
25 wire [7:0] greenLUT;

```

```

reg [7:0] redLUTout;
reg [7:0] blueLUTout;
reg [7:0] greenLUTout;

```

30

```

reg [7:0] redOut;
reg [7:0] greenOut;

```

WO 99/26225

PCT/US98/24215

```
reg [7:0]    blueOut;

reg [7:0]    redIn;
reg [7:0]    greenIn;
5  reg [7:0]  blueIn;

reg [7:0]    redD1;
reg [7:0]    greenD1;
reg [7:0]    blueD1;
10 reg [7:0]  redD0;
reg [7:0]    greenD0;
reg [7:0]    blueD0;
reg [7:0]    redD01;
reg [7:0]    redD02;
15 reg [7:0]  redD03;
reg [7:0]    redD04;
reg [7:0]    redD05;
reg [7:0]    redD06;
reg [7:0]    greenD01;
20 reg [7:0]  greenD02;
reg [7:0]    greenD03;
reg [7:0]    greenD04;
reg [7:0]    greenD05;
reg [7:0]    greenD06;
25 reg [7:0]  blueD01;
reg [7:0]    blueD02;
reg [7:0]    blueD03;
reg [7:0]    blueD04;
reg [7:0]    blueD05;
30 reg [7:0]  blueD06;

reg [7:0]    redOverlay;
```

WO 99/26225

PCT/US98/24215

```

    reg [7:0]    greenOverlay;
    reg [7:0]    blueOverlay;

    reg          vSyncIn;
5   reg          vSyncInD1;
    reg          hSyncIn;
    reg          hSyncInD1;
    reg          vSyncOut;
    reg          vSync1;
10  reg          hSync1;
    reg          dataValid;

    reg          qsfD1;
    reg          qsfD2;
15  reg          qsfD3;
    reg          sClk;

    reg          invert;
    reg          useOverlay;
20  reg          enOverlay;
    reg          wrRedLUT;
    reg          wrGreenLUT;
    reg          wrBlueLUT;

25  reg [15:0]   overlayIn;
    reg [7:0]    olData;
    reg          wrOverlay;

    reg [7:0]    addrReg;
30  //SW
    wire [7:0]   uDataIn = uData;
    reg [7:0]    uDataOut;

```

WO 99/26225

PCT/US98/24215

```

reg      weRed;
reg      weGreen;
reg      weBlue;

5  reg [1:0]  control;
   reg [1:0]  wrMode;
   reg [4:0]  overlayCtl;
   reg [3:0]  outputCtl;
   reg [3:0]  memPage;
10
   reg [6:0]  blankLine;
   reg [3:0]  outputCntl;
   reg [7:0]  testRed;
   reg [7:0]  testGreen;
15  reg [7:0]  testBlue;

   reg [7:0]  readReg;
   reg [7:0]  readLUT;
   reg [7:0]  readOvly;
20
   reg [11:0] hCount;
   reg [10:0] vCount;
   reg      hSyncInt;
   reg      vSyncInt;
25  reg      hValid;
   reg      vValid;
   reg      hSyncReal;
   reg      vSyncReal, validReal;
   reg      qsfMask;
30  //These are all reged in the vram model  //reg memDone;
   //reg cycleType;                        // 0 is uP access 1 is Xfer access
   reg [8:0]  xferAddr;

```

WO 99/26225

PCT/US98/24215

```

reg      readyOut;
wire     endCycle;

reg      wrIOReq, rdIOReq, wrMemReq, rdMemReq;
5  reg    ioReq, ioReqD1, ioReqD2, ioReqD3, ioReqX1, ioReqX2, ioReqX3;
reg      memReq, memReqX1;
reg      redLUTReq, greenLUTReq, blueLUTReq;
reg      redOvlyReq, greenOvlyReq, blueOvlyReq;
reg      readReqLUT, readReqOvly;
10 reg [7:0] redDx;
reg [7:0] greenDx;
reg [7:0] blueDx;

reg      colorDone, colorReq;
15 reg    vSyncMemD1, vSyncMemD2, vSyncMemD3;
reg      sClkL, sClkH, useLowVram, sClkD1, sClkD2, sClkInt;
reg [7:0] olDataD1;
reg [7:0] olDataD2;
reg [7:0] olDataD3;
20 reg [7:0] olRedAddr, olGreenAddr, olBlueAddr;
reg      useOverlayD1, useOverlayD2, useOverlayD3;
reg      splitReq, fullReq;
reg      weOverRed, weOverGreen, weOverBlue;
reg [1:0] sc;
25

reg      uPcolorReq, uPcbrReq, cbrReqX1, cbrReq, colorReqX1;
reg      blockReq, blockReqX1, uPblockReq;
reg [6:0] validDelay;
reg      olInv;
30 reg      valid, blankData;
reg      armStartBlank, waitForLast, armLastBlank, goBlank, endBlank;
reg      freezeD1, freezeD2, freezeOut;

```

WO 99/26225

PCT/US98/24215

```

//wires SW
wire      vramRdReq;      //The pipelined request and uRd_1
wire      vramWrReq;      //The pipelined request and uWr_1
5  wire [7:0] memRdData;   //output from vramsm

//sw
wire [7:0] overRed;
wire [7:0] overGreen;
10 wire [7:0] overBlue;

//Real Code Start

define CHIP_ID 0
15

//SW 7-22
assign uData = (!uRd_1 && (!ioCS_1 || !memCS_1)) ? uDataOut : 8'bz;
assign ready = (!readyOut) ? 1'b0 : 1'bz;

20 //
// uP interface
//

always @(negedge ale) begin
25 addrReg[7:0] <= #4 uData;
end // always

always @(posedge uWr_1 or negedge reset_1) begin
if (!reset_1) begin
30 control <= 2'b0;
wrMode <= 2'b11;
overlayCtl <= 5'b0;

```

WO 99/26225

PCT/US98/24215

```

outputCtl <= 4'b1110;
blankLine <= 7'b0;
end // if
else if (!ioCS_1) begin
5  //SW
   if (uAddr[2:0] == 3'b111) begin
       case (addrReg[3:0])
           4'h1: control <= #4 uData[1:0];
           4'h2: memPage <= #4 uData[3:0];
10  4'h3: wrMode <= #4 uData[1:0];
           4'h4: blankLine <= #4 uData[6:0];
           4'h5: overlayCtl <= #4 uData[4:0];
           4'h6: outputCtl <= #4 uData[3:0];
           4'h7: testRed <= #4 uData;
15  4'h8: testGreen <= #4 uData;
           4'h9: testBlue <= #4 uData;
       endcase
       end // if
       end // else
20  end // always

//Ready low for overlay VRAM and LUT

always @(negedge uRd_1 or negedge uWr_1 or posedge endCycle) begin
25  if (endCycle)
       readyOut <= 1'b1;
       else
       if (!memCS_1 || (!ioCS_1 && (uAddr[2:0] != 3'b111)))
       readyOut <= 1'b0;
30  end // always

//End Cycle

```

WO 99/26225

PCT/US98/24215

//SW

assign endCycle = clrMemReq || clrIOReq;

//Generate requests

5

always @ (negedge uWr_1 or posedge clrIOReq) begin

if (clrIOReq) begin

wrIOReq <= #4 1'b0;

end // if

10 else begin

if (!ioCS_1)

wrIOReq <= (uAddr[2:0] != 3'b111);

end// else

end // always

15

always @ (negedge uRd_1 or posedge clrIOReq) begin

if (clrIOReq) begin

rdIOReq <= #4 1'b0;

end // if

20 else begin

if (!ioCS_1)

rdIOReq <= (uAddr[2:0] != 3'b000 && uAddr[2:0] !=
3'b111);

end// else

25 end // always

//generate VRAM req

always @ (posedge clrMemReq or negedge uWr_1) begin

30 if (clrMemReq)

wrMemReq <= #4 1'b0;

//SW added else

WO 99/26225

PCT/US98/24215

```

else if (!memCS_l)
  wrMemReq <= #4 1'b1;
end // always

5  always @(posedge clrMemReq or negedge uRd_l) begin
    if (clrMemReq)
      rdMemReq <= #4 1'b0;
      //SW added else
    else if (!memCS_l)
10  rdMemReq <= #4 1'b1;
    end // always

    //uPcolor
    always @(posedge clrMemReq or negedge uWr_l) begin
15  if (clrMemReq)
      uPcolorReq <= #4 1'b0;
    else if (!ioCS_l && (uAddr[2:0] == 3'b111) && (addrReg[3:0]
      == 4'ha))
      uPcolorReq <= #4 1'b1;
20  end // always

    //up CBR
    always @(posedge clrMemReq or negedge uWr_l) begin
      if (clrMemReq)
25  uPcbrReq <= #4 1'b0;
    else if (!ioCS_l && (uAddr[2:0] == 3'b111) && (addrReg[3:0]
      == 4'hb))
      uPcbrReq <= #4 1'b1;
    end // always
30

    //up Block Req
    always @(posedge clrMemReq or negedge uWr_l) begin

```

WO 99/26225

PCT/US98/24215

```

if (clrMemReq)
  uPblockReq <= #4 1'b0;
else if (!ioCS_1 && (uAddr[2:0] == 3'b111) && (addrReg[3:0]
  == 4'hc))
5  uPblockReq <= #4 1'b1;
  end // always

  //assign name to io register

10  wire sourceExt = control[0];
    //wire clrVram = control[1]; // control 1 will be unused
    wire [2:0] overlaySel = overlayCtl[2:0];
    wire testData = overlayCtl[3];
    wire invData = overlayCtl[4];
15  wire freeze = outputCtl[0];
    wire enRed = outputCtl[1];
    wire enBlue = outputCtl[2];
    wire enGreen = outputCtl[3];

20

    //synchronize freeze to pClk and vSync

    always @ (posedge pClk) begin
      freezeD1 <= #4 freeze;
25  freezeD2 <= #4 freezeD1;
    end // always

    always @ (posedge pClk) begin
      if (vSyncReal)
30  freezeOut <= #4 freezeD2;
    end // always

```

WO 99/26225

PCT/US98/24215

//read data

```

always @ (addrReg or control or memPage or wrMode or blankLine or
overlayCtl or outputCtl or testRed or testGreen or testBlue or
5  uPcolorReq or uPcbrReq or uPblockReq) begin
case (addrReg[3:0])
4'h0: readReg <= #4 `CHIP_ID;
4'h1: readReg <= #4 {6'b0, control};
4'h2: readReg <= #4 {4'b0, memPage};
10 4'h3: readReg <= #4 {6'b0, wrMode};
4'h4: readReg <= #4 {1'b0, blankLine};
4'h5: readReg <= #4 {3'b0, overlayCtl};
4'h6: readReg <= #4 {4'b0, outputCtl};
4'h7: readReg <= #4 testRed;
15 4'h8: readReg <= #4 testGreen;
4'h9: readReg <= #4 testBlue;
//SW
4'ha: readReg <= #4 {7'b0, uPcolorReq};
4'hb: readReg <= #4 {7'b0, uPcbrReq};
20 4'hc: readReg <= #4 {7'b0, uPblockReq};
endcase
end // always

```

```

always @ (uAddr or readLUT or readOvly or readReg or memCS_1 or
25 memRdData) begin
if (!memCS_1)
uDataOut <= #4 memRdData;
else
case (uAddr[2:0]) //synopsys parallel_case full_case
30 3'b000: uDataOut <= #4 8'b0;
3'b001: uDataOut <= #4 readLUT;
3'b010: uDataOut <= #4 readLUT;

```

WO 99/26225

PCT/US98/24215

```

3'b011:  uDataOut <= #4 readLUT;
3'b100:  uDataOut <= #4 readOvly;
3'b101:  uDataOut <= #4 readOvly;
3'b110:  uDataOut <= #4 readOvly;
5  3'b111:  uDataOut <= #4 readReg;
    endcase
    end //always

10  //synchronize request to the proper clock
    //VRAM

    always @(posedge clk or posedge clrMemReq) begin
    if (clrMemReq) begin
15  memReqX1 <= #4 1'b0;
    memReq <= #4 1'b0;
    cbrReqX1 <= #4 1'b0;
    cbrReq <= #4 1'b0;
    colorReqX1 <= #4 1'b0;
20  colorReq <= #4 1'b0;
    blockReq <= #4 1'b0;
    blockReqX1 <= #4 1'b0;
    end // if
    else begin
25  memReqX1 <= #4 wrMemReq || rdMemReq;
    memReq <= #4 memReqX1;
    cbrReqX1 <= #4 uPcbrReq;
    cbrReq <= #4 cbrReqX1;
    colorReqX1 <= #4 uPcolorReq;
30  colorReq <= #4 colorReqX1;
    blockReq <= #4 blockReqX1;
    blockReqX1 <= #4 uPblockReq;

```

WO 99/26225

PCT/US98/24215

```

end // else
end // always

// LUT and OVERLAY
5  always @(posedge pClk) begin
    ioReqX1 <= #4 wrIOReq || rdIOReq;
    ioReqX2 <= #4 ioReqX1;
    ioReqX3 <= #4 ioReqX2;
    end // always

10

    // rising edges of reqs will generate access strobes and clear req the
    requests

    always @(posedge pClk) begin
15  ioReq <= #4 !ioReqX3 && ioReqX2 ;
    redLUTReq <= #4 ioReq && (uAddr == 3'h1);
    greenLUTReq <= #4 ioReq && (uAddr == 3'h2);
    blueLUTReq <= #4 ioReq && (uAddr == 3'h3);
    redOvlyReq <= #4 ioReq && (uAddr == 3'h4);
20  greenOvlyReq <= #4 ioReq && (uAddr == 3'h5);
    blueOvlyReq <= #4 ioReq && (uAddr == 3'h6);
    ioReqD1 <= #4 ioReq;           // delay strobe to capture read data
    ioReqD2 <= #4 ioReqD1;        // delay equivalent to we strobe
    ioReqD3 <= #4 ioReqD2;        // is this one needed
25  readReqLUT <= #4 ioReqD3 && !uAddr[2];
    readReqOvly <= #4 ioReqD2 && uAddr[2];
    end // always

    wire clrIOReq = !reset_1 || (!uRd_1 && (readReqLUT || (ioReqD2 &&
30  uAddr[2])))
    || (!uWr_1 && ioReqD2) ;
    wire clrMemReq = !reset_1 || (memDone && !cycleType);

```

WO 99/26225

PCT/US98/24215

```
//Internal timing generation
//This will generate 70 Hz refresh with a 58.057Mhz pixel clock

5
//create controls for adding blank lines

wire doBlank = blankLine != 7'b0;

10 //Note: this is probably clunky and could be made more efficient...

always @(posedge pClk) begin
  if (!doBlank)
    armStartBlank <= 1'b0;
15 else begin
  if (vSync1)
    armStartBlank <= #4 1'b1;
    if (startBlank)
      armStartBlank <= #4 1'b0;
20 end // else
end // always

wire startBlank = armStartBlank && hSync1 || waitForLast && hSync1;

25 always @(posedge pClk) begin
  if (!vSync1 || !doBlank)
    armLastBlank <= #4 1'b0;
    if (dataValid && doBlank)
      armLastBlank <= #4 1'b1;
30 if (startBlank)
    armLastBlank <= #4 1'b0;
end // always
```

WO 99/26225

PCT/US98/24215

```

always @ (posedge pClk) begin
  if (!armLastBlank)
    waitForLast <= #4 1'b0;
5   if (armLastBlank && hSync1)
    waitForLast <= #4 1'b1;
    if (dataValid)
      waitForLast <= #4 1'b0;
    end // always
10
always @ (posedge pClk) begin
  if (!doBlank)
    goBlank <= #4 1'b0;
    if (startBlank)
15   goBlank <= #4 1'b1;
    if (endBlank)
      goBlank <= #4 1'b0;
    end // always

20  //horizontal counter  NOTE: == compares probably won't meet timing
always @ (posedge pClk or negedge reset_l) begin
  if (!reset_l)
    hCount <= #4 11'b0;
    else if (doBlank && !goBlank)
25   hCount <= #4 11'b0;
    if (!sourceExt || goBlank) begin
      if (hCount == 11'h403)
        hCount <= #4 11'b0;
      else hCount <= hCount + 1'b1;
30   end // else if
    end // always

```

WO 99/26225

PCT/US98/24215

//Vertical Counter

```

always @ (posedge pClk or negedge reset_l) begin
  if (!reset_l)
5   vCount <= 10'h000;
  else if (doBlank && !goBlank)
    vCount <= 10'h000;
  if ((!sourceExt || doBlank) & hCount == 11'h403) begin
    if (vCount <= 10'h325)
10   vCount <= 10'h000;
    else vCount = vCount + 1'b1;
  end // else
end // always

15  always @ (posedge pClk)
    endBlank <= #4 vCount[6:0] == blankLine && hCount ==
        11'h403;

//vSync Generation
20
always @ (posedge pClk) begin
  hSyncInt <= #4 hCount == 11'h403;
  vSyncInt <= #4 hSyncInt && vCount == 10'h325;
end // always

25
//valid generation

always @ (posedge pClk or negedge reset_l) begin
  if (!reset_l)
30  vValid <= 1'b0;
  else begin
    if (hSyncInt && vCount == 11'h002)

```

WO 99/26225

PCT/US98/24215

```

vValid <= 1'b1;
if (hSyncInt && vCount == 11'h301)
vValid <= #4 1'b0;
end // else
5  end // always

always @ (posedge pClk or negedge reset_l) begin
if (!reset_l)
hValid <= #4 1'b0;
10  else begin
if (hCount == 11'h002)
hValid <= #4 1'b1;
if (hCount <= 11'h401)
hValid <= #4 1'b0;
15  end // else
end // always

wire validInt = hValid && (goBlank || vValid);

20  //note that this is asynchronous. It is very long and will work...
wire vSyncMem = (!sourceExt && vCount == 10'h325) || (sourceExt
&& vSyncIn);

always @ (posedge clk) begin
25  vSyncMemD1 <= #4 vSyncMem;
vSyncMemD2 <= #4 vSyncMemD1;
vSyncMemD3 <= #4 vSyncMemD2;
end // always

30  wire vSyncMemRise = !vSyncMemD3 && vSyncMemD2;

//Change syncs to 1 clock pulses

```

WO 99/26225

PCT/US98/24215

```

always @ (posedge pClk) begin
vSyncIn <= #4 vSync;
vSyncInD1 <= #4 vSyncIn;
5  hSyncIn <= #4 hSync;
hSyncInD1 <= #4 hSyncIn;
vSync1 <= #4 !vSyncIn && vSyncInD1;
hSync1 <= #4 hSyncIn && !hSyncInD1;
vSyncOut <= #4 vSyncReal;
10 end // always

always @ (posedge pClk) begin
if (!sourceExt || goBlank) begin
hSyncReal <= #4 hSyncInt;
15 vSyncReal <= #4 vSyncInt;
validReal <= #4 validInt;
end // begin
else begin
hSyncReal <= #4 hSync1;
20 vSyncReal <= #4 vSync1;
validReal <= #4 dataValid;
end // else
end // always

25 //pipeline Valid

always @ (posedge pClk) begin
validDelay <= #4 {validDelay[5:0], validReal};
valid <= #4 validDelay[6] && !freezeOut;
30 end // always

//Register Input RGB Data (use I/O Cells)

```

WO 99/26225

PCT/US98/24215

```

always @(posedge pClk) begin
    redIn <= #4 red;
    greenIn <= #4 green;
5    blueIn <= #4 blue;
    dataValid <= #4 dEn;
    end // always

    //Mux in test data registers
10
    always @(posedge pClk) begin
        if (!sourceExt || testData || blankData) begin
            redD0 <= #4 testRed;
            blueD0 <= #4 testBlue;
15    greenD0 <= #4 testGreen;
        end // if
        else begin
            redD0 <= #4 redIn;
            greenD0 <= #4 greenIn;
20    blueD0 <= #4 blueIn;
        end // else
        end // always

    //Add pipeline stages to sync up with overlay data
25
    always @(posedge pClk) begin
        redD01 <= #4 redD0;
        redD02 <= #4 redD01;
        redD03 <= #4 redD02;
30    redD04 <= #4 redD03;
        redD05 <= #4 redD04;
        redD06 <= #4 redD05;

```

WO 99/26225

PCT/US98/24215

```

greenD01 <= #4 greenD0;
greenD02 <= #4 greenD01;
greenD03 <= #4 greenD02;
greenD04 <= #4 greenD03;
5  greenD05 <= #4 greenD04;
   greenD06 <= #4 greenD05;
   blueD01 <= #4 blueD0;
   blueD02 <= #4 blueD01;
   blueD03 <= #4 blueD02;
10  blueD04 <= #4 blueD03;
   blueD05 <= #4 blueD04;
   blueD06 <= #4 blueD05;
   end // always

15  //Mux in Negated data

   always @(posedge pClk) begin
   if (invert || olInv) begin
   redD1 <= #4 ! redD06;
20  greenD1 <= #4 ! greenD06;
   blueD1 <= #4 ! blueD06;
   end // if
   else begin
   redD1 <= #4 redD06;
25  greenD1 <= #4 greenD06;
   blueD1 <= #4 blueD06;
   end // else
   end // always

30  //mux in uP address for read/writes to LUT

   always @(posedge pClk) begin

```

WO 99/26225

PCT/US98/24215

```

if (!redLUTReq )
redAddr <= #4 redD1;
else
redAddr <= #4 addrReg[7:0];
5  end // always

always @ (posedge pClk) begin
if (!greenLUTReq)
greenAddr <= #4 greenD1;
10 else
greenAddr <= #4 addrReg[7:0];
end // always

always @ (posedge pClk) begin
15 if (!blueLUTReq)
blueAddr <= #4 blueD1;
else
blueAddr <= #4 addrReg[7:0];
end // always
20

//generate write strobes for LUTs

always @ (posedge pClk) begin
weRed <= redLUTReq && !uWr_l;
25 weGreen <= greenLUTReq && !uWr_l;
weBlue <= #4 blueLUTReq && !uWr_l;
end // always

//Data for lookup table, register @
30 //LUT

syn_ram_256x8_iror u1 (redLUT, uDataIn, weRed, redAddr, pClk,pClk);

```

WO 99/26225

PCT/US98/24215

```
syn_ram_256x8_iror u2 (greenLUT, uDataIn, weGreen, greenAddr,
pClk,pClk);
```

```
syn_ram_256x8_iror u3 (blueLUT, uDataIn, weBlue, blueAddr, pClk,pClk);
```

```
5 //Register LUT outputs (Maximize speed by no muxing here)
```

```
always @ (posedge pClk) begin
```

```
redLUTout <= #4 redLUT;
```

```
greenLUTout <= #4 greenLUT;
```

```
10 blueLUTout <= #4 blueLUT;
```

```
end//always
```

```
//get read data for uP at this stage (may need to decode some earier)
```

```
15 always @ (posedge pClk) begin
```

```
if (readReqLUT) begin
```

```
if (uAddr[1:0] == 2'b01)
```

```
readLUT <= #4 redLUTout;
```

```
if (uAddr[1:0] == 2'b10)
```

```
20 readLUT <= #4 greenLUTout;
```

```
if (uAddr[1:0] == 2'b11)
```

```
readLUT <= #4 blueLUTout;
```

```
end // if
```

```
end // always
```

```
25
```

```
//mux in overlay and use enRed signals
```

```
always @ (posedge pClk) begin
```

```
if (!enRed)
```

```
30 redDx <= #4 8'b0;
```

```
else if (useOverlay)
```

```
redDx <= #4 redOverlay;
```

WO 99/26225

PCT/US98/24215

```

else
redDx <= #4 redLUTout;
if (!enGreen)
greenDx <= #4 8'b0;
5  else if (useOverlay)
greenDx <= #4 greenOverlay;
else
greenDx <= #4 greenLUTout;
if (!enBlue)
10 blueDx <= #4 8'b0;
else if (useOverlay)
blueDx <= #4 blueOverlay;
else
blueDx <= #4 blueLUTout;
15 end // always

//final IO register
always @ (posedge pClk) begin
20 redOut <= #1 redDx;
greenOut <= #1 greenDx;
blueOut <= #1 blueDx;
end // always

25 //Overlay Data path

//Serial Clock

always @ (posedge pClk) begin
30 if (hSyncReal) begin
sClkL <= #4 1'b0;
sClkH <= #4 1'b0;

```

WO 99/26225

PCT/US98/24215

```
end // if
else if (validReal) begin
  if (useLowVram)
    sClkL <= #4 !sClkL;
5  else
    sClkH <= #4 !sClkH;
  end //else if
end // always

10  always @(posedge pClk) begin
    sc <= #4 {sClkH, sClkL};
    sClkInt <= #4 sClkH || sClkL;
    sClkD1 <= #4 sClkInt;
    sClkD2 <= #4 sClkD1;
15  end // always

    //register VRAM serial data

    always @(posedge pClk) begin
20  if (sClkD1)
    overlayIn <= #4 overlay;
    end // always

    always @(posedge pClk) begin
25  if (sClkD2)
    olData <= #4 overlayIn[7:0];
    else
    olData <= #4 overlayIn[15:8];
    end // always
30

    //overlay select
```

WO 99/26225

PCT/US98/24215

```

always @(posedge pClk) begin
  casex (overlaySel)
    3'b000: olDataD1 <= #4 8'b00;
    3'b001: olDataD1 <= #4 {4'b0, olData[3:0]};
5   3'b010: olDataD1 <= #4 {olData[7:4], 4'b0};
    3'b011: begin
      if (olData[3:0] != 4'b0)
        olDataD1 <= #4 {4'b0, olData[3:0]};
      else
10   olDataD1 <= #4 olData;
    end // begin
    3'b1xx: olDataD1 <= #4 olData;
  endcase
  end // always
15

  //generate invert

  always @(posedge pClk)
    olInv <= #4 olDataD1[3:0] == 4'hf || olDataD1[7:4] ==
20   4'hf;

  //add pipeline stages so that data will line up

  always @(posedge pClk) begin
25   olDataD2 <= #4 olDataD1;
    olDataD3 <= #4 olDataD2;
  end // always

  //overlay masking (any 1 will use overlay data)
30  //but not F which signifies invert data...
  //pipeline delay it for time through the EAB

```

WO 99/26225

PCT/US98/24215

```

always @(posedge pClk) begin
  useOverlay <= #4 (olDataD3 != 8'b0 && olDataD3[3:0] != 4'hf &&
    olDataD3[7:4] != 4'hf) || !enOverlay;
  useOverlayD1 <= #4 useOverlay;
5  useOverlayD2 <= #4 useOverlayD1;
  useOverlayD3 <= #4 useOverlayD2;
  end // always

  //mux in uP address for read/writes to LUT
10
  always @(posedge pClk) begin
    if (!redOvlyReq)
      olRedAddr <= #4 olDataD3;
    else
15  olRedAddr <= #4 addrReg[7:0];
    if (!greenOvlyReq)
      olGreenAddr <= #4 olDataD3;
    else
20  olGreenAddr <= #4 addrReg[7:0];
    if (!blueOvlyReq)
      olBlueAddr <= #4 olDataD3;
    else
      olBlueAddr <= #4 addrReg[7:0];
    end // always
25
    always @(posedge pClk) begin
      weOverRed <= redOvlyReq && !uWr_l;
      weOverGreen <= #4 greenOvlyReq && !uWr_l;
      weOverBlue <= #4 blueOvlyReq && !uWr_l;
30  end // always

  //LUTs for Overlay

```

WO 99/26225

PCT/US98/24215

```

syn_ram_256x8_iror u4 (overRed, uDataIn, weOverRed, olRedAddr,
pClk,pClk);
syn_ram_256x8_iror u5 (overGreen, uDataIn, weOverGreen, olGreenAddr,
5  pClk,pClk);
syn_ram_256x8_iror u6 (overBlue, uDataIn, weOverBlue, olBlueAddr,
pClk,pClk);

//register LUT output (for speed)
10
always @(posedge pClk) begin
redOverlay <= #4 overRed;
greenOverlay <= #4 overGreen;
blueOverlay <= #4 overBlue;
15  end // always

//get read data for uP at this stage (may need to decode some earlier)

always @(posedge pClk) begin
20  if (readReqOvly) begin
if (uAddr[1:0] == 2'b00)
readOvly <= #4 overRed;
if (uAddr[1:0] == 2'b01)
readOvly <= #4 overGreen;
25  if (uAddr[1:0] == 2'b10)
readOvly <= #4 overBlue;
end // if
end // always

30
//VRAM IF

```

WO 99/26225

PCT/US98/24215

//sync and edge detect QSF

```
wire qsfIn = useLowVram && qsf[0] || !useLowVram && qsf[1];
```

5

```
always @ (posedge clk) begin
```

```
  qsfD1 <= #4 qsfIn;
```

```
  qsfD2 <= #4 qsfD1;
```

```
  qsfD3 <= #4 qsfD2;
```

10

```
end // always
```

```
wire qsfRise = qsfD2 && !qsfD3;
```

```
wire qsfFall = !qsfD2 && qsfD3;
```

15

```
//Mask off 1st qsf fall (full read has already loaded SAM)
```

```
always @ (posedge clk or negedge reset_l) begin
```

```
  if (!reset_l)
```

```
    qsfMask <= #4 1'b0;
```

20

```
  else begin
```

```
    if (vSyncMemRise)
```

```
      qsfMask <= #4 1'b1;          // set mask on vSync
```

```
      if (qsfRise && xferAddr == 9'h1FF)
```

```
        qsfMask <= #4 1'b1;          // set mask when switching VRAMs
```

25

```
      if (qsfMask && qsfRise)
```

```
        qsfMask <= #4 1'b0;          // clear mask after first rising edge of
        qsf
```

```
      end // else
```

```
    end // always
```

30

```
//generate split read request
```

WO 99/26225

PCT/US98/24215

```

always @(posedge clk or posedge clrXferReq) begin
  if (clrXferReq)
    splitReq <= #4 1'b0;
  else begin
5    if (qsfRise || (!qsfMask && qsfFall))
      splitReq <= #4 1'b1;
    end // else
  end // always

10  wire clrXferReq = !reset_1 || (memDone && cycleType);

    //Keep up with which bank is currently used

    always @(posedge clk or posedge vSyncMem) begin
15    if (vSyncMem)
      useLowVram <= 1'b1;
      if (qsfFall && xferAddr == 8'hFF)
        useLowVram <= 1'b0;
      end // always

20    //generate Full read request (Note Should do both VRAM with the same
    cycle)

    always @(posedge clk or posedge clrXferReq) begin
25    if (clrXferReq)
      fullReq <= #4 1'b0;
      if (vSyncMemRise)
        fullReq <= #4 1'b1;
      end // always

30    //combinatorial request

```

WO 99/26225

PCT/US98/24215

```
assign vramWrReq = memReq && !uWr_l;
```

```
assign vramRdReq = memReq && !uRd_l;
```

```
//memory sequence
```

```
5  vramsm vramsm (clk,reset_l,memPage,wrMode,{uAddr,  
    addrReg},uData,memRdData,  
    cbrReq, vramWrReq, vramRdReq,colorReq,splitReq,fullReq,blockReq,  
    useLowVram,  
    memDone,cycleType,  
10  vAddr,vData,ras_l,cas_l,dsf,dtoe,we_l[1], we_l[0]);
```

```
//
```

```
endmodule
```

CLAIMS

We claim:

- 1 1. A data planarizer comprising:
2 a plurality of data input terminals for receiving n-bit data words;
3 a plurality of data output terminals for outputting planarized m-bit data words; and
4 a first storage bank having an input terminal set coupled to said plurality of data input
5 terminals, an output terminal set coupled to said plurality of data output terminals,
6 and a bit-depth less than m, for receiving and accumulating said n-bit data words, and
7 for outputting said accumulated data as first portions of said m-bit planarized data
8 words.
- 1 2. A data planarizer according to Claim 1, wherein said first storage bank comprises a bi-
2 directional shift register, adapted to shift said n-bit data words into said first storage bank in a
3 first direction and to shift said first portions of said m-bit data words out of said first storage
4 bank in a different direction.
- 1 3. A data planarizer according to Claim 2, wherein said bi-directional shift register
2 comprises:
3 a clock-input terminal for receiving a data shift signal; and
4 a direction terminal for receiving a direction control signal.
- 1 4. A data planarizer according to Claim 3, wherein said bi-directional shift register
2 further comprises:
3 a plurality of flip-flops arranged in a rectangular array of columns and rows, each of said
4 flip-flops having a control terminal coupled to said clock input terminal, an input
5 terminal, and an output terminal; and
6 a plurality of multiplexers arranged in a rectangular array of columns and rows, each of
7 said multiplexers having a control terminal coupled to said direction terminal, an

PEA/US 01 FEB 200

8 output terminal coupled to said input terminal of an associated one of said flip-flops,
9 a first input terminal, and a second input terminal.

1 5. A data planarizer according to Claim 4, wherein:
2 said multiplexers of a first column of said array of multiplexers each has said first input
3 terminal coupled to an associated one of said plurality of data input terminals;
4 said multiplexers of other columns of said array of multiplexers each has said first input
5 terminal coupled to said output terminal of an associated one of said flip-flops located
6 in an adjacent column;
7 said multiplexers of rows other than a top row of said array of multiplexers each has said
8 second input terminal coupled to the output terminal of an associated one of said flip-
9 fops of an adjacent row; and
10 said flip-flops of a bottom row of said array of flip-flops each has said output terminal
11 coupled to an associated one of said plurality of data output terminals.

1 6. A data planarizer in accordance with Claim 1, further comprising a second storage
2 bank having an input terminal set coupled to said plurality of data input terminals, an output
3 terminal set coupled to said plurality of data output terminals, and a bit-depth less than m, for
4 receiving and accumulating said n-bit data words, and for outputting said accumulated data as
5 second portions of said m-bit planarized data words.

1 7. A data planarizer in accordance with Claim 6, wherein said first storage bank and said
2 second storage bank have a combined capacity of less than one frame of data.

1 8. A data planarizer in accordance with Claim 7, wherein:
2 said first storage bank comprises a bi-directional shift register, adapted to shift said n-bit
3 data words into said first storage bank in a first direction and to shift said first
4 portions of said m-bit data words out of said first storage bank in a different direction,
5 and having a clock-input terminal for receiving a data shift signal, and a direction
6 terminal for receiving a direction control signal; and

7 said second storage bank comprises a bi-directional shift register, adapted to shift said n-
8 bit data words into said second storage bank in said first direction and to shift said
9 second portions of said m-bit data words out of said second storage bank in said
10 different direction, and having a clock-input terminal for receiving a data shift signal,
11 and a direction terminal for receiving a direction control signal.

1 9. A data planarizer in accordance with Claim 8, further comprising an inverter having
2 an input terminal coupled to said direction terminal of said first bi-directional shift register and
3 an output terminal coupled to said direction terminal of said second bi-directional shift register.

1 10. A data planarizer according to Claim 9, wherein each of said bi-directional shift
2 registers further comprises:
3 a plurality of flip-flops arranged in a rectangular array of columns and rows, each of said
4 flip-flops having a control terminal coupled to said clock input terminal, an input
5 terminal, and an output terminal; and
6 a plurality of multiplexers arranged in a rectangular array of columns and rows, each of
7 said multiplexers having a control terminal coupled to said direction terminal, an
8 output terminal coupled to said input terminal of an associated one of said flip-flops,
9 a first input terminal, and a second input terminal.

1 11. A data planarizer according to Claim 10, wherein:
2 said multiplexers of a first column of said array of multiplexers each has said first input
3 terminal coupled to an associated one of said plurality of data input terminals;
4 said multiplexers of other columns of said array of multiplexers each has said first input
5 terminal coupled to said output terminal of an associated one of said flip-flops located
6 in an adjacent column;
7 said multiplexers of rows other than a top row of said array of multiplexers each has said
8 second input terminal coupled to the output terminal of an associated one of said flip-
9 flops of an adjacent row; and

10 said flip-flops of a bottom row of said array of flip-flops each has said output terminal
11 coupled to an associated one of said plurality of data output terminals.

1 12. A method for planarizing n-bit data into m-bit planarized data, said method
2 comprising the steps of:
3 receiving a first group of (p) n-bit data words, where $p < m$, and where $n > 1$;
4 storing said first group of (p) data words in a first storage bank; and
5 transferring said first group of (p) data words into a device comprising (n) different
6 memory locations, by transferring a p-bit data word comprising one bit from each of
7 said first group of (p) n-bit data words into a first portion of each of said memory
8 locations.

1 13. A method according to Claim 12, further comprising the steps of:
2 receiving a second group of (p) n-bit data words;
3 storing said second group of (p) data words in said first storage bank; and
4 transferring said second group of (p) data words into said device by transferring a p-bit
5 data word comprising one bit from each of said second group of (p) n-bit data words
6 into a second portion of each of said memory locations.

1 14. A method according to Claim 13, further comprising the steps of:
2 receiving subsequent groups of (p) n-bit data words;
3 storing said subsequent groups of (p) data words in said first storage bank; and
4 transferring said subsequent groups of (p) data words into said device by transferring a p-
5 bit data word comprising one bit from each of said subsequent groups of (p) n-bit data
6 words into subsequent portions of each of said memory locations, until each of said
7 memory locations contains an m-bit data word.

1 15. A method according to Claim 12, further comprising the steps of:
2 receiving a second group of (p) n-bit data words;
3 storing said second group of (p) data words in a second storage bank; and
4 transferring said second group of (p) data words into said device by transferring a p-bit
5 data word comprising one bit from each of said second group of (p) n-bit data words
6 into a second portion of each of said memory locations.

1 16. A method according to Claim 15, further comprising the steps of:
2 receiving subsequent groups of (p) n-bit data words;
3 storing said subsequent groups of (p) data words in one of said first and second storage
4 banks; and
5 transferring said subsequent groups of (p) data words into said device by transferring a p-
6 bit data word comprising one bit from each of said subsequent groups of (p) n-bit data
7 words into subsequent portions of each of said memory locations, until each of said
8 memory locations contains an m-bit data word.

1 17. A method according to Claim 16, wherein said step of storing said subsequent groups
2 of (p) data words in one of said first and second storage banks is performed in alternating fashion
3 between said first and said second storage banks.

1 18. A method according to Claim 15, wherein said step of storing said second group of
2 (p) data words in said second storage bank is performed while said step of transferring said first
3 group of (p) data words is performed.

1 19. A method according to Claim 12, wherein said step of storing said first group of (p)
2 data words in a first storage bank comprises the steps of shifting said (p) data words into said
3 first storage bank along a first direction, (n) bits at a time.

1 20. A method according to Claim 19, wherein said step of transferring said first group of
2 (p) data words into a device comprising (n) different memory locations comprises the steps of
3 shifting said (p) data words out of said first storage bank along a second direction, (p) bits at a
4 time.

1 21. A data planarizer for planarizing n-bit data into m-bit planarized data, said data
2 planarizer comprising:
3 receiving means for receiving a group of (p) n-bit data words, where $p < m$, and where
4 $n > 1$;
5 storage means for storing said group of (p) data words; and
6 transfer means for transferring said group of (p) data words into a device comprising (n)
7 different memory locations, by transferring a p-bit data word comprising one bit from
8 each of said group of (p) n-bit data words into a first portion of each of said memory
9 locations.

1 22. A data planarizer according to Claim 21, wherein said first storage means comprises:
2 first shifting means for shifting data into said first storage means (n) bits at a time; and
3 second shifting means for shifting data out of said first storage means (p) bits at a time.

23. A data planarizer comprising:

a plurality of data input terminals for receiving n-bit data words;

a plurality of data output terminals for outputting at least portions of planarized m-bit data words; and

a bi-directional shift register having an input terminal set coupled to said plurality of data input terminals and an output terminal set coupled to said plurality of data output terminals, said bi-directional shift register operative to shift said n-bit data words from said data input terminals into said bi-directional shift register in a first direction and to shift at least a portion of said planerized m-bit data words out of said bi-directional shift register to said output terminals in a second direction different from said first direction.

24. A data planarizer according to Claim 23, wherein said first direction is orthogonal to said first direction.

1/10

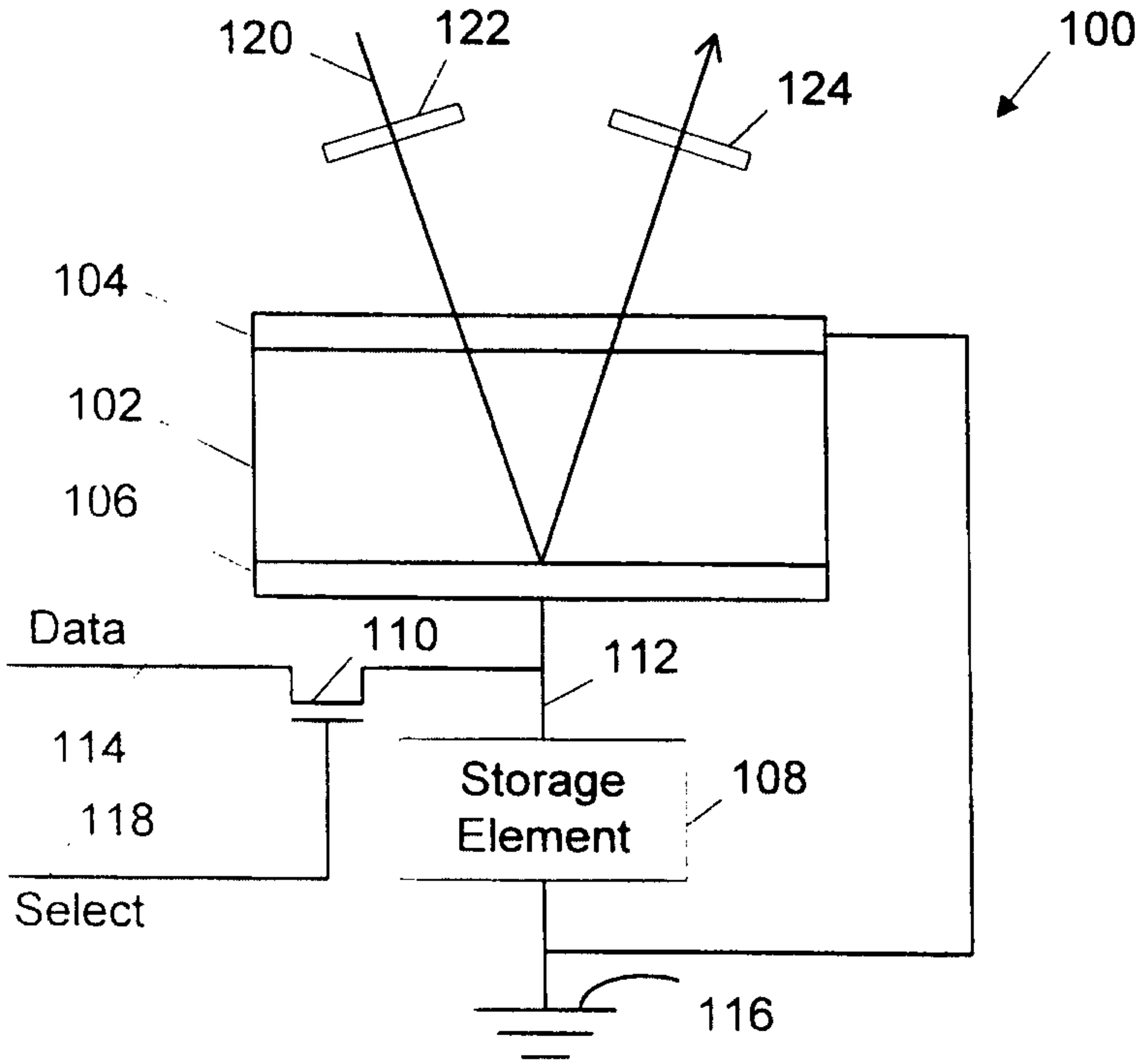


FIG. 1
Prior Art

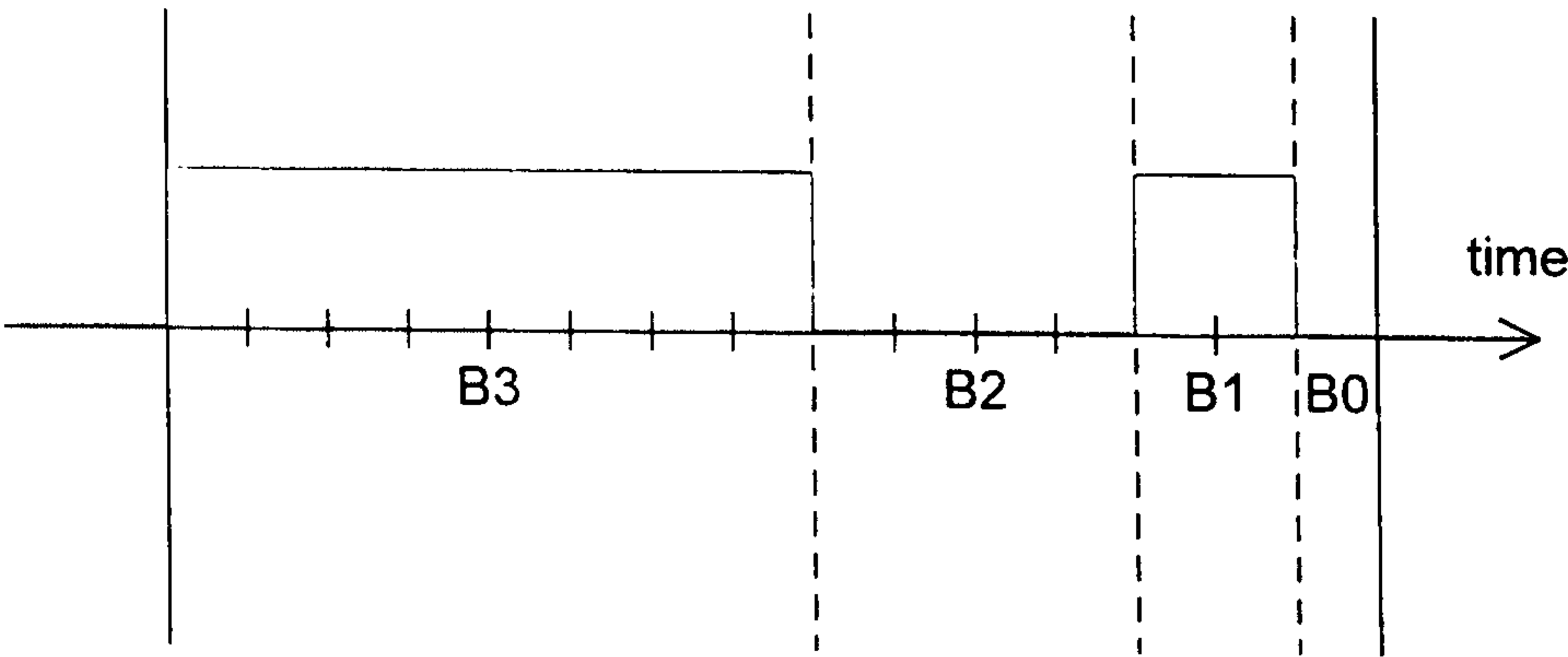
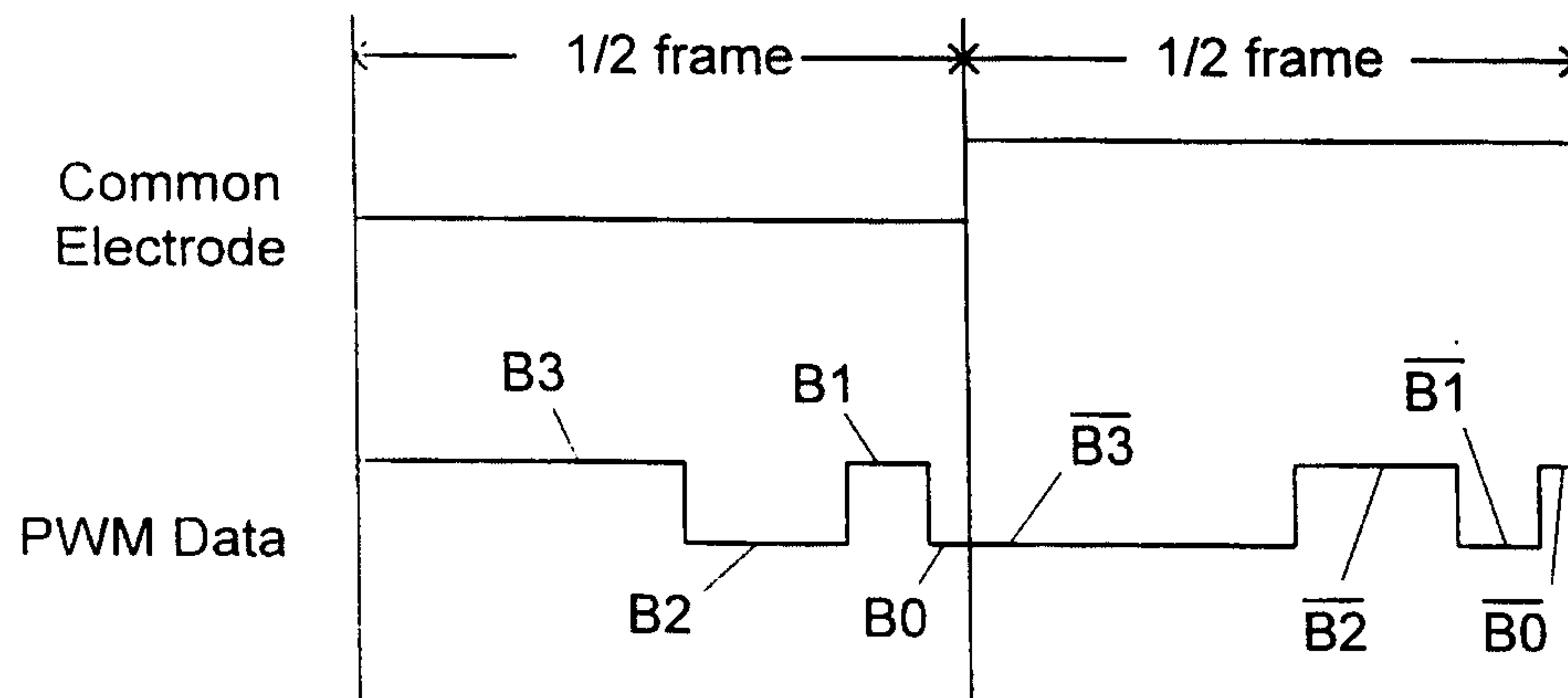
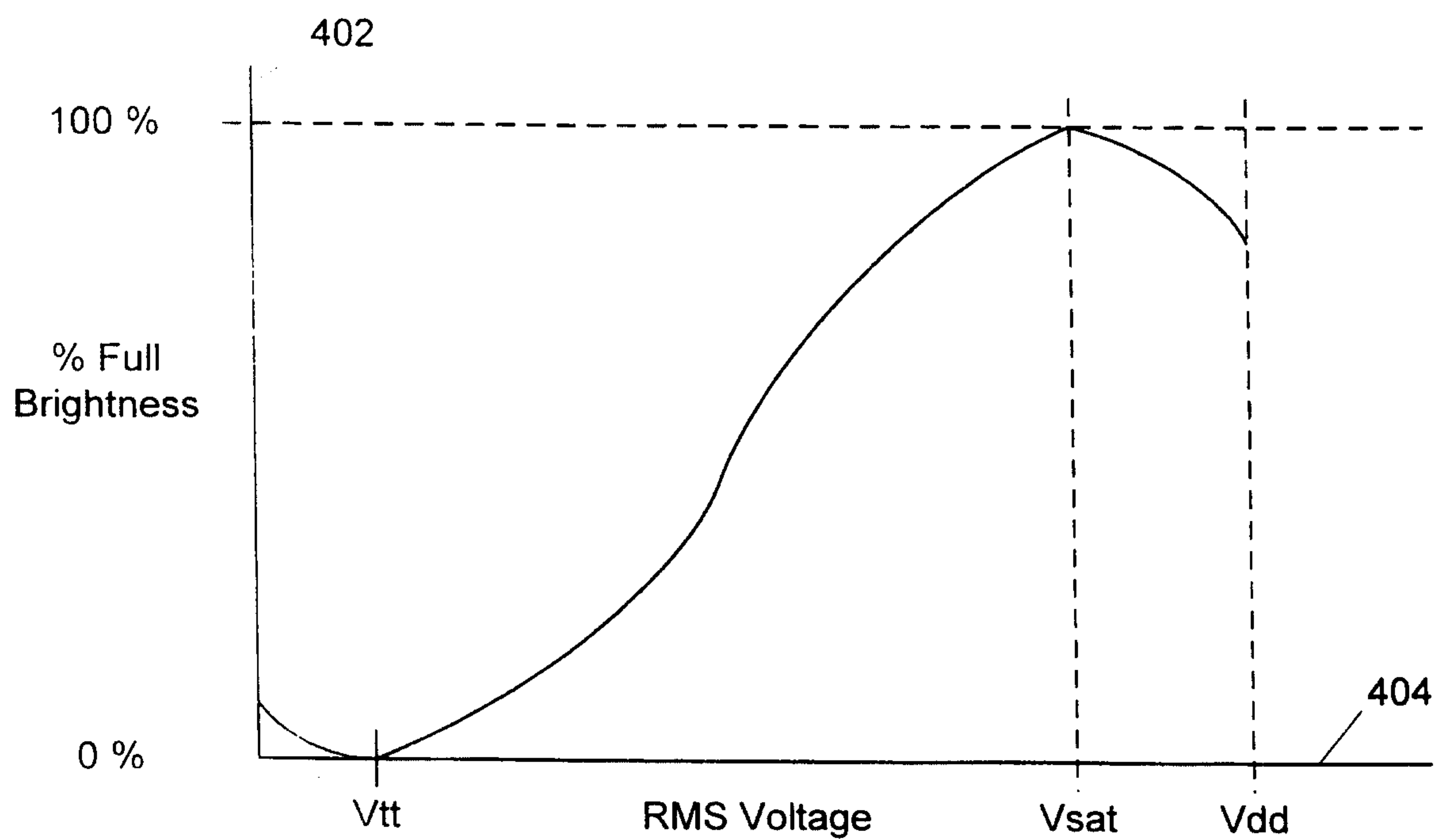
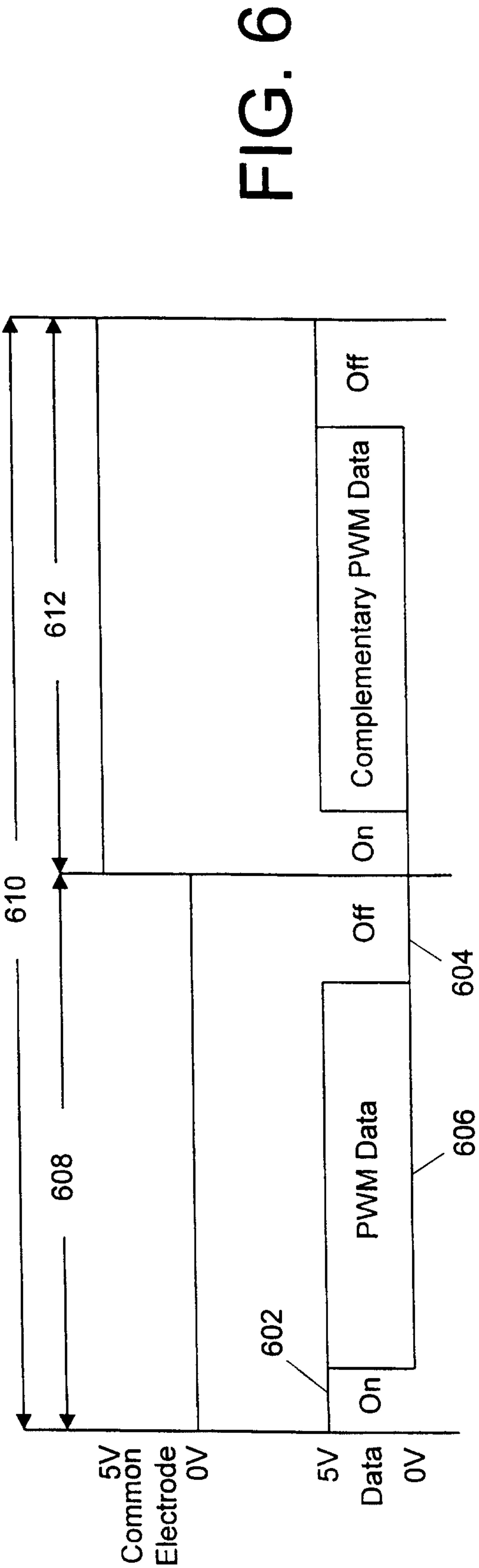
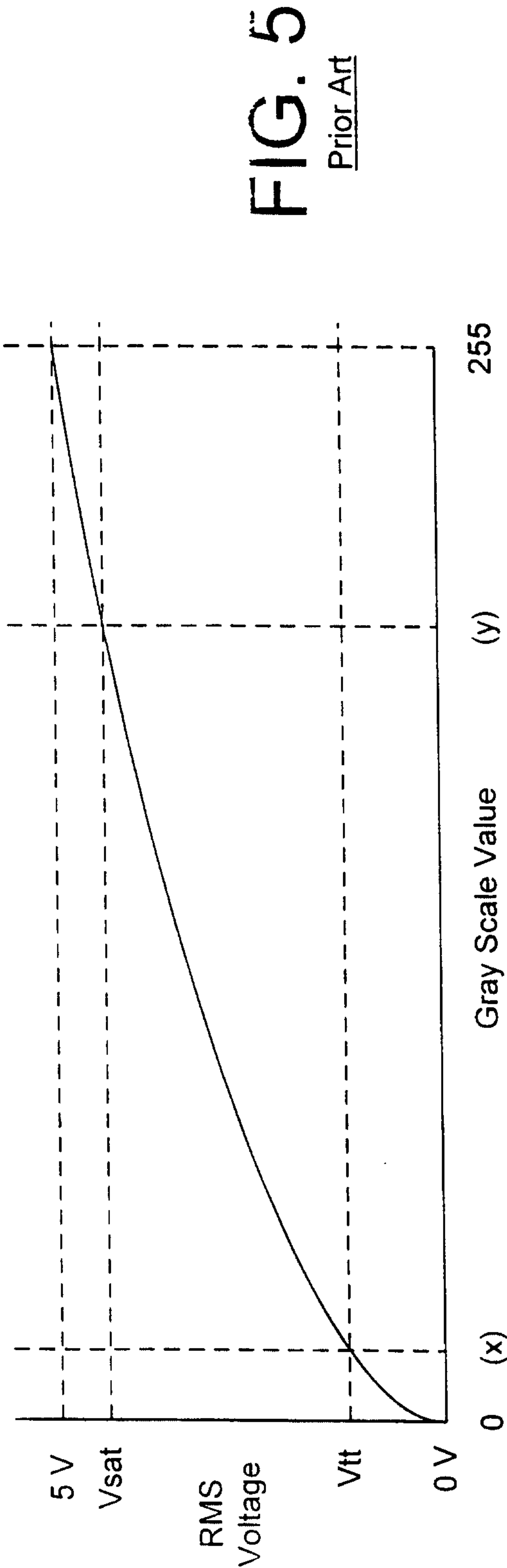


FIG. 2
Prior Art

2/10

**FIG. 3**Prior Art**FIG. 4**Prior Art



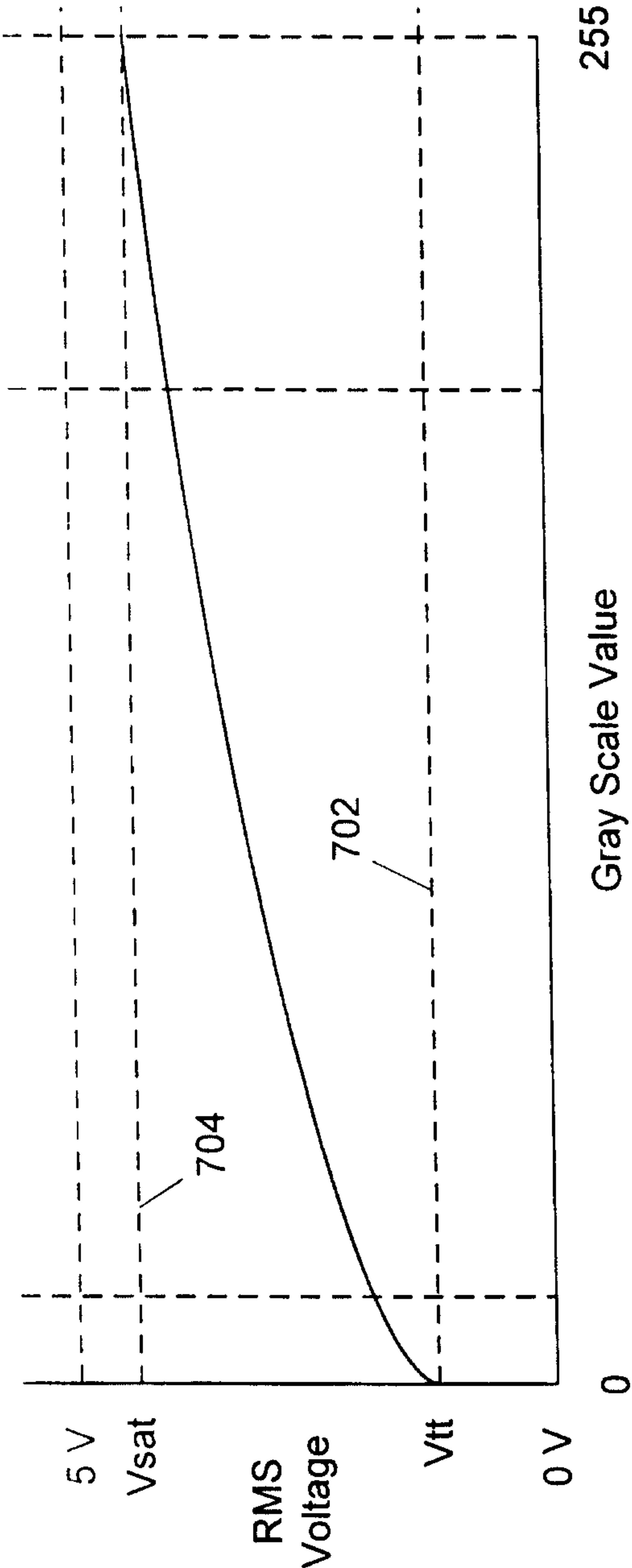


FIG. 7

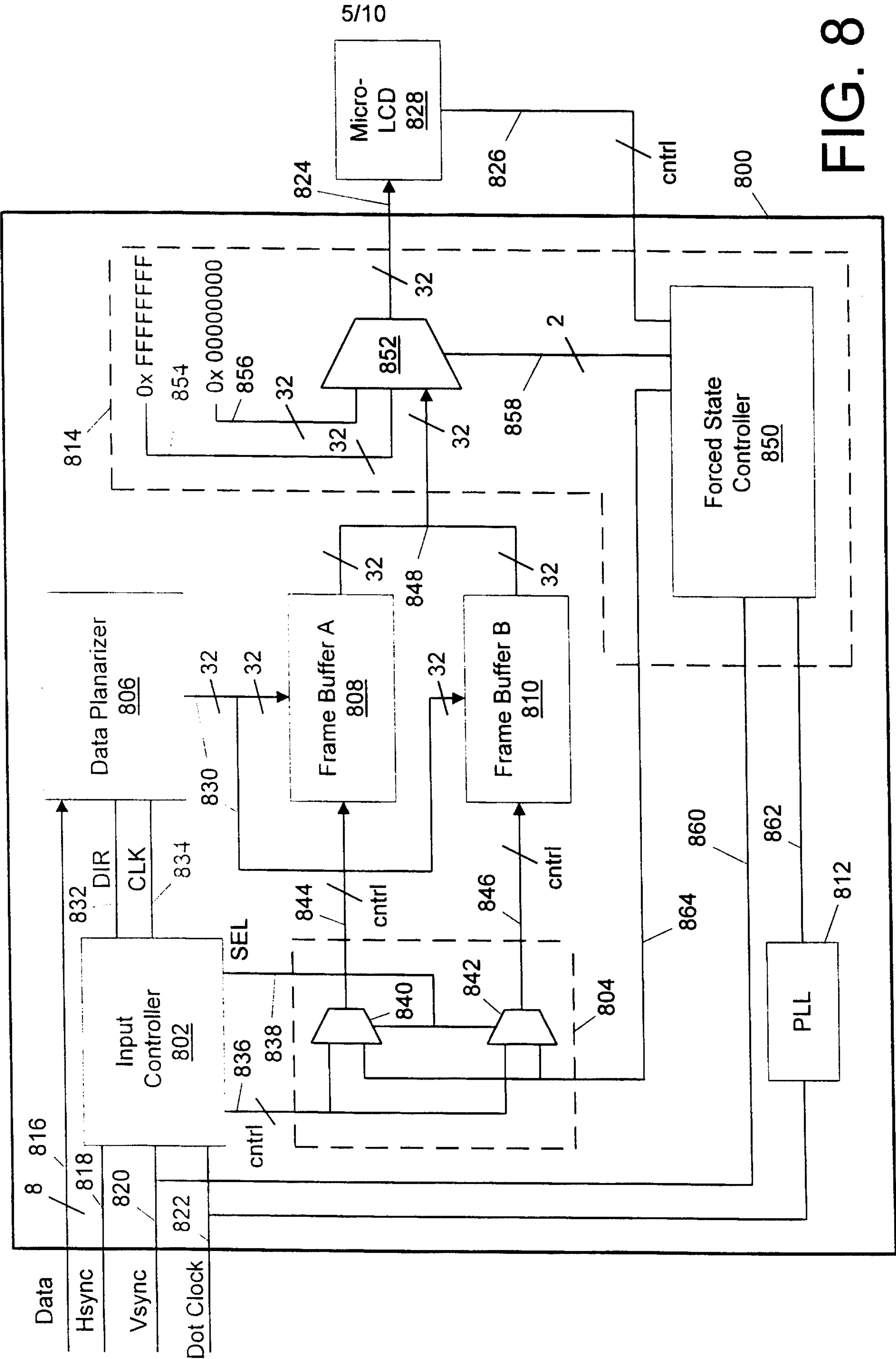


FIG. 8

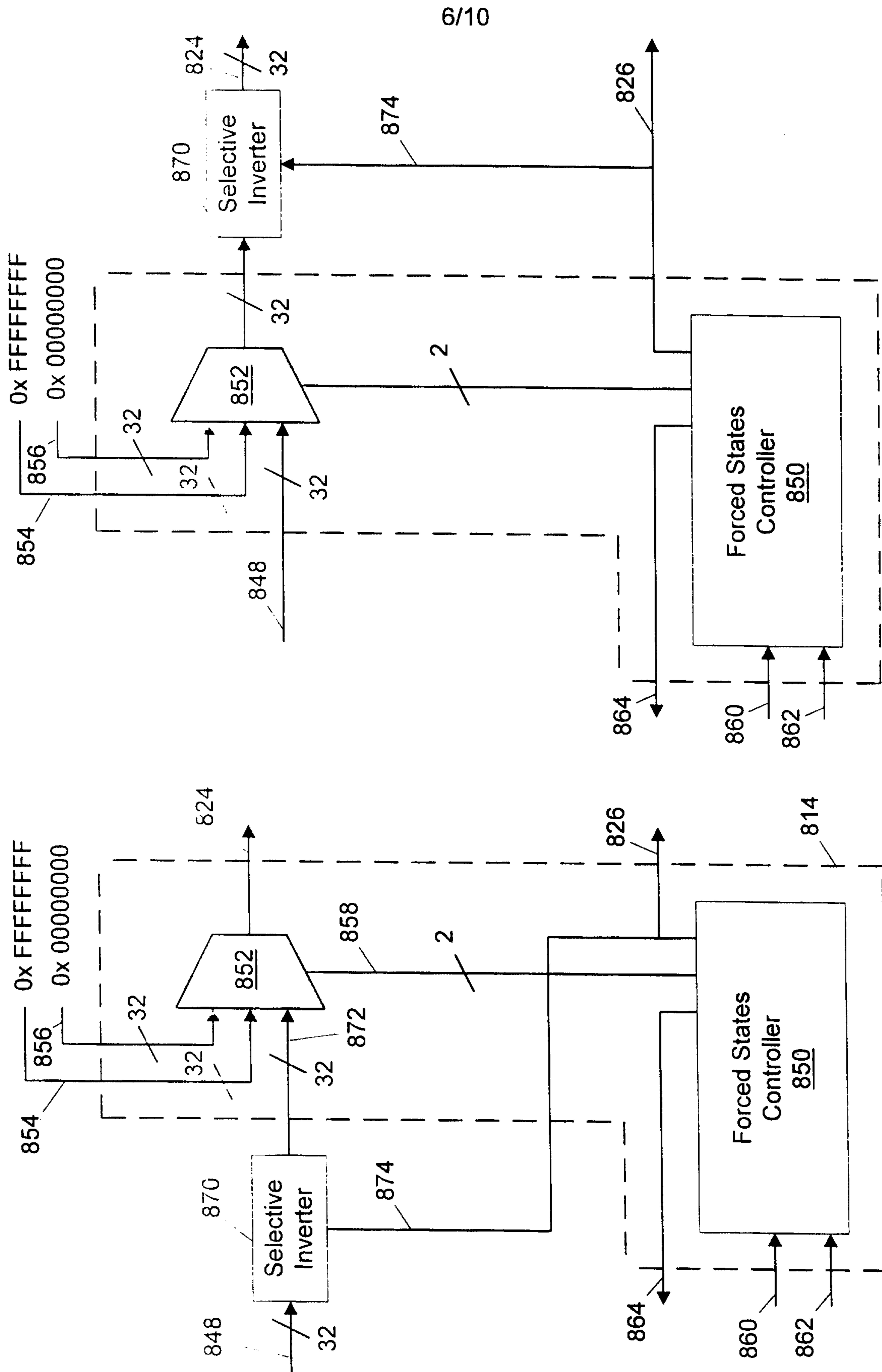


FIG. 8a

FIG. 8b

7/10

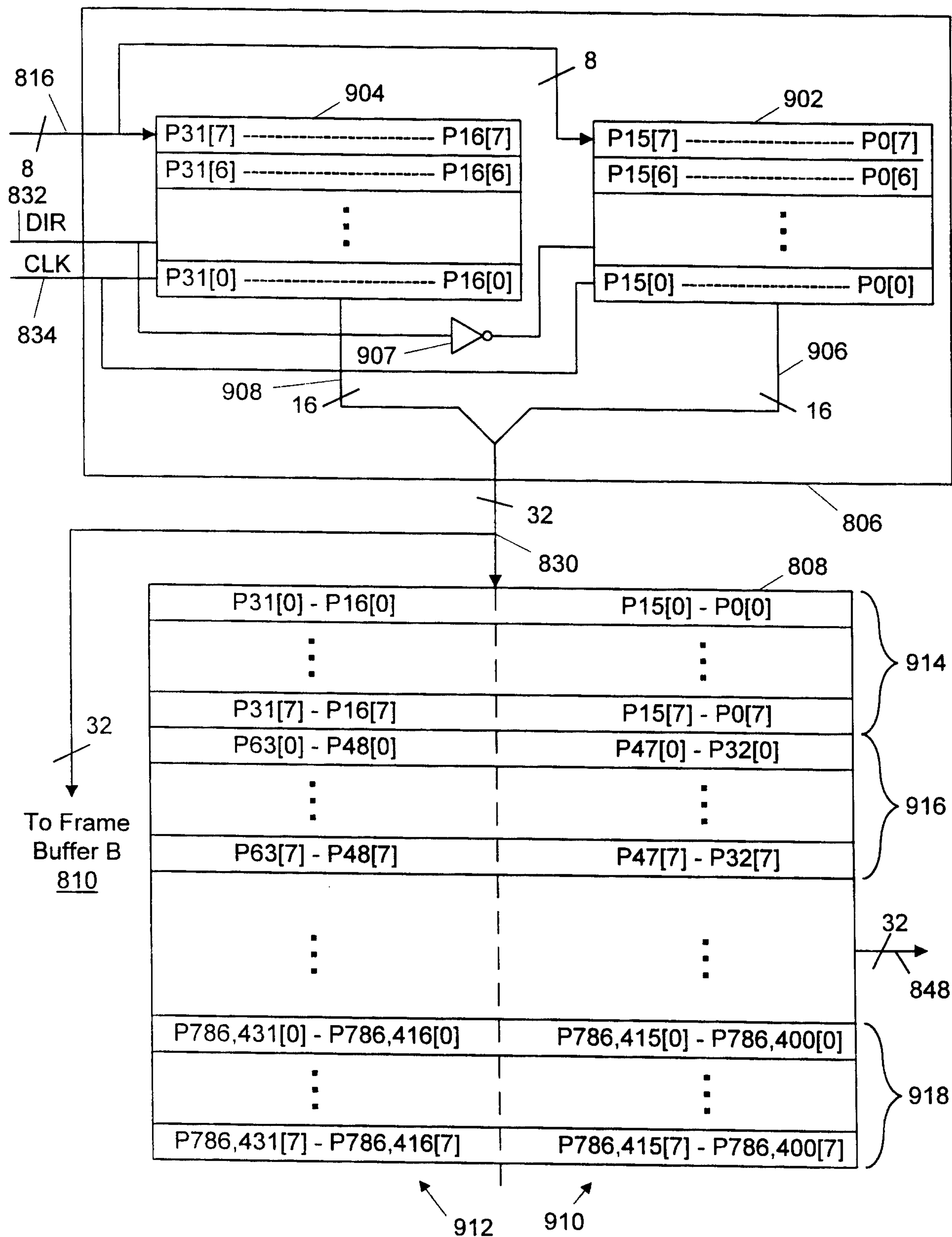
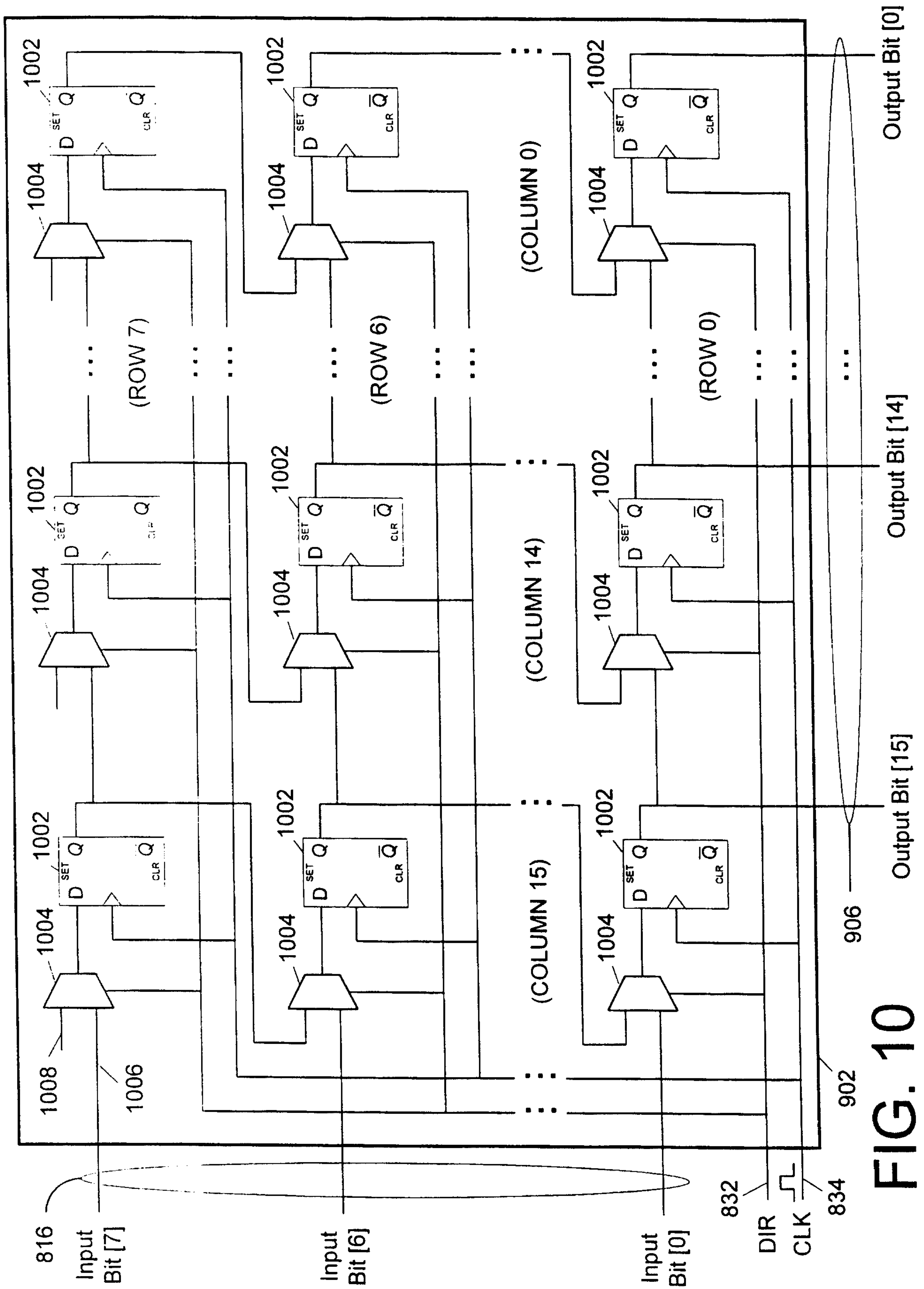


FIG. 9



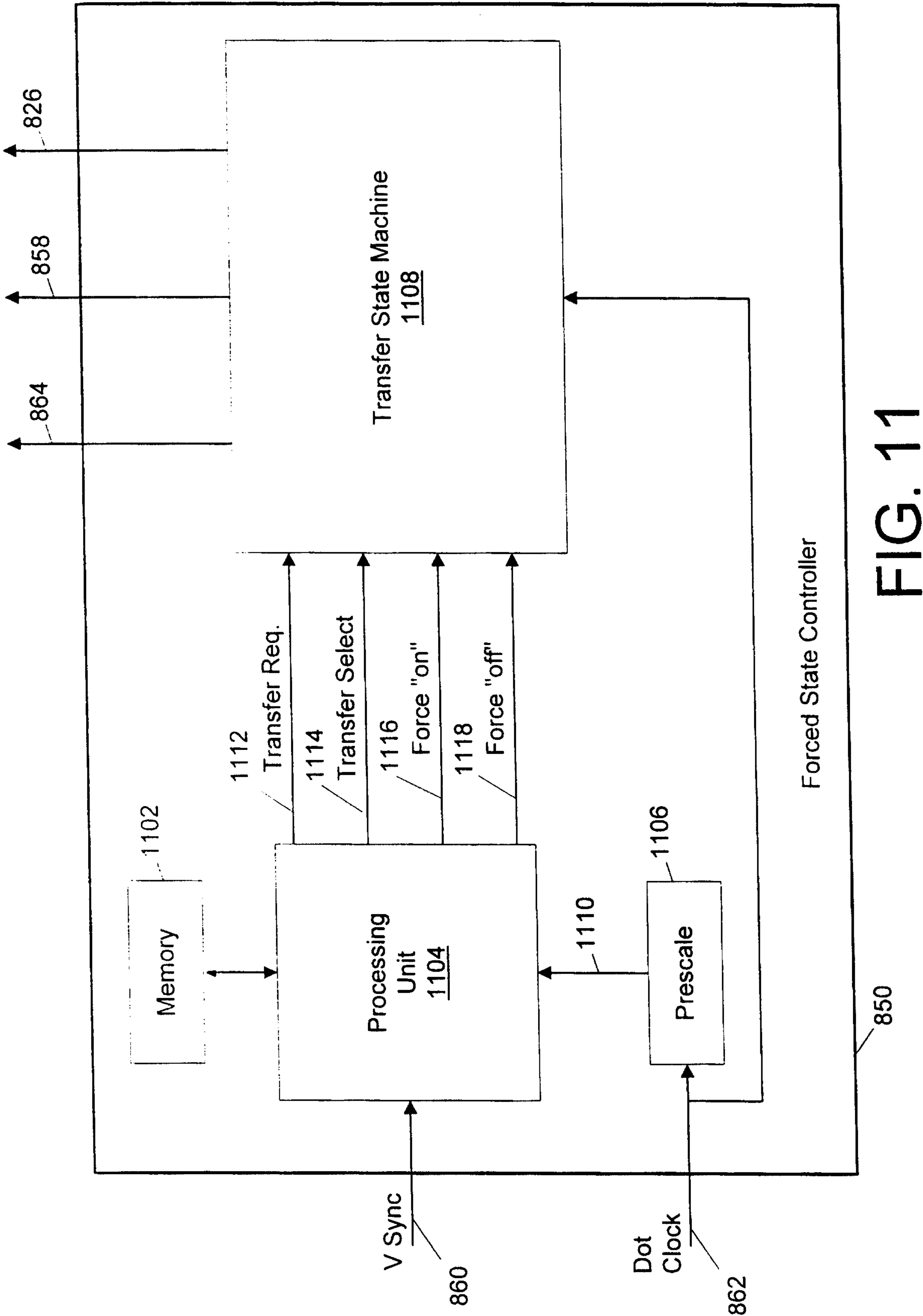


FIG. 11

10/10

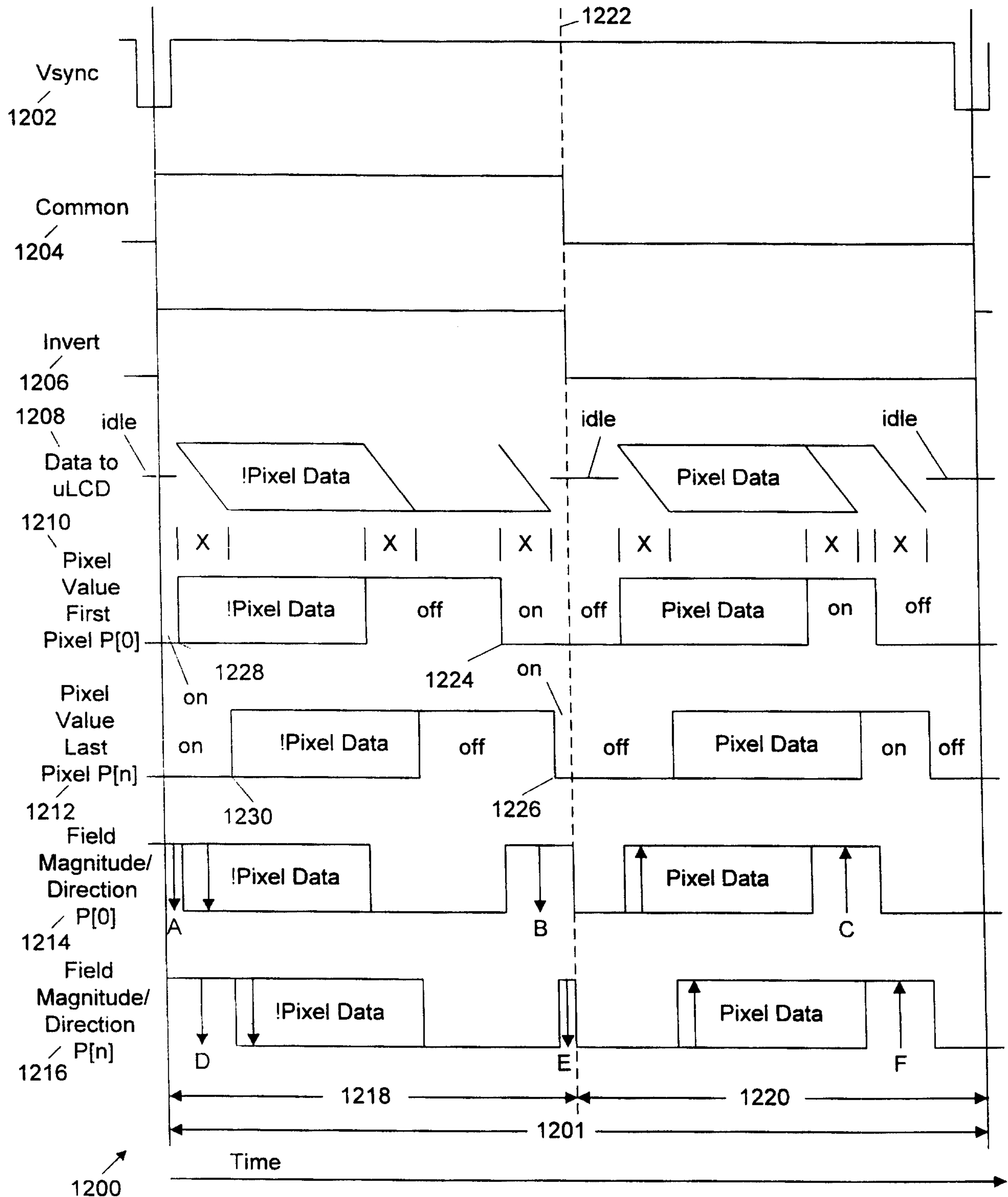


FIG. 12

