The Director's Composite Eyeglass

By:
Phil Barker
May 6, 1994

Supervisor:
Glorianna Davenport

Abstract:

The Director's Composite Eyeglass performs a digital compositing function of two video images, using a blue screen key to matte the foreground image over the background. It is a chroma-key device implemented in digital hardware. There is a live video feed from which a frame can be captured into memory for use as a background image. This image is then composited with a foreground image, which is shot against a blue screen background. The device filters out the blue screen area from the live foreground image and replaces it with the underlying background image from the frame buffer. The Director's Composite Eyeglass uses modern, special and general purpose hardware to successfully implement this compositing function.

Table of Contents:

List of Figures:

List of Tables:

1 Overview:

The goal of this project is to produce a portable, digital compositing device that can be used by directors to preview a scene or a shot in the field from the comfort of his studio. One of the high cost items on film production is the time it takes to design a shot for complex compositing on location. At best, directors have only cartoon story boards from which to visualize the scene that they are trying to capture on film. With a device such as the Director's Composite Eyeglass (DCE), directors will be able to visualize these scenes with far more clarity. This has the potential for curbing cost overruns. The device will also help the director extend his creative mind into his medium. No longer bound by static story boards, the director can arrange more interesting shots in an animatic "rehearsal" and discover whether or not they have potential in final production.

The Director's Composite Eyeglass implements a digital compositing device which can capture a background image from a live video feed, e.g. a video camera, and then form a digital composite between that image and a foreground image shot against a blue screen. It does so with high quality, modern hardware that will produce a quality image for the director to work from. This design and concept are built with the intention of extending the capabilities to add even more utility in the future.

Video Compositing has been around for a long time in television production studios as well as in film production. Compositng allows a selection of a foreground image to be placed over a background. One of its most common

functions is to perform a blue screen composite between a foreground image and a background. This is typically done by filtering out the blue screen portion of the foreground image and replacing it with the background image. Thus, the background shows through the blue screen areas of the foreground image on the output image. This technique has been used by directors for some time. Its uses have varied from special effects in movies, known as a cinema matte, to doing the television weather report with the weather man standing in front of a series of pictures or maps, known as a TV chroma key.

2 System Description:

The system can be broken into four component subsystems: video input and output, system initialization, video frame buffer, and composite implementation. The block diagram for the system is shown in figure 1. The video input and output unit is responsible for taking NTSC video, the standard format for video in the USA, and converting that signal into digital data. It then is responsible for taking digital video data and re-encoding it into NTSC video format. The frame buffer is needed to store a frame of digitized video from the video input section. This stored image can then be used for the background in the video compositing. The user is able to tell the frame buffer when to grab a frame and can thus control what the background image will be in the composite. The composite section is responsible for looking at the digitized video and then deciding whether to send the foreground pixel or the background pixel to the encoder for conversion back to an analog video signal. The user can set switches to determine the color that the composite section keys from. There is also an override on the composite so that the user can either look at the live video undisturbed, or, the user can look at the stored background image undisturbed. System initialization is needed for the digitizer and the encoder. Both of these sections contain control registers that must be written so as the provide the digitizer and encoder with information about the input and output desired. The system initializer subsystem is done with a micro controlled unit that runs a micro code program.

The DCE has a simple user interface. The user need only supply the input

video signal and then connect the output to whatever display device is available. A frame can be captured by pushing the load button on the DCE. A switch will allow the user to view the captured image, or live video from the input source, or, the user can view the composite of the live video source and the captured frame. The blue screen key used in making the composite can be adjusted by rotating a knob and selecting the portion of the key to be adjusted. In this way, the user can select a key that is suitable to the blue screen conditions that he presents to the DCE.

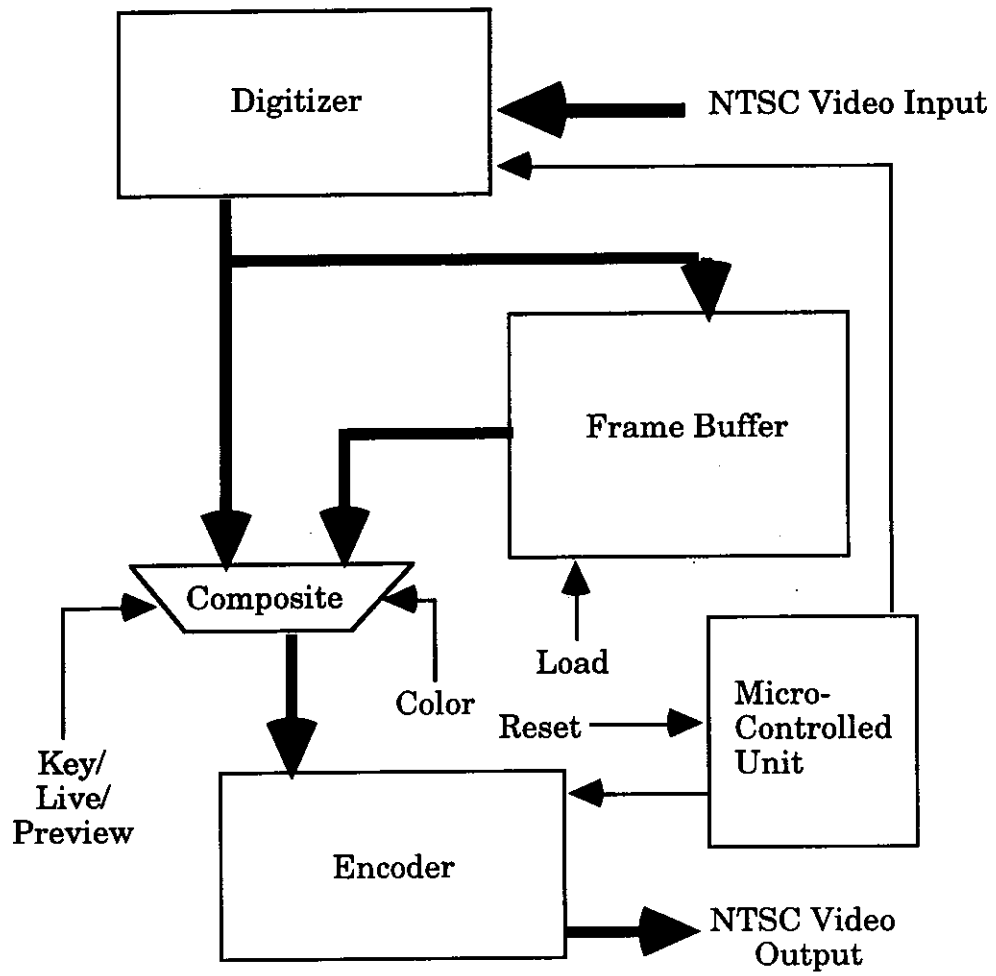


Figure 1 -- System Block Diagram

2.1 Video Input/Output Unit:

The video input/output unit consists of a video digitizer and a digital video encoder. The digitizer takes the video signal from a video camera or other NTSC video source and then digitizes it into 24 bit Red - Green - Blue - (RGB) digital video data. It also uses its initialization information and the sub carrier of the video signal to produce a system pixel clock of 12.27MHz, a double frequency clock of 24.54MHz, a vertical sync signal, a horizontal sync signal, and a field identification signal. The video encoder takes the RGB digital video data and the syncing signals and clock signals to produce a NTSC video output. The encoder uses its initialization information to form proper sub carrier frequency and phase for the output video signal. It also uses this information to build a color lookup table (CLUT) from which each color represented by the RGB data is mapped for encoding back into analog video.

2.1.1 Video Format:

The digitizer and encoder used in this project operate with NTSC standard video. This is the standard video format for the USA and several other countries around the world. In general, video signals carry information about the picture that is present and the scanning speed of the source and monitor.

The video signal has active video portions and non-active portions. The non-active portions, or blanking periods, describe when the scanning beam is moving from the end of a line to the beginning, and, when the beam moves from the bottom of the scan field to the top again. These blanking periods are described by syncing signals. The horizontal sync describes when the blanking period from the end of a line to the beginning of the next. The vertical sync describes when the scan beam is moving from the bottom of the scan field to the top. The total resolution of a frame of video is 640x480 pixels. NTSC sends 60 frames per second of active video. This translates into a 64μsec time on a line of video. NTSC is also an interlaced standard. That is, every other line is in one of two fields. The field allows for better locking onto the video signal for the monitoring device.

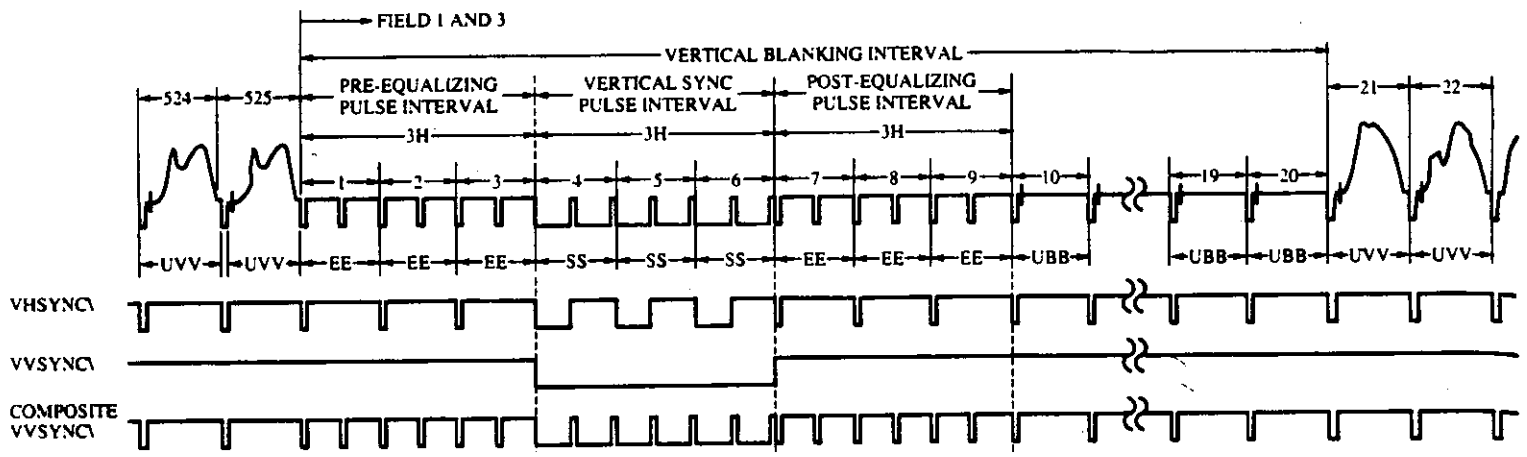


Figure 2 -- Video Format Diagram

2.1.2 Video Digitization:

The digitizing unit consists of a SONY SBX1762-01 24 bit RGB video input module. This is a non-monolithic module of chips on a small printed circuit board. Its design allows for extreme ease of use. Some applications can simply plug in the module and turn it on. The SONY module decodes a composite NTSC video signal into RGB format and digitizes each component into 8 bit digital video data for a total of 24 bits of digital video data. It samples this data at a rate of 12.27MHz providing 640 pixels per horizontal line.

The SONY module needs a +5V power supply for both the analog and digital portions of the module. These power supplies can be brought to the module from separate sources, or, they can be from the same source and de-coupled at the module itself. This project runs from only one +5V power supply, so, the analog and digital power is de-coupled, via a ferrite bead around the leads to the analog power, at the module. There is a similar design for the grounds of the module. The digitizer requires that there be a clean analog ground. This is a requirement forced by the nature of digitization, which works by a voltage comparison. The clean ground is necessary to provide for an even base of comparison. Like the +5V power, the grounds must be separated either at the power source or at the module. This project de-couples the grounds at the module with ferrite beads around the analog ground leads. The SONY module also requires a +12V power supply for the analog portions of the module. This is also provided by the single power supply for the working components.

The SONY module has a video input on one pin. This input is internally terminated with a 75Ω resistor. This allows video to be taken directly to the module from the video line, without extra termination. The module also contains an output enable line. This is tied to ground so that the module is always ready and able to output digital video data. A complete logic diagram for this and all other units can be found in Appendix A.

The SONY module outputs several signals along with the digital video data. The digital video data is 24 bit RGB data. This allows for a total of 2^{24} (approximately 16 million) colors from the digitizer. This is enough information to accurately reproduce the image that was digitized. Digital data is clocked out of the module on a 12.27MHz clock signal.

The clock is another signal that the module outputs. The main pixel clock is used to run the rest of the hardware for the memory and the composite. It is a 12.27MHz clock. The SONY module also outputs a 24.54MHz double pixel rate clock. This is used only by the encoder to help with its internal operations. Both of the clock signals are inverted so as to allow them to drive several more devices than is possible for the SONY to drive alone.

The SONY module also outputs several syncing signals that are used as cues the rest of the hardware as to when video events are occurring. A table of these signals can be found in table 1. Such events include when the video signal is in a blanking period and what field a current line is in. The syncing signals are a horizontal sync (hsync), a vertical sync (vsync), a composite sync

(csync) which is a combination of the hsync and vsync, a field signal, and the video sub carrier signal. The video sub carrier and csync are not used by this project. The hsync, vsync, and field signal are used by various components of this project for control functions and to help with the encoding of the digital data back into a composite NTSC signal. Hsync is, however, inverted so as to comply with the standard description of an hsync, which is a low pulse at the end of an active video line.

Table 1: Syncing Signals

Name	Description
horizontal sync (hsync)	describes the end of a line of active video
vertical sync (vsync)	describes then end of a frame of video
composite sync (csync)	contains both vertical and horizontal sync information
field signal	describes whether a line of video is in the even field or the odd field of the interlaced frame
video sub carrier	this is a stripped video sub carrier from the actual NTSC signal

The SONY module's digitization process can be modified. It is possible to initialize the module so that it will output different picture levels,

sharpness, hue, or saturation. These effects will allow the DCE to be tuned so that the digitizer and encoder work better together to produce a higher quality output image. The SONY module allows for this initialization via an I^2C bus. This is a serial communications system involving three address lines (A2, A1, A0), a serial data line (SDA), and a serial clock line (SCL). A table of these signals can be found in table 2.

Table 2: I^2C Bus Signals

Name	Input/Output	Description
A2	Input	the high address bit for slave addressing
A1	Input	the next most significant bit for slave addressing
A0	Input	the low bit for slave addressing
SDA	Input/Output	the data line for serial communications
SCL	Input	the clock line for serial communications

The I^2C bus works by placing data on the SDA line and then pulsing a high signal on the SCL line. A diagram of these signals for the SONY module can be found in figure 3. The SONY module requires a start condition to take place on the SDA and SCL lines. After the start condition, a slave address is sent. This consists of 8 bits of data pulsed out on the SDA line with clock

pulses on the SCL line for each bit. The slave address contains a section that tells the SONY module that it is receiving an address and then a section that tells it which module on the I^2C bus is to take the following information. This is necessary as the I^2C bus is designed to be a communication medium between more than just two modules. Since there is only one receiver on the bus (that is the SONY module), its address bits, A2 - A0, are tied to ground and the address sent in the slave address byte is address zero or the address assigned to the SONY module by the address bits.

Following the slave address byte is an acknowledgment signal. This is sent over the SDA line from the SONY module to the initialization hardware. It is received when the initialization hardware clocks the SCL line. This acknowledgment bit must be sent and received after every data byte that is sent to the SONY module. The next data byte is a subaddress. It describes which control register address on the module to start reading data into. This byte is sent with the address of the average picture level (APL) register. The next byte is the average picture level which is set at mid level, a value of 32. The subaddress automatically increments after the data byte so that the next byte sent is also a data byte and not a new address byte. The next address is sharpness control (SHP). This is also set at mid level, a value of 32. Next is the saturation (SAT) register. This is set to about 150% saturation, a value of 54. Finally is the hue (HUE) register. This is set to about +.2 degrees, or a value of 25.

The final step in initializing the SONY module is the stop condition on the

SDA and SCL lines. These corrections are necessary for good quality video reproduction. Without them, the encoder will not produce an accurate color map for encoding digital data into NTSC video.

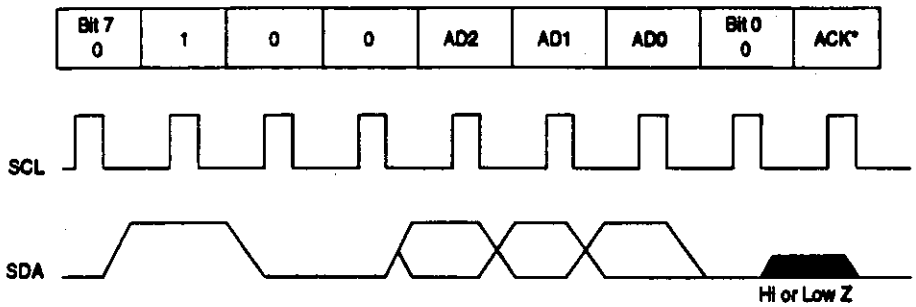


Figure 3 -- I^2C Bus Signal Format

2.1.3 Digitized Video Encoding:

The video encoding unit consists of a TMC22190 (22190) encoding chip, a 74F574 TTL chip, and some analog circuitry. The 22190 converts digital video data, in a variety of formats, to a NTSC standard or PAL standard signal with a modulated color sub carrier. The 22190 requires syncing signals and digital data along with a specific initialization sequence to perform this conversion.

The 22190 requires a +5V power supply for both an analog system and the digital system. In a similar manner to the SONY module, this power can be separated at the power source or at the chip. Since this project runs from a single +5V power supply, the power for the analog section and the digital section is separated at the chip via ferrite beads around the analog power lines. There are also separate grounds for the digital system and the analog

system. These are also separated at the chip via ferrite beads around the analog ground lines. The cleanliness of the power at this chip is very important. It is even more so than at the SONY module as the SONY has some power clean up on board. The 22190 requires bypassing of the power supply at the chip. Several 0.001μF capacitors are soldered in between the power and ground pins on the chip. The entire power supply is bypassed at the board connection with larger capacitors in the 470μF range. This gives a clean power supply to the chip regardless of other activity on the board.

Proper encoding requires that there is a clean set of reference voltages for the comparison of digital data into an analog signal. There are three pins on the 22190 that are responsible for providing the references to the chip for proper encoding. A combination of analog circuitry provides a 1.235V reference at the Vref pin, a 0.1μF capacitor for de-coupling at the COMP pin, and a ground reference at Rref. Encoded video is sent out over the composite video output. This output is connected directly to the transmission medium and is terminated at the 22190 with a 75Ω resistor. Appendix A contains a full set of logic diagrams for this and all other units.

The 22190 requires two clock signals for operation. It needs a double pixel rate clock. For the purpose of using square pixel NTSC format, the pixel rate is 12.27MHz. So, the double pixel rate clock is 24.54MHz. This operates the internal state machines that are responsible for controlling the conversion process. The 22190 also requires the pixel clock described above. This is necessary for clocking pixels into the chip via the PD port. Both of these clocks are provided by the SONY digitizing module. They are taken from the

inverted clock signals from that module.

Syncing signals are required in any attempt to convert digital video data back into analog form. The 22190 requires a vertical syncing signal and a horizontal syncing signal. These two signals are generated by the SONY module. The hsync from that module is inverted to conform to standards for video, which is a low pulse at the end of an active line of video. Both the hsync and the vsync are delayed by the 74F574 chip. This chip is clocked with the pixel clock so as to effectively delay the syncing signals by one pixel of where they would be after coming out of the SONY module. This allows for a one pixel latency in the rest of the board. This latency is induced by the compositing hardware. The result is that the syncing signals now coincide with the rest of the pixel data.

The 22190 also has a variety of other pins that help out with, or control, other functions in the chip. The KEY pin is used to specify that the 22190 should accept digital video data across the PD port rather than the CVBS bus. It unties the chip from its companion digitizer which is not used by the DCE project. The KEY is tied to ground for this purpose. There is also the PDC pin which is setup as an output during the initialization for the 22190. It is meant to be used as a frame buffer direction control. However, the DCE leaves it unconnected and controls the frame buffer through other means. There are several JTAG video test pins which are not in use when the chip is actually performing video conversions. These are also tied to ground. Finally, there are the BYPASS\ and OL pins. These are all grounded. The BYPASS\ is grounded for proper CLUT operation. The OL pins are grounded so as to shut off the 22190's internal overlay operation. The 22190 is

designed to perform a compositing function of a sort using an overlay technique. This overlay option is not used by the DCE and is therefore shut off.

An initialization is also required for the 22190 to operate correctly. Since it is a fully programmable chip which can encode for any kind of video desired, it must be told which video to encode for. Values for the sub carrier phase and frequency must be entered and the chip must be told whether to encode PAL or NTSC levels for the output video. It must also be told what format the digital video data is in and how it is being sent to the chip. Finally, a CLUT must be loaded so that the colors that the digital video data represents can be correctly converted to the corresponding analog signal.

Table 3: 22190 Microprocessor Interface Signals

Name	Input/Output	Description
A1 and A0	Input	address lines for the 22190 interface
RESET\	Input	hardware reset which halts all internal operations
R/W\	Input	read or write line
CS\	Input	chip clock line
D[7:0]	Input/Output	data bus

A standard microprocessor interface is provided on the 22190 for initialization. It supports two address lines (A1 and A0), a reset line (RESET\), a read/write line (R/W\), a processor clock line (CS\), and an 8 bit data bus (D[7:0]). A table of these signals can be found in table 3. Operation relies on both edges of CS\. The falling edge clocks in the values of the A1, A0, and R/W\ lines. This allows the data to be directed at an address register or an actual control register. It also directs the 22190 to either read or write a value. A table of values for these signals is found in table 4. The actual value comes across D[7:0]. This data is clocked in on the rising edge of CS\. An additional falling edge is needed to clock the last value all the way through to its working register. This is usually accomplished by a successive series of operations. Figure 4 describes this format.

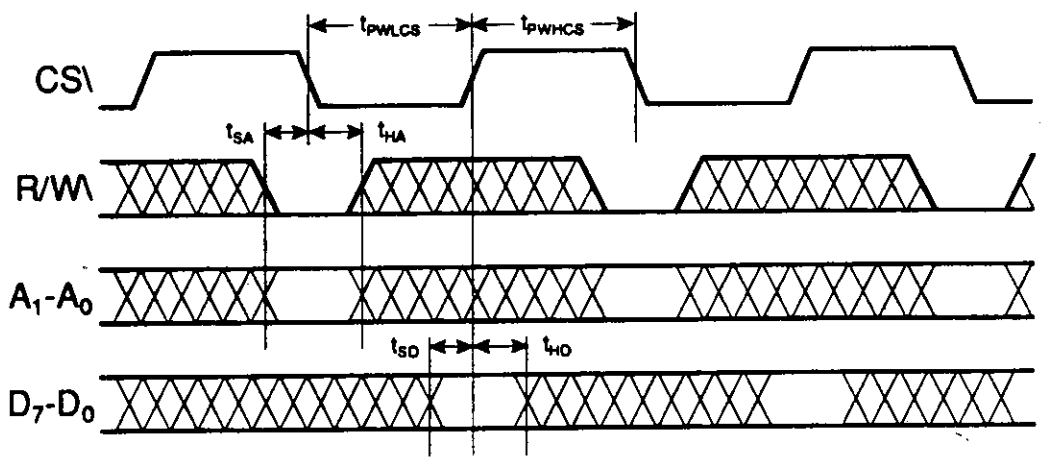


Figure 4 -- 22190 Microprocessor Interface Format

Table 4: 22190 Microprocessor Interface Values

A1	A0	R/W\	Description
0	0	0	load D[7:0] into control register pointer
0	0	1	read control register pointer on D[7:0]
0	1	0	load D[7:0] into CLUT address register
0	1	1	read CLUT address register on D[7:0]
1	0	0	write D[7:0] to addressed control register
1	0	1	read addressed control register on D[7:0]
1	1	0	write D[7:0] to addressed CLUT location
1	1	1	read addressed CLUT location on D[7:0]

Initializing the 22190 for operation in the DCE requires first a low signal on the RESET\ line. This stops the internal state machines and puts the rest of the chip on standby. It is now possible to change the internal registers that control the format and operation of the 22190. First the address of the first control register is loaded into the control register pointer. Then all of the values are written into the control registers. The control register pointer automatically increments with each rising edge of CS\ during a data write or read. This facilitates writing the control registers. A table of addresses and values for the control registers can be found in table 5.

After the control registers are all written with the appropriate values the CLUT address register is loaded with the first CLUT address. The CLUT is

then filled with all of the values for the red, green , and blue digital colors. The values used in the DCE are not altered from the digital video data sent by the SONY module. Therefore, the values put into the CLUT equal the address of the CLUT, i.e. red 0 is mapped to 0. Each CLUT address has three bytes associated with it. This allows the table for the red, green, and blue to be loaded without the need for readdressing. But, they must be loaded for each address accessed. After the loading of the CLUT RESET\ is brought high and the encoder begins operation by locking onto the first field that it finds.

Table 5: 22190 Control Registers and Values

Name	Address(in Hex)	Value(in Hex)	Description
Global Control	00	01	select NTSC and no software reset
Format Control	01	10	select layering, 24 bit RGB, no test and slave mode
Interface Control	02	2C	select enable sync input signals, PDC master, time base
Test Control	03	02	disable test mode
Layering Control	04	40	select enable KEY pin, and no layering mode
Hsync Tip Length	10	3A	hsync tip length
Breezeway Length	11	07	breezeway length
Burst Length	12	1F	color burst length
Color Back Porch Length	13	0F	back porch
Extended Color Back	14	23	8 LSB
Active Video	15	8B	8 LSB
Active Video Start	16	05	8 LSB
Active Video End	17	77	8 LSB
MSB Reg.	18	65	MSB of 14,15,16,17
Front Porch Length	19	12	front porch
EQ Pulse Low Length	1A	1C	low length
EQ Pulse High Length	1B	6A	high length
Vertical Low Length	1C	4C	low length
Vertical High Length	1D	3A	high length
Color Bar Length	1E	52	color bar length
Sub Carrier Freq. byte 4	20	AB	LSB byte
Sub Carrier Freq. byte 3	21	AA	next LSB byte
Sub Carrier Freq. byte 2	22	AA	next LSB byte
Sub Carrier Freq. byte 1	23	4A	MSB byte
Video Phase Offset 1	24	00	LSB byte
Video Phase Offset 2	25	00	MSB byte
Burst Phase Offset 1	26	00	LSB byte
Burst Phase Offset 2	27	20	MSB byte

2.2 Digitizer/Encoder Initializer:

Initialization is done with a micro controlled unit (MCU) which is attached to the microprocessor interface of the 22190 as well as to the I^2C bus of the SONY module. A single program runs, at power up or reset to the micro controlled unit, which carries out the initialization detailed in previous sections. The micro controlled unit then stays in a single state for the rest of the running time of the DCE.

2.2.1 Micro Controlled Unit:

The micro controlled unit is implemented using a Motorola XC68HC811E2 Micro Controller (6811), two 74F574 register files, a Maxim MAX233 RS-232 receiver, and some analog circuitry. Complete logic diagrams for this and all other units can be found in Appendix A. The 6811 also requires a program to be burned into its on board EEPROM memory. This is necessary as it has no other support memory or storage device for the program. The program is thus developed on an outside platform and then sent to the 6811 after assembly. This project used the as11 assembler from Motorola to assemble the micro code program from source and the dl program to download the program into the EEPROM on the 6811. Both of these programs are available in a standard form for a DECStation, or from Randy Sargent at the MIT Media Lab, who has made minor modifications. Both of these programs require a DECStation to run and a free serial port on that computer for attachment to the DCE.

The 6811 is powered only by a +5V power supply. This is the same power supply for the rest of the project. The 6811 does not have the extreme power cleanliness problems that the digitizer and encoder have. So power to the chip needs only minimal bypassing. The ground is also that of the rest of the project. The 6811 also needs a clock to run. It runs from an 8.0MHz fundamental crystal. This controls the internal state machines and forms a 2MHz clock that is used to signal output events.

The 6811 has an on board digitizer which must be set up with reference

voltages for proper chip operation. Reset circuitry is needed to keep the 6811 from activating until the power supply has come up to a proper +5V value. This just consists of a capacitor and resister with a rise time greater than the +5V range up of the power supply. Also, across the reset circuit is the reset switch which will shunt the power to the chip when activated. This reset circuit is connected to the RESET pin on the 6811. This allows the chip to be reset and its program re-run without cycling the power. This is more for a testing purpose when the chip is programmed a variety of times and must be cycled to run the new program.

There are four modes of operation for the 6811. The DCE project requires only two of these modes. By fixing the MODE A pin to ground and then putting a switch on MODE B, from ground and +5V the DCE can access these two modes. The first mode is the standard single chip running mode. This allows the 6811 to run with just itself and no support memory. This is the normal mode of operation for the DCE project. The other mode is the bootstrap mode. This allows the 6811 to be programmed. When in this mode the 6811 can be programmed over a serial line. This allows program development for the initialization.

Serial communications between the programming station and the 6811 are done through the serial line on the station and the MAX233 receiver. The receiver gets the signal from the station and converts it to TTL levels of voltage and this is put through to the 6811 via the port D interface. Communications between the 6811 and the 22190 and SONY module are done through the port C and port B lines. These are a bi-directional and

output only ports, respectively, of 8 bits a piece. These are connected to the 74F574 registers. The registers are clocked by the 2MHz clock from the 6811. This gives a glitch free output to the digitizer and encoder.

2.2.2 Micro Code Program:

The initialization sequence required by the 22190 and the SONY module needs next to no feedback to the micro controlled unit. This allows the 6811 to be used primarily as an output assertion device. Thus, its data ports are used simply to assert signals on the lines to the 22190 and to the SONY module. The SONY module does have some feedback to the controller, the acknowledge bits. This can be handled with the use of the bi-directional port on the 6811. So, the bulk of the program is simply loading values onto the ports in the correct sequence. Table 6 contains information on which port lines go to which microprocessor or I^2C bus lines on the 22190 or SONY module.

Only a small subset of 6811 micro instructions are needed to perform writes to the ports. A table of the instructions used can be found in table 7. The 6811 operates on a load/store basis. Working registers are loaded with data either from memory or from the program itself. This data is then stored in locations that correspond to the ports. Data for the control lines and the data lines are loaded into the B accumulator register and A accumulator. Clocking the clock lines, either CS\ for the 22190 or SCL for the SONY module, is then done as is appropriate.

Table 6: Port Definitions

Port	Bit Number	Unit	Connection
C	7	22190	RESET\
C	6	22190	R/W\
C	5	22190	A1
C	4	22190	A0
C	3	N/A	Unconnected
C	2	22190	CS\
C	1	SONY	SDA
C	0	SONY	SCL
B	7	22190	D7
B	6	22190	D6
B	5	22190	D5
B	4	22190	D4
B	3	22190	D3
B	2	22190	D2
B	1	22190	D1
B	0	22190	D0

However, before the ports are ready to accept writes out, there must be some setup in the 6811 itself. Port C is bi-directional and must be set to be an output when the pins carry assertions. It must also be set back to an input for certain port pins when the acknowledge bits from the SONY are received.

The stack in the 6811 must also be initialized. After this initialization of the 6811 the program that takes care of initializing the rest of the hardware runs. A complete listing of the program can be found in Appendix B.

Table 7: 6811 Instructions Used

Mnemonic	Description
LDAA	load accumulator A
LDAB	load accumulator B
STAA	store value in accumulator A
STAB	store value in accumulator B
LDS	load stack pointer
LDX	load x register
LDY	load y register
INCA	increment A accumulator
INY	increment y register
JSR	jump to subroutine
CPY	compare y register
BLS	branch if less than
BRA	unconditional branch
SEI	disable interrupts
BSET	set bits according to mask
CLI	clear interrupts
RTS	return from subroutine

The first task done is to activate the reset on the 22190. The control registers are then written in the order indicated in the encoder description section of this paper. This is carried out by the 6811 by loading a value into accumulator A and then jumping to a subroutine which performs the necessary pulsing on CS\. This is repeated for each control register value. The CLUT is then loaded into the 22190 as described in the encoder description.

Next, the SONY module is initialized over the serial interface. The serial interface is setup on two bits of port C on the 6811. These lines are pulsed on and off with the description of the I^2C bus interface. This is done one bit at a time as is the nature of serial line communications. There are some other minor details to the workings of the program. Interrupt vectors must be written so that if the processor works incorrectly or receives internal interrupts it has a place to jump to. These interrupts are not expected to be occurring in this project. All of them have been sent to a procedure which branches to itself tying up the processor indefinitely. This is very easy to spot when debugging the program so it is a logical choice. Along with the interrupts, the reset vector must be set to the beginning of the program. There are also points in the program which tell the assembler where to place code. These origins are necessary so that the assembler know to place the program into the EEPROM of the 6811 and not into volatile memory where it will be lost at power down.

2.3 Frame Buffer:

The frame buffer consists of a large FIFO memory structure and a controller to capture a video frame. It also consists of a switch to tell the controller when to load a frame into the memory vs. when to play back to the contents of the frame buffer. The FIFO is a number of memory chips that are all independently FIFO chips wired in such a way as to provide one large FIFO structure. The controller is programmable logic and alerts the FIFO structure how to capture a frame. The FIFO structure used is a two bank memory. Figure 5 is a block diagram of the frame buffer system.

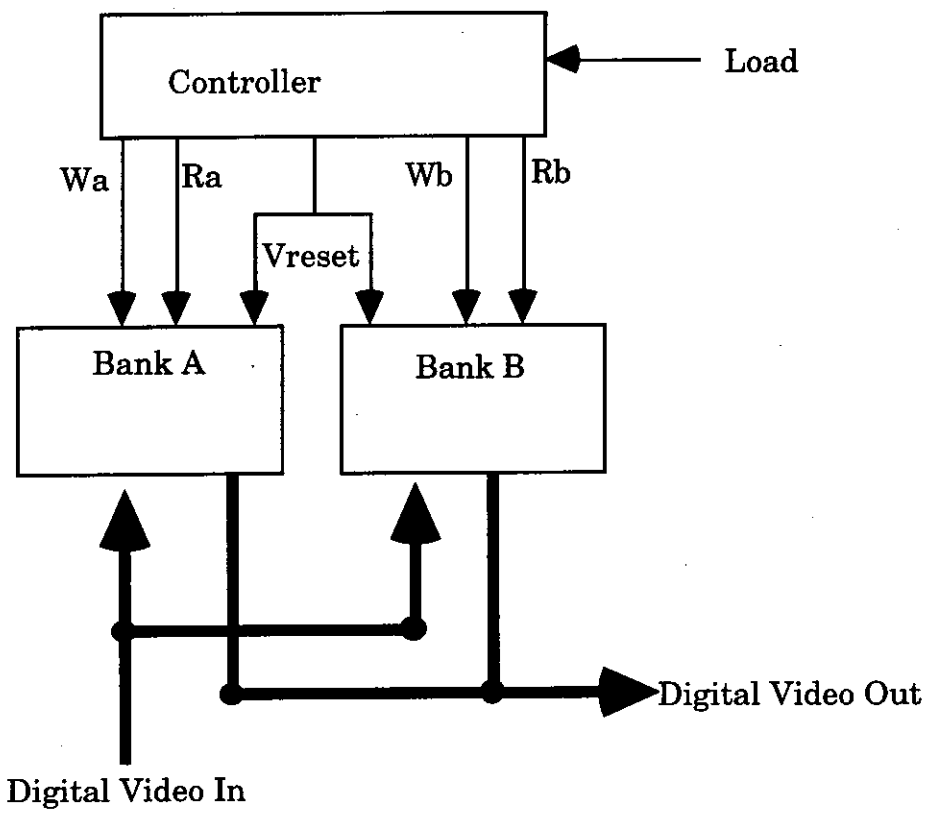


Figure 5 -- Memory Block Diagram

2.3.1 Memory:

The memory used is TMS4C1050DC-40N (1050) FIFO memory chips wired in such a way as to provide a full frame buffer. FIFO stands for First In, First Out and describes a memory where the first item written into the memory will be the first item read out of the memory. This eliminates the need for a counter that covers the address space for an entire frame, cutting down both on design complexity and the number of components. With enough FIFO memory the controller need only tell the memory to read or write when there is active video. The memory structure is able to increment to the next available space in the memory. Only the write and read pointers need to be reset by the controller. The FIFO does not provide random access memory, but, it is not needed in the context of the DCE. Since a frame is read or written from beginning to end each time the memory is accessed and has no other access pattern, the FIFO is a good solution to the problem of frame storage.

The 1050 is not in itself big enough to hold an entire frame of video. Each 1050 stores 262,264 4 bit words. Immediately, it is noticed that the chip does not store a large enough word for a frame of video. Since the DCE works with 24 bit video, the frame buffer must store a 24 bit word. The second problem is that the video frame used in the DCE is 640x480 pixels. This is a total of 307,200 pixels. The FIFO must hold 307,200 24 bit words for an entire frame of video.

Using two banks of 6 1050s a bank allows enough memory space for a frame

on video. Appendix A contains complete wiring diagrams for the memory as well as for all other units. The total memory capacity of the FIFO structure with two banks is 524,528 24 bit words which is enough to capture an entire frame of video. The scheme for using this memory is to alternate on a line by line basis between the two banks when writing or reading a frame of video. Thus each bank holds half of the video frame or 153,600 24 bit words. This fits inside of one bank of memory.

The operation of the memory maintains that on a load signal at a vertical blanking period the memory will start a write operation for an entire frame. This entails writing one line to one bank and then writing one line to the next bank, and so on until the end of the frame. The two banks have their corresponding bit inputs tied together to the SONY digital video data outputs allowing easy access to the video data. When no load signal is present during the vertical blanking period the two banks read out their data in a similar fashion to the way they were loaded, one line per bank at a time. The reads and writes are controlled to occur only during the active video portions of the digital data from the SONY. This is due to the nature of video signals. On occasion there will be lines that are only 639 pixels in length. The syncing signals will describe the active video portions irregardless of the actual pixel counts and thus will ensure that each line is displayed as it should be. Otherwise, errors could occur in the playback of the frame buffer. The read and write pointers are also reset to zero during the vertical blanking period. This allows for the banks of memory to always be writing or reading over the same space, ensuing proper recording and playback of a frame of video.

The operation of the 1050 involves a clock to the write end of the buffer and a clock to the read end of the buffer. It also has a read pointer reset, and a write pointer reset, as well as, a read activation line and a write activation line. Since the chips are always clocked with the pixel clock, as that is when data is ready, both the read and write clocks are tied to the pixel clock. Both read and write pointers are reset during the vertical blanking period. Therefore, only one reset signal is needed and goes to both read reset and write reset. The read and write activation lines are, of course, separate as a frame will either be written or read but never both at the same time. Table 8 contains signal definitions for the entire frame buffer.

Table 8: Frame Buffer Signals

Name	Input/Output	Pos./Neg.	Description
Ra	Input	Pos.	read assertion for bank a
Rb	Input	Pos.	read assertion for bank b
Wa	Input	Pos.	write assertion for bank a
Wb	Input	Pos.	write assertion for bank b
Vreset	Input	Pos.	reset for all address pointers
SRCLK	Input	N/A	clock for read section
SWCLK	Input	N/A	clock for write section
IV	Input	N/A	input digital video data
MV	Output	N/A	output digital video data

The frame buffer is wired with two banks of memory. Each bank has a duplicate data entry format to the other, i.e. the same 24 bits of data go to each bank. The controller then decides and activates the appropriate bank for the write operation. The outputs are also common as only one bank will be read at a time. The output data lines are therefore tied together as well.

The 1050 memory has two timing requirements that must be met in order to ensure proper operation. The first is that the read and write pointers will lose their address values after a finite amount of time with no activation on either the read or write lines. After a one msec period with no activation of the read line, the read pointer will not have a valid address, and, there is no predicting which data will be read out of the buffer. There is similar behavior on the write line, where a one msec period of no activation causes the write pointer to become invalid, after which new writes to the buffer may be stored in random locations about the memory. There is a simple avoidance to this problem. When ever the buffer is clocked with the read or write line activated, the read or write pointer is refreshed. If this occurs before the one msec time period then they will be refreshed with the value that is current and valid. Since the read pointer or the write pointer is in use whenever there is active video, one of the two will be constantly refreshed throughout the frame. This leaves no period of one msec of deactivation for the read or write pointer. After one frame of no use, however, one msec will already have passed and the pointer not active during the frame will no longer be valid. But, the end of a frame is signified by a vertical blanking period. Rather than storing blank video, the pointers are both reset to zero during this period. The reset operation also refreshes the pointers and they are thus valid for the next frame cycle.

The second timing problem for the 1050s is one of data readiness and transition time from data output to a high impedance state. Since two banks of memory are being used, the frame buffer output is coming from two different sources. These two sources both sit on the same output bus to the compositing hardware. It is important that they not try to drive data onto the bus at the same time. Fortunately, the 1050 will only drive the bus when data is being read out of the memory. The rest of the time its outputs will be in a high impedance state. Since the frame buffer is read one line from a bank and then alternates to the next bank, there is the entire horizontal blanking period in which the active bank can shut off its outputs and the next bank can turn its outputs on. The horizontal blanking period is long enough for this operation to occur.

2.3.2 Buffer Control:

The frame buffer involves a piece of control hardware for the FIFO memory. This control is not extremely complex due to the nature of the FIFO memory. Only read, write, and reset must be asserted by the controller. Addressing is not an issue. The memory structure has more than enough room for the entire frame of active video. It does not, however, have enough room for all of the blanking periods as well. The buffer controller must therefore, only capture the active video portions of the frame that is being written. Another problem is that there may be lines if video where only 639 or even as many as 643 pixels come from the SONY. This is due to the analog

nature of video signals. It is up to the buffer control to count the same amount for every active line that it encounters. This is necessary so as to avoid improper looking video.

The solution to the problem is to key the controller off of some blanking information and then count out the active line of video. Since this is a digital operation, each line will be of the same length. Also, since the memory is larger than the active video of a frame, extra writes can occur for every line, ensuring capture of all of the information on that line. The controller is done with some programmable logic. In this case ALTERA EPM5032 programmable logic is used. This is a 32 macro cell chip. The program that is compiled to logic and then burnt into the device is written in the MAXII+ environment.

The code for the controller essentially consists of two different finite state machines and several counters. The first state machine is responsible for determining the vertical state of the frame of video. A state diagram can be seen in figure 6. By using the syncing information, vertical blanking, idle, and active periods can be established for any frame. Table 9 contains a list of the signals that are involved in the controller for the frame buffer.

The vertical state machine is used to determine when to increment or reset the internal counters that keep track of line position. The internal line counter is responsible for counting out the line of active video. The line counter and the vertical state are used to calculate the cblank signal. This tells the rest of the controller when there is active video vs. when there is a

blanking period. The hsync is also used when generating the cblank signal. The cblank and the load signal are used in the other state machine in the buffer controller. A state diagram for this machine can be found in figure 7.

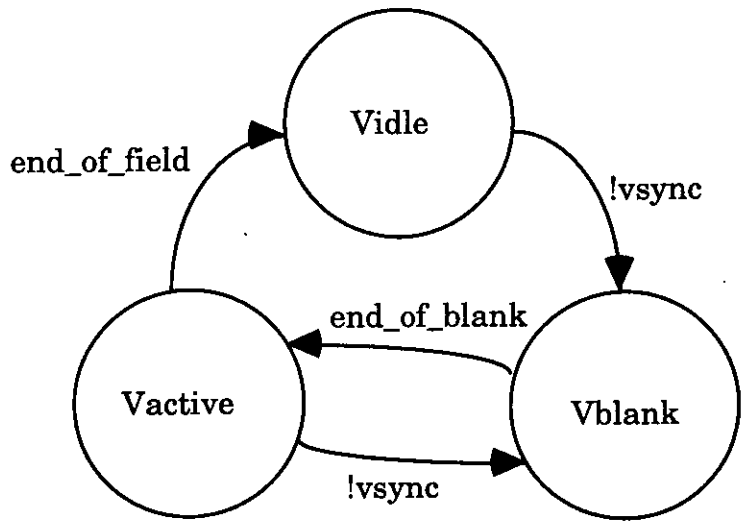


Figure 6 --
Vertical State FSM

Table 9: Buffer Control Signals

Name	Input/Output	Pos./Neg.	Description
hsync	Input	Neg.	horizontal sync pulse train
vsync	Input	Neg.	vertical sync pulse train
clkin	Input	N/A	pixel clock (12.27MHz)
load	Input	Pos.	signal to load a frame
vfield	Input	Neg.	field identification from SONY
cblank	Output	Neg.	composite blank pulse train
vreset	Output	Pos.	reset signal for buffer
ra	Output	Pos.	read line for bank a
rb	Output	Pos.	read line for bank b
wa	Output	Pos.	write line for bank a
wb	Output	Pos.	write line for bank b

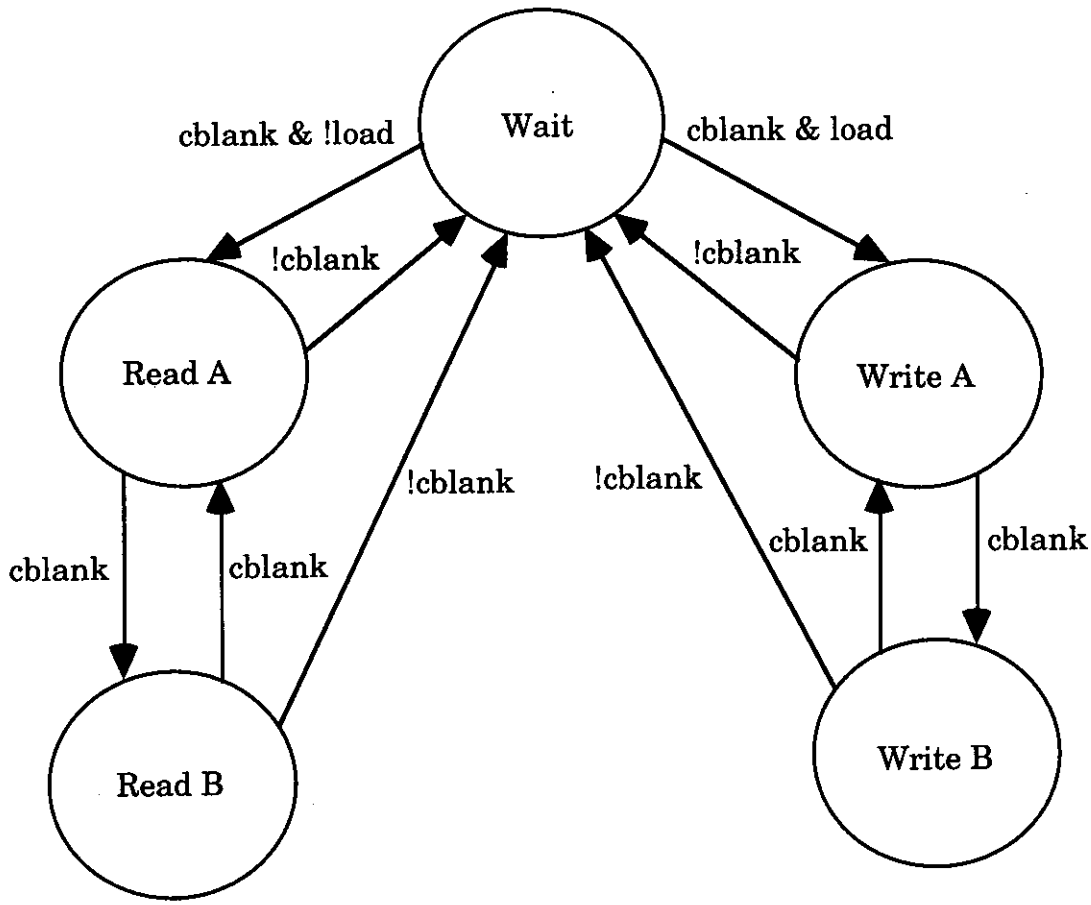


Figure 7 -- Read/Write FSM

The Read/Write FSM issues the signals to read or write bank a or b of the frame buffer. It sits in a wait state until it is in an active video period. It then travels along the other states. When in one of the non-wait states a write or a read signal will be asserted, i.e. when in state Read B, rb will be asserted and bank b will read out data on each clock cycle of SRCLK (the pixel clock). Since it is only activated in the active portion of the video, this will activate the frame buffer only during active video.

The vreset signal is based off of the vfield signal. This occurs only in the vertical blanking period. The field identification and the line counter are used to assure that vreset is asserted at the end/beginning of every frame. Thus the buffer resets both its write and read pointers every frame. Complete code and report files for the controller can be found in Appendix C.

The ALTERA logic has fast propagation delays. It is fast enough that the outputs change before hold times can be satisfied on the 1050 memories. To solve this problem, clkin is a delayed version of the 12.27MHz pixel clock that the rest of the DCE runs on. The delay is 8nsec and is enough to satisfy the hold time on the 1050 chips. This does not really add an asynchronous portion to the DCE, but, only slows down the ALTERA chip so that it appears as if it has a longer propagation delay.

2.4 Compositing:

Compositing is done by muxing the frame buffer data and the live video data on a pixel by pixel basis. The basis for the decision to use the live video or the stored video is a blue screen key. Any pixel matching the color value range of the blue screen key will be dropped and the stored image will be used instead. This is the hardware that is responsible for the effect that the DCE is designed for. The composite image that results from the keying process is a blue screen composite where the foreground image is on the live video feed and the background is the stored video frame. A diagram of the compositing hardware is in figure 8.

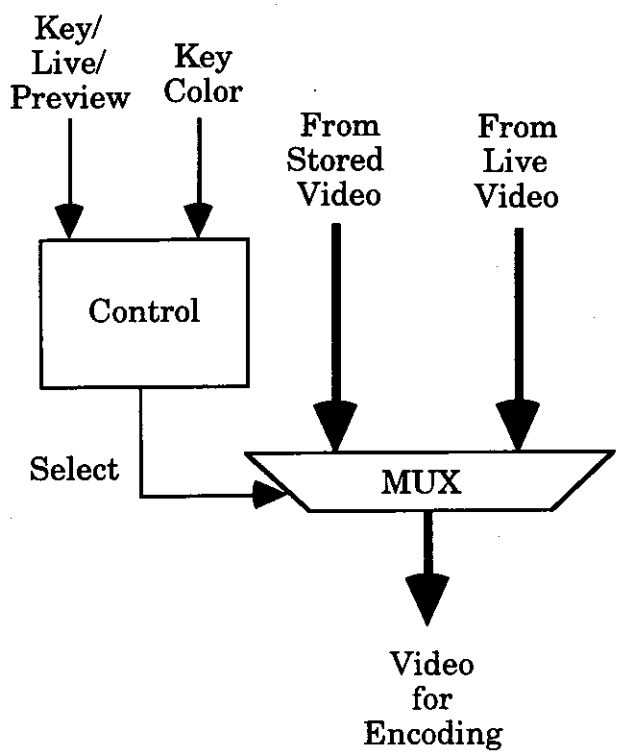


Figure 8 -- Compositing Block Diagram

2.4.1 Muxing:

The actual switching hardware for the DCE composite is a 24 bit 2:1 mux. The live video image is in the A side of the mux and the stored video is in the B side. Selection between the two comes from the mux control hardware. The mux is implemented with six 4 bit 2:1 muxes. The 74F399 (399) is the mux chosen for this project. Logic diagrams for this and all other units are located in Appendix A. The 399 is a clocked mux so that the outputs only change on the rising edge of a clock. The clock used for the 399s is the same pixel clock for the rest of the project.

The operation of the mux requires that the pixels from both the frame buffer and the live video feed be available along with the selection signal before the rising edge of the clock. Since the frame buffer changes to the next pixel only after the clock edge it will have the correct pixel available at the current clock edge of the mux. The live video will have a pixel ready well in advance of the clock edge, as the clock is coming from the SONY module and indicates that a pixel is ready. The selection signal is also designed in the mux control section to arrive in advance of the clock edge.

2.4.2 Mux Control:

The color space being used by the DCE is an RGB space. Proper chroma keying is done in the I-Q color space where chrominance is best described. However, due to the digitizer's use of RGB the DCE is restricted to either

RGB space or performing a complex transform to I-Q space. RGB space can be thought of as a cube of color space with the axes being R, G, and B. This space is viewable as a dimension of 3 - space. The blue screen lies along a plane perpendicular to the farthest point on the blue axis. It also lies along the plane perpendicular to the farthest point on the green axis. The blue screen lies along a small section of the red axis that is close to the origin, but, not quite there. Thus, blue and green only need to be compared as greater than or equal to some comparison values in order to floor them to the described planes (rather than simply comparing them as equal to the planes, some noise is allowed by extending the range a little). The red is compared against two values to determine if it falls between them. This process will slice a volume from the RGB color space, and, inside of this volume is the blue screen color with a noise buffer.

The muxes are controlled by a single selection signal. This signal is produced by the mux control hardware. The mux control is made up of several switches and three 22v10 PALs. The 22v10s are programmed to compare the top 4 bits of the blue, the top four bits of the green, and the top four bits of the red digital data bytes. The comparison is between these 3 nibles (a nible is a 4 bit number) and four other comparison nibles. The 16 DIP switches change 16 of the PAL inputs between high and low digital signals. The blue and green segments are tested to see if they are greater than or equal to their comparison values. The red is compared against two values to see if it is in a certain range, i.e. in between two numbers. This allows a volume of color space that is against the blue plane and green plane and is sized as a rectangle of varying dimensions to be selected as the key for creating the

composite. The result is a key signal that selects the A input to the mux if the video falls outside the key, and, selects the B input if it does not.

Keying is controlled by a set of outside switches. One switch is the key switch. The key switch, when turned to a high input, will direct the PALs to generate a selection signal for the mux based on the inputs from the live video and the DIP switch input, irregardless of other conditions. The other switch alternates between high and low input also and switches viewing modes. Given that the key switch is low so that keying is not occurring, if the view switch is producing a low, then the live video will always be selected from the mux. If it is high under similar conditions, then the stored image will always be selected from the mux. Thus, a live image may be seen undisturbed and a captured image may also be viewed undisturbed. The PAL files for the mux control can be found in Appendix D.

3 Testing and Debugging:

Testing of the DCE was done in stages. As each new section was implemented it was tested, not only to see if it functioned itself, but, also to check that it did not interfere with any other systems on the DCE. Testing started with the micro controlled unit as the video systems will not work without the initialization that takes place via the MCU. Once the ability to load a program and the knowledge that the programs ran as expected, testing went on to the video systems. The encoder was tested first as it could be configured to output a test pattern. The digitizer was then added and tested on the system as a whole. The mux was tested next with a set selection to the live video. After this the frame buffer was added and tested. The mux control was then changed to allow keying and the whole system tested. A fine tuning was also needed for the values that the digitizer uses for hue and saturation. This was done once the rest of the hardware was in place so that any power fluctuations could be compensated for after the power system was fully loaded.

3.1 Micro Controlled Unit Testing:

The testing here was quick and not difficult. Using a design for the 6811 implementation that is well known and used in several other projects saved a lot of time from being spent worrying about correct design. The design was not in question, only the wiring of that design. The wiring was done carefully so that no errors occurred. The only item found in the debugging of this unit was that the outputs can glitch, so the addition of 74F574 registers to calm the glitches was made. The dl program indicated that there was no problem getting the program down loaded to the 6811. Using a digital logic analyzer (DLA) the running of the program could be verified by looking at the outputs on the ports. The use of the DLA also allowed for discovery of errors in the initialization program. Also, the DLA allowed discovery of a faulty 6811 chip which was replaced for a working component.

3.2 Video Systems Testing:

The video systems came next. First a 22090 was put in place and tested. A test program forced the 22090 to emit its own color bar test pattern. This worked, but, a faulty design in the chip prevented other operations from occurring. The 22090, an older version of the 22190, was then traded out for the newer 22190 which is used in the DCE. Once the color bar test pattern was verified on the 22190 it was possible to verify the initialization program from the 6811 as operating properly. It also verified the wiring of the ports on the 6811 to the 74F574s and then to the 22190. The SONY module was then added and video sent to the 22190. This showed that the CLUT on the

22190 was not loading correctly. It also demonstrated that the BYPASS\ pin of the 22190 is not working as stated in the documentation. This is another case of design error on the part of the Raytheon/TRW semiconductor division. After fixing the CLUT loading procedure in the 6811 program, it became apparent that the hue and saturation of the SONY were not quite correct. This leads to the tuning process that occurred at the end of the project. However, the operation of the components involved in the video unit was verified.

3.3 Muxing Testing:

The mux was tested first without the control system in place. The selection line was tied to ground to force the pass through of the live video. During this test it was discovered that the sync signals needed to be delayed for the delay incurred by the muxes. Past that, the muxing section did not effect the rest of the system operation.

3.4 Frame Buffer Testing:

Testing of the frame buffer took a little more time than the rest of the systems. The frame buffer has a complex control system that lends itself well to simulation. The control was simulated and verified in that way. The frame buffer, however, still had an error in it. The vreset signal from the control was not being generated correctly. Thus, the pointers in the memory were being reset every line, and not every frame as intended. This led to vertical striping as the frame buffer repeatedly played out a single line for all

480 lines of the frame. Once this was fixed the frame buffer functioned, but, had a problem with flickering. This was due to a field lock problem in the controller. This was fixed and the frame buffer began to function correctly. This was also verified through the mux, again with a hard switch to select the live feed or the stored image. Compositing was then added to the PAL files and tested showing that the device was functionally correct. The PALs were also tuned at this point to the actual blue screen values. Tuning continued at this point on the hue and saturation of the SONY module.

4 Conclusion:

The DCE was implemented reasonably well with a high degree of quality in the output image. It successfully implemented a blue screen compositing function. The experience from design and debugging the device is necessary and good for upcoming projects. Extensions to this project, discussed in the next section, describe future directions for the DCE project.

5 Extensions:

The DCE is not yet a finished project. There are several directions that the basic platform developed here can go in. Right now the background image must taken from the live video feed. The DCE could be made to load an image from a computer over the serial line interface or an added parallel port. The imagery for the background could also be made to come from a CD-ROM drive with a photo disk (although several photo CD players have video outputs that can easily be plugged into the live video line for the purpose of grabbing a frame). There is also the possibility of storing a series of frames for a moving background image that would simply replay several seconds or more of video. There is also the reverse composite idea where the stored image is the foreground and the background comes from the live feed. There is also the possibility of storing two images and then keying between the two. Any of these options is worthy of investigation for future versions of the DCE project.

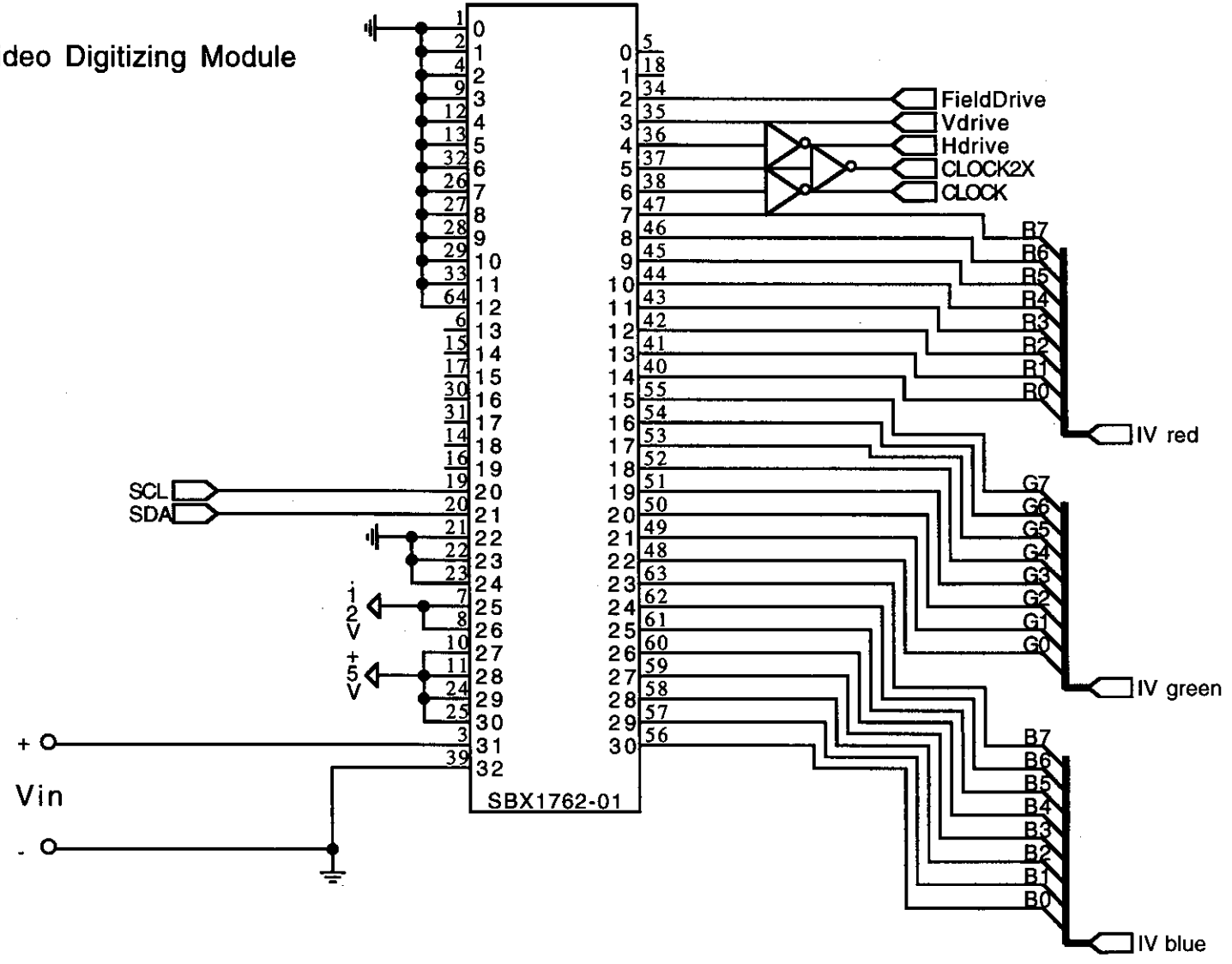
The current version also has more work ahead of it. It still has ground loop problems which interfere with the video quality. It is also not very portable and could be scaled down in size and given a portable, battery operated power supply. Power consumption on the current board averages at 6W. The breakdown of the power consumption between subsystems is: 0.344W for the digitizer, 0.1W for the encoder, 2.1W for the entire frame buffer section, 0.22W for the muxing section, 2.7W for the PAL logic, and 0.625W for the ALTERA control logic. Before the device can be put into a portable format and run from a battery, work must be done to decrease the power

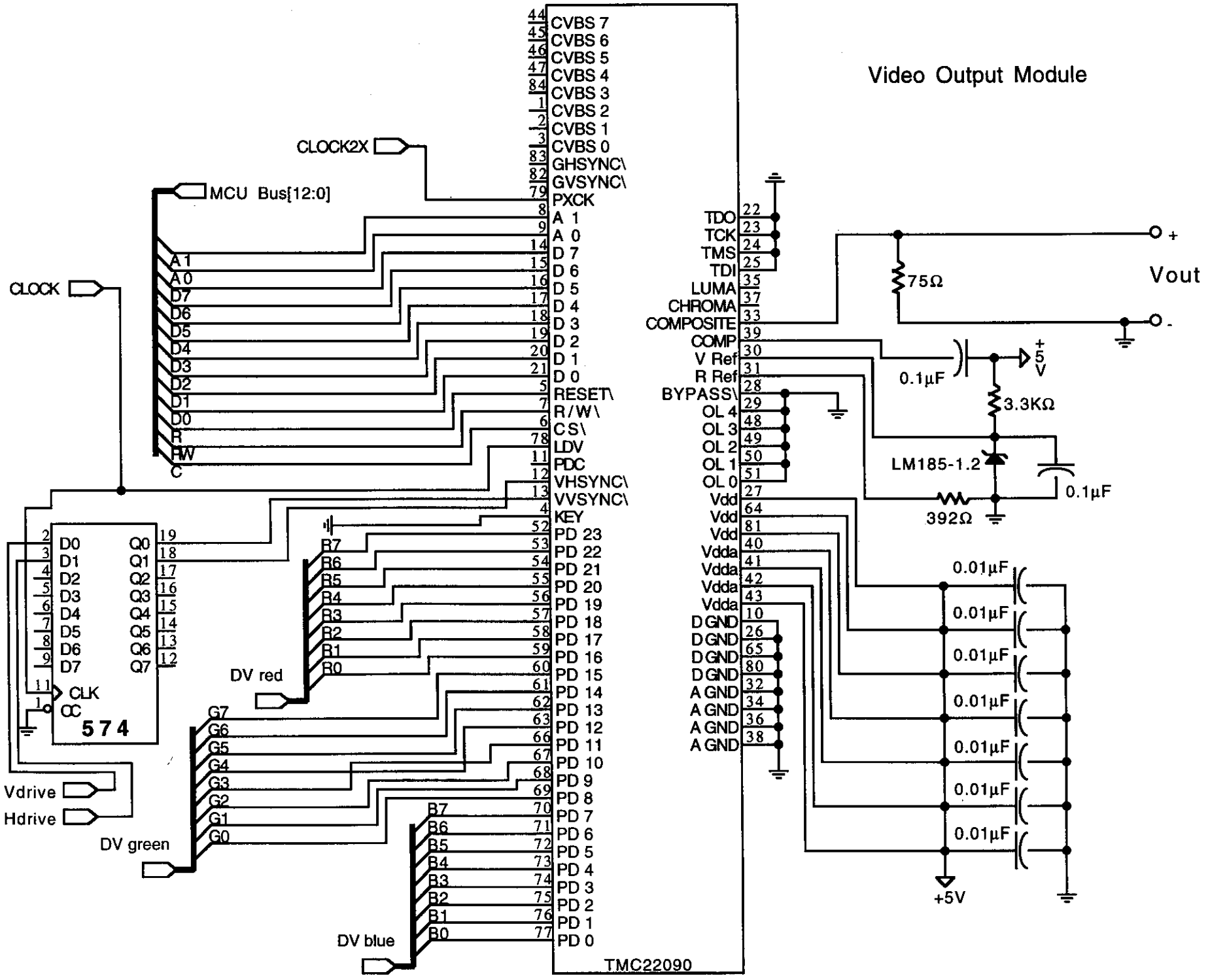
consumption in the PAL logic and the frame buffer memory. Size of the device will also effect portability. Currently the device is approximately 70 square inches. With surface mount technology and a printed circuit board with components on both sides of the board, it should be possible to reduce the device size to 25 inches squared. This not only makes the actual device smaller, but, surface mount components burn less power. Thus, the device gets extra benefits from the power side. Reduction of power consumption using surface mount technology and a little redesign of the power hungry systems will result in a viable device for battery operated use. A starting place for work on the frame buffer is to investigate the use of 18 bit color. This could prove to have good enough quality, while reducing the number of memory chips by two. This would drop the power consumption by 0.35W to 5.65W total.

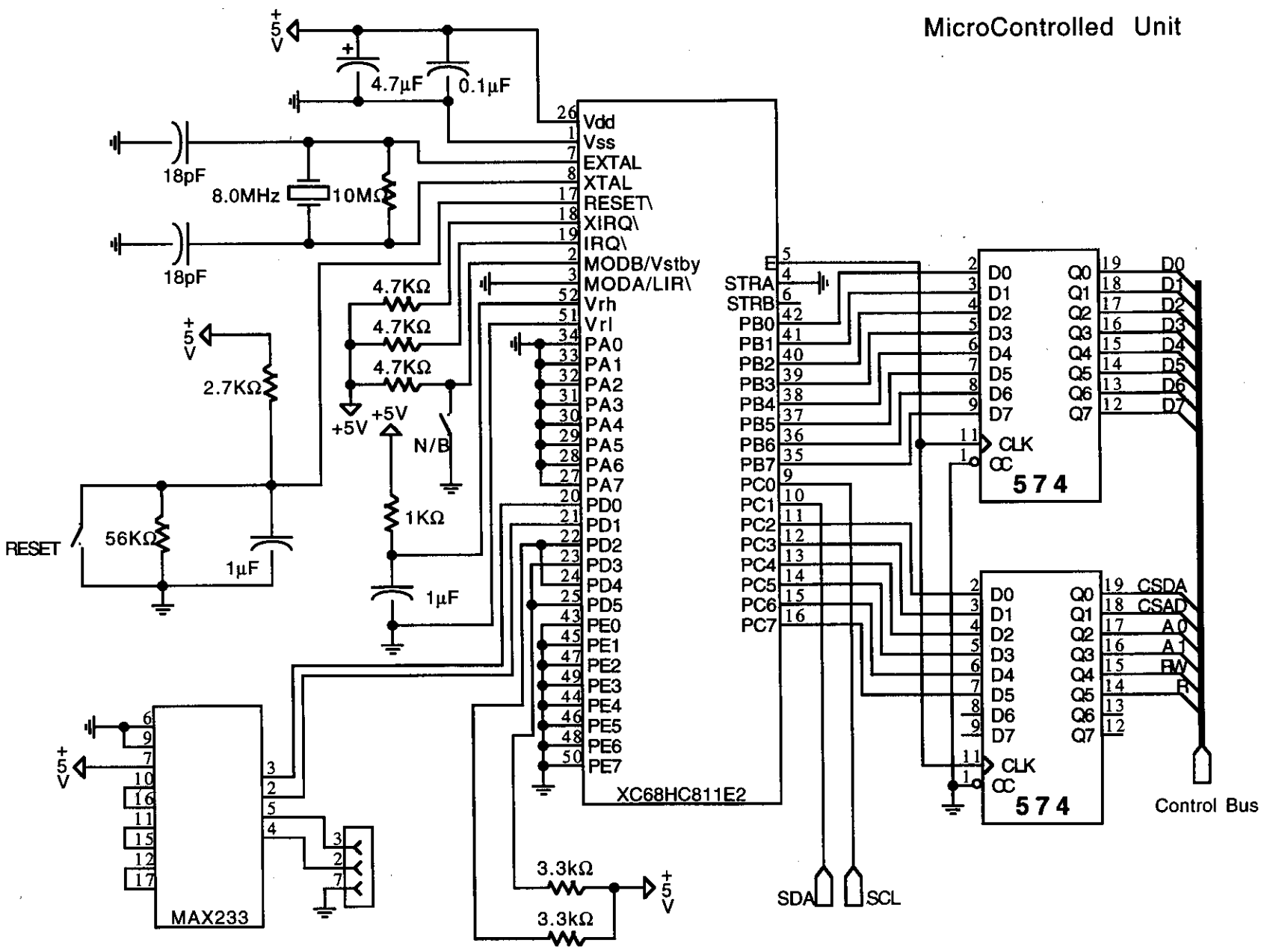
The DCE also could be hooked up to some LCD goggles for viewing purposes and a pen camera used as the live feed. This could yield a totally portable system. The power supply system also needs work so that the power is more evenly applied to the circuit, and cleaner. This will add to the video quality. The DCE also needs to be put into some kind of EMF cage to isolate it from ambient noise. Again, this will add to the final picture quality.

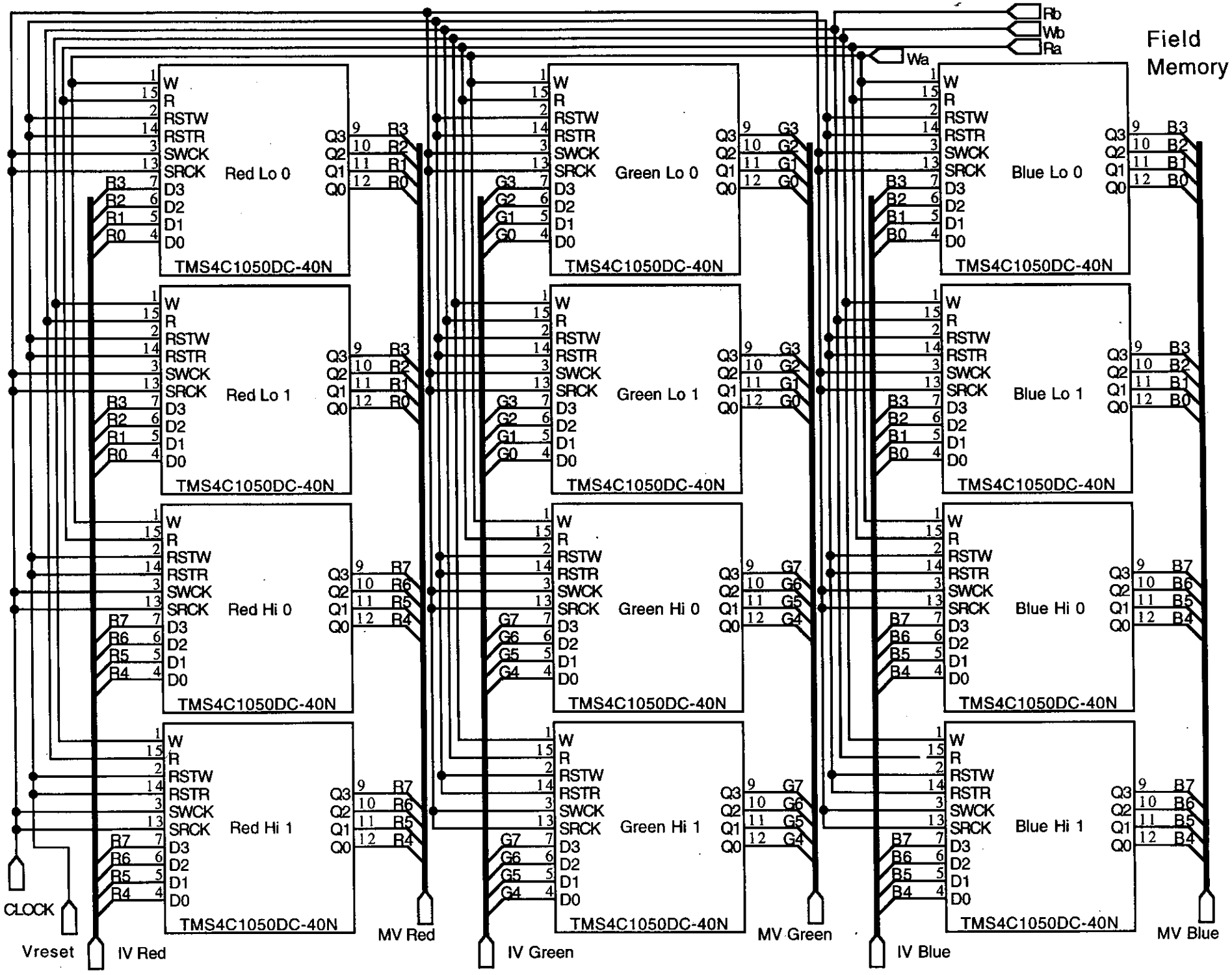
Appendix A -- Logic Diagrams

SONY Video Digitizing Module

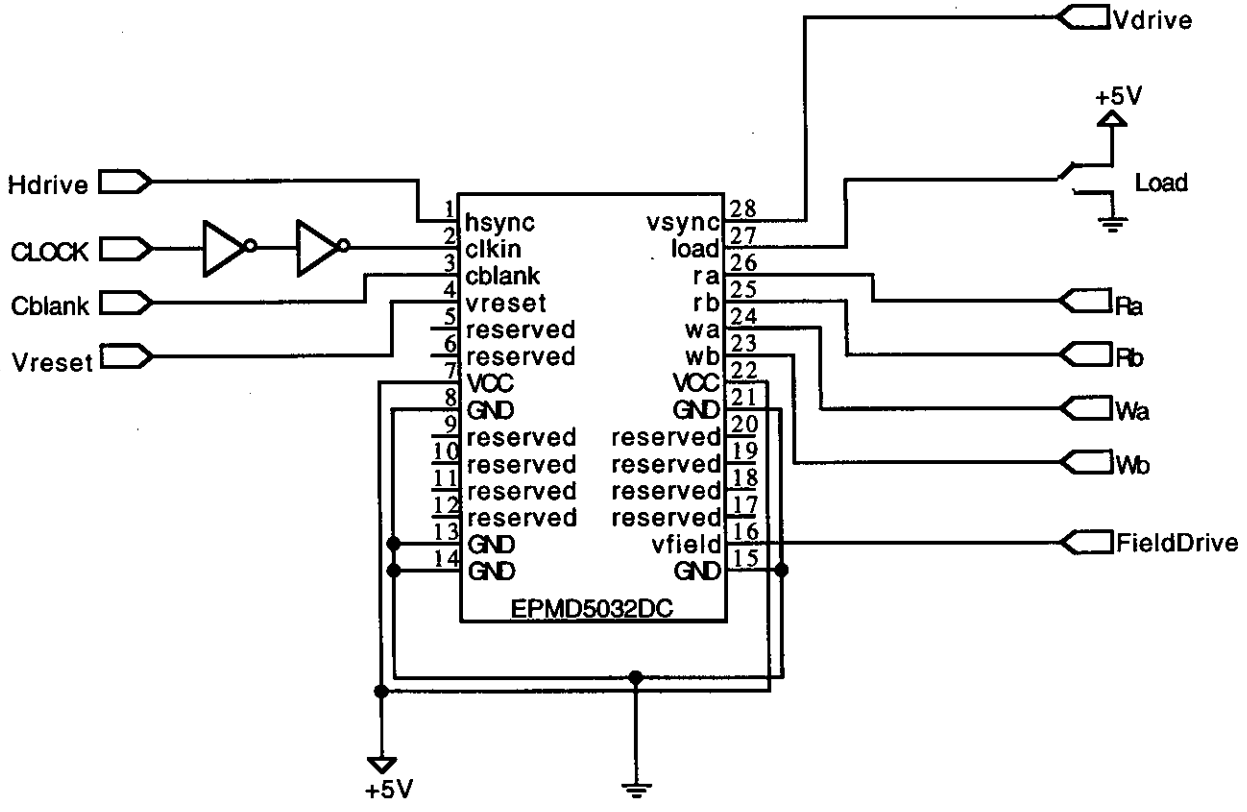



Video Output Module
CLOCK2X
MCU Bus[12:0]
CLOCK
A1
A0
D7
D6
D5
D4
D3
D2
D1
D0
R
W
C
CVBS 7
CVBS 6
CVBS 5
CVBS 4
CVBS 3
CVBS 2
CVBS 1
CVBS 0
GHSYNC\
GVSYNC\
PXCK
A 1
A 0
D 7
D 6
D 5
D 4
D 3
D 2
D 1
D 0
RESET\
R/W\
CS\
LDV
PDC
VHSYNC\
VVSYNC\
KEY
PD 23
PD 22
PD 21
PD 20
PD 19
PD 18
PD 17
PD 16
PD 15
PD 14
PD 13
PD 12
PD 11
PD 10
PD 9
PD 8
PD 7
PD 6
PD 5
PD 4
PD 3
PD 2
PD 1
PD 0
TMC22090
TDO
TCK
TMS
TDI
LUMA
CHROMA
COMPOSITE
COMP
V Ref
R Ref
BYPASS\
OL 4
OL 3
OL 2
OL 1
OL 0
Vdd
Vdd
Vdd
Vdda
Vdda
Vdda
Vdda
DGND
DGND
DGND
DGND
AGND
AGND
AGND
AGND
75Ω
Vout
+
-
0.1µF
+5V
3.3KΩ
LM185-1.2
392Ω
0.1µF
0.01µF
0.01µF
0.01µF
0.01µF
0.01µF
0.01µF
D0
D1
D2
D3
D4
D5
D6
D7
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
CLK
OC
574
Vdrive
Hdrive
DV red
DV green
DV blue
B7
B6
B5
B4
B3
B2
B1
B0
G7
G6
G5
G4
G3
G2
G1
G0

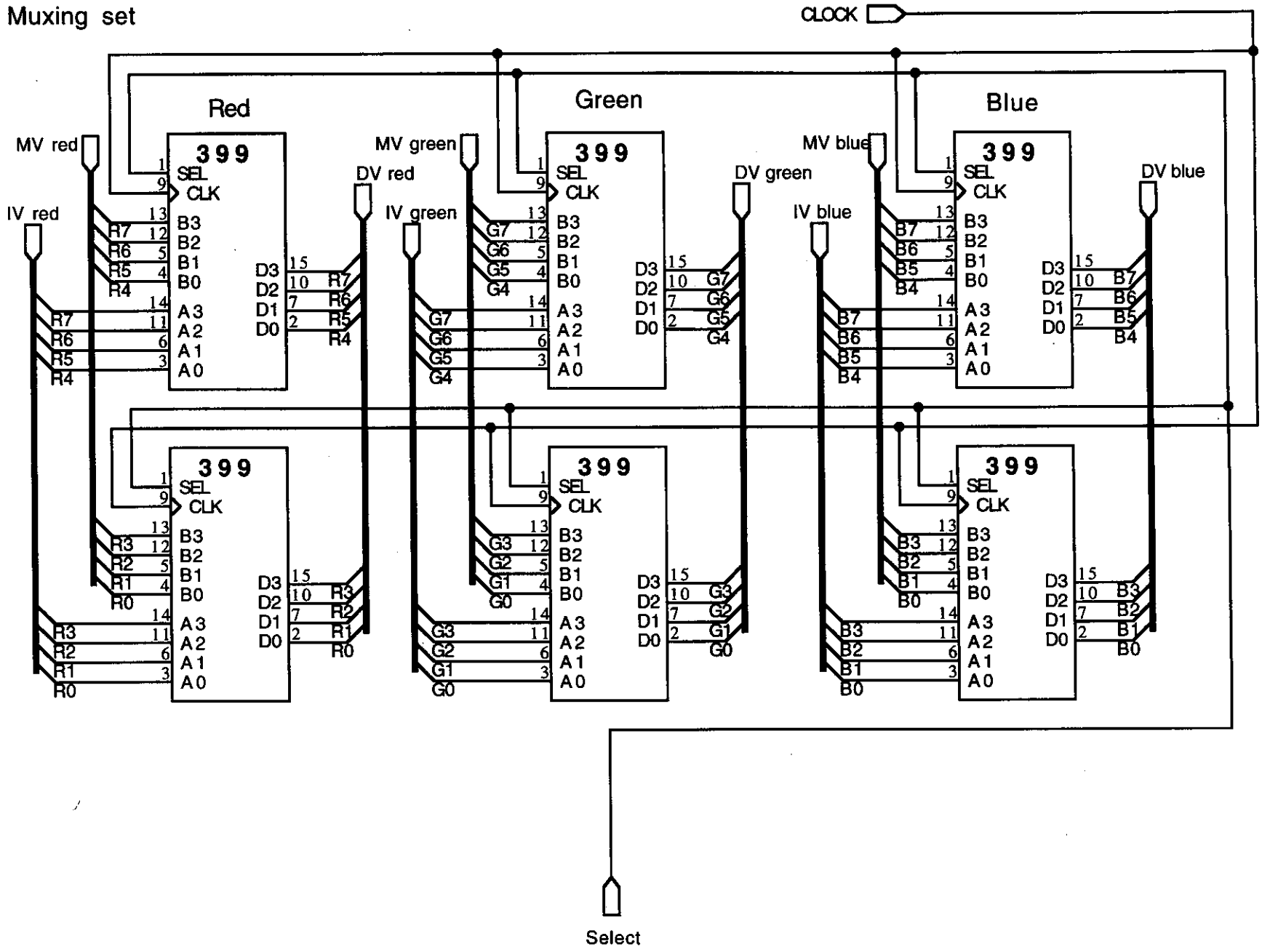

MicroControlled Unit
+5V
4.7µF
0.1µF
18pF
8.0MHz
10MΩ
18pF
4.7KΩ
4.7KΩ
4.7KΩ
2.7KΩ
+5V
+5V
N/B
1KΩ
1µF
RESET
56KΩ
1µF
26 Vdd
1 Vss
7 EXTAL
8 XTAL
17 RESET\
18 XIRQ\
19 IRQ\
2 MODB/Vstby
3 MODA/LIR\
52 Vrh
51 Vrl
34 PA0
33 PA1
32 PA2
31 PA3
30 PA4
29 PA5
28 PA6
27 PA7
20 PD0
21 PD1
22 PD2
23 PD3
24 PD4
25 PD5
43 PE0
45 PE1
47 PE2
49 PE3
44 PE4
46 PE5
48 PE6
50 PE7
XC68HC811E2
E 5
STRA 4
STRB 6
PB0 42
PB1 41
PB2 40
PB3 39
PB4 38
PB5 37
PB6 36
PB7 35
PC0 9
PC1 10
PC2 11
PC3 12
PC4 13
PC5 14
PC6 15
PC7 16
D0 D1 D2 D3 D4 D5 D6 D7
CLK
OC
574
Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7
19 D0
18 D1
17 D2
16 D3
15 D4
14 D5
13 D6
12 D7
19 CSDA
18 CSAD
17 A0
16 A1
15 RW
14 R
Control Bus
MAX233
3.3kΩ
3.3kΩ
+5V
SDA
SCL

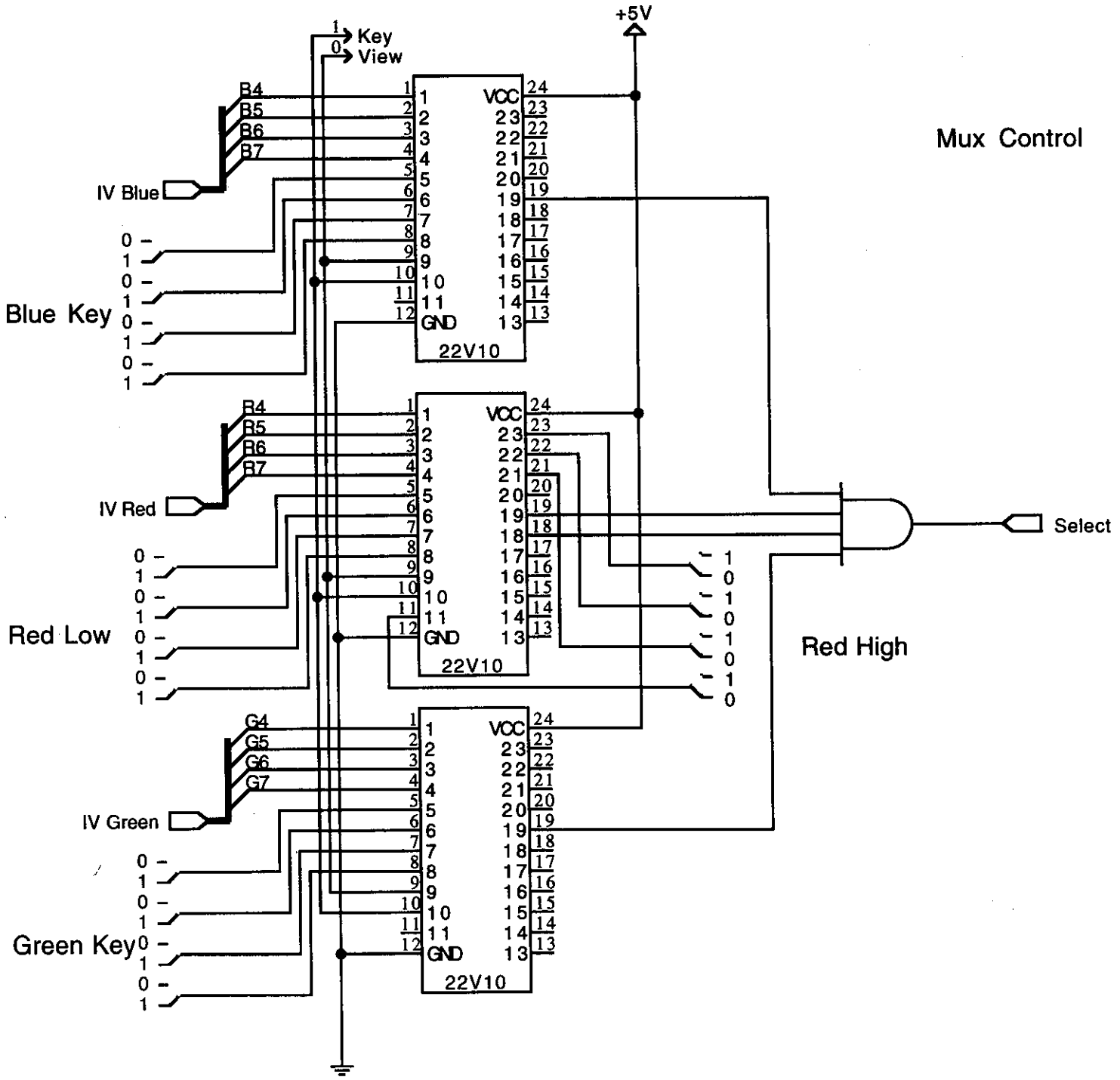

Field Memory
Rb
Wb
Ra
Wa
Red Lo 0
Red Lo 1
Red Hi 0
Red Hi 1
Green Lo 0
Green Lo 1
Green Hi 0
Green Hi 1
Blue Lo 0
Blue Lo 1
Blue Hi 0
Blue Hi 1
TMS4C1050DC-40N
W
R
RSTW
RSTR
SWCK
SRCK
D3
D2
D1
D0
Q3
Q2
Q1
Q0
CLOCK
Vreset
IV Red
MV Red
IV Green
MV Green
IV Blue
MV Blue

Frame Buffer Control



Muxing set


CLOCK
Red
Green
Blue
399
SEL
CLK
B3
B2
B1
B0
A3
A2
A1
A0
D3
D2
D1
D0
MV red
IV red
DV red
MV green
IV green
DV green
MV blue
IV blue
DV blue
Select


1 Key
0 View
+5V
B4
B5
B6
B7
IV Blue
Blue Key
VCC
GND
22V10
Mux Control
R4
R5
R6
R7
IV Red
Red Low
Red High
Select
G4
G5
G6
G7
IV Green
Green Key

Appendix B -- Micro Code

```
%************************************************************************

        Memory controller for the FIFO for the
        Director's Composite Eyeglass

        Phil Barker
        Mar. 31, 1994   (from digitizing control, freezeframe by Wad)
        Apr. 4, 1994    changed the cblank action for negative assert

************************************************************************%

TITLE "Memory Controller, Director's Composite Eyeglass";

DESIGN IS "memctl"
DEVICE IS "EPM5032DC"
        BEGIN           %resource assignment%
        clkin @ 2       : INPUT;
        hsync @ 1       : INPUT;
        vsync @ 28      : INPUT;
        load @ 27       : INPUT;
        vfield @ 16     : INPUT;

        cblank @ 3      : OUTPUT;
        vreset @ 4      : OUTPUT;
        ra @ 26         : OUTPUT;
        rb @ 25         : OUTPUT;
        wa @ 24         : OUTPUT;
        wb @ 23         : OUTPUT;

        END;

SUBDESIGN 'memctl'
(
        clkin           : INPUT;        %12.27 MHz pixel clock%
        vsync           : INPUT;        %vetical sync, active low%
        hsync           : INPUT;        %horizontal sync, active low%
        load            : INPUT;        %load signal%
        vfield          : INPUT;        %field signal%

        vreset          : OUTPUT;       %read/write pointer reset%
        cblank          : OUTPUT;       %composite blanking signal, active low%
        %procxclk       : OUTPUT;%      %processor clock (pixclk/2)%
        ra              : OUTPUT;       %read bank a, active high%
        rb              : OUTPUT;       %read bank b, active high%
        wa              : OUTPUT;       %write bank a, active high%
        wb              : OUTPUT;       %write bank b, active high%
)

VARIABLE
        clkbuf          : SCLK;         %synchronous clock buffer%

        hsync_sync      : DFF;
        real_hsync      : DFF;
        load_reg        : DFF;

        vertical :      MACHINE OF BITS (vstate[1..0])
                                WITH STATES (Vidle, Vblank, Vactive);

        line_ctr[7..0]  : DFF;
        end_of_blank    : NODE;
        end_of_field    : NODE;

        dot_ctr[9..0]   : DFF;
        start_active    : NODE;
        end_active      : NODE;
```

```
        %toggle         : DFF;%
        cblank_reg      : DFF;
        vreset_reg      : DFF;
        ra_reg          : DFF;
        rb_reg          : DFF;
        wa_reg          : DFF;
        wb_reg          : DFF;

        read_write :    MACHINE OF BITS (rstate[2..0])
                        WITH STATES (wait,reada,readb,writea,writeb);

BEGIN
        clkbuf = clkin;

%       toggle = !toggle;
        toggle.clk = clkbuf;
        procxclk = toggle;%

        load_reg.clk = clkbuf;
        load_reg.d = load & (vertical == Vblank);

        %additional syncing for the hsync%

        hsync_sync.d = hsync;
        real_hsync.d = !hsync & hsync_sync;
        hsync_sync.clk = clkbuf;
        real_hsync.clk = clkbuf;

        %vertical state machine%

        vertical.clk = clkbuf;

        CASE (vertical) IS

                WHEN Vidle =>
                        IF (!vsync) THEN
                                vertical = Vblank;
                        ELSE
                                vertical = Vidle;
                        END IF;

                WHEN Vblank =>
                        IF (end_of_blank) THEN
                                vertical = Vactive;
                        ELSE
                                vertical = Vblank;
                        END IF;

                WHEN Vactive =>
                        IF (!vsync) THEN
                                vertical = Vblank;
                        ELSIF (end_of_field) THEN
                                vertical = Vidle;
                        ELSE
                                vertical = Vactive;
                        END IF;

                END CASE;       %vertical%

line_ctr[].clk = clkbuf;
line_ctr[] = ((line_ctr[] + 1) & (vertical == Vblank)
                        & real_hsync & !end_of_blank) #
                (line_ctr[] & (vertical == Vblank) & !real_hsync) #
                ((line_ctr[] + 1) & (vertical == Vactive) & real_hsync) #
                (line_ctr[] & (vertical == Vactive) & !real_hsync);
```

```
end_of_blank = (line_ctr[] == 19);
end_of_field = (line_ctr[] == 255);

dot_ctr[].clk = clkbuf;
dot_ctr[] = (dot_ctr[] + 1) & !real_hsync;

start_active = (dot_ctr[] == 100);
end_active = (dot_ctr[] == 770);

cblank_reg = (start_active & !cblank_reg & (vertical == Vactive)) #
             (!end_active & !real_hsync & cblank_reg);
cblank_reg.clk = clkbuf;
cblank = cblank_reg;


        read_write.clk = clkbuf;

        CASE (read_write) IS

                WHEN wait =>
                        IF (cblank & !load_reg) THEN
                                read_write = reada;
                        ELSIF (cblank & load_reg) THEN
                                read_write = writea;
                        ELSE
                                read_write = wait;
                        END IF;

                WHEN reada =>
                        IF (cblank) THEN
                                read_write = readb;
                        ELSE
                                read_write = wait;
                        END IF;

                WHEN readb =>
                        IF (cblank) THEN
                                read_write = reada;
                        ELSE
                                read_write = wait;
                        END IF;

                WHEN writea =>
                        IF (cblank) THEN
                                read_write = writeb;
                        ELSE
                                read_write = wait;
                        END IF;

                WHEN writeb =>
                        IF (cblank) THEN
                                read_write = writea;
                        ELSE
                                read_write = wait;
                        END IF;

                END CASE;       %read_write%


vreset_reg = (line_ctr[] == 2) & vfield;
vreset_reg.clk = clkbuf;
vreset = vreset_reg;

ra_reg = (read_write == reada);
ra_reg.clk = clkbuf;
```

```
rb_reg = (read_write == readb);
rb_reg.clk = clkbuf;
rb = rb_reg;

wa_reg = (read_write == writea);
wa_reg.clk = clkbuf;
wa = wa_reg;

wb_reg = (read_write == writeb);
wb_reg.clk = clkbuf;
wb = wb_reg;

END;
```

Project Information d:\users\phil\memctl.rpt

MAX+plus II Compiler Report File
Version 4.01 2/07/94
Compiled: 04/13/94 20:37:32

***** Project compilation was successful

Memory Controller, Director's Composite Eyeglass

** DEVICE SUMMARY **

Chip/ POF	Device	Input Pins	Output Pins	Bidir Pins	LCs	Shareable Expanders	% Utilized
memctl	EPM5032DC	5	6	0	32	3	100%
User Pins:		5	6	0			

** PIN/LC/CHIP ASSIGNMENTS **

User Assignments	Actual Assignments (if different)	Node Name
memctl@3		cblank
memctl@2		clkin
memctl@1		hsync
memctl@27		load
memctl@26		ra
memctl@25		rb
memctl@16		vfield
memctl@4		vreset
memctl@28		vsync
memctl@24		wa
memctl@23		wb

** STATE MACHINE ASSIGNMENTS **

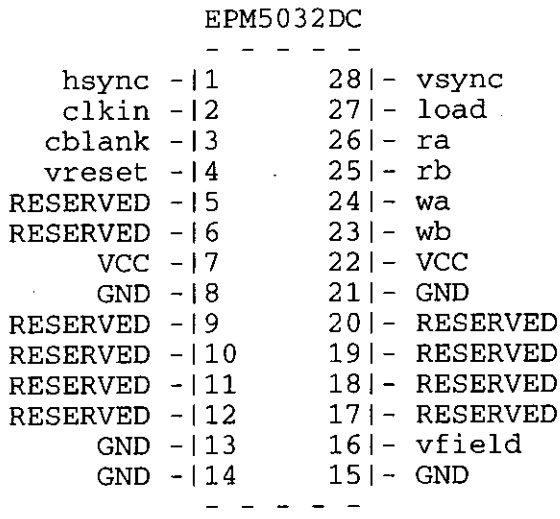
```
vertical: MACHINE
        OF BITS (
           vstate1,
           vstate0
        )
        WITH STATES (
                          Vidle = B"00",
                         Vblank = B"11",
                        Vactive = B"01"
);


read_write: MACHINE
        OF BITS (
           rstate2,
           rstate1,
           rstate0
        )
        WITH STATES (
                           wait = B"000",
                          reada = B"001",
                          readb = B"011",
                         writea = B"010",
                         writeb = B"100"
);
```

Device-Specific Information: d:\users\phil\memctl.rpt
memctl

***** Logic for device 'memctl' compiled without errors.

Device: EPM5032DC
Security: OFF



N.C. = Not Connected.
VCC = Dedicated power pin, which MUST be connected to VCC.
GND = Dedicated ground pin or unused dedicated input, which MUST be connected to GND
RESERVED = Unused I/O pin, which MUST be left unconnected.

** RESOURCE USAGE **

Logic Array Block		Logic Cells	I/O Pins	Shareable Expanders
A:	LC1 - LC32	32/32(100%)	6/16(37%)	3/64(4%)

Total dedicated input pins used:	5/ 8 (62%)
Total I/O pins used:	6/ 16 (37%)
Total logic cells used:	32/ 32 (100%)
Total shareable expanders used:	3/ 64 (4%)
Total shareable expanders not available (n/a):	0/ 64 (0%)
Total input pins required:	5
Total output pins required:	6
Total bidirectional pins required:	0
Total logic cells required:	32
Total flipflops required:	32
Total shareable expanders in database:	3
Synthesized logic cells:	0/ 32 (0%)

** INPUTS **

Pin	LC	LAB	Primitive	Shareable Expanders Total	Shareable Expanders Shared	Shareable Expanders n/a	Fan-In INP	Fan-In FBK	Name
2	-	-	INPUT	0	0	0	0	0	clkin
1	-	-	INPUT	0	0	0	0	0	hsync
27	-	-	INPUT	0	0	0	0	0	load
16	-	-	INPUT	0	0	0	0	0	vfield
28	-	-	INPUT	0	0	0	0	0	vsync

s = Synthesized pin or logic cell
+ = Synchronous flipflop
! = Not gate push-back

** OUTPUTS **

Pin	LC	LAB	Primitive	Shareable Expanders Total	Shared	n/a	Fan-In INP	FBK	Name
3	1	A	FF+	0	0	0	0	13	cblank
26	31	A	FF+	0	0	0	0	3	ra
25	29	A	FF+	0	0	0	0	3	rb
4	3	A	FF+	0	0	0	1	8	vreset
24	27	A	FF+	0	0	0	0	3	wa
23	25	A	FF+	0	0	0	0	3	wb

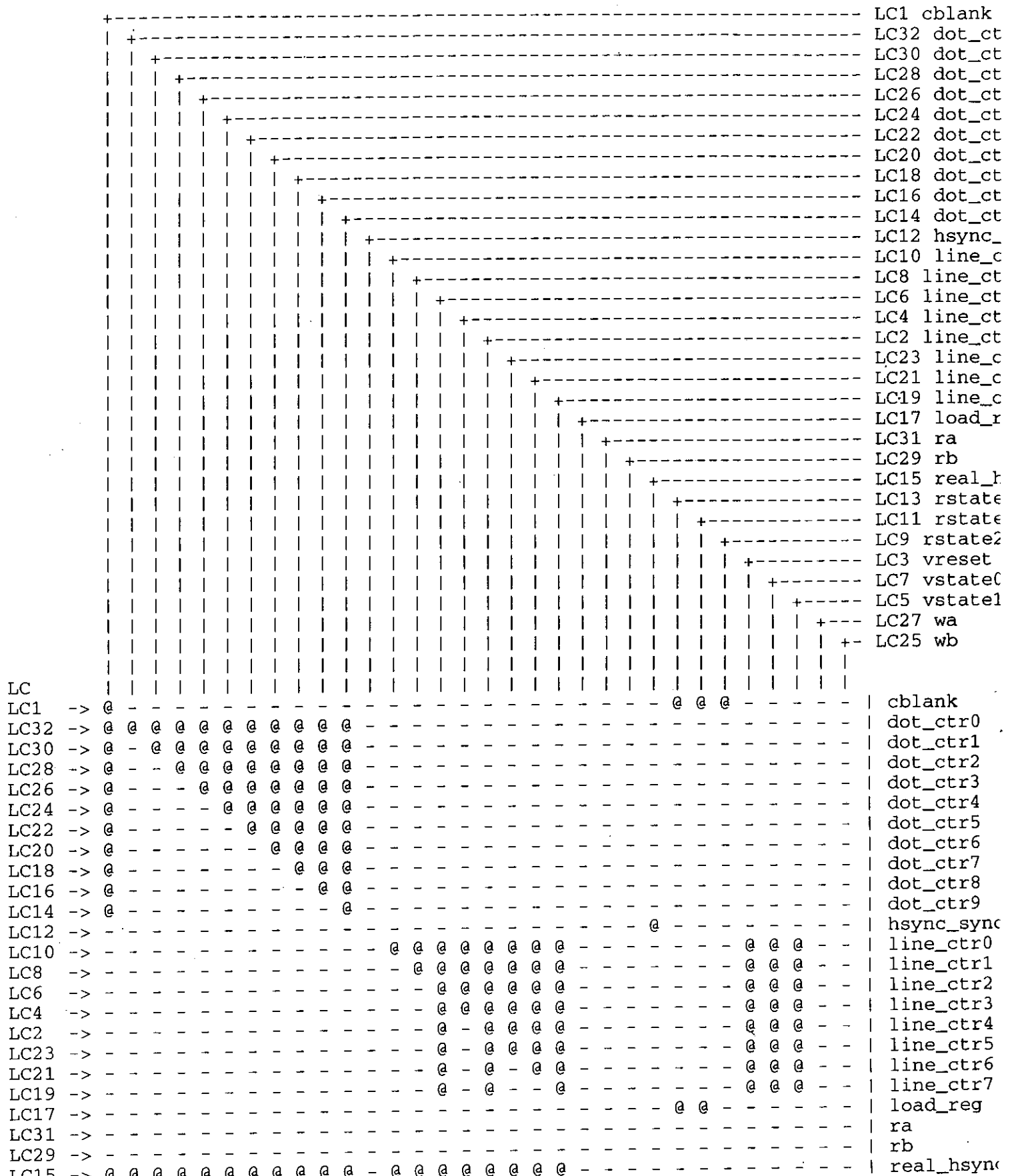
s = Synthesized pin or logic cell
+ = Synchronous flipflop
! = Not gate push-back

** BURIED LOGIC **

Pin	LC	LAB	Primitive	Shareable Expanders Total	Shared	n/a	Fan-In INP	FBK	Name
-	32	A	DFF+	0	0	0	0	1	dot_ctr0
-	30	A	DFF+	0	0	0	0	2	dot_ctr1
-	28	A	DFF+	0	0	0	0	3	dot_ctr2
-	26	A	DFF+	0	0	0	0	4	dot_ctr3
-	24	A	DFF+	0	0	0	0	5	dot_ctr4
-	22	A	DFF+	0	0	0	0	6	dot_ctr5
-	20	A	DFF+	0	0	0	0	7	dot_ctr6
-	18	A	DFF+	0	0	0	0	8	dot_ctr7
-	16	A	DFF+	0	0	0	0	9	dot_ctr8
-	14	A	DFF+	0	0	0	0	10	dot_ctr9
-	12	A	DFF+	0	0	0	1	0	hsync_sync
-	10	A	DFF+	0	0	0	0	2	line_ctr0
-	8	A	DFF+	0	0	0	0	3	line_ctr1
-	6	A	DFF+	1	1	0	0	10	line_ctr2
-	4	A	DFF+	0	0	0	0	5	line_ctr3
-	2	A	DFF+	3	1	0	0	10	line_ctr4
(20)	23	A	DFF+	0	0	0	0	7	line_ctr5
(19)	21	A	DFF+	0	0	0	0	8	line_ctr6
(18)	19	A	DFF+	0	0	0	0	9	line_ctr7
(17)	17	A	DFF+	0	0	0	1	2	load_reg
(12)	15	A	DFF+	0	0	0	1	1	real_hsync
(11)	13	A	DFF+	0	0	0	0	4	rstate0
(10)	11	A	DFF+	0	0	0	0	4	rstate1
(9)	9	A	DFF+	0	0	0	0	3	rstate2
(6)	7	A	DFF+	0	0	0	1	9	vstate0
(5)	5	A	DFF+	0	0	0	1	9	vstate1

s = Synthesized pin or logic cell
+ = Synchronous flipflop
! = Not gate push-back

** LOGIC CELL INTERCONNECTIONS **



```
LC9   -> - - - - - - - - - - - - - - - - - - - - - - - - @ @ - @ @ @ - - - @ @ | rstate2
LC3   -> - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - | vreset
LC7   -> @ - - - - - - - - - - - - @ @ @ @ @ @ @ @ - - - - - - - - @ @ - - | vstate0
LC5   -> @ - - - - - - - - - - - - - - @ - @ - - - @ - - - - - - - - @ @ - - | vstate1
LC27  -> - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - | wa
LC25  -> - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - | wb

Pin
2     -> - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - | clkin
1     -> - - - - - - - - - - - @ - - - - - - - - - - - - @ - - - - - - - - - | hsync
27    -> - - - - - - - - - - - - - - - - - - - - - - - - @ - - - - - - - - - | load
16    -> - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - @ - - - - | vfield
28    -> - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - @ @ - - | vsync
```

* = The logic cell or pin is an input to the logic cell (or LAB) through the PIA.
@ = The input pin or logic cell feeds the logic cell (or LAB) using direct interconn
\- = The logic cell or pin is not an input to the logic cell (or LAB).

** EQUATIONS **

```
clkin    : INPUT;
hsync    : INPUT;
load     : INPUT;
vfield   : INPUT;
vsync    : INPUT;

% cblank_reg = _LC001 from file "memctl.tdf" line 68, column 2 %
 cblank  = DFF( _EQ001 $  cblank, GLOBAL( clkin),  VCC,  VCC);
  _EQ001 = !cblank & !dot_ctr0 & !dot_ctr1 &  dot_ctr2 & !dot_ctr3 &
            !dot_ctr4 &  dot_ctr5 &  dot_ctr6 & !dot_ctr7 & !dot_ctr8 &
            !dot_ctr9 &  vstate0 & !vstate1
         #  cblank & !dot_ctr0 &  dot_ctr1 & !dot_ctr2 & !dot_ctr3 &
            !dot_ctr4 & !dot_ctr5 & !dot_ctr6 & !dot_ctr7 &  dot_ctr8 &
             dot_ctr9
         #  cblank &  real_hsync;

% dot_ctr0 = _LC032 from file "memctl.tdf" line 63, column 9 %
dot_ctr0 = DFF( _EQ002 $  GND, GLOBAL( clkin),  VCC,  VCC);
  _EQ002 = !dot_ctr0 & !real_hsync;

% dot_ctr1 = _LC030 from file "memctl.tdf" line 63, column 9 %
dot_ctr1 = DFF( _EQ003 $  GND, GLOBAL( clkin),  VCC,  VCC);
  _EQ003 = !dot_ctr0 &  dot_ctr1 & !real_hsync
         #  dot_ctr0 & !dot_ctr1 & !real_hsync;

% dot_ctr2 = _LC028 from file "memctl.tdf" line 63, column 9 %
dot_ctr2 = DFF( _EQ004 $  GND, GLOBAL( clkin),  VCC,  VCC);
  _EQ004 =  dot_ctr0 &  dot_ctr1 & !dot_ctr2 & !real_hsync
         # !dot_ctr0 &  dot_ctr2 & !real_hsync
         # !dot_ctr1 &  dot_ctr2 & !real_hsync;

% dot_ctr3 = _LC026 from file "memctl.tdf" line 63, column 9 %
dot_ctr3 = DFF( _EQ005 $  dot_ctr3, GLOBAL( clkin),  VCC,  VCC);
  _EQ005 =  dot_ctr0 &  dot_ctr1 &  dot_ctr2 & !dot_ctr3 & !real_hsync
         #  dot_ctr0 &  dot_ctr1 &  dot_ctr2 &  dot_ctr3
         #  dot_ctr3 &  real_hsync;

% dot_ctr4 = _LC024 from file "memctl.tdf" line 63, column 9 %
dot_ctr4 = DFF( _EQ006 $  dot_ctr4, GLOBAL( clkin),  VCC,  VCC);
  _EQ006 =  dot_ctr0 &  dot_ctr1 &  dot_ctr2 &  dot_ctr3 & !dot_ctr4 &
             !real_hsync
         #  dot_ctr0 &  dot_ctr1 &  dot_ctr2 &  dot_ctr3 &  dot_ctr4
         #  dot_ctr4 &  real_hsync;

% dot_ctr5 = _LC022 from file "memctl.tdf" line 63, column 9 %
dot_ctr5 = DFF( _EQ007 $  dot_ctr5, GLOBAL( clkin),  VCC,  VCC);
  _EQ007 =  dot_ctr0 &  dot_ctr1 &  dot_ctr2 &  dot_ctr3 &  dot_ctr4 &
             !dot_ctr5 & !real_hsync
         #  dot_ctr0 &  dot_ctr1 &  dot_ctr2 &  dot_ctr3 &  dot_ctr4 &
              dot_ctr5
         #  dot_ctr5 &  real_hsync;

% dot_ctr6 = _LC020 from file "memctl.tdf" line 63, column 9 %
dot_ctr6 = DFF( _EQ008 $  dot_ctr6, GLOBAL( clkin),  VCC,  VCC);
  _EQ008 =  dot_ctr0 &  dot_ctr1 &  dot_ctr2 &  dot_ctr3 &  dot_ctr4 &
              dot_ctr5 & !dot_ctr6 & !real_hsync
         #  dot_ctr0 &  dot_ctr1 &  dot_ctr2 &  dot_ctr3 &  dot_ctr4 &
              dot_ctr5 &  dot_ctr6
         #  dot_ctr6 &  real_hsync;
```

```
% dot_ctr7 = _LC018 from file "memctl.tdf" line 63, column 9 %
dot_ctr7 = DFF( _EQ009 $  dot_ctr7, GLOBAL( clkin),  VCC,  VCC);
  _EQ009 =  dot_ctr0 &  dot_ctr1 &  dot_ctr2 &  dot_ctr3 &  dot_ctr4 &
            dot_ctr5 &  dot_ctr6 & !dot_ctr7 & !real_hsync
         #  dot_ctr0 &  dot_ctr1 &  dot_ctr2 &  dot_ctr3 &  dot_ctr4 &
            dot_ctr5 &  dot_ctr6 &  dot_ctr7
         #  dot_ctr7 &  real_hsync;

% dot_ctr8 = _LC016 from file "memctl.tdf" line 63, column 9 %
dot_ctr8 = DFF( _EQ010 $  dot_ctr8, GLOBAL( clkin),  VCC,  VCC);
  _EQ010 =  dot_ctr0 &  dot_ctr1 &  dot_ctr2 &  dot_ctr3 &  dot_ctr4 &
            dot_ctr5 &  dot_ctr6 &  dot_ctr7 & !dot_ctr8 & !real_hsync
         #  dot_ctr0 &  dot_ctr1 &  dot_ctr2 &  dot_ctr3 &  dot_ctr4 &
            dot_ctr5 &  dot_ctr6 &  dot_ctr7 &  dot_ctr8
         #  dot_ctr8 &  real_hsync;

% dot_ctr9 = _LC014 from file "memctl.tdf" line 63, column 9 %
dot_ctr9 = DFF( _EQ011 $  dot_ctr9, GLOBAL( clkin),  VCC,  VCC);
  _EQ011 =  dot_ctr0 &  dot_ctr1 &  dot_ctr2 &  dot_ctr3 &  dot_ctr4 &
            dot_ctr5 &  dot_ctr6 &  dot_ctr7 &  dot_ctr8 & !dot_ctr9 &
           !real_hsync
         #  dot_ctr0 &  dot_ctr1 &  dot_ctr2 &  dot_ctr3 &  dot_ctr4 &
            dot_ctr5 &  dot_ctr6 &  dot_ctr7 &  dot_ctr8 &  dot_ctr9
         #  dot_ctr9 &  real_hsync;

% hsync_sync = _LC012 from file "memctl.tdf" line 52, column 2 %
hsync_sync = DFF( hsync $  GND, GLOBAL( clkin),  VCC,  VCC);

% line_ctr0 = _LC010 from file "memctl.tdf" line 59, column 10 %
line_ctr0 = DFF( _EQ012 $  GND, GLOBAL( clkin),  VCC,  VCC);
  _EQ012 =  line_ctr0 & !real_hsync &  vstate0
         # !line_ctr0 &  real_hsync &  vstate0;

% line_ctr1 = _LC008 from file "memctl.tdf" line 59, column 10 %
line_ctr1 = DFF( _EQ013 $  GND, GLOBAL( clkin),  VCC,  VCC);
  _EQ013 = !line_ctr0 &  line_ctr1 &  real_hsync &  vstate0
         #  line_ctr0 & !line_ctr1 &  real_hsync &  vstate0
         #  line_ctr1 & !real_hsync &  vstate0;

% line_ctr2 = _LC006 from file "memctl.tdf" line 59, column 10 %
line_ctr2 = DFF( _EQ014 $  vstate0, GLOBAL( clkin),  VCC,  VCC);
  _EQ014 =  line_ctr0 &  line_ctr1 & !line_ctr3 &  line_ctr4 & !line_ctr5 &
           !line_ctr6 & !line_ctr7 &  real_hsync &  vstate0 &  vstate1
         #  line_ctr0 &  line_ctr1 &  line_ctr2 &  real_hsync &  vstate0
         # !line_ctr2 &  vstate0 &  _X001;
  _X001  = EXP( line_ctr0 &  line_ctr1 &  real_hsync);

% line_ctr3 = _LC004 from file "memctl.tdf" line 59, column 10 %
line_ctr3 = DFF( _EQ015 $  line_ctr3, GLOBAL( clkin),  VCC,  VCC);
  _EQ015 =  line_ctr0 &  line_ctr1 &  line_ctr2 & !line_ctr3 &  real_hsync &
            vstate0
         #  line_ctr0 &  line_ctr1 &  line_ctr2 &  line_ctr3 &  real_hsync
         #  line_ctr3 & !vstate0;

% line_ctr4 = _LC002 from file "memctl.tdf" line 59, column 10 %
line_ctr4 = DFF( _EQ016 $  _EQ017, GLOBAL( clkin),  VCC,  VCC);
  _EQ016 = !line_ctr2 &  line_ctr4 &  vstate0 &  _X002
         # !line_ctr3 &  line_ctr4 &  vstate0 &  _X003
         #  line_ctr4 &  vstate0 &  _X001;
  _X002  = EXP(!line_ctr3 & !line_ctr5 & !line_ctr6 & !line_ctr7);
  _X003  = EXP(!line_ctr2 &  vstate1);
  _EQ017 =  line_ctr0 &  line_ctr1 &  line_ctr2 &  line_ctr3 & !line_ctr4 &
            real_hsync &  vstate0;

% line_ctr5 = _LC023 from file "memctl.tdf" line 59, column 10 %
```

```
  _EQ018 =  line_ctr0 &  line_ctr1 &  line_ctr2 &  line_ctr3 &  line_ctr4 &
             !line_ctr5 &  real_hsync &  vstate0
         #  line_ctr0 &  line_ctr1 &  line_ctr2 &  line_ctr3 &  line_ctr4 &
              line_ctr5 &  real_hsync
         #  line_ctr5 & !vstate0;

% line_ctr6 = _LC021 from file "memctl.tdf" line 59, column 10 %
line_ctr6 = DFF( _EQ019 $  line_ctr6, GLOBAL( clkin),  VCC,  VCC);
  _EQ019 =  line_ctr0 &  line_ctr1 &  line_ctr2 &  line_ctr3 &  line_ctr4 &
              line_ctr5 & !line_ctr6 &  real_hsync &  vstate0
         #  line_ctr0 &  line_ctr1 &  line_ctr2 &  line_ctr3 &  line_ctr4 &
              line_ctr5 &  line_ctr6 &  real_hsync
         #  line_ctr6 & !vstate0;

% line_ctr7 = _LC019 from file "memctl.tdf" line 59, column 10 %
line_ctr7 = DFF( _EQ020 $  line_ctr7, GLOBAL( clkin),  VCC,  VCC);
  _EQ020 =  line_ctr0 &  line_ctr1 &  line_ctr2 &  line_ctr3 &  line_ctr4 &
              line_ctr5 &  line_ctr6 & !line_ctr7 &  real_hsync &  vstate0
         #  line_ctr0 &  line_ctr1 &  line_ctr2 &  line_ctr3 &  line_ctr4 &
              line_ctr5 &  line_ctr6 &  line_ctr7 &  real_hsync
         #  line_ctr7 & !vstate0;

% load_reg = _LC017 from file "memctl.tdf" line 54, column 2 %
load_reg = DFF( _EQ021 $  GND, GLOBAL( clkin),  VCC,  VCC);
  _EQ021 =  load &  vstate0 &  vstate1;

% ra_reg   = _LC031 from file "memctl.tdf" line 70, column 2 %
 ra      = DFF( _EQ022 $  GND, GLOBAL( clkin),  VCC,  VCC);
  _EQ022 =  rstate0 & !rstate1 & !rstate2;

% rb_reg   = _LC029 from file "memctl.tdf" line 71, column 2 %
 rb      = DFF( _EQ023 $  GND, GLOBAL( clkin),  VCC,  VCC);
  _EQ023 =  rstate0 &  rstate1 & !rstate2;

% real_hsync = _LC015 from file "memctl.tdf" line 53, column 2 %
real_hsync = DFF( _EQ024 $  GND, GLOBAL( clkin),  VCC,  VCC);
  _EQ024 = !hsync &  hsync_sync;

% rstate0  = _LC013 from file "memctl.tdf" line 75, column 38 %
rstate0  = DFF( _EQ025 $  GND, GLOBAL( clkin),  VCC,  VCC);
  _EQ025 =  cblank & !load_reg & !rstate1 & !rstate2
         #  cblank &  rstate0 & !rstate2;

% rstate1  = _LC011 from file "memctl.tdf" line 75, column 38 %
rstate1  = DFF( _EQ026 $  GND, GLOBAL( clkin),  VCC,  VCC);
  _EQ026 =  cblank &  load_reg & !rstate1 & !rstate2
         #  cblank & !rstate0 & !rstate1 &  rstate2
         #  cblank &  rstate0 & !rstate1 & !rstate2;

% rstate2  = _LC009 from file "memctl.tdf" line 75, column 38 %
rstate2  = DFF( _EQ027 $  GND, GLOBAL( clkin),  VCC,  VCC);
  _EQ027 =  cblank & !rstate0 &  rstate1 & !rstate2;

% vreset_reg = _LC003 from file "memctl.tdf" line 69, column 2 %
 vreset  = DFF( _EQ028 $  GND, GLOBAL( clkin),  VCC,  VCC);
  _EQ028 = !line_ctr0 &  line_ctr1 & !line_ctr2 & !line_ctr3 & !line_ctr4 &
             !line_ctr5 & !line_ctr6 & !line_ctr7 &  vfield;

% vstate0  = _LC007 from file "memctl.tdf" line 56, column 36 %
vstate0  = DFF( _EQ029 $  VCC, GLOBAL( clkin),  VCC,  VCC);
  _EQ029 =  line_ctr0 &  line_ctr1 &  line_ctr2 &  line_ctr3 &  line_ctr4 &
              line_ctr5 &  line_ctr6 &  line_ctr7 & !vstate1 &  vsync
         # !vstate0 & !vstate1 &  vsync;

% vstate1  = _LC005 from file "memctl.tdf" line 56, column 36 %
vstate1  = DFF( _EQ030 $  vstate1, GLOBAL( clkin),  VCC,  VCC);
```

```
  _EQ030 =  line_ctr0 &  line_ctr1 & !line_ctr2 & !line_ctr3 &  line_ctr4 &
             !line_ctr5 & !line_ctr6 & !line_ctr7 &  vstate0 &  vstate1
         # !vstate1 & !vsync;

% wa_reg   = _LC027 from file "memctl.tdf" line 72, column 2 %
 wa       = DFF( _EQ031 $  GND, GLOBAL( clkin),  VCC,  VCC);
  _EQ031 = !rstate0 &  rstate1 & !rstate2;

% wb_reg   = _LC025 from file "memctl.tdf" line 73, column 2 %
 wb       = DFF( _EQ032 $  GND, GLOBAL( clkin),  VCC,  VCC);
  _EQ032 = !rstate0 & !rstate1 &  rstate2;
```

** COMPILATION SETTINGS & TIMES **

Processing Menu Commands

```
Design Doctor                              = off

Logic Synthesis:

   Default Synthesis Style                 = Normal

      Logic option settings in 'Normal' style for 'MAX5000' family

      minimization                         = full
      soft_buffer_insertion                = on
      xor_synthesis                        = on

   Other logic synthesis settings:

      Automatic Global Clock               = off
      Automatic Global Clear               = off
      Automatic Global Preset              = off
      Automatic Global Output Enable       = off
      Automatic Peripheral Registers       = off

Default Timing Specifications: None

Cut All Bidirectional Feedback Timing Paths = on
Cut All Clear & Preset Timing Paths        = on

Ignore Timing Assignments                  = on

Functional SNF Extractor                   = off

Linked SNF Extractor                       = off
Timing SNF Extractor                       = on
Optimize Timing SNF                        = off
Ignore Previous Fit                        = on
Automatic LCELL Insertion                  = off
Generate JEDEC Files                       = off
Total Recompile                            = off
```

Interface Menu Commands

```
EDIF Netlist Writer                        = off
Verilog Netlist Writer                     = off
VHDL Netlist Writer                        = off
```

Compilation Times

Compiler Netlist Extractor	00:00:02
Database Builder	00:00:06
Logic Synthesizer	00:00:13
Partitioner	00:00:02
Fitter	00:00:02
Timing SNF Extractor	00:00:01
Assembler	00:00:00
---	---
Total Time	00:00:26

Appendix C -- Buffer Control Code

```
*****************************************************************
*    Prototype Control File for 6811
*
*    This is to perform an initialization of the 22070 video
*    digitizer, and, the 22090 video encoder.  It involves setting
*    up the control registers in the two chips as well as the
*    CLUT for the encoder.
*
*
*    Phil Barker, Sep. 8, 1993
*       Last Modified:  Jan. 31, 1994
*                       Feb. 7, 1994
*                       Feb. 17, 1994
*                       Mar. 22, 1994   (after dumping the 22070)
*                       Mar. 23, 1994
*                       Mar. 28, 1994
*                       Mar. 29, 1994   (timing and subcarrier corrections)
*                       Apr. 11, 1994   (22190 adopted, change to 01 & 04)
*                       Apr. 13, 1994   (changed 01, clut function fixed)
*                       Apr. 14, 1994   (added I^2C bus for SONY)
*****************************************************************
*
*    Significant Memory Addresses
*
*  Global variable, located in on-chip SRAM

STACK           EQU     $FF     * Top of Stack location (top of internal SRAM

*  These addresses are code related, and are located in on-chip EEPROM

EEPROM          EQU     $F800   * Start of eeprom
INT_TBL         EQU     $FFC0   * Interrupt Vector Table Location
RESET           EQU     $FFFE   * Reset Vector location


*  These are 68HC11 Internal Register Assignments
*        For ALL code, the X Index register contains REG_BASE
*           --  INDEX THESE WITH REG_BASE !

REG_BASE        EQU     $1000   *base address for the port registers

PIOC            EQU     $02     *port c control register
PORTC           EQU     $03     *port c data register
PORTB           EQU     $04     *port b data register
DDRC            EQU     $07     *port c data direction register
TMSK2           EQU     $24
PACTL           EQU     $26

*  6811 Register Defaults, for this system

SDA_Z           EQU     $FD     * DDRC SET FOR SDA AS HIGH Z
DDR_OUT         EQU     $FF     * DDRC Port C Direction Register set for out
TMSK2_DEF       EQU     $00     * don't really use it in this application
PACTL_DEF       EQU     $00     * again, not really using it
PIOC_DEF        EQU     $00     * Port C to normal output, else don't care
*****************************************************************
*
*       Some definitions for the 22090
*
*****************************************************************

CSDA_BIT        EQU     $04     * The third control line is the DAC write

*****************************************************************
```

```
*
*               START OF CODE
*
*****************************************************************************
*
*       boot performs some 6811 initialization so that it can perform
*       the initialization of the 22090 and the 22070
*
*****************************************************************************

          ORG       EEPROM

boot      LDS       #STACK          *Initialize the stack pointer
          LDX       #REG_BASE       *The X Index Reg always points to REG_BASE
          LDAA      #TMSK2_DEF
          STAA      TMSK2,X         *Initialize the Real Time Interrupt
          LDAA      #PACTL_DEF      *(Probably don't care about value)
          STAA      PACTL,X         *Handles initialization  of RTI prescaler
          LDAA      #PIOC_DEF
          STAA      PIOC,X          *Initialize the parallel port
          LDAB      #DDR_OUT
          STAB      DDRC,X          *INIT. PORTC TO OUTPUT

*****************************************
*
*       22090 init of control registers
*
*****************************************

dac_init
          LDAB      #$4F            * FIRST CONTROL LINE TO PERFORM RESET
          STAB      PORTC,X         * TRANSFER CONTROL TO PORT C
          LDAA      #$00            * FIRST ADDRESS IS $00
          LDAB      #$0F            * READY TO LOAD FIRST
          STAA      PORTB,X         * ADDRESS INTO PORTB
          STAB      PORTC,X         * CONTROL INTO PORT C
          LDAB      #$0B            * ISSUE CONTROL TO WRITE IN ADDRESS
          STAB      PORTC,X
          LDAB      #$0F
          STAB      PORTC,X
          LDAB      #$0B
          STAB      PORTC,X
          LDAB      #$2F
          STAB      PORTC,X
          LDAA      #$01            * DATA FOR REGISTER 00
          JSR       WRITE_DA        * JUMP TO WRITE SUBROUTINE FOR THE DAC
          LDAA      #$10            * DATA FOR REGISTER 01(50)
          JSR       WRITE_DA
          LDAA      #$2C            * DATA FOR REGISTER 02
          JSR       WRITE_DA
          LDAA      #$02            * DATA FOR REGISTER 03
          JSR       WRITE_DA
          LDAA      #$40            * DATA FOR REGISTER 04(4E)
          JSR       WRITE_DA
          LDAB      #$2B            * CONTROL TO FORCE THE LAST WRITE IN
          STAB      PORTC,X
          LDAB      #$0F
          STAB      PORTC,X
          LDAA      #$10            * LOAD NEXT REGISTER ADDRESS AS $10
          STAA      PORTB,X
          LDAB      #$0B            * CONTROL TO WRITE NEW ADDRESS
          STAB      PORTC,X
          LDAB      #$0F
          STAB      PORTC,X
          LDAB      #$0B
          STAB      PORTC,X
```

```
        LDAB    #$2F
        STAB    PORTC,X
        LDAA    #$3A                    * DATA FOR REGISTER 10
        JSR     WRITE_DA
        LDAA    #$07                    * DATA FOR REGISTER 11
        JSR     WRITE_DA
        LDAA    #$1F                    * DATA FOR REGISTER 12
        JSR     WRITE_DA
        LDAA    #$0F                    * DATA FOR REGISTER 13
        JSR     WRITE_DA
        LDAA    #$23                    * DATA FOR REGISTER 14
        JSR     WRITE_DA
        LDAA    #$8B                    * DATA FOR REGISTER 15
        JSR     WRITE_DA
        LDAA    #$05                    * DATA FOR REGISTER 16
        JSR     WRITE_DA
        LDAA    #$77                    * DATA FOR REGISTER 17
        JSR     WRITE_DA
        LDAA    #$65                    * DATA FOR REGISTER 18
        JSR     WRITE_DA
        LDAA    #$12                    * DATA FOR REGISTER 19
        JSR     WRITE_DA
        LDAA    #$1C                    * DATA FOR REGISTER 1A
        JSR     WRITE_DA
        LDAA    #$6A                    * DATA FOR REGISTER 1B
        JSR     WRITE_DA
        LDAA    #$4C                    * DATA FOR REGISTER 1C
        JSR     WRITE_DA
        LDAA    #$3A                    * DATA FOR REGISTER 1D
        JSR     WRITE_DA
        LDAA    #$52                    * DATA FOR REGISTER 1E
        JSR     WRITE_DA
        LDAA    #$00                    * Reg. 1F is read only
        JSR     WRITE_DA
        LDAA    #$AB                    * DATA FOR REGISTER 20
        JSR     WRITE_DA
        LDAA    #$AA                    * DATA FOR REGISTER 21
        JSR     WRITE_DA
        LDAA    #$AA                    * DATA FOR REGISTER 22
        JSR     WRITE_DA
        LDAA    #$4A                    * DATA FOR REGISTER 23
        JSR     WRITE_DA
        LDAA    #$00                    * DATA FOR REGISTER 24
        JSR     WRITE_DA
        LDAA    #$00                    * DATA FOR REGISTER 25
        JSR     WRITE_DA
        LDAA    #$00                    * DATA FOR REGISTER 26
        JSR     WRITE_DA
        LDAA    #$20                    * DATA FOR REGISTER 27
        JSR     WRITE_DA

*********************************************
*
*       22090 init of clut
*
*********************************************

clut_init
        LDAB    #$1F
        STAB    PORTC,X
        LDAA    #$00                    * FIRST CLUT ADDRESS
        STAA    PORTB,X
        LDAB    #$1B                    *loading first clut address
        STAB    PORTC,X
        LDAB    #$1F
```

```
        LDAB    #$3F
        STAB    PORTC,X
        LDY     #$0000

        LDAA    #$00
        STAA    PORTB,X

clut_load
        LDAB    #$3B                    *write table d (R) value
        STAB    PORTC,X
        LDAB    #$3F
        STAB    PORTC,X
        LDAB    #$3B                    *write table e (G) value
        STAB    PORTC,X
        LDAB    #$3F
        STAB    PORTC,X
        LDAB    #$3B                    *write table f (B) value
        STAB    PORTC,X
        LDAB    #$3F
        STAB    PORTC,X
        INCA
        STAA    PORTB,X
        INY
        CPY     #$0100                  * COMPARE THE CURRENT ADDRESS TO TOP
        BLS     clut_load               * LOOP IF NOT DONE
        LDAA    #$00
        STAA    PORTB,X
        LDAB    #$0F                    * FORCE LAST CLUT INTO REGISTER
        STAB    PORTC,X
        LDAB    #$0B
        STAB    PORTC,X
        LDAB    #$OF
        STAB    PORTC,X

        LDAB    #$8F
        STAB    PORTC,X

*************************************************
*
*       I^2C init if SONY digitizer
*
*************************************************

        LDAB    #$8D                *DROP SDA FOR START SIGNAL
        STAB    PORTC,X
        JSR     WAIT
        LDAB    #$8C                *DROP SCL, NOW READY TO TRANSMIT
        STAB    PORTC,X
        JSR     WAIT
        LDAB    #$8D                *BIT 7 OF SLAVE ADDRESS (0)
        STAB    PORTC,X
        JSR     WAIT
        LDAB    #$8C
        STAB    PORTC,X
        JSR     WAIT
        LDAB    #$8E
        STAB    PORTC,X
        JSR     WAIT
        LDAB    #$8F                *BIT 6 OF SLAVE ADDRESS (1)
        STAB    PORTC,X
        JSR     WAIT
        LDAB    #$8E
        STAB    PORTC,X
        JSR     WAIT
        LDAB    #$8C
        STAB    PORTC,X
```

```
JSR     WAIT
LDAB    #$8D            *BIT 5 OF CLAVE ADDRESS (0)
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 4 OF SLAVE ADDRESS (0)
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 3 OF SLAVE ADDRESS (0)
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 2 OF SLAVE ADDRESS (0)
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 1 OF SLAVE ADDRESS (0)
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 0 OF SLAVE ADDRESS (0)
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAA    #SDA_Z
STAA    DDRC,X
LDAB    #$8D            *ACK BIT OF SLAVE ADDRESS
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAA    #DDR_OUT
STAA    DDRC,X
LDAB    #$8D            *BIT 7 OF SUB ADDRESS
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 6 OF SUB ADDRESS
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 5 OF SUB ADDRESS
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 4 OF SUB ADDRESS
```

```
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 3 OF SUB ADDRESS
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 2 OF SUB ADDRESS
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8E
STAB    PORTC,X
JSR     WAIT
LDAB    #$8F            *BIT 1 OF SUB ADDRESS
STAB    PORTC,X
JSR     WAIT
LDAB    #$8E
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 0 OF SUB ADDRESS
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAA    #SDA_Z
STAA    DDRC,X
LDAB    #$8D            *ACK BIT OF SUB ADDRESS
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAA    #DDR_OUT
STAA    DDRC,X
LDAB    #$8E
STAB    PORTC,X
JSR     WAIT
LDAB    #$8F            *BIT 7 OF APL
STAB    PORTC,X
JSR     WAIT
LDAB    #$8E
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 6 OF APL
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 5 OF APL
STAB    PORTC,X
JSR     WAIT
```

```
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D                *BIT 4 OF APL
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D                *BIT 3 OF APL
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D                *BIT 2 OF APL
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D                *BIT 1 OF APL
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8E
STAB    PORTC,X
JSR     WAIT
LDAB    #$8F                *BIT 0 OF APL
STAB    PORTC,X
JSR     WAIT
LDAB    #$8E
STAB    PORTC,X
JSR     WAIT
LDAA    #SDA_Z
STAA    DDRC,X
LDAB    #$8D                *ACK BIT OF APL
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAA    #DDR_OUT
STAA    DDRC,X
LDAB    #$8E
STAB    PORTC,X
JSR     WAIT
LDAB    #$8F                *BIT 7 OF SHP
STAB    PORTC,X
JSR     WAIT
LDAB    #$8E
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D                *BIT 6 OF SHP
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D                *BIT 5 OF SHP
STAB    PORTC,X
```

```
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 4 OF SHP
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 3 OF SHP
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 2 OF SHP
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 1 OF SHP
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 0 OF SHP
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAA    #SDA_Z
STAA    DDRC,X
LDAB    #$8D            *ACK BIT OF SHP
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAA    #DDR_OUT
STAA    DDRC,X
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 7 OF SAT
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8E
STAB    PORTC,X
JSR     WAIT
LDAB    #$8F            *BIT 6 OF SAT
STAB    PORTC,X
JSR     WAIT
LDAB    #$8E
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 5 OF SAT
```

```
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 4 OF SAT
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 3 OF SAT
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 2 OF SAT
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 1 OF SAT
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8E
STAB    PORTC,X
JSR     WAIT
LDAB    #$8F            *BIT 0 OF SAT
STAB    PORTC,X
JSR     WAIT
LDAB    #$8E
STAB    PORTC,X
JSR     WAIT
LDAA    #SDA_Z
STAA    DDRC,X
LDAB    #$8D            *ACK BIT OF SAT
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAA    #DDR_OUT
STAA    DDRC,X
LDAB    #$8E
STAB    PORTC,X
JSR     WAIT
LDAB    #$8F            *BIT 7 OF HUE
STAB    PORTC,X
JSR     WAIT
```

```
LDAB    #$8E
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 6 OF HUE
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 5 OF HUE
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 4 OF HUE
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 3 OF HUE
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 2 OF HUE
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 1 OF HUE
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
LDAB    #$8D            *BIT 0 OF HUE
STAB    PORTC,X
JSR     WAIT
LDAB    #$8C
STAB    PORTC,X
JSR     WAIT
```

```
        LDAA    #SDA_Z
        STAA    DDRC,X
        LDAB    #$8D            *ACK BIT OF HUE
        STAB    PORTC,X
        JSR     WAIT
        LDAB    #$8C
        STAB    PORTC,X
        JSR     WAIT
        LDAA    #DDR_OUT
        STAA    DDRC,X
        LDAB    #$8C            *STOP BIT SEQUENCE
        STAB    PORTC,X
        JSR     WAIT
        LDAB    #$8D
        STAB    PORTC,X
        JSR     WAIT
        LDAB    #$8F
        STAB    PORTC,X

*       Top of main loop. Just sit here.

main    BRA     main

*****************************************************************************
*
*                   Subroutines
*
*****************************************************************************

*****************************************************************************
*
*       WRITE_DA - takes the control lines and the data to be written out
*                  and puts out that data for the DAC.
*       Arguments:      ACCA - has the data to be written
*       Clobbers:       ACCB
*       Returns:        Nothing.
*
*****************************************************************************

WRITE_DA
        SEI                             * Disable Interupts
        LDAB    #$2B                    * PRE-WRITE CONTROL LINE
        STAB    PORTC,X                 * Put control lines in Port C
        STAA    PORTB,X                 * Put data into port B
        BSET    PORTC,X #CSDA_BIT       * Toggle write control line
        CLI
        RTS

*****************************************************************************
*
*       WAIT - SITS AROUND FOR ABOUT 6uS
*
*       CLOBBERS:       ACCY
*       RETURNS:        NOTHING
*
*****************************************************************************

WAIT
        INY
        INY
        INY
        INY
        INY
        INY
        INY
```

```
        INY
        INY
        INY
        INY
        RTS

*******************************************************************************
*
*         panic - system error.  Just sits the system here.
*            Arguments:        None
*            Clobbers:         ACCB
*            Returns nothing
*
*******************************************************************************

panic
          SEI
          BRA       panic

*******************************************************************************
*
*          cop_panic - system error.  Goes to panic where the system sits.
*              Arguments:       None
*              Clobbers:        ACCB
*              Returns nothing
*
*******************************************************************************

cop_panic
          SEI
          BRA       panic

*******************************************************************************
*
*               Interrupt Vector Table
*
*******************************************************************************

                ORG       $FFC0

                FDB       panic       * Reserved ($FFCO)
                FDB       panic       * Reserved
                FDB       panic       * Reserved
                FDB       panic       * Reserved

                FDB       panic       * Reserved ($FFD0)
                FDB       panic       * Reserved
                FDB       panic       * Reserved ($FFD4)
                FDB       panic       * SCI Serial System

                FDB       panic       * SPI Serial Transfer Complete ($FFD8)
                FDB       panic       * Pulse Acc Input Edge
                FDB       panic       * Pulse Acc Overflow
                FDB       panic       * Timer Overflow

                FDB       panic       * Timer Output Compare 5 ($FFE0)
                FDB       panic       * Timer Output Compare 4
                FDB       panic       * Timer Output Compare 3
                FDB       panic       * Timer Output Compare 2

                FDB       panic       * Timer Output Compare 1 ($FFE8)
                FDB       panic       * Timer Input Capture 3
                FDB       panic       * Timer Input Capture 2
                FDB       panic       * Timer Input Capture 1
```

```
FDB     panic       * Real Time Interrupt ($FFF0)
FDB     panic       * IRQ Input
FDB     panic       * XIRQ Input
FDB     panic       * Software Interrupt

FDB     panic       * Illegal Opcode Trap
FDB     cop_panic   * COP Failure (Reset)
FDB     cop_panic   * COP Clock Monitor Fail (Reset)
FDB     boot        * External RESET ($FFFE)


 ORG    $FFFE
 FDB    boot
```

Appendix D -- Composite Control Code

```
module keypal
title   'key control device
        Directors Composite Eyeglass
        P. Barker,      4/20/94'

        keypal  device  'P22V10';

        "Pin Declarations
        "Inputs

        I7              pin 1;
        I6              pin 2;
        I5              pin 3;
        I4              pin 4;
        S7              pin 5;
        S6              pin 6;
        S5              pin 7;
        S4              pin 8;
        view            pin 9;
        key             pin 10;

        "Outputs

        select          pin 19;

        "Declarations

        LIVE = [I7,I6,I5,I4];
        COLOR = [S7,S6,S5,S4];

equations

select = ((LIVE >= COLOR) & key) + (!key & view);

end keypal;
```

ABEL 4.00 - Device Utilization Chart Wed Apr 20 16:02:09 1994

key control device
Directors Composite Eyeglass
P. Barker, 4/20/94

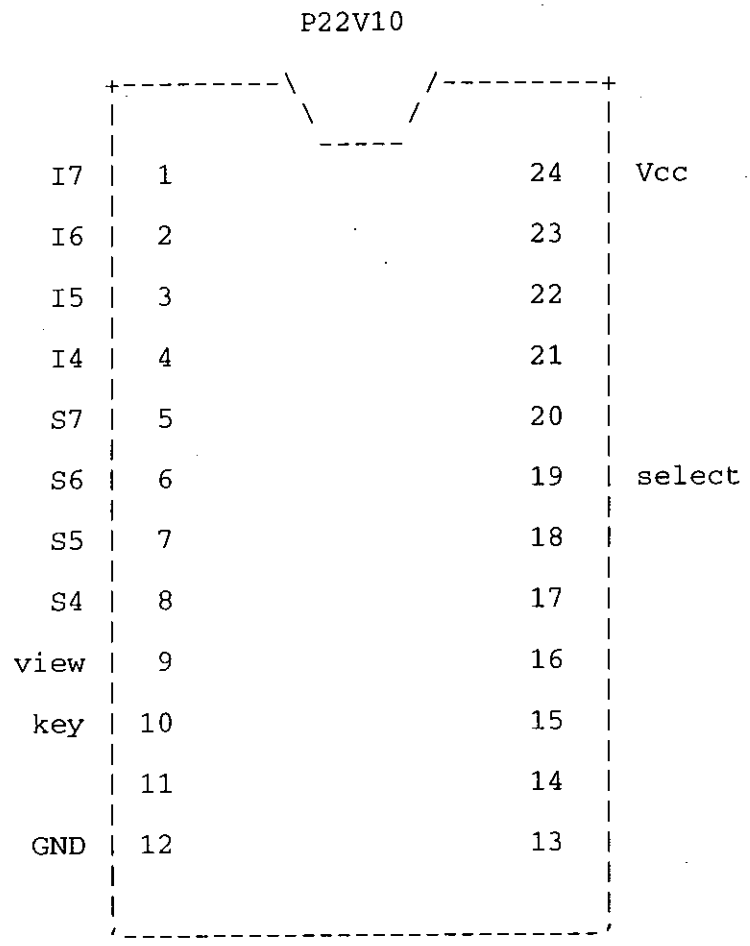
==== P22V10 Programmed Logic ====

```
select      = !(  !I4 & S7 & S6 & S5 & S4 & key
            #    !I7 & !I4 & S6 & S5 & S4 & key
            #    !I6 & !I4 & S7 & S5 & S4 & key
            #    !I7 & !I6 & !I4 & S5 & S4 & key
            #    !I5 & !I4 & S7 & S6 & S4 & key
            #    !I7 & !I5 & !I4 & S6 & S4 & key
            #    !I6 & !I5 & !I4 & S7 & S4 & key
            #    !I7 & !I6 & !I5 & !I4 & S4 & key
            #    !I5 & S7 & S6 & S5 & key
            #    !I7 & !I5 & S6 & S5 & key
            #    !I6 & !I5 & S7 & S5 & key
            #    !I7 & !I6 & !I5 & S5 & key
            #    !view & !key
            #    !I6 & S7 & S6 & key
            #    !I7 & !I6 & S6 & key
            #    !I7 & S7 & key );
```

ABEL 4.00 - Device Utilization Chart Wed Apr 20 16:02:10 1994

key control device
Directors Composite Eyeglass
P. Barker, 4/20/94

==== P22V10 Chip Diagram ====



SIGNATURE: N/A

ABEL 4.00 - Device Utilization Chart Wed Apr 20 16:02:10 1994

key control device
Directors Composite Eyeglass
P. Barker, 4/20/94

==== P22V10 Resource Allocations ====

Device Resources	Resource Available	Design Requirement	Part Utilization	Unused
Dedicated input pins	12	10	10	2 (16 %)
Combinatorial inputs	12	10	10	2 (16 %)
Registered inputs	-	0	-	-
Dedicated output pins	-	1	-	-
Bidirectional pins	10	0	1	9 (90 %)
Combinatorial outputs	-	1	-	-
Registered outputs	-	0	-	-
Reg/Com outputs	10	-	1	9 (90 %)
Two-input XOR	-	0	-	-
Buried nodes	-	0	-	-
Buried registers	-	0	-	-
Buried combinatorials	-	0	-	-

ABEL 4.00 - Device Utilization Chart Wed Apr 20 16:02:10 1994

key control device
Directors Composite Eyeglass
P. Barker, 4/20/94

==== P22V10 Product Terms Distribution ====

Signal Name	Pin Assigned	Terms Used	Terms Max	Terms Unused
select	19	16	16	0

==== List of Inputs/Feedbacks ====

Signal Name	Pin	Pin Type
I7	1	CLK/IN
I6	2	INPUT
I5	3	INPUT
I4	4	INPUT
S7	5	INPUT
S6	6	INPUT
S5	7	INPUT
S4	8	INPUT
view	9	INPUT
key	10	INPUT

ABEL 4.00 - Device Utilization Chart Wed Apr 20 16:02:10 1994

key control device
Directors Composite Eyeglass
P. Barker, 4/20/94

==== P22V10 Unused Resources ====

Pin Number	Pin Type	Product Terms	Flip-flop Type
11	INPUT	-	-
13	INPUT	-	-
14	BIDIR	NORMAL 8	D
15	BIDIR	NORMAL 10	D
16	BIDIR	NORMAL 12	D
17	BIDIR	NORMAL 14	D
18	BIDIR	NORMAL 16	D
20	BIDIR	NORMAL 14	D
21	BIDIR	NORMAL 12	D
22	BIDIR	NORMAL 10	D
23	BIDIR	NORMAL 8	D

ABEL 4.00 - Device Utilization Chart Wed Apr 20 16:02:10 1994

key control device
Directors Composite Eyeglass
P. Barker, 4/20/94

==== I/O Files ====

Module: 'keypal'

Input files
==========
ABEL PLA file: keypal.tt2
Vector file: keypal.tmv
Device library: P22V10.dev

Output files
==========
Report file: keypal.doc
Programmer load file: keypal.jed

```
module keypal2
title   'key control device
        Directors Composite Eyeglass
        P. Barker,      4/20/94'

        keypal2 device  'P22V10';

        "Pin Declarations
        "Inputs

        I7              pin 1;
        I6              pin 2;
        I5              pin 3;
        I4              pin 4;
        S7              pin 5;
        S6              pin 6;
        S5              pin 7;
        S4              pin 8;
        view            pin 9;
        key             pin 10;
        E7              pin 11;
        E6              pin 23;
        E5              pin 22;
        E4              pin 21;

        "Outputs

        select1         pin 19;
        select2         pin 18;

        "Declarations

        LIVE = [I7,I6,I5,I4];
        LEFT = [S7,S6,S5,S4];
        RIGHT = [E7,E6,E5,E4];

equations

select1 = ((LIVE >= LEFT) & key) + (!key & view);
select2 = ((LIVE <= RIGHT) & key) + (!key & view);

end keypal2;
```

ABEL 4.00 - Device Utilization Chart Thu Apr 21 11:34:12 1994

key control device
Directors Composite Eyeglass
P. Barker, 4/20/94

==== P22V10 Programmed Logic ====

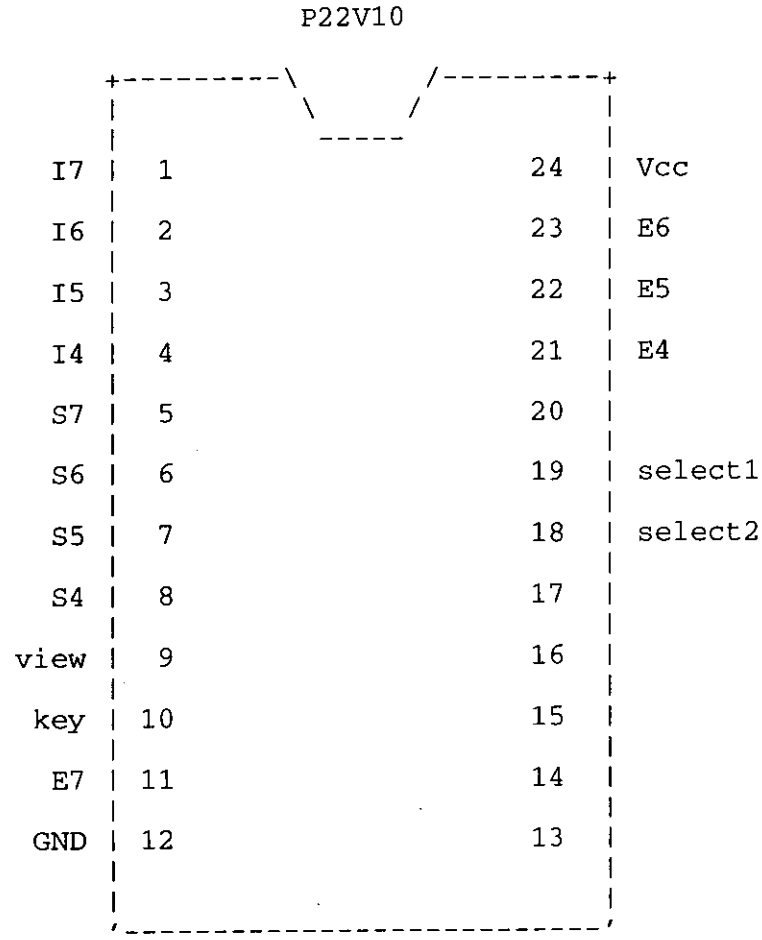
```
select1     = !(  !I4 & S7 & S6 & S5 & S4 & key
          #    !I7 & !I4 & S6 & S5 & S4 & key
          #    !I6 & !I4 & S7 & S5 & S4 & key
          #    !I7 & !I6 & !I4 & S5 & S4 & key
          #    !I5 & !I4 & S7 & S6 & S4 & key
          #    !I7 & !I5 & !I4 & S6 & S4 & key
          #    !I6 & !I5 & !I4 & S7 & S4 & key
          #    !I7 & !I6 & !I5 & !I4 & S4 & key
          #    !I5 & S7 & S6 & S5 & key
          #    !I7 & !I5 & S6 & S5 & key
          #    !I6 & !I5 & S7 & S5 & key
          #    !I7 & !I6 & !I5 & S5 & key
          #    !view & !key
          #    !I6 & S7 & S6 & key
          #    !I7 & !I6 & S6 & key
          #    !I7 & S7 & key );

select2     = !(  !view & !key
          #    I4 & key & !E7 & !E6 & !E5 & !E4
          #    I7 & I4 & key & !E6 & !E5 & !E4
          #    I6 & I4 & key & !E7 & !E5 & !E4
          #    I7 & I6 & I4 & key & !E5 & !E4
          #    I5 & I4 & key & !E7 & !E6 & !E4
          #    I7 & I5 & I4 & key & !E6 & !E4
          #    I6 & I5 & I4 & key & !E7 & !E4
          #    I7 & I6 & I5 & I4 & key & !E4
          #    I5 & key & !E7 & !E6 & !E5
          #    I7 & I5 & key & !E6 & !E5
          #    I6 & I5 & key & !E7 & !E5
          #    I7 & I6 & I5 & key & !E5
          #    I6 & key & !E7 & !E6
          #    I7 & I6 & key & !E6
          #    I7 & key & !E7 );
```

ABEL 4.00 - Device Utilization Chart Thu Apr 21 11:34:12 1994

key control device
Directors Composite Eyeglass
P. Barker, 4/20/94

==== P22V10 Chip Diagram ====



SIGNATURE: N/A

ABEL 4.00 - Device Utilization Chart Thu Apr 21 11:34:12 1994

key control device
Directors Composite Eyeglass
P. Barker, 4/20/94

==== P22V10 Resource Allocations ====

Device Resources	Resource Available	Design Requirement	Part Utilization	Unused
Dedicated input pins	12	14	11	1 (8 %)
Combinatorial inputs	12	11	11	1 (8 %)
Registered inputs	-	0	-	-
Dedicated output pins	-	2	-	-
Bidirectional pins	10	0	5	5 (50 %)
Combinatorial outputs	-	2	-	-
Registered outputs	-	0	-	-
Reg/Com outputs	10	-	2	8 (80 %)
Two-input XOR	-	0	-	-
Buried nodes	-	0	-	-
Buried registers	-	0	-	-
Buried combinatorials	-	0	-	-

ABEL 4.00 - Device Utilization Chart Thu Apr 21 11:34:13 1994

key control device
Directors Composite Eyeglass
P. Barker, 4/20/94

==== P22V10 Product Terms Distribution ====

Signal Name	Pin Assigned	Terms Used	Terms Max	Terms Unused
select1	19	16	16	0
select2	18	16	16	0

==== List of Inputs/Feedbacks ====

Signal Name	Pin	Pin Type
I7	1	CLK/IN
I6	2	INPUT
I5	3	INPUT
I4	4	INPUT
S7	5	INPUT
S6	6	INPUT
S5	7	INPUT
S4	8	INPUT
view	9	INPUT
key	10	INPUT
E7	11	INPUT
E6	23	BIDIR
E5	22	BIDIR
E4	21	BIDIR

ABEL 4.00 - Device Utilization Chart Thu Apr 21 11:34:13 1994

key control device
Directors Composite Eyeglass
P. Barker, 4/20/94

==== P22V10 Unused Resources ====

Pin Number	Pin Type	Product Terms	Flip-flop Type
13	INPUT	-	-
14	BIDIR	NORMAL 8	D
15	BIDIR	NORMAL 10	D
16	BIDIR	NORMAL 12	D
17	BIDIR	NORMAL 14	D
20	BIDIR	NORMAL 14	D

ABEL 4.00 - Device Utilization Chart Thu Apr 21 11:34:13 1994

key control device
Directors Composite Eyeglass
P. Barker, 4/20/94

==== I/O Files ====

Module: 'keypal2'

Input files
==========
ABEL PLA file: keypal2.tt2
Vector file: keypal2.tmv
Device library: P22V10.dev

Output files
==========
Report file: keypal2.doc
Programmer load file: keypal2.jed

Appendix E -- Acknowledgments

Acknowledgments:

Glorianna Davenport:	My supervisor who took set backs in stride.
John Watlington:	The technical consultant without whom this project would not exist.
Hanoz:	For the help in debugging and getting things around the lab to work.
Chad Mikkelson:	Whose comment one night showed me the error of the past week.
Stephan Fitch:	Who gave me the project in the first place.
John Eldon:	The chip designer that helped me figure out his chip's errors.
Betsy Brown:	For sanity in adminstravia.