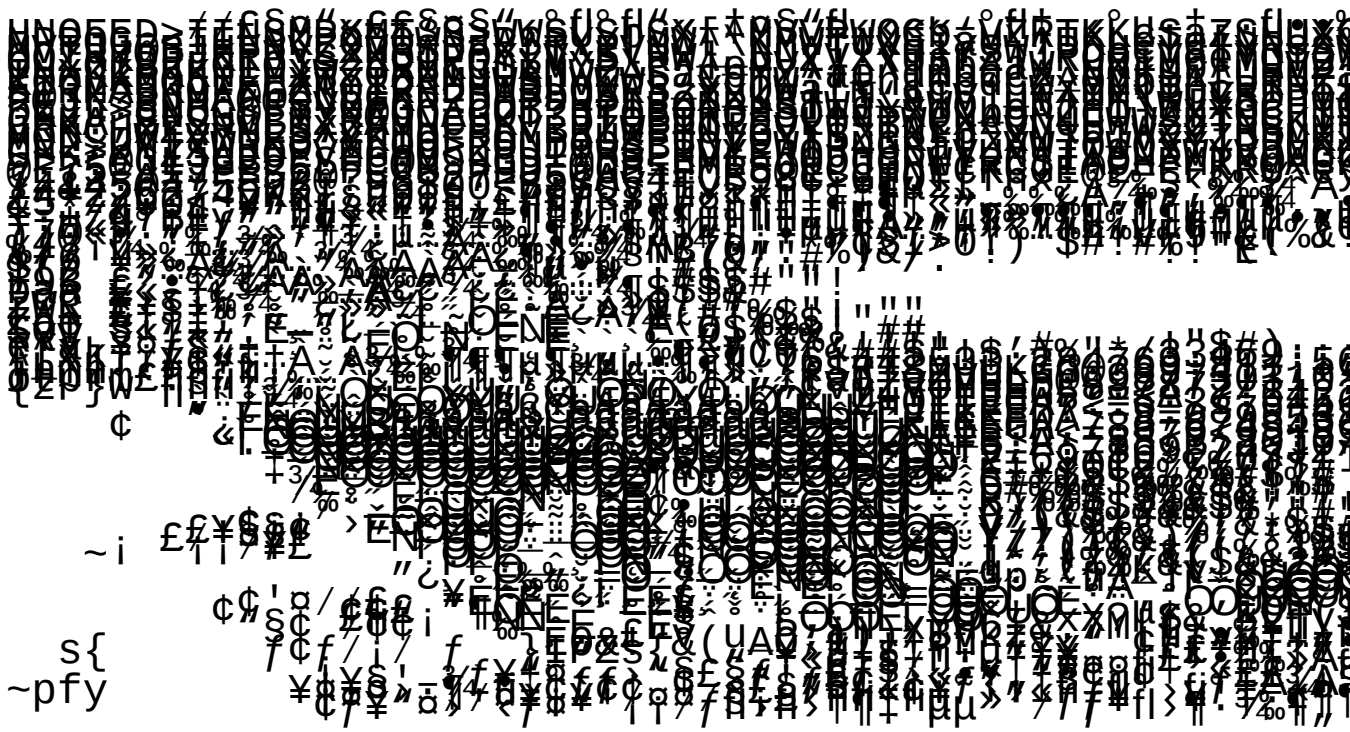
Jasmine Scanner Manual

BY Joe Maleson

Version of August 11, 1980

[IVY]<Jasmine>Docs>JasmineManual.PRESS

This is the manual for Jasmine, a low-cost experimental optical scanner which operates at a maximum resolution of 96 8-bit samples per inch. It is intended to contain all the information necessary to build hardware and software interfaces, and to understand the scanner electronics.



XEROX
PALO ALTO RESEARCH CENTER
3333 Coyote Hill Road
Palo Alto California 94304

Contents

1. Introduction

This document describes Jasmine, a low-cost experimental scanner. Jasmine moves an original document past a linear detector array to produce a gray-level image which can be displayed immediately on a CRT, stored on a file, or transmitted to a printer for hardcopy. It operates at a maximum resolution of 96 8-bit samples per inch, and moves paper at a maximum speed of 2 inches per second. It communicates through a simple computer interface which allows it to connect to an unmodified AltoII or D0. The scanner speed and resolution closely matches the Alto/D0 capabilities: neither machine can accept data at paper speeds above 2"/sec; doubling the resolution would produce about 4Mbits/page which is unrealistic for the Alto/D0 storage capacities.

The scanner hardware consists of two printed circuit boards containing about 40 TTL chips, two power supplies (+5V and -12V), a modified TC200 telecopier paper transport driven by a small stepper motor, an f4.0 28mm lens, a front-surface mirror, and a light bar containing ten 48V AC bulbs. The unit is packaged for desktop use in a sheet metal box measuring 15 3/4" x 19" x 4"; the paper transport housing rises about 3" above the top of the box, and lifts on hinges for clearing documents. The on/off button is mounted on the transport top. Four screws attach the metal cover of the scanner box to the base. On the rear plate are a female AC plug, a 3/4 amp fuse, and a female 25 pin connector.

The electronics consists of a simple state machine which drives an A/D converter and feeds the 8-bit output values into a FIFO. In addition, per-pixel correction is provided in order to produce a uniform digital output range for each element in the detector array. There are 6 bits of offset correction and 6 bits of gain correction for each element, held in three 1024x4 static RAMs. The output variance among elements when no light is incident is called *dark current leakage*, and produces a DC offset which must be corrected. The gain correction takes care of the light response variation among elements.

People

The Jasmine scanner represents a joint effort between the Computer Science Lab and the Optical Sciences Lab at Xerox PARC. The analog electronics were designed by Ed McCreight (CSL), the original mechanical and optical design were done by Gary Starkweather (OSL) and the digital electronics and software were produced by Joe Maleson (CSL). Fabrication and test of initial build of 25 units was carried out by Terry Haney, Frank Vest, and John Henning (Garage).

2. External Interface

Jasmine connects to an external computer through an 8-bit input and 8-bit output bus. Data transfer from Jasmine requires an additional input and output control bit.

2.1 Jasmine Control

The command byte from the controlling computer to the scanner is functionally divided into: Enable[0], Command[1-3], and Data[4-7]. To avoid hardware transition states, the command byte is first sent with Enable low (0), then high (1), and finally low (0). For commands which do not require any data, the state of the four Data bits is not important.

The Jasmine logic package is controlled by a state machine (sequencer), implemented with a ROM. There are four possible sequencer states: *Wait*, *Scan*, *Load*, and *Init*. When Jasmine is powered up, this sequencer will be in some random state. The sequencer is moved from any state to its *Wait* state by a LOAD command followed by a START command. From the *Wait* state, a START command moves the sequencer into *Scan* state, in which it produces the next scan line of samples. From the *Wait* state, a LOAD command moves the sequencer into *Load* state, in which it loads the correction RAMs with data received from the external computer. The *Init* state is entered when either 1) the sequencer is *Scan*ning, and a LOAD command is received, or 2) the sequencer is *Load*ing, and a START command is recived. The sequencer moves from *Init* to *Wait* on the falling edge of the START command line.

2.2 Commands

START (111)

The start command begins a new *Scan* cycle. In this cycle, the detector array first receives a pulse which causes it to begin clocking out samples, and then a uniform train of clock pulses is generated. The amount of time that light is integrated on a particular detector element is exactly the interval between output clocks for that element. The timing interval *between* start pulses is entirely regulated by the controlling computer. Variations in the duration of this interval will cause variations in the output levels of the samples.

In addition, the 4 bit Data field of the start command specifies the number of samples to skip after a sample has been digitized. This feature enables a scanner with 1024 elements to deliver either 1024, 512, 340, 256, 204, ..., or 64 [=1024/(skipCount+1)] digitized samples per scan line.

LOAD (110)

The load command begins a new *Load* cycle, and sets the internal address counters to receive the correction values for the first detector. Each successive load command causes the address counters to increment by one. When the last element has been corrected, or when a start command is received, the sequencer enters its *Wait* state.

MOTORCTL (101)

This command controls the stepper motor which moves the paper. Taking advantage of the motor's short duty cycle, increased torque is obtained by supplying high current for short periods (< 60 seconds). Current is cut off from the motor by asserting data bit 2 (x1xx). *Leaving the motor on for extended periods will cause overheating and possible damage.* The motor is turned by applying power to two coils according to the two low order data bits. The motor moves forward one step for each value in the sequence {3,2,0,1}, and in reverse for {1,0,2,3}.

WE1 (011)
WE2 (010)
WE3 (001)

These three commands load the 4 bit Data field into the top, middle, or low 4 bit fields of the currently selected 12 bit word of the correction RAMs. The top 6 bits of this word correspond to the offset correction, and the low 6 bits are the gain correction.

SETDELAY (000)

The potentially large bandwidth required to transfer samples from the Jasmine FIFO to the external computer is a problem in some applications where the computer cannot keep up with the transfer rate. A delay counter is implemented which provides additional delay between digitized samples in units of 1 microsecond. The 4 bit Data field thus specifies up to 15 microseconds of additional delay between output samples. (see also section 2.4 for timing details).

2.3 Data Transfer

A 64x8 FIFO buffers data bytes between Jasmine and the external computer. An output bit from the scanner (XBus.0) is high when the FIFO is ready with a data word. Two control bits from the computer are used to shift a data word out of the FIFO: the falling edge of XBus.Read' sends XBus.A0' on the FIFO shift-out line. After the FIFO receives the shift signal, the output ready bit will stay low until the next data byte is ready. This will take more than a microsecond for some FIFOs.

On the Alto, the XBus provides the input path from the scanner. Each time the XBus is read, two address bits are available on the connector, and a third bit indicates that the read is taking place. For convenience and speed, the "XBus read" signal is used to clock a flip-flop, with one of the address bits used as the input data. Thus, two XBus addresses are "safe," causing no change to the FIFO, and two XBus addresses cause the FIFO to shift out. For computers with other I/O interfaces, both the clock and data can be driven from complementary versions of the same output bit (clock low, data high causes a shift).

Note that the XBus read signal goes high when any XBus address fetch is issued, so in order to shift the FIFO, a non-XBus address must be stored into the Alto MAR.

The FIFO is reset at every START pulse, so that any data not read out at that time will be lost. Because the inter-sample timing is critical, it is impossible to do anything if a new data byte is ready and the FIFO is full: in this case, either the new data byte, or the first data byte in the FIFO, must be discarded. It is up to the external computer to ensure that the combination of integration time, skip count, and delay setting will not cause data to be lost. On the other hand, if the only purpose of the START pulse is to maintain constant integration time, there is no error signal generated if none of the data bytes are read out.

2.4 Timing

The maximum speed of the Reticon chip used as the detector array is 1MHz. Each sample which is skipped thus produces a minimum of 1 microsecond of delay. The A to D converter implemented for Jasmine provides a bit every clock time. Currently, Jasmine runs on a 250 nsec clock; including overhead for other control functions, Jasmine will deliver a digitized value every 15 clocks (3.75 microseconds). Thus, the time between digitized samples is (3.75+skipCount+delayCount) microseconds. The minimum time per scan line for the various values of skipCount and delayCount are given in Table 1.

skipCount	(nSamples)	delayCount 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	(1024)	3.84	4.86	5.89	6.91	7.94	8.96	9.98	11.01	12.03	13.06	14.08	15.10	16.13	17.15	18.18	19.20
1	(512)	2.43	2.94	3.46	3.97	4.48	4.99	5.50	6.02	6.53	7.04	7.55	8.06	8.58	9.09	9.60	10.11
2	(340)	1.96	2.30	2.64	2.98	3.33	3.67	4.01	4.35	4.69	5.03	5.37	5.71	6.05	6.39	6.74	7.08
3	(256)	1.73	1.98	2.24	2.50	2.75	3.01	3.26	3.52	3.78	4.03	4.29	4.54	4.80	5.06	5.31	5.57
4	(204)	1.58	1.79	1.99	2.19	2.40	2.60	2.81	3.01	3.21	3.42	3.62	3.83	4.03	4.23	4.44	4.64
5	(170)	1.49	1.66	1.83	2.00	2.17	2.34	2.51	2.68	2.85	3.02	3.19	3.36	3.53	3.70	3.87	4.04
6	(146)	1.42	1.57	1.72	1.86	2.01	2.15	2.30	2.45	2.59	2.74	2.88	3.03	3.18	3.32	3.47	3.61
7	(128)	1.38	1.50	1.63	1.76	1.89	2.02	2.14	2.27	2.40	2.53	2.66	2.78	2.91	3.04	3.17	3.30
8	(113)	1.33	1.44	1.55	1.67	1.78	1.89	2.01	2.12	2.23	2.35	2.46	2.57	2.68	2.80	2.91	3.02
9	(102)	1.30	1.40	1.51	1.61	1.71	1.81	1.91	2.02	2.12	2.22	2.32	2.42	2.53	2.63	2.73	2.83
10	(93)	1.28	1.37	1.47	1.56	1.65	1.74	1.84	1.93	2.02	2.12	2.21	2.30	2.40	2.49	2.58	2.67
11	(85)	1.25	1.34	1.42	1.51	1.59	1.68	1.76	1.85	1.93	2.02	2.10	2.19	2.27	2.36	2.44	2.53
12	(78)	1.23	1.31	1.39	1.46	1.54	1.62	1.70	1.78	1.85	1.93	2.01	2.09	2.17	2.24	2.32	2.40
13	(73)	1.22	1.30	1.37	1.44	1.52	1.59	1.66	1.73	1.81	1.88	1.95	2.03	2.10	2.17	2.25	2.32
14	(68)	1.21	1.28	1.34	1.41	1.48	1.55	1.62	1.68	1.75	1.82	1.89	1.96	2.02	2.09	2.16	2.23
15	(64)	1.20	1.26	1.33	1.39	1.46	1.52	1.58	1.65	1.71	1.78	1.84	1.90	1.97	2.03	2.10	2.16

Table 1. Scan line time in milliseconds for various sample resolutions and rates

3. Analog Electronics

The analog electronics converts the detector array outputs to 8-bit digital values, individually corrected for gain and offset variations. A separate analog board is used due to packaging constraints. For noise reduction, some of the analog components are in a shielded area formed by a solid ground plane and the metal lens shroud. A single point ground connection is used between the analog and digital electronics.

3.1 Detector Array

The Reticon RL1024G detector array uses photodiodes to discharge capacitors. The charge removed from each capacitor is proportional to the light incident on the associated photodiode. Each detector cell is connected to the video line by a MOS switch. The switches are sequentially closed for one clock period, during which time the residual charge must be sensed, and then the capacitor must be recharged.

The differential video buffer outputs are common source amplifiers, and are connected to +5v through a 10K pot. The pot is adjusted to provide balanced inputs to the LM733 video amplifier. The buffer outputs operate with an offset voltage near +5V, and the zener/capacitor/resistor circuit is designed to provide -2.3 volts to the video recharge lines so as to maintain a common mode voltage at the LM733 differential inputs midway between the amplifier reference voltages of 0V and +5V. The end of scan signal is not used, and is tied high to avoid introducing noise. The recharge gate is normally held high (off) by a 10k resistor to +5v, and is turned on (connected to -10v) when the LM339 comparator's negative input rises above TTL threshold (~1.5v).

3.2 Video Signal

The LM733 differential video amplifier provides a linear voltage gain of ~100 to boost the ~25mv detector array output to the 2.5v range. A CD4053 analog multiplexor is switched for 1 msec to allow charge sharing between the .01 mf capacitor and a 220pf capacitor connected to the hold amplifier. When the switch is opened, the detector cell is recharged and a DC restore is performed, while the analog sample is digitized. During the recharge operation, the video buffer outputs are at their fully charged (black) state.

DC restore is accomplished by connecting the analog multiplexor input to ground, thereby making the voltage drop across the .01 mf capacitor equal to the difference between V_{black} and ground. When the next sample is presented, the LM733 output voltage will increase to V_{sample}; the multiplexor input voltage must increase by the same amount, and since it was originally at ground, it is now at $V_{sample}-V_{black}$, the sample voltage with the DC component removed. When the multiplexor is switched for charge sharing between the capacitors, the hold capacitor will move to this level . Note that the previous voltage difference across the hold capacitor will have a very small effect on the final voltage, typically ~1%. This slight blurring between adjacent sample levels is equivalent to shifting the input image by .0001".

The hold amplifier is implemented with a CA3140 Op Amp, providing a final voltage gain setting. The exact gain is the ratio of the feedback resistor (marked 5.1K, but typically 820 ohms) and the resistor to ground (1.1k). To avoid clipping problems near black level in the A-D conversion, an offset bias is added by a large resistor to +5v (marked 80K, but typically 1M); decreasing the value of this resistor will increase the offset bias. On the output of the hold amplifier is a 1k bias balancing resistor.

3.3 A-D Conversion

The A-D converter is implemented with a NE5008 multiplying DAC, a comparator, and a successive approximation register. The SAR provides an 8-bit input value to the DAC, with the most significant unknown bit set to 1, and the other unkown bits set to 0. The DAC produces an output between 0 and 2.55 volts (~.01 volts/bit). This voltage is compared to the output of the hold amplifier; if it is higher, the most significant unknown bit is set to 0, and if lower 1. The SAR

then provides a new 8-bit input value, using the known high order bits, and a 1 in the most significant unknown position. After 8 clocks (2 msecs), all 8 bits are known, and the SAR's EOC' goes low, indicating that the digital approximation is complete.

The per-pixel gain correction is implemented by using an ADC-MC8BC DAC to vary the NE5008 negative reference voltage. The scale DAC reference output is connected through a unity gain buffer amp to provide a high impedence positive reference for the NE5008. 15pf to -10?

3.4 Noise Filtering

The low level analog outputs of the detector array make it important to reduce electrical noise. This is the motivation for the shielded analog section, the various capacitors on the analog board, and the standard pi filters on the power supplies.

4. Digital Electronics

The digital electronics provide the control signals for image sampling, handling the correction RAMs, A/D conversion, and stepper motor control. The user's view of the digital functionality was covered in section 2: External Interface. This section discusses the internal operation in more detail.

4.1 Finite State Machine

Digital control is based on a small finite state machine (FSM) with 32 states, built out of two ROMs and two latches. The FSM operates on a 250 nsec clock. The 512x8 State ROM produces the next state number based on the current state and the values of four input signals: the START and LOAD signals from the controlling processor, and the Sample and ScanDone signals from the sequencing logic. The 32x8 Control ROM produces 7 output signals based on the current state number. The ROM outputs are loaded into the latches at every clock cycle. Five signals (VirScanClk, AnalogOn, DCRestore, and ScanStart) are concerned with the analog sampling logic. One signal (ResetAD) controls the successive approximation register used for A/D conversion. The last signal (ResetCtrs) controls various registers in the sequencing logic. A complete code listing for the state machine is included as Appendix 2.2.

4.2 Sequencing Logic

The sequencing logic consists of three counters: the Delay counter, the Sample counter, and the Scan counters. The Delay counter is provided for control flow, as described in section 2 on Timing. The Sample counter provides another level of bandwidth control: reducing the number of input samples per scan line allows the controlling processor to accept more scan lines per unit time, and will also reduce the storage necessary for holding an image. The Sample counter is loaded on each external START pulse with the four data bits, as described in section 2.2.

The Scan counters are three four-bit counters which cycle through the RAM addresses. They are initialized with a constant value (-1025, based on the array size of 1024 elements) at every external START pulse. The Scan counters increment once per scan clock. The high order carry-out bit is the input signal ScanDone to the FSM, and causes the FSM to return to its *Wait* state.

4.3 Stepper Motor Control

The stepper motor is made up of two coils, each with two windings. The motor turns when current is applied in proper sequence to each winding. The two windings of each coil are always in opposite states, so that when current is flowing through one winding no current is flowing through the other. Each winding is connected to -12V through a TIP 122 Darlington pair. An LM339 comparator is used to produce a 0V or -12V input at the TIP 122 base from the TTL output of a NAND gate. Each NAND gate has one input from the RunMotor signal, and one input from either StepControlA, StepControlB, StepControlA', or StepControlB'. These three signals are held in a latch which is set by the external MotorCtl command. The RunMotor signal should normally be FALSE, in order to avoid motor overheating. See Appendix 1.4 for the electronics schematic.

5. Mechano-Optical Design

5.1 Paper Transport

The Jasmine scanner uses a modified Xerox TC-200 telecopier paper transport. TC-200 parts removed include: DC motor assembly, illuminator bar, LED and light sensor, automatic paper feed wheel, and plastic cover. The base casting is somewhat modified (see Appendix 2.8). A new cover and a release bar extender are added. After the paper moves through the transport, it exits the scanner housing through a rectangular rear aperture, and is turned up to an angle of about 60 ^{o} by a curved metal platform. The final paper turner allows the Jasmine unit to be placed close to a wall, and will generally handle documents as stiff as photographic paper. For installations where excessively heavy input material commonly causes jamming problems at the paper turner, a modified platform with smaller curvature should be used.

One turn of the paper drive gear results in 2" of paper movement. A 48 step/revolution stepper motor turns a 12 tooth pulley, connected by a drive belt to the transport's 48 tooth gear, resulting in 96 steps/inch. The stepper motor typically generates 6 oz-inches of torque, and so the 4:1 gearing provides 24 oz-inches at the paper. The torque decreases as paper speed increases above .4"/sec. Jamming problems may be caused by attempting to move thick documents at high speed. (The standard software positions the document by moving the paper at 2"/sec; there should be a "thick document" setting which slows down the positioning rate).

5.2 Document Illumination

Illumination is provided by a strip of 48V incandescent bulbs. This causes non-uniform illumination, which is brightest in the center, and falls off at either side. In order to partially compensate for this non-uniformity, the center two bulbs are removed. The remaining variation is corrected by the Gain RAMs, in the same way that non-uniform detector element response is handled.

5.3 Optical Path

A scan line from the document is reflected off a 45 ^{o} mirror and imaged through a Schneider Componon f4.0 28mm lens onto the detector array. The optical path measurements for figure 5.1 are derived as follows:

array length = 1.008"; (8.5" x 96pixels/inch)x1.008" = .803"
magnification m = 8.5"/.803" = 10.585
F = effective focal length = 29.4 mm
O = F(1+m) = 339.44mm = 13.363"
P = F(1+1/m) = 32.17mm = 1.267"
D = object to image = O+P+(N1-N2) = 368.41mm = 14.504"
Allowing for + or - 1% focal length variations,
364.75mm = 14.36" < $D_{typical}$ < 14.65" = 372.16mm
and image plane variation is +/-0.32mm

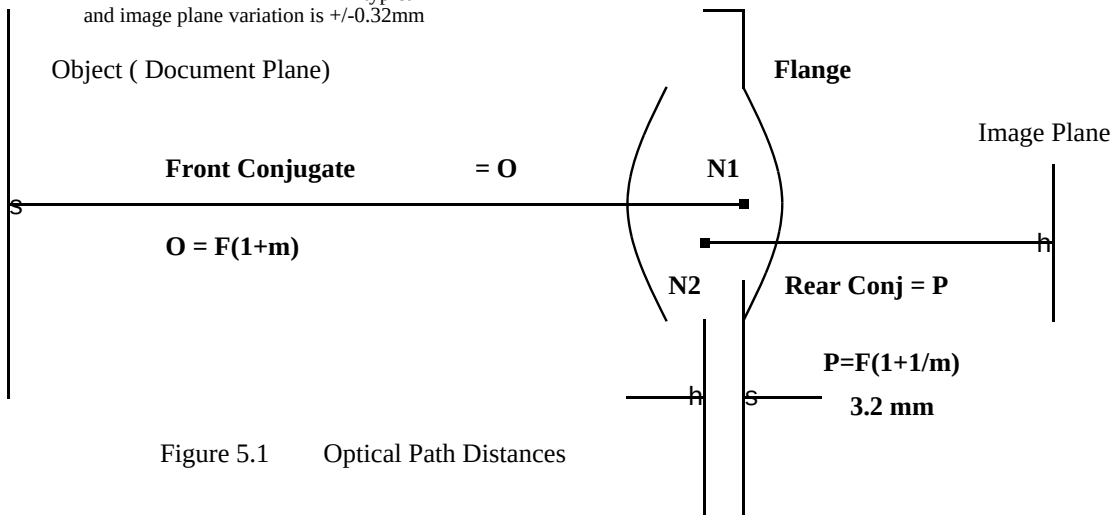


Figure 5.1 Optical Path Distances

5.4 Lens Alignment/Focus/Magnification

The scanner must be set up so that each scan line from the document falls across the detector array in focus and at the appropriate magnification. Any skew present in the detector array due to inexact bonding is corrected by adjusting the position of the analog board in its mounting assembly. Minor adjustments to the angle of the lens are made with the top alignment screw so that the scan line hits the entire detector array. Focus is set by changing the distance between the lens and the detector. This can be accomplished by inserting a test pattern into the scanner, and rotating the lens in the lens mount until optimal focus is reached. Magnification is determined by the distance from the lens to the document. The lens assembly and circuit boards are moved as a unit until the desired magnification is reached. For additional set-up instructions, see Chapter 7: Check-out Procedures.

6. User Software

6.1 User level interface: Scan.BCD, Scan.RUN

[IVY]<Jasmine>Scan.BCD is a Mesa 6 system for driving the Jasmine scanner. It runs on either D0s or AltoIIs. It will display input pages on the screen, allows zooming, enables a user to set various of the Jasmine device parameters, and writes output files. The output files have an AIS header, and a PRESS trailer, and can be used as input to systems expecting either AIS or PRESS format files. Scan4.BCD provides the same interface, but with output to the D0 4-bit per point display. A similar package exists in BCPL for AltoIIs as Scan.RUN.

In specifying to a page position, the term *Jasmine coordinate* is used: this refers to the basic Jasmine pixel resolution of 1/96". In addition to the hardware state maintained in the scanner, there is some state maintained in the software. The *current window* refers to a rectangle on the page in Jasmine coordinates; when a new page is inserted, the current window is set to (xstart: 0,xlength: 1024,ystart: 0,ylength: 1024). Any of the four coordinates can be changed using the XS, XL, YS, or YL commands. All four coordinates will be updated by the Zoom command. The *current scan length* refers to the number of output dots per scan line to be displayed on the screen. Normally, the scan length is set to 608 (full screen width), but the user may desire a narrower length in order to display a narrow, tall image. The current scan length is also used in creating an output file.

The user interface to Scan consists of a key letter interpreter. Each command is initiated by typing the first (and sometimes second) characters of the command. Unrecognized letters are ignored.

The key letters in the following list of commands are capitalized.

?: The question mark causes the current settings of black, white, xstart, xlength, ystart, and ylength to be printed out.

Black: Prompts the user for a decimal number to be used by the halftoning routine; any pixels less than or equal to this number will be printed as black.

Calibrate: The scanner has a separate correction value for each pixel. The calibrate command prompts the user to "Insert sheet of white paper" and to "Press any key when ready." Best results will be obtained from an opaque, flat white sheet such as photographic paper. Any irregularities in the calibration sheet will cause uneven calibration. When the user presses a key to indicate that the calibration sheet is in place, the scanner will average a number of white lines, and set the correction values so that each pixel produces the same white response. This is not an optimal calibration scheme: it searches iteratively for the best solution. After each iteration it prints out the minimum and maximum value for the line, and the cumulative difference of the 1024 elements from the desired value. This routine does not incorporate dark correction, so that while a light image will be fairly uniform, a dark image will contain streaks. John Warnock is currently working on a better procedure.

Delay: Prompts the user for a decimal number from 0 to 15 to be used as the delay (in microcoseconds) between pixels delivered from the Jasmine. *This control is mainly for hardware debugging. Incorrect settings will cause the scanner to stop working, and all settings in the correct range will cause identical output. The Scan program will automatically set the delay correctly* .

Erase: Erases the user screen.

Forward Prompts the user for a decimal number ("nSteps"). The paper will be moved forward this many steps. (1 step = 1 Jasmine coordinate = 1/96")

Halftone: The scanner will read in pixels from the *current window* and halftone them onto the screen using the *current scan length* . Pressing any key will terminate the halfonte command.

Initialize: The correction values in the scanner will all be initialized to a constant value. This command is useful for determining how well the calibration routines are working.

Length of scan : Prompts the user for a decimal number to be used as the *current scan length* .

New page: This command ejects the current page, and prompts the user to "Press any key when ready"; the user should insert the new page, and press a key. This command sets the *current window* to (xstart: 0,xlength: 1024,ystart: 0,ylength: 1024).

Quit: Ejects the current page, and returns to the Executive.

Reverse: Prompts the user for a decimal number ("nSteps"). The paper will be moved in reverse this many steps. (1 step = 1 Jasmine coordinate = 1/96").

Skip count : Prompts the user for a decimal number from 0 to 15 to be used by the scanner as a "skip count": that many pixels will be skipped for each pixel delivered. The software will use the same skip count to drop out scan lines. This command is useful for reducing the storage required for a low resolution image. Note that the image produced by skipping pixels is *undersampled*, which may cause severe aliasing and other problems. A better way to create the lower resolution image would be by averaging the pixels instead of throwing them away. The hardware does not support this function .

Time for integration : Prompts the user for a decimal number to be used as the integration time in 38.08 microsecond "ticks". This setting defaults to 656 ticks, or about 1/40 second. For dark documents, it is possible to get more dynamic range by increasing the integration time. *Reducing the integration time may create impossible timing constraints. If the integration time is less than the time required to read in a scan line, the scanner will stop working* .

White: Prompts the user for a decimal number to be used by the halftoning routine; any pixels greater than or equal to this number will be printed as white.

XStart: Prompts the user for a decimal number from 0 to 1023 which will be used as the current window horizontal start in Jasmine coordinates.

XLen: Prompts the user for a decimal number which will be used as the current window length in Jasmine coordinates. If XStart+XLen is greater than 1024, XLen will be set to 1024-XStart.

YStart: Prompts the user for a decimal number which will be used as the current window vertical start in Jasmine coordinates.

YLen: Prompts the user for a decimal number which will be used as the current window height in Jasmine coordinates.

Zoom: The zoom command waits for a new window to be indicated by the mouse. Pressing the left mouse button indicates the corner of the window. Holding down the left mouse button will allow the user to stretch the rectangle indicating the current window from the window origin to the mouse. Holding down the middle mouse button allows the user to reposition the window origin. When the appropriate window has been selected, hitting the right mouse button will terminate the zoom command, and the new window will be scanned in and displayed. Note that in the BCPL version, no rectangle is displayed, and the new window is indicated by depressing the left mouse button at one window corner, holding the button down as you move the mouse to the second corner, and then releasing the button.

6.2 Procedural interface

The interface to the Jasmine scanner is implemented in both MESA and BCPL. The MESA syntax is used in this document, but the transformation to BCPL syntax is straightforward. The interface described here handles the scanner data as an input stream. For speed, a number of buffers are pre-filled with input data; typically the calling routine will not have to wait for additional input from the scanner, because at least one of the buffers will already be full. The following routines are sufficient for most scanner operations.

JasmineInit: PROCEDURE;
Loads the device microcode, sets up appropriate Debugger and Finish processes, and gets the scanner into the Wait state with a reasonable delay setting. This routine must be called exactly once, before any of the other routines.

JasmineScanInit: PROCEDURE[block: POINTER,nWords: CARDINAL] RETURNS [scanStream: POINTER];
This routine initializes the *nWords* of memory beginning at address *block* for use as a ring of input buffers. It moves the paper so that the first scan line of the current window is imaged on the detector array. The routine returns a pointer which must be passed to subsequent procedures.

JasmineReadLine: PROCEDURE [scanStream: POINTER] RETURNS [data: POINTER TO PACKED ARRAY OF [0..377B]];
This routine returns a pointer to the next scan line of data. The number of bytes in the scan line is determined by the current window setting, which defaults to the entire scan line, and the current resolution (setting of the Sample counter). Because of the input buffering, the data will only be valid until the next call to JasmineReadLine.

JasmineScanClose: PROCEDURE [scanStream: POINTER];
This routine should be called before the next call to JasmineScanInit. It ensures that everything is left in a clean state.

JasmineSetWindow: PROCEDURE [xStart,xLen: CARDINAL, yStart: CARDINAL _ 0,yLen: CARDINAL _ 1400];
Sets the input window for the reading routines. After changing the input window, JasmineScanInit must be called before any additional calls to JasmineReadLine.

JasmineSetReadMode: PROCEDURE [readMode: [1..3]];
There are three modes of running JasmineReadLine: when *readMode* = 1, the current scan line is read without moving the paper; for *readMode* = 2, the line is read and the paper is moved (this is the default setting); for *readMode* = 3, the scan line is read, the paper is moved, and a delay of one integration time is inserted. On the Alto, when the display is running, the current microcode often "misses" clock ticks when reading a scan line (this is seen in the jittery motion of the cursor). By adding a delay cycle, two desirable effects are achieved: there will be no paper motion during the integration time, and the integration time of the line being digitized will not vary since that integration time was calculated when the microcode was not reading, and therefore not missing clocks .

The following low level routines allow the user to set various scanning parameters. Some combinations of parameters are impossible (i.e. integration time too short for digitizing the required samples, or delay time too short to allow the controlling processor to acquire the data); not all of

the impossible combinations are checked for, so that the programmer should invoke the follwoing routines with caution.

JasmineSetDelay: PROCEDURE [delay: [0..15]];
Sets the delay in microseconds/sample.
JasmineSetResolution: PROCEDURE [nSkips: [0..15]];
Sets the number of samples to be skipped for each sample digitized. The software will also skip *nSkips* scan lines between each digitized line. Resolution defaults to 0 (1024 elements).
JasmineSetTime: PROCEDURE [ticks: CARDINAL];
This routine sets the integration time per scan line in 38.08 microsecond ticks. Because there is AC variation in illuminator brightness at 120Hz, the integration time should be set to close multiples of 218.84 ticks (=1/120 second).
JasmineLoadRam: PROCEDURE [array: POINTER,len: CARDINAL];
A low level interface used to set the Gain/Offset RAMs. This routine is called by the calibration routine, and is of limited use to the casual programmer.
JasmineStep: PROCEDURE [nSteps: CARDINAL,forward: BOOLEAN];
A low level routine which moves the stepper motor forward and back. Moving the motor may cause the current page position to be lost.
JasmineCoord: PROCEDURE [coord: CARDINAL] RETURNS[nSamples: CARDINAL];
Converts scanning resolution coordinates (~250 microns) to pixel coordinates based on the current resolution setting. This routine will return the number of pixels on a scan line when *coord*=1024.
JasmineNewPage: PROCEDURE;
Feeds the new sheet of paper up to the document window, and resets the internal line counter.
JasmineEject: PROCEDURE;
Ejects the current page.
JasmineMotorOff: PROCEDURE;
Sets the RunMode line to FALSE, thereby cutting off current to the stepper motor.

6.3 Microcode Interface and Implementation

The microcode uses three fixed locations in low memory. Location 526B contains the integration time in 38.08 microsecond ticks. Location 736B is a pointer to a linked list of control blocks. Location 737B contains the data bits for the START command. The Alto microcode runs as part of the Memory Refresh Task, which is activated every 38.08 microseconds. A counter is initialized to the contents of 526B (*ScanTime*), and is decremented once per MRT activation. Normally, when the counter reaches zero the command at 737B (*StartCommand*) is issued, and location 736B (*ScanCBHead*) is examined: if it is non-zero, the command it points to is performed. If, however, a current command block is still being processed when the integration time counter reaches zero, the status of that command block is set to "StatusDATALATE," and ScanCBHead is set to zero.

The data types for a command block are:

```
ScanCommand: TYPE = CARDINAL;
    CommandREAD: ScanCommand = 0
    CommandDELAY: ScanCommand = 1
    CommandFORWARD: ScanCommand = 2
    CommandBACK: ScanCommand = 3
ScanStatus: TYPE = CARDINAL;
    StatusINUSE: ScanStatus = 1          //in general, all positive numbers are "in use" flags
    StatusFREE: ScanStatus = 0
    StatusDONE: ScanStatus = -1
    StatusDATALATE: ScanStatus = -2
ScanCB: TYPE = RECORD
[   link: POINTER TO ScanCB,
    command: ScanCommand,
    status: ScanStatus,
    buffer: POINTER
];
```

CommandREAD interprets the status field as the number of bytes to be read. If buffer=NIL (0) the input data is discarded. The high level read commands implement a horizontal window by inserting two read

commands on the command block chain, with the first block containing a NIL buffer pointer. When the current read command is completed, the status field is set to StatusDONE, and the next command on the chain is immediately interpreted. The other three commands cause command block execution to stop until the integration counter reaches zero. In addition to waiting for the next line time, CommandFORWARD and CommandBACK issue the appropriate output signals to move the paper in the desired direction.

7. Check-out Procedures

7.1 Adjusting the Output Level

The value of the 10K pot on the analog board is set to provide balanced input to the LM733 video amplifier. With your scope set at 2mV/division, sync on the *ScanStart* signal (pin #3B-10). Place a white strip of paper on the document window and examine the output of the video amplifier (pin #IC2-7). When the pot is below the minimum allowable value, the output is a constant high value (~4V). Turning the pot clockwise will cause the output signal to move through the correct setting range (Figure 7.1) to a point above the maximum allowable value, where the output will be a constant low value (~2V). The optimum setting is where high light level produces an output value just above the bottom constant line, and zero light level produces an output value just below the top line.

These images are printed using gray level fonts. If you have access to a full PRESS printer, the real images are contained in [IVY]<Jasmine>Docs>CheckOut.PRESS.

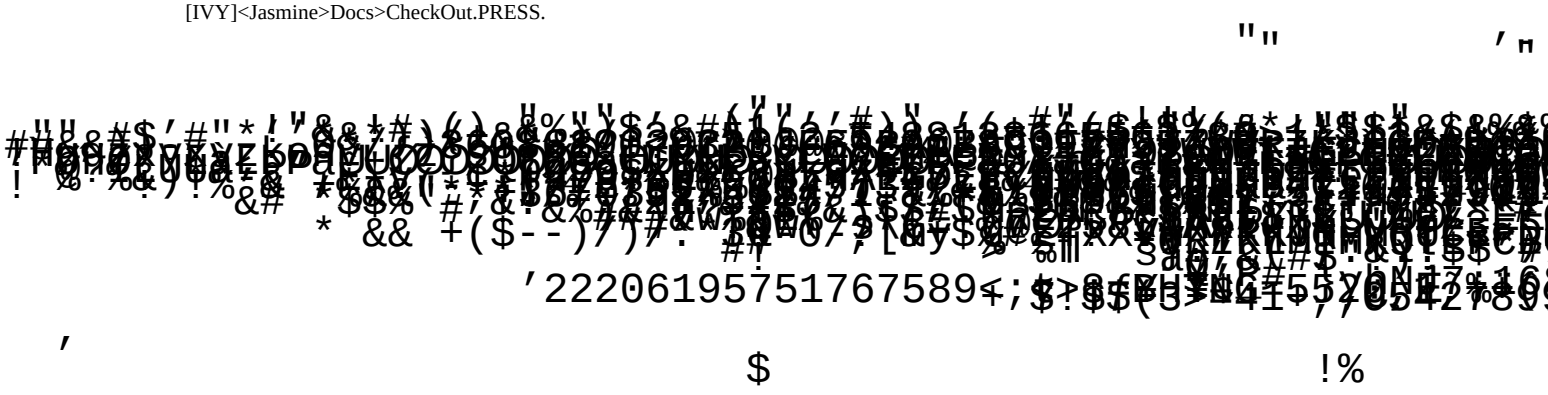


Figure 7.1: output at #IC2-7 with low, optimum, and high settings of pot

With the pot set to its optimum position, examine pin #IC8-3. The signal here should vary between 0V (black) and about -2.2V (white). If the signal goes below -2.5V, reduce the value of the final gain resistor #R10 until the maximum value falls into the permissable range. If the signal goes much above -2V, increase the value. (Levels below -2.5V will cause the A/D system to wrap around so that high light levels read as "black", and levels above -2.5V will reduce the maximum range of output values. Setting the gain for a nominal value of -2.2V reduces output range only slightly, and guarantees no wrap-around).

7.2 Optical Assembly

Fasten the analog board to the lens mount assembly. There is play in the board position due to oversized mounting holes: this allows corrections for minor variations in the position of the detector array elements. Fasten the lens mount assembly to the chassis at the midpoint of the adjustment slots. Thread the lens into the mounting collar as far as it will go. Set the lens aperture to *f*4 (full open).

The three optical adjustment steps are performed with an alternating pattern of broad black and white stripes in the transport; this allows easy identification of the image window for alignment and provides the appropriate patterns for magnification and focus adjustment. There is a small amount of interdependence in the adjustment steps: changing the focus may alter the alignment slightly.

7.2.1 Alignment

The lens axis must be positioned so that the image of the document window reflected by the mirror falls across the detector elements. Small variations in the tilt of the lens to achieve this goal are

controlled by the 2" lens mount adjusting screw. A rough setting can be performed by visually checking that the image is correctly aligned. Fine adjustment is performed by watching the scanner output on the Alto screen, and ajusting for the best image.

7.2.2 Focus

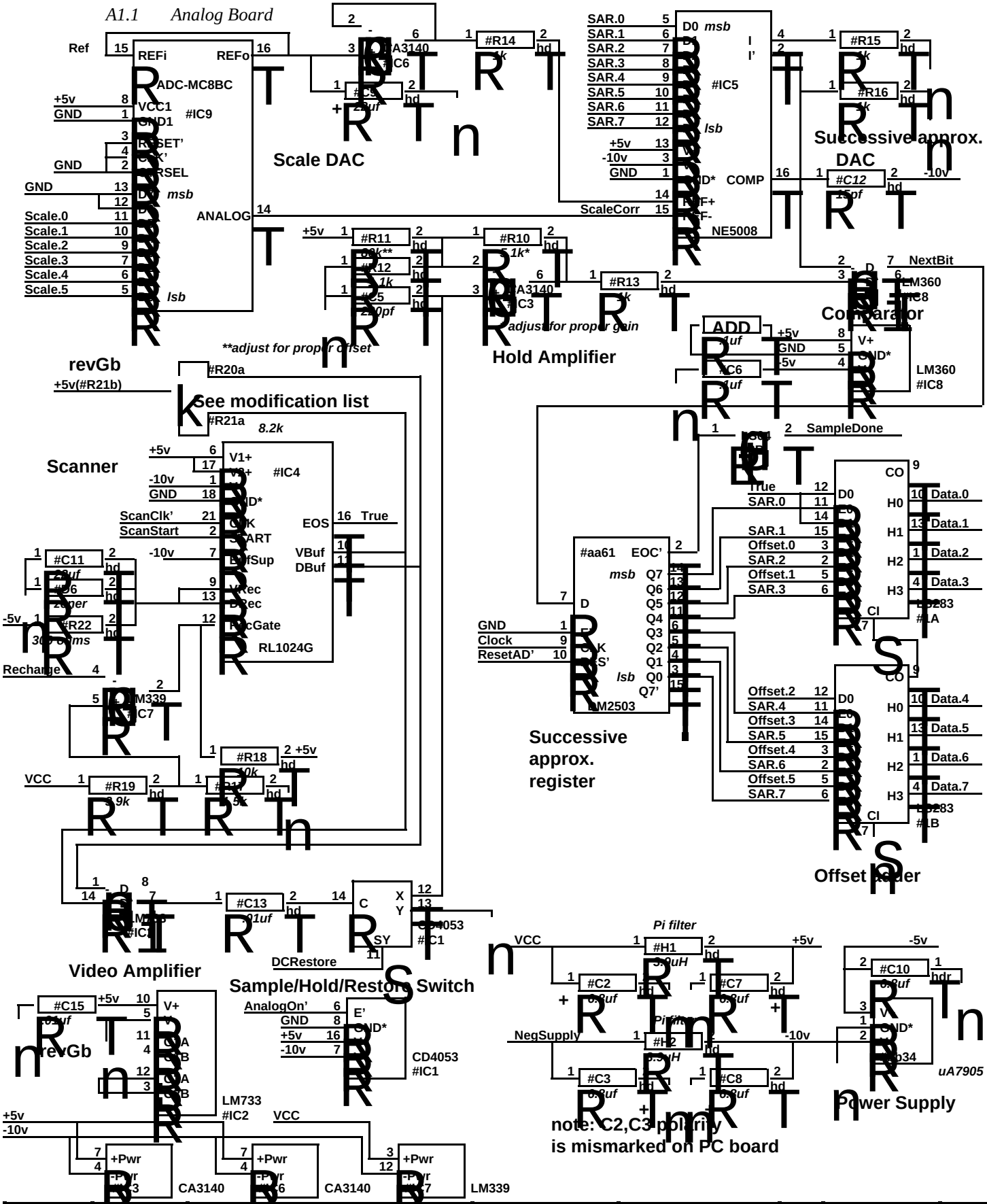
The image is focused by varying the distance from the lens to the detector array. This is achieved by turning the lens in the mounting threads until the edges of the stripe pattern are as sharp as possible. When the mounting bracket is under tension from the adjusting screw, the resulting pressure on the lens threads will hold the lens securely. For final focus, insert a text page into the scanner, and use the software to zoom in on a small (~100x100 pixel) area. Rescan the area while adjusting the focus for an optimal image. While adjusting focus, it is easy to change the lens aperture, thereby cutting down the brightness of the image. Make sure the aperture is always fully open.

7.2.3 Magnification

The image magnification is determined by the distance from the lens mount assembly to the document window. Moving the assembly away from the document window increases the magnification.

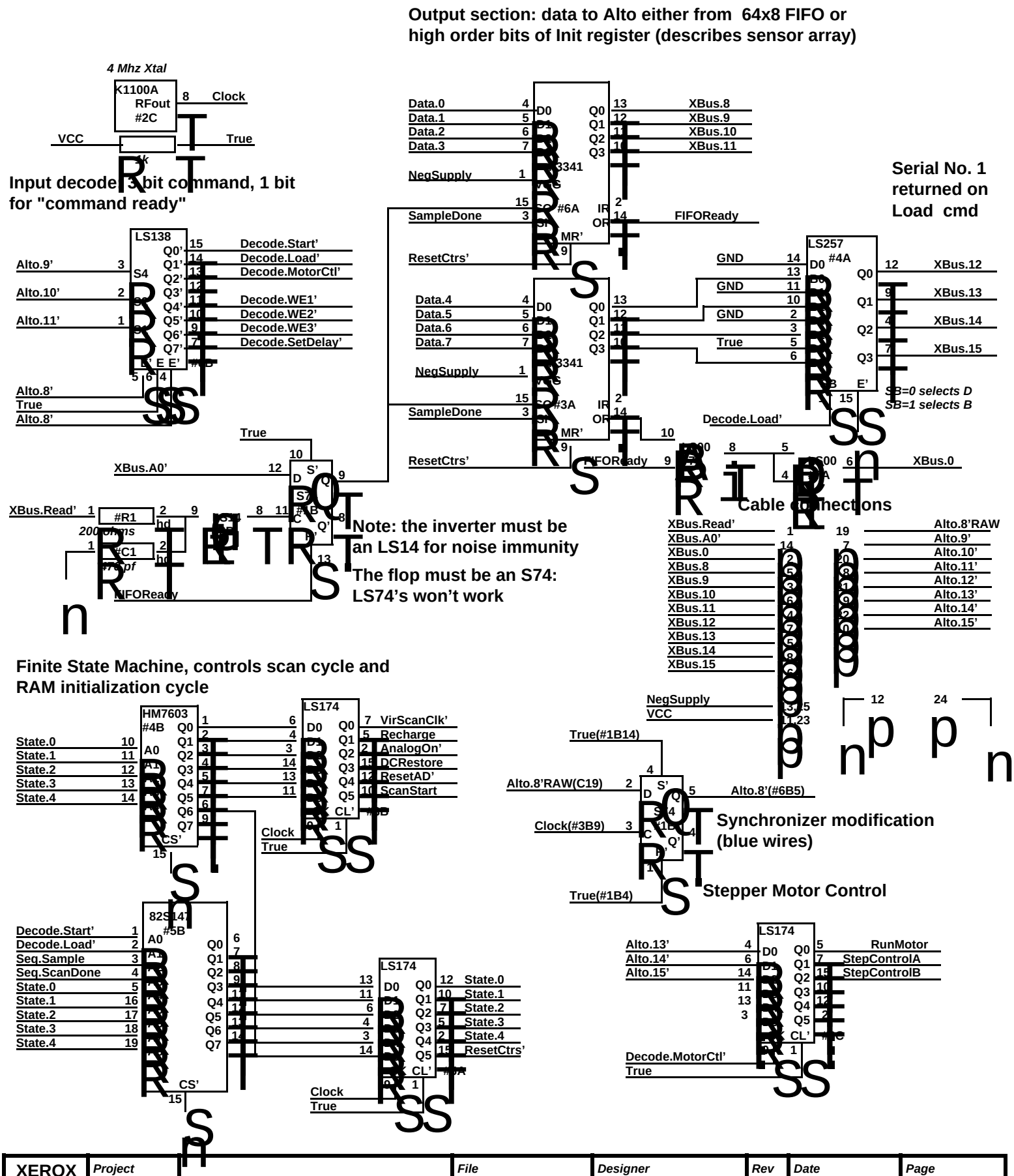
Appendix 1. Schematics

A1.1 Analog Board



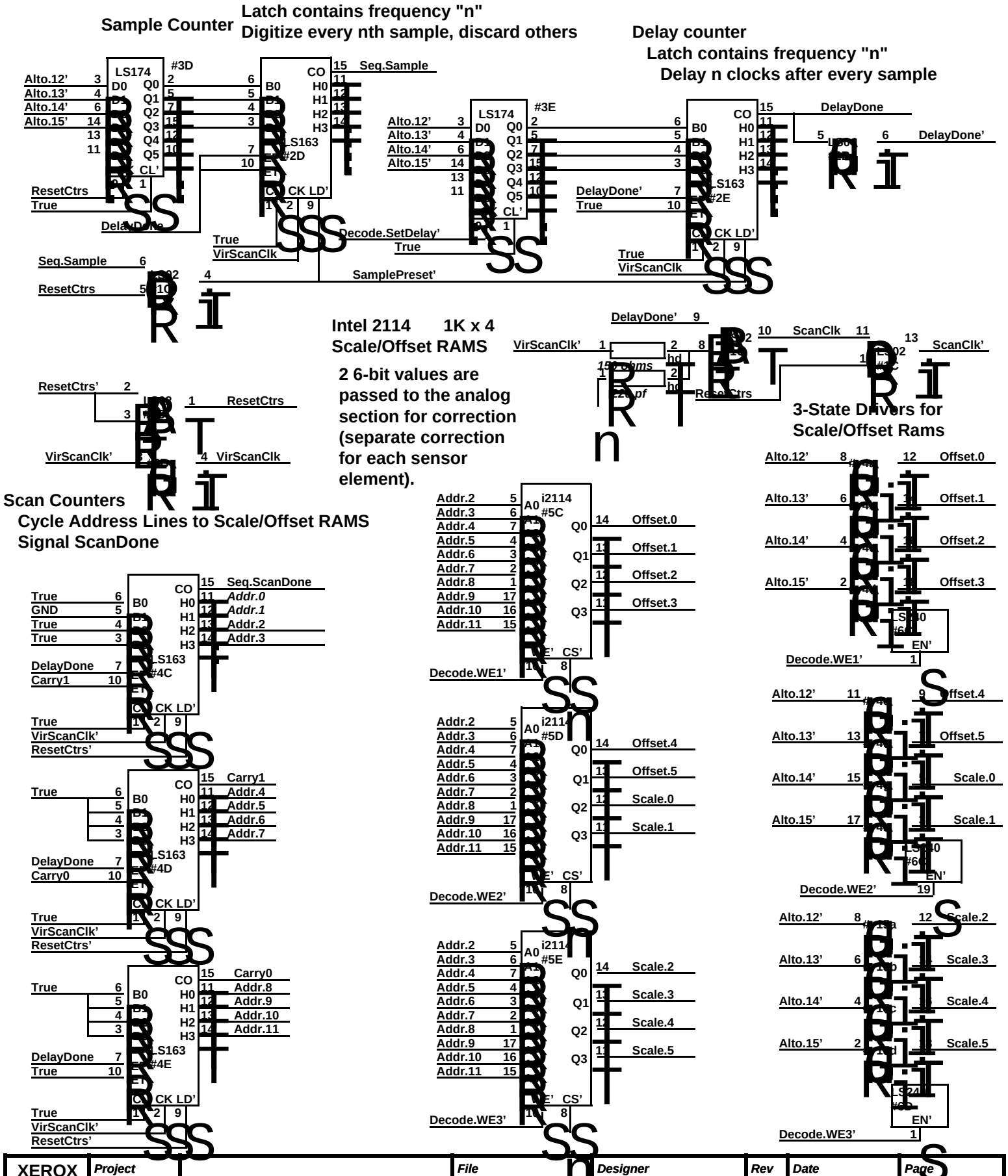
XEROX Parc CSL	Project: Jasmine	Analog section	File: jasmana.sil	Designer: McCreight	Rev: B	Date: 7/19/79	Page: 01

A1.2 Finite State Machine



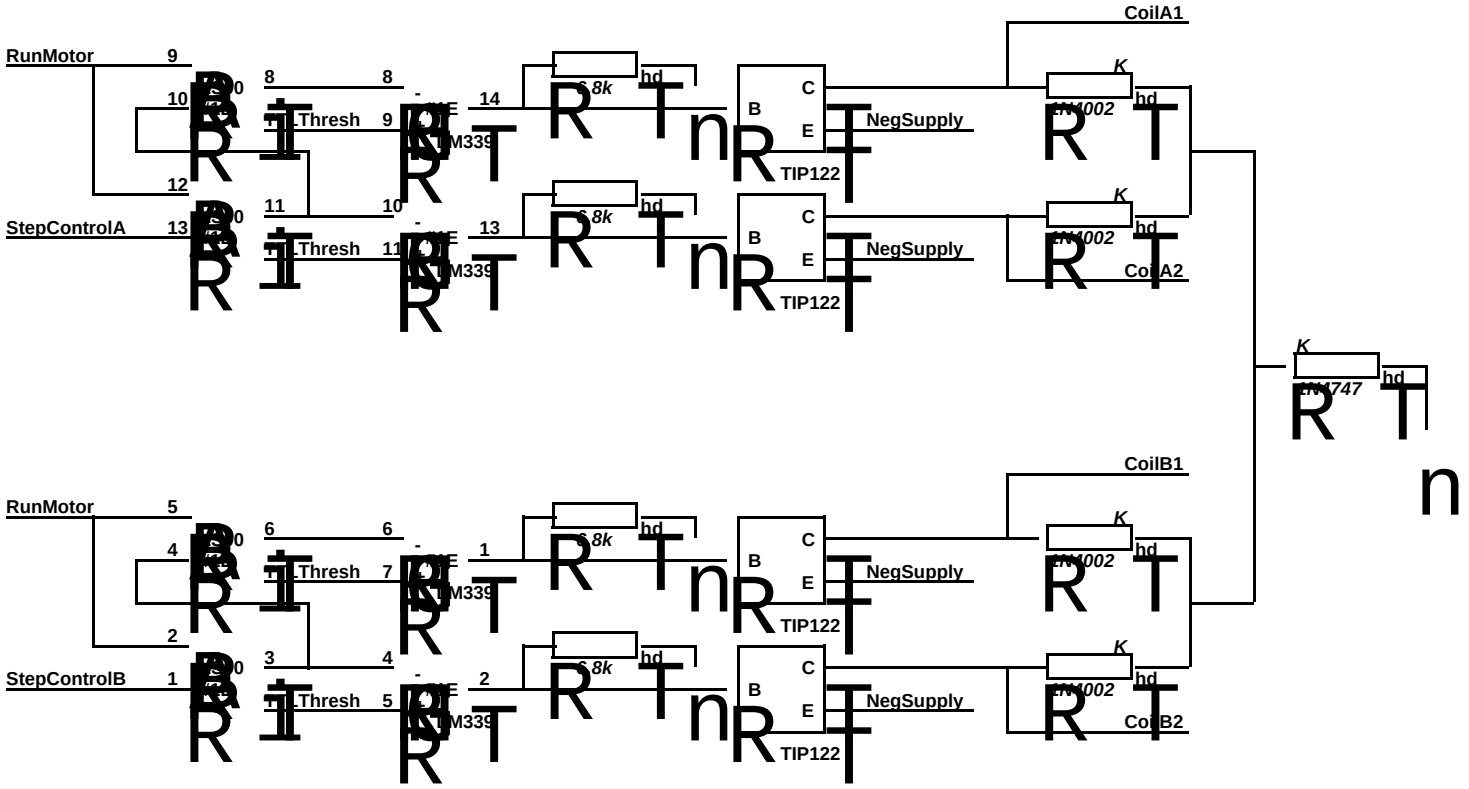
XEROX CSL	Project JASMINE	FINITE STATE MACHINE	File fsm.sil	Designer Maleson	Rev B	Date 7/19/79	Page 02

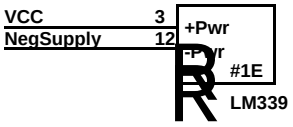
A1.3 Sequencing Logic



XEROX CSL	Project: JASMINE	SEQUENCING LOGIC	File: Sequence.SIL	Designer: Maleson	Rev: B	Date: 7/19/79	Page: 03

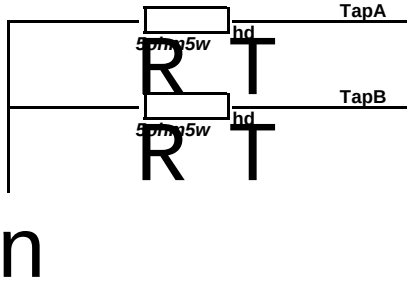
A1.4 Stepper Motor Drivers





Note: for diodes,
K = cathode = banded end

TIP 122 pin order: B,C,E

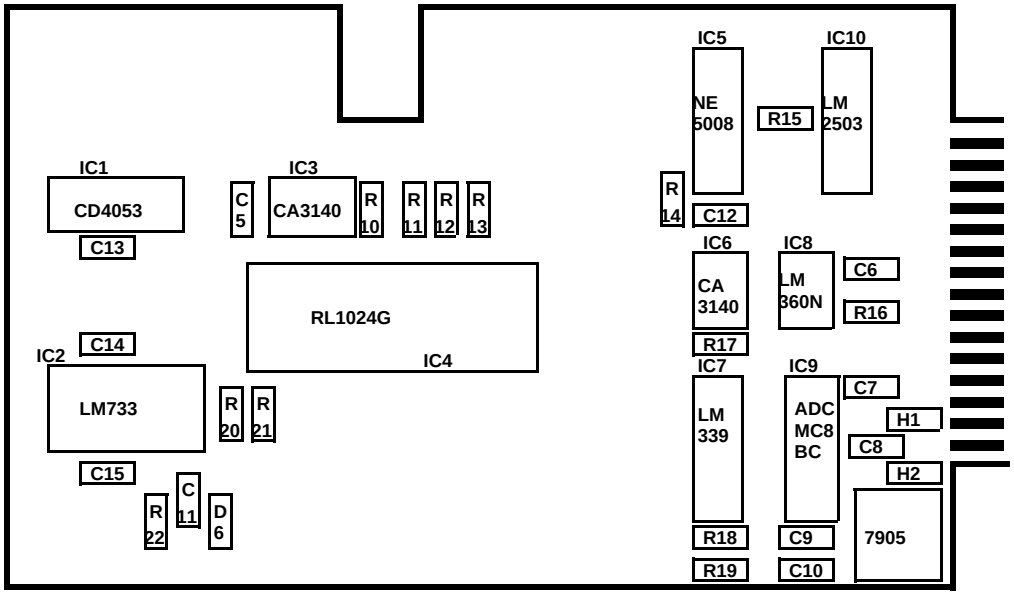


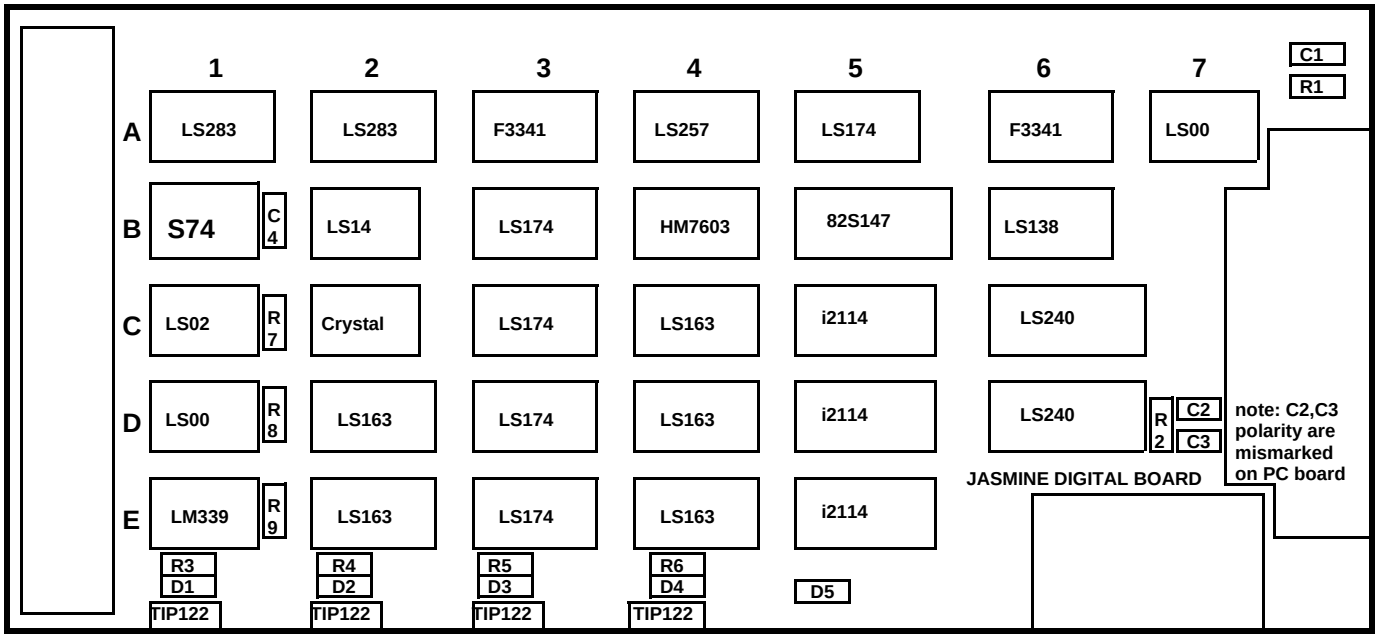
Stepper motor sequencing:
Forward
- ***A On B On***
- ***A On B Off***
- ***A Off B Off***
- ***A Off B On***

Back
- ***A Off B On***
- ***A Off B Off***
- ***A On B Off***
- ***A On B On***

XEROX CSL	Project JASMINE	Stepper Motor Drivers	File Stepper.SIL	Designer Maleson McCreight	Rev A	Date 7/2/79	Page 04

A1.5 PC board layout

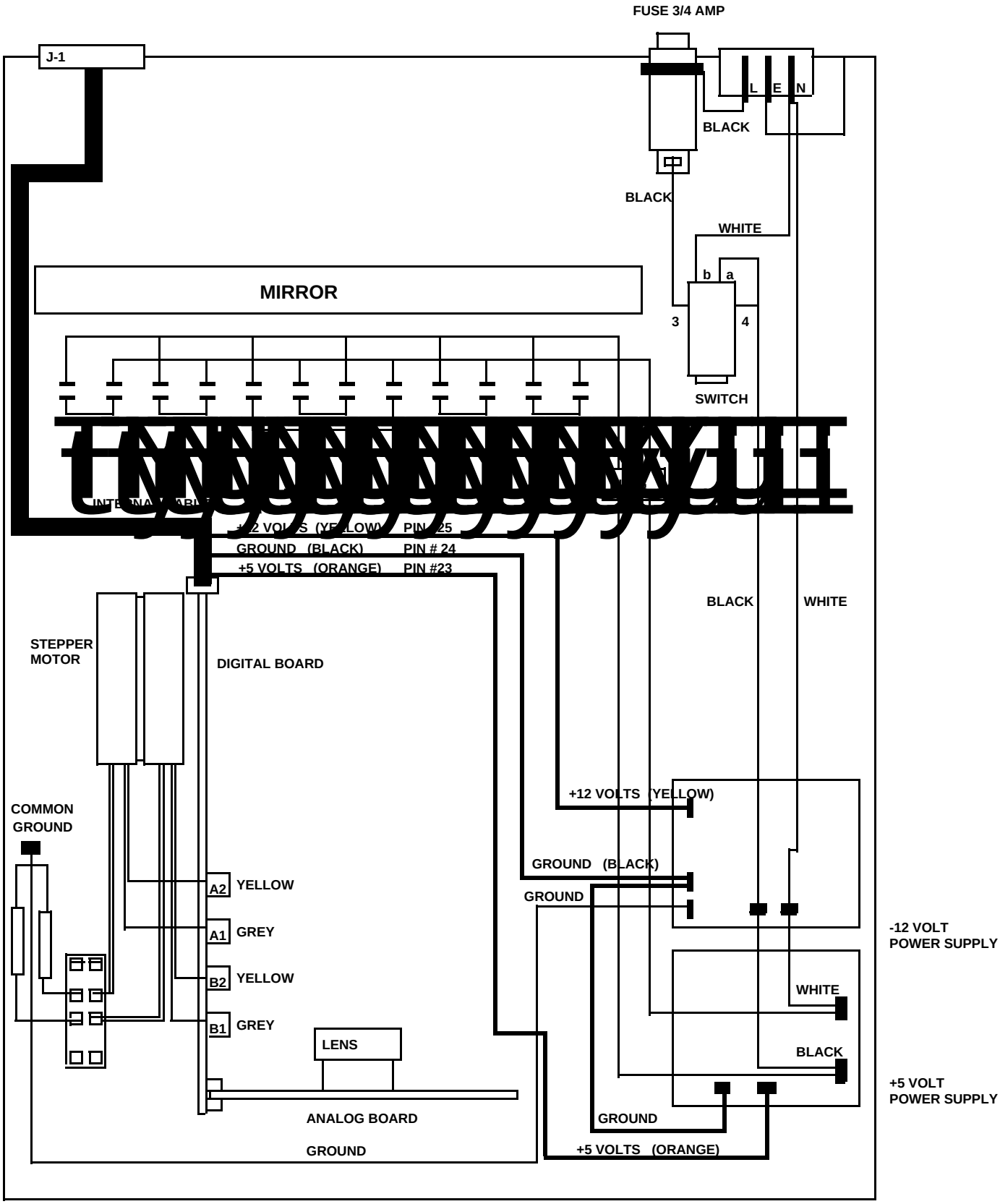




As viewed from the component side

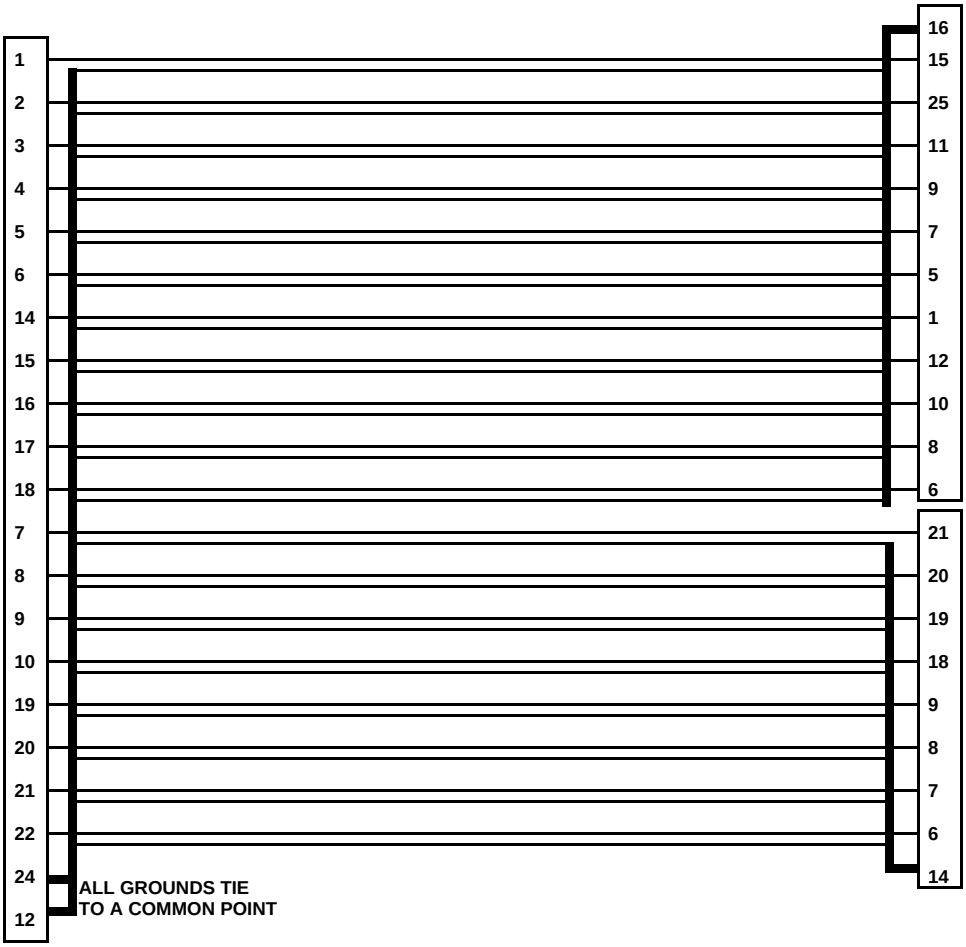
XEROX Parc CSL	Project: Jasmine	Reference: PC Board Layout	File: PCLayout.SIL	Designer: Merino	Rev: A	Date: 9/21/79	Page: 05

A1.6 Power Wiring



XEROX CSL	Project G	SCANNER BASIC WIRING	File <IBIS> SCANNER.SIL	Designer VEST	Rev A	Date 4/8/80	Page 1

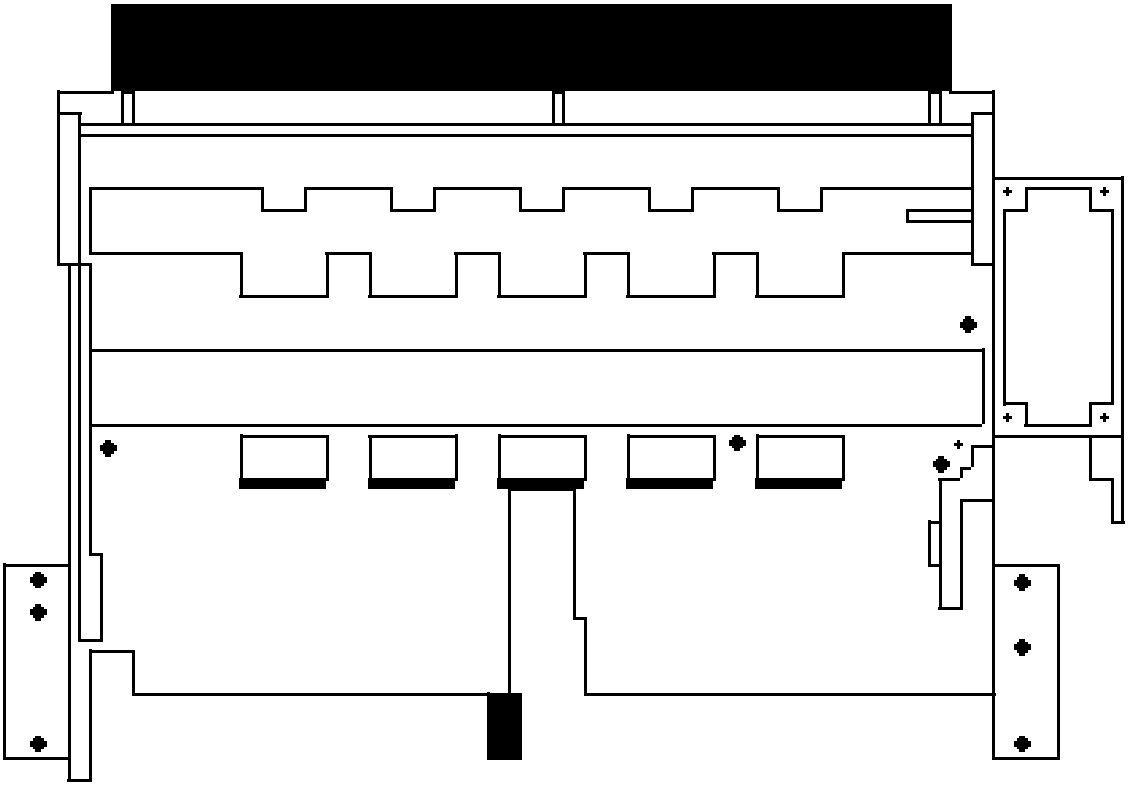
A1.7 External Cable

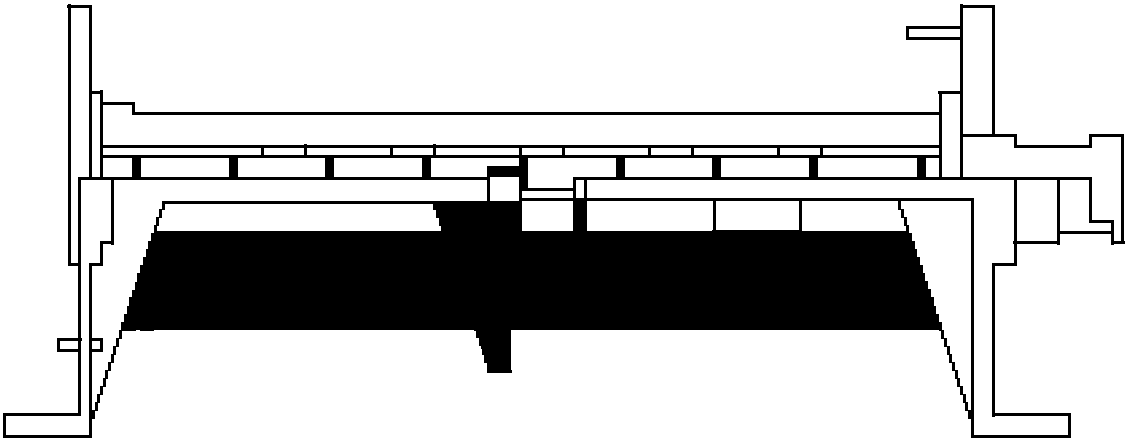


NOTES:
1. CABLE IS 10 FT. LONG
2. CABLE IS 19 TWISTED PAIRS
3. ALL PLUGS ARE AMP. P/N 205208-1

XEROX CSL	Project G	CABLE FOR ALTO 2 SCANNER	File <IBIS> SCANNERCABLE.SIL	Designer VEST	Rev A	Date 4/8/80	Page 1

A1.8 TC200 Paper Transport Modifications





Red areas indicate sections to be modified on TC200 Transport assembly

XEROX *Parc CSL*	*Project* Jasmine	Transport mods	*File* TransportMods.SIL	*Designer* Maleson	*Rev* A	*Date* 1/2/80	*Page* 08

Appendix 2. PROM Microcode

A2.1 PROM Blowing Procedure

Both PROMs are blown with the Prolog PROM blower. The Harris HM7603 32x8 PROM requires the PM9039 Personality Module, the PA16-2 Adaptor, and the 32x8(H) Configurator. The Texas Instruments 74S472 512x8 PROM requires the PM9046 Personality Module, the PA20-1 Adapter, and the 512x8(L) Configurator.

The easist way to blow a set of PROMs is to copy an existing PROM. If you need to blow PROMs from scratch, you will need:

[IVY]<Jasmine>JasProm.MB
[???]<???>PNew.RUN.

The command lines for the 74S472 and HM7603 respectively are:

PNew jasprom.mb/f 22/t StateMem/m 0/c 0/b 0/a 0/s
PNew jasprom.mb/f 7/t ControlMem/m 0,1,2,3,4/p 0/c 0/b 0/a 0/s.

A2.2 Microcode Listing

To produce JasProm.MB from the following source code, you will need:
[MAXC]<Alto>BCPL.RUN
[???]<???>RomDef.RUN.
The command lines necessary are:
BCPL.RUN JasminePromGen.BCPL //produces JasminePromGen.BR
RomDef.RUN JasProm.MB JasminePromGen.BR.

```
//J A S M I N E  P R O M  G E N
//
// generate PROMs for Jasmine scanner

//RomDef jasprom.mb jasminepromgen.br
//pnew jasprom.mb/f 22/t StateMem/m 0/c 0/b 0/a 0/s
//pnew jasprom.mb/f 7/t ControlMem/m 0,1,2,3,4/p 0/c 0/b 0/a 0/s

//12/13/78: first try
//12/21/78: modify control mem to fix inverted signals
//12/28/78: modify state mem for additional recharge cycle
//                  (useless)
//1/25/79: modify control mem to fix initialization sequence
//1/27/79: modify control mem for inverted sample/delay clocks
//                  modify state mem to remove additional cycle on recharge
//4/30/79: fix Load sequence to access first, last locations
//1/24/80: hold DCRestore high while A/D conversion is taking
//                  place because there seems to be charge leaking into the
//                  analog switch when DCRestore is low, even though the
//                  AnalogOn' signal is high (shouldn't happen but does)
//3/10/80: four cycle Recharge
//
//The Alto supplies 8 data bits, and receives 8 data bits + ready flag from Jasmine
//Data bits: 3 bits are decoded into 8 states:
//      {Start',Load',Forward',Back',WE1',WE2',WE3',unused'}
//1 bit enables the decode chip, and 4 bits are data
//      Forward',Back' are stepper motor drive signals
//      WE1',WE2',WE3' are write enables to the three 4x1024 RAMs
//Input bits: two locations are used (both reading from XBUS)
//      one is "SafeLoc" (no state change caused by access)
//      the other is "StepLoc" (reading causes the FIFO to shift out)

//PROM initialization
//The PROM has 4 possible places it may be looping at startup:
//              <Wait>,<Scan>,<Load>,<Init>
//To go from <AnyState> to <Wait>, do:
//              pulse Load', then pulse Start'
//causing the following state transitions
// currentState  Load' 0   Start' 0  Start' 1
//--------------|---------|---------|---------
//   <Wait>     | <Load>  | <Init2> |  <Wait>
//   <Load>     | <Load>  | <Init2> |  <Wait>
//   <Scan>     | <Init1> | <Init2> |  <Wait>
//   <Init1>    | <Init1> | <Init2> |  <Wait>
//   <Init2>    | <Init2> | <Init2> |  <Wait>

//To load the scale/offset ROMs, Pulse LoadPrime to advance the address counters,
//and then each 4-bit nybble is written by the appropriate WE signal and data

//To Scan, pulse StartPrime
//let count=0
//until count eq nSamplesPerScanLine do
// [ data!count=@SafeLoc repeatuntil data!count ls 0
//       [ count=count+1
//        data!count=@StepLoc
//       ] repeatuntil data!count ge 0
// ]
//

//outgoing procedures
external [ RomDef
```

```
                    ]

//incoming procedures
external [ WriteMem
                    ]

//internal manifest and structure declarations
structure
 [ blank                bit 7
       StartPrime       bit
       LoadPrime        bit
       Sample           bit
       ScanDone         bit
       State                bit 5
 ]

structure
 [ blankbyte                byte
       ScanClkPrime         bit
       Recharge             bit
       AnalogOnPrime        bit
       DCRestore                bit
       ResetADPrime   bit
       ResetCtrsPrime       bit
       ScanStart            bit
       blank                    bit
 ]

//manifest names for state numbers
manifest
 [ WAIT=0
       SCAN1=1
       SCAN2=2
       SCAN3=3
       SCAN4=4
       SCAN5=5
       SCAN6=6
       SCANLOOP1=7
       SCANLOOP2=8
       SCANLOOP3=9
       SCANLOOP4=10
       SAMPLE1=11
       SAMPLE2=12
       SAMPLE3=13
       SAMPLE4=14
       SAMPLE5=15
       SAMPLE6=16
       SAMPLE7=17
       SAMPLE8=18
       SAMPLE9=19
       SAMPLE10=20
       SAMPLE11=21
       SAMPLE12=22
       FREESTATE=23
       LOAD1=24
       LOAD2=25
       LOADLOOP1=26
       LOADLOOP2=27
       LOADLOOP3=28
       LOADLOOP4=29
       INIT1=30
       INIT2=31
 ]

let RomDef() be
 [ let S147Mem=vec 512     //Signetics 512x8 PROM
       let HM7603Mem=vec 32    //Harris 32x8 PROM

       //input variables are: StartPrime,LoadPrime,Sample,ScanDone
       for State=0 to 31 do
        for Start=0 to 1 do
         for Load=0 to 1 do
```

```
            for Sample=0 to 1 do
             for ScanDone=0 to 1 do
              [ //set up Signetics PROM address
                    let Saddr=State
                    Saddr<<StartPrime=not Start
                    Saddr<<LoadPrime=not Load
                    Saddr<<Sample=Sample
                    Saddr<<ScanDone=ScanDone

                    //set up default values of output parameters
                    let nextState=State+1
                    let ScanClk=0
                    let Recharge=0
                    let AnalogOn=0 //analog switch enable
                    let DCRestore=0
                    let ResetAD=0
                    let ResetCtrs=0
                    let ScanStart=0
                    switchon State into
                    [ case WAIT: //  standard wait state: branch on Start or Load
                          if Start&Load then
                                [ nextState=WAIT  ;endcase] //NEVER HAPPENS
                          if Start then [  nextState=SCAN1  ;endcase]
                          if Load then [  nextState=LOAD1  ;endcase]
                          nextState=WAIT
                          endcase

//for reading a sample:
//      pulse ScanClk
//      wait 500nsec for detector output
//      hold AnalogOn for 1 us (to transfer charge to hold register)
//      StartAD
// wait 9 clock times for AD to complete, during which
//            Recharge+DCRestore
//            DCRestore+AnalogOn for 1 us (to do DCRestore)
//            DCRestore        (to avoid glitches from analog switch)

                     case SCAN1: //  set up for array start
                          ScanStart=1
                          if Load then  nextState=INIT1
                          endcase
                     case SCAN2: //  array start, (resetctrs for ScanClk)
                          ResetCtrs=1
                          ScanStart=1       //  array starts on falling edge of clock
                          if Load then  nextState=INIT1
                          endcase
                     case SCAN3: //  array start goes away while clock low
                          ScanClk=1         //  counter reset on rising edge of clock
                          ResetCtrs=1
                          if Load then  nextState=INIT1
                          endcase
                     case SCAN4: //  delay for first "real" clock pulse
                          if Load then  nextState=INIT1
                          endcase
                     case SCAN5: if Load then   nextState=INIT1  ;endcase
                     case SCAN6: if Load then   nextState=INIT1  ;endcase
                     case SCANLOOP1: //   until ScanDone do . . .
                          DCRestore=1     //1/24/80 --last cycle of previous A/D
                          ScanClk=1
                          if Load then [  nextState=INIT1  ;endcase]
                          if ScanDone then   nextState=WAIT          //loop exit
                          endcase
                     case SCANLOOP2: //   wait 500 nsec after trailing edge of ScanClk
                          if Load then  nextState=INIT1
                          endcase
                     case SCANLOOP3: //   decide whether to use sample, or skip
                          if Load then [  nextState=INIT1  ;endcase]
                          if Sample then  nextState=SAMPLE1   //  do the sample
                          endcase
                    case SCANLOOP4: //   recharge
                          Recharge=1
                          nextState=SCANLOOP1
```

```
                    if Load then  nextState=INIT1
                    endcase
                case SAMPLE1: //   Grab output voltage in hold register
                    AnalogOn=1
                    if Load then  nextState=INIT1
                    endcase
                case SAMPLE2:
                    AnalogOn=1
                    if Load then  nextState=INIT1
                    endcase
                case SAMPLE3:
                    AnalogOn=1
                    if Load then  nextState=INIT1
                    endcase
                case SAMPLE4:
                    AnalogOn=1
                    if Load then  nextState=INIT1
                    endcase
                case SAMPLE5:       //  reset presented to SAR
                    ResetAD=1
                    if Load then  nextState=INIT1
                    endcase
                case SAMPLE6:       //  Q7 low, Q6-Q0 high
                    Recharge=1
                    DCRestore=1
                    if Load then  nextState=INIT1
                    endcase
                case SAMPLE7:       //  Q7 correct,Q6 low, Q5-Q0 high
                    AnalogOn=1
                    Recharge=1      //3/10/80
                    DCRestore=1
                    if Load then  nextState=INIT1
                    endcase
                case SAMPLE8:       //  Q7-Q6 correct,Q5 low, Q4-Q0 high
                    AnalogOn=1
                    Recharge=1      //3/10/80
                    DCRestore=1
                    if Load then  nextState=INIT1
                    endcase
                case SAMPLE9:       //  Q7-Q5 correct,Q4 low, Q3-Q0 high
                    AnalogOn=1
                    Recharge=1      //3/10/80
                    DCRestore=1
                    if Load then  nextState=INIT1
                    endcase
                case SAMPLE10:      //  Q7-Q4 correct,Q3 low, Q2-Q0 high
                    AnalogOn=1
                    DCRestore=1
                    if Load then  nextState=INIT1
                    endcase
                case SAMPLE11:      //  Q7-Q3 correct,Q2 low, Q1-Q0 high
                    DCRestore=1
                    if Load then  nextState=INIT1
                    endcase
                case SAMPLE12:      //  Q7-Q2 correct, Q1 low, Q0 high
                    DCRestore=1     //1/24/80
                    if Load then  nextState=INIT1
                    nextState=SCANLOOP1         //  finish AD while reading next sample
                    endcase

                case LOAD1: //   load scale/offset RAMS
                    ResetCtrs=1
                    ScanClk=1
                    if Start then  nextState=INIT2
                    endcase
                case LOAD2: //   counter load on clock rising edge
                    ResetCtrs=1
//wait here until Load goes low (to access 0th location) [4/30]
if Load do nextState=LOAD2
                    if Start then  nextState=INIT2
                    endcase
                case LOADLOOP1: //    step counter on every Load pulse
```

```
                        if Start then [  nextState=INIT2  ;endcase]
                        unless Load do  nextState=LOADLOOP1
                        endcase
                    case LOADLOOP2:
                        ScanClk=1
                        if Start then  nextState=INIT2
                        endcase
                    case LOADLOOP3:
                        if Start then [  nextState=INIT2  ;endcase]
                        if Load then  nextState=LOADLOOP3
                        endcase
                    case LOADLOOP4:
                        if Start then [  nextState=INIT2  ;endcase]
//we miss the last location with this here code [4/30]
                        if ScanDone then   nextState=WAIT
                        nextState=LOADLOOP1
                        endcase

                    case INIT1: //   wait for Start
                        if Start then [  nextState=INIT2  ;endcase]
                        nextState=INIT1
                        endcase
                    case INIT2: //   wait for Start to fall
                        if Start then [  nextState=INIT2  ;endcase]
                        nextState=WAIT
                        endcase
                    default:  nextState=WAIT
                        endcase
                  ]
                  let Scommand=0
                  Scommand<<ScanClkPrime=not ScanClk
                  Scommand<<Recharge=Recharge
                  Scommand<<AnalogOnPrime=not AnalogOn
                  Scommand<<DCRestore=DCRestore
                  Scommand<<ResetADPrime=not ResetAD
                  Scommand<<ResetCtrsPrime=not ResetCtrs
                  Scommand<<ScanStart=ScanStart
                  S147Mem!Saddr=nextState
                  HM7603Mem!State=Scommand
              ] //end of nested for loops
      WriteMem(S147Mem,"StateMem",8,511)
      WriteMem(HM7603Mem,"ControlMem",8,31)
 ]
```

Appendix 3. Revisions

A3.1 *REV Gb: modifications to original PC boards*

Analog Board

Connect IC4 pin 21 to 22
Cut trace to IC2 pin 5 (was connected to -5V)
Connect IC2 pin 5 to C14 (GND)
(C14 is now useless with both legs connected to GND)
Add .1uF cap on IC8 from pin 5 to 8
Add .01uF cap on IC5 from pin 1 to 13
Remove resistors R20 and R21
Add 10K pot with adjusting screw at top as follows:
top leg to top hole of R21
middle leg to bottom hole of R21
bottom leg to top hole of R20
(no connection to bottom hole of R20)
initial setting is about 5K
Substitute 1M resistor for R12 (marked 80K)
Substitute 820 ohm resistor for R10 (marked 5.1K)
(NOTE: this resistor is the final gain set. Increasing the resistance will increase the output values)

Digital Board

reverse both 6.8uF caps (pc artwork is backwards)
Cut the etch from Cable connection 19 to #6B pin 5
Add wire from #1B pin 5 to #6B pin 5
Add wire from #1B pin 1 to #1B pin 4
Add wire from #1B pin 4 to #1B pin 14
Add wire from #1B pin 3 to #3B pin 9
Add wire from #1B pin 2 to Cable connection 19

D0 NOTE: The D0 provides TTL termination on its printer port, but the F3341 FIFO is a MOS device. For proper D0 operation, replace the F3341 with the pin-compatible MMI67401 TTL FIFO. The MMI67401 will work on Altos, but costs ten times as much.

NOTE: #1B Must be an S74, not LS as indicated
#6B may want to be an S138, although the twisted pair cable and synchronizer modifications should allow the LS138 to work.

Light Bar

Remove center two bulbs
Add foam tape to front of light bar base, to elevate lights