A Retail Video Acquisition System

by

Mark D. Hansen

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 1995

Author_____

Department of Electrical Engineering and Computer Science

May 12, 1995

Certified by_____

Richard D. Thornton

Thesis Supervisor

Accepted by_____

Frederic R. Morgenthaler,

Chairman, Committee on Graduate Students

A Retail Video Acquisition System

by

Mark D. Hansen

Submitted to the

Department of Electrical Engineering and Computer Science

May 12, 1995

In Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Electrical Engineering

and Master of Engineering in Electrical Engineering and Computer Science

Abstract

The retail industry offers many applications for a video capture board, such as video surveillance. To meet the demands of the industry, a video capture board must have high resolution, a universal interface, and inexpensive components. In addition, the board must have drivers and software to support it on a personal computer. This software must be easy to use and have the ability to load and store captured images.

A video capture board was designed and constructed to meet these requirements. This board accepts any NTSC video signal, digitizes a single field in one pass, and transmits the video data to a personal computer via a parallel printer port. Parts for this board are commercially available and inexpensive, with an overall board cost of under $100. Software was also designed to support the board on an IBM-compatible computer running Microsoft Windows. This software provides an attractive interface to the board and has features desired for a video surveillance application, such as capturing video and storing digitized pictures to disk in popular image formats.

Thesis Supervisor: Richard D. Thornton

Title: Professor of Electrical Engineering

Table of Contents

Table of Figures

1. Overview

The retail environment presents several opportunities for a suitably designed video capture board. In a retail setting, such as a grocery or department store, a video capture board could be used as a part of a store security system. Images could be captured from a video camera, stored in a database with a time/date stamp, and recalled immediately. Store managers could also analyze digitized video for customer movement patterns to identify items capturing the consumers' interest. Ordinary videotape could be used for these purposes, but the medium is bulky and expensive. Furthermore, extracting two images from a videotape that are separated in time is a time-consuming process. Images stored on a computer's hard disk, on the other hand, are easily retrievable and cheap to store. Digital image enhancement techniques can be applied quickly, for the images are already stored in a digital format.

In this project, a video capture board was designed to meet the requirements presented by the retail environment. This video capture board captures high resolution, eight bit grayscale images with a relatively high transfer rate to a computer. Additionally, it uses inexpensive components to meet the low budget constraints of the industry. To offer compatibility with a wide range of computer architectures, the board has a parallel eight bit input, eight bit output bus, which can be used by IBM-compatible computers with a parallel port. To demonstrate the board's capabilities, an application called "Third Eye" was created. This application drives the video capture board and offers features that would be useful for a video surveillance application. "Third Eye" can capture live video images continuously, store them to disk with a time/date stamp, and display them at a later time.

2. Video Capture Board

2.1 Overview

The video capture board digitizes a single field from a video source and transmits the digital picture data to a personal computer. On the board, a video sync separator detects the horizontal and vertical synchronization pulses from the input signal. With this synchronization information, a finite state machine (FSM) determines the start of new video fields and lines. When the video source is starting to transmit a new video line, a flash analog-to-digital converter begins taking eight bit samples of the signal and stores the samples into a 64 KB field buffer. This field buffer is addressed by a fifteen bit memory address register. The sixteenth bit of the address to the field buffer is provided by a control FSM, which directs the operation of the other systems as well. Once the field buffer holds a complete video field, the buffer transmits its contents across an eight bit bus to the board interface. This interface converts the data for transmission across a parallel data port. Figure 2-1 contains a block diagram of the video capture board.

2.2 Data Paths

An eight bit data bus connects the flash analog-to-digital converter to the field buffer and the board interface. Because the analog-to-digital converter needs to be enabled at all times, a 74LS244 octal tristate buffer is required to buffer the data from the converter. The /ADOE signal from the control FSM enables the 74LS244 to allow data from the analog-to-digital converter onto the bus. The field buffer has two output enable signals, /SRAMAOE and /SRAMBOE, which tristate its output when not driving the bus. The board interface only reads data from the bus, so the interface itself does not require any buffering.

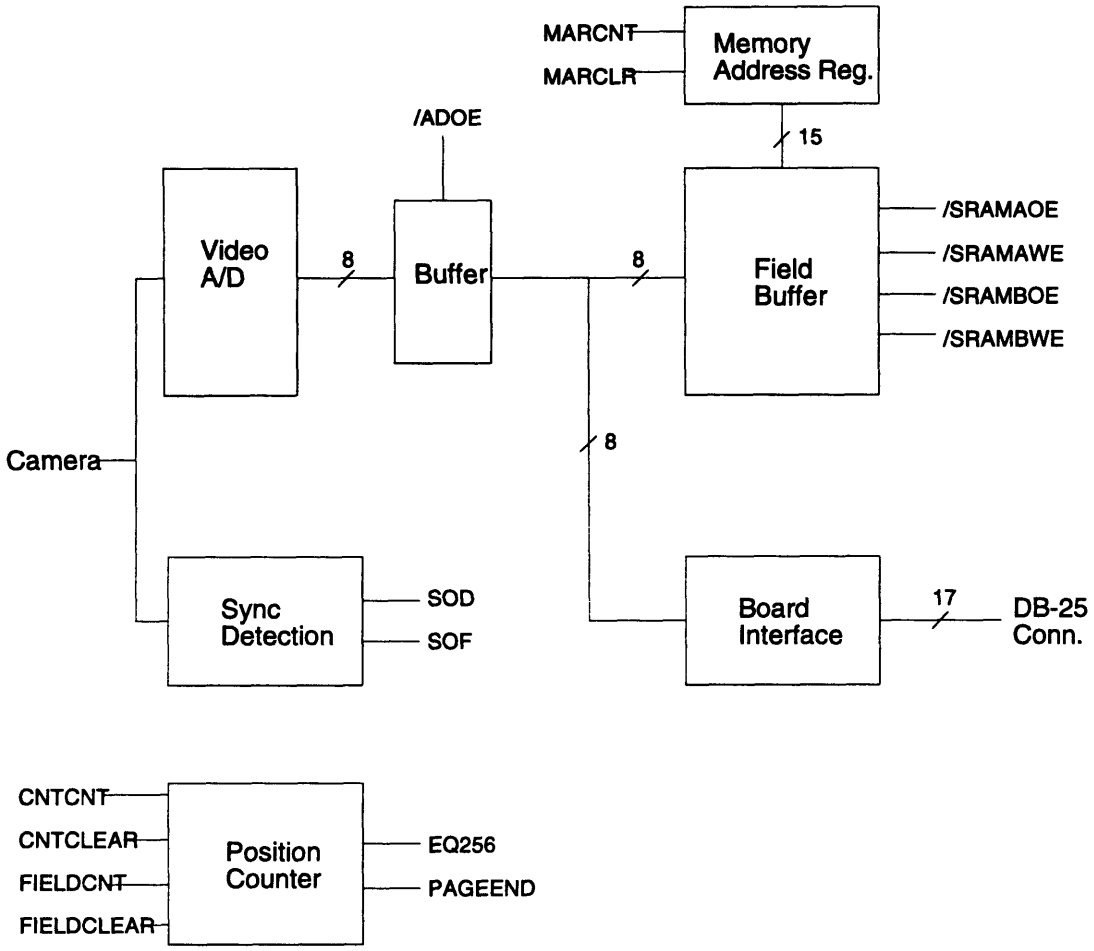

Figure 2-1: Video Capture Board Block Diagram

2.3 Clock Generation

To obtain a horizontal resolution of 256 pixels per line, the analog-to-digital converter requires a sample clock of 5.0 MHz. The system clock speed is limited by the worst case delay in the system, which is the write delay for the 62256 SRAMs. Since the chip select (/CS) of the two SRAMs are tied to the system clock, a write operation can only occur on the bottom half of the clock cycle. The delay for a write operation for these SRAMs is 70 ns, allowing a minimum clock period of 140 ns (7 MHz). This maximum clock speed is close to the required sampling frequency. As a result, the system and sample clocks are both run at 5.0 MHz so that samples are created and stored within the same clock cycle.

To create the 5.0 MHz clock, a 74LS163 four bit counter divides a 10 MHz crystal oscillator output in half on the least significant bit of its output. The 5.0 MHz output of the counter drives one 74LS04 inverter to buffer the clock signal from the counter and then drives additional chains of inverters

to further buffer the clock. To minimize clock skew, the clock generator occupies a central location on the protoboard. Additionally, each clocked chip has its own unshared wire to the clock generator to avoid clock skew caused by daisy-chaining. Figure 2-2 contains a schematic for the clock module.

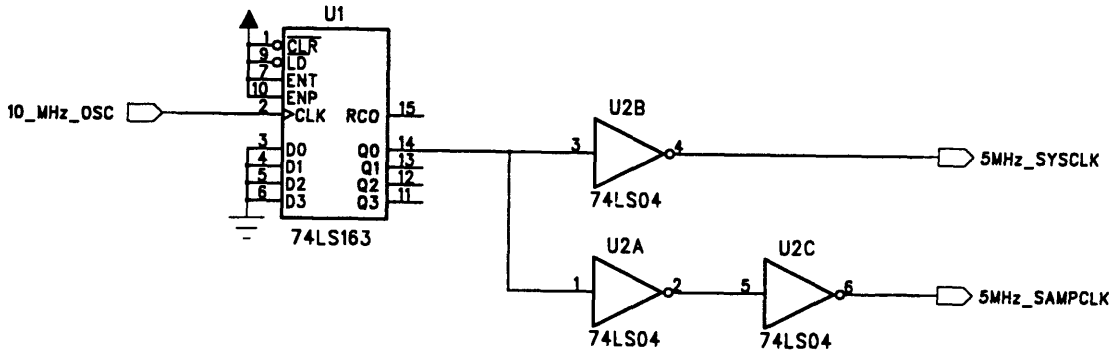

Figure 2-2: System and Sample Clock Schematic

2.4 Video Digitization

2.4.1 Video Input

The video capture board accepts any composite, NTSC video source through a coaxial or twisted pair connector. A Panasonic Black-and-White WV-1500 camera was used for testing of the board. The camera outputs a picture field sixty times a second, and every two fields are interlaced to create a single picture frame. One field contains the odd lines of the frame, and the other field contains the even lines. Each field from the camera is sent in an analog signal which contains image data as well as an overlaid synchronization signal. The synchronization signal is composed of two signals, HDRIVE and VDRIVE, which indicate when the camera is about to output picture data corresponding to the start of a new line or field. These signals are indicated by pulses in the camera output below 0.3 volts. Figure 2-3 shows the timing of the synchronization signals with the active picture data. The camera asserts HDRIVE at the start of every horizontal line, which occurs around 262 times a field. HDRIVE pulses occur every 64 μs within a field, and the analog picture data for a given line is transmitted between the pulses. The voltage range of the analog picture data varies depending on the video source, but the Panasonic camera's signal ranges from .8 volt to 3.0 volts. The picture data signal represents image luminance, with .8 volts representing a black area and 3.0 volts representing a white area. Although the camera has 64 μs to transmit the picture data for a line, actual data is only transmitted for 51.8 μs. This 51.8 μs window determines the sampling frequency; since the desired horizontal resolution is 256 pixels, the sampling rate must be 256 / 51.8 μs ≅ 5.0 MHz. The vertical synchronization pulses (VDRIVE) signal the start of a field, which occur 60 times a second. Like HDRIVE, VDRIVE is represented by pulses in the analog waveform dropping below .3 v. However, the VDRIVE pulses are asserted for a much longer time than HDRIVE pulses, lasting about the same time as it takes to transmit nine horizontal lines. Figure 2-4 contains an oscilloscope picture of the camera analog output.

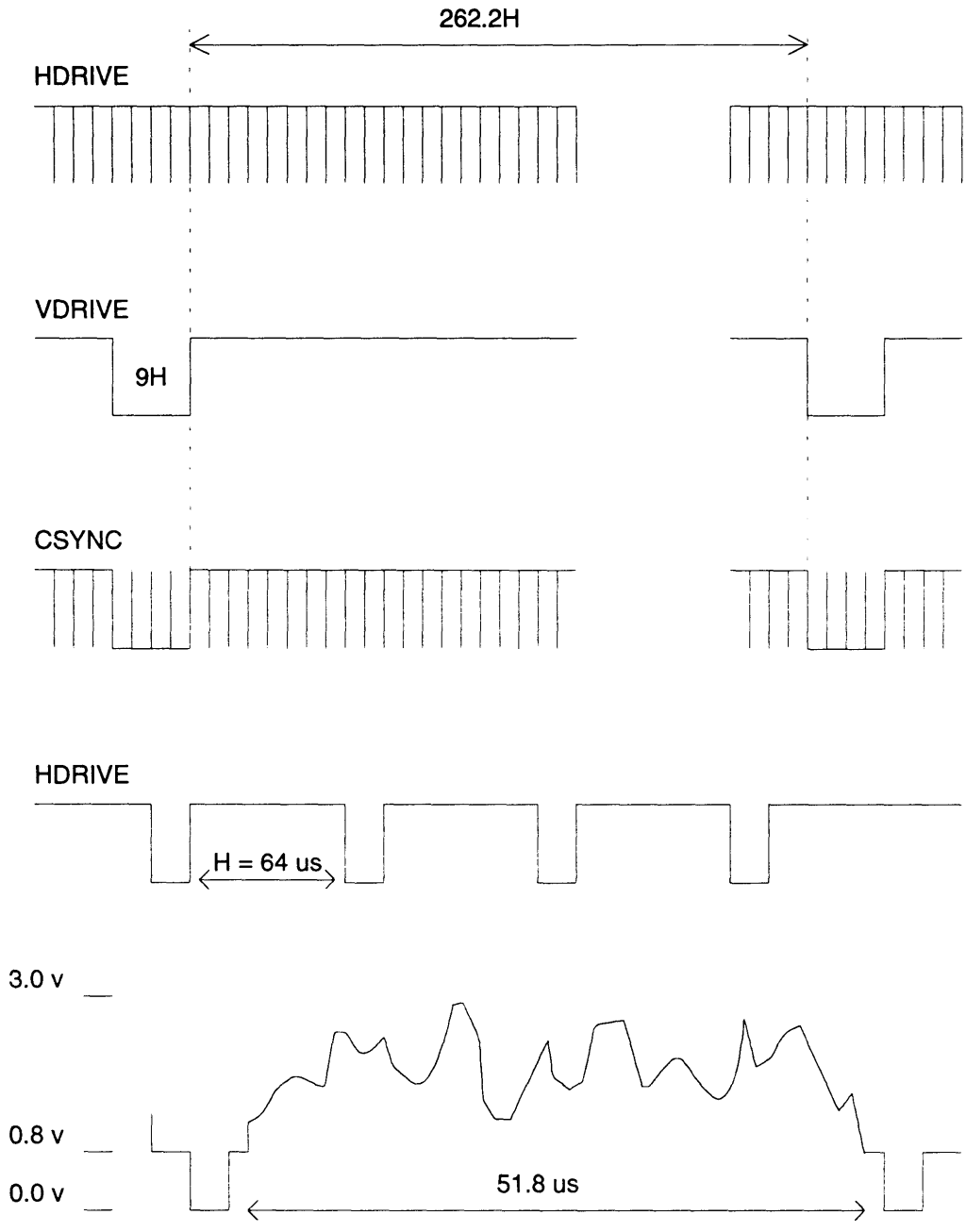

Figure 2-3: Video Signal Timing

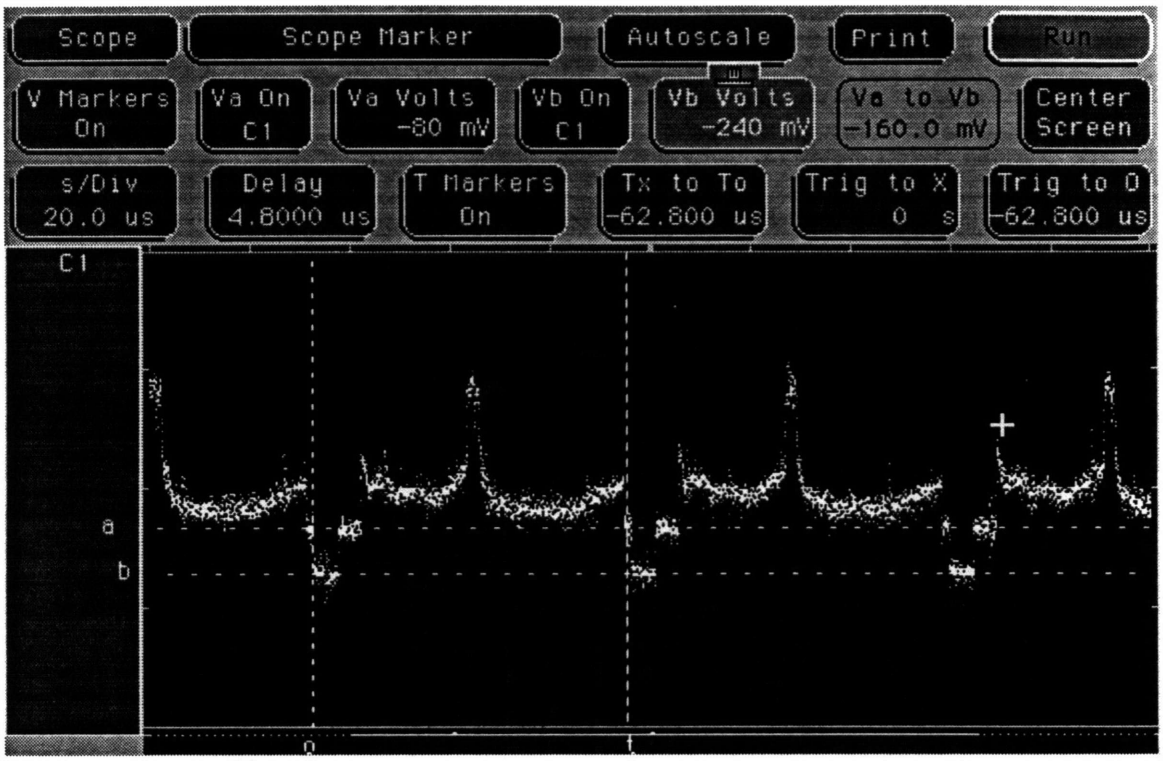

Figure 2-4: Video Signal Input

2.4.2 Sync Detection

The Sync Detection system decodes the synchronization signals from the analog input to determine the correct times to start sampling the waveform. The GENNUM GS4981 Sync Separator accepts an analog video input and outputs active high, TTL level signals HDRIVE and VSYNC when it detects the appropriate synchronization signals. However, these signals do not provide enough information to accurately determine when to start sampling. The A/D must start sampling immediately after the rising edge of HDRIVE, but only when HDRIVE occurs during an active video line. To keep track of the state of the video input, a three-state Mealy FSM called Videoflag accepts the HDRIVE and VSYNC signals and provides the following outputs: Start of Field (SOF), End of Field (EOF), Start of Data (SOD), and End of Data (EOD). Figure 2-5 shows a state transition diagram for the Videoflag FSM. When VDRIVE is detected in any state, the FSM enters the "Between Fields" to wait for the next field to begin. When VDRIVE is deasserted, the Videoflag FSM asserts SOF and enters the "Active Data" state. However, the control FSM will not start sampling yet because it has not received the SOD signal. The Videoflag FSM will remain in the "Active Data" state until HDRIVE is asserted, where it will enter the "Between Lines" state. After HDRIVE is deasserted, the Videoflag FSM will return to "Active Data" and assert SOD, which will cause the control FSM to start sampling. Figure 2-6 shows the FSM asserting SOD at the start of a horizontal line, and Figure 2-7 shows the assertion of SOF at the start of a field.

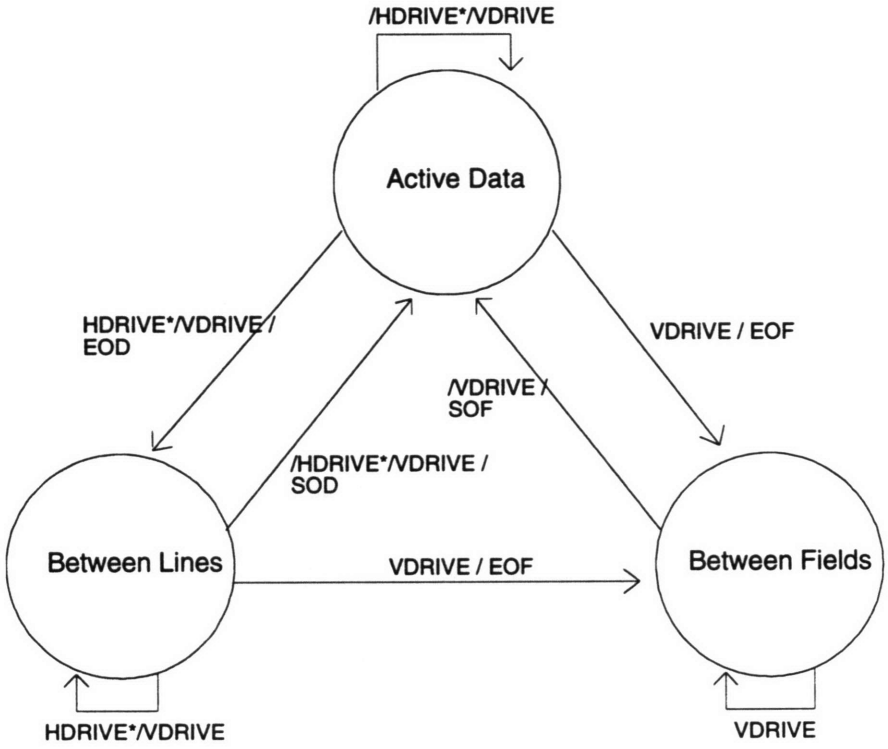

Figure 2-5: Videoflag FSM

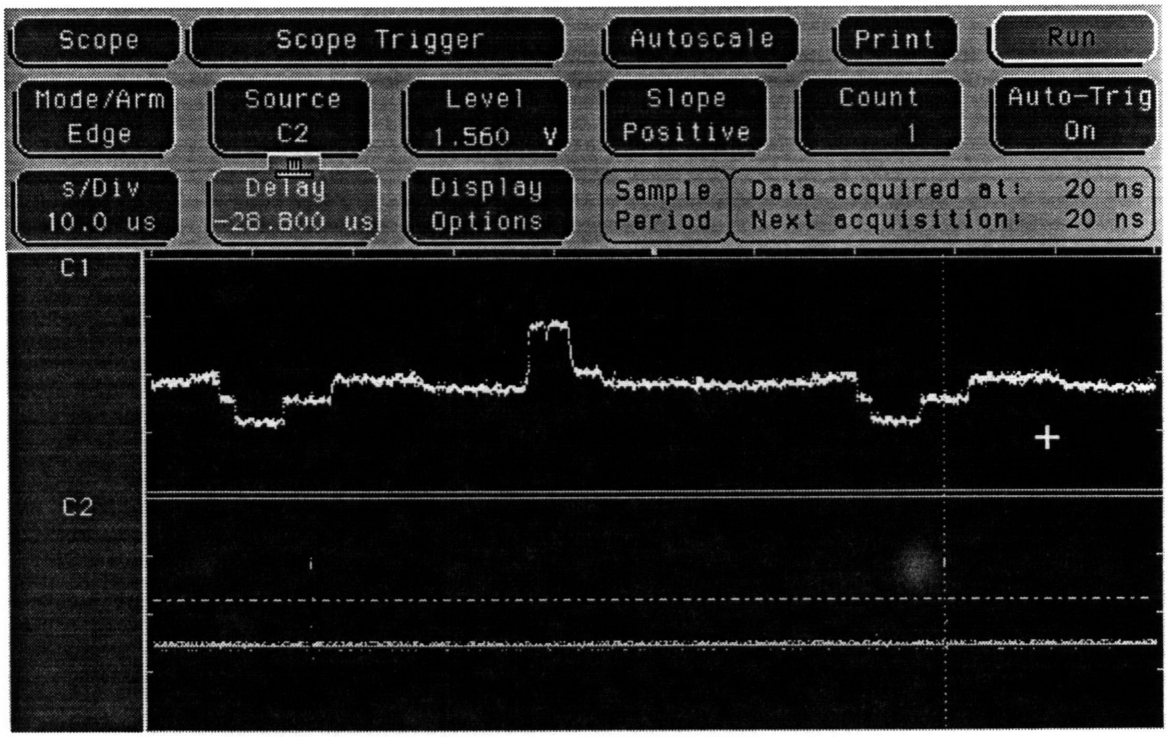

Figure 2-6: Videoflag SOD Assertion

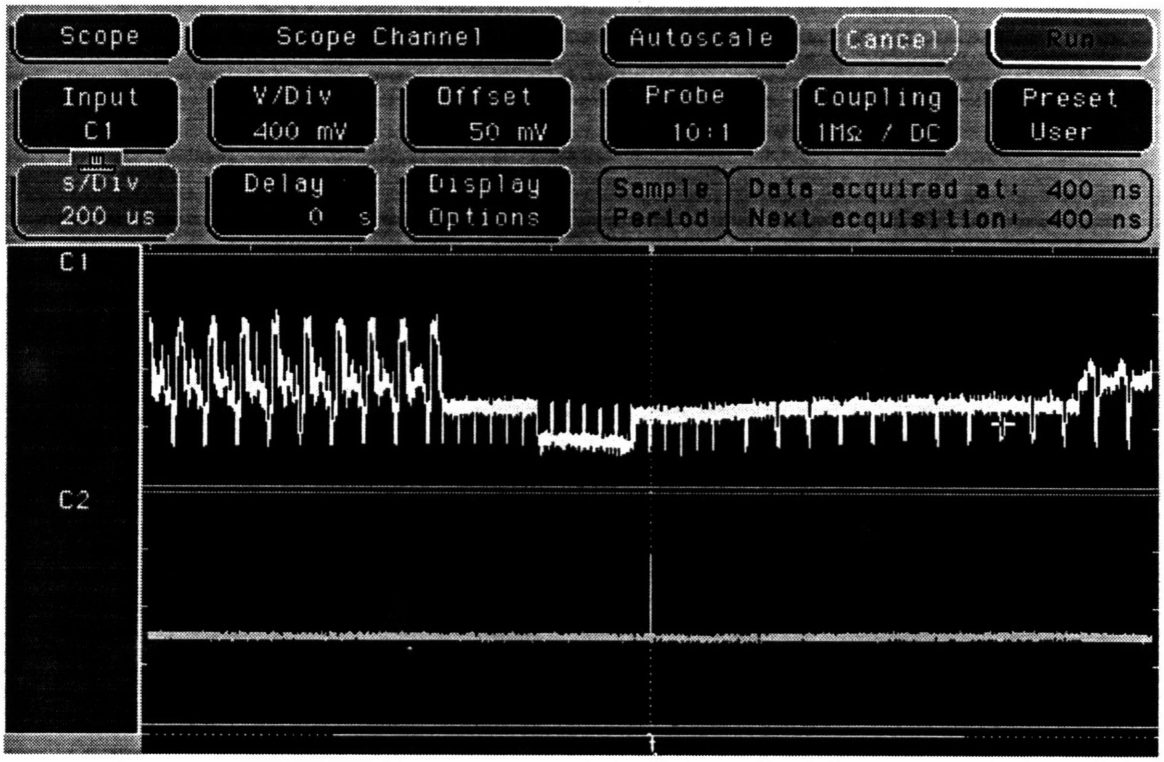

Figure 2-7: Videoflag SOF Assertion

Figure 2-8 contains a schematic for the sync separator. The 75 Ω resistor and the .01 μF capacitor connected to the video input filter the chrominance component of the signal. Since the chrominance component contains relatively high frequencies, it can interfere with the sync detection within the chip. The 680 KΩ resistor and .1 μF capacitor in parallel on the Rset input form an RC circuit, which the chip uses to predict synchronization pulse widths.

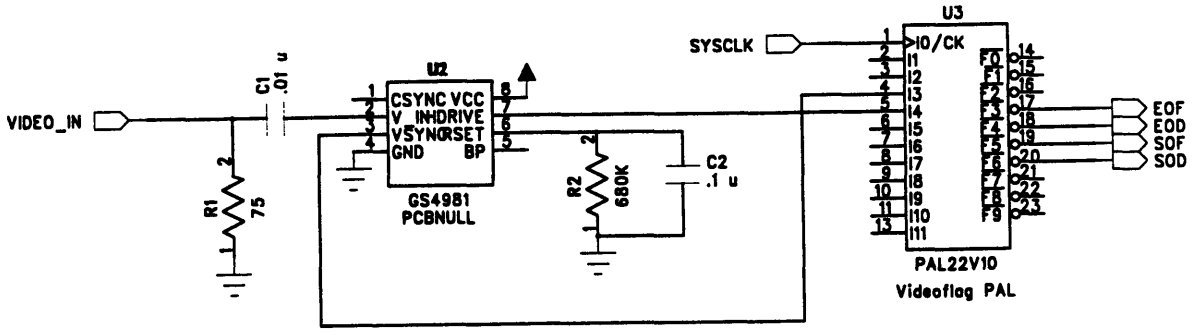

Figure 2-8: Sync Separator Schematic

2.4.3 Analog to Digital Conversion

The A/D converter, a MC10319, is responsible for sampling the analog output of the camera and driving the digital output onto a bus to be stored in a SRAM. The analog input is converted to an eight bit grayscale value at a rate of 5.0 MHz. Because this sampling rate is the same as the system clock, the A/D places new sample values on the bus every clock cycle. The MC10319 uses an internal resistor ladder to quantize the analog signal to digital levels, and the sampling voltage range is set by fixing the reference voltages at the top (V_{RT}), midpoint (V_{RM}), and bottom (RB) of the ladder. Since the MC10319 works particularly well when sampling negative voltages, the ladder is set to sample between negative two volts (-2V) and ground. To create the -2 Vreference signal for the A/D converter, a 10 KΩ potentiometer is connected to -5 V, and a multimeter is used to calibrate the output to -2 V. The -5 V reference for the potentiometer is created by a 7905 voltage regulator connected to -12 V.

Since the input video signal lies between 0.3Vand 1.0V, an voltage offset and amplifier is required to move signal within the -2V and 0V range. A .47μF capacitor in series removes the .3V DC offset from the input video signal, and two 741 op-amps add an offset and negatively amplify the input within the required range. A 10 KΩ potentiometer adjusts the amount of the offset voltage to account for variations in the input signal. Figure 2-9 contains a schematic of the MC10319 and the analog amplifier circuitry.

The MC10319 is very sensitive to noise, and any fluctuations in the power supply or reference voltages can appear as random bit errors in the output. To minimize voltage variations, 10 μF bypass capacitors decouple the voltage references and power supplies. Also, the analog power supply on the protoboard kit is used instead of the noisy switched digital supplies. The video input amplifier also helps reduce noise by increasing the dynamic range of the signal. Since the 1.0 V range of the input is amplified to a 2.0 V range, small voltage fluctuations do not have as much effect on the conversion process.

The MC10319 does not require a sample/hold operation, and it outputs a new value every clock cycle. However, the data is only guaranteed to be valid during the bottom half of the clock cycle since a new value is latched 19 ns after the falling edge of the clock. Since the field buffer, comprised of two SRAMs, are only enabled during the second half of the clock cycle, an inverted system clock signal is provided to the A/D. With this system, new samples are latched onto the bus 19 ns after a rising clock edge and should remain valid through the second half of the clock cycle.

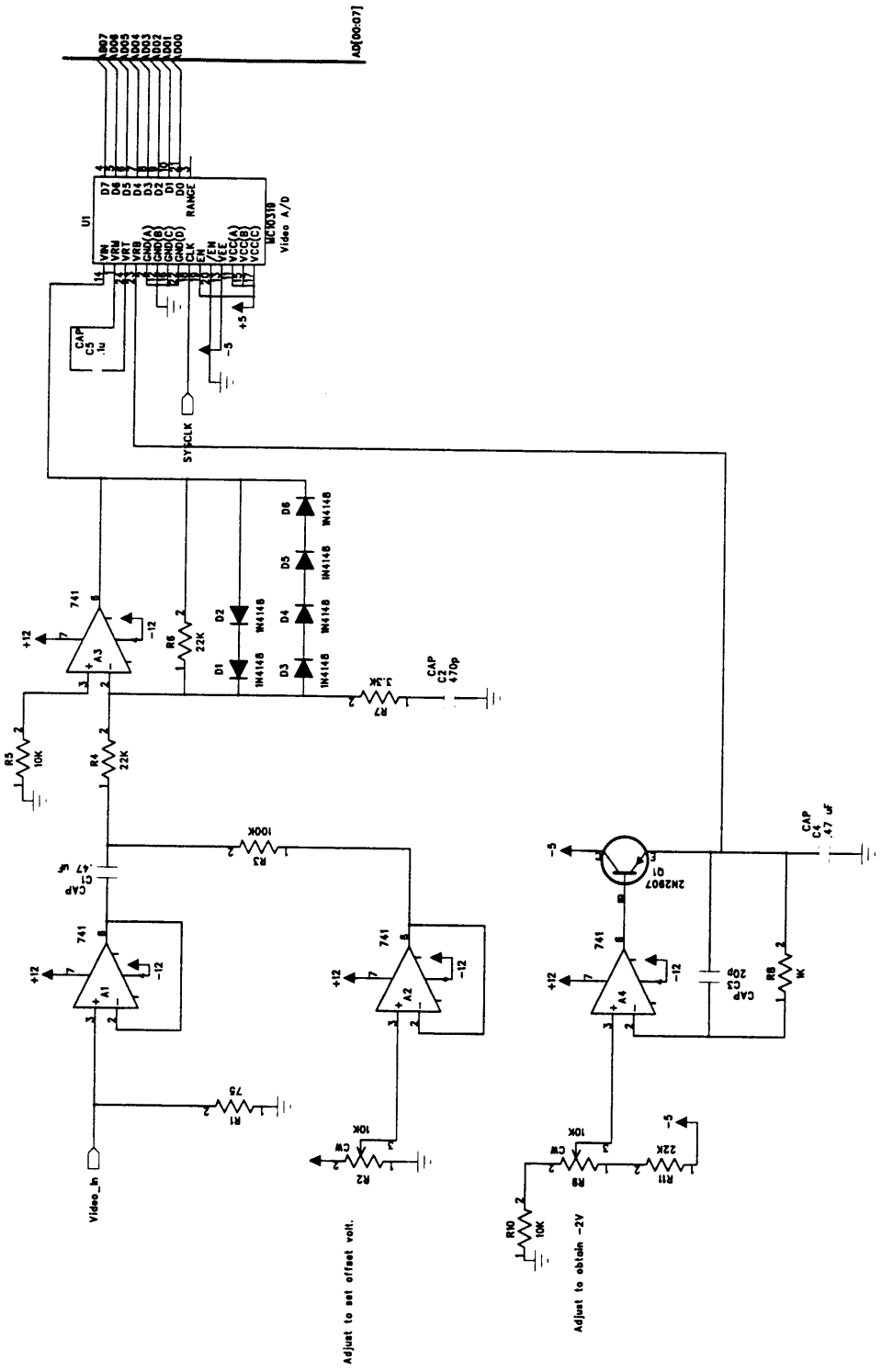

Figure 2-9: A/D Converter Schematic

2.5 Field Buffer

2.5.1 Static Memory

As the A/D converter samples a field, the field buffer stores the individual pixel values. Since the resolution of the sampled image is 256 pixels by 240 lines, the field buffer must be able to store 61,440 bytes. This amount of memory is realized by two 32 KB 62256 SRAMs, each of which stores half of the field. Each SRAM shares fifteen bits of address space, which are provided by the memory address registers. They also share the same eight bit input/output data pins. The output enable and write enable pins of the two SRAMs (/SRAMAOE, /SRAMBOE, /SRAMAWE, and /SRAMBWE) are individually controlled by the finite state machine controller. By individually controlling these pins, the two 32K SRAMs collectively act as a 64 KB SRAM. Pixels alternate between being stored in SRAM A and SRAM B by the toggling of the /SRAMAWE and /SRAMBWE signals. With this method, both SRAMs can share the same memory address and input/output pins, eliminating the need for a second memory address counter or additional bus wiring.

The chip select inputs for both SRAMs (/SRAMACS, /SRAMBCS) are tied to the system clock. Since the chip select pins are active low inputs, connecting them to the system clock only enables the SRAM during the low part of the clock period. This technique is useful for preventing bus contention and invalid data writes to the SRAM. At rising clock edges, clocked devices are in the process of turning on and turning off. Enabling the SRAM on the second half of a clock period ensures that the SRAM will not begin asserting a value on its bus while another device is in the process of turning off. Also, delaying the write cycle until the second half of the clock period guarantees values written to the SRAM will be stable during that time. The technique is especially convenient for use with the MC10319 A/D converter, which latches new values onto the bus 19 ns after the falling clock edge of an inverted system clock signal. Figure 2-10 shows a typical clock cycle of a SRAM write operation. A side-effect of enabling a read or write operation on the SRAM during the second half of the clock cycle is it decreases the maximum clock rate, mentioned in Section 2.3. Since the write delay for these particular SRAMs is 70 ns, the shortest clock period is around 140 ns.

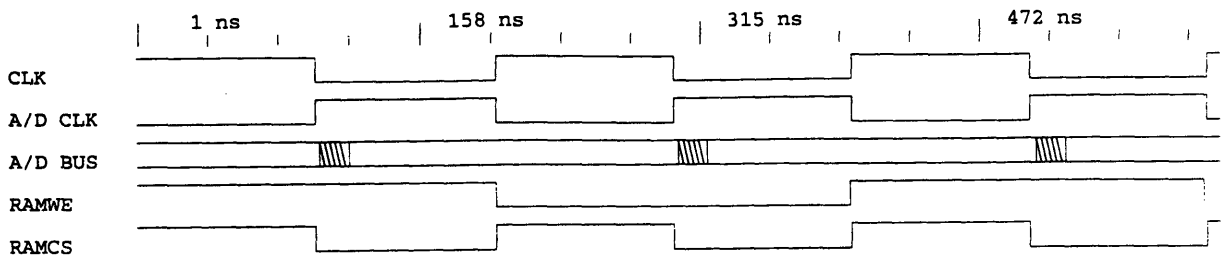

Figure 2-10: SRAM Write Timing Diagram

2.5.2 Memory Address Register

The memory address register is responsible for providing fifteen bit storage addresses to the field buffer. Locations in the field buffer are accessed sequentially; as a result, the memory address register has the ability to increment its address. Additionally, since the register only needs to access 30,720 locations instead of the 32768 possible locations with a 15 bit address, the register also has the ability to reset the counter to zero.

Cascading a series of counters such as a 74LS163 or 74LS169 would be a traditional approach to creating a 15 bit memory address register. However, this implementation is wasteful in terms of chip count and board space. Since the '163 and '169 are four bit counters, a 15 bit counter would require four of these chips cascaded together. This memory address register uses two 22V10 PALs to implement a 15 bit address. One PAL provides the lower seven bits of address and a carry output. The other PAL accepts the carry output and outputs the upper eight bits of address. Both PALs count sequentially with the MARCNT signal, and their addresses can be reset with the MARCLR signal.

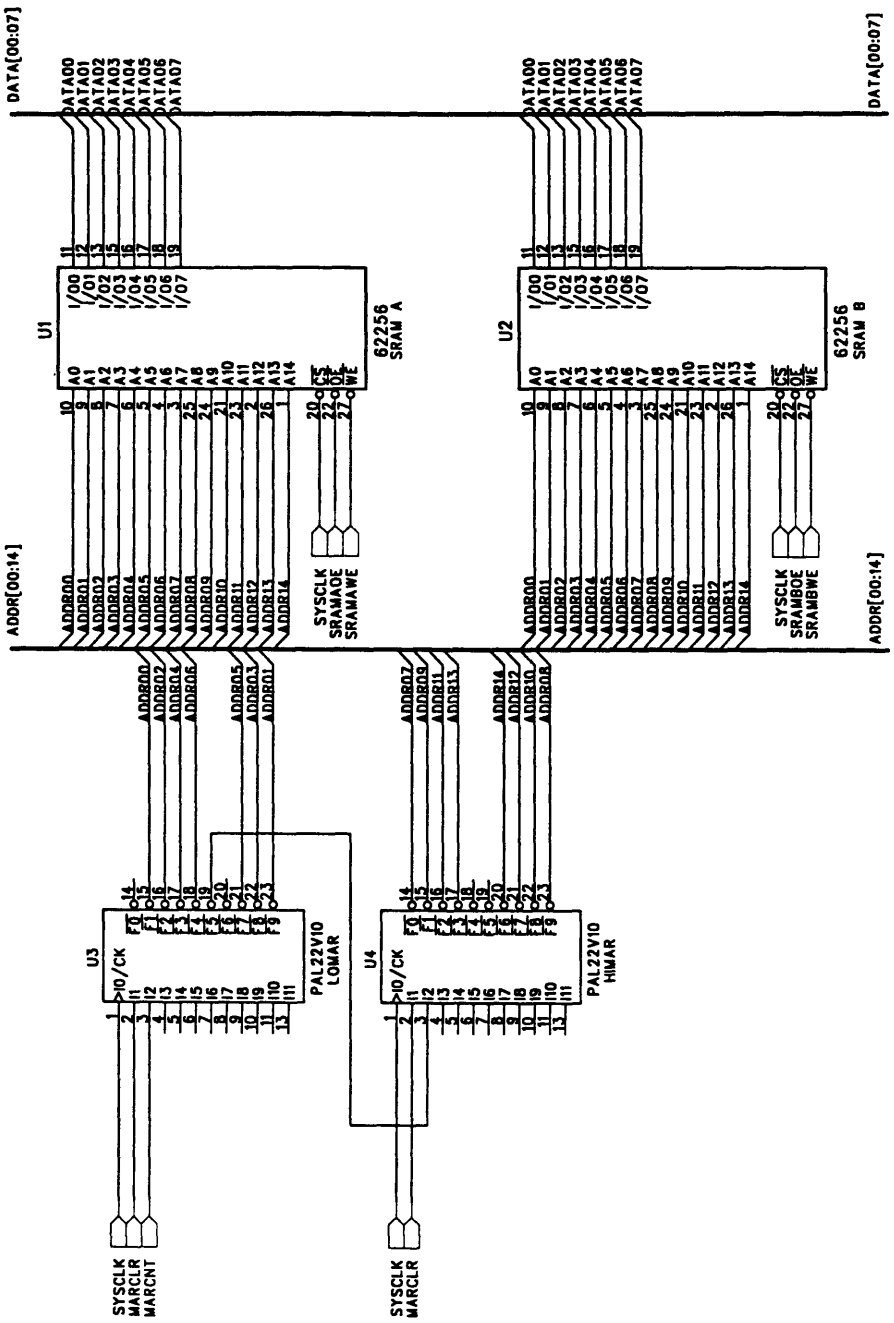

Figure 2-11: Field Buffer Schematic

2.6 Position Registers

2.6.1 Pixel Position Register

Despite the signals from the Sync Detection module such as Start of Data, these signals do not provide enough information about the current state of the video signal. The finite state machine controller needs to know when it has finished sampling 256 pixels on a horizontal line. The End of Data (EOD) signal from the Sync Detection module is not useful, for the signal could be received as late as the 260th pixel on a horizontal line, depending on the alignment of the sampling clock with the video signal. For this reason, the pixel position register keeps track of how many samples have been taken on a horizontal line. As the 256th sample is taken, the register outputs a status signal (EQ256) to the finite state machine controller. The FSM controller then uses this status signal to stop the horizontal line sampling cycle.

The pixel position register is an 8 bit counter implemented in a 22V10 PAL. This PAL is very similar to the ones used for the field buffer's memory address register; in fact, the register could be implemented as additional logic in the memory address register. However, the 22V10's product term and output pin limitations prevent incorporating the pixel position register in the memory address register. Instead, the pixel position register occupies its own 22V10 PAL. The logic expressions for the pixel position register are a modified form of the lower half of the memory address register. The carry out pin of the memory address register is replaced with an eighth counting bit on the pixel position register. The position register accepts a count signal (CNTCNT) which causes it to incrementally count. It also accepts a clear signal (CNTCLR), which is used at the end of every horizontal line. The output of the pixel position register, EQ256, is asserted when the counter's value is equal to 256, indicating the end of a line.

2.6.2 Line Position Register

The control FSM must also keep track of how many horizontal lines it has sampled. The NTSC video standard specifies that each field should have around 262 horizontal lines. However, only about 240 of these lines contain useful video information. As a result, a counter must be used to keep track of the number of sampled lines since the End of Field signal will come too late. The line position register is a PAL counter, similar to the pixel position register, that asserts the signal PAGEEND when the FSM has sampled 240 horizontal lines. Like the pixel position register, the line position register accepts a clear signal (PAGECLEAR) and an increment signal (PAGECNT). With these two signal, the control FSM can increment the counter at the end each horizontal line, check for the PAGEEND signal, and reset the counter once the end of the current field is reached.

2.7 Control Finite State Machine

The control FSM contains the "program" for running the operation of the video capture board. The FSM accepts status signals from the various modules on the board and from the interface, and it provides the appropriate assertion signals to run the modules. Figure 2-12 contains a flow chart for the FSM code, Table 1 contains a list of input signals to the FSM, and Table 2 contains a list of output signals from the FSM. The code is divided into two main parts, capture and transmission. In the capture part, the video capture board waits for a signal from the computer to start capturing an image. Once the signal is received, the board waits until the video signal contains the start of a new field. After the new field is detected, the board enters a loop where it stores the sampled values of a new line into the field buffer. This loop continues until the video signal has reached the end of the field; at this point, the FSM enters the transmission portion of its code. To transmit pixel values to the computer, the FSM uses a handshaking protocol to ensure that the computer receives valid data. Transmission continues until the FSM reaches the end of the field, where it returns to the idle state waiting for the computer's capture signal.

The FSM code is implemented in two 22V10 PALs containing thirteen states. The two PALs are cascaded by four state bits, which are output from one PAL and input to the other. The FSM is clocked by the 5.0 MHz system clock, and all outputs are registered with the system clock. Inputs to the FSM, listed in Table 1, are synchronized to the system clock before entering the FSM. Synchronizing the inputs prevents the FSM from entering a metastable state if an input should change immediately prior to a rising clock edge. Since the FSM contains thirteen defined states and uses four state bits, three states, called "trap states" are left undefined. If the FSM should ever enter one of these trap states, the system would probably become unstable and require power-cycling. To prevent this situation, the unused states are defined to transition to the IDLE state on the clock cycle following the transition to the trap state. With this precaution, a single field might be lost if the FSM entered a trap state during transmission, but the computer could request another field without power-cycling the board. Figure 2-13 contains a state diagram for the control FSM.

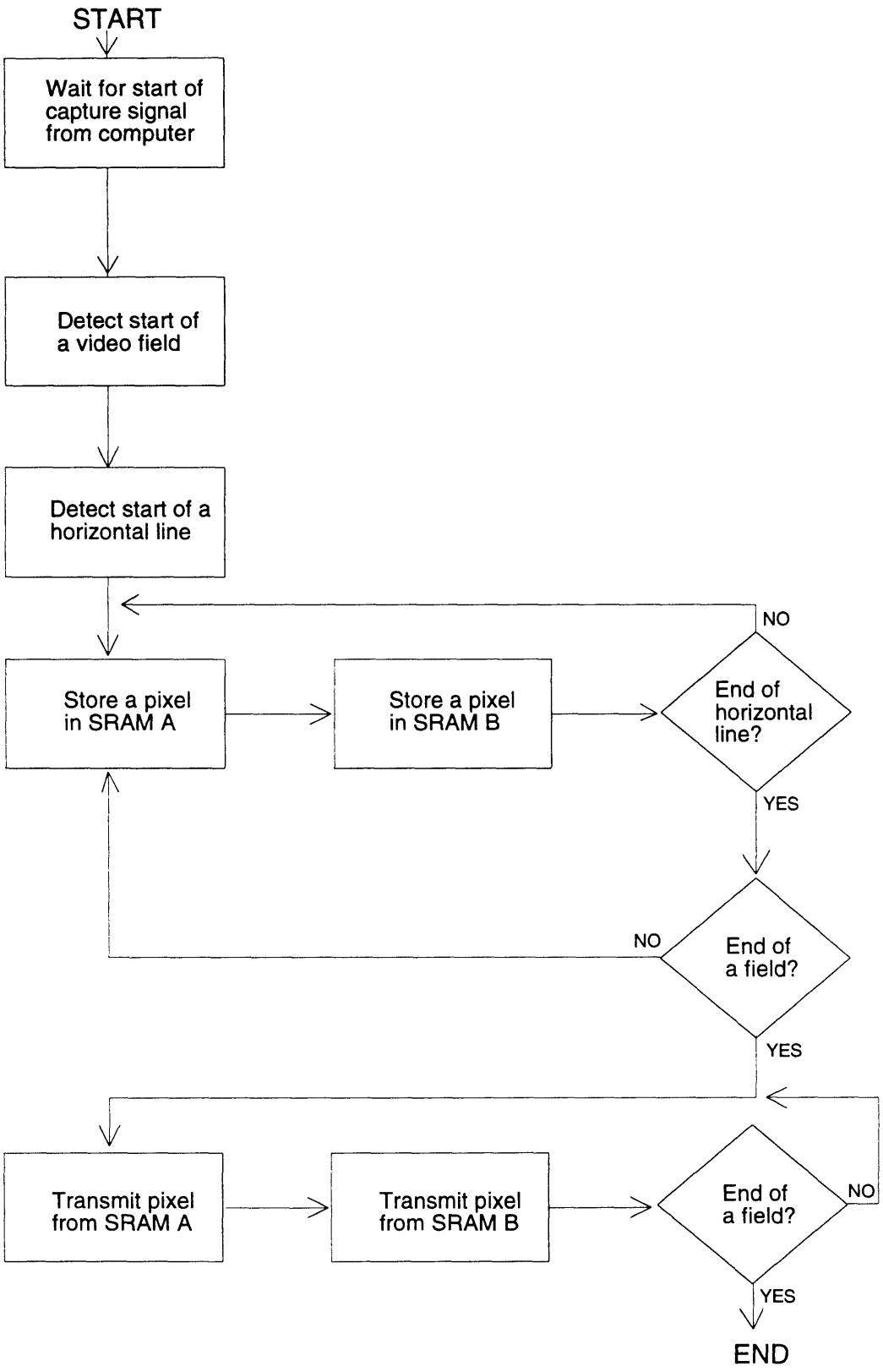

Figure 2-12: Control FSM Flow Chart

Pin Name	From	Purpose
CACK	Board Interface	Acknowledge signal from computer for handshaking protocol
CAPIMAGE	Board Interface	"Start Capture" signal from computer
EQ256	Pixel Position Register	Indicates when MAR is addressing end of a horizontal line
PAGEEND	Line Position Register	Indicates when MAR is addressing end of a video field
SOD	Sync Detection Module	Indicates when video signal is at the start of a horizontal line
SOF	Sync Detection Module	Indicates when video signal is at the start of a video field

Table 1: Control FSM Input Signals

Pin Name	To	Purpose
ADOE	Analog-to-Digital Tristate Buffer	Enables A/D tristate buffer
bACK	Board Interface	Acknowledge signal to computer for handshaking protocol
CNTCLEAR	Pixel Position Register	Clears Pixel Position Register
CNTCOUNT	Pixel Position Register	Increments Pixel Position Register
MARCLEAR	Memory Address Register	Clears Memory Address Register
MARCOUNT	Memory Address Register	Increments Memory Address Register
PAGECLEAR	Line Position Register	Clears Line Position Register
PAGECOUNT	Line Position Register	Increments Line Position Register
SRAMAOE	Field Buffer	Output Enables SRAM A
SRAMBOE	Field Buffer	Output Enables SRAM B
SRAMAWE	Field Buffer	Write Enables SRAM A
SRAMBWE	Field Buffer	Write Enables SRAM B

Table 2: Control FSM Output Signals

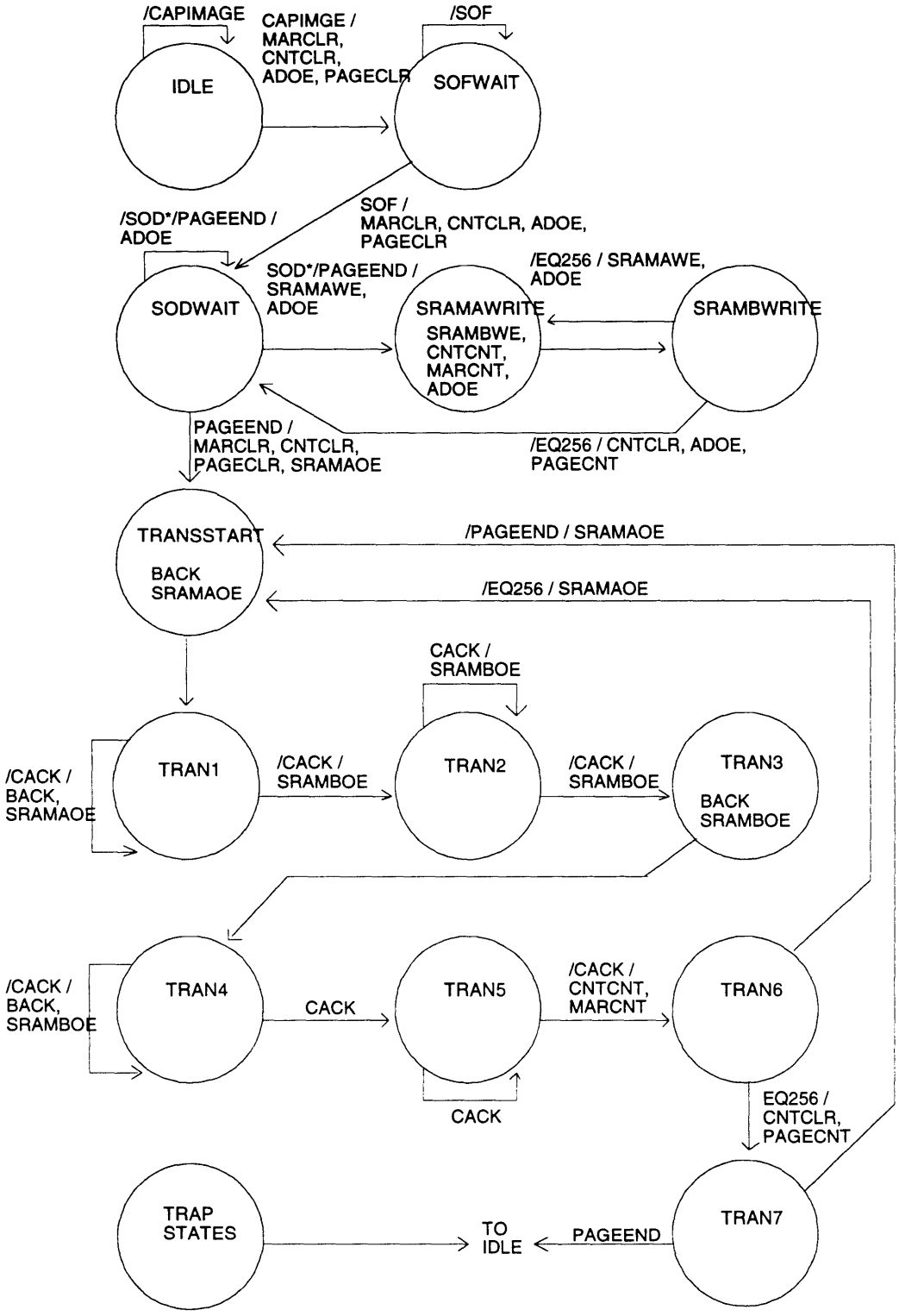

Figure 2-13: Control FSM State Diagram

3. System Interface

3.1 Overview

An IBM PC's standard parallel port provides a useful interface between the computer and the video capture board. The port is capable of relatively fast data transfer, which is convenient for the large amount of image data that must be transferred. Since the port has an external connector, installation of the board merely requires attaching a cable between the board and the computer. Additionally, the software support for the parallel port is easier than the support for a serial port, for the parallel port lacks the rigid timing requirements of the serial port. Finally, the board's parallel input/output bus allows it to be connected to not only the IBM PC but also to any computer which has general input/output buses available. This design makes it especially useful for control by embedded processors for use in specialized video applications.

3.2 IBM PC Parallel Port

Originally, the parallel port on the IBM PC family of computers was designed as a high-speed port for printers. For this reason, the port's main bus is an eight bit output bus, which is used to send character data to a printer. However, the parallel port specifications reserves eight input pins on the port for status signals from the printer. These status pins, with the eight bit output bus, can be used as a sixteen bit bus to non-printer products. The port appears to the computer as a set of addressed registers. Values can be read from the port by examining the contents of the register assigned to the port, and data can be written to the port by placing values in an addressed register. Since the parallel port lacks a defined communication protocol like a RS-232 serial port, the video capture board reserves pins on the port for a handshaking protocol with the computer.

A single parallel port occupies three addressed registers in the computer's address space. One address references an eight bit output bus on the port, called Dor. The other two addresses specify five bit input buses to the computer, called Dil and Dih. Since a computer can have more than one port, each port has its own set of unique addresses for the registers. Table 3 contains a list of addresses for the three registers according the port number. All register addresses are given in hexadecimal notation.

Register Name	Port #1 (LPT1)	Port #2 (LPT2)	Port #3 (LPT3)
Dor	0x3BC	0x378	0x278
Dil	0x3BE	0x37A	0x27A
Dih	0x3BD	0x379	0x279

Table 3: Parallel Port Register Addresses

The addressed registers are peculiar in that registers Dil and Dih are each only five bit input registers. Moreover, some pins on the input registers are defined as active low signals. To accommodate this definition, the video capture board inverts signals on the active low pins before transferring the values to the computer. Table 4 describes how each register is formatted. Doh, the eight bit output register, contains no inverted values and is conventionally mapped to the parallel port. Dil, however, only contains valid data from the port on the least significant five bits. Of these five bits, D3, D2 and D0 are inverted in the register. Similarly, Dih contains valid data on the five most significant bits, with D7 containing an inverted value. By combining the four most significant bits from the Dih register and the four least significant bits of the Dil register, an eight bit input bus from the board is formed.

Register Name	Bit 7 (MSB)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 (LSB)
Dor	D7	D6	D5	D4	D3	D2	D1	D0
Dil	X	X	X	D4	/D3	D2	/D1	/D0
Dih	/D7	D6	D5	D4	D3	X	X	X

Notes: X = Undefined

Inverted pins indicates that the signals are active low

Table 4: Parallel Port Register Specifications

The parallel port usually has a DB-25 connector on the computer to connect external devices to the port. Table 5 lists the pin assignments on the connector to the parallel port's three registers. The signal names are the original names of the pins, when the parallel port was exclusively used for printers. Pins 18 through 25 are connected to the computer's ground; the video capture board connects these pins to its own ground.

DB-25 Pin	Signal Name	Register Assignment	DB-25 Pin	Signal Name	Register Assignment
1	/STROBE	Dil - /D0	14	/AUTO FEED	Dil - /D1
2	D0	Dor - D0	15	/FAULT	Dih - D3
3	D1	Dor - D1	16	/INIT PTR	Dil - D2
4	D2	Dor - D2	17	/SEL PTR	Dil - /D3
5	D3	Dor - D3	18	GND	GND
6	D4	Dor - D4	19	GND	GND
7	D5	Dor - D5	20	GND	GND
8	D6	Dor - D6	21	GND	GND
9	D7	Dor - D7	22	GND	GND
10	/ACK	Dih - D6	23	GND	GND
11	BUSY	Dih - /D7	24	GND	GND
12	PAPER END	Dih - D5	25	GND	GND
13	PTR SLTD	Dih - D4			

Table 5: Parallel Port Pin Assignments

3.3 Board Interface

The video capture board interfaces to the cable from the computer with a male DB-25 connector. Wires from the DB-25 connector to the board contain output signals from the computer, input signals from the board, and common ground signals. The ground wires are connected to the video capture board's power supply ground to provide a common reference for the bus signals. Both the output bus and input bus are buffered by two 74LS244 tristate octal buffers before connecting to the DB-25 port. The buffers provide a means of driving the output bus from the computer to the board, and they also are capable of driving the eight bit data bus to the computer across a twenty foot parallel cable. Since several bits of the input bus require inversion before being sent to the computer, a set of 74LS04 inverters flip the input values before being buffered by the 74LS244s. Figure 3-1 contains a schematic of the board interface. The bACK signal from the board is the only signal not buffered by a 74LS244, since an additional 74LS244 would be required. Instead, two cascaded 74LS04 inverters buffer the bACK signal. This solution does not waste any board space, for the same 74LS04 used to invert the input signals is used to buffer the bACK signal.

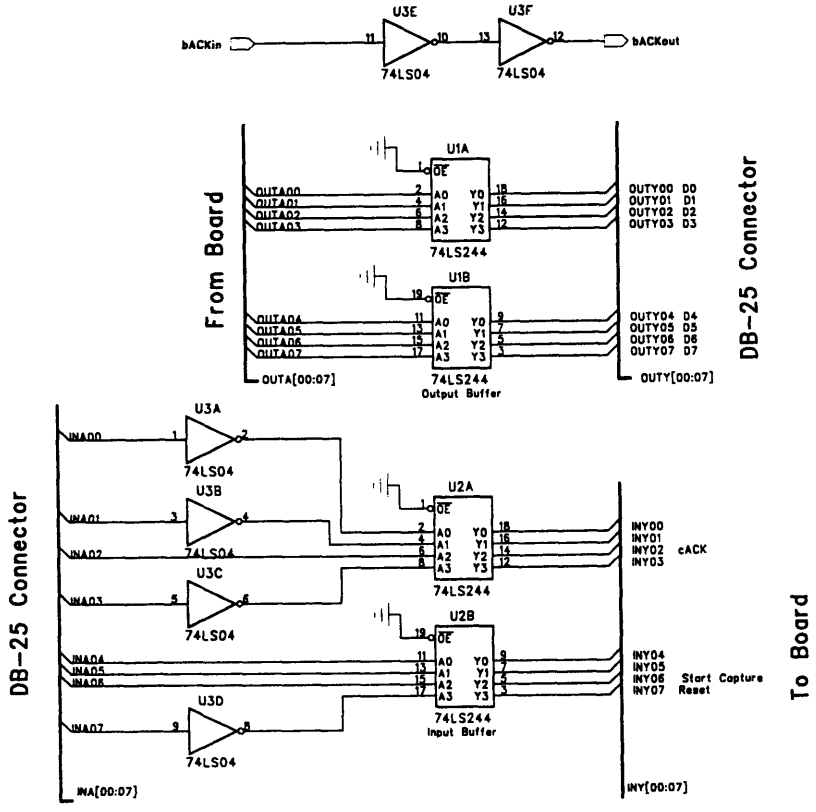

Figure 3-1: Board Interface Schematic

The eight bit input bus to the computer is used to transfer the eight bit grayscale picture data. Three bits of the eight bit output bus from the computer are used to control the video capture board. Table 6 contains a list of the register assignments for the three output signals from the computer. The cACK signal is used in the handshaking protocol described in Section 3.4. The Start Capture signal initiates a video field capture, and the Reset signal returns the board to an idle state even if the board is in the middle of a field capture. The Reset signal is particularly useful if an error occurs during data transmission between the board and the computer,and communication cannot be reestablished.

Output Signal	Register Assignment	Purpose
cACK	Dor - D2	Acknowledgment signal from computer used in handshaking protocol
Start Capture	Dor - D6	Tells video capture board to initiate capture
Reset	Dor - D7	Resets video capture board

Table 6: Computer Output Signals

3.4 Communication Protocol

After the computer sends a Start Capture signal to the video capture board, the board immediately begins storing image data in the field buffer once it has detected the start of a new field. Upon completion, the board begins transmitting the data to the computer using a ready/acknowledge handshaking protocol. Picture data is transmit to the computer starting with the upper-left hand corner of the image, and the transmission proceeds left to right through the image. When the board has a byte to send to the computer, it drives the value onto the interface input bus. On the following clock cycle, the board asserts the bACK signal and waits for the computer to acknowledge the data with the cACK signal. When the computer has received the data and has asserted cACK, the board deasserts bACK and waits for cACK to be deasserted. Once cACK is deasserted, the board can transmit the next byte when it is ready. Figure 3-2 shows a single transmission cycle. Most of the transmission time is taken up by waiting for the computer to receive the data. When the computer does respond by raising cACK and lowering it, the video capture board is ready to place a new value on the bus on the following clock cycle. Figure 3-3 shows a transmission sequence of multiple bytes on a horizontal row.

With this transmission method, data can be transmit between the board and the computer asynchronously without fear of losing a data value. However, if an error occurs with the assertion of either acknowledge signal, communication between the board and computer can breakdown. To reestablish communication, the Reset signal can be used to initialize the board and transmit the another field.

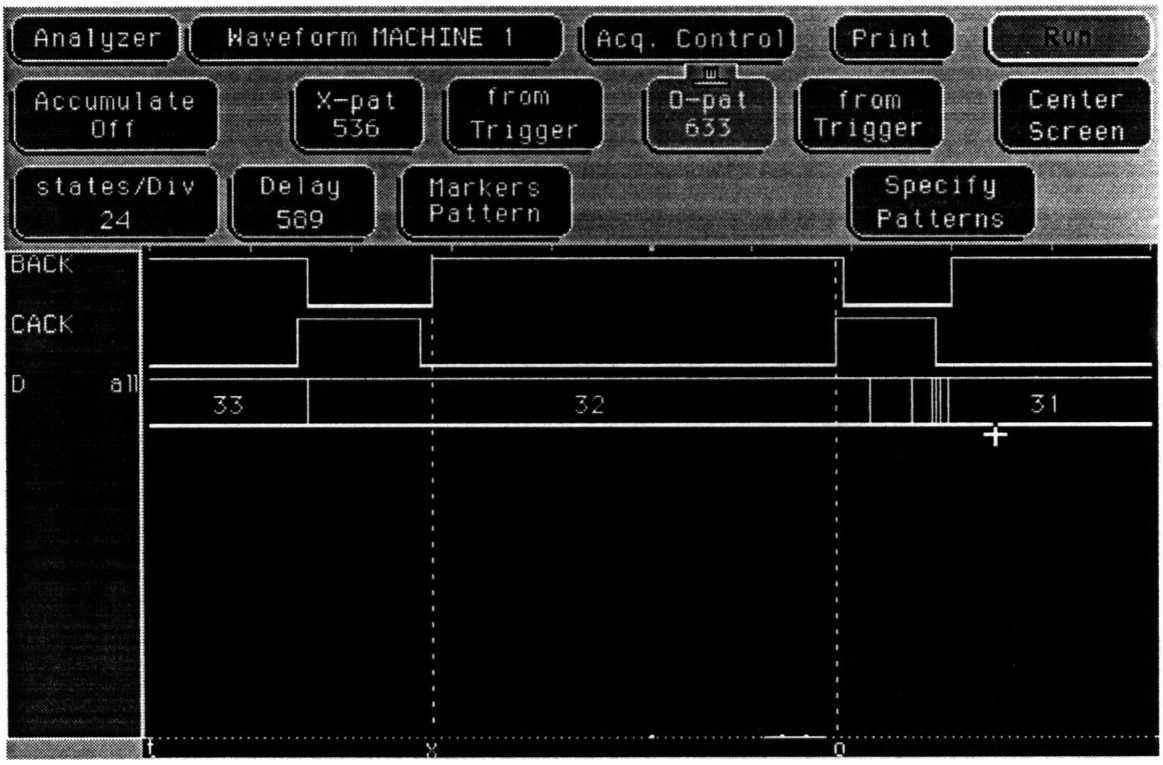

Figure 3-2: Transmission Protocol-Single Cycle

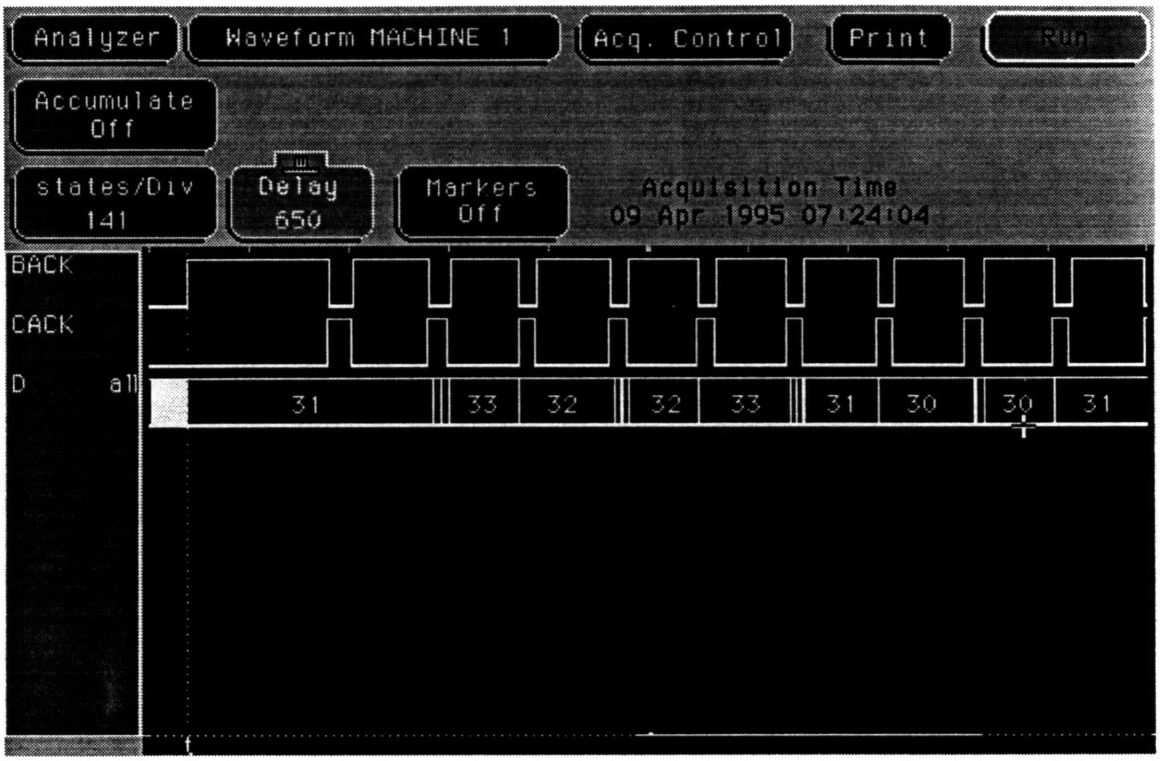

Figure 3-3: Transmission Protocol-Multiple Cycles

4. Software

4.1 Overview

An application called "Third Eye" utilitizes the features of the video capture board and demonstrates how it might be integrated into a retail surveillance system. The program has the ability to capture live video continuously, store the images to disk, and identify images with a time/date stamp. Since the application runs on IBM-compatible computers, it can be easily incorporated into most retail terminals. Furthermore, "Third Eye" is a multitasking Microsoft Windows application, allowing the computer to be used for more tasks than just surveillance. For example, a computer running "Third Eye" could also be used as a checkout terminal or a multimedia kiosk in a grocery store. To minimize the learning curve involved with using the program, "Third Eye" provides a very straightforward graphical interface and a minimal installation procedure.

4.2 System Requirements

Although the video capture board can be used with any type of computer with a parallel port, the software written for the board can only be run on IBM PC compatible computers. Additionally, the software requires that the computer have a bi-directional parallel port. Most IBM PC compatible computers have this type of parallel port, but often this mode must be specified in the BIOS setup screen. The software runs under Microsoft Windows 3.1 with a minimum of 2 MB of RAM. Any type of processor can be used, although the data transfer rate is largely dependent on the speed of the processor. To show the captured images in eight bit grayscale, a video card capable of supporting at least 256 colors at 640x480 resolution is required. A video accelerator card is recommended but not required.

4.3 Development Tools

"Third Eye" was developed on a 486DX33 computer using AMI BIOS with 8 MB of RAM. All of the source code was written in C/C++ under the Windows environment using Borland's Turbo C++ for Windows version 3.1. The software does not contain any C++ class structures, but a few of C++'s I/O functions were used for debugging purposes.

4.4 "Third Eye" Application

4.4.1 Overview

The "Third Eye" application provides a straightforward graphical interface to the video capture board. The main window displays most of the controls for the board, although functions can also be chosen from the menu bar. The application displays still images and live video from the camera in the main window. These images can be stored to disk in a common file format and loaded back into memory at a later time. The application can run as a background process so that other applications may be run simultaneously. Additionally, multiple copies of "Third Eye" can run at the same time so that video from multiple cameras may be captured if the computer has more than one parallel port. Figure 4-1 shows a layout of the main application window.

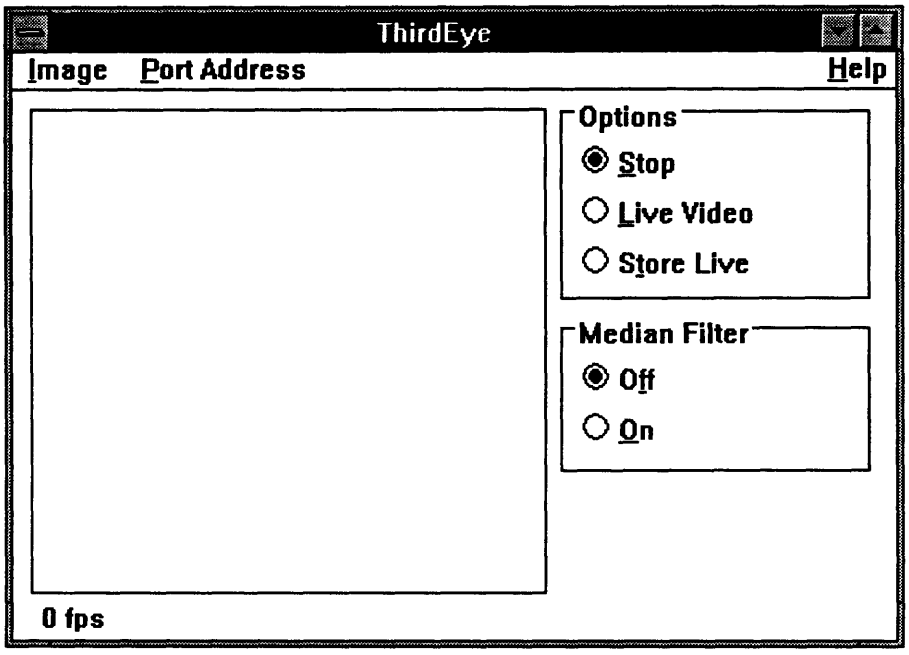

Figure 4-1: Third Eye Application Window

4.4.2 Program Setup

The program needs minimal setup, requiring only the first of the three register addresses for the parallel port connected to the video capture board. The desired register address can be changed by selecting "Port Address" on the main menu bar, which will display the addresses corresponding to LPT1, LPT2, and LPT3. Upon selecting the desired address, global variable *lgPortad* is updated with the new register address, and all subsequent video capture board commands are sent to the corresponding register address.

4.4.3 Capturing Images

The "Third Eye" application displays either still images or live video captured from the board on a 256x240 pixel area on the main window. Radio pushbuttons on the main window and items under "Image" on the menu bar provide choices for still or live video. The global Boolean variable *hbLive* holds the state of the user's request. If the user wishes to capture video, the program calls the **CaptureImage()** function, which initiates a field capture on the board. When the board has successfully stored an image into its field buffer, the function handles the computer's handshaking protocol for transferring data. As image data is received from the video capture board, it is stored in three 61,440 byte arrays in the global memory space. A globally declared array of memory handles called *hgImageBits* contains handles to these three arrays, so that any function may access the image data. By periodically polling the system clock during the data transmission, the function can detect if it has been waiting too long for an acknowledge signal from the board. If the function waits for more than one second for new data, then it aborts the communication protocol and resets the board. After displaying a warning window to the user, the function exits so that the problem with the board may be corrected.

After the **CaptureImage()** function stores the image data in the data arrays, the **DisplayImage()** function copies data from the array into a memory bitmap. From the memory bitmap, the Windows **BitBlt()** function can paste the image onto the screen very quickly, avoiding noticeable updating of the screen by the user. If live video capture has been selected, indicated by *hbLive*, the program repeatedly calls the **CaptureImage()** and **DisplayImage()** functions to update the screen as fast as possible. Otherwise, a still image remains on the main window.

Whenever an application under Microsoft Windows is maximized, resized, or moved, the application's window must be redrawn. When this situation occurs, the Windows kernel sends the application a WM_PAINT message, indicating that the image needs to be redisplayed. To update the screen when the "Third Eye" application receives this message, another call to the **DisplayImage()** is made, which redraws the current image in memory.

4.4.4 Loading and Storing Images

A useful feature for video surveillance is the ability to store captured images to disk and later display them. The "Third Eye" application has this feature and can store frames either once per request or continuously. The application stores captured images as 24 bit grayscale bitmaps in the uncompressed Windows BMP format. The Windows BMP format is convenient, for it is fairly universal and has a straightforward file structure. Images stored by the "Third Eye" application can be read by any other graphic viewer supporting the BMP format.

The user can load a stored image by selecting "Image" on the menu bar and clicking on "Load". A new window appears where the user can select the desired file from a list box. When the user has made a choice, a routine reads the file from disk and places the image data into a temporary memory buffer. Since the Windows BMP format has images stored left-to-right from the bottom of the image to the top, the contents of the temporary buffer are flipped horizontally to orient the image data correctly. When the temporary buffer contains the image data in a left-to-right, top-to-bottom format, it is copied to the global image memory array. Afterward, the program calls the **DisplayImage()** function to update the screen with the new contents of the memory array.

The user can save captured images to disk in a similar manner. Individual captured images can be saved to disk by selecting "Image" on the menu bar and clicking on "Save". A new window will appear requesting the file name for the saved image. When the user types in the filename and clicks on the "OK" button, the image will be saved to disk in the Windows BMP format. The image is saved to disk in the same manner that the images are loaded from disk. The Windows BMP format stores images from the bottom row to the top, so image data is stored into a temporary buffer backwards before being stored to disk.

The user also has the option of storing captured fields continuously, which is selected by clicking on the "Store live video" radio button on the main window. When the user chooses this option, the program repeatedly captures a field and stores it to disk with a date/time stamp as a filename. The filename is in the form of MMDDHHNN.0SS, where MM = Month, DD = Day, HH = Hour, NN = Minute, and SS = Second. The hour field uses twenty-four hour notation. With this convention, each field stored will have its own unique filename, and the filename identifies when the field was captured. This feature is very useful for a security application, for images taken can any time can quickly be found and viewed.

4.4.5 Noise Filtering

The analog-to-digital converter on the video capture board is very susceptible to slight fluctuations in the power supply. These fluctuations result in erroneous sample data and show up as "salt-and-pepper" noise in the sampled image. The "Third Eye" application provides a filtering technique called median filtering, which reduces "salt-and-pepper" noise without significantly affecting the actual picture. Median filtering is not a computationally expensive operating, but it does add approximately two-tenths of a second to the capture time on a 486DX33. For this reason, the filter can be turned off and on by clicking on the Median Filter radio button on the main window.

Median filtering is performed by replacing each pixel with the median of the values of itself and its surrounding neighbors. The "Third Eye" application uses one dimensional three pixel filtering, so that each pixel's new value is the median of itself and its horizontal neighbors. Two dimensional median filtering could also be performed, but the computation time becomes prohibitively expensive. A bubble sort algorithm sorts the pixel values of the three pixels, and the middle value of the sorted array becomes the new pixel value. Using the median filter either eliminates impulsive values in the image caused by noise or it reduces the noise's impact. However, median filtering also has the side effect of blurring high-frequency details in the image, such as edges and fine details.

4.5 Source Code

All of the source code files use standard WINDOWS.H constructs and can be compiled with any compiler with the Windows Standard Development Kit with little modification. The project file THIRDEYE.PRJ, however, can be used only with Borland's family of compilers. Table 7 contains a list of the source code files for the "Third Eye" application and their purpose. All variables declared in the source code files use a naming convention called "Hungarian Notation", popularized by Charles Simonyi of Microsoft Corporation. In this convention, a short prefix precedes the actual variable name to identify what type of data the variable represents.

Filename	Purpose
THIRDEYE.C	Contains main Windows program
THIRDEYE.H	Header file for THIRDEYE.C
THIRDEYE.RC	Resource file for THIRDEYE.C
THIRDEYE.RES	Compiled resource file for THIRDEYE.C
THIRDEYE.DEF	Definition file for THIRDEYE.C
THIRDEYE.ICO	Icon file for THIRDEYE.C
THIRDEYE.PRJ	Project file for THIRDEYE application
VIO.C	Board interface files for THIRDEYE.C
VIO.H	Header file for VIO.C

Table 7: Third Eye Source Code Files

5. Results

5.1 Overview

The video capture board and supporting software were completed according to the design specifications. The initial idea for the project came during work experience in the summer of 1994 with the Retail Products and Systems division of AT&T Global Information Solutions. The video capture board and interface was developed between September 1994 and January 1995. The "Third Eye" application was written between December 1994 and March 1995.

5.2 Video Noise

During development, a large amount of noise appeared in the sampled image. The noise was impulsive, showing up as salt-and-pepper values placed randomly in the image. Fluctuations in the analog power supply on the protoboard were probably responsible for much of this noise, despite bypass capacitors across power and ground. Median filtering in software removed most of the noise with little effect on the original picture, although the process increased the time between sampling images. Future work on the board could attempt to further regulate the power supplies to the analog to digital converter.

5.3 Image Sampling Rate

The image sampling rate is largely dependent on the processor speed of the computer. On a 486DX33, the fastest sampling rate was 0.9 seconds per image. When the images were median filtered, the time between successive images was 1.1 seconds per image. The sampling rate could be improved by modifying the **WinMain()** in the "Third Eye" application to work without the **PeekMessage()** function. This function allows the application to run as a background process, although it also impacts the application's performance even when the application is run as a foreground process. Removing **PeekMessage()** would improve the sampling rate, but "Third Eye" could not be run in the background according to the desired specifications.

5.4 Parallel Port Compatibility

Some multi-function peripheral cards with a parallel port are not capable of bi-directional data transfer. Other cards may have to be modified to support bi-directional transfer by changing jumpers on the card. Also, some computers require a special setting in their BIOS to support bi-directional transfer through their parallel port. The first peripheral card bought for the 486DX33 testbed did not work because it did not support this capability. The second card, a VESA local bus multi-function I/O card made by Data Technology, did support bi-directional transfer but only with a jumper modification.

Bibliography

Conger, James L. Windows API Bible. Corte Madera: Waite Group P, 1992.

Conger, James L. Windows Programming Plus. Corte Madera: Waite Group P, 1992.

Kernighan, Brian W., and Dennis M. Ritchie. The C Programming Language. Englewood Cliffs: Prentice Hall, 1988.

Lim, Jae S. Two-Dimensional Signal and Image Processing. Englewood Cliffs: Prentice Hall, 1990.

Mangieri, Adolph A. "Build A Parallel Printer Port I/O Interface." Computer Craft. April 1991: 54+.

Murray, James D., and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol: O'Reilly and Associates, 1994.

Portugal, Ronald J. "Serial/Parallel-Port Interface." Computer Craft October 1992: 32-43.

Appendix A: PAL Programming Files

A.1 Videoflag PAL Files

A.1.1 Videoflag .FSM File

```
' VIDEO ACQUISITION SYSTEM - Videoflag FSM
' MDH - 11/5/94
' This chip creates the SOD, SOF, EOD, EOF
' signals from the outputs of the sync separator.

begin_p

device 22v10

'inputs

negative HSYNC@4, VSYNC@3;

'outputs

register SOD@20, SOF@19, EOD@18, EOF@17;
register Q0@15, Q1@16;

end_p

ACTIVEDATA:             if (/HSYNC * /VSYNC) then stay;
                        else {
                                if (VSYNC) then {
                                        EOF;
                                        goto BETWEENFIELDS;
                                }
                                else {
                                        EOD;
                                        goto BETWEENLINES;
                                }
                        }

BETWEENFIELDS:          if (VSYNC) then stay;
                        else {
                                SOF;
                                goto ACTIVEDATA;
                        }

BETWEENLINES:           if (HSYNC * /VSYNC) then stay;
                        else {
                                if (VSYNC) then {
                                        EOF;
                                        goto BETWEENFIELDS;
                                }
                                else {
                                        SOD;
                                        goto ACTIVEDATA;
                                }
                        }

STATE3:                 goto ACTIVEDATA;
```

A.1.2 Videoflag .EQN file

```
' VIDEO ACQUISITION SYSTEM - VIDEOFLAG .EQN File

device 22v10

'inputs

negative HSYNC@4, VSYNC@3;

'outputs

register SOD@20, SOF@19, EOD@18, EOF@17;
register Q0@15, Q1@16;


register Q0;
register Q1;


Q0 =            VSYNC * /Q1 +
                VSYNC * /Q0;

Q1 =            HSYNC * /VSYNC * /Q0;

EOF =           VSYNC * /Q0;

EOD =           HSYNC * /VSYNC * /Q0 * /Q1;

SOF =           /VSYNC * Q0 * /Q1;

SOD =           /HSYNC * /VSYNC * /Q0 * Q1;
```

A.1.3 Videoflag .PAL File

```
22v10   C D D D D D D C C C
palasgn output from source file: videoflag.eqn

Massachusetts Institute of Technology
CLK         NC          /VSYNC      /HSYNC      NC          NC          NC
NC          NC          NC          NC          GND         NC          NC
Q0          Q1          EOF         EOD         SOF         SOD         NC
NC          NC          VCC

SOD :=  /VSYNC*Q1*/Q0*/HSYNC

SOF :=  /VSYNC*/Q1*Q0

EOD :=  /VSYNC*/Q1*/Q0*HSYNC

EOF :=  VSYNC*/Q0

Q0 :=   VSYNC*/Q0+
        VSYNC*/Q1

Q1 :=   /VSYNC*/Q0*HSYNC
```

A.1.4 Videoflag Chip Pinout

```
'palasgn pin assignments from file videoflag.eqn

                        PAL 0 : 22v10
                        -------------
                   CLK -|1         24|- VCC
                    NC -|2         23|- NC
                 /VSYNC -|3         22|- NC
                 /HSYNC -|4         21|- NC
                    NC -|5         20|- SOD
                    NC -|6         19|- SOF
                    NC -|7         18|- EOD
                    NC -|8         17|- EOF
                    NC -|9         16|- Q1
                    NC -|10        15|- Q0
                    NC -|11        14|- NC
                   GND -|12        13|- NC
                        -------------
```

A.2 Pixel Position Counter

A.2.1 Pixel Position Counter .FSM File

```
'This PAL is used to keep track of how many stores or retrieves have been
'done by the SRAM.  After each store or retrieve, a counter is incremented.
'When the counter reaches 256, the EQ256 signal is asserted, indicating that
'the SRAM is addressing a pixel at the end of a line.

begin_p

device 22v10

'inputs
positive CNTCLEAR@2, CNTCNT@3;

'outputs
register A0@15, A1@23, A2@16, A3@22, A4@17, A5@21, A6@18;
positive EQ256@14;

end_p

EQ256 = A6 * A5 * A4 * A3 * A2 * A1 * A0;

if CNTCLEAR then {
        A0 = 0;
        A1 = 0;
        A2 = 0;
        A3 = 0;
        A4 = 0;
        A5 = 0;
        A6 = 0;
} else {
        if /CNTCNT then {
                A0 = A0;
                A1 = A1;
                A2 = A2;
                A3 = A3;
                A4 = A4;
                A5 = A5;
                A6 = A6;
        } else {
                A0 = /A0;
                if /A0 then {
                        A1 = A1;
                        A2 = A2;
                        A3 = A3;
                        A4 = A4;
                        A5 = A5;
                        A6 = A6;
                } else {
                        A1 = /A1;
                        if /A1 then {
                                A2 = A2;
                                A3 = A3;
                                A4 = A4;
                                A5 = A5;
                                A6 = A6;
                        } else {
                                A2 = /A2;
                                if /A2 then {
                                        A3 = A3;
                                        A4 = A4;
                                        A5 = A5;
                                        A6 = A6;
                                } else {
                                        A3 = /A3;
                                        if /A3 then {
                                                A4 = A4;
                                                A5 = A5;
                                                A6 = A6;
                                        } else {
```

```
A4 = /A4;
if /A4 then {
        A5 = A5;
        A6 = A6;
} else {
        A5 = /A5;
        if /A5 then {
                A6 = A6;
        } else {
                A6 = /A6;
        }}}}}}}
```

A.2.2 Pixel Position Counter .EQN File

```
device 22v10

'inputs
positive CNTCLEAR@2, CNTCNT@3;

'outputs
register A0@15, A1@23, A2@16, A3@22, A4@17, A5@21, A6@18;
positive EQ256@14;


EQ256 =         A0 * A1 * A2 * A3 * A4 * A5 * A6;

A0 =            A0 * /CNTCLEAR * /CNTCNT +
                /A0 * /CNTCLEAR * CNTCNT;

A1 =            A0 * /A1 * /CNTCLEAR * CNTCNT +
                A1 * /CNTCLEAR * /CNTCNT +
                /A0 * A1 * /CNTCLEAR;

A2 =            /A1 * A2 * /CNTCLEAR +
                A0 * A1 * /A2 * /CNTCLEAR * CNTCNT +
                A2 * /CNTCLEAR * /CNTCNT +
                /A0 * A2 * /CNTCLEAR;

A3 =            /A1 * A3 * /CNTCLEAR +
                /A2 * A3 * /CNTCLEAR +
                A0 * A1 * A2 * /A3 * /CNTCLEAR * CNTCNT +
                A3 * /CNTCLEAR * /CNTCNT +
                /A0 * A3 * /CNTCLEAR;

A4 =            /A1 * A4 * /CNTCLEAR +
                /A2 * A4 * /CNTCLEAR +
                /A3 * A4 * /CNTCLEAR +
                A0 * A1 * A2 * A3 * /A4 * /CNTCLEAR * CNTCNT +
                A4 * /CNTCLEAR * /CNTCNT +
                /A0 * A4 * /CNTCLEAR;

A5 =            /A1 * A5 * /CNTCLEAR +
                /A2 * A5 * /CNTCLEAR +
                /A3 * A5 * /CNTCLEAR +
                /A4 * A5 * /CNTCLEAR +
                A0 * A1 * A2 * A3 * A4 * /A5 * /CNTCLEAR * CNTCNT +
                A5 * /CNTCLEAR * /CNTCNT +
                /A0 * A5 * /CNTCLEAR;

A6 =            /A1 * A6 * /CNTCLEAR +
                /A2 * A6 * /CNTCLEAR +
                /A3 * A6 * /CNTCLEAR +
                /A4 * A6 * /CNTCLEAR +
                /A5 * A6 * /CNTCLEAR +
                A0 * A1 * A2 * A3 * A4 * A5 * /A6 * /CNTCLEAR * CNTCNT +
                A6 * /CNTCLEAR * /CNTCNT +
                /A0 * A6 * /CNTCLEAR;
```

A.2.3 Pixel Position Counter .PAL File

```
22v10   C D D D D C C D D D
palasgn output from source file: count128.eqn

Massachusetts Institute of Technology
CLK         CNTCLEAR   CNTCNT     NC         NC         NC         NC
NC          NC         NC         NC         GND        NC         EQ256
A0          A2         A4         A6         NC         NC         A5
A3          A1         VCC

A0 :=   A0*/CNTCLEAR*/CNTCNT+
        /A0*/CNTCLEAR*CNTCNT

A1 :=   A1*/CNTCLEAR*/CNTCNT+
        A0*/A1*/CNTCLEAR*CNTCNT+
        /A0*A1*/CNTCLEAR

A2 :=   /A1*A2*/CNTCLEAR+
        A2*/CNTCLEAR*/CNTCNT+
        A0*A1*/A2*/CNTCLEAR*CNTCNT+
        /A0*A2*/CNTCLEAR

A3 :=   /A1*A3*/CNTCLEAR+
        /A2*A3*/CNTCLEAR+
        A3*/CNTCLEAR*/CNTCNT+
        A0*A1*A2*/A3*/CNTCLEAR*CNTCNT+
        /A0*A3*/CNTCLEAR

A4 :=   /A1*A4*/CNTCLEAR+
        /A2*A4*/CNTCLEAR+
        /A3*A4*/CNTCLEAR+
        A4*/CNTCLEAR*/CNTCNT+
        A0*A1*A2*A3*/A4*/CNTCLEAR*CNTCNT+
        /A0*A4*/CNTCLEAR

A5 :=   /A1*A5*/CNTCLEAR+
        /A2*A5*/CNTCLEAR+
        /A3*A5*/CNTCLEAR+
        /A4*A5*/CNTCLEAR+
        A5*/CNTCLEAR*/CNTCNT+
        A0*A1*A2*A3*A4*/A5*/CNTCLEAR*CNTCNT+
        /A0*A5*/CNTCLEAR

A6 :=   /A1*A6*/CNTCLEAR+
        /A2*A6*/CNTCLEAR+
        /A3*A6*/CNTCLEAR+
        /A4*A6*/CNTCLEAR+
        /A5*A6*/CNTCLEAR+
        A6*/CNTCLEAR*/CNTCNT+
        A0*A1*A2*A3*A4*A5*/A6*/CNTCLEAR*CNTCNT+
        /A0*A6*/CNTCLEAR

EQ256 =         A0*A1*A2*A3*A4*A5*A6
```

A.2.4 Pixel Position Counter Chip Pinout

```
'palasgn pin assignments from file count128.eqn

                        PAL 0 : 22v10
                        -------------
                  CLK -|1         24|- VCC
              CNTCLEAR -|2         23|- A1
                CNTCNT -|3         22|- A3
                    NC -|4         21|- A5
                    NC -|5         20|- NC
                    NC -|6         19|- NC
                    NC -|7         18|- A6
                    NC -|8         17|- A4
                    NC -|9         16|- A2
                    NC -|10        15|- A0
                    NC -|11        14|- EQ256
                   GND -|12        13|- NC
                        -------------
```

A.3 Line Position Counter

A.3.1 Line Position Counter .FSM File

```
'This PAL is used to keep track of how many lines have been stored into the
'SRAM.  After each line is stored, a counter is incremented.
'When the counter reaches 240, the EQ240 signal is asserted, indicating that
'the SRAM has stored the entire field.

begin_p

device 22v10

'inputs
positive FIELDCLEAR@2, FIELDCNT@3, CACKIN@4;

'outputs
register A0@15, A1@23, A2@16, A3@22, A4@17, A5@21, A6@18, A7@20, CACKOUT@19;
positive EQ240@14;

end_p

EQ240 = A4 * A5 * A6 * A7;

CACKOUT = CACKIN;


if FIELDCLEAR then {
        A0 = 0;
        A1 = 0;
        A2 = 0;
        A3 = 0;
        A4 = 0;
        A5 = 0;
        A6 = 0;
        A7 = 0;
} else {
        if /FIELDCNT then {
                A0 = A0;
                A1 = A1;
                A2 = A2;
                A3 = A3;
                A4 = A4;
                A5 = A5;
                A6 = A6;
                A7 = A7;
        } else {
                A0 = /A0;
                if /A0 then {
                        A1 = A1;
                        A2 = A2;
                        A3 = A3;
                        A4 = A4;
                        A5 = A5;
                        A6 = A6;
                        A7 = A7;
                } else {
                        A1 = /A1;
                        if /A1 then {
                                A2 = A2;
                                A3 = A3;
                                A4 = A4;
                                A5 = A5;
                                A6 = A6;
                                A7 = A7;
                        } else {
                                A2 = /A2;
                                if /A2 then {
                                        A3 = A3;
                                        A4 = A4;
                                        A5 = A5;
                                        A6 = A6;
                                        A7 = A7;
```

```
} else {
        A3 = /A3;
        if /A3 then {
                A4 = A4;
                A5 = A5;
                A6 = A6;
                A7 = A7;
        } else {
                A4 = /A4;
                if /A4 then {
                        A5 = A5;
                        A6 = A6;
                        A7 = A7;
                } else {
                        A5 = /A5;
                        if /A5 then {
                                A6 = A6;
                                A7 = A7;
                        } else {
                                A6 = /A6;
                                if /A6 then {
                                        A7 = A7;
                                } else {
                                        A7 = /A7;
                                }}}}}}}}}
```

A.3.2 Line Position Counter .EQN File

```
device 22v10

'inputs
positive FIELDCLEAR@2, FIELDCNT@3, CACKIN@4;

'outputs
register A0@15, A1@23, A2@16, A3@22, A4@17, A5@21, A6@18, A7@20, CACKOUT@19;
positive EQ240@14;



EQ240 =         A4 * A5 * A6 * A7;

A4 =            A4 * /FIELDCLEAR * /FIELDCNT +
                A4 * /FIELDCLEAR * /A0 +
                A4 * /FIELDCLEAR * /A1 +
                A4 * /FIELDCLEAR * /A2 +
                A4 * /FIELDCLEAR * /A3 +
                /A4 * /FIELDCLEAR * FIELDCNT * A0 * A1 * A2 * A3;

A5 =            A4 * /A5 * /FIELDCLEAR * FIELDCNT * A0 * A1 * A2 * A3 +
                A5 * /FIELDCLEAR * /FIELDCNT +
                A5 * /FIELDCLEAR * /A0 +
                A5 * /FIELDCLEAR * /A1 +
                A5 * /FIELDCLEAR * /A2 +
                A5 * /FIELDCLEAR * /A3 +
                /A4 * A5 * /FIELDCLEAR;

A6 =            /A5 * A6 * /FIELDCLEAR +
                A4 * A5 * /A6 * /FIELDCLEAR * FIELDCNT * A0 * A1 * A2 * A3 +
                A6 * /FIELDCLEAR * /FIELDCNT +
                A6 * /FIELDCLEAR * /A0 +
                A6 * /FIELDCLEAR * /A1 +
                A6 * /FIELDCLEAR * /A2 +
                A6 * /FIELDCLEAR * /A3 +
                /A4 * A6 * /FIELDCLEAR;

A7 =            /A5 * A7 * /FIELDCLEAR +
                /A6 * A7 * /FIELDCLEAR +
                A4 * A5 * A6 * /A7 * /FIELDCLEAR * FIELDCNT * A0 * A1 *
                     A2 * A3 +
                A7 * /FIELDCLEAR * /FIELDCNT +
                A7 * /FIELDCLEAR * /A0 +
                A7 * /FIELDCLEAR * /A1 +
                A7 * /FIELDCLEAR * /A2 +
                A7 * /FIELDCLEAR * /A3 +
                /A4 * A7 * /FIELDCLEAR;

CACKOUT =       CACKIN;

A0 =            /FIELDCLEAR * /FIELDCNT * A0 +
                /FIELDCLEAR * FIELDCNT * /A0;

A1 =            /FIELDCLEAR * /FIELDCNT * A1 +
                /FIELDCLEAR * /A0 * A1 +
                /FIELDCLEAR * FIELDCNT * A0 * /A1;

A2 =            /FIELDCLEAR * /FIELDCNT * A2 +
                /FIELDCLEAR * /A0 * A2 +
                /FIELDCLEAR * /A1 * A2 +
                /FIELDCLEAR * FIELDCNT * A0 * A1 * /A2;

A3 =            /FIELDCLEAR * /FIELDCNT * A3 +
                /FIELDCLEAR * /A0 * A3 +
                /FIELDCLEAR * /A1 * A3 +
                /FIELDCLEAR * /A2 * A3 +
                /FIELDCLEAR * FIELDCNT * A0 * A1 * A2 * /A3;
```

A.3.3 Line Position Counter .PAL File

```
22v10   C D D D D D D D D D
palasgn output from source file: fieldcount.eqn

Massachusetts Institute of Technology
CLK         FIELDCLEAR FIELDCNT  CACKIN     NC          NC          NC
NC          NC         NC        NC         GND         NC          EQ240
A0          A2         A4        A6         CACKOUT     A7          A5
A3          A1         VCC

A0 :=   /FIELDCLEAR*/FIELDCNT*A0+
        /FIELDCLEAR*FIELDCNT*/A0

A1 :=   /FIELDCLEAR*/FIELDCNT*A1+
        /FIELDCLEAR*/A0*A1+
        /FIELDCLEAR*FIELDCNT*A0*/A1

A2 :=   /FIELDCLEAR*/FIELDCNT*A2+
        /FIELDCLEAR*/A0*A2+
        /FIELDCLEAR*/A1*A2+
        /FIELDCLEAR*FIELDCNT*A0*A1*/A2

A3 :=   /FIELDCLEAR*/FIELDCNT*A3+
        /FIELDCLEAR*/A0*A3+
        /FIELDCLEAR*/A1*A3+
        /FIELDCLEAR*/A2*A3+
        /FIELDCLEAR*FIELDCNT*A0*A1*A2*/A3

A4 :=   A4*/FIELDCLEAR*/FIELDCNT+
        A4*/FIELDCLEAR*/A0+
        A4*/FIELDCLEAR*/A1+
        A4*/FIELDCLEAR*/A2+
        A4*/FIELDCLEAR*/A3+
        /A4*/FIELDCLEAR*FIELDCNT*A0*A1*A2*A3

A5 :=   A5*/FIELDCLEAR*/FIELDCNT+
        A5*/FIELDCLEAR*/A0+
        A5*/FIELDCLEAR*/A1+
        A5*/FIELDCLEAR*/A2+
        A5*/FIELDCLEAR*/A3+
        A4*/A5*/FIELDCLEAR*FIELDCNT*A0*A1*A2*A3+
        /A4*A5*/FIELDCLEAR

A6 :=   /A5*A6*/FIELDCLEAR+
        A6*/FIELDCLEAR*/FIELDCNT+
        A6*/FIELDCLEAR*/A0+
        A6*/FIELDCLEAR*/A1+
        A6*/FIELDCLEAR*/A2+
        A6*/FIELDCLEAR*/A3+
        A4*A5*/A6*/FIELDCLEAR*FIELDCNT*A0*A1*A2*A3+
        /A4*A6*/FIELDCLEAR

A7 :=   /A5*A7*/FIELDCLEAR+
        /A6*A7*/FIELDCLEAR+
        A7*/FIELDCLEAR*/FIELDCNT+
        A7*/FIELDCLEAR*/A0+
        A7*/FIELDCLEAR*/A1+
        A7*/FIELDCLEAR*/A2+
        A7*/FIELDCLEAR*/A3+
        A4*A5*A6*/A7*/FIELDCLEAR*FIELDCNT*A0*A1*A2*A3+
        /A4*A7*/FIELDCLEAR

CACKOUT :=      CACKIN

EQ240 =         A4*A5*A6*A7
```

A.3.4 Line Position Counter Chip Pinout

```
'palasgn pin assignments from file fieldcount.eqn

                    PAL 0 : 22v10
                    -------------
           CLK -|1         24|- VCC
   FIELDCLEAR -|2         23|- A1
     FIELDCNT -|3         22|- A3
       CACKIN -|4         21|- A5
           NC -|5         20|- A7
           NC -|6         19|- CACKOUT
           NC -|7         18|- A6
           NC -|8         17|- A4
           NC -|9         16|- A2
           NC -|10        15|- A0
           NC -|11        14|- EQ240
          GND -|12        13|- NC
                    -------------
```

A.4 Memory Address Register

A.4.1 Memory Address Register .FSM File

```
' VIDEO ACQUISITION SYSTEM - Low Byte of 16-bit MAR
' A0 to A6 are the low bits for the basic 16-bit MAR.  It can clear to zero
' and increment.  CO is the carry out to the high byte of the MAR.

begin_p

device 22v10

'inputs
positive Clear@2, Cnt@3;


'outputs
register A0@15, A1@23, A2@16, A3@22, A4@17, A5@21, A6@18;
positive CO@19;

end_p

if (Clear) then        {
                       A0 = 0;
                       A1 = 0;
                       A2 = 0;
                       A3 = 0;
                       A4 = 0;
                       A5 = 0;
                       A6 = 0;
                       CO = 0;
               }

if (/Clear) then {
               if (/Cnt) then {
                                      A0 = A0;
                                      A1 = A1;
                                      A2 = A2;
                                      A3 = A3;
                                      A4 = A4;
                                      A5 = A5;
                                      A6 = A6;
                                      CO = 0;
                                }
               else { A0 = /A0;
               if (/A0) then  {
                                      A1 = A1;
                                      A2 = A2;
                                      A3 = A3;
                                      A4 = A4;
                                      A5 = A5;
                                      A6 = A6;
                                      CO = 0;
                                }
               else {                 A1 = /A1;
               if (/A1) then  {
                                      A2 = A2;
                                      A3 = A3;
                                      A4 = A4;
                                      A5 = A5;
                                      A6 = A6;
                                      CO = 0;
                                }
               else {                 A2 = /A2;
               if (/A2) then  {
                                      A3 = A3;
                                      A4 = A4;
                                      A5 = A5;
                                      A6 = A6;
                                      CO = 0;
                                }
               else {                 A3 = /A3;
```

```
                if (/A3) then  {
                                        A4 = A4;
                                        A5 = A5;
                                        A6 = A6;
                                        CO = 0;
                               }
                else {                  A4 = /A4;
                if (/A4) then  {
                                        A5 = A5;
                                        A6 = A6;
                                        CO = 0;
                               }
                else {                  A5 = /A5;
                if (/A5) then  {
                                        A6 = A6;
                                        CO = 0;
                               }
                else {                  A6 = /A6;
                if (/A6) then  {
                                        CO = 0;
                               }
                else           {
                                        CO = 1;
                }}}}}}}}}
```

```
' VIDEO ACQUISITION SYSTEM - High byte of 16-bit MAR
' A7 to A15 are the high bits for the basic 16 bit address MAR.
' It can clear to zero and increments when CI is hi.  CI is the carry
' input from the low byte.

begin_p

device 22v10

'inputs
positive Clear@2, CI@3;


'outputs
register A7@14, A8@23, A9@15, A10@22, A11@16, A12@21, A13@17, A14@20, A15@18;


end_p


if (Clear) then {
                        A15 = 0;
                        A14 = 0;
                        A13 = 0;
                        A12 = 0;
                        A11 = 0;
                        A10 = 0;
                        A9  = 0;
                        A8  = 0;
                        A7  = 0;
                }
if (/Clear) then {

        if (/CI) then  {
                                A7  = A7;
                                A8  = A8;
                                A9  = A9;
                                A10 = A10;
                                A11 = A11;
                                A12 = A12;
                                A13 = A13;
                                A14 = A14;
                                A15 = A15;
                        }

        else {  A7 = /A7;
                if (/A7) then  {
                                        A8  = A8;
```

```
                                A9  = A9;
                                A10 = A10;
                                A11 = A11;
                                A12 = A12;
                                A13 = A13;
                                A14 = A14;
                                A15 = A15;
                            }
else {  A8 = /A8;
        if (/A8) then  {
                                A9  = A9;
                                A10 = A10;
                                A11 = A11;
                                A12 = A12;
                                A13 = A13;
                                A14 = A14;
                                A15 = A15;
                            }
else {  A9 = /A9;
        if (/A9) then  {
                                A10 = A10;
                                A11 = A11;
                                A12 = A12;
                                A13 = A13;
                                A14 = A14;
                                A15 = A15;
                            }
else {  A10 = /A10;
        if (/A10) then {
                                A11 = A11;
                                A12 = A12;
                                A13 = A13;
                                A14 = A14;
                                A15 = A15;
                            }
else {  A11 = /A11;
        if (/A11) then {
                                A12 = A12;
                                A13 = A13;
                                A14 = A14;
                                A15 = A15;
                            }
else {  A12 = /A12;
        if (/A12) then {
                                A13 = A13;
                                A14 = A14;
                                A15 = A15;
                            }
else {  A13 = /A13;
        if (/A13) then {
                                A14 = A14;
                                A15 = A15;
                            }
else {  A14 = /A14;
        if (/A14) then {
                                A15 = A15;
                            }
else {  A15 = /A15;     }
}}}}}}}}}
```

A.4.2 Memory Address Register . EQN File

```
' VIDEO ACQUISITION SYSTEM - Low MAR .EQN File
device 22v10

'inputs
positive Clear@2, Cnt@3;


'outputs
register A0@15, A1@23, A2@16, A3@22, A4@17, A5@21, A6@18;
positive CO@19;


A0 =            /Clear * /Cnt * A0 +
                /Clear * Cnt * /A0;

A1 =            /Clear * /Cnt * A1 +
                /Clear * /A0 * A1 +
                /Clear * Cnt * A0 * /A1;

A2 =            /Clear * /Cnt * A2 +
                /Clear * /A0 * A2 +
                /Clear * /A1 * A2 +
                /Clear * Cnt * A0 * A1 * /A2;

A3 =            /Clear * /Cnt * A3 +
                /Clear * /A0 * A3 +
                /Clear * /A1 * A3 +
                /Clear * /A2 * A3 +
                /Clear * Cnt * A0 * A1 * A2 * /A3;

A4 =            /Clear * /Cnt * A4 +
                /Clear * /A0 * A4 +
                /Clear * /A1 * A4 +
                /Clear * /A2 * A4 +
                /Clear * /A3 * A4 +
                /Clear * Cnt * A0 * A1 * A2 * A3 * /A4;

A5 =            /Clear * /Cnt * A5 +
                /Clear * /A0 * A5 +
                /Clear * /A1 * A5 +
                /Clear * /A2 * A5 +
                /Clear * /A3 * A5 +
                /Clear * /A4 * A5 +
                /Clear * Cnt * A0 * A1 * A2 * A3 * A4 * /A5;

A6 =            /Clear * /Cnt * A6 +
                /Clear * /A0 * A6 +
                /Clear * /A1 * A6 +
                /Clear * /A2 * A6 +
                /Clear * /A3 * A6 +
                /Clear * /A4 * A6 +
                /Clear * /A5 * A6 +
                /Clear * Cnt * A0 * A1 * A2 * A3 * A4 * A5 * /A6;

CO =            /Clear * Cnt * A0 * A1 * A2 * A3 * A4 * A5 * A6;


' VIDEO ACQUISITION SYSTEM - High MAR .EQN File
device 22v10

'inputs
positive Clear@2, CI@3;


'outputs
register A7@14, A8@23, A9@15, A10@22, A11@16, A12@21, A13@17, A14@20, A15@18;


A15 =           /Clear * /CI * A15 +
                /Clear * /A7 * A15 +
                /Clear * /A8 * A15 +
                /Clear * /A9 * A15 +
```

```
                /Clear * /A10 * A15 +
                /Clear * /A11 * A15 +
                /Clear * /A12 * A15 +
                /Clear * /A13 * A15 +
                /Clear * /A14 * A15 +
                /Clear * CI * A7 * A8 * A9 * A10 * A11 * A12 * A13 * A14 *
                        /A15;

A14 =           /Clear * /CI * A14 +
                /Clear * /A7 * A14 +
                /Clear * /A8 * A14 +
                /Clear * /A9 * A14 +
                /Clear * /A10 * A14 +
                /Clear * /A11 * A14 +
                /Clear * /A12 * A14 +
                /Clear * /A13 * A14 +
                /Clear * CI * A7 * A8 * A9 * A10 * A11 * A12 * A13 * /A14;

A13 =           /Clear * /CI * A13 +
                /Clear * /A7 * A13 +
                /Clear * /A8 * A13 +
                /Clear * /A9 * A13 +
                /Clear * /A10 * A13 +
                /Clear * /A11 * A13 +
                /Clear * /A12 * A13 +
                /Clear * CI * A7 * A8 * A9 * A10 * A11 * A12 * /A13;

A12 =           /Clear * /CI * A12 +
                /Clear * /A7 * A12 +
                /Clear * /A8 * A12 +
                /Clear * /A9 * A12 +
                /Clear * /A10 * A12 +
                /Clear * /A11 * A12 +
                /Clear * CI * A7 * A8 * A9 * A10 * A11 * /A12;

A11 =           /Clear * /CI * A11 +
                /Clear * /A7 * A11 +
                /Clear * /A8 * A11 +
                /Clear * /A9 * A11 +
                /Clear * /A10 * A11 +
                /Clear * CI * A7 * A8 * A9 * A10 * /A11;

A10 =           /Clear * /CI * A10 +
                /Clear * /A7 * A10 +
                /Clear * /A8 * A10 +
                /Clear * /A9 * A10 +
                /Clear * CI * A7 * A8 * A9 * /A10;

A9 =            /Clear * /CI * A9 +
                /Clear * /A7 * A9 +
                /Clear * /A8 * A9 +
                /Clear * CI * A7 * A8 * /A9;

A8 =            /Clear * /CI * A8 +
                /Clear * /A7 * A8 +
                /Clear * CI * A7 * /A8;

A7 =            /Clear * /CI * A7 +
                /Clear * CI * /A7;
```

A.4.3 Memory Address Register .PAL File

```
22v10   C D D D D C C D D D
palasgn output from source file: lowmar.eqn

Massachusetts Institute of Technology
CLK         Clear       Cnt         NC          NC          NC          NC
NC          NC          NC          NC          GND         NC          NC
A0          A2          A4          A6          CO          NC          A5
A3          A1          VCC

A0 :=   /Clear*/Cnt*A0+
        /Clear*Cnt*/A0

A1 :=   /Clear*/Cnt*A1+
        /Clear*/A0*A1+
        /Clear*Cnt*A0*/A1

A2 :=   /Clear*/Cnt*A2+
        /Clear*/A0*A2+
        /Clear*/A1*A2+
        /Clear*Cnt*A0*A1*/A2

A3 :=   /Clear*/Cnt*A3+
        /Clear*/A0*A3+
        /Clear*/A1*A3+
        /Clear*/A2*A3+
        /Clear*Cnt*A0*A1*A2*/A3

A4 :=   /Clear*/Cnt*A4+
        /Clear*/A0*A4+
        /Clear*/A1*A4+
        /Clear*/A2*A4+
        /Clear*/A3*A4+
        /Clear*Cnt*A0*A1*A2*A3*/A4

A5 :=   /Clear*/Cnt*A5+
        /Clear*/A0*A5+
        /Clear*/A1*A5+
        /Clear*/A2*A5+
        /Clear*/A3*A5+
        /Clear*/A4*A5+
        /Clear*Cnt*A0*A1*A2*A3*A4*/A5

A6 :=   /Clear*/Cnt*A6+
        /Clear*/A0*A6+
        /Clear*/A1*A6+
        /Clear*/A2*A6+
        /Clear*/A3*A6+
        /Clear*/A4*A6+
        /Clear*/A5*A6+
        /Clear*Cnt*A0*A1*A2*A3*A4*A5*/A6

CO =    /Clear*Cnt*A0*A1*A2*A3*A4*A5*A6


22v10   D D D D D C D D D D
palasgn output from source file: himar.eqn

Massachusetts Institute of Technology
CLK         Clear       CI          NC          NC          NC          NC
NC          NC          NC          NC          GND         NC          A7
A9          A11         A13         A15         NC          A14         A12
A10         A8          VCC

A7 :=   /Clear*/CI*A7+
        /Clear*CI*/A7

A8 :=   /Clear*/CI*A8+
        /Clear*/A7*A8+
        /Clear*CI*A7*/A8

A9 :=   /Clear*/CI*A9+
        /Clear*/A7*A9+
```

```
        /Clear*/A8*A9+
        /Clear*CI*A7*A8*/A9

A10 :=  /Clear*/CI*A10+
        /Clear*/A7*A10+
        /Clear*/A8*A10+
        /Clear*/A9*A10+
        /Clear*CI*A7*A8*A9*/A10

A11 :=  /Clear*/CI*A11+
        /Clear*/A7*A11+
        /Clear*/A8*A11+
        /Clear*/A9*A11+
        /Clear*/A10*A11+
        /Clear*CI*A7*A8*A9*A10*/A11

A12 :=  /Clear*/CI*A12+
        /Clear*/A7*A12+
        /Clear*/A8*A12+
        /Clear*/A9*A12+
        /Clear*/A10*A12+
        /Clear*/A11*A12+
        /Clear*CI*A7*A8*A9*A10*A11*/A12

A13 :=  /Clear*/CI*A13+
        /Clear*/A7*A13+
        /Clear*/A8*A13+
        /Clear*/A9*A13+
        /Clear*/A10*A13+
        /Clear*/A11*A13+
        /Clear*/A12*A13+
        /Clear*CI*A7*A8*A9*A10*A11*A12*/A13

A14 :=  /Clear*/CI*A14+
        /Clear*/A7*A14+
        /Clear*/A8*A14+
        /Clear*/A9*A14+
        /Clear*/A10*A14+
        /Clear*/A11*A14+
        /Clear*/A12*A14+
        /Clear*/A13*A14+
        /Clear*CI*A7*A8*A9*A10*A11*A12*A13*/A14

A15 :=  /Clear*A15*/A7+
        /Clear*A15*/A8+
        /Clear*A15*/A9+
        /Clear*A15*/A10+
        /Clear*A15*/A11+
        /Clear*A15*/A12+
        /Clear*A15*/A13+
        /Clear*A15*/A14+
        /Clear*CI*/A15*A7*A8*A9*A10*A11*A12*A13*A14+
        /Clear*/CI*A15
```

A.4.4 Memory Address Register Chip Pinout

```
'palasgn pin assignments from file lowmar.eqn

                        PAL 0 : 22v10
                        -------------
                  CLK -|1         24|- VCC
                Clear -|2         23|- A1
                  Cnt -|3         22|- A3
                   NC -|4         21|- A5
                   NC -|5         20|- NC
                   NC -|6         19|- CO
                   NC -|7         18|- A6
                   NC -|8         17|- A4
                   NC -|9         16|- A2
                   NC -|10        15|- A0
                   NC -|11        14|- NC
                  GND -|12        13|- NC
                        -------------


'palasgn pin assignments from file himar.eqn

                        PAL 0 : 22v10
                        -------------
                  CLK -|1         24|- VCC
                Clear -|2         23|- A8
                   CI -|3         22|- A10
                   NC -|4         21|- A12
                   NC -|5         20|- A14
                   NC -|6         19|- NC
                   NC -|7         18|- A15
                   NC -|8         17|- A13
                   NC -|9         16|- A11
                   NC -|10        15|- A9
                   NC -|11        14|- A7
                  GND -|12        13|- NC
                        -------------
```

A.5 Control FSM

A.5.1 Control FSM .FSM File

```
' VIDEO ACQUISITION SYSTEM
' FSMC file for Video Capture Board control FSM
begin_p
device 0:22v10;
device 1:22v10;
'outputs
negative ADOE@14, SRAMAWE@21, SRAMBWE@17, SRAMAOE@20, SRAMBOE@18;
register ADOE@14, SRAMAWE@21, SRAMBWE@17, SRAMAOE@20, SRAMBOE@18,
         MARCLEAR@19, CNTCLEAR@114, PAGECLEAR@123, MARCNT@115, CNTCNT@122,
         BACK@116;
'inputs
positive CAPIMAGE@6, SOF@5, SOD@7, PAGEEND@4, EQ128@8, CACK@3, RESET@2,
         RESET@102, PAGEEND@107, SOF@108, PAGECNT@121, EQ128@109, CACK@110,
         SOD@113;


end_p

' Return to IDLE whenever RESET is asserted
assume {
        if RESET then
                goto Idle;
        }

Idle:  if CAPIMAGE then {
                MARCLEAR;
                CNTCLEAR;
                ADOE;
                PAGECLEAR;
                goto Sofwait;
        } else stay;

Sofwait:  if SOF then {
                MARCLEAR;
                CNTCLEAR;
                ADOE;
                PAGECLEAR;
                goto Sodwait;
          } else stay;

Sodwait:  if PAGEEND then {
                MARCLEAR;
                CNTCLEAR;
                PAGECLEAR;
                SRAMAOE;
                goto Transstart;
          } else if (SOD * /PAGEEND) then {
                        SRAMAWE;
                        ADOE;
                        goto Sramawrite;
                 } else {
                                ADOE;
                                stay;
                        }

Sramawrite:  SRAMBWE;
             CNTCNT;
             MARCNT;
             ADOE;
             goto Srambwrite;

Srambwrite:  if EQ128 then {
               CNTCLEAR;
               ADOE;
               PAGECNT;
               goto Sodwait;
             } else {
                        SRAMAWE;
```

```
                    ADOE;
                    goto Sramawrite;
                }

Transstart:
                BACK;
                SRAMAOE;
                goto Tran1;


Tran1:  if CACK then {
                SRAMBOE;
                goto Tran2;
        } else {
                BACK;
                SRAMAOE;
                stay;
                }

Tran2:  if /CACK then {
                SRAMBOE;
                goto Tran3;
        } else {
                SRAMBOE;
                stay;
                }

Tran3:  BACK;
        SRAMBOE;
        goto Tran4;

Tran4:  if CACK then {
                goto Tran5;
        } else {
                BACK;
                SRAMBOE;
                stay;
                }

Tran5:  if /CACK then {
                CNTCNT;
                MARCNT;
                goto Tran6;
        } else {
                stay;
                }

Tran6:  if EQ128 then {
                CNTCLEAR;
                PAGECNT;
                goto Tran7;
        } else {
                SRAMAOE;
                goto Transstart;
                }

Tran7:  if PAGEEND then {
                goto Idle;
        } else {
                SRAMAOE;
                goto Transstart;
        }

' Have trap states return to IDLE
State13:  goto Idle;

State14:  goto Idle;

State15:  goto Idle;
```

A.5.2 Control FSM .EQN File

```
' VIDEO ACQUISITION SYSTEM - Control FSM .EQN File

device 0:22v10;
device 1:22v10;
'outputs
negative ADOE@14, SRAMAWE@21, SRAMBWE@17, SRAMAOE@20, SRAMBOE@18;
register ADOE@14, SRAMAWE@21, SRAMBWE@17, SRAMAOE@20, SRAMBOE@18,
        MARCLEAR@19, CNTCLEAR@114, PAGECLEAR@123, MARCNT@115, CNTCNT@122,
        BACK@116;
'inputs
positive CAPIMAGE@6, SOF@5, SOD@7, PAGEEND@4, EQ128@8, CACK@3, RESET@2,
        RESET@102, PAGEEND@107, SOF@108, PAGECNT@121, EQ128@109, CACK@110,
        SOD@113;


register Q0;
register Q1;
register Q2;
register Q3;


Q0 =            /RESET * /EQ128 * Q0 * Q1 * /Q2 * Q3 +
                /RESET * /CACK * /Q0 * /Q2 * Q3 +
                /RESET * /CACK * /Q1 * /Q2 * Q3 +
                /RESET * CAPIMAGE * /Q0 * /Q1 * /Q2 +
                /RESET * /Q0 * /Q1 * /Q2 * Q3 +
                /RESET * CACK * Q1 * Q2 * /Q3 +
                /RESET * /EQ128 * /Q0 * /Q1 * Q2 * /Q3 +
                /RESET * PAGEEND * /Q0 * Q1 * /Q2 * /Q3 +
                /RESET * SOD * /Q0 * Q1 * /Q2 * /Q3 +
                /RESET * /SOF * Q0 * /Q1 * /Q2 * /Q3 +
                /RESET * /PAGEEND * /Q0 * /Q1 * Q3;

Q1 =            /RESET * CACK * Q0 * /Q1 * /Q2 * Q3 +
                /RESET * SOF * Q0 * /Q1 * /Q3 +
                /RESET * /PAGEEND * /Q0 * Q1 * /Q3 +
                /RESET * CACK * Q2 * /Q3 +
                /RESET * /Q0 * Q2 * /Q3 +
                /RESET * /Q1 * Q2 * /Q3 +
                /RESET * /Q0 * Q1 * /Q2 * Q3;

Q2 =            /RESET * Q0 * Q1 * /Q2 +
                /RESET * CACK * Q1 * Q2 * /Q3 +
                /RESET * /Q0 * Q1 * Q2 * /Q3 +
                /RESET * Q0 * /Q1 * Q2 * /Q3 +
                /RESET * PAGEEND * Q1 * /Q2 * /Q3 +
                /RESET * /PAGEEND * /Q0 * /Q1 * Q2 * Q3;

Q3 =            /RESET * /Q0 * /Q2 * Q3 +
                /RESET * /Q1 * /Q2 * Q3 +
                /RESET * /CACK * Q0 * Q1 * Q2 * /Q3 +
                /RESET * EQ128 * /Q2 * Q3;

MARCLEAR =      /RESET * SOF * Q0 * /Q1 * /Q2 * /Q3 +
                /RESET * CAPIMAGE * /Q0 * /Q1 * /Q2 * /Q3 +
                /RESET * PAGEEND * /Q0 * Q1 * /Q2 * /Q3;

CNTCLEAR =      /RESET * EQ128 * /Q0 * /Q1 * Q2 * /Q3 +
                /RESET * PAGEEND * /Q0 * Q1 * /Q2 * /Q3 +
                /RESET * SOF * Q0 * /Q1 * /Q2 * /Q3 +
                /RESET * CAPIMAGE * /Q0 * /Q1 * /Q2 * /Q3 +
                /RESET * EQ128 * Q0 * Q1 * /Q2 * Q3;

ADOE =          /RESET * /Q0 * /Q1 * Q2 * /Q3 +
                /RESET * SOF * Q0 * /Q2 * /Q3 +
                /RESET * /PAGEEND * Q1 * /Q2 * /Q3 +
                /RESET * Q0 * Q1 * /Q2 * /Q3 +
                /RESET * CAPIMAGE * /Q0 * /Q1 * /Q3;

PAGECLEAR =     /RESET * SOF * Q0 * /Q1 * /Q2 * /Q3 +
                /RESET * CAPIMAGE * /Q0 * /Q1 * /Q2 * /Q3 +
```

```
                /RESET * PAGEEND * /Q0 * Q1 * /Q2 * /Q3;

SRAMAOE =       /RESET * /EQ128 * Q0 * Q1 * /Q2 * Q3 +
                /RESET * /CACK * /Q0 * Q1 * Q2 * /Q3 +
                /RESET * Q0 * /Q1 * Q2 * /Q3 +
                /RESET * PAGEEND * /Q0 * Q1 * /Q2 * /Q3 +
                /RESET * /PAGEEND * /Q0 * /Q1 * Q2 * Q3;

SRAMAWE =       /RESET * /PAGEEND * SOD * /Q0 * Q1 * /Q2 * /Q3 +
                /RESET * /EQ128 * /Q0 * /Q1 * Q2 * /Q3;

SRAMBWE =       /RESET * Q0 * Q1 * /Q2 * /Q3;

CNTCNT =        /RESET * Q0 * Q1 * /Q2 * /Q3 +
                /RESET * /CACK * /Q0 * Q1 * /Q2 * Q3;

MARCNT =        /RESET * Q0 * Q1 * /Q2 * /Q3 +
                /RESET * /CACK * /Q0 * Q1 * /Q2 * Q3;

PAGECNT =       /RESET * EQ128 * /Q0 * /Q1 * Q2 * /Q3 +
                /RESET * EQ128 * Q0 * Q1 * /Q2 * Q3;

BACK =          /RESET * /Q0 * /Q1 * /Q2 * Q3 +
                /RESET * /CACK * /Q0 * Q1 * Q2 * /Q3 +
                /RESET * Q0 * /Q1 * Q2 * /Q3 +
                /RESET * /CACK * /Q1 * /Q2 * Q3;

SRAMBOE =       /RESET * /Q0 * /Q1 * /Q2 * Q3 +
                /RESET * CACK * Q1 * Q2 * /Q3 +
                /RESET * Q0 * Q1 * Q2 * /Q3 +
                /RESET * /CACK * /Q1 * /Q2 * Q3;
```

A.5.3 Control FSM .PAL File

```
22v10   /D D D /D /D D /D /D D D
palasgn output from source file: control.eqn

Massachusetts Institute of Technology
CLK       RESET     CACK      PAGEEND   SOF       CAPIMAGE  SOD
EQ128     NC        NC        NC        GND       NC        /ADOE
Q2        Q0        /SRAMBWE  /SRAMBOE  MARCLEAR  /SRAMAOE  /SRAMAWE
Q3        Q1        VCC

ADOE :=         /RESET*Q1*/Q2*/Q3*/PAGEEND+
        /RESET*/Q0*/Q1*Q2*/Q3+
        /RESET*/Q0*/Q1*/Q3*CAPIMAGE+
        /RESET*Q0*/Q2*/Q3*SOF+
        /RESET*Q0*Q1*/Q2*/Q3

SRAMAWE :=      /RESET*/Q0*Q1*/Q2*/Q3*/PAGEEND*SOD+
        /RESET*/EQ128*/Q0*/Q1*Q2*/Q3

SRAMBWE :=      /RESET*Q0*Q1*/Q2*/Q3

SRAMAOE :=      /RESET*/Q0*/Q1*Q2*Q3*/PAGEEND+
        /RESET*/EQ128*Q0*Q1*/Q2*Q3+
        /RESET*/Q0*Q1*Q2*/Q3*/CACK+
        /RESET*/Q0*Q1*/Q2*/Q3*PAGEEND+
        /RESET*Q0*/Q1*Q2*/Q3

SRAMBOE :=      /RESET*/Q0*/Q1*/Q2*Q3+
        /RESET*/Q1*/Q2*Q3*/CACK+
        /RESET*Q1*Q2*/Q3*CACK+
        /RESET*Q0*Q1*Q2*/Q3

MARCLEAR :=     /RESET*/Q0*Q1*/Q2*/Q3*PAGEEND+
        /RESET*Q0*/Q1*/Q2*/Q3*SOF+
        /RESET*/Q0*/Q1*/Q2*/Q3*CAPIMAGE

Q0 :=   /RESET*/Q0*/Q1*Q3*/PAGEEND+
        /RESET*/Q0*/Q1*/Q2*Q3+
        /RESET*/EQ128*Q0*Q1*/Q2*Q3+
        /RESET*/Q0*/Q2*Q3*/CACK+
        /RESET*/Q1*/Q2*Q3*/CACK+
        /RESET*Q1*Q2*/Q3*CACK+
        /RESET*/Q0*/Q1*/Q2*CAPIMAGE+
        /RESET*/Q0*Q1*/Q2*/Q3*PAGEEND+
        /RESET*/Q0*Q1*/Q2*/Q3*SOD+
        /RESET*Q0*/Q1*/Q2*/Q3*/SOF+
        /RESET*/EQ128*/Q0*/Q1*Q2*/Q3

Q1 :=   /RESET*/Q0*Q1*/Q3*/PAGEEND+
        /RESET*/Q0*Q2*/Q3+
        /RESET*/Q1*Q2*/Q3+
        /RESET*Q2*/Q3*CACK+
        /RESET*Q0*/Q1*/Q2*Q3*CACK+
        /RESET*Q0*/Q1*/Q3*SOF+
        /RESET*/Q0*Q1*/Q2*Q3

Q2 :=   /RESET*Q0*/Q1*Q2*/Q3+
        /RESET*/Q0*/Q1*Q2*Q3*/PAGEEND+
        /RESET*Q0*Q1*/Q2+
        /RESET*Q1*/Q2*/Q3*PAGEEND+
        /RESET*Q1*Q2*/Q3*CACK+
        /RESET*/Q0*Q1*Q2*/Q3

Q3 :=   /RESET*/Q0*/Q2*Q3+
        /RESET*/Q1*/Q2*Q3+
        /RESET*Q0*Q1*Q2*/Q3*/CACK+
        /RESET*EQ128*/Q2*Q3


22v10   D D D C C C C C D D
palasgn output from source file: control.eqn

Massachusetts Institute of Technology
```

```
CLK       RESET     Q0        Q1        Q2        Q3        PAGEEND
SOF       EQ128     CACK      CAPIMAGE  GND       SOD       CNTCLEAR
MARCNT    BACK      NC        NC        NC        NC        PAGECNT
CNTCNT    PAGECLEAR VCC

CNTCLEAR :=    /RESET*EQ128*/Q0*/Q1*Q2*/Q3+
       /RESET*/Q0*/Q1*/Q2*/Q3*CAPIMAGE+
       /RESET*/Q0*Q1*/Q2*/Q3*PAGEEND+
       /RESET*Q0*/Q1*/Q2*/Q3*SOF+
       /RESET*EQ128*Q0*Q1*/Q2*Q3

PAGECLEAR :=   /RESET*/Q0*Q1*/Q2*/Q3*PAGEEND+
       /RESET*Q0*/Q1*/Q2*/Q3*SOF+
       /RESET*/Q0*/Q1*/Q2*/Q3*CAPIMAGE

MARCNT :=      /RESET*/Q0*Q1*/Q2*Q3*/CACK+
       /RESET*Q0*Q1*/Q2*/Q3

CNTCNT :=      /RESET*/Q0*Q1*/Q2*Q3*/CACK+
       /RESET*Q0*Q1*/Q2*/Q3

BACK :=        /RESET*/Q0*/Q1*/Q2*Q3+
       /RESET*/Q1*/Q2*Q3*/CACK+
       /RESET*/Q0*Q1*Q2*/Q3*/CACK+
       /RESET*Q0*/Q1*Q2*/Q3

PAGECNT =      /RESET*EQ128*/Q0*/Q1*Q2*/Q3+
       /RESET*EQ128*Q0*Q1*/Q2*Q3
```

A.5.4 Control FSM Chip Pinout

```
'palasgn pin assignments from file control.eqn

                        PAL 0 : 22v10
                        -------------
                   CLK -|1         24|- VCC
                 RESET -|2         23|- Q1
                  CACK -|3         22|- Q3
               PAGEEND -|4         21|- /SRAMAWE
                   SOF -|5         20|- /SRAMAOE
              CAPIMAGE -|6         19|- MARCLEAR
                   SOD -|7         18|- /SRAMBOE
                EQ128 -|8         17|- /SRAMBWE
                    NC -|9         16|- Q0
                    NC -|10        15|- Q2
                    NC -|11        14|- /ADOE
                   GND -|12        13|- NC
                        -------------

'palasgn pin assignments from file control.eqn

                        PAL 1 : 22v10
                        -------------
                   CLK -|1         24|- VCC
                 RESET -|2         23|- PAGECLEAR
                    Q0 -|3         22|- CNTCNT
                    Q1 -|4         21|- PAGECNT
                    Q2 -|5         20|- NC
                    Q3 -|6         19|- NC
               PAGEEND -|7         18|- NC
                   SOF -|8         17|- NC
                EQ128 -|9         16|- BACK
                  CACK -|10        15|- MARCNT
              CAPIMAGE -|11        14|- CNTCLEAR
                   GND -|12        13|- SOD
                        -------------
```

Appendix B: "Third Eye" Application Files

B.1 THIRDEYE.H

```
/* thirdeye.h */

/* Windows controls ID handles */
#define IDM_LOAD              1
#define IDM_SAVE              2
#define IDM_STILL             3
#define IDM_LIVE              4
#define IDM_EDIT              5
#define IDM_RESET             6
#define IDM_HELP              7
#define IDM_QUIT              10
#define IDM_LPT1              11
#define IDM_LPT2              12
#define IDM_LPT3              13
#define IDLOADB_OK            100
#define IDLOADB_CANCEL        101
#define IDLOADL_FILE          102
#define IDSAVEB_OK            103
#define IDSAVEB_CANCEL        104
#define SCROLL_ID             105
#define STATIC_ID             106
#define RADIO1                107
#define RADIO2                108
#define RADIO3                109
#define STATIC1_ID            110
#define RADIO4                111
#define RADIO5                112

/* Function declarations */
long FAR PASCAL WndProc(HWND, WORD, WORD, LONG);
long FAR PASCAL LoadPopupProc(HWND, WORD, WORD, LONG);
long FAR PASCAL SavePopupProc(HWND, WORD, WORD, LONG);
int DisplayImage(HWND);
```

B.2 THIRDEYE.C

```
/* thirdeye.c */

#include <windows.h>

#include "thirdeye.h"
#include "vio.h"

/* global variables */
HANDLE hgImageBits[3];
LONG     lgPortad;
DWORD   hdwStartTime;
BOOL    hbLive, hbMedian;

int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
        LPSTR lpszCmdLine, int nCmdShow)
{
        HWND                    hWnd;
        MSG                     msg;
        WNDCLASS                wndclass;

        if (!hPrevInstance)
        {
                /* define main window class */
                wndclass.style                  = CS_HREDRAW | CS_VREDRAW;
                wndclass.lpfnWndProc            = WndProc;
                wndclass.cbClsExtra             = 0;
                wndclass.cbWndExtra             = 0;
                wndclass.hInstance              = hInstance;
                wndclass.hIcon                  = LoadIcon(hInstance, "ThirdEyeIcon");
                wndclass.hCursor                = LoadCursor(NULL, IDC_ARROW);
                wndclass.hbrBackground          = GetStockObject(WHITE_BRUSH);
                wndclass.lpszMenuName           = "ThirdEyeMenu";
                wndclass.lpszClassName          = "ThirdEyeClass";

                if (!RegisterClass(&wndclass))
                        return 0;
        }

        hWnd = CreateWindow("ThirdEyeClass", "ThirdEye", WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT, CW_USEDEFAULT, 450, 320, NULL,
                NULL, hInstance, NULL);
        ShowWindow(hWnd, nCmdShow);

        /* initialize global variables */
        hbLive = FALSE;
        hbMedian = FALSE;

        /* main program loop */
        while(TRUE)
        {
                if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
                {
                        if (msg.message == WM_QUIT)
                                return msg.wParam;
                        else
                        {
                                TranslateMessage(&msg);
                                DispatchMessage(&msg);
                        }
                } else if (hbLive)
                                {
                                        if (CaptureImage(lgPortad, hWnd))
                                                hbLive = FALSE;
                                        if (hbMedian)
                                                MedianFilter();
                                        DisplayImage(hWnd);
                                }
   }


}
```

```
long FAR PASCAL WndProc(HWND hWnd, WORD wMessage, WORD wParam, LONG lParam)
{
        WNDCLASS                                wndclass;
        PAINTSTRUCT                             ps;
        HDC                                     hDC;
        int                                     nLoop;
        HANDLE                                  hInstance;
        static HWND                             hLoadPopup = NULL;
        static HWND                             hSavePopup = NULL;
        static HWND                             hGroup1, hRad1, hRad2, hRad3,
                                                hStatic, hStatic1;
        static HWND                             hStatic2, hGroup2, hRad4, hRad5;
        HMENU                                   hMenu;
        unsigned char FAR                       *lpImageBits0, *lpImageBits1,
                                                *lpImageBits2;
        unsigned LONG                           lLoop;
        char                                    cBuf[256], cFPS[256];

        switch(wMessage)
        {
                case WM_CREATE:

                        /* set default port address */
                        lgPortad = 0x378;

                        /* allocate memory for image buffer */
                        hgImageBits[0] = GlobalAlloc(GMEM_MOVEABLE, 61440);
                        hgImageBits[1] = GlobalAlloc(GMEM_MOVEABLE, 61440);
                        hgImageBits[2] = GlobalAlloc(GMEM_MOVEABLE, 61440);

                        lpImageBits0 = GlobalLock(hgImageBits[0]);
                        lpImageBits1 = GlobalLock(hgImageBits[1]);
                        lpImageBits2 = GlobalLock(hgImageBits[2]);

                        /* initialize image buffer */
                        for (lLoop=0; lLoop<=61439; lLoop++) {
                                *(lpImageBits0+lLoop) = (unsigned LONG) 0;
                                *(lpImageBits1+lLoop) = (unsigned LONG) 0;
                                *(lpImageBits2+lLoop) = (unsigned LONG) 0;
                        }

                        GlobalUnlock(hgImageBits[0]);
                        GlobalUnlock(hgImageBits[1]);
                        GlobalUnlock(hgImageBits[2]);

                        /* reset board */
                        WritePort(lgPortad, 0x00);

                        /* create window controls */
                        hInstance = GetWindowWord(hWnd, GWW_HINSTANCE);
                        hGroup1 = CreateWindow("BUTTON", "Options", WS_CHILD | BS_GROUPBOX,
                                        274, 2, 155, 102, hWnd, NULL, Instance, NULL);
                        ShowWindow(hGroup1, SW_SHOWNORMAL);
                        hRad1 = CreateWindow("BUTTON", "&Stop", WS_CHILD | BS_RADIOBUTTON,
                                        285, 25, 80, 20, hWnd, RADIO1, hInstance, NULL);
                        ShowWindow(hRad1, SW_SHOWNORMAL);
                        hRad2 = CreateWindow("BUTTON", "&Live Video", WS_CHILD |
                                        BS_RADIOBUTTON, 285, 50, 90, 20, hWnd, RADIO2,
                                        hInstance, NULL);
                        ShowWindow(hRad2, SW_SHOWNORMAL);
                        hRad3 = CreateWindow("BUTTON", "S&tore Live", WS_CHILD |
                                        BS_RADIOBUTTON, 285, 75, 90, 20, hWnd, RADIO3,
                                        hInstance, NULL);
                        ShowWindow(hRad3, SW_SHOWNORMAL);
                        SendMessage(hRad1, BM_SETCHECK, TRUE, 0L);
                        hStatic = CreateWindow("STATIC", "", WS_CHILD | SS_CENTER, 3, 255,
                                        50, 15, hWnd, STATIC_ID, hInstance, NULL);
                        hGroup2 = CreateWindow("BUTTON", "Median Filter", WS_CHILD |
                                        BS_GROUPBOX, 274, 110, 155, 80, hWnd, NULL,
                                        hInstance, NULL);
                        ShowWindow(hGroup2, SW_SHOWNORMAL);
                        hRad4 = CreateWindow("BUTTON", "&On", WS_CHILD | BS_RADIOBUTTON,
                                        285, 158, 90,20, hWnd, RADIO4, hInstance, NULL);
                        ShowWindow(hRad4, SW_SHOWNORMAL);
```

```
        hRad5 = CreateWindow("BUTTON", "O&ff", WS_CHILD | BS_RADIOBUTTON,
                        285, 133, 90,20, hWnd, RADIO5, hInstance, NULL);
                        ShowWindow(hRad5, SW_SHOWNORMAL);

        wsprintf(cFPS, "%2d fps", 0);
        SetWindowText(hStatic, cFPS);
        ShowWindow(hStatic, SW_SHOWNORMAL);
        SendMessage(hRad1, BM_SETCHECK, TRUE, 0L);
        SendMessage(hRad2, BM_SETCHECK, FALSE, 0L);
        SendMessage(hRad3, BM_SETCHECK, FALSE, 0L);
        hbLive = FALSE;
        SendMessage(hRad4, BM_SETCHECK, FALSE, 0L);
        SendMessage(hRad5, BM_SETCHECK, TRUE, 0L);
        hbMedian = FALSE;

        break;

case WM_PAINT:
        /* if screen needs updating, redraw image */
        BeginPaint(hWnd, &ps);
        DisplayImage(hWnd);
        EndPaint(hWnd, &ps);

        break;

case WM_COMMAND:
        switch(wParam)
        {
                case RADIO1:
                        /* radio1 has been pressed */
                        SendMessage(hRad1, BM_SETCHECK, TRUE, 0L);
                        SendMessage(hRad2, BM_SETCHECK, FALSE, 0L);
                        SendMessage(hRad3, BM_SETCHECK, FALSE, 0L);
                        hbLive = FALSE;
                        break;

                case RADIO2:
                        /* radio2 has been pressed */
                        SendMessage(hRad1, BM_SETCHECK, FALSE, 0L);
                        SendMessage(hRad2, BM_SETCHECK, TRUE, 0L);
                        SendMessage(hRad3, BM_SETCHECK, FALSE, 0L);
                        hbLive = TRUE;
                        break;

                case RADIO3:
                        /* radio3 has been pressed */
                        SendMessage(hRad1, BM_SETCHECK, FALSE, 0L);
                        SendMessage(hRad2, BM_SETCHECK, FALSE, 0L);
                        SendMessage(hRad3, BM_SETCHECK, TRUE, 0L);
                        hbLive = FALSE;
                        break;

                case RADIO4:
                        /* radio4 has been pressed */
                        SendMessage(hRad4, BM_SETCHECK, TRUE, 0L);
                        SendMessage(hRad5, BM_SETCHECK, FALSE, 0L);
                        hbMedian = TRUE;
                        break;

                case RADIO5:
                        /* radio5 has been pressed */
                        SendMessage(hRad4, BM_SETCHECK, FALSE, 0L);
                        SendMessage(hRad5, BM_SETCHECK, TRUE, 0L);
                        hbMedian = FALSE;
                        break;

                case IDM_LPT1:
                        /* set port to LPT1 */
                        lgPortad = 0x3BC;
                        hMenu = GetMenu(hWnd);
                        CheckMenuItem(hMenu, IDM_LPT1,
                                MF_BYCOMMAND|MF_CHECKED);
                        CheckMenuItem(hMenu, IDM_LPT2,
                                MF_BYCOMMAND|MF_UNCHECKED);
```

```
            CheckMenuItem(hMenu, IDM_LPT3,
                    MF_BYCOMMAND|MF_UNCHECKED);
            break;

    case IDM_LPT2:
            /* set port to LPT2 */
            lgPortad = 0x378;
            hMenu = GetMenu(hWnd);
            CheckMenuItem(hMenu, IDM_LPT1,
                    MF_BYCOMMAND|MF_UNCHECKED);
            CheckMenuItem(hMenu, IDM_LPT2,
                    MF_BYCOMMAND|MF_CHECKED);
            CheckMenuItem(hMenu, IDM_LPT3,
                    MF_BYCOMMAND|MF_UNCHECKED);
            break;

    case IDM_LPT3:
            /* set port to LPT3 */
            lgPortad = 0x278;
            hMenu = GetMenu(hWnd);
            CheckMenuItem(hMenu, IDM_LPT1,
                    MF_BYCOMMAND|MF_UNCHECKED);
            CheckMenuItem(hMenu, IDM_LPT2,
                    MF_BYCOMMAND|MF_UNCHECKED);
            CheckMenuItem(hMenu, IDM_LPT3,
                    MF_BYCOMMAND|MF_CHECKED);
    break;

    case IDM_LOAD:
            if (hLoadPopup == NULL) {
                    /* initialize load window */
                    hInstance = GetWindowWord(hWnd,
                                    GWW_HINSTANCE);

                    wndclass.style = CS_HREDRAW | CS_VREDRAW |
                                            CS_PARENTDC;
                    wndclass.lpfnWndProc = LoadPopupProc;
                    wndclass.cbClsExtra = 0;
                    wndclass.cbWndExtra = 0;
                    wndclass.hInstance = hInstance;
                    wndclass.hIcon = NULL;
                    wndclass.hCursor = LoadCursor(NULL,
                                            IDC_ARROW);
                    wndclass.hbrBackground =
                            GetStockObject(LTGRAY_BRUSH);
                    wndclass.lpszMenuName = NULL;
                    wndclass.lpszClassName = "LoadPopupClass";

                    if (RegisterClass(&wndclass)) {
                            hLoadPopup =
                                    CreateWindow("LoadPopupClass",
                                    "Load Image",
                                    WS_POPUP | WS_VISIBLE |
                                    WS_BORDER | WS_CAPTION, 18,
                                    29, 300, 250, hWnd, NULL,
                                    hInstance, NULL);
                    }
            }
            ShowWindow(hLoadPopup, SW_SHOW);

    break;

    case IDM_SAVE:
            if (hSavePopup == NULL) {
                    /* initialize save window */
                    hInstance = GetWindowWord(hWnd,
                            GWW_HINSTANCE);

                    wndclass.style = CS_HREDRAW | CS_VREDRAW |
                            CS_PARENTDC;
                    wndclass.lpfnWndProc = SavePopupProc;
                    wndclass.cbClsExtra = 0;
                    wndclass.cbWndExtra = 0;
                    wndclass.hInstance = hInstance;
                    wndclass.hIcon = NULL;
```

```
                                        wndclass.hCursor = LoadCursor(NULL,
                                                IDC_ARROW);
                                        wndclass.hbrBackground =
                                                GetStockObject(LTGRAY_BRUSH);
                                        wndclass.lpszMenuName = NULL;
                                        wndclass.lpszClassName = "SavePopupClass";

                                        if (RegisterClass(&wndclass)) {
                                                hSavePopup =
                                                        CreateWindow("SavePopupClass",
                                                        "Save Image", WS_POPUP |
                                                        WS_VISIBLE | WS_BORDER |
                                                        WS_CAPTION, 18, 29, 300, 250,
                                                        hWnd, NULL, hInstance, NULL);
                                                }
                                        }
                                        ShowWindow(hSavePopup, SW_SHOW);

                                break;


                                case IDM_STILL:
                                        /* capture still image and display */
                                        CaptureImage(lgPortad, hWnd);
                                        if (hbMedian)
                                                MedianFilter();
                                        DisplayImage(hWnd);
                                        hbLive = FALSE;
                                        /* update radio check buttons */
                                        SendMessage(hRad1, BM_SETCHECK, TRUE, 0L);
                                        SendMessage(hRad2, BM_SETCHECK, FALSE, 0L);
                                        SendMessage(hRad3, BM_SETCHECK, FALSE, 0L);
                                        break;

                                case IDM_LIVE:
                                        /* update radio check buttons */
                                        SendMessage(hRad1, BM_SETCHECK, FALSE, 0L);
                                        SendMessage(hRad2, BM_SETCHECK, TRUE, 0L);
                                        SendMessage(hRad3, BM_SETCHECK, FALSE, 0L);
                                        hbLive = TRUE;
                                        break;

                                case IDM_RESET:
                                        /* reset board */
                                        ResetDevice(lgPortad);
                                        MessageBox(hWnd, (LPSTR) "Video Capture Board
                                                Reset", "Device Reset", MB_OK);
                                        break;

                                case IDM_QUIT:
                                        /* quit program */
                                        DestroyWindow(hWnd);
                                        break;
                        }
                        break;

                case WM_DESTROY:
                        /* deallocate memory */
                        GlobalFree(hgImageBits[0]);
                        GlobalFree(hgImageBits[1]);
                        GlobalFree(hgImageBits[2]);
                        PostQuitMessage(0);
                        break;

                default:
                        return DefWindowProc(hWnd, wMessage, wParam, lParam);
        }
        return(0L);
}

long FAR PASCAL SavePopupProc(HWND hSavePopup, WORD wMessage, WORD wParam,
                                                                        LONG lParam)
{
        HANDLE                  hInstance, hgImageTemp;
        static HANDLE           hEditBuf;
```

```
static BOOL             bFirstTime = TRUE;
static HWND             hSaveEdit, hSaveText;
char                    cBuf[14];
int                     nFileHandle, nLoop;
OFSTRUCT                of;
HDC                     hDC;
LPSTR                   lpStr;
BITMAPFILEHEADER        hFileHeader, *pFileHeader;
BITMAPINFOHEADER        hInfoHeader, *pInfoHeader;
unsigned char FAR       *lpImageBits0, *lpImageBits1, *lpImageBits2, *lpImageTemp;
unsigned LONG           lLoop, lCount;
unsigned char           nVal, nVal1;

switch (wMessage)
{
        case WM_CREATE:
                if (bFirstTime) {
                        /* initialize save popup window */
                        bFirstTime = FALSE;
                        hInstance = GetWindowWord(hSavePopup, GWW_HINSTANCE);
                        hSaveText = CreateWindow("STATIC", "Filename:", WS_CHILD |
                                        SS_CENTER, 49,80,70,18, hSavePopup, NULL,
                                        hInstance, NULL);
                        ShowWindow(hSaveText, SW_SHOW);
                        hSaveEdit = CreateWindow("EDIT", "", WS_CHILD | WS_VISIBLE |
                                        WS_BORDER | ES_MULTILINE, 125, 78, 110, 24,
                                        hSavePopup, NULL, hInstance, NULL);

                        hEditBuf = (HANDLE) SendMessage(hSaveEdit, EM_GETHANDLE, 0,
                                        0L);
                        if (hEditBuf)
                                LocalFree(hEditBuf);
                        hEditBuf = LocalAlloc(LMEM_MOVEABLE | LMEM_ZEROINIT, 14);
                        PostMessage(hSaveEdit, EM_SETHANDLE, hEditBuf, 0L);

                        CreateWindow("BUTTON", "OK", WS_CHILD | BS_PUSHBUTTON |
                                WS_VISIBLE, 49, 180, 71, 18, hSavePopup, IDSAVEB_OK,
                                hInstance, NULL);
                        CreateWindow("BUTTON", "CANCEL", WS_CHILD | BS_PUSHBUTTON |
                                WS_VISIBLE, 180, 180, 71, 18, hSavePopup,
                                IDSAVEB_CANCEL, hInstance, NULL);
                }
        break;

        case WM_COMMAND:
                        switch(wParam)
                        {
                                case IDSAVEB_OK:

                                        lpStr = LocalLock(hEditBuf);
                                        nFileHandle = OpenFile((DWORD)(LPSTR) lpStr,
                                        &of, OF_CREATE);

                                        if (nFileHandle==-1) {
                                                LocalUnlock((HANDLE) lpStr);
                                                MessageBox(hSavePopup, (LPSTR) "Save
                                                Image Unsuccessful", "Load Image",
                                                MB_OK);
                                                _lclose(nFileHandle);
                                                ShowWindow(hSavePopup, SW_HIDE);
                                                break;
                                        }

                                        /* write BMP file to disk */
                                        hFileHeader.bfType = (WORD) 'BM';
                                        hFileHeader.bfSize = (DWORD) 0x0002D036;
                                        hFileHeader.bfReserved1 = (WORD) 0;
                                        hFileHeader.bfReserved2 = (WORD) 0;
                                        hFileHeader.bfOffBits = (DWORD) 0x00000036;

                                        hInfoHeader.biSize = (DWORD) 0x28;
                                        hInfoHeader.biWidth = (DWORD) 0x0100;
                                        hInfoHeader.biHeight = (DWORD) 0xF0;
                                        hInfoHeader.biPlanes = (WORD) 1;
                                        hInfoHeader.biBitCount = (WORD) 24;
```

```
hInfoHeader.biCompression = (DWORD) BI_RGB;
hInfoHeader.biSizeImage = (DWORD)
        (256*240*3);
hInfoHeader.biXPelsPerMeter = (DWORD) 0;
hInfoHeader.biYPelsPerMeter = (DWORD) 0;
hInfoHeader.biClrUsed = (DWORD) 0;
hInfoHeader.biClrImportant = (DWORD) 0;

pFileHeader = &hFileHeader;
pInfoHeader = &hInfoHeader;

_lwrite(nFileHandle, pFileHeader,
        sizeof(BITMAPFILEHEADER));
_lwrite(nFileHandle, pInfoHeader,
        sizeof(BITMAPINFOHEADER));

lpImageBits0 = GlobalLock(hgImageBits[0]);
lpImageBits1 = GlobalLock(hgImageBits[1]);
lpImageBits2 = GlobalLock(hgImageBits[2]);

hgImageTemp = GlobalAlloc(GMEM_MOVEABLE,
        61440);
lpImageTemp = GlobalLock(hgImageTemp);

/* reverse picture vertically
   for BMP format */
for (lLoop=0; lLoop <= 61437; lLoop=lLoop+3)
{
        nVal = *(lpImageBits2+lLoop);
        *(lpImageTemp+(61437 - lLoop)) =
                nVal;
        nVal = *(lpImageBits2+lLoop+1);
        *(lpImageTemp+(61438 - lLoop)) =
                nVal;
        nVal = *(lpImageBits2+lLoop+2);
        *(lpImageTemp+(61439 - lLoop)) =
                nVal;
}

for (lLoop=0; lLoop<=60672; lLoop=lLoop+768)
{
        for (lCount=0; lCount<=381;
                lCount=lCount+3)
        {

        nVal=*(lpImageTemp+lLoop+lCount);
        *(lpImageTemp+lLoop+lCount) =
        *(lpImageTemp+lLoop+(765-lCount));
        *(lpImageTemp+lLoop+(765-lCount)) =
                nVal;

        nVal=*(lpImageTemp+lLoop+lCount+1);
        *(lpImageTemp+lLoop+lCount+1) =
        *(lpImageTemp+lLoop+(766-lCount));
        *(lpImageTemp+lLoop+(766-lCount)) =
                nVal;


        nVal=*(lpImageTemp+lLoop+lCount+2);
        *(lpImageTemp+lLoop+lCount+2) =
        *(lpImageTemp+lLoop+(767-lCount));
        *(lpImageTemp+lLoop+(767-lCount)) =
                nVal;
        }
}
_lwrite(nFileHandle, lpImageTemp,
        (256*80*3));

for (lLoop=0; lLoop <= 61437; lLoop=lLoop+3)
{
        nVal = *(lpImageBits1+lLoop);
        *(lpImageTemp+(61437 - lLoop)) =
                nVal;
        nVal = *(lpImageBits1+lLoop+1);
```

```
        *(lpImageTemp+(61438 - lLoop)) =
                nVal;
        nVal = *(lpImageBits1+lLoop+2);
        *(lpImageTemp+(61439 - lLoop)) =
                nVal;
}

for (lLoop=0; lLoop<=60672; lLoop=lLoop+768)
{
        for (lCount=0; lCount<=381;
                lCount=lCount+3)
        {
        nVal=*(lpImageTemp+lLoop+lCount);
        *(lpImageTemp+lLoop+lCount) =
        *(lpImageTemp+lLoop+(765-lCount));
        *(lpImageTemp+lLoop+(765-lCount)) =
                nVal;

        nVal=*(lpImageTemp+lLoop+lCount+1);
        *(lpImageTemp+lLoop+lCount+1) =
        *(lpImageTemp+lLoop+(766-lCount));
        *(lpImageTemp+lLoop+(766-lCount)) =
                nVal;

        nVal=*(lpImageTemp+lLoop+lCount+2);
        *(lpImageTemp+lLoop+lCount+2) =
        *(lpImageTemp+lLoop+(767-lCount));
        *(lpImageTemp+lLoop+(767-lCount)) =
                nVal;
        }
}
_lwrite(nFileHandle, lpImageTemp,
        (256*80*3));

for (lLoop=0; lLoop <= 61437; lLoop=lLoop+3)
{
        nVal = *(lpImageBits0+lLoop);
        *(lpImageTemp+(61437 - lLoop)) =
                nVal;
        nVal = *(lpImageBits0+lLoop+1);
        *(lpImageTemp+(61438 - lLoop)) =
                nVal;
        nVal = *(lpImageBits0+lLoop+2);
        *(lpImageTemp+(61439 - lLoop)) =
                nVal;
}

for (lLoop=0; lLoop<=60672; lLoop=lLoop+768)
{
        for (lCount=0; lCount<=381;
                lCount=lCount+3)
        {

        nVal=*(lpImageTemp+lLoop+lCount);
        *(lpImageTemp+lLoop+lCount) =
        *(lpImageTemp+lLoop+(765-lCount));
        *(lpImageTemp+lLoop+(765-lCount)) =
                nVal;
        nVal=*(lpImageTemp+lLoop+lCount+1);
        *(lpImageTemp+lLoop+lCount+1) =
        *(lpImageTemp+lLoop+(766-lCount));
        *(lpImageTemp+lLoop+(766-lCount)) =
                nVal;

        nVal=*(lpImageTemp+lLoop+lCount+2);
        *(lpImageTemp+lLoop+lCount+2) =
        *(lpImageTemp+lLoop+(767-lCount));
        *(lpImageTemp+lLoop+(767-lCount)) =
                nVal;
        }
}
_lwrite(nFileHandle, lpImageTemp,
        (256*80*3));

LocalUnlock((HANDLE) lpStr);
```

```
                                        _lclose(nFileHandle);

                                        /* unlock memory region */
                                        GlobalUnlock(hgImageBits[0]);
                                        GlobalUnlock(hgImageBits[1]);
                                        GlobalUnlock(hgImageBits[2]);
                                        GlobalUnlock(hgImageTemp);
                                        GlobalFree(hgImageTemp);

                                        /* hide window */
                                        ShowWindow(hSavePopup, SW_HIDE);
                                break;

                                case IDSAVEB_CANCEL:
                                        ShowWindow(hSavePopup, SW_HIDE);
                                break;
                        }
                break;

                default:
                        return DefWindowProc(hSavePopup, wMessage, wParam, lParam);
        }
        return(0L);
}


long FAR PASCAL LoadPopupProc(HWND hLoadPopup, WORD wMessage, WORD wParam,
                                                                        LONG lParam)
{
        HANDLE                  hInstance, hgImageTemp;
        static BOOL             bFirstTime = TRUE;
        static HWND             hListBox;
        int                     nSel;
        char                    cBuf[13], cSelBuf[13];
        OFSTRUCT                of;
        int                     nFileHandle, nFileLong;
        PSTR                    pFileData;
        LPBITMAPINFOHEADER      pFileData2;
        unsigned char FAR       *lpImageBits0, *lpImageBits1, *lpImageBits2, *lpImageTemp;
        HWND                    hParent, hLoadText;
        unsigned LONG           lLoop, lCount;
        unsigned char           nVal, nVal1;
        DWORD                   biWidth, biHeight;


        switch (wMessage)
        {
                case WM_CREATE:
                        if (bFirstTime) {
                                /* initialize load popup window */
                                bFirstTime = FALSE;
                                hInstance = GetWindowWord(hLoadPopup, GWW_HINSTANCE);
                                hLoadText = CreateWindow("STATIC", "Filename:", WS_CHILD |
                                        SS_CENTER, 20,8,70,14, hLoadPopup, NULL, hInstance,
                                        NULL);
                                ShowWindow(hLoadText, SW_SHOW);
                                hListBox = CreateWindow("LISTBOX", "", WS_CHILD |
                                        LBS_STANDARD, 20, 25, 140, 180, hLoadPopup,
                                        IDLOADL_FILE, hInstance, NULL);
                                ShowWindow(hListBox, SW_SHOWNORMAL);
                                CreateWindow("BUTTON", "OK", WS_CHILD | BS_PUSHBUTTON |
                                        WS_VISIBLE, 200, 25, 71, 18, hLoadPopup, IDLOADB_OK,
                                        hInstance, NULL);
                                CreateWindow("BUTTON", "Cancel", WS_CHILD | BS_PUSHBUTTON |
                                        WS_VISIBLE, 200, 64, 71, 18, hLoadPopup,
                                        IDLOADB_CANCEL, hInstance, NULL);
                        }
                        SendMessage(hListBox, LB_DIR, 0x0000, (DWORD) (LPSTR) "*.BMP");

                        break;

                case WM_COMMAND:
                                switch(wParam)
                                {
```

```
case IDLOADB_OK:
        nSel = (int) SendMessage(hListBox,
                        LB_GETCURSEL, 0, 0L);
        SendMessage(hListBox, LB_GETTEXT, nSel,
                DWORD)(LPSTR)cSelBuf);
        nFileHandle = OpenFile((LPSTR)cSelBuf, &of,
                        OF_READ);
        if (nFileHandle == -1) {
                MessageBox(hLoadPopup, (LPSTR)
                "Load Image Unsuccessful",
                "Load Image", MB_OK);
                _lclose(nFileHandle);
                ShowWindow(hLoadPopup, SW_HIDE);
                break;
        }

        lpImageBits0 = GlobalLock(hgImageBits[0]);
        lpImageBits1 = GlobalLock(hgImageBits[1]);
        lpImageBits2 = GlobalLock(hgImageBits[2]);

        hgImageTemp = GlobalAlloc(GMEM_MOVEABLE,
                61440);
        lpImageTemp = GlobalLock(hgImageTemp);

        pFileData = (PSTR) LocalAlloc(LPTR,
                sizeof(BITMAPFILEHEADER));
        _lread(nFileHandle, pFileData,
                sizeof(BITMAPFILEHEADER));

        pFileData2 = (PSTR) LocalAlloc(LPTR,
                sizeof(BITMAPINFOHEADER));
        _lread(nFileHandle, pFileData2,
                sizeof(BITMAPINFOHEADER));

        biWidth = (pFileData2->biWidth);
        biHeight = (pFileData2->biHeight);
        if ((biWidth == 256) && (biHeight == 240)) {
                _lread(nFileHandle, lpImageTemp,
                        (256*80*3));

                for (lLoop=0; lLoop <= 61437;
                        lLoop=lLoop+3) {
                        nVal = *(lpImageTemp+lLoop);
                        *(lpImageBits2+(61437 -
                                lLoop)) = nVal;
                        nVal = *(lpImageTemp+lLoop+1);
                        *(lpImageBits2+(61438 -
                                lLoop)) = nVal;
                        nVal = *(lpImageTemp+lLoop+2);
                        *(lpImageBits2+(61439 -
                                lLoop)) = nVal;
                }

                for (lLoop=0; lLoop<=60672;
                        lLoop=lLoop+768) {
                for (lCount=0; lCount<=381;
                        lCount=lCount+3) {

                nVal=*(lpImageBits2+lLoop+lCount);

                *(lpImageBits2+lLoop+lCount) =
                *(lpImageBits2+lLoop+(765-lCount));

                *(lpImageBits2+lLoop+(765-lCount)) =
                        nVal;

                nVal=*(lpImageBits2+lLoop+lCount+1);

                *(lpImageBits2+lLoop+lCount+1) =
                *(lpImageBits2+lLoop+(766-lCount));

                *(lpImageBits2+lLoop+(766-lCount)) =
                        nVal;
```

```
nVal=*(lpImageBits2+lLoop+lCount+2);

*(lpImageBits2+lLoop+lCount+2) =
*(lpImageBits2+lLoop+(767-lCount));

*(lpImageBits2+lLoop+(767-lCount)) =
       nVal;
}
}

_lread(nFileHandle, lpImageTemp,
(256*80*3));

for (lLoop=0; lLoop <= 61437;
       lLoop=lLoop+3) {
       nVal = *(lpImageTemp+lLoop);
       *(lpImageBits1+(61437 -
              lLoop)) = nVal;
       nVal = *(lpImageTemp+lLoop+1);
       *(lpImageBits1+(61438 -
              lLoop)) = nVal;
       nVal = *(lpImageTemp+lLoop+2);
       *(lpImageBits1+(61439 -
              lLoop)) = nVal;
}

for (lLoop=0; lLoop<=60672;
       lLoop=lLoop+768) {
for (lCount=0; lCount<=381;
       lCount=lCount+3) {
nVal=*(lpImageBits1+lLoop+lCount);

*(lpImageBits1+lLoop+lCount) =
*(lpImageBits1+lLoop+(765-lCount));

*(lpImageBits1+lLoop+(765-lCount)) =
       nVal;

nVal=*(lpImageBits1+lLoop+lCount+1);

*(lpImageBits1+lLoop+lCount+1) =
*(lpImageBits1+lLoop+(766-lCount));

*(lpImageBits1+lLoop+(766-lCount)) =
       nVal;

nVal=*(lpImageBits1+lLoop+lCount+2);

*(lpImageBits1+lLoop+lCount+2) =
*(lpImageBits1+lLoop+(767-lCount));

*(lpImageBits1+lLoop+(767-lCount)) =
       nVal;
}
}

_lread(nFileHandle, lpImageTemp,
       (256*80*3));

for (lLoop=0; lLoop <= 61437;
       lLoop=lLoop+3) {
       nVal = *(lpImageTemp+lLoop);
       *(lpImageBits0+(61437 -
              lLoop)) = nVal;
       nVal = *(lpImageTemp+lLoop+1);
       *(lpImageBits0+(61438 -
              lLoop)) = nVal;
       nVal = *(lpImageTemp+lLoop+2);
       *(lpImageBits0+(61439 -
              lLoop)) = nVal;
}

for (lLoop=0; lLoop<=60672;
       lLoop=lLoop+768) {
```

```
                                                for (lCount=0; lCount<=381;
                                                        lCount=lCount+3) {

                                                nVal=*(lpImageBits0+lLoop+lCount);

                                                *(lpImageBits0+lLoop+lCount) =
                                                *(lpImageBits0+lLoop+(765-lCount));

                                                *(lpImageBits0+lLoop+(765-lCount)) =
                                                        nVal;

                                                nVal=*(lpImageBits0+lLoop+lCount+1);

                                                *(lpImageBits0+lLoop+lCount+1) =
                                                *(lpImageBits0+lLoop+(766-lCount));

                                                *(lpImageBits0+lLoop+(766-lCount)) =
                                                        nVal;

                                                nVal=*(lpImageBits0+lLoop+lCount+2);

                                                *(lpImageBits0+lLoop+lCount+2) =
                                                *(lpImageBits0+lLoop+(767-lCount));

                                                *(lpImageBits0+lLoop+(767-lCount)) =
                                                        nVal;
                                                }
                                                }

                                        } else {
                                                MessageBox(hLoadPopup,
                                                        (LPSTR) "Not a Valid Image",
                                                        "Load Image", MB_OK);
                                                }

                                                _lclose(nFileHandle);

                                                hParent = GetParent(hLoadPopup);
                                                DisplayImage(hParent);

                                                LocalFree((HANDLE) pFileData);
                                                LocalFree((HANDLE) pFileData2);
                                                GlobalUnlock(hgImageBits[0]);
                                                GlobalUnlock(hgImageBits[1]);
                                                GlobalUnlock(hgImageBits[2]);
                                                GlobalUnlock(hgImageTemp);
                                                GlobalFree(hgImageTemp);
                                                ShowWindow(hLoadPopup, SW_HIDE);

                                        break;

                                case IDLOADL_FILE:
                                        if (HIWORD(lParam) == LBN_DBLCLK) {
                                           SendMessage(hLoadPopup, WM_COMMAND,
                                                IDLOADB_OK, 0L);
                                        }

                                        break;
                                case IDLOADB_CANCEL:
                                        ShowWindow(hLoadPopup, SW_HIDE);
                                        break;
                                }
                        default:
                                return DefWindowProc(hLoadPopup, wMessage, wParam, lParam);
                }
                return(0L);
}

int DisplayImage(HWND hWnd)
/* This procedure retrieves an image from a global memory bitmap and copies it
    to the screen. */
{
        HDC                             hDC, hMemDC;
        HPEN                            hPen;
        unsigned char FAR               *lpImageBits0, *lpImageBits1, *lpImageBits2;
```

```
        static HBITMAP              hBitmap;
        unsigned char               bVal;
        int                         nVer, nHor;
        long                        lLoop;

        /* draw image border */
        hDC = GetDC(hWnd);
        hPen = CreatePen(PS_INSIDEFRAME, 1, RGB(0,0,0));
        SelectObject(hDC, hPen);
        MoveTo(hDC, 9, 9);
        LineTo(hDC, 266, 9);
        LineTo(hDC, 266, 250);
        LineTo(hDC, 9, 250);
        LineTo(hDC, 9, 9);
        DeleteObject(hPen);

        /* get pointers to image memory bitmap */
        lpImageBits0 = GlobalLock(hgImageBits[0]);
        lpImageBits1 = GlobalLock(hgImageBits[1]);
        lpImageBits2 = GlobalLock(hgImageBits[2]);

        hMemDC = CreateCompatibleDC(hDC);

        /* paste memory bitmap to screen */
        hBitmap = CreateBitmap(256, 80, 1, 24, lpImageBits0);
        SelectObject(hMemDC, hBitmap);
        BitBlt(hDC, 10, 10, 256, 80, hMemDC, 0, 0, SRCCOPY);

        hBitmap = CreateBitmap(256, 80, 1, 24, lpImageBits1);
        SelectObject(hMemDC, hBitmap);
        BitBlt(hDC, 10, 90, 256, 80, hMemDC, 0, 0, SRCCOPY);

        hBitmap = CreateBitmap(256, 80, 1, 24, lpImageBits2);
        SelectObject(hMemDC, hBitmap);
        BitBlt(hDC, 10, 170, 256, 80, hMemDC, 0, 0, SRCCOPY);

        DeleteDC(hMemDC);

        /* unlock memory */
        GlobalUnlock(hgImageBits[0]);
        GlobalUnlock(hgImageBits[1]);
        GlobalUnlock(hgImageBits[2]);
        ReleaseDC(hWnd, hDC);

        return(0);
}
```

B.3 THIRDEYE.DEF

```
/* THIRDEYE.DEF - Definition File */

NAME                    thirdeye
EXETYPE                 WINDOWS
STUB                    'WINSTUB.EXE'
CODE                    PRELOAD MOVEABLE
DATA                    PRELOAD MOVEABLE MULTIPLE
HEAPSIZE                1024
STACKSIZE               5120
EXPORTS                 WndProc
                        LoadPopupProc
                        SavePopupProc
```

B.4 THIRDEYE.RC

```
/* thirdeye.rc */

#include "thirdeye.h"

ThirdEyeIcon            ICON            thirdeye.ico

ThirdEyeMenu            MENU
BEGIN
        POPUP "&Image"
        BEGIN
                MENUITEM "&Load"                IDM_LOAD
                MENUITEM "&Save"                IDM_SAVE
                MENUITEM SEPARATOR
                MENUITEM "Capture s&till"       IDM_STILL
                MENUITEM "Capture &live"        IDM_LIVE
                MENUITEM SEPARATOR
                MENUITEM "&Reset board"         IDM_RESET
                MENUITEM "&Quit"        IDM_QUIT
        END
        POPUP "&Port Address"
        BEGIN
                MENUITEM "LPT1 0x3BC",          IDM_LPT1
                MENUITEM "LPT2 0x378",          IDM_LPT2, CHECKED
                MENUITEM "LPT3 0x278",          IDM_LPT3
        END

        MENUITEM "\a&Help"                       IDM_HELP
END
```

B.5 VIO.H

```
/* vio.h */

unsigned char ReadPort(int portid);
void WritePort(int portid, unsigned char value);
unsigned char GetbACK(int portid);
int CaptureImage(int portid, HWND hWnd);
void ResetDevice(int portid);
void MedianFilter(void);
```

B.6 VIO.C

```
/* vio.c */

#include <dos.h>
#include <windows.h>
#include "vio.h"

unsigned char ReadPort(int portid)
/* read a byte from the requested port */
{
        unsigned char bDil, bDih, bDib;

        bDil = inportb(portid + 2);
        bDih = inportb(portid + 1);
        bDib = ((bDil & 0x0F) + (bDih & 0xF0));

        return(bDib);
}

void WritePort(int portid, unsigned char value)
/* write a byte to the requested port */
{
        outportb(portid, value);

        return;
}

unsigned char GetbACK(int portid)
/* wait for a board acknowledge signal */
{
        unsigned char bDih;

        bDih = inportb(portid + 1);
        bDih = (bDih & 0x08);
        bDih = bDih >> 3;

        return(bDih);
}

void ResetDevice(int portid)
/* reset video capture board */
{
        WritePort(portid, 0x00);
        return;
}

void MedianFilter()
/* perform median filter on image in memory bitmap */
{
        unsigned char FAR       *lpImageBits0, *lpImageBits1, *lpImageBits2, *lpTempBits;
        unsigned long           lLoop = 0;
        int                     nVal0, nVal1, nVal2, nRed, nBlue, nGreen, nTemp, j, k;
        HANDLE                  hTemp;
        unsigned char           val[3];
        extern HANDLE           hgImageBits[3];

        /* initialize variables */
        nVal0 = 0;
        nVal1 = 0;
        nVal2 = 0;

        /* get pointers to memory bitmap */
        lpImageBits0 = GlobalLock(hgImageBits[0]);
        lpImageBits1 = GlobalLock(hgImageBits[1]);
        lpImageBits2 = GlobalLock(hgImageBits[2]);

        /* alloc temporary memory space */
        hTemp = GlobalAlloc(GMEM_MOVEABLE, 61439);
        lpTempBits = GlobalLock(hTemp);

        for (lLoop = 0; lLoop <= 61439; lLoop++) {
                *(lpTempBits+lLoop) = *(lpImageBits0+lLoop);
```

```
    }

    for (lLoop = 3; lLoop <= 61434; lLoop = lLoop + 3) {
            /* convert RGB to luminance */
            nRed = *(lpTempBits+lLoop-3);
            nGreen = *(lpTempBits+lLoop-2);
            nBlue = *(lpTempBits+lLoop-1);
            nVal0 = (int) (.299*nRed + .587*nGreen + .114*nBlue);

            nRed = *(lpTempBits+lLoop);
            nGreen = *(lpTempBits+lLoop+1);
            nBlue = *(lpTempBits+lLoop+2);
            nVal1 = (int) (.299*nRed + .587*nGreen + .114*nBlue);

            nRed = *(lpTempBits+lLoop+3);
            nGreen = *(lpTempBits+lLoop+4);
            nBlue = *(lpTempBits+lLoop+5);
            nVal2 = (int) (.299*nRed + .587*nGreen + .114*nBlue);

            val[0] = nVal0;
            val[1] = nVal1;
            val[2] = nVal2;

            /* perform bubble sort */
            for (j = 0; j <= 3; j++) {
                    for (k = 0; k <= 3; k++) {
                            if (val[k] < val[j]) {
                                    nTemp = (int) val[j];
                                    val[j] = val[k];
                                    val[k] = (unsigned char) nTemp;
                            }
                    }
            }

            if (nTemp == nVal0) {
                    nRed = *(lpTempBits+lLoop-3);
                    nGreen = *(lpTempBits+lLoop-2);
                    nBlue = *(lpTempBits+lLoop-1);
                    *(lpImageBits0+lLoop) = nRed;
                    *(lpImageBits0+lLoop+1) = nGreen;
                    *(lpImageBits0+lLoop+2) = nBlue;
            }

            if (nTemp == nVal1) {
                    nRed = *(lpTempBits+lLoop);
                    nGreen = *(lpTempBits+lLoop+1);
                    nBlue = *(lpTempBits+lLoop+2);
                    *(lpImageBits0+lLoop) = nRed;
                    *(lpImageBits0+lLoop+1) = nGreen;
                    *(lpImageBits0+lLoop+2) = nBlue;
            }

            if (nTemp == nVal2) {
                    nRed = *(lpTempBits+lLoop+3);
                    nGreen = *(lpTempBits+lLoop+4);
                    nBlue = *(lpTempBits+lLoop+5);
                    *(lpImageBits0+lLoop) = nRed;
                    *(lpImageBits0+lLoop+1) = nGreen;
                    *(lpImageBits0+lLoop+2) = nBlue;
    }
}

    for (lLoop = 0; lLoop <= 61439; lLoop++) {
            *(lpTempBits+lLoop) = *(lpImageBits1+lLoop);
    }

    for (lLoop = 3; lLoop <= 61434; lLoop = lLoop + 3) {
            nRed = *(lpTempBits+lLoop-3);
            nGreen = *(lpTempBits+lLoop-2);
            nBlue = *(lpTempBits+lLoop-1);
            nVal0 = (int) (.299*nRed + .587*nGreen + .114*nBlue);

            nRed = *(lpTempBits+lLoop);
            nGreen = *(lpTempBits+lLoop+1);
            nBlue = *(lpTempBits+lLoop+2);
```

```
            nVal1 = (int) (.299*nRed + .587*nGreen + .114*nBlue);

            nRed = *(lpTempBits+lLoop+3);
            nGreen = *(lpTempBits+lLoop+4);
            nBlue = *(lpTempBits+lLoop+5);
            nVal2 = (int) (.299*nRed + .587*nGreen + .114*nBlue);

            val[0] = nVal0;
            val[1] = nVal1;
            val[2] = nVal2;

            for (j = 0; j <= 3; j++) {
                    for (k = 0; k <= 3; k++) {
                            if (val[k] < val[j]) {
                                    nTemp = (int) val[j];
                                    val[j] = val[k];
                                    val[k] = (unsigned char) nTemp;
                            }
                    }
            }

            if (nTemp == nVal0) {
                    nRed = *(lpTempBits+lLoop-3);
                    nGreen = *(lpTempBits+lLoop-2);
                    nBlue = *(lpTempBits+lLoop-1);
                    *(lpImageBits1+lLoop) = nRed;
                    *(lpImageBits1+lLoop+1) = nGreen;
                    *(lpImageBits1+lLoop+2) = nBlue;
            }

            if (nTemp == nVal1) {
                    nRed = *(lpTempBits+lLoop);
                    nGreen = *(lpTempBits+lLoop+1);
                    nBlue = *(lpTempBits+lLoop+2);
                    *(lpImageBits1+lLoop) = nRed;
                    *(lpImageBits1+lLoop+1) = nGreen;
                    *(lpImageBits1+lLoop+2) = nBlue;
            }

            if (nTemp == nVal2) {
                    nRed = *(lpTempBits+lLoop+3);
                    nGreen = *(lpTempBits+lLoop+4);
                    nBlue = *(lpTempBits+lLoop+5);
                    *(lpImageBits1+lLoop) = nRed;
                    *(lpImageBits1+lLoop+1) = nGreen;
                    *(lpImageBits1+lLoop+2) = nBlue;
    }
}

    for (lLoop = 0; lLoop <= 61439; lLoop++) {
            *(lpTempBits+lLoop) = *(lpImageBits2+lLoop);
    }

    for (lLoop = 3; lLoop <= 61434; lLoop = lLoop + 3) {
            nRed = *(lpTempBits+lLoop-3);
            nGreen = *(lpTempBits+lLoop-2);
            nBlue = *(lpTempBits+lLoop-1);
            nVal0 = (int) (.299*nRed + .587*nGreen + .114*nBlue);

            nRed = *(lpTempBits+lLoop);
            nGreen = *(lpTempBits+lLoop+1);
            nBlue = *(lpTempBits+lLoop+2);
            nVal1 = (int) (.299*nRed + .587*nGreen + .114*nBlue);

            nRed = *(lpTempBits+lLoop+3);
            nGreen = *(lpTempBits+lLoop+4);
            nBlue = *(lpTempBits+lLoop+5);
            nVal2 = (int) (.299*nRed + .587*nGreen + .114*nBlue);

            val[0] = nVal0;
            val[1] = nVal1;
            val[2] = nVal2;

            for (j = 0; j <= 3; j++) {
                    for (k = 0; k <= 3; k++) {
```

```
                        if (val[k] < val[j]) {
                                nTemp = (int) val[j];
                                val[j] = val[k];
                                val[k] = (unsigned char) nTemp;
                        }
                }
        }

        if (nTemp == nVal0) {
                nRed = *(lpTempBits+lLoop-3);
                nGreen = *(lpTempBits+lLoop-2);
                nBlue = *(lpTempBits+lLoop-1);
                *(lpImageBits2+lLoop) = nRed;
                *(lpImageBits2+lLoop+1) = nGreen;
                *(lpImageBits2+lLoop+2) = nBlue;
        }

        if (nTemp == nVal1) {
                nRed = *(lpTempBits+lLoop);
                nGreen = *(lpTempBits+lLoop+1);
                nBlue = *(lpTempBits+lLoop+2);
                *(lpImageBits2+lLoop) = nRed;
                *(lpImageBits2+lLoop+1) = nGreen;
                *(lpImageBits2+lLoop+2) = nBlue;
        }

        if (nTemp == nVal2) {
                nRed = *(lpTempBits+lLoop+3);
                nGreen = *(lpTempBits+lLoop+4);
                nBlue = *(lpTempBits+lLoop+5);
                *(lpImageBits2+lLoop) = nRed;
                *(lpImageBits2+lLoop+1) = nGreen;
                *(lpImageBits2+lLoop+2) = nBlue;
        }
    }
  }

    GlobalUnlock(hTemp);
    GlobalFree(hTemp);

    /* unlock memory bitmap */
    GlobalUnlock(hgImageBits[0]);
    GlobalUnlock(hgImageBits[1]);
    GlobalUnlock(hgImageBits[2]);

    return;

}

int CaptureImage(int portid, HWND hWnd)
/* this procedure obtains an image from the board and places it into a memory buffer */
{
    unsigned char FAR           *lpImageBits0, *lpImageBits1, *lpImageBits2;
    unsigned long               lLoop = 0;
    int                         nVal = 0;
    extern HANDLE               hgImageBits[3];
    DWORD                       dwStart, dwCurrent;

    nVal = 0;
    lpImageBits0 = GlobalLock(hgImageBits[0]);
    lpImageBits1 = GlobalLock(hgImageBits[1]);
    lpImageBits2 = GlobalLock(hgImageBits[2]);

    /* test acknowledge signal */
    WritePort(portid, 0x80);
    WritePort(portid, 0x40);

    /* reset board */
    WritePort(portid, 0x00);

    for (lLoop = 0; lLoop <= 61437; lLoop=lLoop+3) {
    /* capture image */
        /* get starting time to use as a timeout */
        dwStart = GetCurrentTime();
        while(1) {
                nVal = GetbACK(portid);
```

```
                if (nVal) break;
                dwCurrent = GetCurrentTime();
                if (dwCurrent > (dwStart + 1000))
                {
                        WritePort(portid, 0x00);
                        MessageBox(hWnd, "Device Reset", "Error",
                                MB_OK);
                        GlobalUnlock(hgImageBits[0]);
                        GlobalUnlock(hgImageBits[1]);
                        GlobalUnlock(hgImageBits[2]);
                        return(1);
                }
        }

        nVal = ReadPort(portid);
        nVal = 255 - nVal;

        *(lpImageBits0 + lLoop) = (unsigned char) nVal;
        *(lpImageBits0 + lLoop + 1) = (unsigned char) nVal;
        *(lpImageBits0 + lLoop + 2) = (unsigned char) nVal;

        WritePort(portid, 0x04);
        while(1) {
                nVal = GetbACK(portid);
                if (!nVal) break;
                if (dwCurrent > (dwStart + 1000))
                {
                        WritePort(portid, 0x00);
                        MessageBox(hWnd, "Device Reset", "Error",
                                MB_OK);
                        GlobalUnlock(hgImageBits[0]);
                        GlobalUnlock(hgImageBits[1]);
                        GlobalUnlock(hgImageBits[2]);
                        return(1);
                }
        }

        WritePort(portid, 0x00);

}

dwStart = GetCurrentTime();

    for (lLoop = 0; lLoop <= 61437; lLoop=lLoop+3) {

        while(1) {
                nVal = GetbACK(portid);
                if (nVal) break;
                dwCurrent = GetCurrentTime();
                if (dwCurrent > (dwStart + 1000))
                {
                        WritePort(portid, 0x00);
                        MessageBox(hWnd, "Device Reset", "Error",
                                MB_OK);
                        GlobalUnlock(hgImageBits[0]);
                        GlobalUnlock(hgImageBits[1]);
                        GlobalUnlock(hgImageBits[2]);
                        return(1);
                }
        }

    nVal = ReadPort(portid);
    nVal = 255 - nVal;

    *(lpImageBits1 + lLoop) = (unsigned char) nVal;
    *(lpImageBits1 + lLoop + 1) = (unsigned char) nVal;
    *(lpImageBits1 + lLoop + 2) = (unsigned char) nVal;

    WritePort(portid, 0x04);
    while(1) {
            nVal = GetbACK(portid);
            if (!nVal) break;
            dwCurrent = GetCurrentTime();
            if (dwCurrent > (dwStart + 1000))
            {
```

```
                WritePort(portid, 0x00);
                MessageBox(hWnd, "Device Reset", "Error",
                        MB_OK);
                GlobalUnlock(hgImageBits[0]);
                GlobalUnlock(hgImageBits[1]);
                GlobalUnlock(hgImageBits[2]);
                return(1);
        }
    }

    WritePort(portid, 0x00);

    }

    dwStart = GetCurrentTime();

    for (lLoop = 0; lLoop <= 61437; lLoop=lLoop+3) {

        while(1) {
                nVal = GetbACK(portid);
                if (nVal) break;
                dwCurrent = GetCurrentTime();
                if (dwCurrent > (dwStart + 1000))
                {
                        WritePort(portid, 0x00);
                        MessageBox(hWnd, "Device Reset", "Error",
                                MB_OK);
                        GlobalUnlock(hgImageBits[0]);
                        GlobalUnlock(hgImageBits[1]);
                        GlobalUnlock(hgImageBits[2]);
                        return(1);
                }
        }

    nVal = ReadPort(portid);
    nVal = 255 - nVal;

    *(lpImageBits2 + lLoop) = (unsigned char) nVal;
    *(lpImageBits2 + lLoop + 1) = (unsigned char) nVal;
    *(lpImageBits2 + lLoop + 2) = (unsigned char) nVal;

    WritePort(portid, 0x04);
    while(1) {
        nVal = GetbACK(portid);
        if (!nVal) break;
        dwCurrent = GetCurrentTime();
        if (dwCurrent > (dwStart + 1000))
        {
                WritePort(portid, 0x00);
                MessageBox(hWnd, "Device Reset", "Error",
                        MB_OK);
                GlobalUnlock(hgImageBits[0]);
                GlobalUnlock(hgImageBits[1]);
                GlobalUnlock(hgImageBits[2]);
                return(1);
        }
    }

    WritePort(portid, 0x00);

    }

    GlobalUnlock(hgImageBits[0]);
    GlobalUnlock(hgImageBits[1]);
    GlobalUnlock(hgImageBits[2]);

    return(0);
}
```