SPPU019A

Color Graphics Controller Board

User's Guide

Graphics Products

TEXAS INSTRUMENTS

Color Graphics Controller User's Guide

Graphics Products

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes in the devices or the device specifications identified in this publication without notice. TI advises its customers to obtain the latest version of device specifications to verify, before placing orders, that the information being relied upon by the customer is current.

In the absence of written agreement to the contrary, TI assumes no liability for TI applications assistance, customer's product design, or infringement of patents or copyrights of third parties by or arising from use of semiconductor devices described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor devices might be or are used.

Contents

Illustrations

Tables

1. Introduction

The Color Graphics Controller Board is a TI PC and IBM PC I/O expansion bus compatible graphics card which allows you to become familiar with the TMS34061 Video System Controller and TMS34070 Color Palette. The board and demonstration software are configured at shipment to support a 640 pixel horizontal by 480 pixel vertical resolution, similar to that of the IBM Professional Graphics Display. By changing the crystal oscillator and reprogramming the TMS34061 control registers, the board can support a maximum resolution of 1024 by 512 pixels, with four bits per pixel.

Note:

Hexidecimal numbers are designated herein by the suffix "h".

1.1 Functional Overview

The Color Graphics Controller Board is a single card designed around the IBM PC I/O Expansion Bus card format. The board is intended to demonstrate the simplicity and strengths of the TMS34061 Video System Controller and TMS34070 Color Palette in controlling text and graphics while showing the ease of hardware design. The board has the capability to drive both digital and analog raster scan color monitors.

The frame buffer consists of 32 TMS4161 multiport memories organized as four color planes, allowing for the possibility of 16 colors per frame from the digital outputs. The color palette incorporates a 12-bit color look-up table which gives the programmer a choice of 16 colors from a palette of 4096 colors per frame. Furthermore, the palette incorporates a unique line feature which allows the color look-up table to be reloaded on every line, allowing 16 of 4096 colors per line. The programmer can change this mode under software control.

The maximum resolution that the board can support is 1024 by 512 pixels with four bits per pixel. The memory map shown in Section 2.2 can be changed by reprogramming the memory and control decode PALs (programmable array logic).

1.2 Features

- 256K-byte frame buffer (1024 x 512 pixels, four bits per pixel)
- Direct interface to most digital and analog RGB monitors
- Totally programmable resolution (640 x 480 pixels as shipped)
- IBM and TI PC I/O expansion bus compatible
- 16 of 4096 colors per line color palette
- Palette can be loaded on a line or frame basis under software control

The block diagram for the Color Graphics Controller Board is shown in Figure 1-1.

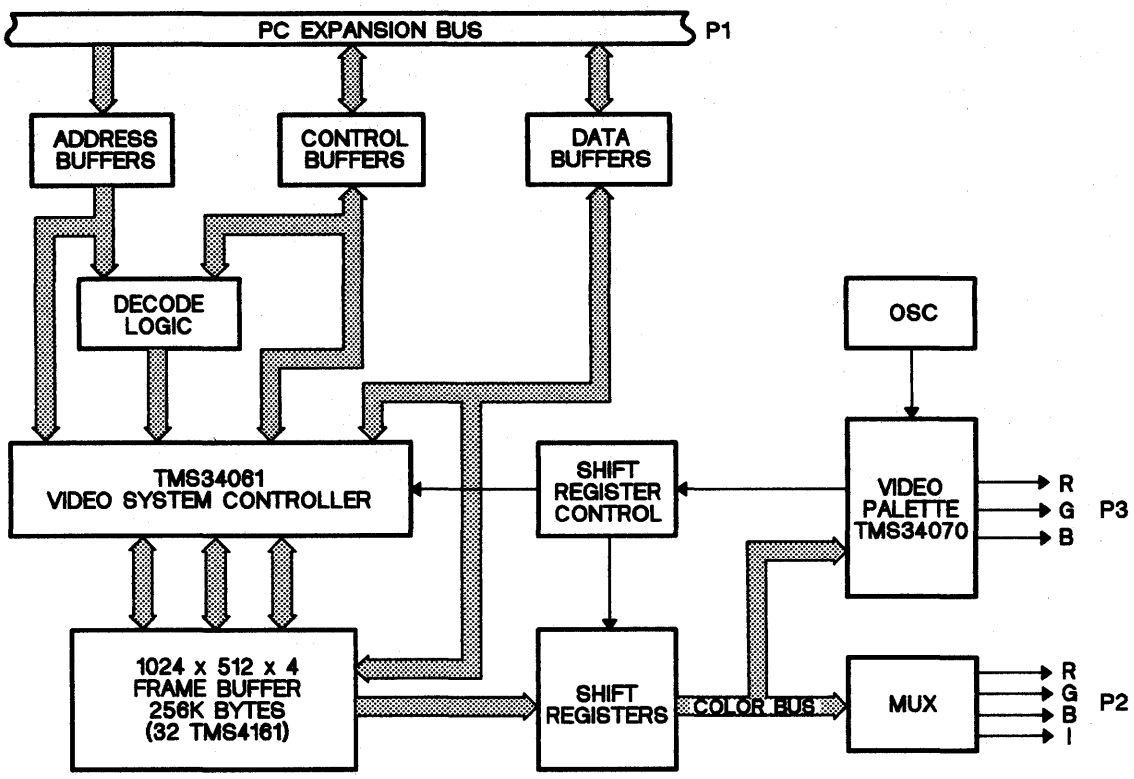


Figure 1-1. Color Graphics Controller Board Block Diagram

2. Quick Installation Guide

The Texas Instruments TMDS3471804000 Color Graphics Controller (CGC) is designed to operate when installed in a TI Professional Computer or IBM Personal Computer. The following is a step-by-step quick installation guide explaining how to install the CGC card and execute the main demonstration program. See Section 3 for further installation information.

1) The CGC comes configured for display on an IBM Professional Graphics Display or Princeton Graphics SR-12 driven by an IBM PC as the host computer. If a TI PC is being used as the host, then the jumper options will have to be reconfigured. The jumper options are shown below.

2) Make certain the host computer has 256K bytes of system RAM; otherwise, system RAM will overlap CGC RAM, and the demonstration will not operate. The CGC is memory-mapped from 50000h to AFFFFh.

3) Install the CGC in a vacant slot in the host computer, and connect the CGC video output (the top port, P3) to the IBM Professional Graphics Display or the Princeton Graphics SR-12.

4) Boot an MS-DOS system master on the host computer. Make certain that the version of MS-DOS is version 2.1 or later. Once MS-DOS is loaded, remove the system master and insert the CGC demonstration disk into the drive.

5) Now that everything is configured, type "AUTOEXEC" to start execution of the main demonstration. Once execution begins, press "Q" to terminate. Additional information on the demonstration programs is contained in the text file called "READ.ME" located on the demonstration diskette.

HOST: IBM PC
MONITOR: IBM Professional Graphics Display
OUTPUT PORT: P3
JUMPERS:

J01	1 to 2
J02	2 to 3
J03	2 to 3 and 4 to 5
J04	2 to 5
J05	Not Connected
J06	Not Connected

HOST: TI PC
MONITOR: IBM Professional Graphics Display
OUTPUT PORT: P3
JUMPERS:

J01	1 to 2
J02	1 to 2
J03	2 to 3 and 4 to 5
J04	2 to 5
J05	Not Connected
J06	Not Connected

3. Installation and Operation

This section describes the necessary configuration procedures for correct operation of the Color Graphics Controller Board. For information on how to program the TMS34061 Video System Controller and the TMS34070 Color Palette, refer to the TMS34061 User's Guide, part number SPPU014, and the TMS34070 User's Guide, part number SPPU016.

3.1 Hardware Configurable Jumpers

The Color Graphics Controller Board has hardware-configurable jumpers for such things as interrupts and monitor interfaces. Table 3-1 lists the jumpers and the features that they control.

Table 3-1. Jumper Settings and Features

FEATURE ENABLED	JUMPER	POSITION
On board oscillator	J01	1 to 2†
Backplane oscillator	J01	2 to 3
Interrupt level 2 (TI PC)	J02	1 to 2
Interrupt level 3 (IBM PC)	J02	2 to 3†
Negative horizontal sync to P2-8	J03	1 to 2
Positive horizontal sync to P2-8	J03	1 to 4 2 to 3
Negative vertical sync to P2-9	J04	1 to 2
Positive vertical sync to P2-9	J04	1 to 4 2 to 3
Ground to P2-6 (TI PC digital monitor)	J05	2 to 3
I to P2-6 (IBM Personal Computer color display)	J05	1 to 2†
Negative composite sync to P2-9	J04	2 to 5
Positive composite sync to P2-9	J04	2 to 3 4 to 5
Negative horizontal sync to P3-5	J03	1 to 2
Positive horizontal sync to P3-5	J03	1 to 4 2 to 3
Negative vertical sync to P3-4	J04	1 to 2
Positive vertical sync to P3-4	J04	1 to 4 2 to 3
Negative composite sync to P3-4	J04	2 to 5†
Positive composite sync to P3-4	J04	2 to 3 4 to 5
P3-5 to logic 0	J03	2 to 5
P3-5 to logic 1	J03	2 to 3† 4 to 5
Remove host direct memory from board memory map	J06	1 to 2

† Jumper setting as shipped

Figure 3-1 shows the location of the jumpers on the Color Graphics Controller Board.

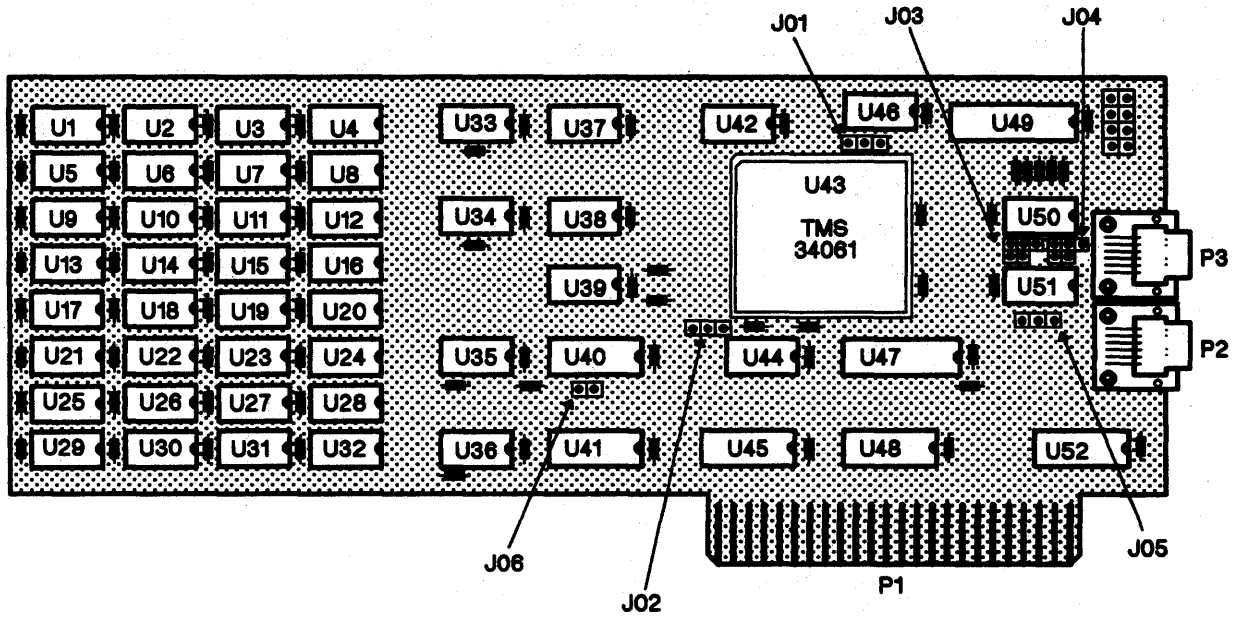


Figure 3-1. Location of Jumpers on the Board

3.2 Memory Map

The TMS34061 allows you to control five functions through the function select lines FS0-FS2:

- Host memory direct access
- Shift register to memory
- Memory to shift register
- TMS34061 register access
- X-Y indirect access range

These functions have been placed in the PC memory space on the board. The Color Palette Modes are memory-mapped and programmable under software control. The memory map for the board is determined by the equations programmed in the decode PALs U40, U45, and U48. Table 3-2 defines the memory map and contains a brief description of the function mapped in each area. Table 3-3 defines the palette memory map and how to program the mode by reading a set of addresses.

Table 3-2. Color Graphics Controller Board Memory Map

MEMORY RANGE	TMS34061 FUNCTION	FS0	FS1	FS2
50000h–8FFFFh†	HOST MEMORY DIRECT ACCESS	1	1	0
903F0h–903FFh	PALETTE MODE CONTROL	1	1	1
90400h–907XXh	SHIFT REGISTER TO MEMORY	0	0	1
90800h–90BXXh	MEMORY TO SHIFT REGISTER	1	0	1
90C00h–90DXXh	TMS34061 REGISTER ACCESS	0	0	0
90E00h–90F00h‡	X-Y INDIRECT ACCESS RANGE	1	0	0
→ $X000_2$	X-Y INDIRECT BYTE ACCESS	1	0	0
→ $X001_2$	X-Y INDIRECT RIGHT NIBBLE	1	0	0
→ $X010_2$	X-Y INDIRECT LEFT NIBBLE	1	0	0
→ $X011_2$	X-Y INDIRECT NO ACCESS	1	0	0
A0000h–AXXXXh‡	X-Y INDIRECT 2-WORD ACCESS	1	0	0

Note: Left nibble = D7 - D4, right nibble = D3 - D0 on the host/controller board interface.

† If jumper J06 is installed (1 to 2), all host direct accesses are ignored to allow for a larger memory space for system RAM.

‡ See Section 3.5 for determining how to construct an X-Y address.

The three Color Palette modes are latched by reading the addresses listed in Table 3-3. Reading these addresses latches signals P0 and P1 as shown.

Table 3-3. Latching the Color Palette Modes

PALETTE MODE	SIGNALS SET		READ ADDRESS
	P0	P1	
Frame Load Mode	0	0	903F0h and 903F2h
Line Load Mode	0	1	903F0h and 903F6h
High (No Load)	1	0	903F4h and 903F2h
Reserved	1	1	903F4h and 903F6h

3.3 Video Memory Organization

The frame buffer (video memory) is comprised of 32 TMS4161 multiport video RAM. These 256K bytes of multiport memory provide the board with the capability of displaying screen sizes up to 1024 x 512 pixels with four bits per pixel.

The memory is organized in packed pixels, with two pixels sharing one byte of memory. Figure 3-2 illustrates the correlation between the contents of two consecutive memory addresses and the pixels they correspond to on the display screen.

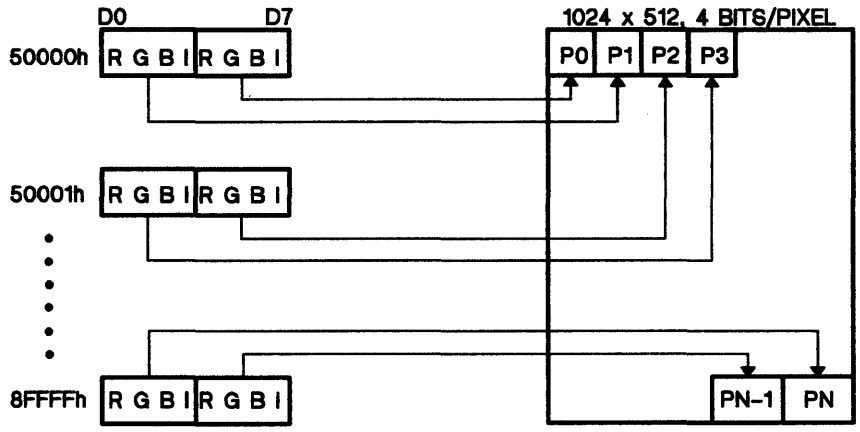


Figure 3-2. Frame Buffer to Display Screen Memory Translation

The frame buffer occupies a contiguous block of memory from 50000h to 8FFFFh. The memory is not mapped contiguous to the system memory so that the auto-sizing program will not load the operating system into video memory space.

3.4 TMS34061 Internal Registers

The 18 internal registers of the TMS34061 are mapped to the base address 90C00h. The least significant byte of each register is mapped on 00000h boundaries, while the most significant byte is mapped on 00008h boundaries. The individual register addresses are derived by adding a multiple of 00010h to the base address. Table 3-4 lists the addresses of all accessable registers.

Table 3-4. TMS34061 Internal Register Addressing

REGISTER NAME	PC MEMORY BASE ADDRESS LSB	PC MEMORY BASE ADDRESS MSB	MEMORY ADDRESS A8	A7	A6	A5	A4	A3
			TMS34061 ADDRESS† CA6	CA5	CA4	CA3	CA2	CA1
Horizontal end sync	90C00h	90C08h	0	0	0	0	0	X
Horizontal end blank	90C10h	90C18h	0	0	0	0	1	X
Horizontal start blank	90C20h	90C28h	0	0	0	1	0	X
Horizontal total	90C30h	90C38h	0	0	0	1	1	X
Vertical end sync	90C40h	90C48h	0	0	1	0	0	X
Vertical end blank	90C50h	90C58h	0	0	1	0	1	X
Vertical start blank	90C60h	90C68h	0	0	1	1	0	X
Vertical total	90C70h	90C78h	0	0	1	1	1	X
Display update	90C80h	90C88h	0	1	0	0	0	X
Display start	90C90h	90C98h	0	1	0	0	1	X
Vertical interrupt	90CA0h	90CA8h	0	1	0	1	0	X
Control register 1	90CB0h	90CB8h	0	1	0	1	1	X
Control register 2	90CC0h	90CC8h	0	1	1	0	0	X
Status register	90CD0h	90CD8h	0	1	1	0	1	X
X-Y offset register	90CE0h	90CE8h	0	1	1	1	0	X
X-Y address register	90CF0h	90CF8h	0	1	1	1	1	X
Display address register	90D00h	90D08h	1	0	0	0	0	X
Vertical count register	90D10h	90D18h	1	0	0	0	1	X

† In CA1, when X = 0 the least significant byte is accessed.
When X = 1 the most significant byte is accessed.

Figure 3-3 shows how the register select address lines are mapped into a particular PC memory address. Address lines A19 - A9 are used for address decode, and A8 - A3 are used to select one of the eighteen internal registers in the TMS34061, shown in Table 3-4.

PC MEMORY ADDRESS															
A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
1	0	0	1	0	0	0	0	1	1	0	CA6	CA5	CA4	CA3	CA2	CA1	0	0	0

Figure 3-3. Register Select Address Mapping to Address

3.5 X-Y Indirect Accesses

Host initiated X-Y indirect memory cycles are memory-mapped at base address 90E00h. The 16 associated adjust functions of the X-Y registers after an X-Y indirect cycle are selected by the address formed by the base address plus a multiple of 00008h. Table 3-5 shows the addresses for the 16 functions of the X-Y standard indirect byte cycle.

Table 3-5. PC Memory Addressing for X-Y Indirect Access

PC MEMORY ADDRESS	VSC X-Y INDIRECT BYTE ACCESSES FUNCTION MODIFICATION	MEMORY ADDRESS A6	A5	A4	A3
		TMS34061 ADDRESS CA4	CA3	CA2	CA1
90E00h	No adjustment	0	0	0	0
90E08h	Increment X	0	0	0	1
90E10h	Decrement X	0	0	1	0
90E18h	Clear X	0	0	1	1
90E20h	Increment Y	0	1	0	0
90E28h	Increment X, Increment Y	0	1	0	1
90E30h	Decrement X, Increment Y	0	1	1	0
90E38h	Clear X, Increment Y	0	1	1	1
90E40h	Decrement Y	1	0	0	0
90E48h	Decrement Y, Increment X	1	0	0	1
90E50h	Decrement Y, Decrement X	1	0	1	0
90E58h	Decrement Y, Clear X	1	0	1	1
90E60h	Clear Y	1	1	0	0
90E68h	Clear Y, Increment X	1	1	0	1
90E70h	Clear Y, Decrement X	1	1	1	0
90E78h	Clear Y, Clear X	1	1	1	1

External hardware has been added in the CAS lines to allow X-Y indirect access to nibbles (4 bits) and double words (32 bits), in conjunction with bytes to be accessed. This has been achieved by mapping the X-Y addresses into several address spaces. Figure 3-4 shows how the X-Y functions modifications are mapped into a PC memory address. The upper address lines A19 - A7 are used for address decode. Address lines A6 - A3 are used to select one of the sixteen X-Y modifications. Address lines A1 and A0 are used to select no access, nibble access, or byte access.

PC MEMORY ADDRESS															
A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
1	0	0	1	0	0	0	0	1	1	1	0	0	CA4	CA3	CA2	CA1	0	S1	S0

CAx = TMS34061's column address line

S1	S0	
0	0	X-Y byte access
0	1	X-Y left nibble access D0 - D3
1	0	X-Y right nibble access D4 - D7
1	1	X-Y no access, only changes pointer

Figure 3-4. X-Y, None, Nibble, or Byte Function Mapping to PC Address

Figure 3-5 shows the mapping for X-Y two word accesses. This is for write functions only. With this mode all 32 memories can be written to at once with address lines A15 - A8 used as a write mask. Figure 3-6 shows the correlation between the mask on an X-Y double word access and the pixel position that is affected. Note that the actual pixels modified on this access come from the TMS34061's internal X-Y registers.

PC MEMORY ADDRESS															
A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
1	0	1	0	M0	M1	M2	M3	M4	M5	M6	M7	0	CA4	CA3	CA2	CA1	X	X	X

CAx = TMS34061's column address line

Figure 3-5. X-Y, Two-Word Function Mapping to PC Address

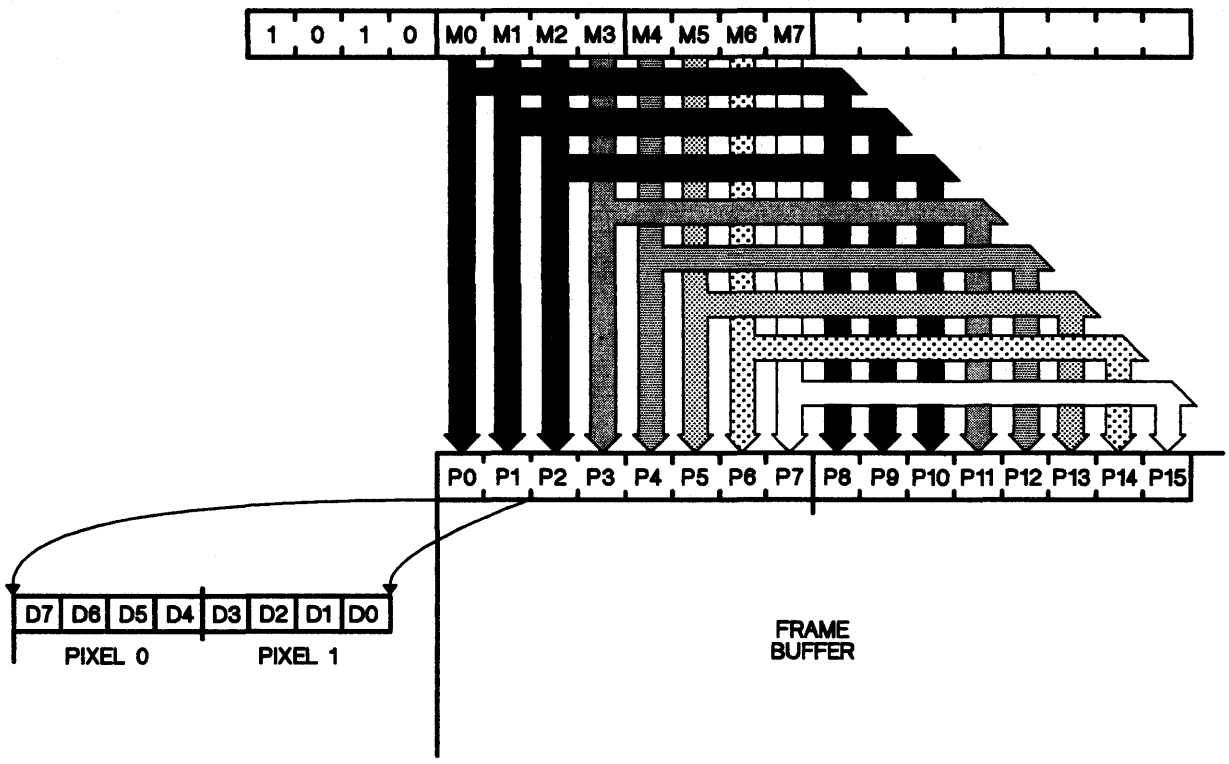


Figure 3-6. X-Y Double Word Mask to Screen Correlation

3.6 Shift Register Transfer Cycles

Since the multiport memories are equipped with a 256-bit internal shift register, the TMS34061 supports a shift register transfer from the memory to the shift register and from the shift register to the memory array. The shift register transfer cycles are memory-mapped starting at base addresses 90400h (shift register to memory) and 90800h (memory to shift register). An access to 90400h transfers the contents of the shift registers in bank 0 to row 000 of the memory array. An access to 90800h will transfer the contents of row 000 of the video memories in bank 0 to their internal registers. Likewise, an access to the other 255 rows or the other three banks of memories can be done by adding a displacement to the base address. Figure 3-7 illustrates how the row and bank addresses are formed for a shift register transfer cycle.

A A A A 1 1 1 1 9 8 7 6	A A A A 1 1 1 1 5 4 3 2	A A A A 1 1 9 8 1 0	A A A A 7 6 5 4	A A A A 3 2 1 0
1 0 0 1	0 0 0 0	X1 X0 R7 R6	R5 R4 R3 R2	R1 R0 B1 B0

X1	X0	
0	1	Shift register to memory
1	0	Memory to shift register

R7 - R0 row address select

B1	B0	
0	0	Bank 0 select
0	1	Bank 1 select
1	0	Bank 2 select
1	1	Bank 3 select

Figure 3-7. Shift Register Transfer Addressing

3.7 Interrupts

The Color Graphics Controller Board has a jumper-configurable interrupt, which can be set to either interrupt level 2 for the TI PC or for interrupt level 3 for the IBM PC. Figure 3-8 shows the how to configure the interrupt jumper for both of these machines.

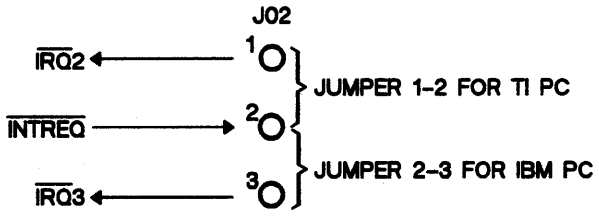


Figure 3-8. Interrupt Jumper J02

3.8 Expansion Bus

The Color Graphics Controller Board fits into the TI PC or IBM PC expansion bus, and into the expansion connector of most IBM-compatible machines. The signals that the board uses are shown in Table 3-6. Note that the only difference between the IBM PC and the TI PC for the Color Graphics Controller Board is the interrupt pins. This difference is handled by a jumper option, as described in Section 3.7.

Table 3-6. IBM/TI PC Bus Pin Assignments

SIGNAL NAME	IBM PIN NO.	TI PIN NO.	SIGNAL NAME	IBM PIN NO.	TI PIN NO.
Ground	B1,B10,B31	B1,B10,B31	RESET	B2	B2
+5 Volts	B3,B29	B3,B29	CLOCK	B20	B20
			OSC	B30	B30
$\overline{\text{MEMW}}$	B11	B11	$\overline{\text{RDY}}$	A10	A10
$\overline{\text{MEMR}}$	B12	B12	$\overline{\text{AEN}}$	A11	A11
$\overline{\text{IRQ2}}$	N/A	B24	$\overline{\text{IRQ3}}$	B25	N/A
D0	A9	A9	D1	A8	A8
D2	A7	A7	D3	A6	A6
D4	A5	A5	D5	A4	A4
D6	A3	A3	D7	A2	A2
A0	A31	A31	A1	A30	A30
A2	A29	A29	A3	A28	A28
A4	A27	A27	A5	A26	A26
A6	A25	A25	A7	A24	A24
A8	A23	A23	A9	A22	A22
A10	A21	A21	A11	A20	A20
A12	A19	A19	A13	A18	A18
A14	A17	A17	A15	A16	A16
A16	A15	A15	A17	A14	A14
A18	A13	A13	A19	A12	A12

3.9 Video Output Connectors

The Color Graphics Controller Board has two DB9 video output connectors, P2 and P3. Connector P2 outputs a digital RGBI signal and connector P3 outputs an RGB signal at 1 volt peak to peak with 75-ohm drive capability. Both connectors have positive and negative horizontal syncs and composite sync so that you can interface to a variety of monitors. The P2 pinout supports a TI or an IBM Personal Computer color graphics display digital color monitor with no cable modification, and the P3 pinout supports the IBM professional graphics display analog color monitor with no cable change. Other monitors can be interfaced easily by making your own cable.

PORT P2 PINOUT	
PIN NUMBER	SIGNAL DESCRIPTION
1	Ground
2	Ground
3	Red
4	Green
5	Blue
6	I or Ground†
7	Reserved
8	HSYNC†
9	VSYNC†

PORT P3 PINOUT	
PIN NUMBER	SIGNAL DESCRIPTION
1	Red
2	Green
3	Blue
4	CSYNC†
5	MODE†
6	Ground
7	Ground
8	Ground
9	Ground

† Jumper Options: See Table 3-1 for different options.

3.10 Power Consumption

The Color Graphics Controller Board uses only the +5 volt supply. The average current consumption is 1.2 A.

3.11 Sample Jumper Configurations

The following are sample jumper configurations for various host PCs and monitors. The Color Graphics Controller Board is shipped with software to support the IBM Professional Graphics Display. Software is also available to support 720 x 300 resolution like that of the TI Professional Computer and 640 x 200 resolution like that of the IBM Personal Computer color display.

HOST: IBM PC
MONITOR: IBM Professional Graphics Display
OUTPUT PORT: P3
SOFTWARE: TMDS3440879202
JUMPERS:

J01	1 to 2
J02	2 to 3
J03	2 to 3 and 4 to 5
J04	2 to 5
J05	Not Connected
J06	Not Connected

HOST: IBM PC
MONITOR: IBM Personal Computer Color Display
OUTPUT PORT: P2
SOFTWARE: TMDS3440879102†
JUMPERS:

J01	2 to 3
J02	2 to 3
J03	1 to 2
J04	1 to 2
J05	1 to 2
J06	Not Connected

HOST: IBM PC
MONITOR: TI Digital Monitor
OUTPUT PORT: P2
SOFTWARE: TMDS3440879002‡
JUMPERS:

J01	1 to 2, change oscillator to 18.432 MHz
J02	2 to 3
J03	1 to 4 and 2 to 3
J04	1 to 2
J05	2 to 3
J06	Not Connected

HOST: TI PC
MONITOR: IBM Professional Graphics Display
OUTPUT PORT: P3
SOFTWARE: TMDS3440879202
JUMPERS:

J01	1 to 2
J02	1 to 2
J03	2 to 3 and 4 to 5
J04	2 to 5
J05	Not Connected
J06	Not Connected

HOST: TI PC
MONITOR: TI Digital Monitor
OUTPUT PORT: P2
SOFTWARE: TMDS3440879002‡
JUMPERS:

J01	1 to 2, change oscillator to 18.432 MHz
J02	1 to 2
J03	1 to 4 and 2 to 3
J04	1 to 2
J05	2 to 3
J06	Not Connected

† Supports 640 x 200 display
‡ Supports 720 x 300 display
Contact your local sales office to order this software.

For systems using 512K bytes of system memory, connect jumper J06 1 to 2. This disables host direct accesses to the frame buffer; therefore, only the X-Y mode can be used to read or write to the frame buffer.

4. Theory of Operation

The following sections explain the theory of operation of the Color Graphics Controller Board. The theory of operation is divided into four parts: the PC bus to TMS34061 interface, the CPU to palette interface, the TMS34061 to frame buffer interface, and the frame buffer to video ouput interface.

4.1 PC Bus to TMS34061 Interface

The heart of the interface between the PC bus and the TMS34061 is PALs U48 and U40. The equations for these PALS are shown in Appendix B, with a brief explanation of their function.

The main control signals on the TMS34061 are $\overline{CEL}$, $\overline{CEH}$, SYSCLK, $\overline{ALE}$, R/$\overline{W}$, and RDY/$\overline{HOLD}$. The TMS34061's SYSCLK is connected to the PC's CLK signal from the expansion bus. The TMS34061 is reset when the PC is reset since the TMS34061's reset is generated from the PC's expansion bus reset. The $\overline{CS}$ line has been grounded so the TMS34061 is always active. The $\overline{ALE}$ determines when a CPU request will be generated.

When the host CPU executes a memory access to the memory range in which the Color Graphics Controller Board is mapped, the following actions take place. PAL U48 decodes the address and sends the appropriate FS0 - FS2 code to the TMS34061 (Table 3-2 shows the FS codes generated as a function of the memory address location). PAL U48 also sends a two bit binary code (CASOPT0 and CASOPT1) to PAL U40 and PAL U45. The information encoded on these lines is shown in Table 4-1.

Finally PAL U48 remaps the memory so that when the system auto-sizes the operating system starts at the bottom of memory (00000h) and looks for the first non-contiguous block. After the hole is found it loads the operating system in the highest section of the contiguous section of memory as possible. To prevent the operating system from being loaded into the Color Graphics Controller Board frame buffer the location between 40000h and 50000h cannot be used.

Since the memory now starts at 50000h the upper most address lines need to be remapped so that it is on a binary value. Figure 4-1 shows the necessary memory system memory map and the Color Graphics Controller Board memory remap.

Table 4-1. CAS Option Decode

CASOPT1	CASOPT0	TYPE OF CYCLE
0	0	X-Y double word access
0	1	X-Y access
1	0	Palette access
1	1	Other

SYSTEM ADDRESS					TMS34061 RA7 AND RA6 ADDRESS		
	A19	A18	A17	A16	A17PAL	A16PAL	
00000h	0	0	0	0	X	X	System RAM
10000h	0	0	0	1	X	X	System RAM
20000h	0	0	1	0	X	X	System RAM
30000h	0	0	1	1	X	X	System RAM
40000h	0	1	0	0	X	X	Empty†
50000h	0	1	0	1	0	0	CGC frame buffer†
60000h	0	1	1	0	0	1	CGC frame buffer†
70000h	0	1	1	1	1	0	CGC frame buffer†
80000h	1	0	0	0	1	1	CGC frame buffer†
90000h	1	0	0	1	X	X	CGC registers
A0000h	1	0	1	0	X	X	CGC XY double word
B0000h	1	0	1	1	X	X	Available
C0000h	1	1	0	0	X	X	System or user
D0000h	1	1	0	1	X	X	System or user
E0000h	1	1	1	0	X	X	System or user
F0000h	1	1	1	1	X	X	System or user

† This section of memory can be reclaimed for system use by installing J06.

Figure 4-1. System Address Remapping

After U48 decodes the address, one or more of the FS0 - FS2 lines will be active low if the Color Graphics Controller Board is selected. If all of the FS lines are high then the Color Graphics Controller Board is not selected and no host operation will take place. When PAL U40 receives a low input on one of the FS lines, it qualifies the input with $\overline{AEN}$ from the PC bus and waits for memory read ($\overline{MEMR}$) or memory write (MEMW) to go active low.

When either of these signals go active low PAL U40 generates the $\overline{ALE}$ and $\overline{CEL}$ signals, which start the TMS34061 on the host operation defined by the FS lines. The TMS34061 R/$\overline{W}$ line is supplied by the PC bus $\overline{MEMW}$ line. When $\overline{ALE}$ falls the TMS34061 will respond to the PC bus by dropping its $\overline{RDY}$ line, which will cause the PC's CPU to wait until the cycle has completed.

PAL U40 also takes in the CASOPT0 and CASOPT1 lines and in conjunction with the two least significant address lines, A1 and A0, it develops the control lines CASCTL0, CASCTL1, and CASCTL2. These three control lines are used during X-Y address cycles to control the CAS lines, thus allowing software address control over which memories are written to. This allows a performance increase in executing algorithms by reducing masking operations and allowing multiple memories to be written to selectively. This technique will be explained in more detail in Section 4.3. Table 4-2 describes the operations encoded on the CASCTL0 - CASCTL2 lines as a function of CASOPT1, CASOPT2, A1, and A0.

Table 4-2. Operations Encoded o

CASOPT1	CASOPT0	A1
0	0	X
0	1	0
0	1	0
0	1	1
0	1	1
1	1	X

Figure 4-2 shows the PC bus t
TMS34061.

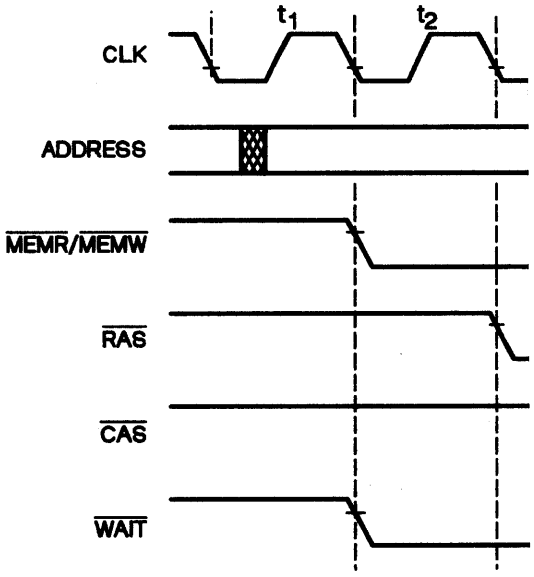


Figure 4-2. Memory R

SYSTEM ADDRESS					TMS34061 RA7 AND RA6 ADDRESS		
	A19	A18	A17	A16	A17PAL	A16PAL	
00000h	0	0	0	0	X	X	System RAM
10000h	0	0	0	1	X	X	System RAM
20000h	0	0	1	0	X	X	System RAM
30000h	0	0	1	1	X	X	System RAM
40000h	0	1	0	0	X	X	Empty†
50000h	0	1	0	1	0	0	CGC frame buffer†
60000h	0	1	1	0	0	1	CGC frame buffer†
70000h	0	1	1	1	1	0	CGC frame buffer†
80000h	1	0	0	0	1	1	CGC frame buffer†
90000h	1	0	0	1	X	X	CGC registers
A0000h	1	0	1	0	X	X	CGC XY double word
B0000h	1	0	1	1	X	X	Available
C0000h	1	1	0	0	X	X	System or user
D0000h	1	1	0	1	X	X	System or user
E0000h	1	1	1	0	X	X	System or user
F0000h	1	1	1	1	X	X	System or user

† This section of memory can be reclaimed for system use by installing J06.

Figure 4-1. System Address Remapping

After U48 decodes the address, one or more of the FS0 - FS2 lines will be active low if the Color Graphics Controller Board is selected. If all of the FS lines are high then the Color Graphics Controller Board is not selected and no host operation will take place. When PAL U40 receives a low input on one of the FS lines, it qualifies the input with $\overline{AEN}$ from the PC bus and waits for memory read ($\overline{MEMR}$) or memory write (MEMW) to go active low.

When either of these signals go active low PAL U40 generates the $\overline{ALE}$ and $\overline{CEL}$ signals, which start the TMS34061 on the host operation defined by the FS lines. The TMS34061 R/$\overline{W}$ line is supplied by the PC bus $\overline{MEMW}$ line. When $\overline{ALE}$ falls the TMS34061 will respond to the PC bus by dropping its $\overline{RDY}$ line, which will cause the PC's CPU to wait until the cycle has completed.

PAL U40 also takes in the CASOPT0 and CASOPT1 lines and in conjunction with the two least significant address lines, A1 and A0, it develops the control lines CASCTL0, CASCTL1, and CASCTL2. These three control lines are used during X-Y address cycles to control the CAS lines, thus allowing software address control over which memories are written to. This allows a performance increase in executing algorithms by reducing masking operations and allowing multiple memories to be written to selectively. This technique will be explained in more detail in Section 4.3. Table 4-2 describes the operations encoded on the CASCTL0 - CASCTL2 lines as a function of CASOPT1, CASOPT2, A1, and A0.

Table 4-2. Operations Encoded on Control Lines CASCTL0 - CASCTL2

CASOPT1	CASOPT0	A1	A0	CASCTL 2	CASCTL 1	CASCTL 0	OPERATION
0	0	X	X	0	1	1	X-Y Double word
0	1	0	0	1	0	0	X-Y Byte Access
0	1	0	1	1	1	0	X-Y Access Left nibble
0	1	1	0	1	0	1	X-Y Access Right nibble
0	1	1	1	1	1	1	X-Y No Access
1	1	X	X	0	0	0	0 Host Direct

Figure 4-2 shows the PC bus timing for a memory read and memory write to the TMS34061.

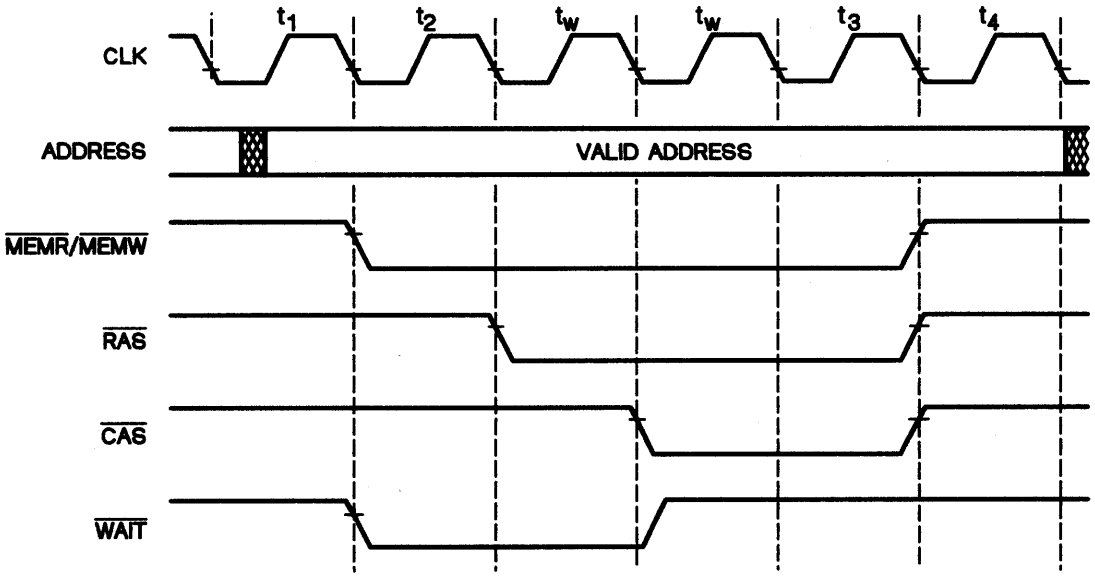


Figure 4-2. Memory Read/Write PC Bus Timing

4.2 CPU to Palette Interface

The TMS34070 Color Palette loads directly from the frame buffer. The blanking signal from the TMS34061 is aligned by two flip-flops to form the $\overline{\text{DATAEN}}$ signal on the palette, which controls the palette loading. The Mode pin on the palette defines if the palette loads at the beginning of a frame (frame load mode), on every line (line load mode), or if it is transparent (no load). The Mode pin has been designed on the Color Graphics Controller Board to be controlled by software program control. The function has been placed in memory space at address 903F0h to 903F6h.

PAL U45 controls the operation of the palette by decoding the memory address. There are two latches programmed into U45, P0 and P1. These two latches determine the state of the palette Mode pin. Table 4-3 shows the relationship between these latches, the memory locations which control them, and the Mode pin on the palette.

Table 4-3. Latching the Color Palette Modes

PALETTE MODE	SIGNALS SET		READ ADDRESS
	P0	P1	
Frame Load Mode	0	0	903F0h and 903F2h
Line Load Mode	0	1	903F0h and 903F6h
High (No Load)	1	0	903F4h and 903F2h
Reserved	1	1	903F4h and 903F6h

4.3 TMS34061 to Frame Buffer Interface

The TMS34061 makes designing a frame buffer extremely simple since it directly interfaces to the TMS4161. The maximum size frame buffer chosen on the Color Graphics Controller Board is 1024 by 512 pixels with four bits per pixel. This requires 32 TMS4161 memories. The memories are organized in four banks of eight memories each, and are sectioned off by the four RAS outputs, RAS0 - RAS3, on the TMS34061. These bits are controlled by the least significant address lines of the PC bus, which are connected to the TMS34061's RS0 and RS1 inputs respectively. Because every eighth pixel comes from the same memory, the TMS34061 and multiport memories operate at only one-eighth of the speed of the dotclock.

The address lines are multiplexed by the TMS34061 and are driven out on the MA0 - MA8 lines. The data bus goes from the PC bus directly to the frame buffer allowing any data bus width to be used. The $\overline{\text{CEL}}$ and $\overline{\text{CEH}}$ inputs on the TMS34061 control the $\overline{\text{CASHI}}$ and $\overline{\text{CASLO}}$ outputs of the TMS34061 to the frame buffer. The $\overline{\text{CEH}}$ input is not used because a trick has been played on the CAS inputs to the memories in X-Y mode so that an external PAL (U47) increases performance.

PAL U47 sections the memory into eight blocks of four memories. Therefore every pixel has its own CAS. The $\overline{\text{CASHI}}$ output determines when a shift register transfer is taking place so that all the CAS signals (CAS0 - CAS7) from U47 are driven low. On any cycle except an X-Y cycle, all the CAS signals are driven low when CASLO goes low. But on X-Y cycles the code on the CASCTL0 - CASCTL2 inputs to U47, along with the mask bits A8 - A15, determine which CAS0 - CAS7 output will go low when $\overline{\text{CASLO}}$ goes low. Table 4-4 shows the truth table for the logic in U47.

Table 4-4. PAL U47 Truth Table

MODE	CASHI	CASLO	CASCTL			CAS0	CAS1	CAS2	CAS3	CAS4	CAS5	CAS6	CAS7
			2	1	0								
Idle	1	1	X	X	X	1	1	1	1	1	1	1	1
SR Update	0	0	X	X	X	0	0	0	0	0	0	0	0
Host Direct	1	0	0	0	0	0	0	0	0	0	0	0	0
X-Y	1	0	1	0	0	0	0	0	0	0	0	0	0
X-Y Lnibble	1	0	1	1	0	1	0	1	0	1	0	1	0
X-Y Rnibble	1	0	1	0	1	0	1	0	1	0	1	0	1
X-Y No Access	1	0	1	1	1	1	1	1	1	1	1	1	1
X-Y Doubleword	1	0	0	1	1	$\overline{M0}$	$\overline{M1}$	$\overline{M2}$	$\overline{M3}$	$\overline{M4}$	$\overline{M5}$	$\overline{M6}$	$\overline{M7}$

Notes:
1. Lnibble = left nibble, D4 - D7
   Rnibble = right nibble, D0 - D3
2. Double word is all memories selected:
   M0 = A15 M4 = A11
   M1 = A14 M5 = A10
   M2 = A13 M6 = A9
   M3 = A12 M7 = A8

4.4 Frame Buffer to Video Output Interface

The CRT timing is generated from the TMS34061. The dot clock is input into the color palette and is divided by two, since the palette operates on two pixels at time because of its parallel architecture. The result is divided by four and fed into the TMS34061's VIDCLK input. The TMS4161s are clocked by two clock signals, SCLK1 and SCLK0. These clocks are generated by gating the TMS34061's BLANK output with VIDCLK. This allows the memory shift registers to shut off during blanking so that the shift registers can reload for the next line.

The outputs of the shift registers are input into four 74AS194 shift registers that are configured as eight 2-bit shift registers. These shift registers are reloaded every two pixels. The trick behind this method is wiring two memories together and alternatingly turning the serial outputs on. This is done by creating two output enable clocks, $\overline{SOE0}$ and $\overline{SOE1}$. An example timing diagram is shown in Figure 4-3.

There are eight bits output at a time because the palette operates on two 4-bit pixels at a time. For the digital output a 74AS157 is used to multiplex the 8-bit wide color bus back to four bits. The palette drives the analog 75 ohm impedance directly.

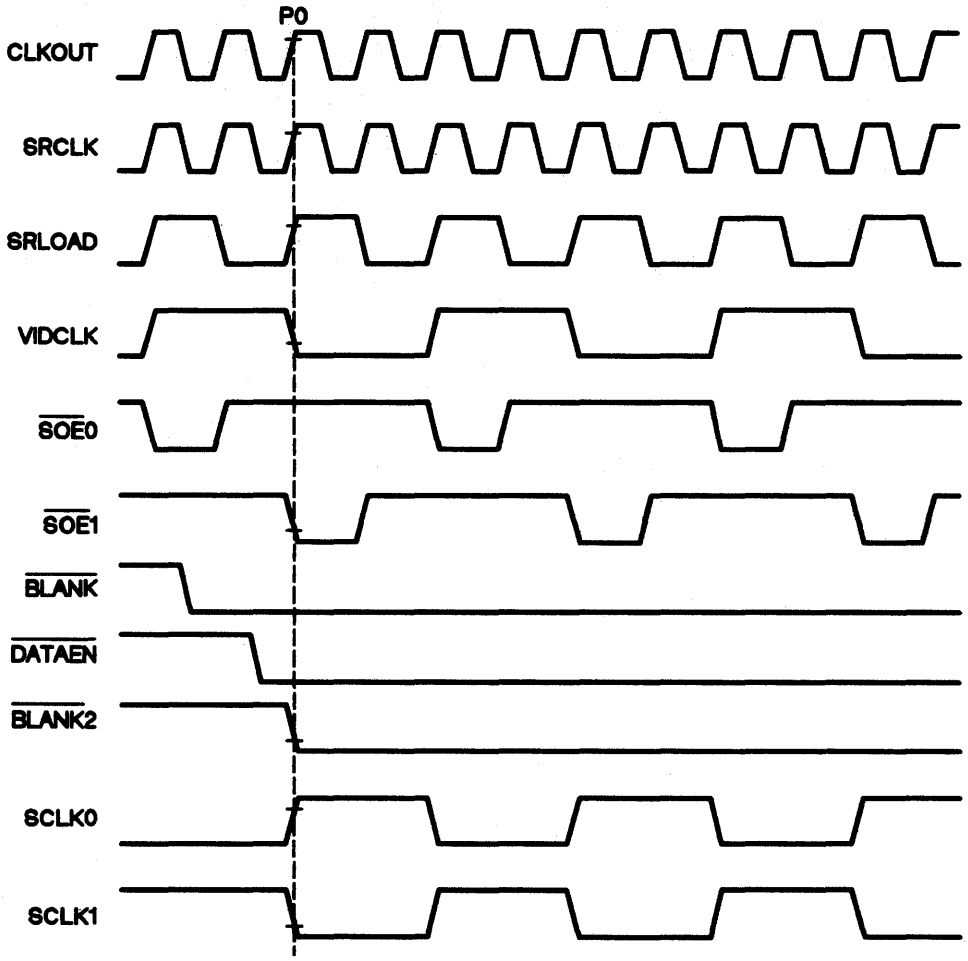


Figure 4-3. Frame Buffer to Video Output Interface Timing

5. Programming the CGC Board

This section discusses the following topics:

- Initializing the TMS34061 registers
- X-Y addressing
- Shift register control
- Horizontal and vertical scrolling
- Pixel replication zoom
- Vector drawing
- Clearing the screen
- Using the palette

5.1 Initializing the TMS34061 Registers

The first thing that needs to be done upon powerup of the Color Graphics Controller Board is to initialize the TMS34061's internal registers according to your current display requirements.

The following code example initializes the Color Graphics Controller Board to produce a display of 640 x 480 pixels; it assumes the display device connected to the board is an IBM Personal Computer Graphics Display or has similar specifications. Since no bulk clear of video RAM is performed here, the monitor will display whatever data happens to be in video RAM after initialization.

```
;-------------------------------------------------------;
;                                                       ;
;  Sample subroutine to Initialize TMS34061 registers:  ;
;                                                       ;
;-------------------------------------------------------;
;
;  TMS34061 Address Equates
;
VSCREG_BASE    EQU  9000H  ;segment register pointer
FIRST_REG      EQU  0C00H  ;address of first TMS34061 register
;
;  Initialize Extra Segment register to point to the segment
;  in which the TMS34061 registers are mapped.
;
               MOV  AX,VSCREG_BASE
               MOV  ES,AX
;
;  Point destination index [DI] to first TMS34061 register address
;  within segment [ES] and then point source index [SI] to
;  the first byte in the Register Initialization table. Set the loop count
;  CX to the number of table entries.
;
               MOV  DI,FIRST_REG            ;points to first TMS34061 reg.
               MOV  SI,OFFSET INIT_TBL      ;points to first data byte
;
;  Register initialization loop
;
               MOV  CX,26                   ;loop counter (bytes)
ILOOP:         MOV  BL,BYTE PTR CS:[SI]     ;read a byte from table
               MOV  BYTE PTR ES:[DI],BL     ;move it to TMS34061 register
               ADD  DI,8                    ;point DI to next register address
               INC  SI                      ;point to next table entry
               LOOP ILOOP                   ;dec CX, and continue till done
               RET                          ;return to caller
```

```
;-------------------------------------------------------;
;          TMS34061 register initialization table       ;
;                                                       ;
;  Register Values for 640 x 480 screen resolution      ;
;-------------------------------------------------------;
;
INIT_TBL      DB    03H     ;Horizontal End Sync     [LSB]
              DB    00H                              [MSB]
              DB    0CH     ;Horizontal End Blank
              DB    00H
              DB    5CH     ;Horizontal Start Blank
              DB    00H
              DB    64H     ;Horizontal Total
              DB    00H
              DB    01H     ;Vertical End Sync
              DB    00H
              DB    1DH     ;Vertical End Blank
              DB    00H
              DB    FDH     ;Vertical Start Blank
              DB    01H
              DB    FFH     ;Vertical Total
              DB    01H
              DB    02H     ;Display Update
              DB    00H
              DB    00H     ;Display Start
              DB    00H
              DB    00H     ;Vertical Interrupt
              DB    00H
              DB    00H     ;Control Register 1
              DB    10H
              DB    00H     ;Control Register 2
              DB    20H
```

5.2 X-Y Addressing

One of the TMS34061's most useful features is X-Y addressing mode. In brief, the programmer supplies the logical X-Y pixel location to the TMS34061, and the TMS34061 generates the proper address into the video RAM buffer.

Once the logical X-Y address has been loaded into the TMS34061, the programmer can perform any of ten operations on the logical address without reloading it into the TMS34061. These ten operations include the ability to increment and decrement the X-Y coordinates in any combination just by accessing a memory location.

5.2.1 Advantages of X-Y Addressing Over Conventional Addressing

Suppose you want to draw a horizontal line starting at coordinates 80,100 and continuing 20 pixels to the right. The first step is to load the X coordinate (80) and the Y coordinate (100) into the TMS34061. From that point on, whenever an operation is performed, the TMS34061 will generate the proper address, and after each subsequent read or write operation is finished, the TMS34061 will modify the address accordingly. In this example, the address that contains the pixel located at 80,100 is C828h. Normally, the programmer would have to calculate the address for every pixel, which would be a very time-consuming task. X-Y addressing simplifies this. Once the X-Y coordinate has been loaded into the TMS34061, you need only to write the proper data to the X-Y adjustment address that performs the function INCREMENT X, and the horizontal line is drawn.

The formula used to calculate the pixel address without using the TMS34061's X-Y addressing feature is:

PIXEL_ADDRESS = BASE_ADDRESS + (Y * 0200h) + (X/2)

On the Color Graphics Controller Board, the resulting PIXEL_ADDRESS can be a number greater than 16 bits, so the 8088's segment register would also have to be set everytime a new pixel address was calculated. But when using X-Y addressing there is no need to worry about the segment register, since the TMS34061 provides a 20-bit address into video RAM. This provides a tremendous advantage to the programmer as it is no longer necessary to keep track of the pointer into VRAM.

The following is a comparison of the code needed to draw a horizontal white line starting at 80,100 and continuing 20 pixels to the right. Note that the version not using X-Y addressing requires an extra instruction in the innermost draw loop, significantly increasing execution time.

```
;
;      Example using X-Y addressing
;
           MOV        AX,80       ;starting X coordinate
           MOV        DX,100      ;starting Y coordinate
           CALL       SETXY       ;load logical coordinates into TMS34061
           MOV        AL,0FFH     ;color code for 2 white pixels
           MOV        CX,10       ;each byte contains 2 pixels
DRAWLOOP:  MOV        ES:INCX,AL
           LOOP       DRAWLOOP
;
;      Example using conventional addressing
;
           MOV        DX,80       ;starting X coordinate
           MOV        AX,100      ;starting Y coordinate
           CALL       FINDXY      ;load logical coordinates into TMS34061
           MOV        AL,0FFH     ;color code for 2 white pixels
           MOV        CX,10       ;each byte contains 2 pixels
DRAWLOOP:  MOV        ES:[SI],AL
           INC        SI          ;extra instruction to increment pointer
           LOOP       DRAWLOOP
```

The next example is similar. This is a vertical white line starting at 80,100 and continuing 20 pixels down. Note the added computations when X-Y addressing is not used. The program must also keep track of whether a segment boundary has been crossed.

```
;
;     Example using X-Y addressing
;
          MOV       AX,80       ;starting X coordinate
          MOV       DX,100      ;starting Y coordinate
          CALL      SETXY       ;load logical coordinates into TMS34061
          MOV       AL,0FFH     ;color code for 2 white pixels
          MOV       CX,10       ;each byte contains 2 pixels
DRAWLOOP: MOV       ES:INCY,AL  ;draw and move down 1 line
          LOOP      DRAWLOOP
;
;     Example using conventional addressing
;
          MOV       DX,80       ;starting X coordinate
          MOV       AX,100      ;starting Y coordinate
          CALL      FINDXY      ;find address of X-Y coordinate
          MOV       AL,0FFH     ;color code for 2 white pixels
          MOV       CX,10       ;each byte contains 2 pixels
DRAWLOOP: MOV       ES:[SI],AL
          CLC
          ADC       SI,0200H    ;add offset for next vertical line
          JNC       SAMESEG     ;if still in same segment, cont. normally
          PUSH      AX          ;else: save color data
          MOV       AX,ES       ;  get current segment
          ADD       AX,1000H    ;  change to next segment
          MOV       ES,AX       ;  write it back
          POP       AX          ;  restore color data
SAMESEG:  LOOP      DRAWLOOP
```

The above examples clearly show the advantage of X-Y addressing over the more conventional appoach. Consider also that both examples call a subroutine to set the address that contains the desired pixel to be written, but only the conventional method requires a time-consuming multiply and several add instructions.

Also consider that the two examples draw very simple horizontal lines. If a slanted line was to be drawn, then you would have to resort to calling the pixel calculation routine for each new pixel, which would quickly become burdensome.

5.2.2 Setting Up the X-Y Address and Offset Registers

The following two code examples show the subroutine required to obtain the proper VRAM address given the logical X-Y coordinates. Note that in the second example there are two instructions that always load the X-Y offset register with the same values. This is to describe how many bits of X and Y are required in a system. Since the resolution of the board is always the same (1024 x 512) these two instructions do not change. Therefore they could be removed from the subroutine and executed only once when the system is powered up and the initial values are loaded into the TMS34061 registers.

```
;-------------------------------------------------------;
; Calculate Pixel Address in VRAM (conventional method) ;
;                                                       ;
;   This subroutine takes the specified logical X-Y     ;
;   coordinates and calculates the corresponding VRAM   ;
;   address. After execution, ES:[DI] will point to     ;
;   the proper pixel address.                           ;
;                                                       ;
; Inputs: AX = X                                        ;
;         DX = Y                                        ;
;Outputs: DI = Contains the address offset              ;
;         ES is set to the proper VRAM segment          ;
;-------------------------------------------------------;
; The range of the supplied X is 0-1023 (0-3FFH) which
; can be specified by 10 bits maximum.
;
; The range of the supplied Y is 0-511 (0-1FFH) which
; can be specified by 9 bits maximum
;
        PUSH    AX          ;Save X coordinate
        MOV     AX,DX       ;Get Y coordinate
        MOV     BX,0200H    ;Vline increment
        MUL     BX          ;Find Vertical start address
        MOV     CL,4
        XCHG    DL,DH
        SHL     DX,CL
        ADD     DX,VRAMBASE;add VRAM BASE ADDRESS
        POP     BX          ;restore X coordinate
        SHR     BX,1        ;find horizontal byte
        ADD     AX,BX       ;add to Vertical start address
        PUSH    AX          ;save offset address
        MOV     AX,DX       ;get segment
        MOV     ES,AX       ;set segment register
        POP     DI          ;set DI to the offset address
        RET
```

```
;-------------------------------------------------------;
; Set Pointer to Pixel in VRAM (using X-Y addressing)   ;
;                                                       ;
;   This subroutine takes the specified logical X-Y     ;
;   coordinates and loads the TMS34061's X-Y address    ;
;   and offset registers with the proper values.        ;
;                                                       ;
;  Inputs: DX = Logical X coordinate                    ;
;          AX = Logical Y coordinate                    ;
; Outputs: After execution, the TMS34061's 20-bit       ;
;          X-Y address pointer will point to the memory ;
;          location that contains the pixel specified   ;
;          by the logical coordinates.                  ;
;   Notes: ES is assumed to point to TMS34061 register  ;
;          segment                                      ;
;                                                       ;
;        MSB---------------------------------LSB       ;
; Offset reg 0 0 0 0 0 0 X X 0 0 1 0 0 0 0 0 0          ;
;           +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+           ;
;Address reg  Y (9-bit value)    X (7 MSBs)             ;
;           +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+           ;
;-------------------------------------------------------;
; The range of the supplied X is 0-1023 (0-3FFH) which
; can be specified by a maximum of 10 bits.
;
; The range of the supplied Y is 0-511 (0-1FFH) which
; can be specified by a maximum of 9 bits.
;
        SHR     DX,1            ;DX=9 MSBs of x1
        ROR     DX,1            ;DL=7 MSBs of x1
        ROR     DX,1            ;DH=2 LSBs of x1
        ROL     DH,1            ;Use 2 LSBs of DH
        ROL     DH,1            ; as bank selects
; note that the next two instructions describe the relationship between
; the X and Y resolution.
        MOV     BL,20H          ;XY addr increment
        MOV     ES:XYOFFSL,BL ;load low byte of X-Y offset register
        MOV     ES:XYOFFSH,DH ;load high byte of X-Y offset register
;
; calculate value for TMS34061's XY Address Register
;
        XCHG    AL,AH           ;swap y1 bytes
        ROR     AX,1            ;align y1 with x1
        OR      AL,DL           ;concat x1 & y1
        MOV     ES:XYADDRL,AL ;load XY
        MOV     ES:XYADDRH,AH ; Address Reg
        RET
```

5.3 Shift Register Control

Another unique and extremely useful feature of the TMS34061 is its ability to control the transfer of data to and from the TMS4161's internal shift register. The 65,536 bits of memory contained within the TMS4161 are arranged into rows. There are 256 rows each containing 256 bits each. Under control of the TMS34061 the TMS4161 memory can be manipulated in one of two ways. The first is the ability to transfer data contained in any of the 256 rows to the internal shift in a single instruction cycle. The second is the complementary ability to transfer the shift register data back to any of the 256 rows.

This direct control of the shift register results in extremely fast execution of several functions that normally require a lot of host processor time to execute. For example:

- Screen clear
- Screen fill

- Vertical scrolling (in either direction)
- Block memory moves and copies

Some of these functions are explained later in this document.

5.3.1 Memory Organization

Various decisions are necessary when designing a memory organization for your system. One of these decisions concerns how shift register transfers will operate. A common VRAM organization is such that a shift register transfer has the ability to move a horizontal scan line of data to another section of the screen. Depending on the needs of a particular system, you might choose to arrange VRAM such that a single shift register transfer will move more than one line of data.

For example, the board is arranged so that a shift register reload loads the shift registers of eight video RAMs. This means that every shift register reload loads enough data for two scan lines of the display. To illustrate this, consider that the horizontal resolution of the board is 1024 pixels, and the data contained in eight shift register reloads is 8x256. That calculates to 2048 pixels of data, or two scan lines of the display. Figure 5-1 shows the organization of the memory and external shift registers and the relationship to the screen display.

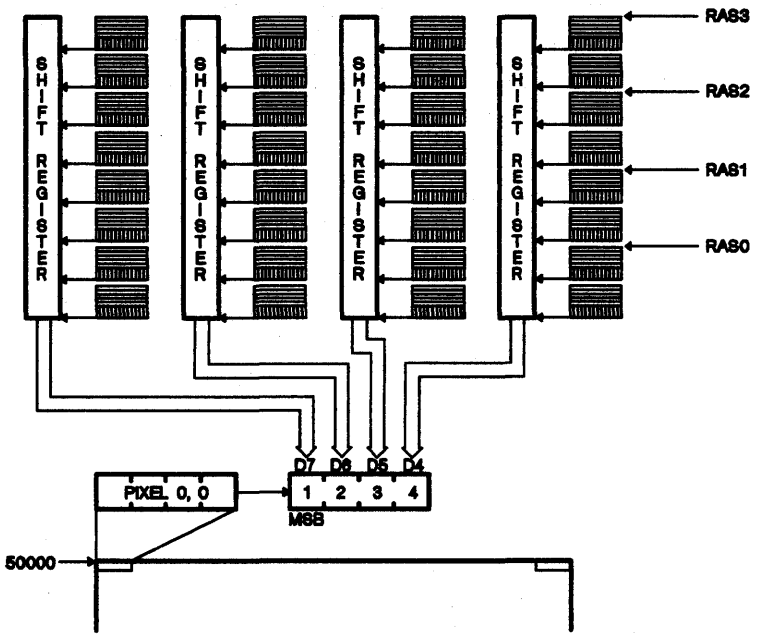


Figure 5-1. Memory and External Shift Register Organization

5.3.2 Initiating a Shift Register Transfer

The shift register to memory transfers are memory-mapped starting at address 90400h, and the memory to shift register transfers start at 90800h. As shown in Figure 5-2, the lower two bits of the address select the appropriate memory bank, the next eight bits select the row address within the TMS4161, and the next two bits determine the direction of the transfer.

Suppose you want to copy the data in Row 0 to Row 1. The first step would be to inhibit automatic generation of the shift register transfer addresses. This is done by setting bit 5 in Control Register 1 to a logical 1. Now you can control the addresses manually. The second step is referencing address 90800h. This will transfer Row 0 of memory to the TMS4161's internal shift register. The third step is to reference address 90404h, which moves the contents of the shift register to Row 1 of memory. The following is an example of the code required to perform this copy.

```
;
;     Turn display update off
;
          MOV       AL,020H     ;bit 5=true
          MOV       CNTRL1L,AL  ;set control reg-1
;
          MOV       SI,0800H    ;read address for Row 0
          MOV       DI,0404H    ;write address for Row 1
          MOV       DS:[SI],AX  ;Read Row 0 to shift-register
          MOV       DS:[DI],AX  ;Write shift-register to Row 1
;
;      Turn display update back on
;
          MOV       AL,0        ;bit 5=false
          MOV       CNTRL1L,AL  ;set control reg-1
```

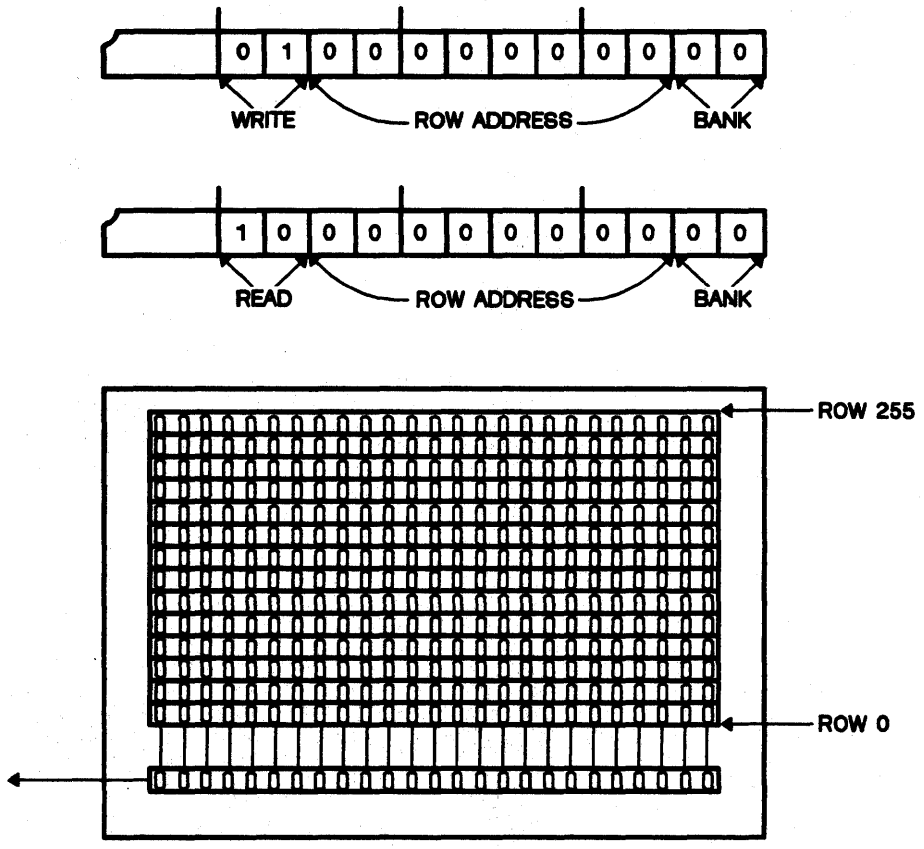


Figure 5-2. Shift Register Transfer Addressing

5.4 Horizontal Scrolling Using X-Y Addressing

The following is an example of how horizontal screen scrolling can be implemented using the TMS34061's X-Y addressing feature. The advantage of using this method is that after the initial size of the region to be scrolled has been determined, the region can be moved with a series of move instructions, and the program does not have to monitor and adjust memory pointers. Since the Color Graphics Controller Board memory is arranged so that one byte contains the information for two pixels, movement of data in done a byte at a time for speed. Figure 5-3 shows a representation of the steps involved when scrolling the three rightmost columns of pixels one column to the left.

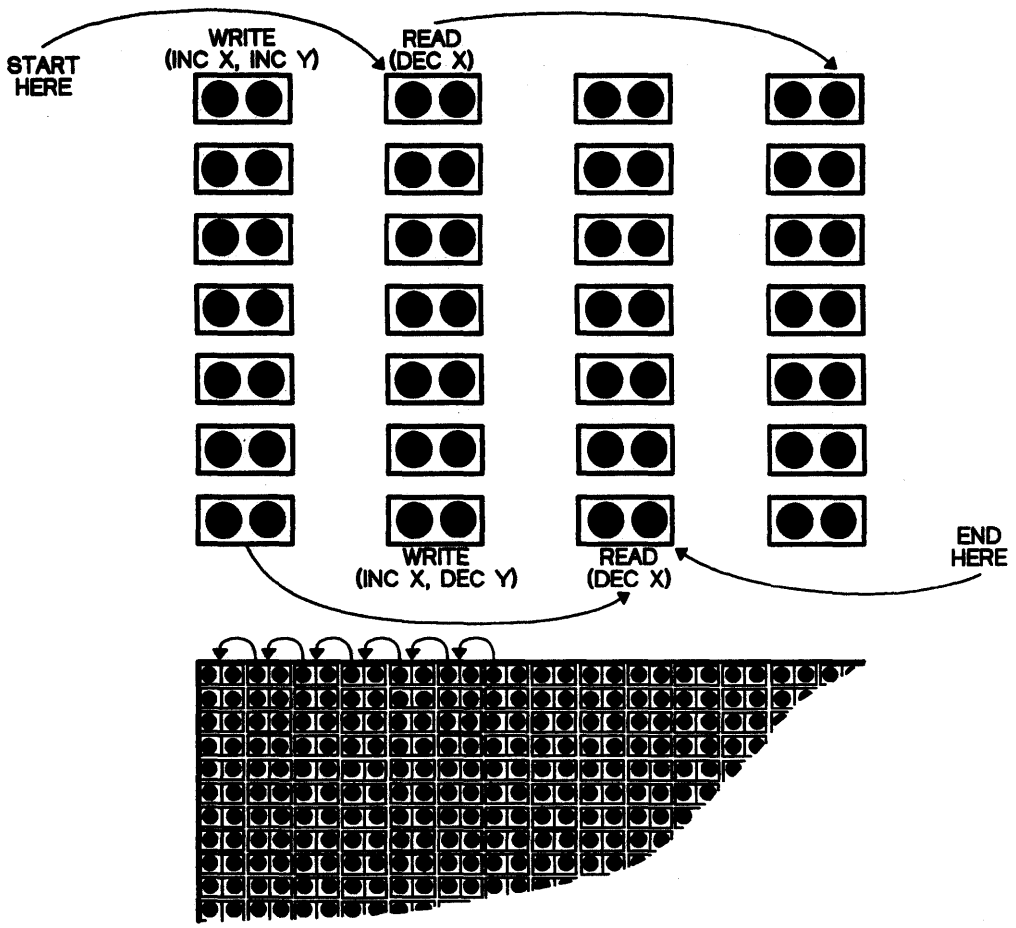


Figure 5-3. Horizontal Scrolling

The first step is to position the X-Y address so it points to the memory location which contains the upper left pixel in the region being scrolled. This position is designated in Figure 5-3 by the words "Start Here". The next step is to read the byte of data from the X-Y adjustment address that decrements X. This moves the pointer over one byte to the left. The data is ready to be written back by writing to the X-Y adjustment location that increments X and decrements Y. The result is that the first element of the first column has been moved to the left and the pointer is positioned such that the entire sequence can be repeated on the second element.

This operation is repeated until the desired number of elements have been moved. Then the pointer is moved to the next column by performing two "dummy" read operations from the X-Y adjustment location that increments X. The sequence is then repeated moving up, and over columns until the entire region has been moved. The following is an example of the code required to perform the scrolling.

```
;
;      This routine assumes that ES points to segment 9000H, which contains
;      the X-Y adjustment addresses. It also assumes the X-Y pointer has
;      been initialized to the upper-left hand corner of the region to
;      be scrolled.
;
          MOV     DX,Number_of_Columns      ;set column counter
;
NXTCLMN:  MOV     CX,Number_of_Rows         ;set row counter
NXTROWD:  MOV     AL,ES:DECX                ;read byte and move left
          MOV     ES:INCX_INCY,AL           ;write byte and move down and right
          LOOP    NXTROWD                   ;repeat until all rows done
;
          DEC     DX                        ;is this the last column?
          JE      ALLDONE                   ;if so, exit
;
          MOV     AL,ES:INCX                ;if not, move to next column
          MOV     AL,ES:INCX
;
          MOV     CX,Number_of_Rows         ;set row counter
NEXROWU:  MOV     AL,ES:DECX                ;read byte and move left
          MOV     ES:INCX_DECY,AL           ;write byte and move up and right
          LOOP    NEXROWU                   ;repeat until all rows done
;
          DEC     DX                        ;is this the last column
          JNE     NEXCLMN                   ;if not, repeat
;
ALLDONE:  RTS
```

5.5 Pixel Replication Zoom

The following is an example of how a region of the screen can be zoomed using the TMS34061's X-Y addressing feature. The advantage in using this method is the programmer no longer has to keep track of and adjust memory pointers. For the sake of simplicity the region to be zoomed is read using direct memory addressing, and then written back in its expanded form using X-Y addressing.

Figure 5-4 shows a repesentation of a 2X zoom. A single pixel of data is read from the source region, and written to the destination region using X-Y adjustment addresses. The following code implements the zoom.

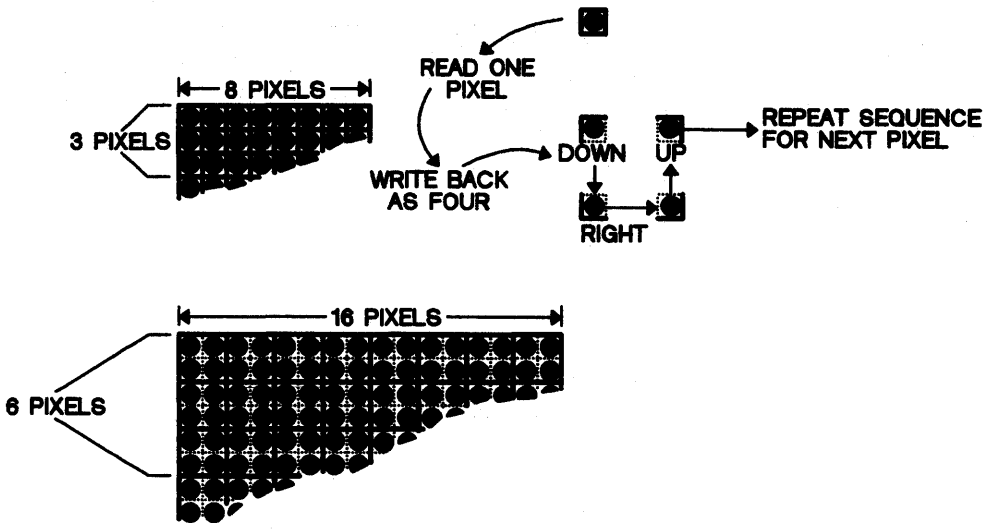


Figure 5-4. Example of a 2X Pixel Replication Zoom Using X-Y Addressing

```
;
;    This routine assumes the following:
;
;   -DS:SI points to the byte which contains the upper-left hand
;    pixel of the source region.
;   -ES is assumed to point to segment 9000H, which contains the TMS34061's
;    register functions.
;   -The logical X-Y coordinates of the upper left-hand corner of the
;    destination region have already been loaded into the TMS34061's X-Y
;    address register.
;
; zoom the even pixel first - this is the upper nibble of the source byte
;
ZOOMLINE: MOV       AL,DS:[SI]      ;get byte from source region
          MOV       AH,AL           ;replicate byte
          SHR       AH,1            ;justify in lower nibble
          SHR       AH,1
          SHR       AH,1
          SHR       AH,1
          AND       AL,0F0H         ;strip off old odd pixel data
          OR        AL,AH           ;duplicate even pixel in both nibbles
          MOV       ES:INCY,AL      ;write 2 pixels and move down
          MOV       ES:INXDCY,AL    ;write 2 pixels and move up and right
```

```
;
; zoom the odd pixel next, this is the lower nibble of the same source byte
;
          MOV       AL,DS:[SI]      ;get two pixels of data
          MOV       AH,AL           ;duplicate pixel
          SHL       AH,1            ;justify in upper nibble
          SHL       AH,1
          SHL       AH,1
          SHL       AH,1
          AND       AL,00FH         ;strip off old even pixel data
          OR        AL,AH           ;duplicate odd pixel in both nibbles
          MOV       ES:INCY,AL      ;write 2 pixels and move down
          MOV       ES:INXDCY,AL    ;write 2 pixels and move right and up
;
; increment the source address to the byte which contains the next two
; pixels we wish to zoom.
;
          INC       SI
          LOOP      ZOOMLINE        ;repeat until zoom complete
```

5.6 Vector Drawing Using X-Y Addressing

The following is an example of how a shape or font can be drawn using the TSM34061's X-Y addressing feature. A representation showing the steps involved in vector drawing are shown in Figure 5-5. The octogon shown at the bottom of the figure is the object being drawn. The graphic data used to describe the octogon is stored in a table as a series of X-Y adjustment addresses followed by repeat counts.

For example, if a horizontal line 20 pixels wide was to be drawn, the table would contain two words. The first word would be the X-Y adjustment address for increment X, and the second word would contain the repeat count. In the case of the octogon used in Figure 5-5, the data table contains 16 words of data. The program to display this data would first load the address from the table, and then load the repeat count. The next step would be to write the appropriate color value to the address the number of times specified by the loop count. After the loop was complete, the next address and repeat count would be loaded from the table and the procedure repeated. A simple way to terminate the table would be to set the address or loop count to zero.

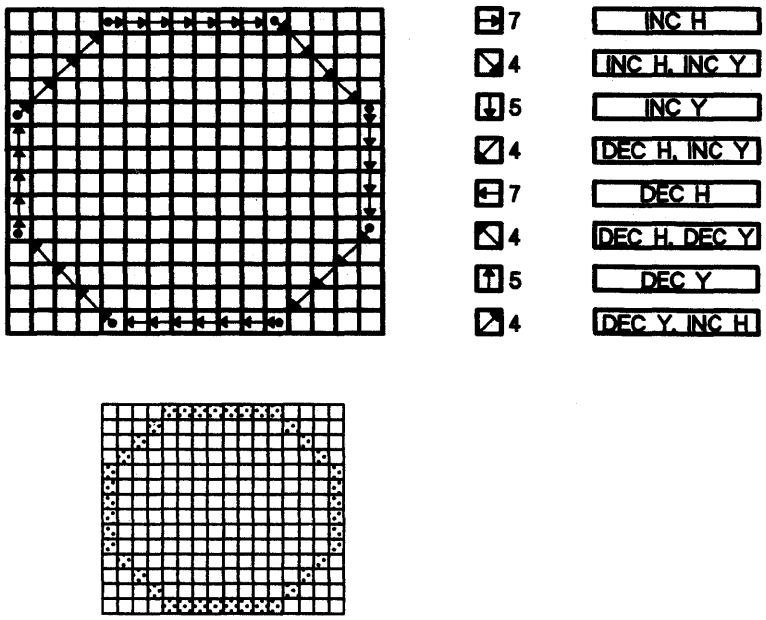


Figure 5-5. Vector Drawing Using X-Y Addressing

An interesting modification to this routine is storing the X-Y adjustment address as a number instead of the absolute address. Then a table of the adjustment addresses could be kept and the number used as an index to the appropriate address in that table. This modification lends itself well to rotating the figure at 45 degree increments. For example, a lookup table could be defined with its first entry as the absolute address for INCREMENT X, and the number 0 in the graphic data table used to represent it. That way a second lookup table could be defined whose first entry is INCREMENT X and INCREMENT Y. Then when 0 is used to look up the absolute address the vector would no longer be drawn horizontally (INCREMENT X). Instead it would be drawn at a 45 degree angle sloping downwards to the right (INCREMENT X, INCREMENT Y).

The following is a code example which draws the octogon shown in Figure 5-5. Even though the octogon doesn't use the MSB of the repeat count, setting it will result in the vector being drawn without actually writing to memory. This is useful for repositioning the X-Y pointer without actually plotting a new vector.

```
;
;     This routine assumes BL has already been loaded with the proper color
;     information, and the initial logical X-Y coordinates have been loaded
;     into the TMS34061 XY address register.
;
         MOV    SI,OFFSET GRTABLE  ;setup pointer to table start
;
;    get an entry from the table, which will later be used as our
;    destination address.
;
DRAW:    MOV    DI,CS:[SI] ;move first table entry to destination pointer
         CMP    DI,0       ;was the entry zero?
         JE     ALLDONE    ; if so, we are done.
         ADD    SI,2       ;increment pointer to next table address
;
         MOV    CX,CS:[SI] ;use repeat value from table as loop counter
         SHL    CH,1       ;is the upper bit set
         JC     READVEC    ; if so, just move without drawing
;
REPVEC:  MOV    ES:[DI],BL ;store color at X-Y adjustment address
         LOOP   REPVEC     ;repeat until vector is done
         ADD    SI,2       ;increment pointer to next table address
         JMP    DRAW       ;repeat process until entire shape is done
;
READVEC: MOV    AL,ES:[DI] ;dummy read at X-Y adjustment address
         LOOP   READVEC    ;repeat until vector is done
         ADD    SI,2       ;increment pointer to next table address
         JMP    DCHR       ;repeat process until entire shape is done
ALLDONE: RET
;=======================================================;
;  Graphic data table for octogon                       ;
;                                                       ;
;  WORD1: TMS34061 XY operation address (0000=terminate);
;  WORD2: Draw count (high bit set means don't plot)    ;
;                                                       ;
;=======================================================;
GRTABLE   DW  INC_X,0007H       ;move right 7 times
          DW  INC_X_INC_Y,0004H ;move right and down 4 times
          DW  INC_Y,0005H       ;move down 5 times
          DW  DEC_X_INC_Y,0004H ;move left and down 4 times
          DW  DEC_X,0007H       ;move left 7 times
          DW  DEC_X_DEC_Y,0004H ;move left and up 4 times
          DW  DEC_Y,0005H       ;move up 5 times
          DW  INC_X_DEC_Y,0004H ;move right and up 4 times
          DW  0000H
```

5.7 Clearing the Screen Using Shift Register Transfers

Manual control of shift register transfers lets you clear the screen very quickly. To perform this operation, the first 1024 bytes of memory are filled with the desired screen color (each scan line line consists of 512 bytes of RAM, therefore the first two scan lines are filled). Shift register writes to RAM are then executed to move this data to every line of the display. A step-by-step procedure and code example are given below that describe how to accomplish this task.

1) The display screen is blanked by setting bit 13 (B13) in Control Register 2 to a zero. It is a good idea to check for the vertical interrupt before initiating this so as not to disable the display in the middle of the active display region. Any manipulation of data in video RAM while the display is active can be perceived to some degree by the human eye and will usually result in a visible flash or glitch.

2) The shift register display update must be turned off; otherwise, the system will continue to move new information to the shift registers. This is done by setting bit 5 (B5) in Control Register 1 to a one.

3) RAS override is initiated by setting bits 2 thru 5 of Control Register 2 high. This will cause all four RAS lines to be active at the same time, thus allowing information written to one bank to be written simultaneously into all four banks.

4) The desired screen clear color is written into all 256 bits of ROW 0 in each of the four banks. On the board this is done by filling memory from 5000h to 53FFh. Remember that it takes a bit from each of the four banks to form the pixel. Therefore 256 x 4 (nibbles) is actually 128 x 8 (bytes). Decimal 128 is 80 in hex, and since each bank consists of eight RAMs, memory is from 5000h to 53FFh (80h multiplied by 8).

5) Row 0 is then transferred to the shift register. The shift register data should be transferred back to every row of RAM. Normally you would start with ROW 1 and increment to the next address until all 256 rows have been filled. Since RAS override has been initiated and all four RAS lines are active at the same time, only 256 rows need to be cleared, as the remaining RAMs will get the same access.

6) After the shift register data has been transferred to each of the 256 rows, RAS override is turned off, display update should be re-enabled, and the screen should be unblanked. Just as before, it is good practice to unblank the display during the vertical blanking interval, so as not to cause a visible glitch.

The following sample program follows the steps described above to fill video RAM with zeros, thus clearing the screen. The data contained in register AL is written to all RAM locations. The complete source code for this routine is contained on the TMS34061 demo disk included with the board.

```
;  DS is assumed to point to the segment which contains the TMS34061 register
;  addresses.
;
         MOV     AL,0                          ;value to fill RAM with
         CLD                                   ;set flag to inc SI
         MOV     DS:CNTRL1L,020H               ;inhibit display update
         MOV     DS:CNTRL2L,03CH               ;ras override
         MOV     DS:CNTRL2H,000H               ;blank display
;
;  fill first 1024 bytes of memory with desired color
;
         MOV     CX,256                        ;do 256 bits/rows
         MOV     DI,VscMemPtr                  ;point to memory
SRBLANK: STOSB                                 ;al → es:vsc memory
         ADD     DI,3                          ;di=di+3+1 - next addr
         LOOP    SRBLANK                       ;es:[di]
         MOV     SI,OFFSET SRLOAD              ;load sr - ds:[si]
         MOV     AL,[SI]                       ;row 000 → sr
         MOV     SI,OFFSET SRSTORE+4           ;row 001
;
;  Move shift register to each RAM row
;
         MOV     CX,255                        ;255 rows
SR0:     LODSB                                 ;sr → row ds:[si]
         ADD     SI,3                          ;si=si+3+1 - 255 rows
         LOOP    SR0
;
;  re-enable display and return to caller
;
         MOV     CNTRL1L,020H                  ;turn on display update
         MOV     CNTRL2L,000H                  ;set normal ras select
         MOV     CNTRL2H,020H                  ;enable display
         RET
```

5.8 Vertical Scrolling Using Shift Register Transfers

Manual control of shift register transfers gives you the ability to scroll the screen in a fast and smooth manner. Figure 5-6 shows the steps needed to scroll a region of the screen with wraparound. Remember that on the Color Graphic Controller Board every shift register transfer manipulates two scan lines of data. In Figure 5-6 the screen is being scrolled up. The first step is labeled with the number 1, and shows the movement of two lines (0-1) of data located at the top of the selected region to a temporary off-screen buffer. The second step shows lines 2-3 moving up when to where lines 0-1 used to be. The third step moves lines 4-5 up to 2-3. This is repeated until the last two lines of the selected region have been moved up. Then the data located in the off-screen buffer is shifted into the last line of the display. The region of the screen has now been scrolled two lines, with the data previously at the top of the region wrapped around to the bottom. To continue to scroll the screen the process is repeated. The following is a program example which scrolls each pair of lines up.

```
;
;     Turn display update off, set RAS override
;
          MOV       CNTRL1L,020H ;inhibit display update
          MOV       CNTRL2L,03CH ;ras override
;
          MOV       SI,0804H    ;read address for Lines 2-3
          MOV       DI,0400H    ;write address for Lines 0-1
MOVEUP:   MOV       DS:[SI],AX ;read lines to shift register
          MOV       DS:[DI],AX ;write shift register to lines
          ADD       SI,04       ;change read address to next pair of lines
          ADD       SI,04       ;change write address to next pair of lines
          DEC       CX          ;all 15 line pairs done?
          JNE       MOVEUP
;
;      Set normal RAS control and turn display update back on
;
          MOV       CNTRL1L,020H                ;turn on display update
          MOV       CNTRL2L,000H                ;set normal ras select
          RET
```

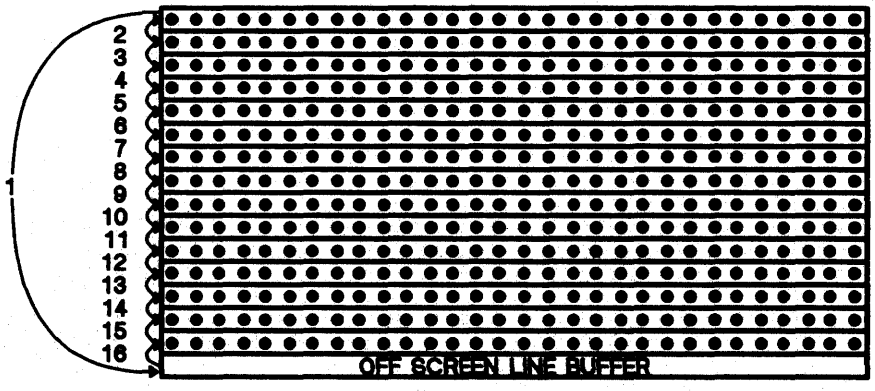


Figure 5-6. Vertical Scrolling Using Shift Register Transfers

5.9 Vertical Screen Flip

Manual control of shift register transfers lends itself well to various screen operations. One practical application is flipping the entire screen vertically. The steps involved are shown in Figure 5-7. Row 0 (lines 0-1) are moved to the shift register, and then written to the temporary off-screen buffer. The last two lines of the display are then moved to the top, and replaced by the temporary data in the buffer. This process is repeated moving inwards until the two innermost pairs of line on the screen have been exchanged.

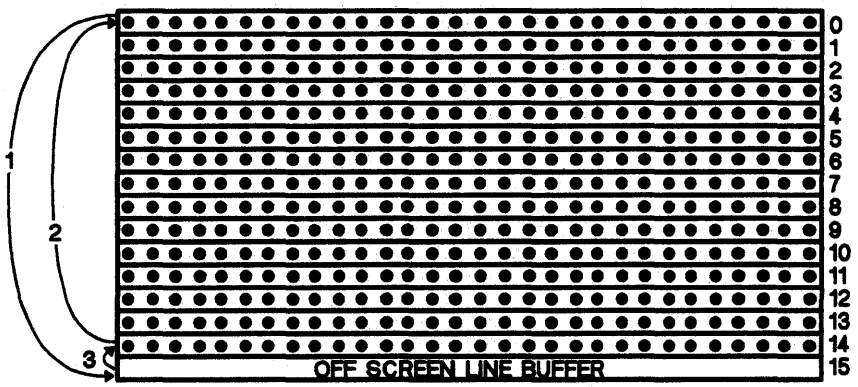


Figure 5-7. Vertical Screen Flip Using Shift Register Transfers

5.10 Using the Palette

Figure 5-8 shows how the palette is mapped in relationship to the Color Graphics Controller Board screen. The palette consists of 16 registers, each 16 bits long. These registers are loaded automatically from the first 16 words of display memory. Since each pixel on the Color Graphics Controller Board is represented by a nibble of data, the first 64 pixels are used by the palette. If the palette is in frame load mode, then only the first 64 pixels of Line 0 are used to load the registers, and the palette is only loaded once per frame. If the palette is in line load mode, then it is loaded with fresh data from each line of the display. This means the first 64 pixels (32 bytes) of data on every line is used by the palette.

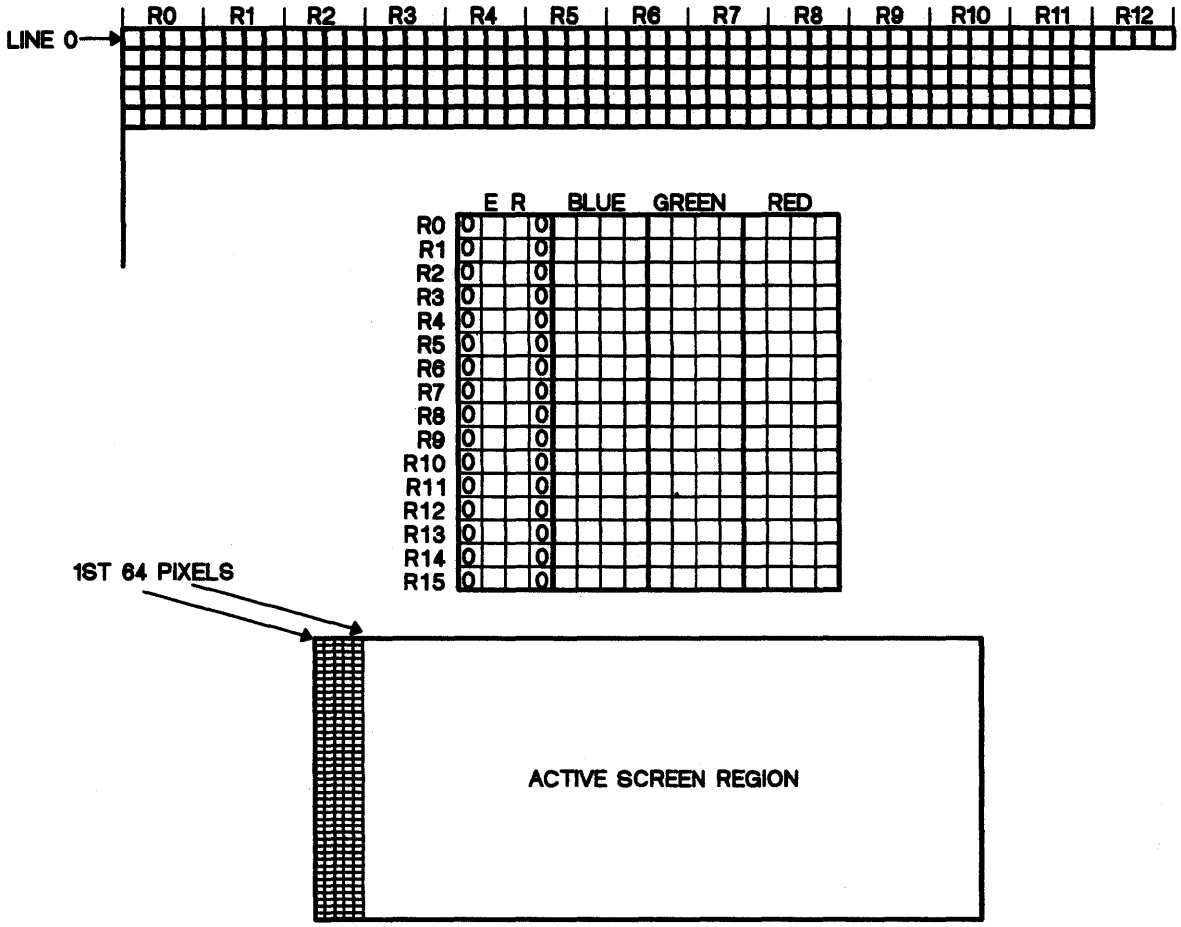


Figure 5-8. Palette to Screen Mapping

As an example, assume that the palette is configured in frame load mode, and uses the first 32 bytes of data at the beginning of line 0 to load its 16 internal registers. This means that, on the Color Graphics Controller Board, memory locations 50000h and 50001h define palette register 0, and 50002h and 50003h define register 1, and so forth. A pixel (nibble) in memory can be 0 to F hex. If a pixel is set to 0, then palette register 0 will be selected. If the pixel is set to 1, then palette register 1 is selected, and so forth. If the desired action was to display all nibbles containing the value 0 on the screen as the color black, palette register 0 must contain all zeros, which would mean setting memory bytes 50000h and 50001h to zeros. If the intent was to display all nibbles containing the value 1 as red, then palette register 1 would have to be loaded with the color code for red. Maximum intensity red would be defined by loading memory location 50002h with zeros, and 50003h with the value 0Fh. Nibbles containing the value 1 could then be displayed in maximum intensity green by changing memory location 50003h to F0h.

Line load mode is almost identical, except the values located in memory from 50000h to 5001Fh only define the palette register values to be displayed on line 0 of the display. Line 1 of the display would be defined by the first 32 bytes on that line,

50200h through 5021Fh in memory. Likewise the first 32 bytes on each line are used for the remaining lines of the display.

5.10.1 Initializing the Palette to Default Values

The first generation boards without the TMS34070 on board used only three bits of every nibble to drive the monitor. These bits directly drove the Red, Green, and Blue inputs of the TI or IBM monitor. Red was represented by the LSB of the nibble (1 hex), blue by 2 hex, and green by 4 hex. The MSB of the nibble representing one pixel was not used. The following is an example of how the palette needs to be initialized in order to duplicate these colors on an analog RGB monitor.

PIXEL		PALETTE	PALETTE REGISTERS			
BITS	COLOR	REGISTER	0000	BLUE	GREEN	RED
X000	BLACK	REG. 0	0000	0000	0000	0000
X001	RED	REG. 1	0000	0000	0000	1111
X010	GREEN	REG. 2	0000	0000	1111	0000
X011	YELLOW	REG. 3	0000	0000	1111	1111
X100	BLUE	REG. 4	0000	1111	0000	0000
X101	MAGENTA	REG. 5	0000	1111	0000	1111
X110	CYAN	REG. 6	0000	1111	1111	0000
X111	WHITE	REG. 7	0000	1111	1111	1111
X000	BLACK	REG. 8	0000	0000	0000	0000
X001	RED	REG. 9	0000	0000	0000	1111
X010	GREEN	REG. 10	0000	0000	1111	0000
X011	YELLOW	REG. 11	0000	0000	1111	1111
X100	BLUE	REG. 12	0000	1111	0000	0000
X101	MAGENTA	REG. 13	0000	1111	0000	1111
X110	CYAN	REG. 14	0000	1111	1111	0000
X111	WHITE	REG. 15	0000	1111	1111	1111

If the palette were initialized in frame mode, then these values would be loaded into memory starting at the beginning of line 0, at 50000h. A memory dump of the first 32 bytes would look like this:

```
ADDRESS DATA
------- -----
50000H: 0000H
50002H: 000FH
50004H: 00F0H
50006H: 00FFH
50008H: 0F00H
5000AH: 0F0FH
5000CH: 0FF0H
5000EH: 0FFFH
50010H: 0000H
50012H: 000FH
50014H: 00F0H
50016H: 00FFH
50018H: 0F00H
5001AH: 0F0FH
5001CH: 0FF0H
5001EH: 0FFFH
```

If the palette were initialized in line load mode, then these values would have to be loaded in the first 32 bytes of every line of the display.

5.10.2 Using Repeat Mode

Repeat mode is a simple function that can be used in a lot of interesting applications. Each of the palette's 16 color lookup registers has a repeat bit associated with it. The position of the repeat bit is shown in Figure 5-8 and is designated by "R". When the repeat bit is set, the palette ignores the color data in that register. For example, the repeat bit is set for palette register 0, and is not set for palette register 1. If the palette receives the color code 1, it will load the D-A outputs with the values located in register 1. If the next color code was 0, the palette would check to see if the repeat bit is set, and since it is, it would not load the D-A outputs with the values in register 0. Instead, it would just ignore the request and continue displaying the values it had previously loaded from register 1.

A. List of Materials

ITEM	PART	LOCATION	QUANTITY
1	74AS194 shift register	U33,U34,U35,U36	4
2	74AS00 quad NAND gate	U37	1
3	74AS04 hex inverter	U50	1
4	74AS74 dual D flip-flop	U39,U42	2
5	74AS157 quad 2-input MUX	U51	1
6	74ALS245 bidirectional buffer	U52	1
7	74ALS541 octal buffer	U41	1
8	PAL16L8 programmable logic	U40,U45,U48	3
9	PAL20L8 programmable logic	U47	1
10	74LS125 quad buffer	U44	1
11	TMS34061 video system controller	U43	1
12	TMS34070 color palette	U49	1
13	TMS4161-20 video memory	U1 - U32	32
14	Oscillator, 25 MHz	U46	1
15	Resistor network 16-pin 8 Resistor 22 ohm	U38	1
16	Socket 68-pin PLCC	XU43	1
17	Socket 20-pin DIP	XU1-XU32,XU40,XU45,XU48	35
18	Socket 14-pin DIP	XU46	1
19	Socket 22-pin 400 MIL	XU49	1
20	Socket 24-pin 300 MIL	XU47	1
21	Stake pins	J01 - J06	29
22	Jumper plug		9
23	Resistors 1/4 watt 4.7K	R1 - R10,R20 - R22	13
24	Resistors	R11,R14,R17	3
25	Resistors	R12,R15,R18	3
26	Resistors	R13,R16,R19	3
27	Connector DB9 pin din female	P2,P3	2
28	Capacitor tantalum 39 uF 15 Volt	C34,C37,C51,C58	4
29	Capacitor decoupling 0.01 pF	C1 - C33,C35,C36 D38 - C50,C52 - C56	53

B. PAL Equations

In the following equations, "&" means logical AND, "#" means logical OR, and "!" means negation.

IBM PC Bus to TMS34061 Control Decode

U48 device P16L8

A19,A18,A17,A16,A15,A14,A13,A12,A11,A10,A9 pin 1,2,3,4,5,6,7,8,9,11,13
CASOPT1,CASOPT0,A16PAL,A17PAL,FS2,FS1,FS0 pin 12,14,15,16,17,18,19

```
!FS0=((A19&!A18&!A17&A16&!A15&!A14&!A13&!A12&!A11&A10)#          SR TO MEM
     (A19&!A18&!A17&A16&!A15&!A14&!A13&!A12&A11&A10&!A9))       REG ACCESS

!FS1=((A19&!A18&!A17&A16&!A15&!A14&!A13&!A12&A11&!A10)#         MEM TO SR
     (A19&!A18&!A17&A16&!A15&!A14&!A13&!A12&!A11&A10)#          SR TO MEM
     (A19&!A18&!A17&A16&!A15&!A14&!A13&!A12&A11&A10&!A9)#       REG ACCESS
     (A19&!A18&!A17&A16&!A15&!A14&!A13&!A12&A11&A10&A9)#        X-Y
     (A19&!A18&A17&!A16))                                       X-Y 2 WORD

!FS2=((A19&!A18&!A17&A16&!A15&!A14&!A13&!A12&A11&A10&!A9)#      REG ACCESS
     (A19&!A18&!A17&A16&!A15&!A14&!A13&!A12&A11&A10&A9)#        X-Y
     (A19&!A18&A17&!A16)#                                       X-Y 2 WORD
     (A19&!A18&!A17&!A16)#                                      HOST DIRECT
     (!A19&A18&A17)#                                            HOST DIRECT
     (!A19&A18&A16))                                            HOST DIRECT

!CASOPT0=((A19&!A18&A17&!A16)#                                  X-Y 2 WORD
         (A19&!A18&!A17&A16&!A15&!A14&!A13&!A12&!A11&!A10&A9)) PALETTE

!CASOPT1=((A19&!A18&A17&!A16)#                                  X-Y 2 WORD
         (A19&!A18&!A17&A16&!A15&!A14&!A13&!A12&A11&A10&A9))    X-Y

!A17PAL=((!A19&A18&!A17&A16)#                                   REMAP 5
        (!A19&A18&A17&!A16))                                    REMAP 6

!A16PAL=!A19&A18&A16                                            REMAP 5 AND 7
```

IBM PC TMS34061 Controller

U40 device P16L8

```
CASOPT0,CASOPT1,FS0,FS1,FS2,A0,A1,MEMR,MEMW      pin 1,2,3,4,5,6,7,8,9
AEN,CASCTL1,CASCTL1,HDOFF,HDSEL,ALE              pin 11,15,16,13,17,19
CMD,CASCTL2                                      pin 18,14

!HDSEL=(!FS2&FS1&FS0)                        HOST DIRECT CYCLE

!CMD=((!MEMR&!AEN)#                          COMMAND
     (!MEMW&!AEN))

!ALE=((!AEN&!FS0&HDOFF&!CMD)#                SELECT ON !FS0 AND HOST DIRECT ON
     (!AEN&!FS1&HDOFF&!CMD)#                 SELECT ON !FS1 AND HOST DIRECT ON
     (!AEN&!FS2&HDOFF&!CMD)#                 SELECT ON !FS2 AND HOST DIRECT ON
     (!AEN&!FS0&HDSEL&!HDOFF&!CMD)#          SELECT ON !FS0 AND NOT HOST DIRECT
     (!AEN&!FS1&HDSEL&!HDOFF&!CMD)#          SELECT ON !FS1 AND NOT HOST DIRECT
     (!AEN&!FS2&HDSEL&!HDOFF&!CMD))          SELECT ON !FS2 AND NOT HOST DIRECT

!CASCTL0=((CASOPT0&!CASOPT1&!A1&!A0)#        X-Y
         (!FS2&FS1&FS0)#                     HOST DIRECT
         (CASOPT0&!CASOPT1&!A1&A0))          X-Y LEFT NIBBLE

!CASCTL1=((CASOPT0&!CASOPT1&!A1&!A0)#        X-Y
         (!FS2&FS1&FS0)#                     HOST DIRECT
         (CASOPT0&!CASOPT1&A1&!A0))          X-Y RIGHT NIBBLE

!CASCTL2=((!CASOPT0&!CASOPT1)#               X-Y 2 WORD
         (!FS2&FS1&FS0))                     HOST DIRECT
```

IBM PC TMS34061 to TMS4161 CAS Decode

U47 device P20L8

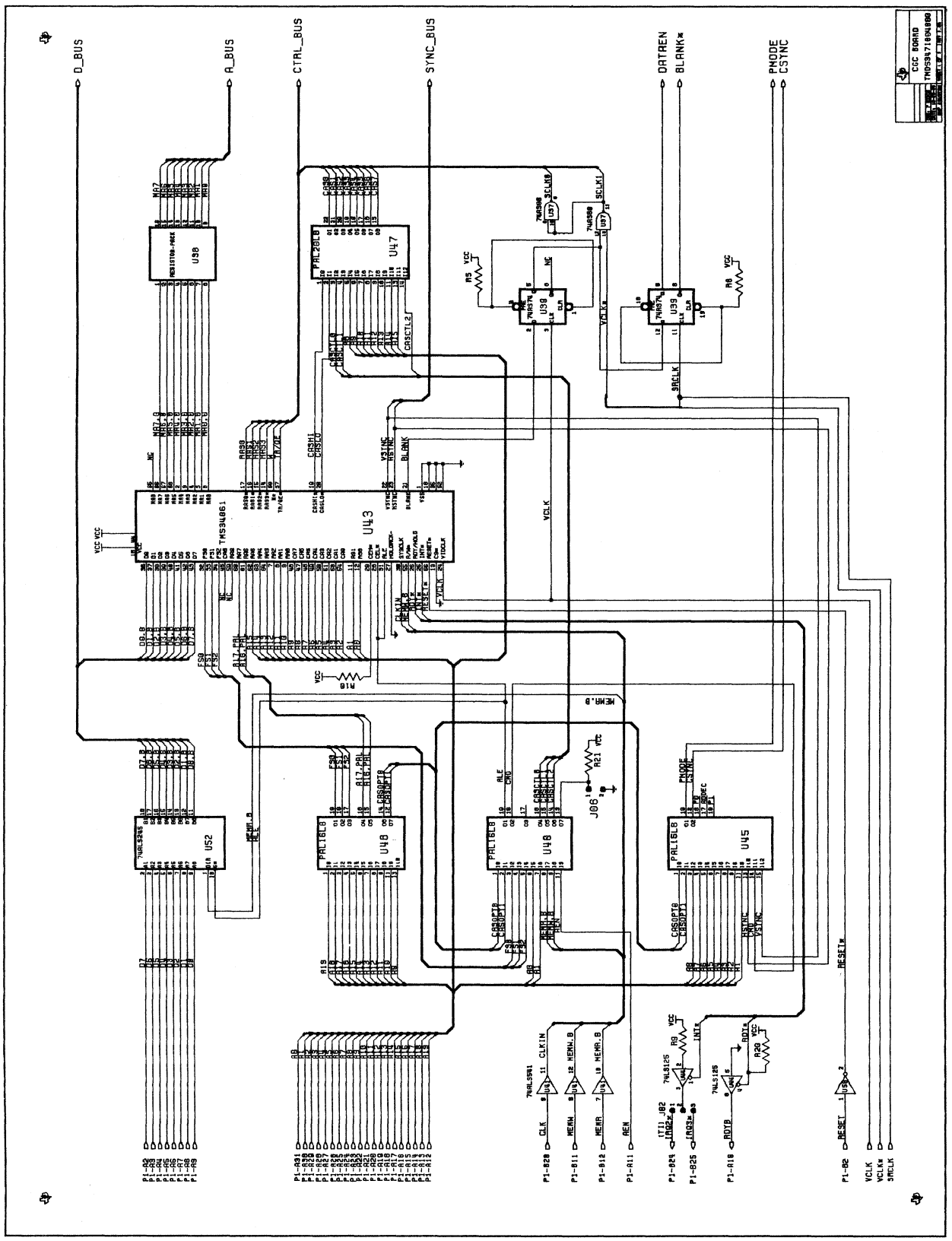
```
CASHI,CASLO,CASCTL0,CASCTL1,A8,A9,A10,A11,A12      pin 1,2,3,4,5,6,7,8,9
A13,A14,A15,CASCTL2                                pin 10,11,13,14
CAS7,CAS6,CAS5,CAS4,CAS3,CAS2,CAS1,CAS0            pin 15,16,17,18,19,20,21,22

!CAS0=((!CASLO&A15&CASCTL0&CASCTL1&!CASCTL2)#       X-Y 2 WORD MASK WITH A15
       (!CASLO&!CASCTL0&!CASCTL1&!CASCTL2)#        HOST DIRECT
       (!CASLO&!CASCTL0&!CASCTL1&CASCTL2)#         X-Y
       (!CASLO&!CASCTL0&CASCTL1&CASCTL2)#          X-Y RIGHT NIBBLE
       (!CASHI))                                   SR TRANSFER

!CAS1=((!CASLO&A14&CASCTL0&CASCTL1&!CASCTL2)#      X-Y 2 WORD MASK WITH A14
       (!CASLO&!CASCTL0&!CASCTL1&!CASCTL2)#        HOST DIRECT
       (!CASLO&!CASCTL0&!CASCTL1&CASCTL2)#         X-Y
       (!CASLO&CASCTL0&!CASCTL1&CASCTL2)#          X-Y LEFT NIBBLE
       (!CASHI))                                   SR TRANSFER

!CAS2=((!CASLO&A13&CASCTL0&CASCTL1&!CASCTL2)#      X-Y 2 WORD MASK WITH A13
       (!CASLO&!CASCTL0&!CASCTL1&!CASCTL2)#        HOST DIRECT
       (!CASLO&!CASCTL0&!CASCTL1&CASCTL2)#         X-Y
       (!CASLO&!CASCTL0&CASCTL1&CASCTL2)#          X-Y RIGHT NIBBLE
       (!CASHI))                                   SR TRANSFER

!CAS3=((!CASLO&A12&CASCTL0&CASCTL1&!CASCTL2)#      X-Y 2 WORD MASK WITH A12
       (!CASLO&!CASCTL0&!CASCTL1&!CASCTL2)#        HOST DIRECT
       (!CASLO&!CASCTL0&!CASCTL1&CASCTL2)#         X-Y
       (!CASLO&CASCTL0&!CASCTL1&CASCTL2)#          X-Y LEFT NIBBLE
       (!CASHI))                                   SR TRANSFER

!CAS4=((!CASLO&A11&CASCTL0&CASCTL1&!CASCTL2)#      X-Y 2 WORD MASK WITH A11
       (!CASLO&!CASCTL0&!CASCTL1&!CASCTL2)#        HOST DIRECT
       (!CASLO&!CASCTL0&!CASCTL1&CASCTL2)#         X-Y
       (!CASLO&!CASCTL0&CASCTL1&CASCTL2)#          X-Y RIGHT NIBBLE
       (!CASHI))                                   SR TRANSFER

!CAS5=((!CASLO&A10&CASCTL0&CASCTL1&!CASCTL2)#      X-Y 2 WORD MASK WITH A10
       (!CASLO&!CASCTL0&!CASCTL1&!CASCTL2)#        HOST DIRECT
       (!CASLO&!CASCTL0&!CASCTL1&CASCTL2)#         X-Y
       (!CASLO&CASCTL0&!CASCTL1&CASCTL2)#          X-Y LEFT NIBBLE
       (!CASHI))                                   SR TRANSFER

!CAS6=((!CASLO&A9&CASCTL0&CASCTL1&!CASCTL2)#       X-Y 2 WORD MASK WITH A9
       (!CASLO&!CASCTL0&!CASCTL1&!CASCTL2)#        HOST DIRECT
       (!CASLO&!CASCTL0&!CASCTL1&CASCTL2)#         X-Y
       (!CASLO&!CASCTL0&CASCTL1&CASCTL2)#          X-Y RIGHT NIBBLE
       (!CASHI))                                   SR TRANSFER

!CAS7=((!CASLO&A8&CASCTL0&CASCTL1&!CASCTL2)#       X-Y 2 WORD MASK WITH A8
       (!CASLO&!CASCTL0&!CASCTL1&!CASCTL2)#        HOST DIRECT
       (!CASLO&!CASCTL0&!CASCTL1&CASCTL2)#         X-Y
       (!CASLO&CASCTL0&!CASCTL1&CASCTL2)#          X-Y LEFT NIBBLE
       (!CASHI))                                   SR TRANSFER
```

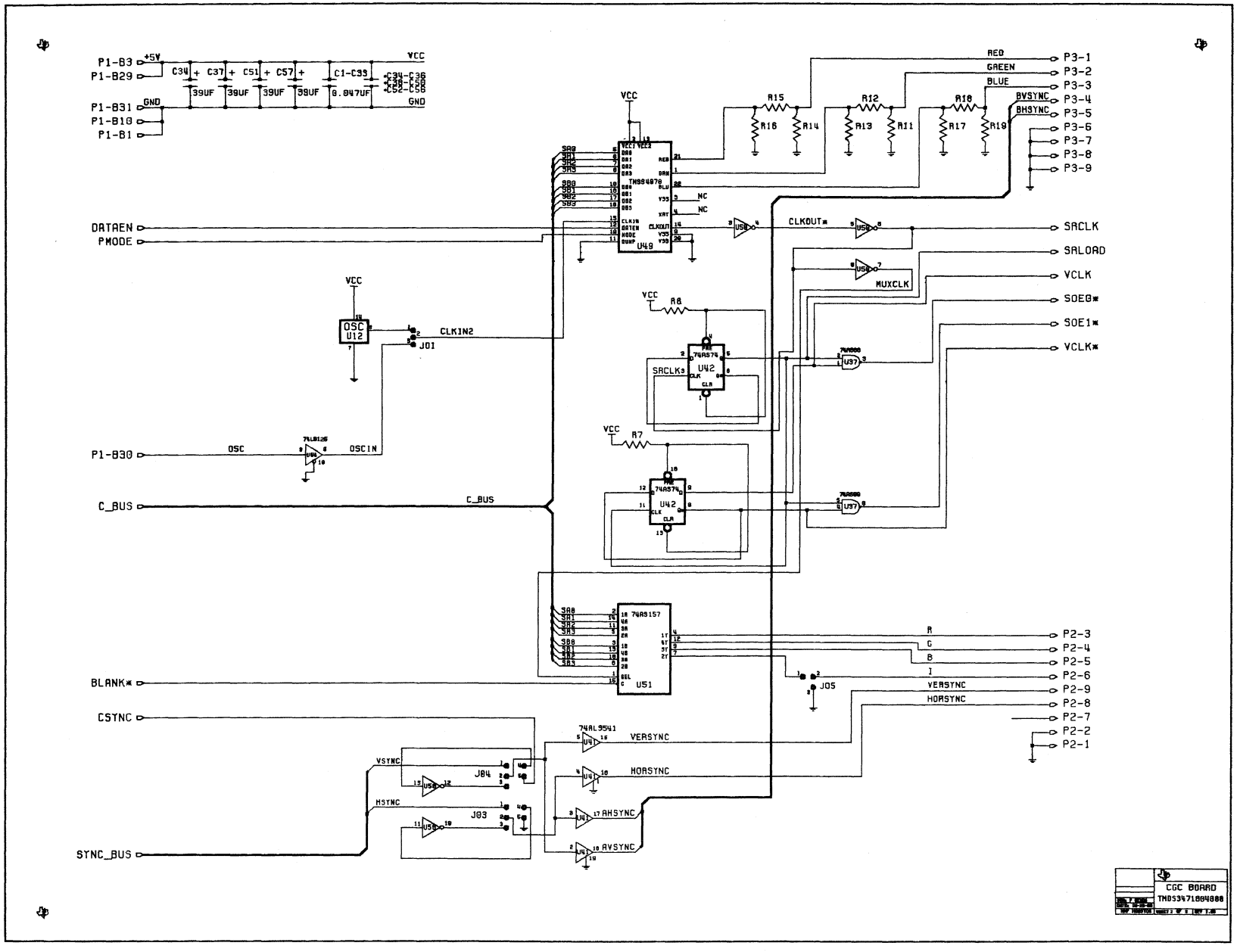
TMS34070 Palette Controller

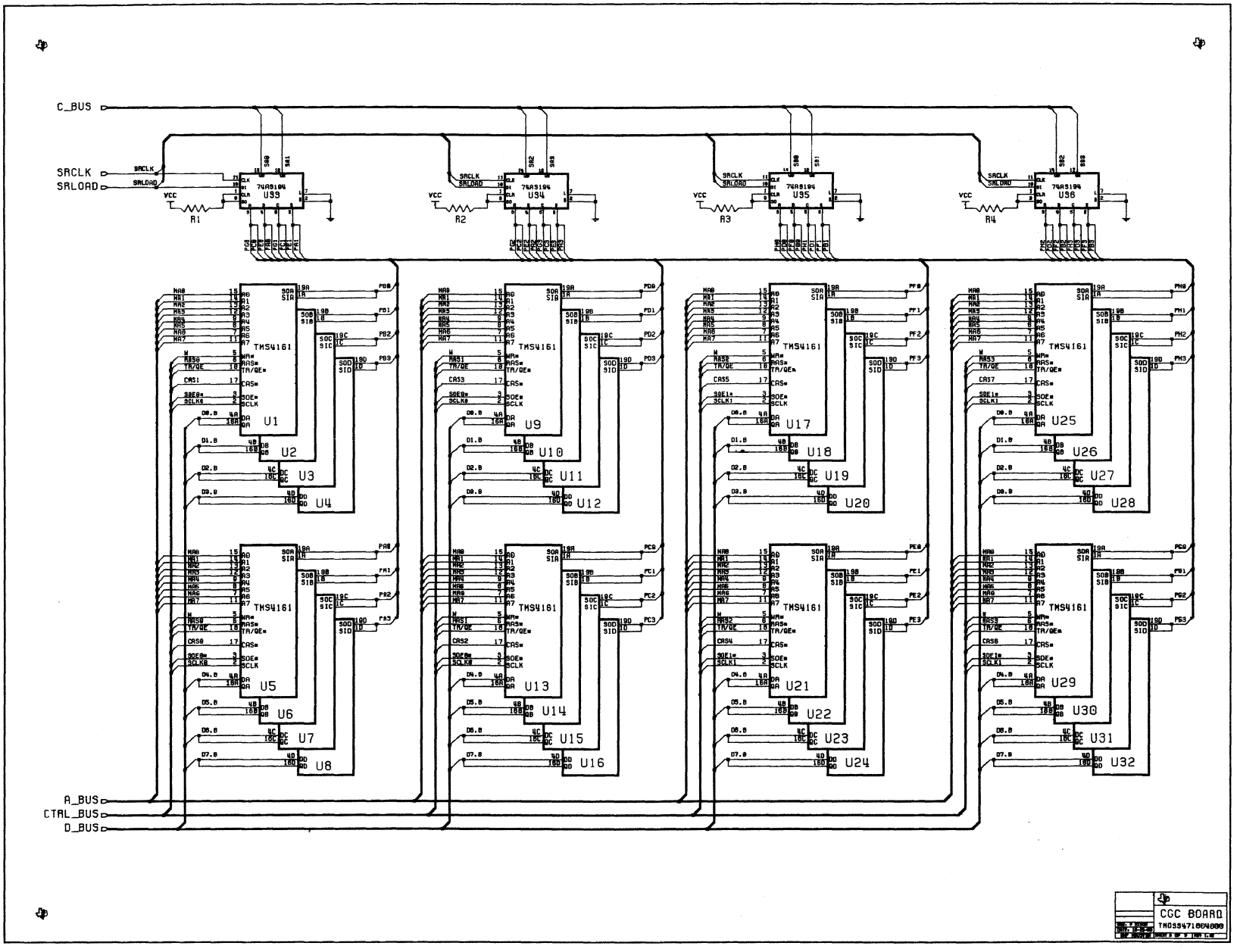
U45 device P16L8

```
CASOPT0,CASOPT1,A8,A7,A6,A5,A4,A3,A2              pin 1,2,3,4,5,6,7,8,9
A1,CSYNC,HSYNC,CMD,VSYNC,P1,ADDEC,P0,PMODE  pin 11,12,13,14,15,16,17,18,19

!PMODE=((!VSYNC&!P1&!P0)#                          PASS VSYNC TO MODE PIN
       (!P0&P1))                                   MODE PIN LOW

!ADDEC=(!CASOPT0&CASOPT1&A8&A7&A6&A5&A4&!A3&!CMD)   SELECT ON P0 AND P1

!P0=((!ADDEC&!A2&!A1)#                             RESET P0 ON ADD SELECT
    (!ADDEC&!P0&A2&A1)#                            HOLD !P0 ON P1 OPERATION
    (ADDEC&!P0)#                                   HOLD !P0 ON NO ADD SELECT
    (!ADDEC&!P0&!A2&A1))                           HOLD !P0 ON P1 OPERATION

!P1=((!ADDEC&!A2&A1)#                              RESET P1 ON ADD SELECT
    (!ADDEC&!P1&A2&A1)#                            HOLD !P1 ON P0 OPERATION
    (ADDEC&!P1)#                                   HOLD !P1 ON NO ADD SELECT
    (!ADDEC&!P1&!A2&!A1))                          HOLD !P1 ON P0 OPERATION

!CSYNC=((!HSYNC&VSYNC)#                            COMPOSITE SYNC
       (!VSYNC&HSYNC))
```

C. Schematics




P1-B3
P1-B29
P1-B31
P1-B10
P1-B1
+5V
GND
VCC
C34
C37
C51
C57
C1-C33
C34-C36
C38-C50
C52-C56
39UF
0.047UF
RED
GREEN
BLUE
BVSYNC
BHSYNC
P3-1
P3-2
P3-3
P3-4
P3-5
P3-6
P3-7
P3-8
P3-9
R15
R16
R14
R12
R13
R11
R18
R17
R19
U49
DATAEN
PMODE
CLKOUT*
SRCLK
SRLOAD
MUXCLK
VCLK
SOE0*
SOE1*
VCLK*
OSC
U12
CLKIN2
J01
R8
R7
U42
SRCLK3
74AS74
74AS157
U51
P1-B30
OSC
OSCIN
C_BUS
BLANK*
CSYNC
SYNC_BUS
R
G
B
I
VERSYNC
HORSYNC
P2-3
P2-4
P2-5
P2-6
P2-9
P2-8
P2-7
P2-2
P2-1
J05
J04
J03
VSYNC
HSYNC
74ALS541
AHSYNC
AVSYNC
CGC BOARD
TMDS3471804888


C_BUS
SRCLK
SRLOAD
A_BUS
CTRL_BUS
D_BUS
U33
U34
U35
U36
R1
R2
R3
R4
TMS4161
U1
U2
U3
U4
U5
U6
U7
U8
U9
U10
U11
U12
U13
U14
U15
U16
U17
U18
U19
U20
U21
U22
U23
U24
U25
U26
U27
U28
U29
U30
U31
U32
CGC BOARD

Index

V

TEXAS INSTRUMENTS

Printed in U.S.A.

SPPU019A