

rel d.
9/29/91
Phase II
Final

5010-108-01-001
Release to the public

FINAL REPORT

NASA CONTRACT NO. NAS5-30171

IN-74-CR

140697

P. 415

ADVANCED IMAGING SYSTEM

SUBMITTED 31 DECEMBER 1992

ADVANCED TECHNOLOGIES DIVISION

PHOTOMETRICS LTD.

TUCSON, ARIZONA

(NASA-CR-191800) ADVANCED IMAGING
SYSTEM Final Report (Advanced
Technologies) 415 p

N93-16712

Unclas

G3
1/74 0140697

SBIR RIGHTS NOTICE

This SBIR data is furnished with SBIR rights under NASA Contract No. NAS5-30171. For a period of 2 years after acceptance of all items to be delivered under this contract the Government agrees to use this data for Government purposes only, and it shall not be disclosed outside the Government (including disclosure for procurement purposes) during such period without permission of the Contractor, except that, subject to the foregoing use and disclosure prohibitions, such data may be disclosed for use by support contractors. After the aforesaid 2-year period the Government has a royalty-free license to use, and to authorize others to use on its behalf, this data for Government purposes, but is relieved of all disclosure prohibitions and assumes no liability for unauthorized use of this data by third parties. This Notice shall be affixed to any reproductions of this data, in whole or in part.

(End of notice)

Table of Contents

1. Introduction
2. System Hardware Overview
3. Controller Module
 - 3.1 Controller Module Overview
 - 3.2 Connectors and Pinouts
 - 3.3 Controller card
 - 3.3.1 Controller Card Schematic Diagram
 - 3.3.2 Controller Card Logic Equations
 - 3.3.3 Controller Card PCB Artwork
 - 3.4 Sequencer card
 - 3.4.1 Sequencer Card Schematic Diagram
 - 3.4.2 Sequencer Card Logic Equations
 - 3.4.3 Sequencer Card PCB Artwork
 - 3.5 Temperature/ Shutter card
 - 3.5.1 Temperature/Shutter Card Schematic Diagram
 - 3.5.2 Temperature/Shutter Card Logic Equations
 - 3.6 Controller Motherboard
 - 3.6.1 Motherboard Schematic Diagram
 - 3.6.2 Motherboard PCB Artwork
 - 3.7 Fiber-Optic Interface
 - 3.7.1 Fiber Optic Interface Schematic Diagram
 - 3.7.2 Fiber Optic Interface PCB Artwork
4. Clock/Analog Modules
 - 4.1 Module Overview
 - 4.2 Connector Pinouts
 - 4.3 Clock Generator Card
 - 4.3.1 Sequencer Data Interface
 - 4.3.2 CAM Configuration Latch
 - 4.3.3 CCD Voltage Digital to Analog Converters
 - 4.3.4 CCD Clock Switching
 - 4.3.5 Analog Processor Control
 - 4.3.6 Clock Generator Card Schematic Diagram
 - 4.3.7 Clock Generator Card PCB Artwork
 - 4.4 Analog Processor Card
 - 4.4.1 CCD Video Processing Circuitry
 - 4.4.2 Temperature Regulation Circuitry
 - 4.4.3 CCD Clock Signal Buffers
 - 4.4.4 Analog Processor Card Schematic Diagram

- 4.4.5 Analog Processor Card Logic Equations
- 4.4.6 Analog Processor Card PCB Artwork

- 5. Camera Head
 - 5.1 Overview
 - 5.2 Connector Pinouts
 - 5.3 Filter Card
 - 5.3.1 Filter Card Schematic Diagram
 - 5.3.2 Filter Card PCB Artwork
 - 5.4 Socket Card
 - 5.4.1 CCD Socket Card Schematic Diagram
 - 5.4.2 CCD Socket Card PCB Artwork
 - 5.5 Tektronix M745A CCD
- 6. VMEbus interface
 - 6.1 Overview
 - 6.2 Connector Pinouts
 - 6.3 VME200a Data Interface Description
 - 6.4 Fiber-Optic Adapter
 - 6.4.1 Fiber-Optic Adapter Schematic Diagram
 - 6.4.2 Fiber-Optic Adapter PCB Artwork
- 7. Power Supply Unit
- 8. Sun 4/260 Workstation
- 9. System Software Overview
 - 9.1 Embedded Software
 - 9.2 'C' Language Control Library
 - 9.3 UNIX Utility Programs
 - 9.4 IRAF Interface
- 10. IRAF interface
 - 10.1 Starting the Sun 4/260
 - 10.2 Running IRAF
 - 10.3 **ccdacq** Commands
 - 10.4 **ccdacq** Parameters
 - 10.4.1 Detector Parameters
 - 10.4.2 Observing Parameters
- 11. UNIX Utilities
 - 11.1 **AISsay** utility
 - 11.2 **AISfile** utility

- 11.3 *AISsetup* utility
- 12. 'C' Language Control Library
 - 12.1 Function Reference
 - 12.1.1 Interface Functions
 - 12.1.2 High Level Functions
 - 12.1.3 Communications Functions
- 13. Programmable CCD Clock Timing
 - 13.1 Timing Tables
 - 13.1.1 Parallel Clock Timing
 - 13.1.2 Serial Clock Timing
 - 13.1.3 Analog Processor Control
 - 13.1.4 Clock Recombination Anti-blooming
 - 13.2 Filling the Timing Tables
 - 13.3 Timing Table Example
- 14. Programmable CCD voltages
 - 14.1 CCD Clock Rails and DC potentials
 - 14.1.1 CCD parallel Clocks
 - 14.1.2 CCD Serial Clocks and Summing Well
 - 14.1.3 CCD Reset Gate and Last Gate
 - 14.1.4 CCD Output FET Potentials
 - 14.1.5 Other Clocks and Potentials
 - 14.2 CCD Protection Circuitry
 - 14.3 Software Interface for Programmable Voltages
 - 14.3.1 Sequencer Control of Programmable Voltages
 - 14.3.2 FORTH Interface for Programmable Voltages
 - 14.4 Setting the CCD Potentials
 - 14.5 Programmable Voltage Example File
- 15. Camera Configuration Parameters
 - 15.1 Format Parameters
 - 15.1.1 Serial Read Parameters
 - 15.1.2 Parallel Read Parameters
 - 15.2 Exposure Parameters
 - 15.3 Acquisition Sequence Parameters
- 16. Camera Controller FORTH Program
 - 16.1 FORTH Command Interpreter
 - 16.2 FORTH Command Descriptions
 - 16.2.1 Short Form Commands
 - 16.2.2 High Level Image Acquisition Commands
 - 16.2.3 Low Level Image Acquisition Commands

- 16.2.4 Shutter Control
- 16.2.5 Temperature Measurement and Control
- 16.2.6 Format Commands
- 16.2.7 CCD Voltages
- 16.2.8 CCD Timing
- 16.2.9 Camera Readout Speed
- 16.2.10 Continuous Clearing of the CCD
- 16.2.11 Clock Recombination Anti-blooming
- 16.2.12 CCD Image Integration
- 16.2.13 Time Measurement
- 16.2.14 System Maintenance Commands
- 16.2.15 DSP Software Support
- 16.2.16 Test Functions
- 16.3 Creating Custom FORTH Definitions
- 16.4 Saving and Restoring the FORTH Dictionary

17. DSP56001 Sequencer Software

- 17.1 68HC11 Host Interface
- 17.2 DSP Functions
- 17.3 DSP Parameters
- 17.4 Creating Custom DSP56001 Software

18. System performance

- Appendix A: Camera Controller FORTH Source Code
- Appendix B: Camera Controller DSP56001 Source Code
- Appendix C: VMEbus Interface Board Source Code
- Appendix D: 'C' Language Interface
- Appendix E: UNIX Utility Source Code

1. Document Overview

This document describes the Advanced Imaging System CCD based camera. The AIS1 camera system was developed at Photometrics Ltd. in Tucson, Arizona as part of a Phase 2 SBIR contract No. NAS5-30171 from the NASA/Goddard Spaceflight Center in Greenbelt, Maryland. The camera project was undertaken as a part of the Space Telescope Imaging Spectrograph (STIS) project.

This document is intended to serve as a complete manual for the use and maintenance of the camera system. All the different parts of the camera hardware and software are discussed and complete schematics and source code listings are provided.

2. System Overview

The Advanced Imaging System (AIS) is a slow scan, high precision CCD imaging system designed specifically for low noise image acquisition and precise, highly flexible CCD testing and characterization. In addition, the system is designed to allow CCD mosaics to be supported with separate, programmable clock voltages and output amplifier operating points for each device.

2.1 Introduction

The development of new CCD technology has been proceeding at a rapid rate in recent years. Large format devices of 2K x 2K and 4K x 4K resolution with multiple low readout noise amplifiers, flexible operating modes (such as MPP inverted operation and clocked recombination anti-blooming) are now available with even more new devices in development. New output amplifier architectures such as the non-destructive readout floating gate "skipper" amplifiers have proven feasible. Butted arrays of multiple large format devices are now being developed at several institutions. These innovations have rendered current CCD cameras obsolete and developed a demand for a new generation camera system that will capitalize on the advances in CCD technology.

The Advanced Imaging System (AIS) was developed to support all known large format CCD arrays and mosaics under development, and allow provision to future expansion. Several goals were identified for the system including:

1. The system should be sufficiently flexible to operate virtually any CCD imager, and multiple output CCDs and CCD mosaics, so the user could change CCDs with minimal hardware modifications.
2. The system signal/noise should be limited either by the characteristics of the CCD or photon shot noise and the digitization performed to the highest accuracy available with current analog to digital converter technology.
3. The clock voltages, timing, and other system parameters should be alterable through software in order to optimize CCD performance for a given set of experimental conditions.

4. The image acquisition computer system should be capable of accommodating images of multi-megabyte size.
5. The camera head should be able to be separated from the host computer by long distances.

In response to the rapidly changing technologies and techniques associated with scientific CCD imaging, system flexibility, programmability, and adaptability have been emphasized.

Flexibility is a crucial feature of imaging systems which are intended for use into the next century. CCD imaging systems designed today need to be flexible enough to operate a variety of CCD devices in a wide variety of operating modes. Sufficient clock signals must be provided to operate different CCD architectures. Operating point voltages need to be controllable over a wide enough range to support different types of output amplifier structures.

Systems which are intended to be useful over the long term must be flexible enough to deliver virtually any desired waveforms to the image sensor clock inputs, and must be ready to measure the data thus produced as quickly and accurately as possible. Modularity of system hardware allows for integration of new technologies as they become available without complete reconfiguration of the system. This is especially important in the areas of analog processing, clock sequence generation, and analog to digital conversion.

Programmability implies the ability of the system's operation to be controlled not only through built-in commands, but through the creation of new commands and command sequences which may be referred to indirectly. By eliminating overhead in the command interface, and by reducing the amount of time spent transmitting commands and the chance of communication corruption caused by outside noise sources.

Programmability allows a system to be more efficiently used in the collection of data. This reduces the amount of time spent transmitting commands and the chance of communication corruption caused by outside noise sources. It also aids in the elimination of redundant initialization tasks.

An adaptable system is able to be permanently or semi-permanently modified to better meet the requirements of the experiment in progress. A CCD imaging system, in order to be considered adaptable, must be capable of "remembering" user defined commands, CCD clock and operating point voltages, and

desired timing sequences.. Use of an EEPROM memory, which may be programmed in-circuit, allows the system state to be stored at any time. Default parameters and command sequences may be retained from session to session, and may be modified at any time by the user.

2.2 System Hardware

The system was designed in such a way as to allow the cost of the system, at least to some extent, to be scaled with the flexibility required by the end user, without sacrificing performance. Modular construction allows a particular system configuration to be established for each application, without eliminating any of the more advanced features of the system. Single port camera systems are supported with a set of five printed circuit boards. Additional ports on the same device may be supported by the addition of one more board per port. Additional ports on other devices requires one board to support the clock voltages required for the additional CCD and one board for each CCD output port used. The system software supports up to eight CCD devices and an unlimited number of output ports.

The amount of hardware required for the AIS1 camera system depends on the application, and system configuration. The minimum requirement is as follows:

- 1 System Controller, performing system control tasks such as communication with the host computer/user, CCD temperature measurement and control, and shutter control.
- 1 DSP Sequencer, generates the CCD clock voltages and timing waveforms.
- 1 Clock Generator, containing the programmable CCD voltage references, and switching circuitry,.
- 1 Analog Processor, containing signal processing chains, and CCD clock buffers.
- 1 CCD socket card and preamplifier, located in the cryostat.

Multiple ports on the CCD may be operated by adding one analog processor for each additional output port desired. If different clock or CCD output FET operating voltages are desired for the

different ports, or if additional CCD imagers are added, additional clock generator cards are required.

2.3 STIS AIS1 Camera Configuration

This specific camera system, the STIS camera system consists of the following hardware:

- 1 Camera controller, comprised of three circuit boards. There is a controller board, a sequencer board, and a temperature and shutter control board. Each of these is described in more detail below.
- 2 Clock generators, one for each of the two ports being used on the CCD in this system. Duplication of the clock generators allows maximum optimization of the operating points of the two output amplifiers.
- 2 Analog processors, each a 16 bit Dual slope integrator operating at pixel rates of 50 kHz and below.
- 1 Filter board, located in the cryostat. The filter board is used to filter the DC voltages being applied to the CCD as well as serving as the location of the reset and summing well switches.
- 1 Socket card holding the CCD and the first stage preamplifier circuitry.
- 1 Tektronix M745A CCD.

3. Controller Module

3.1 Controller Module Overview

The AIS1 controller module contains the digital electronics that communicate with the host computer, in this case the Sun workstation, and control the CCD and analog processor to read the data off the imager. The controller module also contains the circuitry that is used to measure and control the temperature of the CCD and the circuitry required to drive the shutter solenoid.

There are five circuit cards inside the module. Three of these cards are the embedded controller electronics consisting of microprocessor based "controller" and "sequencer" cards, and a "temperature/shutter control" card. These cards are connected to one another via a simple "motherboard" and the connectors on the module's chassis are connected to them through a "fiber optic interface" card.

The camera controller module is normally bolted to the side of the camera head. The controller module may be remotely located from the camera head if that is necessary. This requires the use of some longer cables to connect it to the Clock/Analog Modules and the camera head.

Below, the camera control module's external connectors are described, and the boards are then examined individually. Complete schematics and programmable logic device equation listings are provided for each of the boards.

3.2 Connectors and Pinouts

There are several connectors on the camera controller module. These are used either to connect the module to the host computer or to connect the controller module to the clock/analog modules. The connectors will be discussed below.

The Connector labeled "Camera Control Source" connects to the clock/analog modules. It carries the signals from the DSP sequencer which control the clock and analog cards. These signals are generated by the DSP at a maximum rate of 10 MHz. They are differentially driven on the cable, but over limited distance. The data sheets for the drivers use indicate that these signals could be driven as far as 100 meters. We have successfully tested 25 meters but have not tried longer distances.

Camera Control Source Connector Pinout : DB37M

pin	function	pin	function	pin	function
1	WR+	17	D0+	33	unused
2	WR-	18	D0-	34	unused
3	A0+	19	D1+	35	unused
4	A0-	20	D1-	36	unused
5	A1+	21	D2+	37	unused
6	A1-	22	D2-		
7	A2+	23	D3+		
8	A2-	24	D3-		
9	A3+	25	D4+		
10	A3-	26	D4-		
11	A4+	27	D5+		
12	A4-	28	D5-		
13	A5+	29	D6+		
14	A5-	30	D6-		
15	extra+	31	D7+		
16	extra-	32	D7-		

The connector labeled "RS422" may be used to connect an RS422 terminal to the camera controller. It is also used to connect the camera controller to the VMEbus interface. The nature of the contents of these transmissions is described in the camera system software manual.

RS422 Communications Connector Pinout: DB9M

pin	name	function
1	TX-	transmit data -
2	TX+	transmit data +
3	RX-	receive data-
4	RX+	receive data +
5	GND	controller ground
6	unused	
7	unused	
8	unused	
9	unused	

There are two 'ST' type fiber optic connectors on the controller module that are used when the camera controller is connected to the host computer via a fiber optic line. The optical fibers may be used instead of the RS422 option. Switching from one to the other requires moving a jumper on the fiber optic interface card

The connector labeled "power" is used to provide the DC voltages necessary to operate the controller electronics. The power is normally provided by connecting this connector to the power supply module with the cables provided. These cables are approximately 3 meters in length. The power supply voltages provided must be regulated, as no regulation takes place inside the module.

Controller Module Power Connector : DB15M

pin	voltage	pin	voltage
1	-15V	9	GND
2	-15V	10	GND
3	+5V	11	GND
4	+5V	12	GND
5	+15V	13	GND
6	+15V	14	GND
7	+28V	15	GND
8	+28V		

3.3 Controller Card

The camera system controller is based around a version of the Motorola MC68HC11 microcontroller. Produced by Motorola for New Micros Inc., the F68HC11FN microcontroller contains a FORTH interpreter in the internal ROM. All camera operation is coded in a mixture of FORTH and assembler code. System software may be extended by the user, and modifications may be retained from session to session. See the AIS1 camera system software manual for details concerning the FORTH software.

The controller board is a modified version of a camera controller board we designed, based on the AIS1 prototype, for a somewhat different camera. Therefore, some of the parts of the controller board are not populated, and are irrelevant to this discussion.

The central processing unit is U1, the 68HC11. The 68HC11 contains a variety of on chip peripherals which are used in this system. The 8k ROM contains the Max-FORTH interpreter. The 256

bytes of SRAM are used by the FORTH system. The on chip EEPROM is used for interrupt vectors. The timer subsystem is used as a free running millisecond timer. The Serial Communications Interface (SCI) is used for communication with the host computer. Various port bits are used to control other circuits such as the shutter driver. The Serial Peripheral Interface (SPI) is unused.

The 68HC11 is clocked by U4, an 8 MHz crystal oscillator. The 68HC11's instruction cycle when clocked at this frequency is 2 MHz.

The 68HC11's multiplexed address/data bus is demultiplexed by U3, a 74HC573 transparent latch.

U5 is a DS1231 from Dallas Semiconductor which provides a clean reset pulse to the 68HC11 during power up and power down.

U6 is a DS8921 IC from National Semiconductor. This IC translates the TTL level SCI signals from the 68HC11 into the RS422 level signals connected to the RS422 connector on the outside of the camera controller module chassis.

U8 is a 32k x 8 bit static RAM device used to hold the FORTH dictionary while the 68HC11 is running. U15, shown on the schematic as a 32k x 8 bit EPROM is actually a 28C256 32k x 8 EEPROM device which may be programmed in the circuit. This EEPROM holds the default FORTH dictionary. When the system is reset, the contents of this EEPROM are copied into the SRAM. The contents of the SRAM may be copied into the EEPROM at any time via the FORTH command set.

U16 is a 74HC574 latch which may be written to by the 68HC11 to control external hardware. At this time U16 and the outputs located on P3 of the controller card are uncommitted, and not brought out through the controller module chassis.

U7 is a programmable logic device (PLD) used to decode the addresses generated by the 68HC11 and produce the chip select signals needed by the various peripherals. The listing for this PLD is provided after the schematics at the end of this section.

U18 is not installed. Before modifying the board to accept the 28C256 EEPROM, the SRAM was supplied with backup current from a battery when the system was not powered up. This IC managed the SRAM's power.

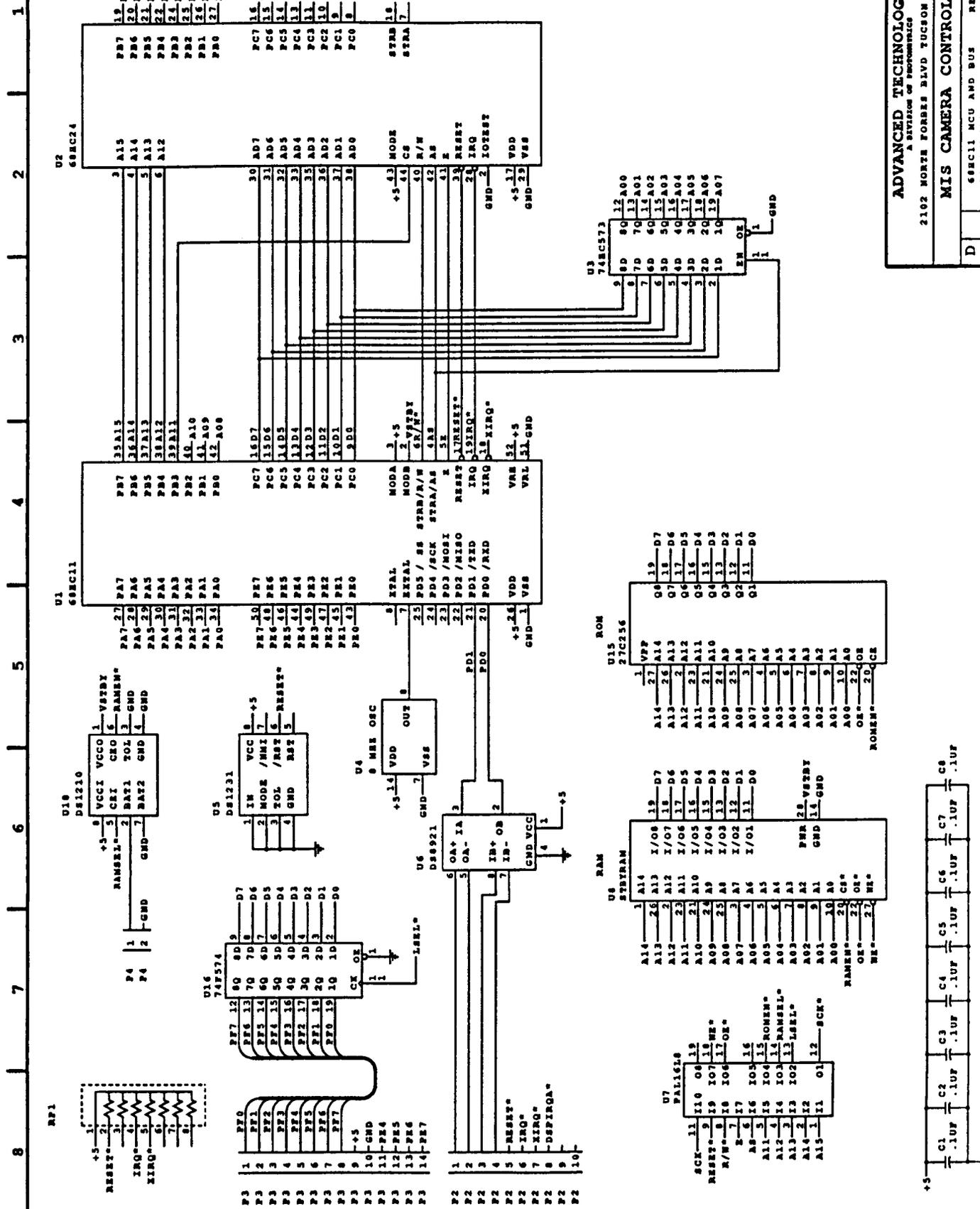
U2, a 68HC24 "port replacement unit" from Motorola is not installed. The card was designed for use in a camera controller using a DSP located on this card as sequencer. U2 was used by the 68HC11 to communicate with that DSP. U9 was the DSP sequencer in that camera. It is not installed. U10, U11, U12, U13, and U17 were associated with that sequencer and are not installed. This camera has a separate sequencer card, and there is a DSP56001 on it instead.

The 68HC11's address and data bus, along with several I/O port lines, are connected to P1, the motherboard connector. These lines are then fed across the motherboard to the other cards in the system. The connector is shown on page 3 of the controller card schematic. The relevant lines are :

D0 - D7	The 68HC11's data bus
A08 - A15	The 68HC11's address bus
OE*	A qualified 'read' signal from the 68HC11
WE*	a qualified 'write' signal from the 68HC11
AS	the 68HC11's address strobe
E	the 68HC11's instruction clock
RESET*	the system reset, produced by U5
PA0 - PA7	68HC11 port A
PE0 - PE3	4 bits from 68HC11 port E

The controller card uses power only from the +5V supply, and returns current only to the digital ground, named 'gnd' on the schematics. There is no +15V or -15V supply current used on this card.

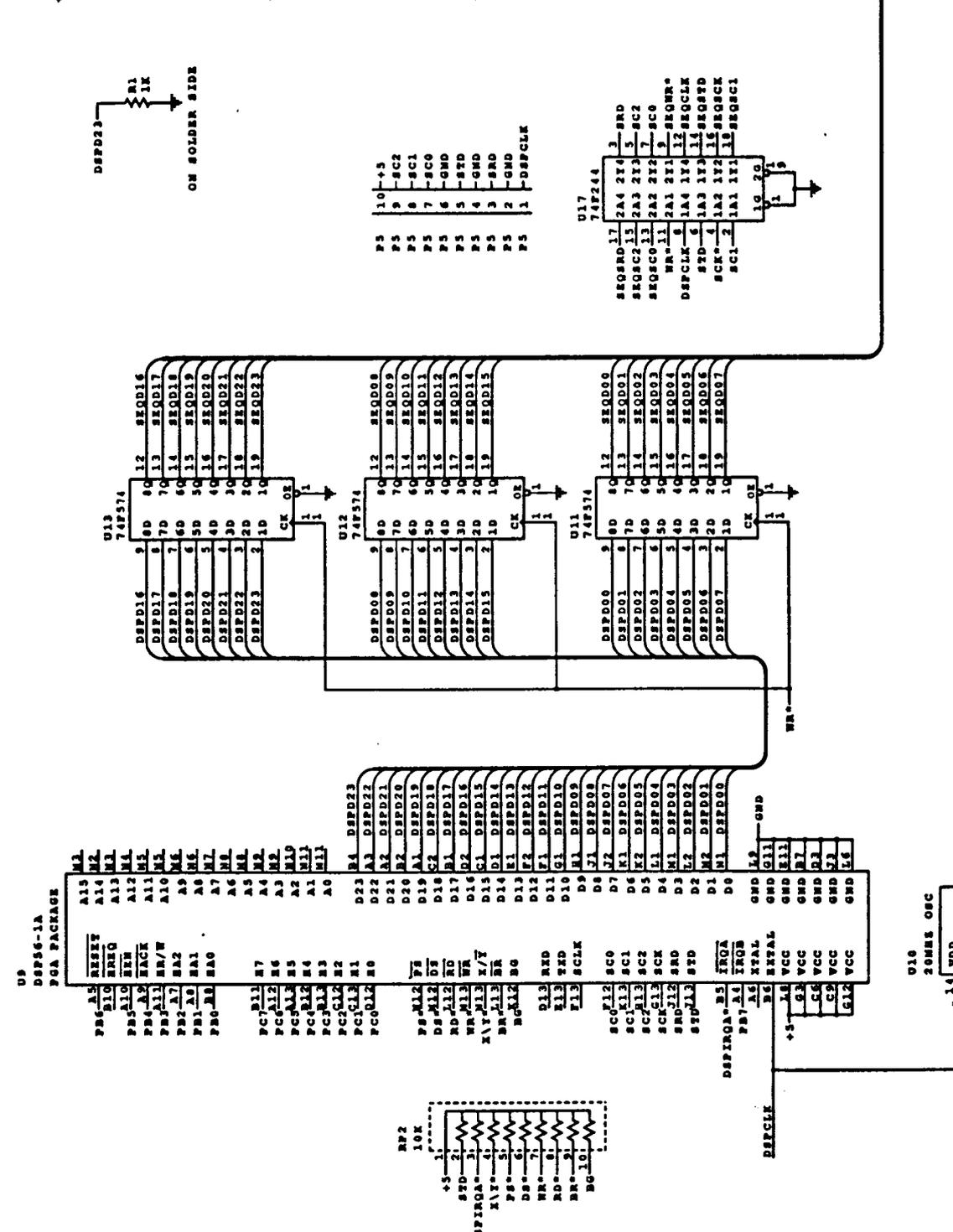
3.3.1 Controller Card Schematic Diagram



1 2 3 4 5 6 7 8

D C B A

8 7 6 5 4 3 2 1



1 2 3 4 5 6 7 8

SEQSC0 C1 P1
 SEQSC1 C2 P1
 SEQSC2 C3 P1
 SEQSC3 C4 P1
 SEQSC4 C5 P1
 SEQSC5 C6 P1
 SEQSC6 C7 P1
 SEQSC7 C8 P1
 SEQSC8 C9 P1
 SEQSC9 C10 P1
 SEQSC10 C11 P1
 SEQSC11 C12 P1
 SEQSC12 C13 P1
 SEQSC13 C14 P1
 SEQSC14 C15 P1
 SEQSC15 C16 P1
 SEQSC16 C17 P1
 SEQSC17 C18 P1
 SEQSC18 C19 P1
 SEQSC19 C20 P1
 SEQSC20 C21 P1
 SEQSC21 C22 P1
 SEQSC22 C23 P1
 SEQSC23 C24 P1
 SEQSC24 C25 P1
 SEQSC25 C26 P1
 SEQSC26 C27 P1
 SEQSC27 C28 P1
 SEQSC28 C29 P1
 SEQSC29 C30 P1
 SEQSC30 C31 P1
 SEQSC31 C32 P1

SEQD00 B1 P1
 SEQD01 B2 P1
 SEQD02 B3 P1
 SEQD03 B4 P1
 SEQD04 B5 P1
 SEQD05 B6 P1
 SEQD06 B7 P1
 SEQD07 B8 P1
 SEQD08 B9 P1
 SEQD09 B10 P1
 SEQD10 B11 P1
 SEQD11 B12 P1
 SEQD12 B13 P1
 SEQD13 B14 P1
 SEQD14 B15 P1
 SEQD15 B16 P1
 SEQD16 B17 P1
 SEQD17 B18 P1
 SEQD18 B19 P1
 SEQD19 B20 P1
 SEQD20 B21 P1
 SEQD21 B22 P1
 SEQD22 B23 P1
 SEQD23 B24 P1

SEQD12 A1 P1
 SEQD13 A2 P1
 SEQD14 A3 P1
 SEQD15 A4 P1
 SEQD16 A5 P1
 SEQD17 A6 P1
 SEQD18 A7 P1
 SEQD19 A8 P1
 SEQD20 A9 P1
 SEQD21 A10 P1
 SEQD22 A11 P1
 SEQD23 A12 P1
 SEQD24 A13 P1
 SEQD25 A14 P1
 SEQD26 A15 P1
 SEQD27 A16 P1
 SEQD28 A17 P1
 SEQD29 A18 P1
 SEQD30 A19 P1
 SEQD31 A20 P1
 SEQD32 A21 P1
 SEQD33 A22 P1
 SEQD34 A23 P1
 SEQD35 A24 P1
 SEQD36 A25 P1
 SEQD37 A26 P1
 SEQD38 A27 P1
 SEQD39 A28 P1
 SEQD40 A29 P1
 SEQD41 A30 P1
 SEQD42 A31 P1
 SEQD43 A32 P1

3.3.2 Controller Card Logic Equations

```
Name      AISDCS;
Partno    none assigned;
Date      1/24/91;
Revision  01;
Designer  Doherty;
Company   Advanced Technologies;
Assembly  new AISCTL address select;
Location  U?;
Office    G16V8;
```

```
/*
/* This file contains the source code for the AIS controller */
/* address decode PAL. */
/* */
/* */
/* */
/* Revision history : */
/* Rev. 01 1/25/91 */
/* */
/* */
/*
/* Target Device = Lattice GAL16V8 as PAL16L8 */
/*
/*
/*
*/
```

```
/* inputs */
Pin 1 = a15; /* 6811 address 15 */
Pin 2 = a14; /* 6811 address 14 */
Pin 3 = a13; /* 6811 address 13 */
Pin 4 = a12; /* 6811 address 12 */
Pin 5 = a11; /* 6811 address 11 */
Pin 6 = as; /* 6811 address strobe */
Pin 7 = eclk; /* 6811 E clock */
Pin 8 = rd; /* 6811 RD/WR* strobe */
Pin 9 = !reset; /* system reset */

/* outputs */
Pin 13 = !latch; /* output latch select */
Pin 14 = !ram; /* chip select for RAM */
Pin 15 = !eeprom; /* chip select for EEPROM */
Pin 16 = memdis; /* memory disable; used internally */
Pin 17 = !oe; /* system output enable */
Pin 18 = !we; /* system write enable */
```

```
/* addr is the address inputs as a five bit word */
FIELD addr = {a15..11};

/* RAM will go from $0000 to $5fff for a total of 24k */
ram = addr;0000..5FFF; & !memdis & !reset;

/* EEPROM will go from $6000 to $DFFF for 32k with some holes */
eeprom = addr;6000..DFFF; & !memdis & !reset;

/* the output latch will go from $B800 to $B8FF */
latch = addr;B800..B8FF; & !memdis & !reset;

memdis = addr;B000..BFFF; /* internal 68HC11 I/O registers */
```

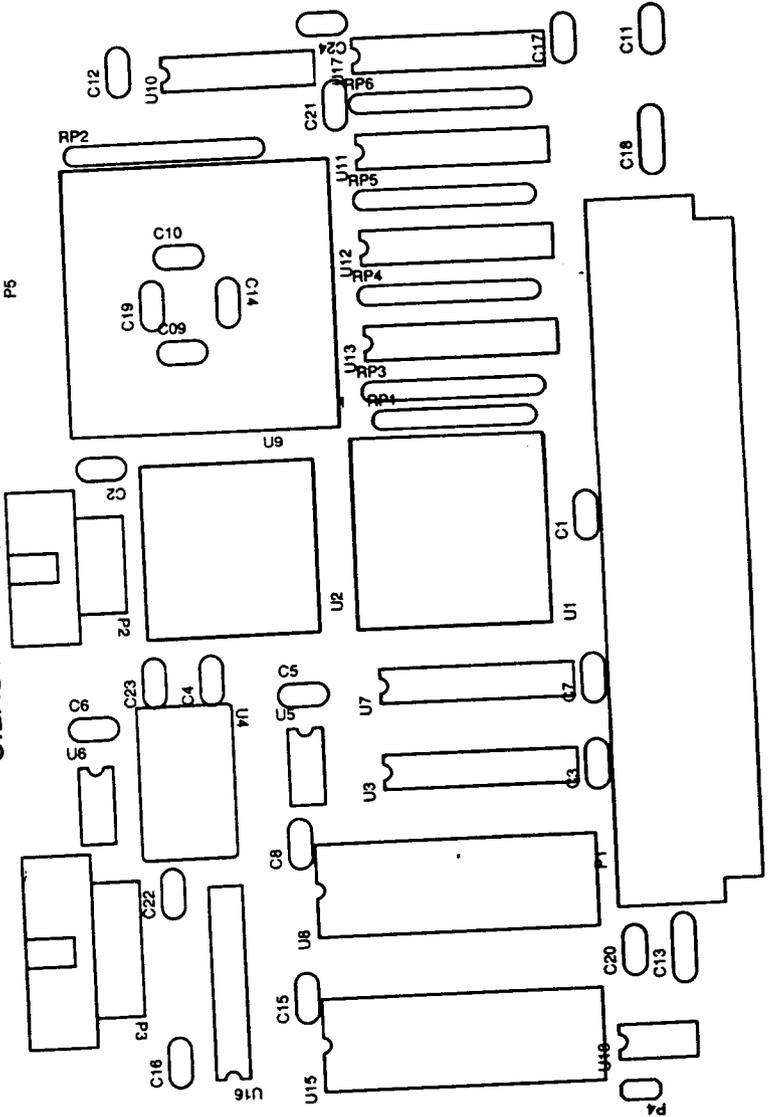
```
P addr:;C000..C7FF; /* DSP location */
P addr:;C300..CFFF; /* temp/shutter board location */
P addr:;E000..FFFF; /* internal FORTH in ROM */
P reset ;
```

```
/* output enable for various devices */
oe = rd & eclk & !reset ;
```

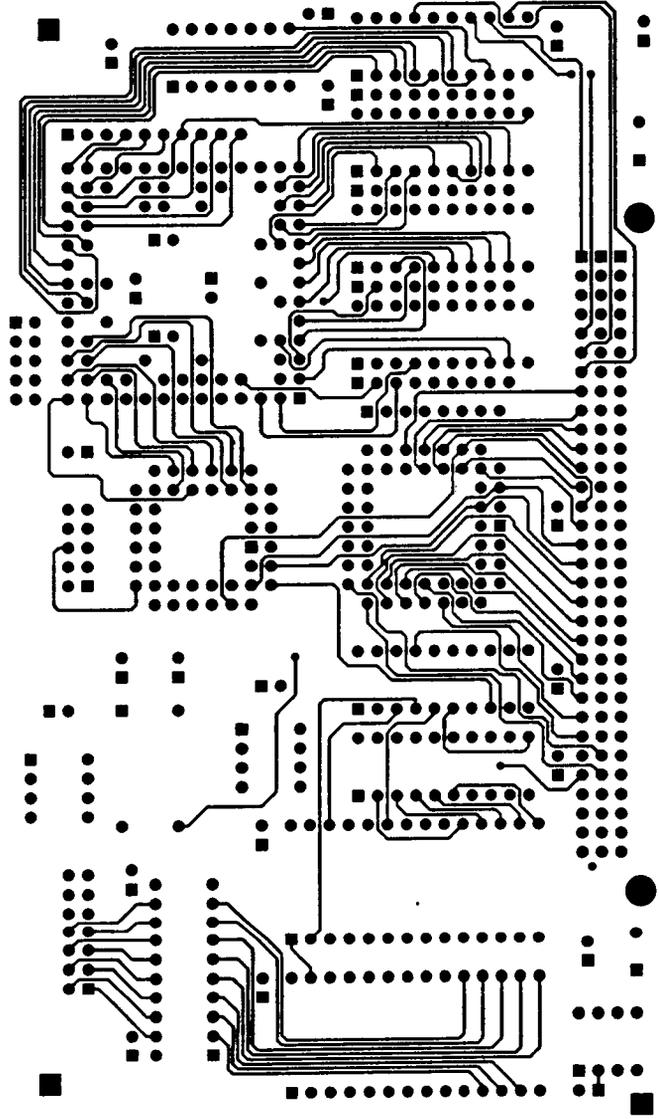
```
/* write enable for writable devices */
we = !rd & eclk & !reset ;
```

3.3.3 Controller Card PCB Artwork

SILKSCREEN (TOP ONLY)



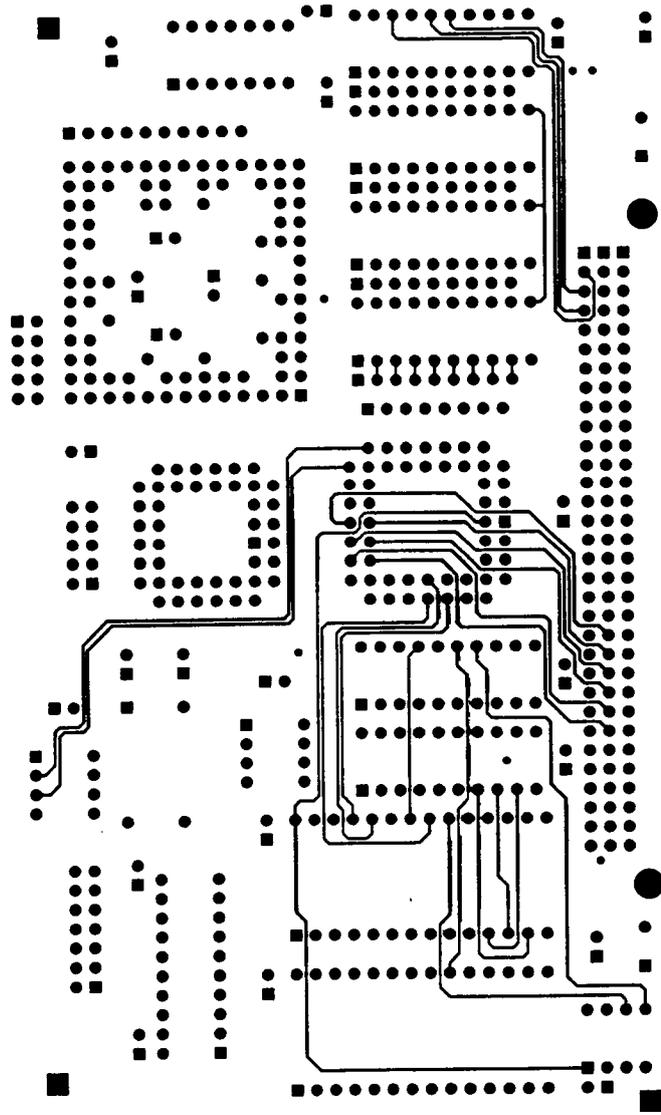
COMPONENT SIDE



• ■ •

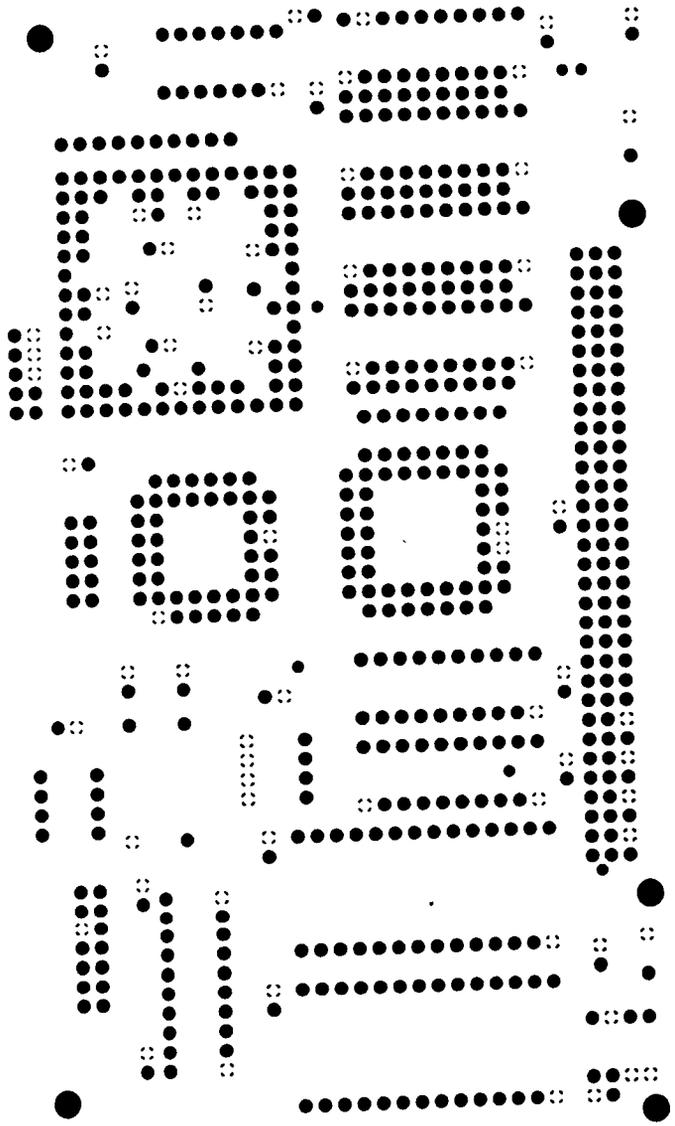


MID LAYER 2



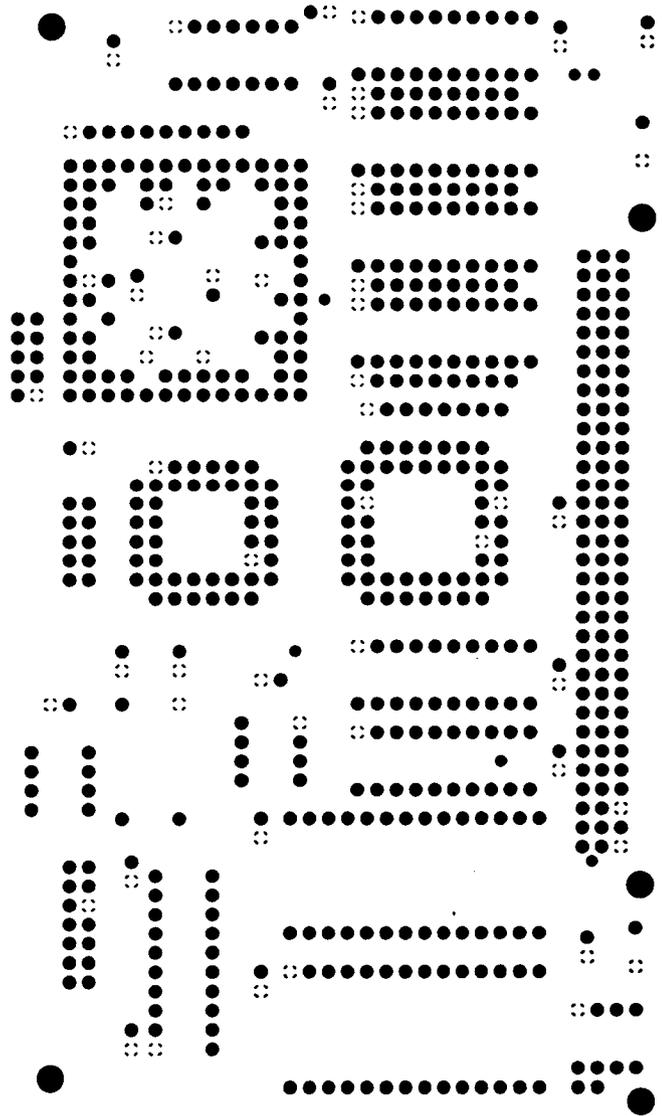


GROUND PLANE



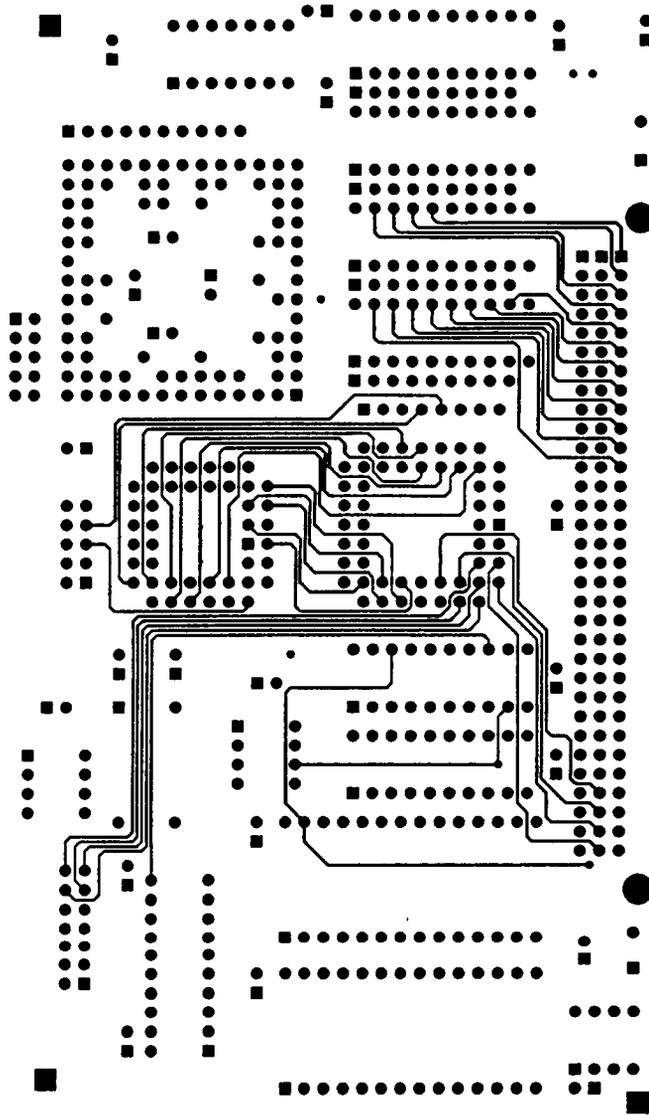


POWER PLANE

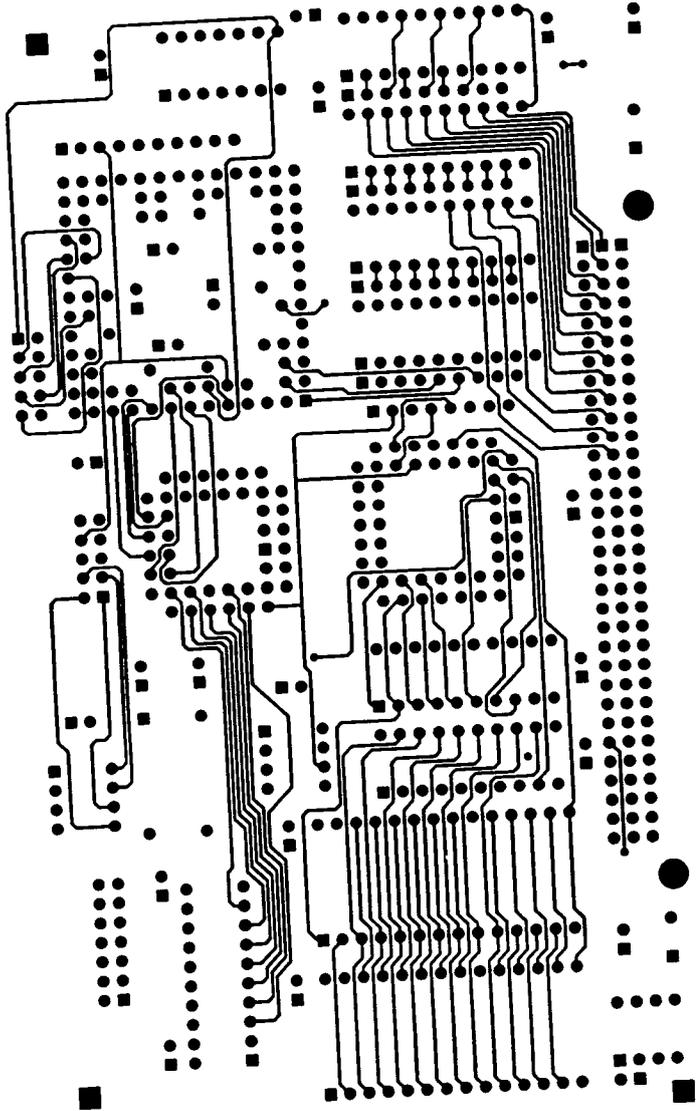




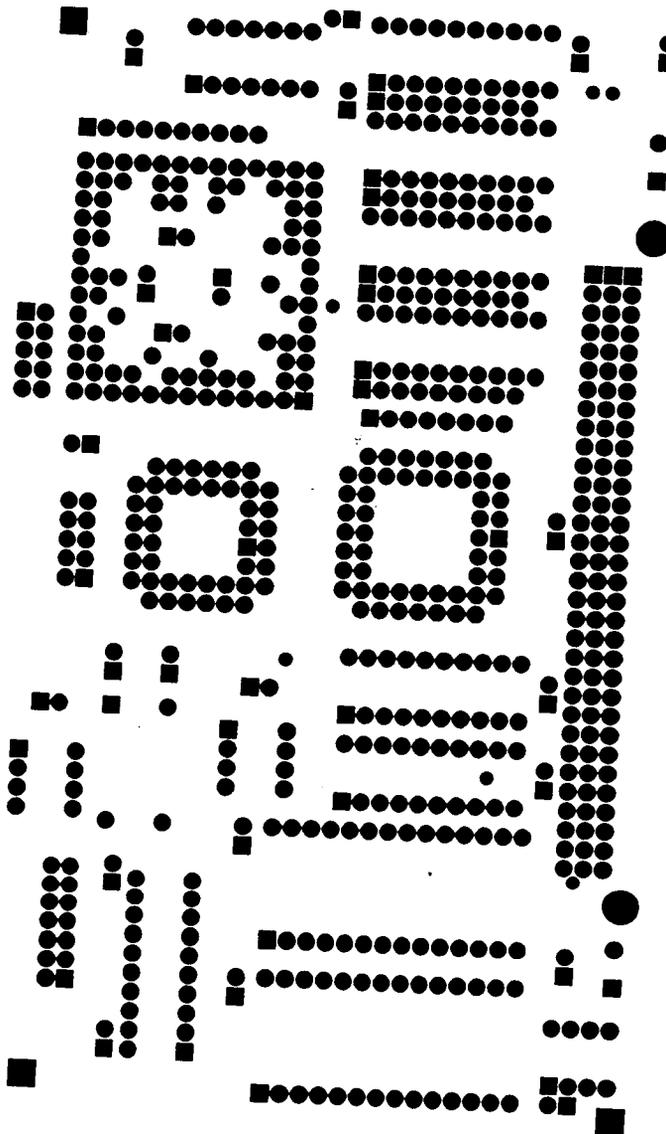
MID LAYER 1



SOLDER SIDE



SOLDER MASK



3.4 Sequencer Card

The sequencer card generates all the timing necessary to operate the CCD and the analog processor. The sequencer card includes a Motorola DSP56001 digital signal processor (DSP). The DSP operates with a 100 ns instruction cycle. The DSP bootstraps its program code from the 68HC11 through the host interface.

A portion of the DSP's address and data busses is differentially driven on a short cable to the clock generator. Eight bits of data and six bits of address information, as well as a write strobe, are provided. The DSP performs CCD control by writing to memory mapped devices on the clock generator. Up to eight unique clock generators may be addressed. Any subset of the eight clock generators may be accessed simultaneously. The DSP may write to the clock generator locations at its full execution rate of 10 MHz.

The heart of the sequencer card is U1, the DSP56001, it produces all the signals required to operate the CCD and analog processor by writing data into registers on the clock cards. It does so under command from the 68HC11 on the controller card.

The DSP includes a host interface port, through which commands and data may be transferred. This byte wide bi-directional interface is tied to the 68HC11's address data bus through the motherboard. The eight registers of the DSP host interface appear to the 68HC11 as memory locations at addresses \$C000 to \$C007. The details of the operation of the host interface are beyond the scope of this document. See chapter 10 of the DSP56001 user's guide for more information. Since the 68HC11's address/data bus is multiplexed, U5, a 74HC573 transparent latch, is used to latch the low order address bits. The 68HC11 controls the operating state of the DSP through U4, a 74HC574 latch, via which it may reset the DSP and issue interrupts. This latch appears in the 68HC11's memory space at address \$C100. A programmable logic device, U3, a PAL16L8, is used to decode the 68HC11's address bus and generate the chip selects for the DSP interface and the DSP control latch.

The DSP has three memory spaces. These are referred to as 'program' memory, 'x data' memory, and 'y data' memory. The DSP program resides in program memory of course. At startup the DSP has no program to execute. The DSP is then reset by the 68HC11 in a bootstrap mode in which it loads its internal program memory with words transferred through the host interface. Additionally, the DSP has access to an external memory device, U2, a MCM56824 8k x 24 bit static RAM from Motorola designed specifically for the DSP56001. The DSP also loads this memory through the host interface.

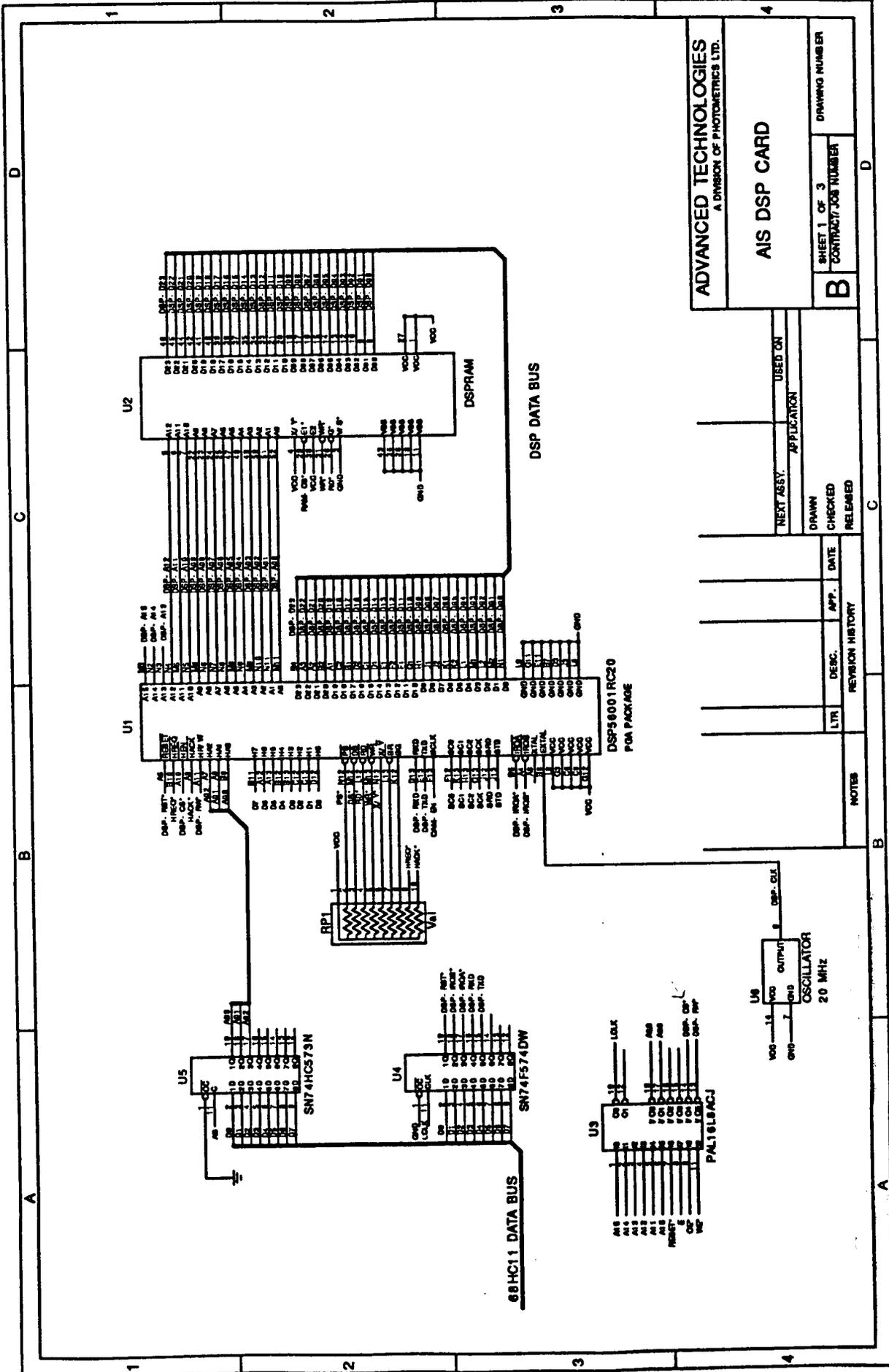
The DSP's address bus is decoded by U8, a PAL16L8-10 from Texas Instruments. Chip selects are generated for the DSP's various peripherals. U2, the external SRAM, is one of these. Additionally, there are two 16 bit data ports. One of these, composed of P2 U9, and U10 is a 16 bit wide input port. The other, composed of P3, U11 and U12, is a 16 bit output port. These ports are not currently used in the system and the connections are not brought to the outside of the controller module chassis.

The DSP's lowest 8 data bits and lowest 7 address bits are connected to a set of differential drivers, U13, U14, U15, U16. These differential drivers are of the 41LG type manufactured by AT&T Microelectronics. They produce a "pseudo ECL" output which may be driven at high speeds over relatively long distances. The output pairs from these drivers are tied to a PC board mounted male DB37 connector. The pin out of this connector is shown below. Note that this is the same pinout as for the DB37 on the controller module chassis. There is a one to one ribbon cable connecting the two.

Sequencer Card P4 Connector Pinout:

pin	function	pin	function	pin	function
1	WR+	17	D0+	33	unused
2	WR-	18	D0-	34	unused
3	A0+	19	D1+	35	unused
4	A0-	20	D1-	36	unused
5	A1+	21	D2+	37	unused
6	A1-	22	D2-		
7	A2+	23	D3+		
8	A2-	24	D3-		
9	A3+	25	D4+		
10	A3-	26	D4-		
11	A4+	27	D5+		
12	A4-	28	D5-		
13	A5+	29	D6+		
14	A5-	30	D6-		
15	A6+	31	D7+		
16	A6-	32	D7-		

3.4.1 Sequencer Card Schematic Diagram



ADVANCED TECHNOLOGIES
A DIVISION OF PHOTOMETRICS LTD.

AIS DSP CARD

SHEET 1 OF 3
CONTRACT/JOB NUMBER

B

USED ON APPLICATION

DRAWN CHECKED RELEASED

REVISION HISTORY

LTR	DESC.	APP.	DATE

NOTES

D

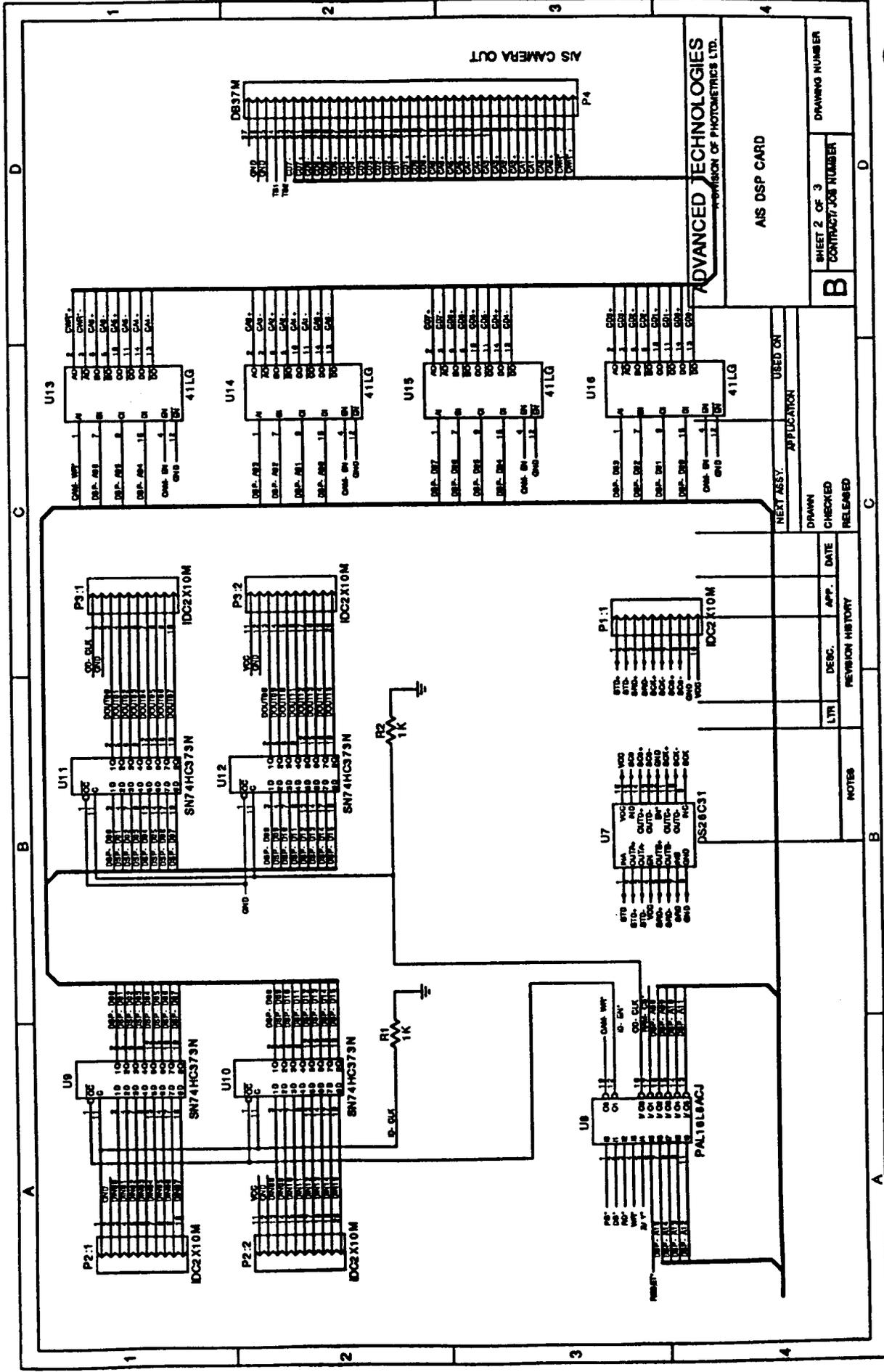
C

B

A

1 2 3 4

A B C D



ADVANCED TECHNOLOGIES
DIVISION OF PHOTOMETRICS LTD.

AIS DSP CARD

SHEET 2 OF 3
CONTRACT JOB NUMBER

DRAWN
CHECKED
RELEASED

REVISION HISTORY

LTR	DESC.	APP.	DATE

NOTES

APPLICATION USED ON

DRAWING NUMBER

A	B	C	D																				
1	2	3	4																				
			<p>ADVANCED TECHNOLOGIES A DIVISION OF PHOTOMETRICS LTD.</p> <p>AIS DSP CARD</p> <p>SHEET 3 OF 3 CONTRACT/JOB NUMBER</p> <p>B</p>																				
<p>NOTES</p>			<p>APPLICATION</p> <p>USED ON</p>																				
<p>REVISION HISTORY</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 10%;">LTR</th> <th style="width: 40%;">DESC.</th> <th style="width: 10%;">APP.</th> <th style="width: 10%;">DATE</th> </tr> </thead> <tbody> <tr> <td> </td> <td> </td> <td> </td> <td> </td> </tr> <tr> <td> </td> <td> </td> <td> </td> <td> </td> </tr> <tr> <td> </td> <td> </td> <td> </td> <td> </td> </tr> <tr> <td> </td> <td> </td> <td> </td> <td> </td> </tr> </tbody> </table>			LTR	DESC.	APP.	DATE																	<p>DRAWN</p> <p>CHECKED</p> <p>RELEASED</p>
LTR	DESC.	APP.	DATE																				
<p> </p>			<p>DRAWING NUMBER</p>																				
A	B	C	D																				
1	2	3	4																				

3.4.2 Sequencer Card Logic Equations

```

Name          DSPIO;
Partno        none assigned;
Date          4/2/91;
Revision      01;
Designer      Doherty;
Company       Advanced Technologies;
Assembly      DSP card DSP address decoder;
Location      U8;
Device        P16L8;

```

```

/*****/
/*
/* DSP address decoder
/*
/*
/*
/*
/* Revision history :
/*
/* Rev. 01 4/2/91
/*
/*
/*****/
/*
/* Target Device = 16L8
/*
/*****/

```

```

/* inputs */
Pin 1 = !ps;          /* program memory select */
Pin 2 = !ds;          /* data memory select */
Pin 3 = !rd;          /* dsp read signal */
Pin 4 = !wr;          /* dsp write signal */
Pin 5 = xdata;        /* x data memory select */
Pin 6 = !reset;       /* system reset */
Pin 7 = a15;          /* DSP address 15 */
Pin 8 = a14;          /* DSP address 14 */
Pin 9 = a13;          /* DSP address 13 */
Pin 11 = a12;         /* DSP address 12 */
Pin 13 = a11;         /* DSP address 11 */
Pin 14 = a10;         /* DSP address 10 */
Pin 15 = a09;         /* DSP address 09 */
Pin 16 = a08;         /* DSP address 8 */

```

```

/* outputs */
Pin 17 = !ram_cs;     /* DSP RAM chip select */
Pin 18 = !od_clk;     /* output data clock */
Pin 12 = !id_en;      /* input data enable */
Pin 19 = !cam_wr;     /* qualified write strobe for camera */

```

```

/* addr is the address inputs as a five bit word */
FIELD addr = {a15..a08};

```

```

/* the DSP's external RAM is at P:$0000 to P:$1FFF */
/* for a total of 8K */

```

```

ram_cs = addr;0000..1FFF; & ps & !reset;

```

```

/* output data latch is write only at y:$0000 to y:$00FF */
od_clk = addr;0000..00FF; & ds & !xdata & wr & !reset;

```

```
/* input data latch is read only at y:S0000 to y:$FFFF */  
id_en = addr:;0000..00FF& & ds & !xdata & rd & !reset;
```

```
/* cam_wr is a qualified version of the DSP's write pulse */  
/* the camera is write only at Y:$FF00 to Y:$FFFF */  
am_wr = addr:;FF00..FFFF& & ds & !xdata & wr & !reset;
```

```
Name      DSPDCD;
Partno    none assigned;
Date      1/24/91;
Revision  01;
Designer  Doherty;
Company   Advanced Technologies;
Assembly  new AISDSP address select;
Location  U3;
Device    G16V8;
```

```
/* ***** */
/* This file contains the source code for the AIS controller */
/* address decode PAL. */
/* */
/* */
/* */
/* */
/* Revision history : */
/* Rev. 01 1/25/91 */
/* */
/* ***** */
/* Target Device = Lattice GAL16V8 as PAL16L8 */
/* ***** */
```

```
/* inputs */
```

```
Pin 1 = a15; /* 6811 address 15 */
Pin 2 = a14; /* 6811 address 14 */
Pin 3 = a13; /* 6811 address 13 */
Pin 4 = a12; /* 6811 address 12 */
Pin 5 = a11; /* 6811 address 11 */
Pin 6 = a10; /* 6811 address 10 */
Pin 18 = a09; /* 6811 address 09 */
Pin 17 = a08; /* 6811 address 08 */
Pin 7 = !reset; /* system reset */
Pin 8 = eclk; /* 6811 E clock */
Pin 9 = !oe; /* system output enable */
Pin 11 = !we; /* system write enable */
```

```
/* outputs */
```

```
Pin 13 = !dsp_wr; /* DSP host port write signal */
Pin 14 = !dsp_cs; /* DSP host port chip select */
Pin 19 = !dsp_ctl; /* DSP control latch select */
```

```
/* addr is the address inputs as a five bit word */
```

```
FIELD addr = ;a15..a08;
```

```
/* The DSP's actual address range is $C000 to $C7FF */
```

```
dsp_cs = ( addr;C000..C0FF; & we & eclk & !reset ) /* write cycle */
P ( addr;C000..C0FF; & !reset ); /* read cycle */
```

```
!dsp_wr = dsp_cs & we;
```

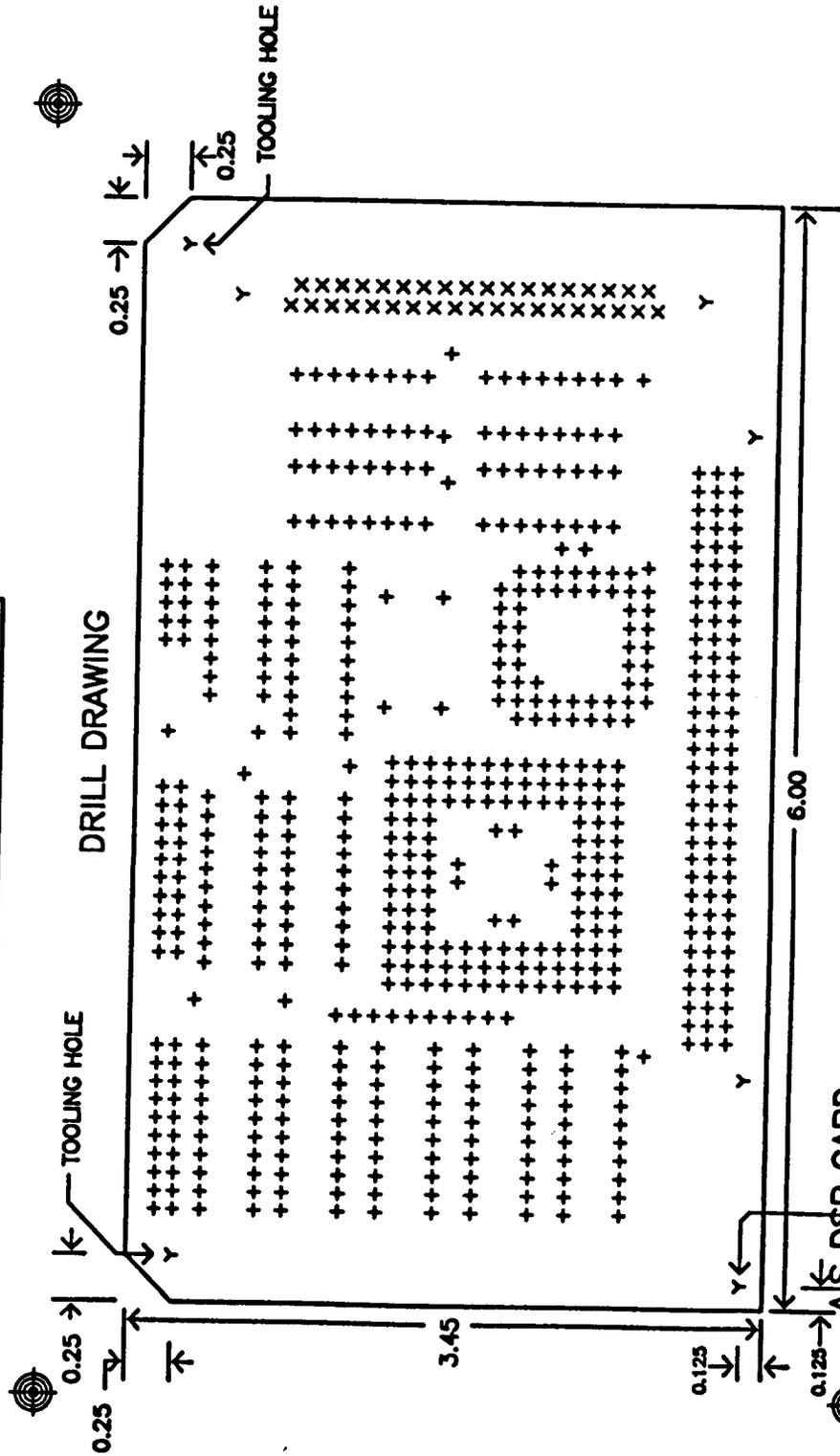
```
dsp_ctl = addr;C100..C1FF; & !reset;
```

3.4.3 Sequencer Card PCB Artwork

1. MAKE FROM COPPER CLAD FR-4 EPOXY GLASS LAMINATE PER MIL-P-13946/4
2. BOARD THICKNESS 0.062 OVERALL, 1 OZ. COPPER ON ALL LAYERS
3. SOLDER REFLOW BOARD EXCEPT AS NOTED, PROCESS OPTIONAL
4. SOLDER MASK FRONT AND REAR SIDES COLOUR DYE BLUE, SR1003, MAXIMUM ALLOWABLE BLEED IS 0.006
5. SILKSCREEN WHITE AS PER ARTWORK
6. ALL DIMENSIONS IN INCHES
7. TOLERANCES, UNLESS OTHERWISE SPECIFIED: X.XXX +/-0.005
X.XX +/-0.010
8. MOUNTING HOLE PLACEMENT +/-0.005
9. HOLE TOLERANCE +0.006 / -0.003
10. AFTER PLATING
11. ASSEMBLE LAYERS AS FOLLOWS:

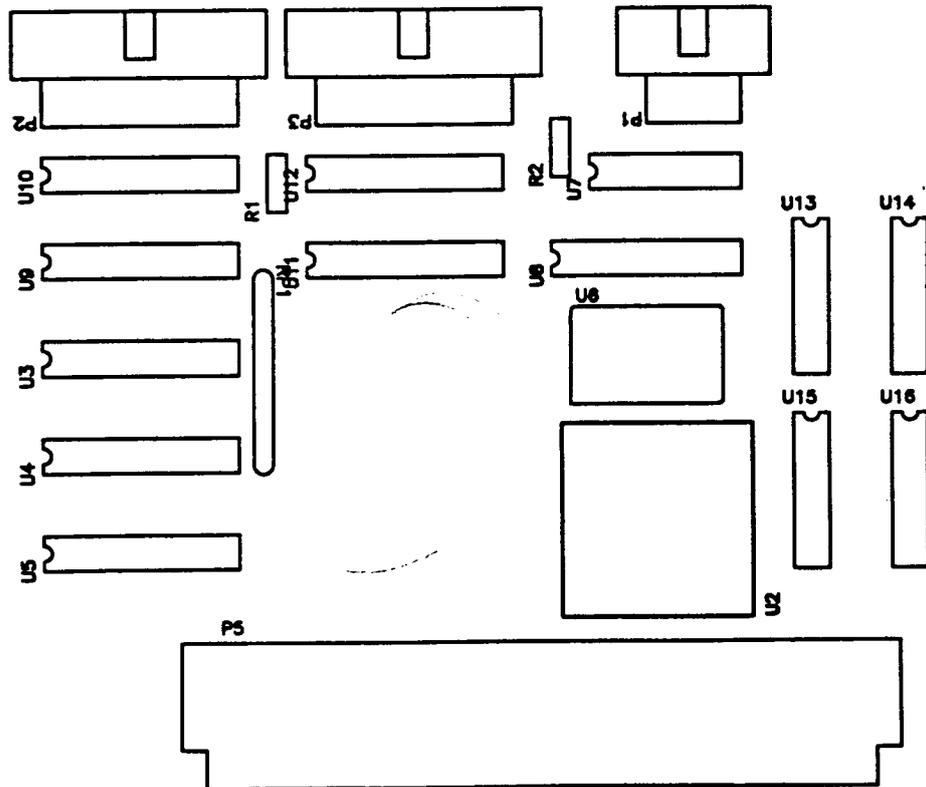
- COMPONENT (TOP)
- MIDLAYER 1
- POWER PLANE
- GROUND PLANE
- MIDLAYER 2
- SOLDER (BOTTOM)

DRILL TABLE			
SYMBOL	DESCRIPTION	QTY	PLATE
+	0.038 DIA.	606	YES
Y	0.110 DIA.	7	YES
X	0.046 DIA.	37	YES
TOTAL		658	

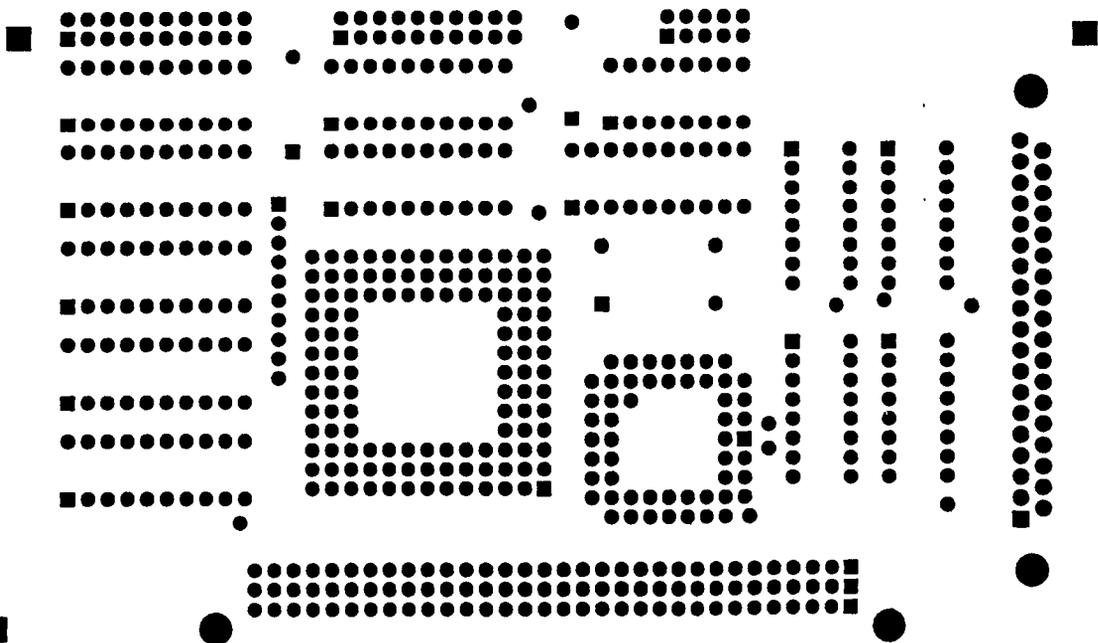


A/S DSP CARD
 ADVANCED TOOLING TECHNOLOGIES
 3440 E. BRITANNIA DRIVE TUCSON AZ 85706

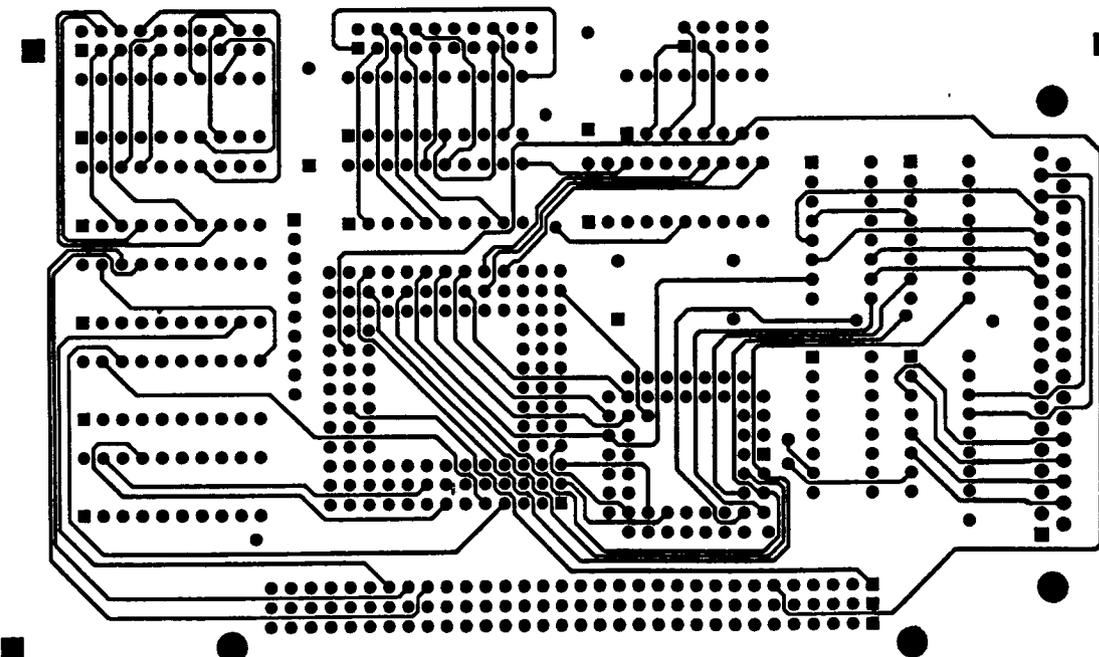
SILKSCREEN (TOP ONLY)



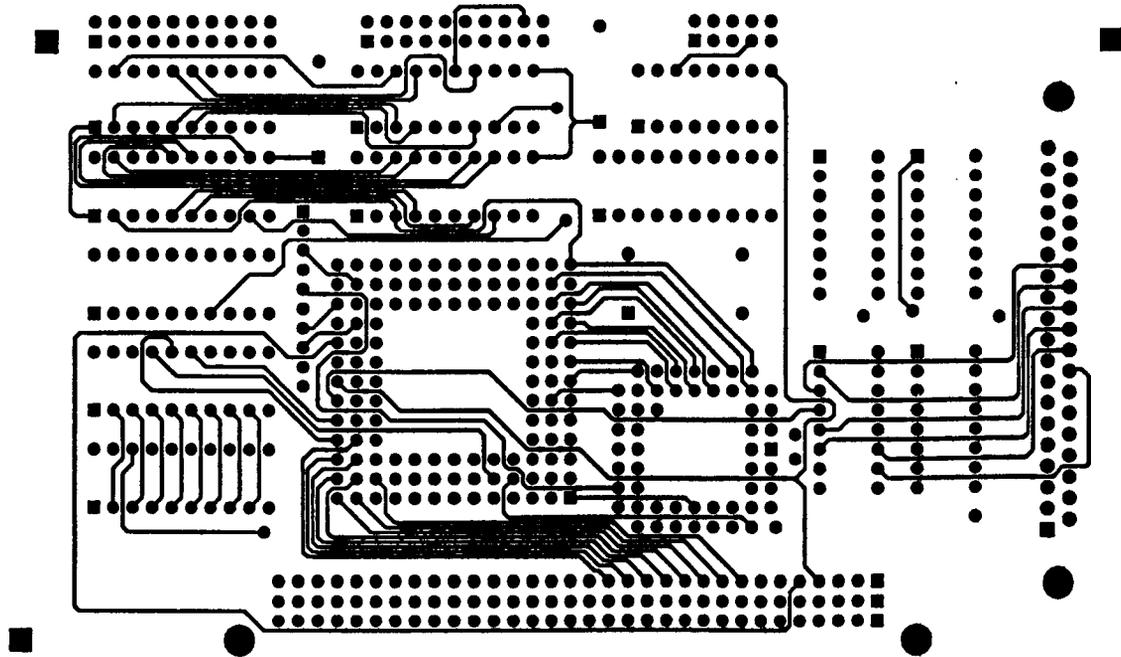
SOLDER MASK



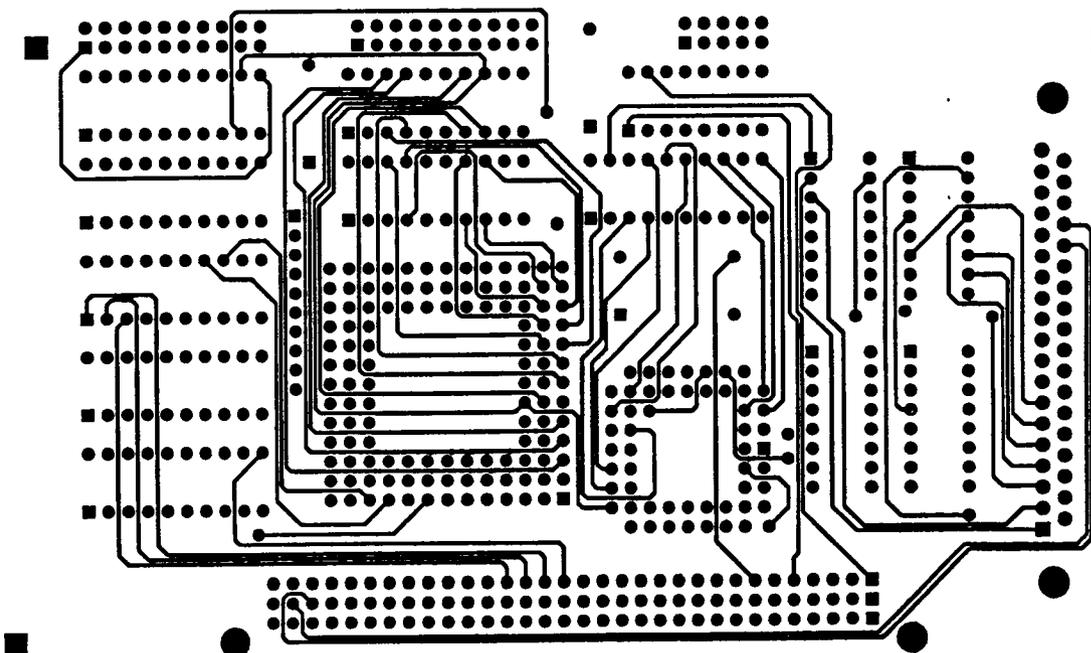
TOP LAYER



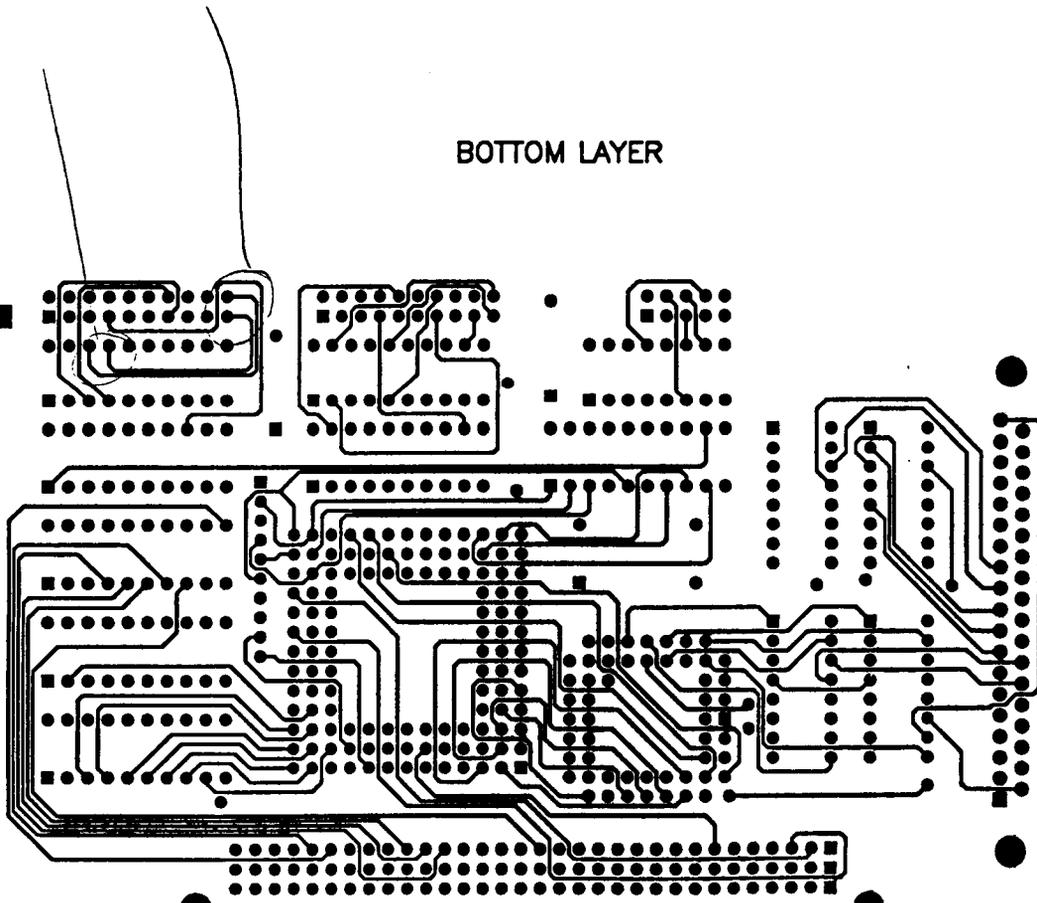
MID LAYER 1



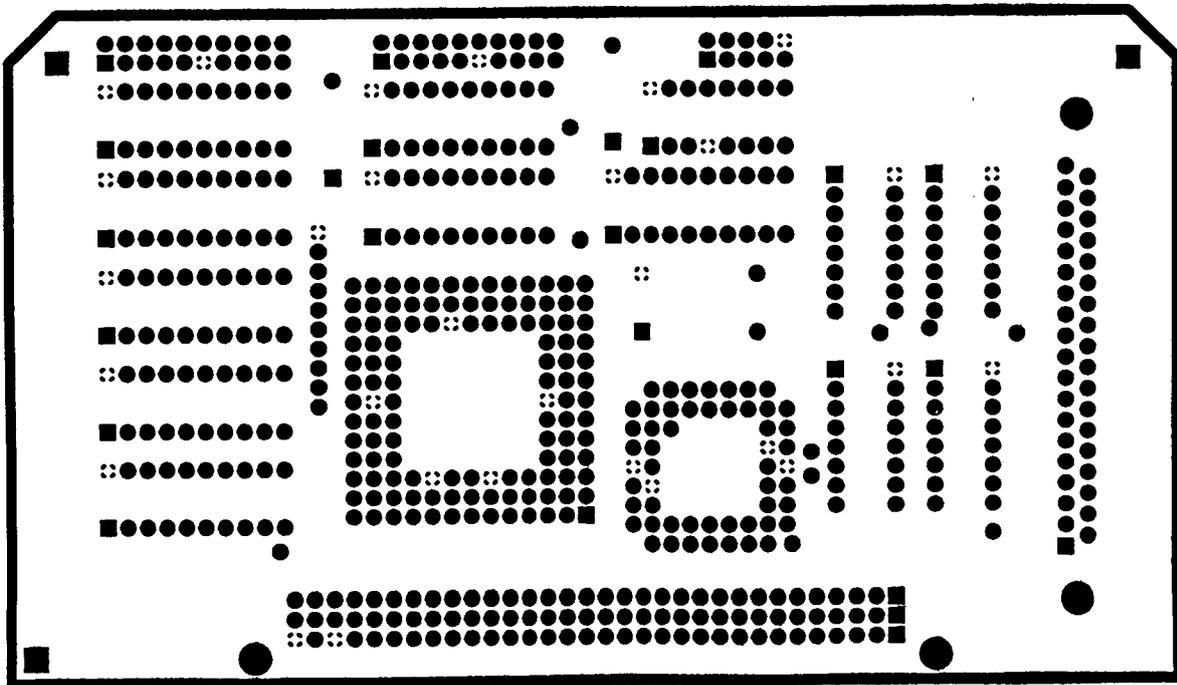
MID LAYER 2



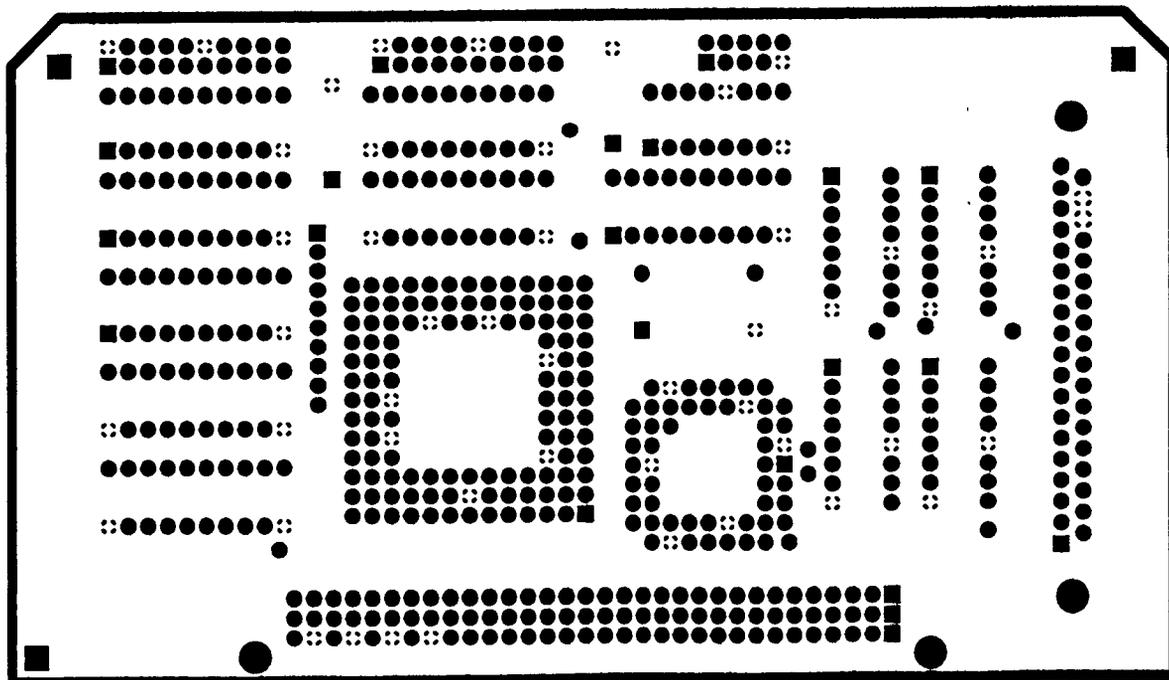
BOTTOM LAYER



Power
+ top Assy



GND
+ Bottom Assy



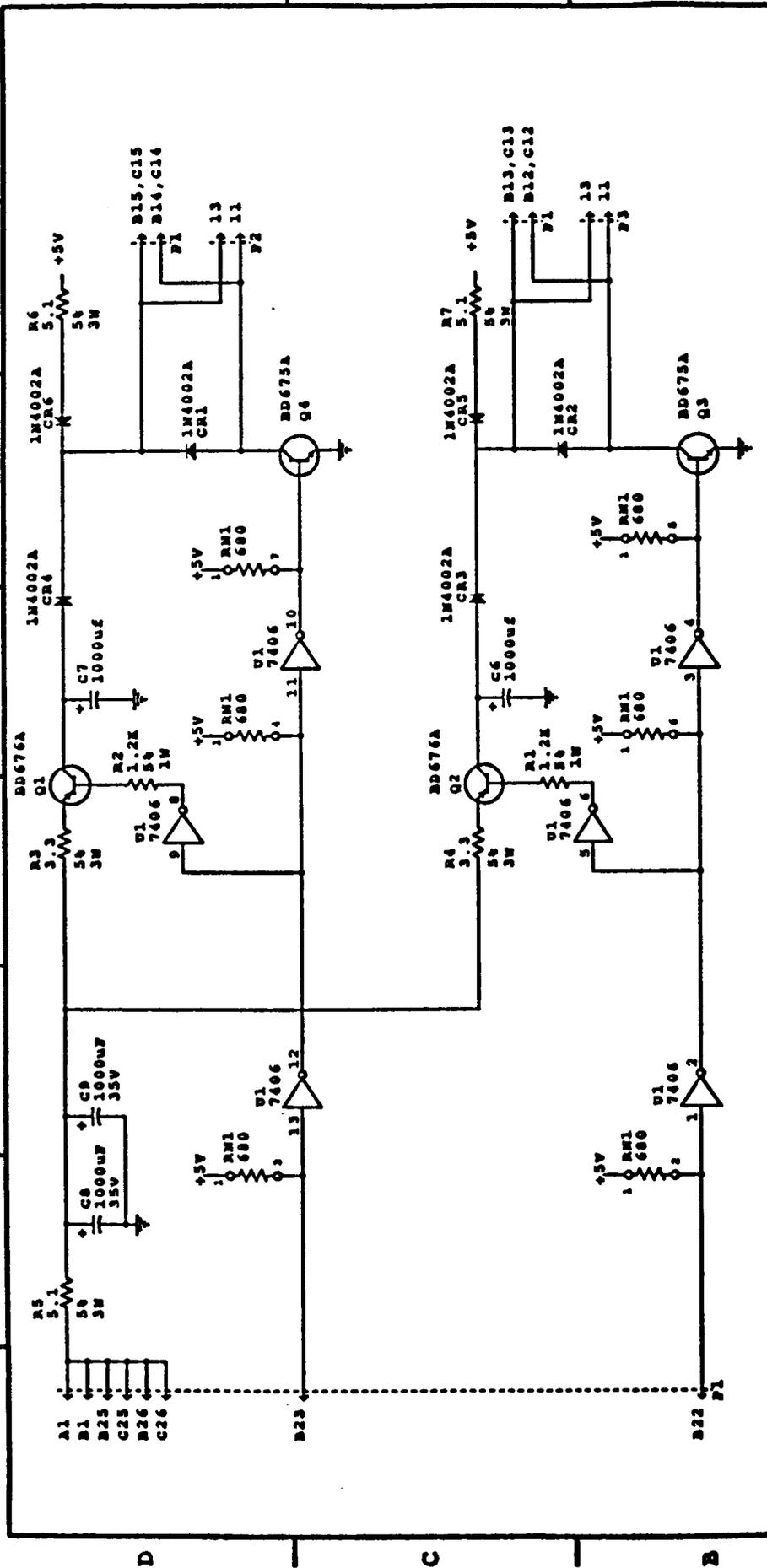
3.5 Temperature and Shutter Control Card

The temperature/shutter card contains the circuitry needed by the 68HC11 to operate an electric solenoid shutter and to measure and control the CCD temperature.

The board used in this system was, as has been previously mentioned, designed originally for use in a different camera system. It has been adapted for use in this camera system. The original system incorporated two shutters and four temperature sensors. In this application we require only one shutter and one temperature sensor, and so various parts of the board have been left un-installed because they are not being used.

3.5.1 Temperature and Shutter Control Board Schematic Diagrams

8 7 6 5 4 3 2 1



ADVANCED TECHNOLOGIES
 A DIVISION OF PHOTONICS, L.P., FOCOM, AZ

DUAL CHANNEL TEMPERATURE / SHUTTER CONTROL CARD
 SHUTTER DRIVERS

SHEET 1 OF 5 DRAWING #

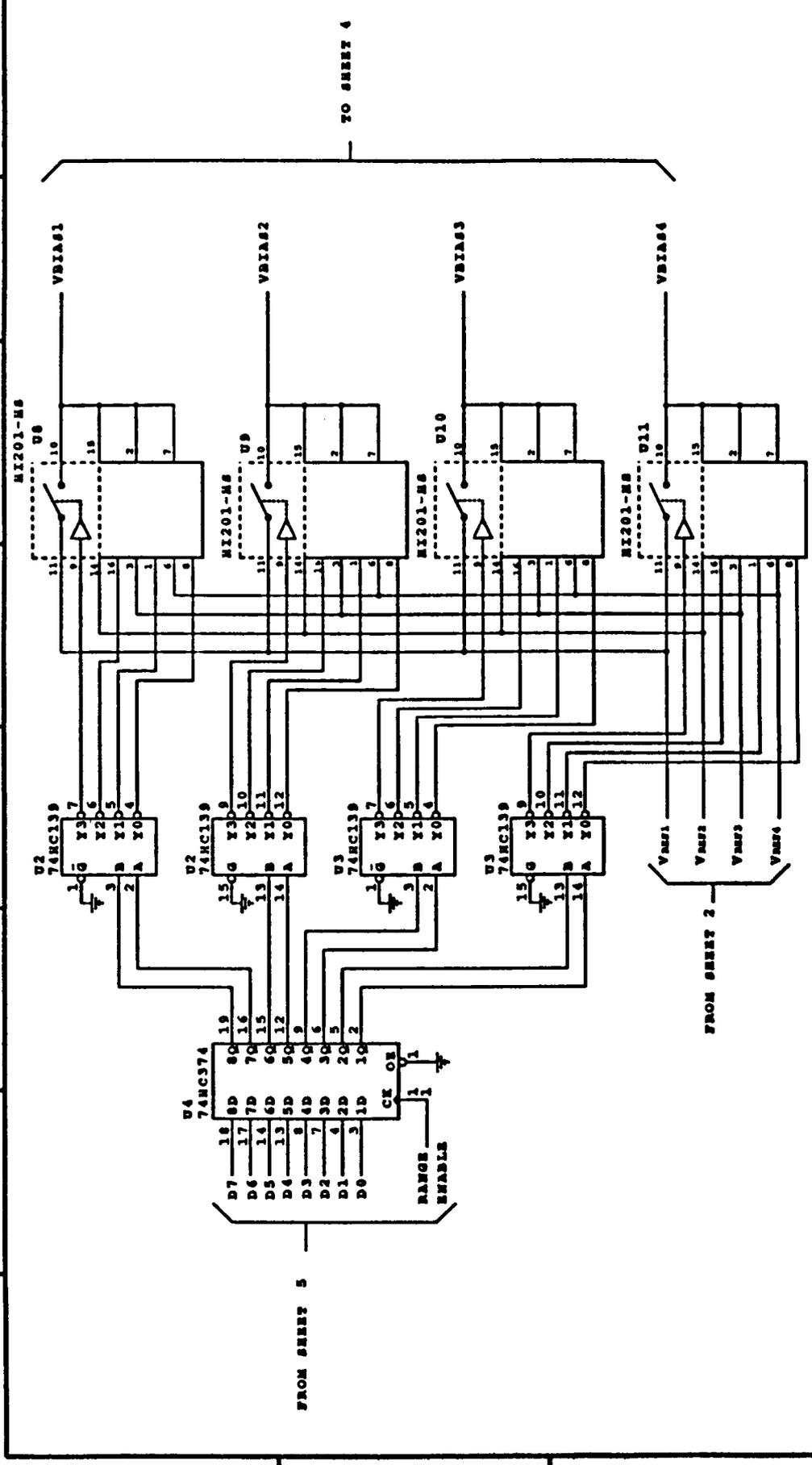
C CONTRACT/JOB NUMBER 8911-007

NEXT ASSY.	USED ON
APPLICATION	
DRAWN	DWD 02/06/90
CHECKED	
RELEASED	

LTR DESCRIPTION	APPROVED	DATE
REVISION HISTORY		

NOTES

8 7 6 5 4 3 2 1



ADVANCED TECHNOLOGIES
 A DIVISION OF BIOCHEMICALS, LP. FORT COCKS, LA

DUAL CHANNEL TEMPERATURE / SHUTTER CONTROL CARD
 TEMPERATURE RANGE SELECT

SHEET 3 OF 5 / DRAWING #
 CONTRACT/JOB NUMBER
 8911-007

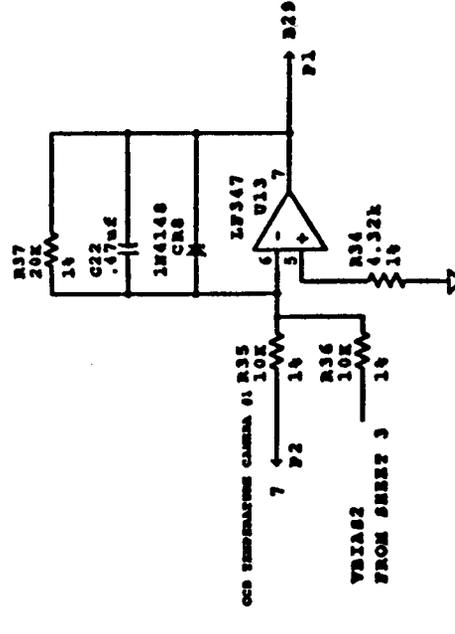
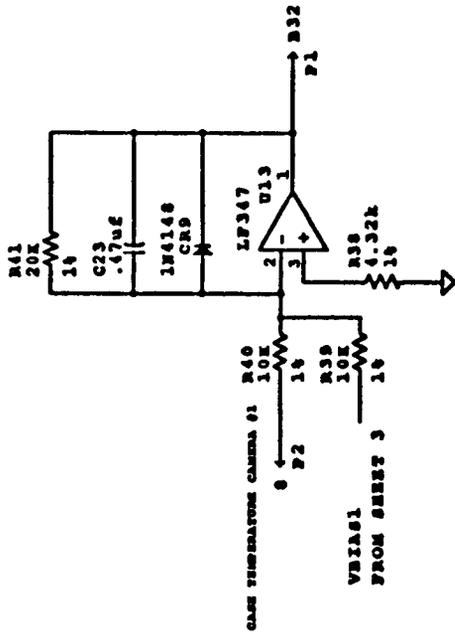
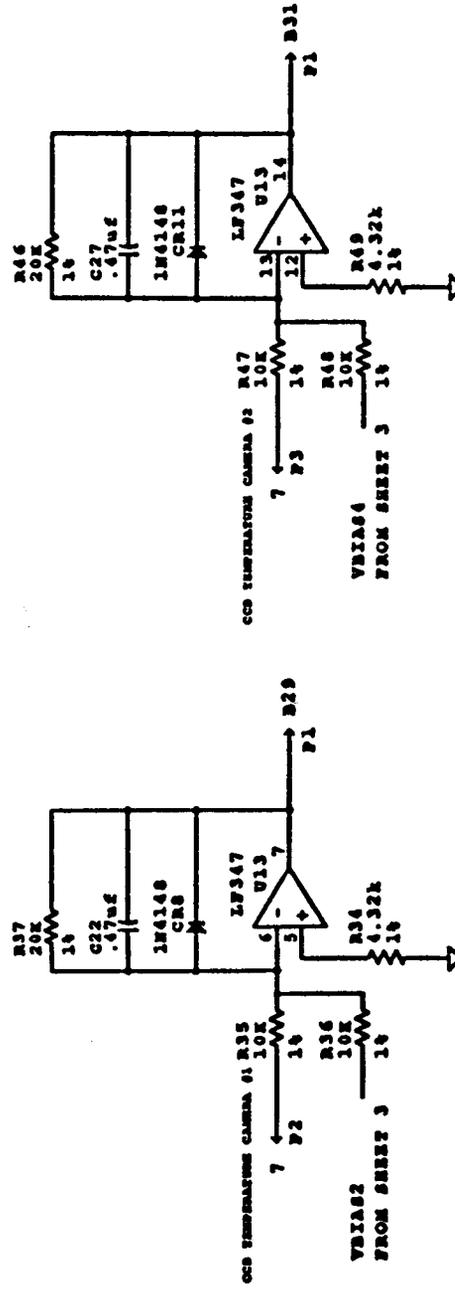
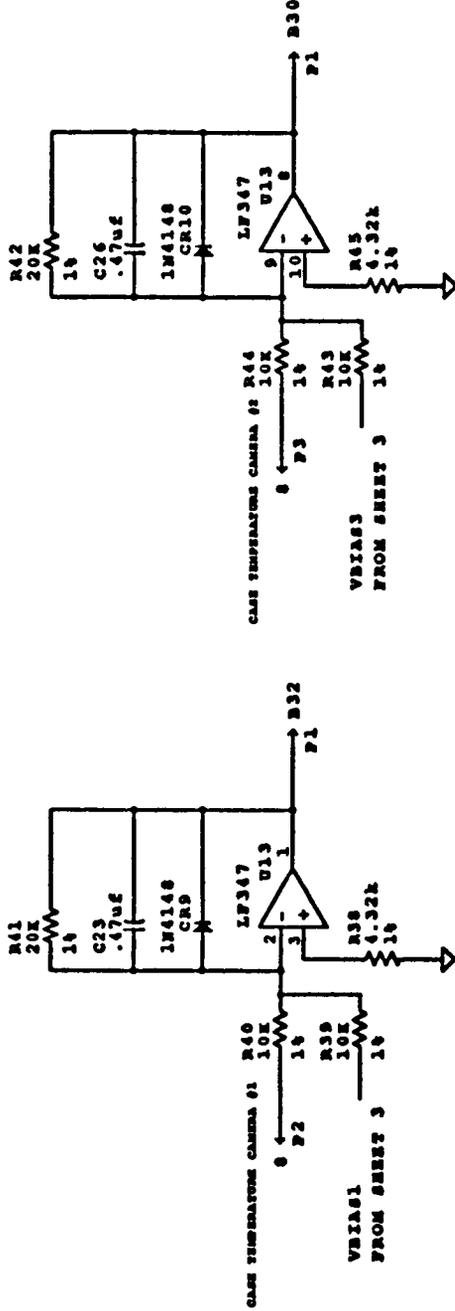
NEXT ASSY. USED ON
 APPLICATION
 DRAWN DWD 02/06/90
 CHECKED
 RELEASED

LTR	DESCRIPTION	APPROVED	DATE

REVISION HISTORY

NOTES

8 7 6 5 4 3 2 1



<p>ADVANCED TECHNOLOGIES A DIVISION OF PHOTONICS, INC., SECON, NJ</p> <p>DUAL CHANNEL TEMPERATURE / SHUTTER CONTROL CARD TEMPERATURE SIGNAL PROCESSING</p>		SHEET 4 OF 5		DRAWING #	
		C		CONTRACT/JOB NUMBER 8911-007	
NEXT ASSY. USED ON		APPLICATION		DRAWN DWD 02/06/90	
CHECKED		RELEASED		DATE	
LTR		DESCRIPTION		APPROVED	
NOTES		REVISION HISTORY			

3.5.2 Temperature and Shutter Control Board Logic Equations

```

Name      AISTPAL;
Partno    none assigned;
Date      4/12/91;
Revision  01;
Designer  Doherty;
Company   Advanced Technologies;
          sembly   AIS temperature control board decoder;
Location  U?;
Device    G22V10;

```

```

/*****/
/*                                           */
/* AIS temperature/shutter board decoder pal */
/* this board is the same as in the APL cameras. */
/*                                           */
/*                                           */
/*                                           */
/* Revision history :                       */
/*                                           */
/* Rev. 01  4/12/91                        */
/*                                           */
/*                                           */
/*****/
/*                                           */
/* Target Device = 22v10                   */
/*                                           */
/*****/

```

```

/* inputs */
Pin 1 = a15; /* 6811 address 15 */
Pin 2 = a14; /* 6811 address 14 */
Pin 3 = a13; /* 6811 address 13 */
Pin 4 = a12; /* 6811 address 12 */
Pin 5 = a11; /* 6811 address 11 */
Pin 6 = a10; /* 6811 address 10 */
Pin 7 = a09; /* 6811 address 09 */
Pin 8 = a08; /* 6811 address 08 */
Pin 9 = d7; /* 6811 address/data 07 ( not used ) */
Pin 10 = d6; /* 6811 address/data 06 ( not used ) */
Pin 11 = d5; /* 6811 address/data 05 ( not used ) */
Pin 13 = d4; /* 6811 address/data 04 ( not used ) */
Pin 14 = d3; /* 6811 address/data 03 ( not used ) */
Pin 15 = d2; /* 6811 address/data 03 ( not used ) */
Pin 16 = d1; /* 6811 address/data 02 ( not used ) */
Pin 17 = d0; /* 6811 address/data 01 ( not used ) */
Pin 18 = !we; /* system write strobe */
Pin 19 = as; /* 6811 address strobe ( not used ) */

```

```

/* outputs */
Pin 20 = !write; /* write strobe for this board */
Pin 21 = !dac_sel; /* dac chip select */
Pin 22 = !chanA; /* dac channel select A */
Pin 23 = !range; /* range latch chip select */

```

```

/* addr is the upper address inputs as an eight bit word */
FIELD addr = ;a15..a08;

```

```

/* the temperature DAC needs three signals : a write pulse, */
/* a chip select, and a channel select. */

```

```

/* the write pulse will be generated directly from the we input */

```

```
write = we;
```

```
/* the DAC will be located at address $C200 to $ C2FF */
```

```
c_sel = addr;C200..C2FF;
```

```
/* the channel select will be based directly on address line 3 */
```

```
chanA = !a08;
```

```
/* the range select latch, used for setting the temp sense range */
```

```
/* will be located at address $C300 to $C3FF. It is write only. */
```

```
range = addr;C300..C3FF; & write ;
```

3.6 Controller Module Motherboard

The controller card, sequencer card, and temperature/shutter control card are connected via a simple motherboard. The motherboard has four slots. Of these, one is unused and available for future expansion.

Each slot on the motherboard contains a 96 pin female DIN connector. The first three slots on the motherboard are bussed together on a one to one basis. The fourth slot, which the temperature/shutter control card must occupy, has a different pinout.

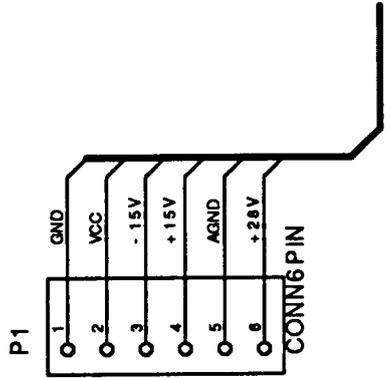
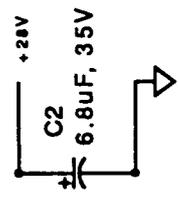
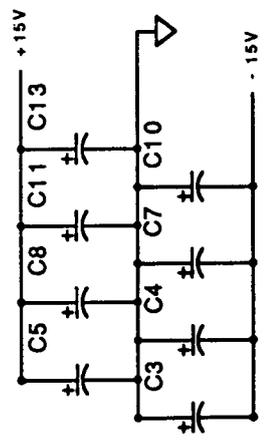
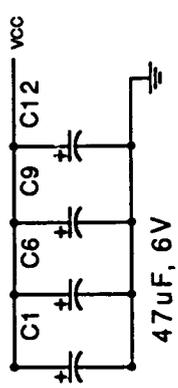
Power for the three cards comes through the motherboard. It is connected from the DB15 connector labeled 'POWER' on the outside of the controller module chassis to the P1 connector on the motherboard. The connector is just a set of pads on the circuit card and the wires supplying the power are soldered directly into the holes in the card. The pinout of this connector is shown below.

Motherboard Power Connector Pinout:

pin	function
1	ground
2	+5v
3	-15V
4	+15V
5	analog ground
6	+28V

3.6.1 Controller Motherboard Schematic

A B C D



ADVANCED TECHNOLOGIES
A DIVISION OF PHOTOMETRICS LTD.

AIS MOTHER BOARD

A SHEET 3 OF 3
CONTRACT/ JOB NUMBER

USED ON
APPLICATION
DRAWN
CHECKED
RELEASED

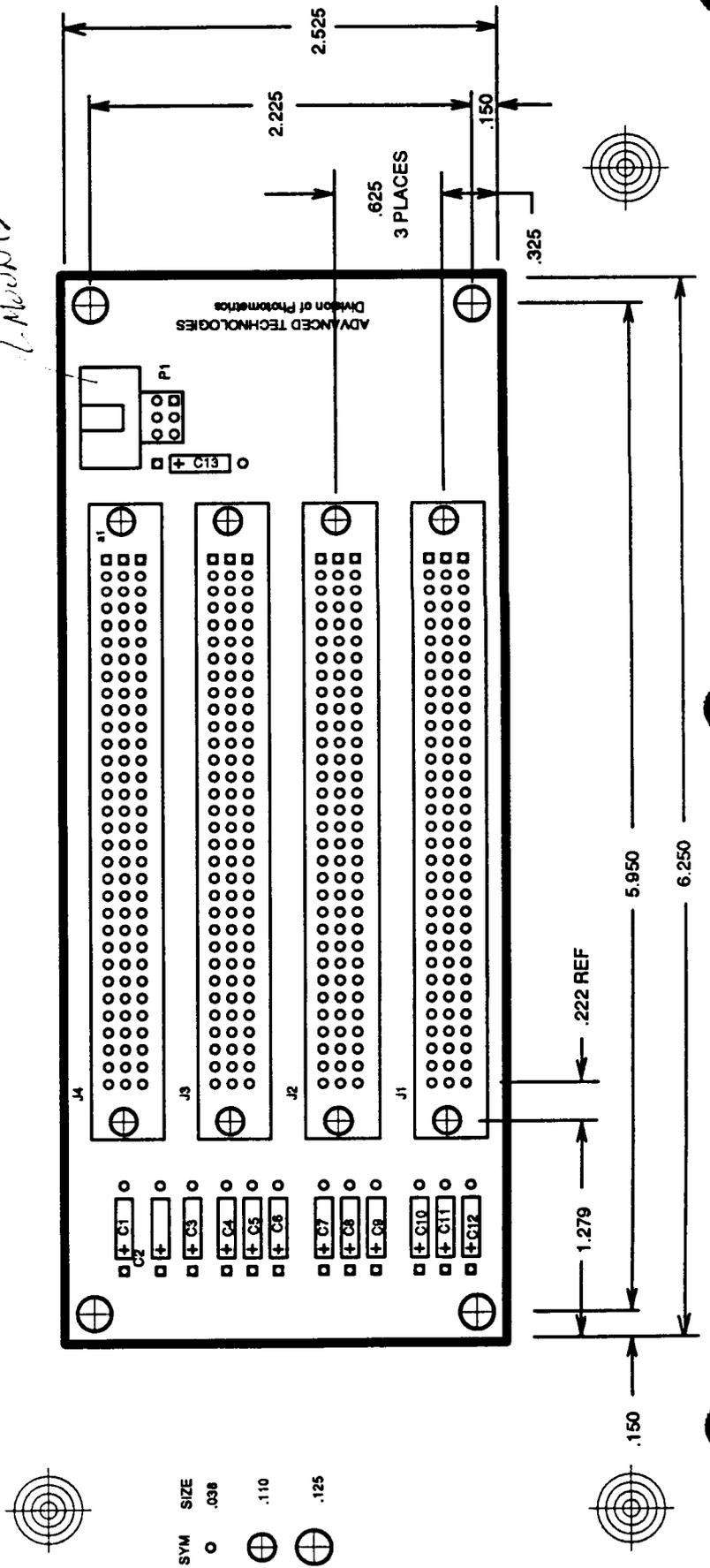
REVISION HISTORY	
LTR	DATE
DESC.	APP.

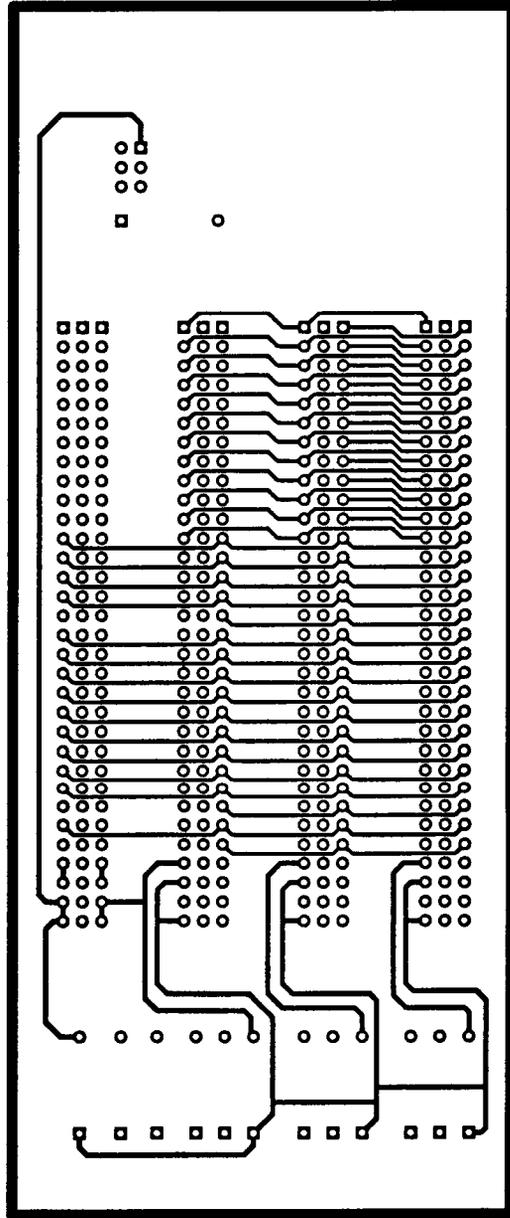
NOTES

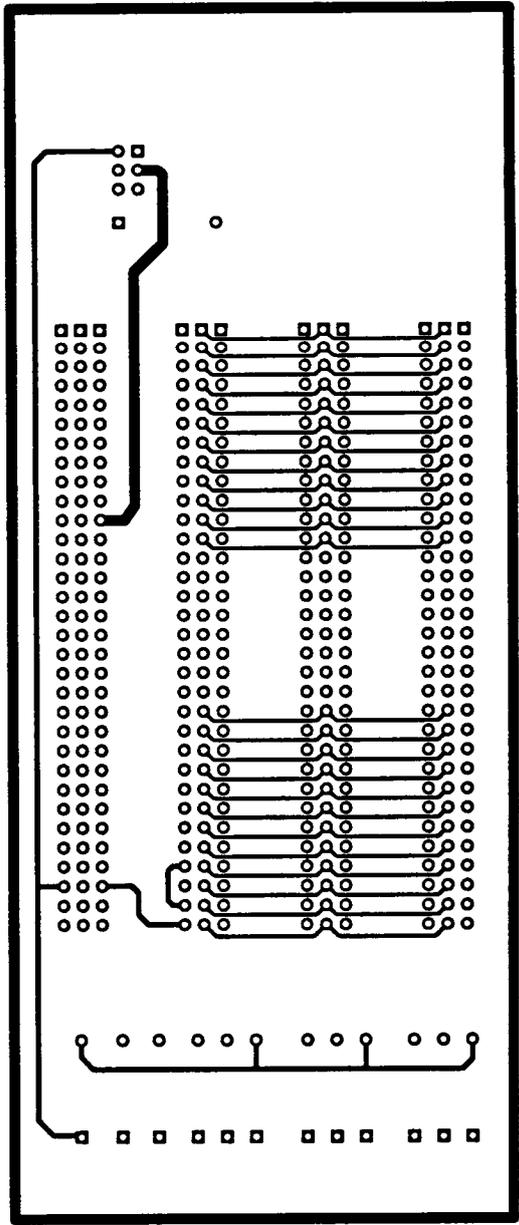
1 2 3 4

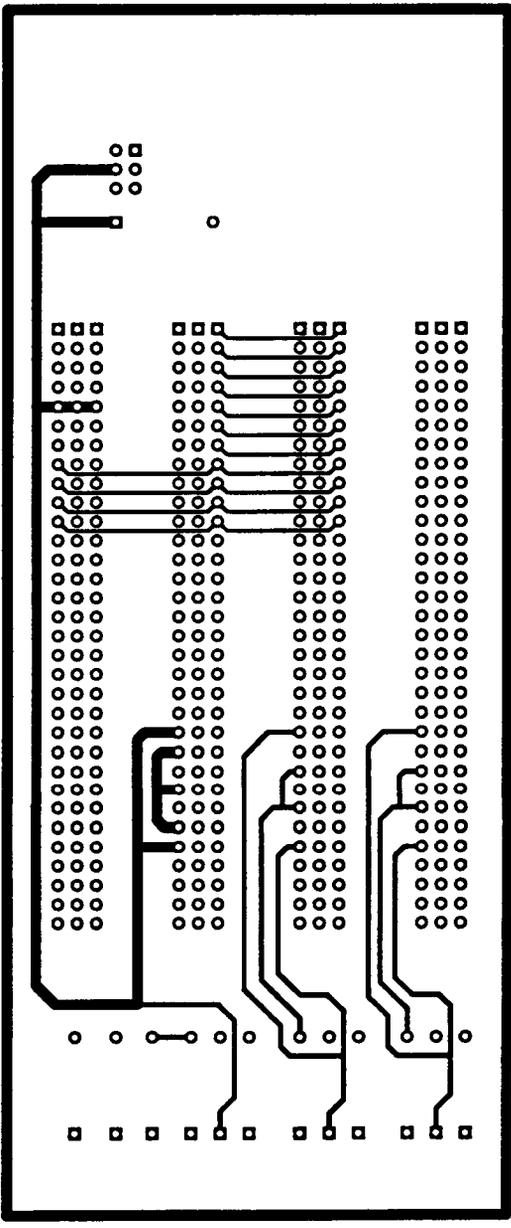
3.6.2 Controller Motherboard PCB Artwork

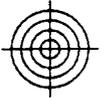
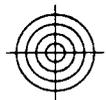
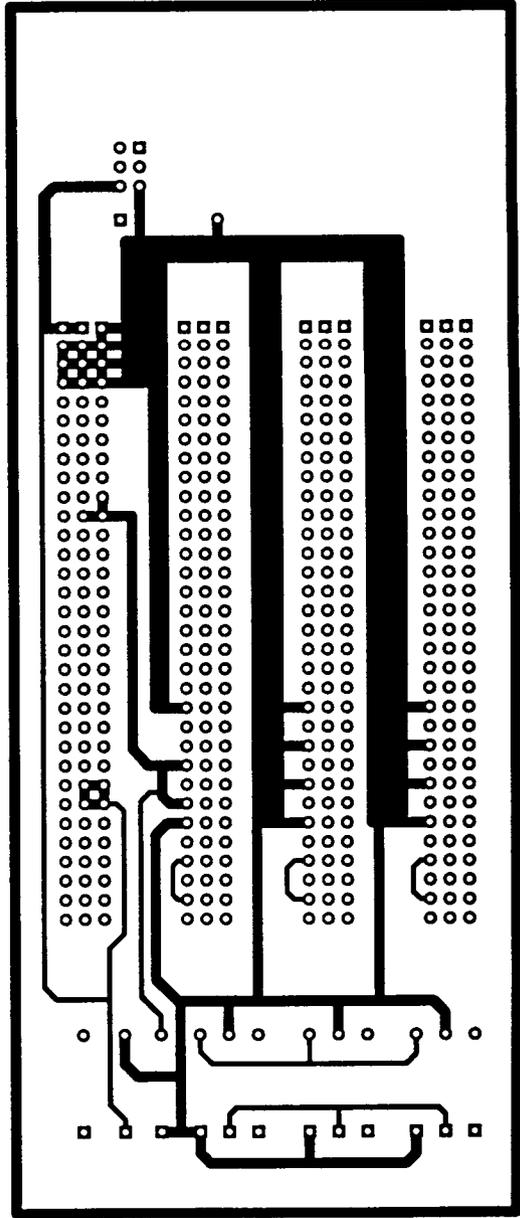
*IDC-6 RI ANGLE
1. MOUNTS ON BACK*











3.7 Fiber Optic Interface

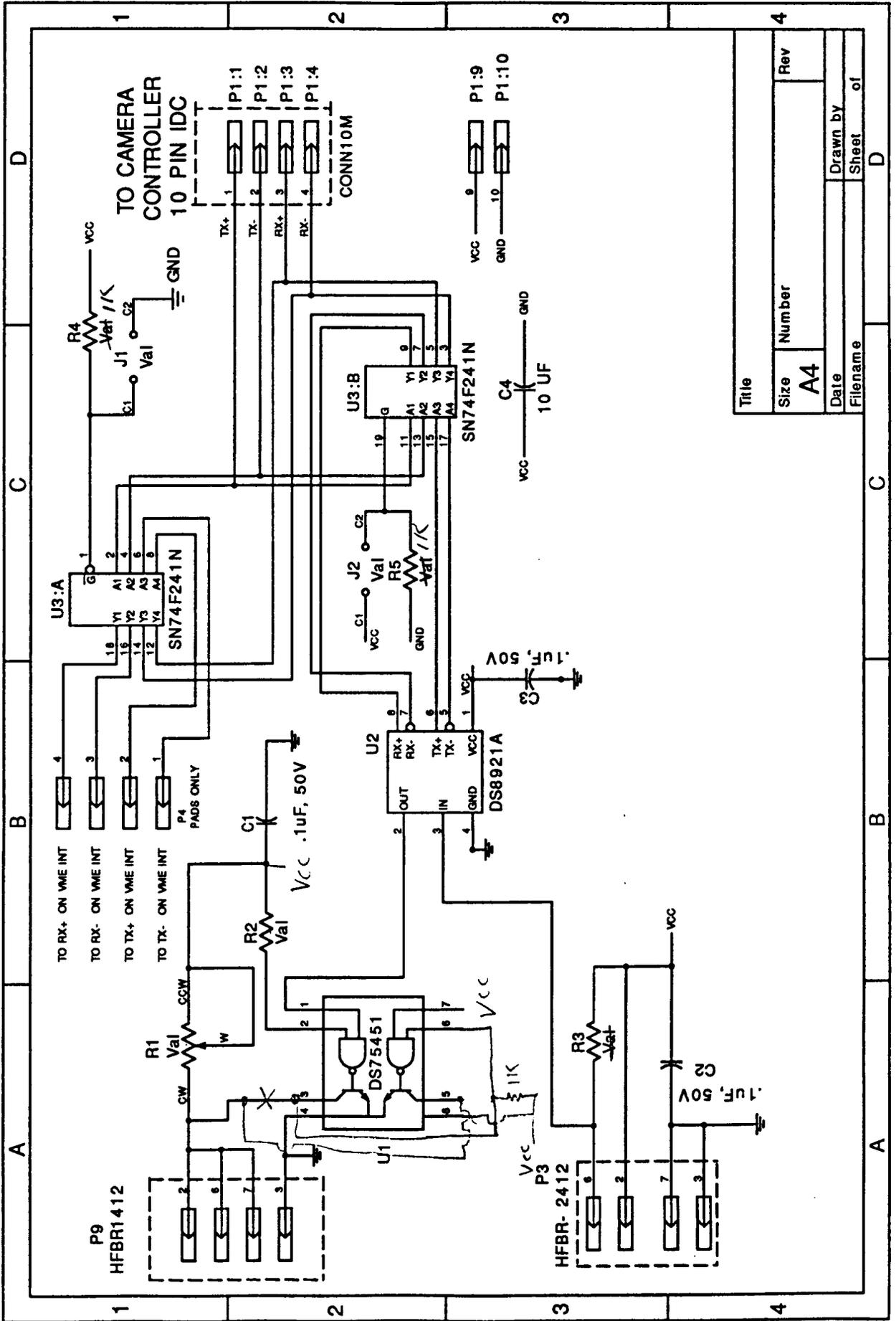
There is a small circuit board inside the controller module that is not connected to the motherboard. This circuit board holds the fiber optic driver receiver pair that may be used to connect the camera to the VMEbus interface. It also holds the connections for the RS422 data transmissions as well as an integrated circuit, a 74HC244, and jumpers which allow the user to select either RS422 or fiber optic communications.

Connector P1 is used to connect the fiber optic interface to the controller card. A 10 wire IDC ribbon cable is used. It connects from this point to P2 on the controller card.

P3 and P9 are the fiber optic components themselves.

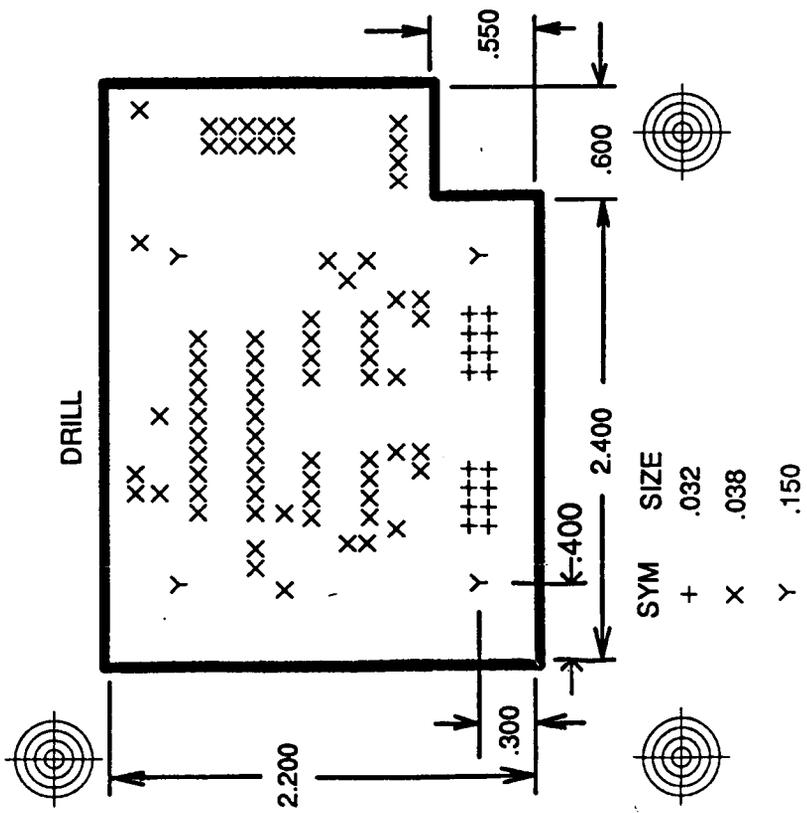
U3, the 74HC241, is used to select between the two communications channels. Jumpers J1 and J2 are used to make the selection.

3.7.1 Fiber Optic Interface Schematic Diagram

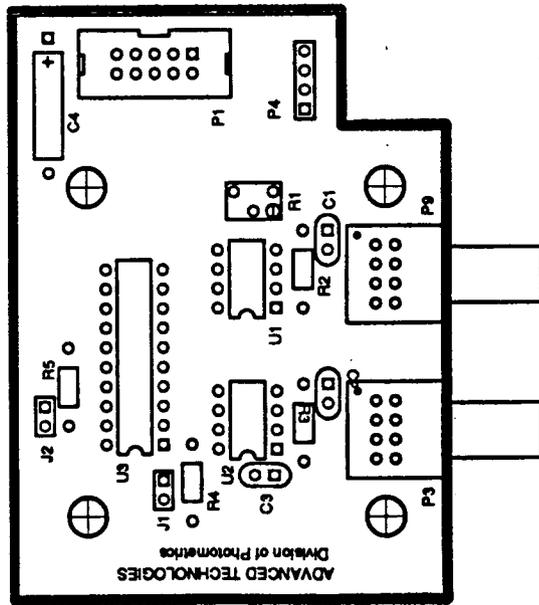


Title	
Size	Number
A4	
Date	Drawn by
Filename	Sheet of
	D

3.7.2 Fiber Optic Interface PCB Artwork



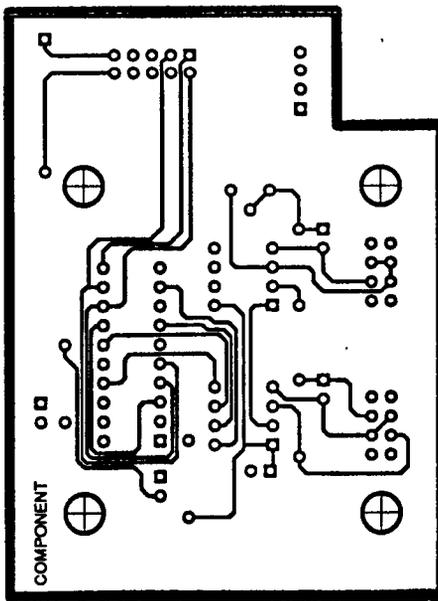
SILKSCREEN



ADVANCED TECHNOLOGIES
Division of Photometrics

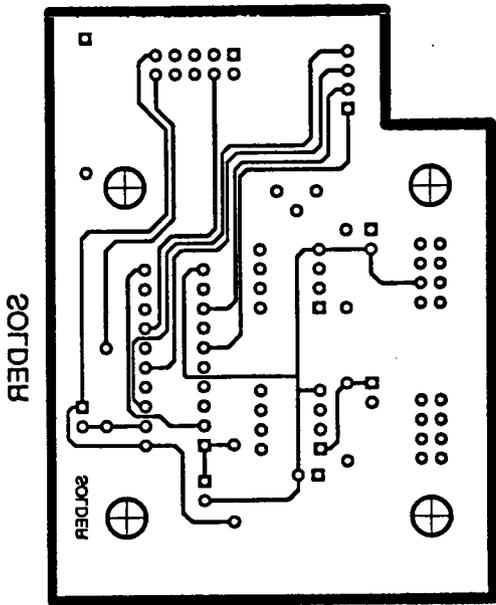


COMPONENT



COMPONENT





SOLDER

SOLDER



4. Clock/Analog Modules

4.1 Module Overview

The clock/analog modules contain the circuitry required to measure the charge of the pixels on the CCD imager. They contain digital to analog converters to generate the CCD clock rail potentials and CCD reference voltages. They contain the switches used to clock the CCD, the latches that control those switches, and the buffers that drive the CCD. They also contain the circuitry used to process and digitize the CCD video signal and to send that data to the VMEbus interface cards. The clock/analog modules' temperatures are regulated to assure that temperature related drift in the video processing circuitry does not interfere with the accuracy of the CCD data.

One clock/analog module (CAM) consists of two circuit boards in a temperature regulated enclosure. The clock board contains the digital circuits used to communicate with the controller and CCD clock timing, voltage and drive circuitry. The analog board consists of the analog processor, analog to digital converter (ADC), and data transmission logic. These boards are each discussed in turn below.

4.2 Connectors and Pinouts

There are several connectors on the outside of the CAM chassis. Two connectors serve as input and output for the control signals from the sequencer.

One connector provides the camera data and clock signals on a pair of RS422 serial lines. Additionally, there are two ST-type fiber optic connectors providing the clock and data signals in an optical form. The pinout of the camera data connector follows.

Camera Data Connector : DB9

1	data clock +
2	data clock -
3	data -
4	data +
5	ground
6	unused
7	unused
8	unused
9	unused

The connector labeled "power" is used to provide the DC voltages necessary to operate the controller electronics. The power is normally provided by connecting this connector to the power supply module with the cables provided. These cables are approximately 3 meters in length. The power supply voltages provided must be regulated, as no regulation takes place inside the module.

CAM Power Connector : DB15M

pin	voltage	pin	voltage
1	-15V	9	GND
2	-15V	10	GND
3	+5V	11	GND
4	+5V	12	GND
5	+15V	13	GND
6	+15V	14	GND
7	+28V	15	GND
8	+28V		

The CAMs are connected to the controller module in a 'daisy-chain' fashion in which the control signals pass through one board to the next. A cable is connected from the 'control source' connector on the controller module to the 'control in' connector on one of the CAMs. Another cable is then routed from the 'control out' connector on that CAM to the 'control in' connector on the next. The cables used are of the one to one type. The pinout of the connectors on the CAMs is the same as on the controller module.

Camera Control Connector Pinout : DB37

pin	function	pin	function	pin	function
1	WR+	17	D0+	33	unused
2	WR-	18	D0-	34	unused
3	A0+	19	D1+	35	unused
4	A0-	20	D1-	36	unused
5	A1+	21	D2+	37	unused
6	A1-	22	D2-		
7	A2+	23	D3+		
8	A2-	24	D3-		
9	A3+	25	D4+		
10	A3-	26	D4-		

11	A4+	27	D5+
12	A4-	28	D5-
13	A5+	29	D6+
14	A5-	30	D6-
15	extra+	31	D7+
16	extra-	32	D7-

4.3 Clock Generator Card

The clock generator card consists of all the analog circuitry associated with the production of the CCD clock and reference signals for one CCD output port. It also contains enough digital circuitry to interface to the camera control module. The clock generator is the only interface between the camera control module and the CAM.

The 'control in' and 'control out' connectors on the CAM chassis are connectors P1 and P2 on the clock generator card. The 'control in' connector is tied to the 'control source' connector on the camera control module by a DB37 cable with one to one wiring. Another cable may be used to connect this CAM with another. As many as eight uniquely addressable CAMs may be daisy chained in this way. Termination resistor packages RP1 to RP6 are installed in the last module in the chain.

The address map of the clock generator modules contains 30 write-only byte wide registers that are addressed by the controller. These registers are used to direct all operations of both the clock and analog boards. Twenty-four registers are used to set CCD clock rail and operating voltages. The remaining registers are used to operate the analog processor and ADC, and operate switches used to generate CCD register clocks. The address map of the clock generator card is shown below.

AIS1 Clock Generator Address Map

CAM ID.	\$FF00
Voltage DACs	\$FF08 -> \$FF1F
Serial Clock Control	\$FF20
Parallel Clock Control	\$FF24
Other Clock Control	\$FF28
Analog Processor Control	\$FF2C
CAM Configuration	\$FF30

4.3.1 Sequencer Data Interface

The differentially driven DSP address and data signals from the controller module are received by a set of 41LF type receivers from AT&T, U1 to U4. All the devices on the clock generator card that the DSP may control appear as memory locations in the DSP's 'y data' memory space. The various clock generator cards in the system share common address locations, but may be enabled or disabled selectively by the DSP program. At address location \$F000, commonly available on all boards at all times, is the CAM ID latch, used by the DSP program to determine which CAMs will receive the following control sequences. The DSP writes a value to this latch. Each bit of the output is tied to on pin on one side of eight position jumper block J1. The other pins on the other side of J1 are all connected to the enable line referred to as ID* on the schematics. If this line is low, then the address decoders will respond to write accesses from the DSP. Each CAM is configured with one jumper in place on J1. The position of this jumper determines which 'ID' bit the CAM will respond to. In this way, as many as eight uniquely addressable CAMs may coexist in the system, CAMs may share IDs if desired, and the DSP may access any subset of the addressable CAMs simultaneously.

4.3.2 CAM Configuration Latch

The clock generator card contains one latch referred to as the CAM configuration latch. This latch is used to enable and disable various parts of the clock generator for diagnostic and CCD test purposes. The latch, U10, a 74F273 type, has 5 outputs which are used. Bit 0 is used to enable and disable the CCD clocks as discussed below in section 4.3.4. This bit has an active high sense. The other bits are active low. Bit 1 may be used to disable the analog control signals, bit 2 may be used to disable the parallel clocks (useful when testing CCD devices for charge injection through these clocks), bit 3 enables the serial clock control latch, also useful for testing, and bit 4 controls the output of U14, which controls a variety of CCD signals, including the transfer gate, CCD reset gate, and summing well.

The default condition for the CAM configuration latch is determined by U5, a power monitor IC which resets this latch at power up. The CCD control latch outputs are then all enabled, but the CCD clocks are disconnected from the analog card.

4.3.3 CCD Voltage Digital to Analog Converters

All the DC voltages needed, both as clock rail settings and DC references, are generated on the clock module for the CCD or CCD output port that it is connected to. Separate clock voltages and

control signals are generated for the CCD parallel register, serial register, parallel transfer gate, summing well, and reset gate. Two spare clock lines are available for future use. The parallel and serial clocks may be set to any of three voltages: low, midrange, or high. There are four serial and four parallel register clock lines that may be programmed with independent timing, but common clock voltage rails.

Eighteen programmable voltages are available as clock rail voltages for driving the CCD clocks. Other programmable voltages are used to control the operating point of the output amplifier and other static levels such as substrate and last or output gate potential. The voltages are established by twenty-four eight bit DACs programmed by the DSP. Each clock rail may be programmed in 256 equal increments over the range of voltages listed in Table 4.1.

TABLE 4.1
Programmable Clock Rails and Voltage Ranges

<u>address</u>	<u>clock signal</u>	<u>range (volts)</u>	
\$FF08	parallel low	-12.5	+12.5
\$FF09	parallel midrange	-12.5	+12.5
\$FF0a	parallel high	-12.5	+12.5
\$FF0b	serial low	-12.5	+12.5
\$FF0c	serial midrange	-12.5	+12.5
\$FF0d	serial high	-12.5	+12.5
\$FF0e	transfer gate low	-12.5	+12.5
\$FF0f	transfer gate high	-12.5	+12.5
\$FF10	summing well low	-12.5	+12.5
\$FF11	summing well high	-12.5	+12.5
\$FF12	substrate	-3.0	+3.0
\$FF13	last gate	-5.0	+5.0
\$FF14	reset gate low	0	+15.0
\$FF15	reset gate high	0	+15.0
\$FF16	reset drain	+5.0	+20.0
\$FF17	output drain	+5.0	+25.0
\$FF18	spare gate 1 low	-12.5	+12.5
\$FF19	spare gate 1 high	-12.5	+12.5
\$FF1a	spare gate 2 low	-12.5	+12.5
\$FF1b	spare gate 2 high	-12.5	+12.5
	current source	-12.5	+12.5
	spare 1	-12.5	+12.5

The voltages from the DACs are then filtered and buffered using simple operational amplifier based filter circuits. These circuits, constructed around U16, U17, U21, U22, and U23, provide gain and offset correction to transform the voltages from the 0 to 5 volt range of the DACs to the ranges listed above in Table x. All the DC voltages thus produced are tied to test points located in one corner of the printed circuit board. The voltages used as CCD clock rail potentials are then tied to the inputs of a set of analog switches used to toggle the CCD clock line from one potential to the other.

4.3.4 CCD Clock Switching

The CCD clock switches are controlled by the sequencer writing eight bit data values in to the CCD control latches U12, U13, and U14, 74F574 type octal flip flops. U12 controls the CCD parallel clocks, U13 controls the CCD serial clocks, and U14 controls the other clocked signals, including the CCD reset gate, summing well, and transfer gate. Since the parallel and serial clocks are tri-level, each CCD parallel and serial clock is controlled by two bits in the appropriate register. The reset gate, summing well, transfer gate and other signal are controlled by only one bit since they have only two possible settings.

The CCD parallel clocks are switched by U27, U28, U29, and U30, DG413 type analog switches from Intersil. The resulting clock signals then pass through U31, an HI-201HS type analog switch, and to P4 the connector that brings the CCD clocks to the buffers and resistor networks on the analog card.

The CCD serial clocks are switched by U34, U35, U36, and U37, HI-201HS type analog switches. These switches are chosen for their speed. The outputs of these switches then pass through U38, also an HI-201HS type switch, then to the P4 connector.

The switches U31 and U38 mentioned above, as well as U42 for the reset and summing well potentials, U39 for the transfer gate, and U43 and U44 for various other CCD signals, are used to allow the camera controller to effectively disconnect the CCD signals generated on the clock card from the analog card. This allows the control program to essentially turn the CCD off. These switches are controlled by bit 0 in the CAM configuration latch. At power-up reset, a DS1231 power monitor circuit resets the CAM configuration latch, assuring that the CCD clocks are initially disconnected from the analog card.

P4 : CCD clock connector

2	parallel clock 1	28	unused
---	------------------	----	--------

4	parallel clock 2	30	unused
6	parallel clock 3	32	CCD substrate (unused)
8	parallel clock 4	34	summing well clock
10	transfer gate	36	reset gate clock
12	serial clock 1	38	summing well low
14	serial clock 2	40	summing well high
16	serial clock 3	42	reset gate low (unused)
18	serial clock 4	44	reset gate high
20	extra clock 4	46	VLG CCD last gate
22	extra clock 3	48	VOD CCD output drain
24	extra clock 2	50	VRD CCD reset drain
26	extra clock 1		

All other pins on the P4 connector are tied to ground.

4.3.5 Analog Processor Control

The CCD video processing circuitry, located on the analog card, must be operated synchronously with the CCD clock timing. The various signals that control the CCD signal processing and digitization are generated on the clock card. Eight bits of digital control are provided by U11, the analog control latch, a 74F574 type octal flip flop.

Of the eight signals provided, six are currently used. They are covered in more detail in the description of the analog card itself. On the clock card, they are simply taken from the control latch and tied to the connector labeled P3. Connector P3 is also where the clock card power is provided. The CAM power is connected to P1 on the analog card, and connected to the clock card by a ribbon cable at P3. The pinout for this connector is shown below.

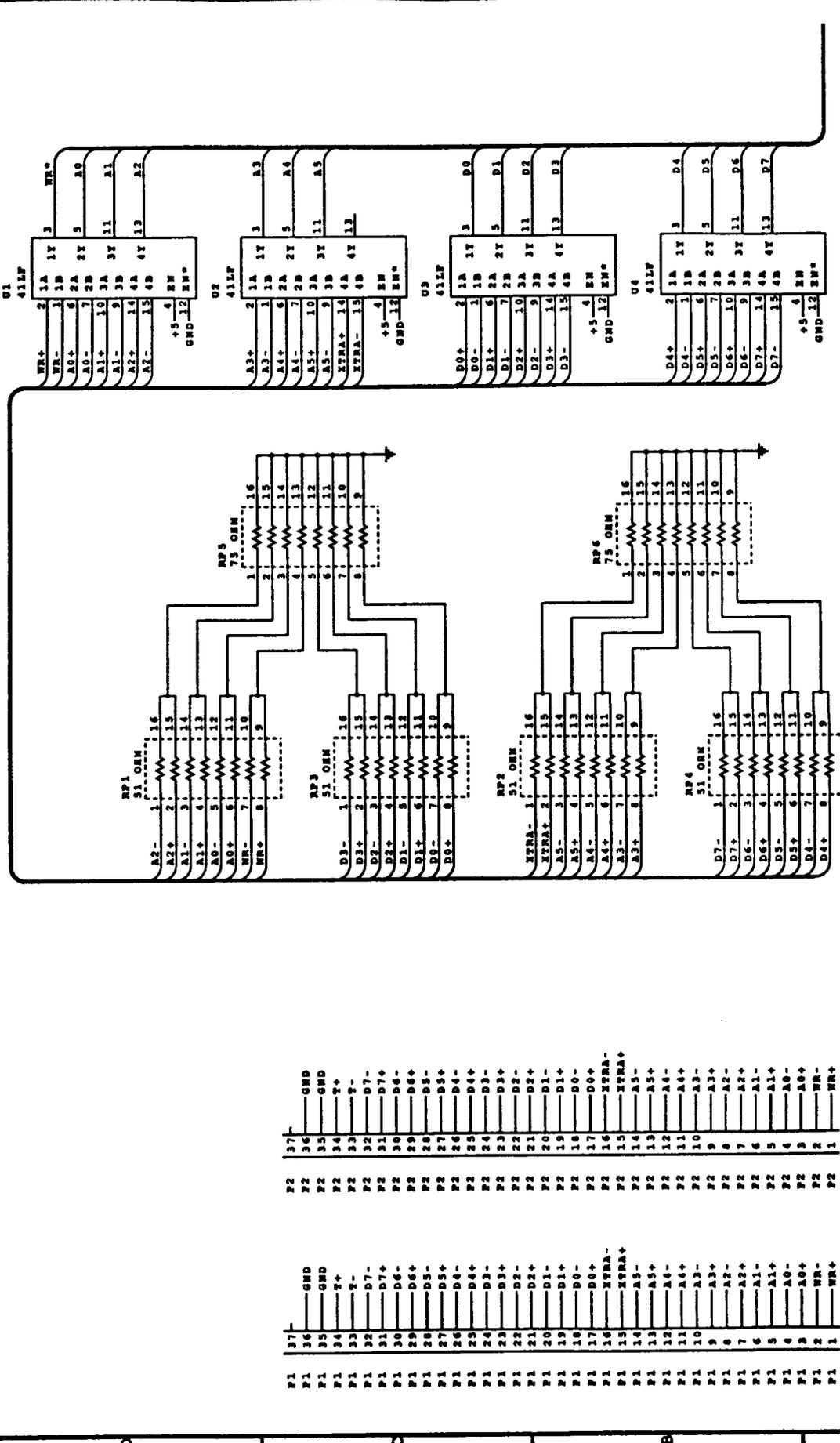
P3 : Power/Analog Connector

1, 2, 3, 4		+28 Volts
5, 6, 7, 8		+15 Volts
9, 10, 11, 12		+5 Volts
13, 14, 15, 16		-15 Volts
18, 20, 22, 24		unused
26	RIN	integrator reset
28	DCR	DC Restore

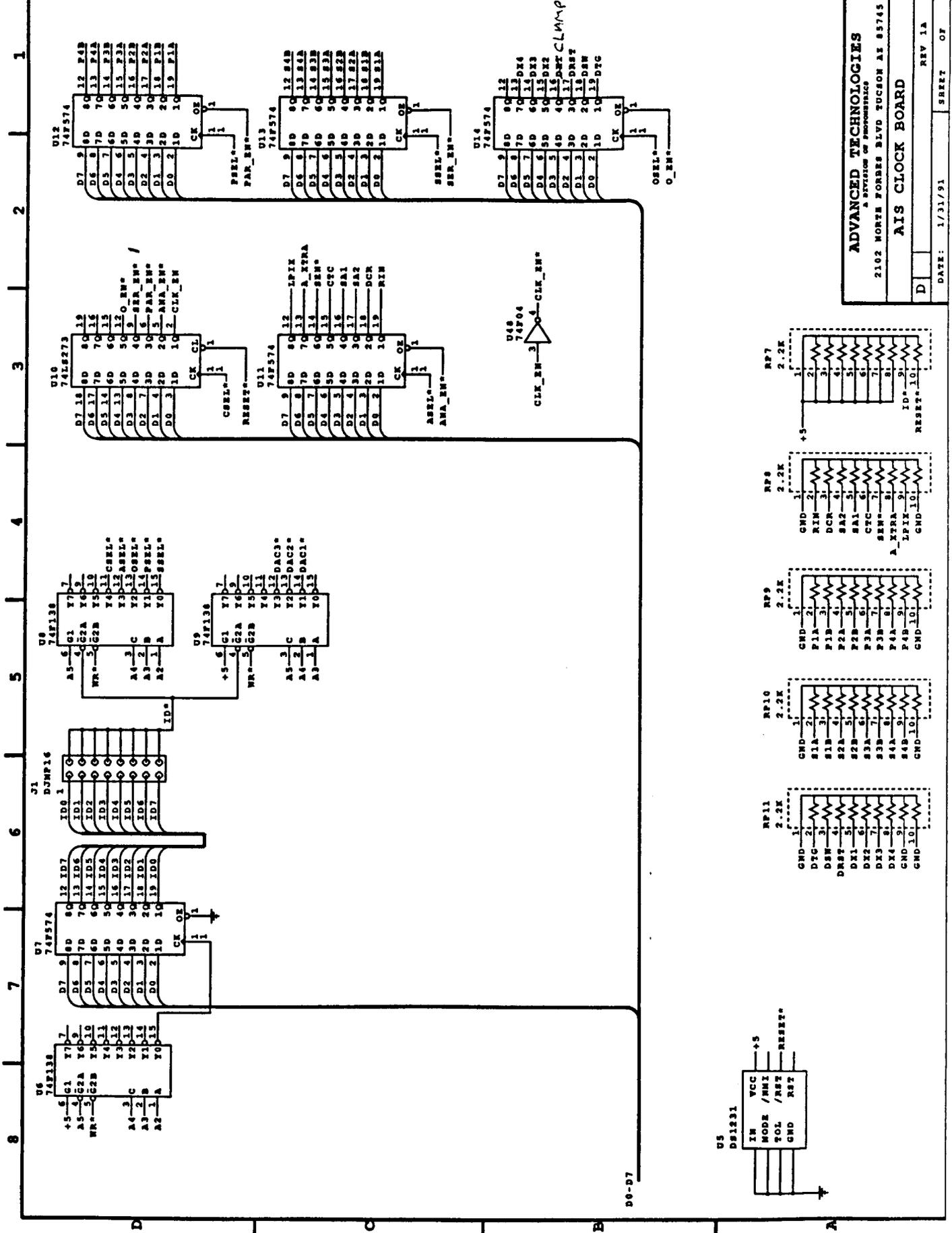
30	SA1	sample 1
32	SA2	sample 2
34	CTC	command to convert
36	SEN*	data send enable
38	extra	
40	LPIX	'last pixel' (no longer used)

4.3.6 Clock Generator Card Schematic

1 2 3 4 5 6 7 8



Csel = 110C x x



ADVANCED TECHNOLOGIES
 A DIVISION OF PHOENIX

2102 MORRIS FORBES BLVD TUCSON AZ 85745

AIS CLOCK BOARD

D

DATE: 1/31/91

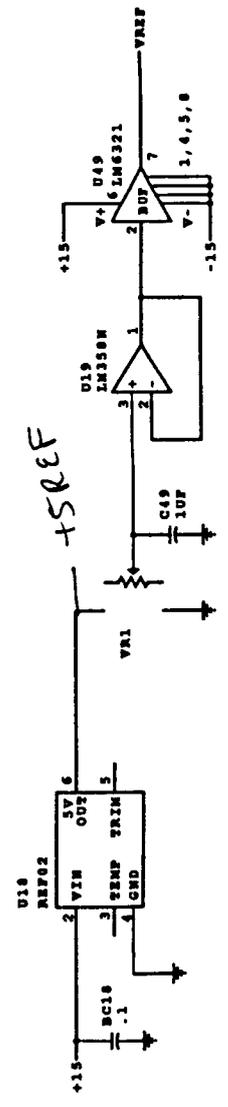
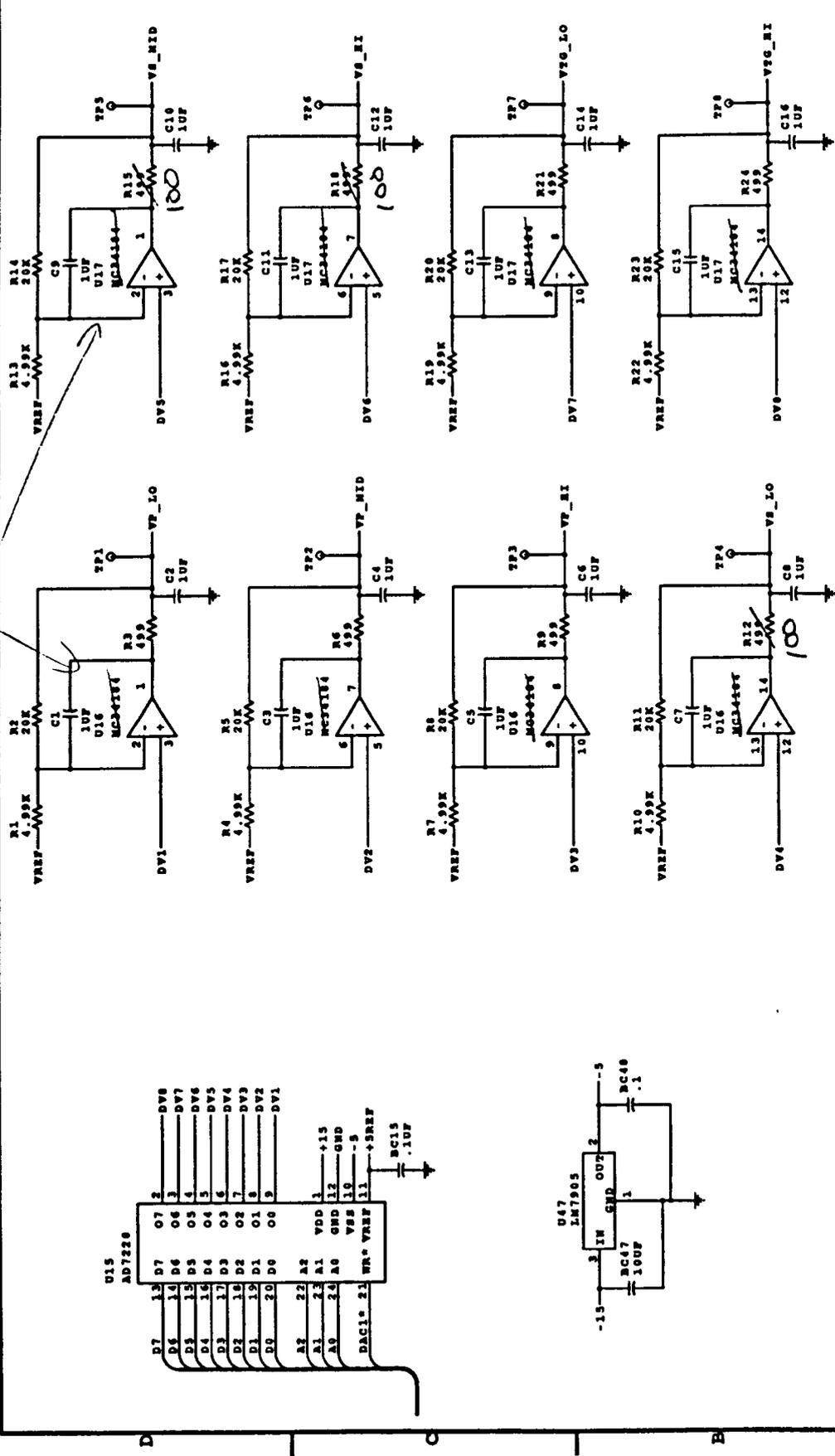
REV 1A

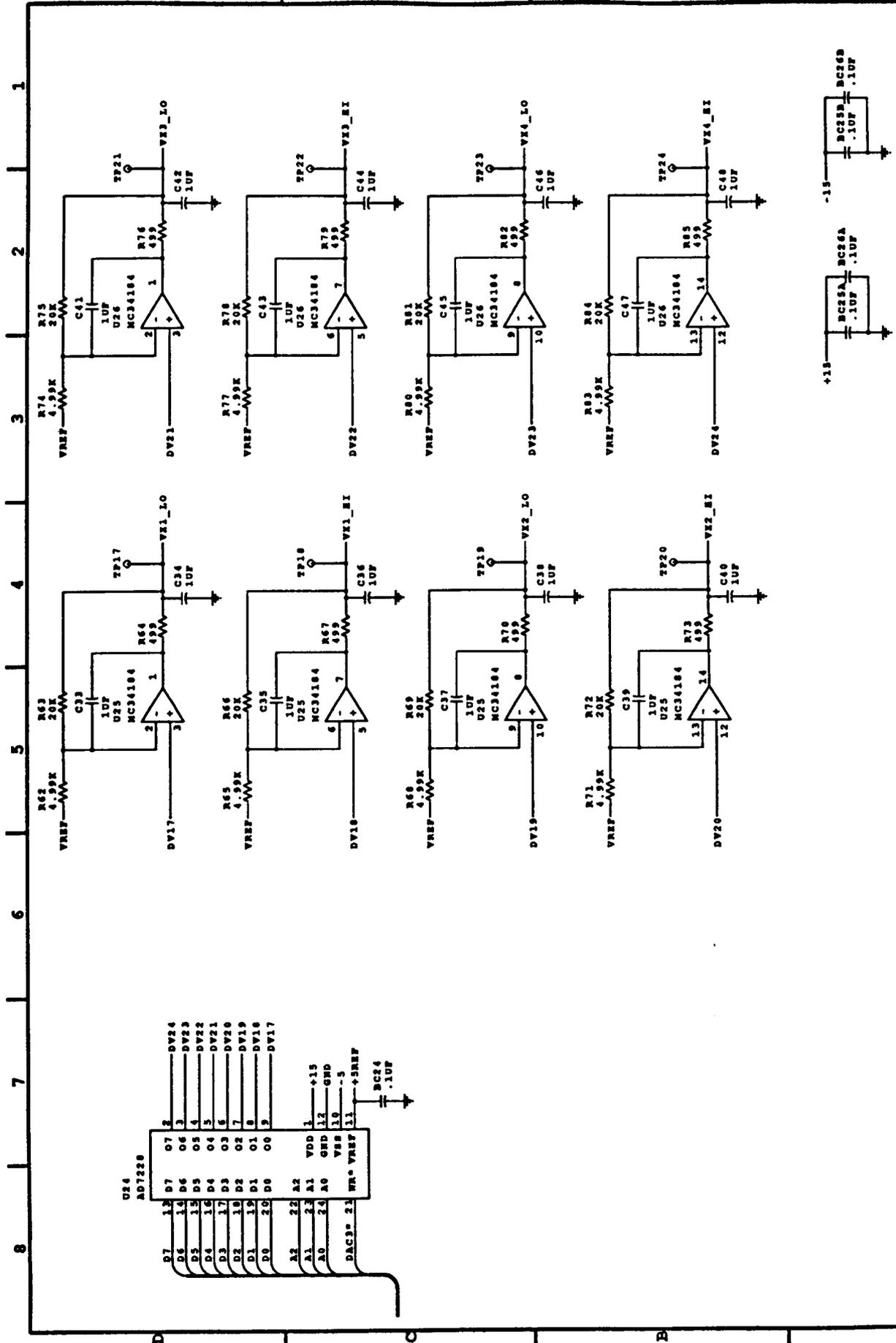
SHEET OF

00-D7

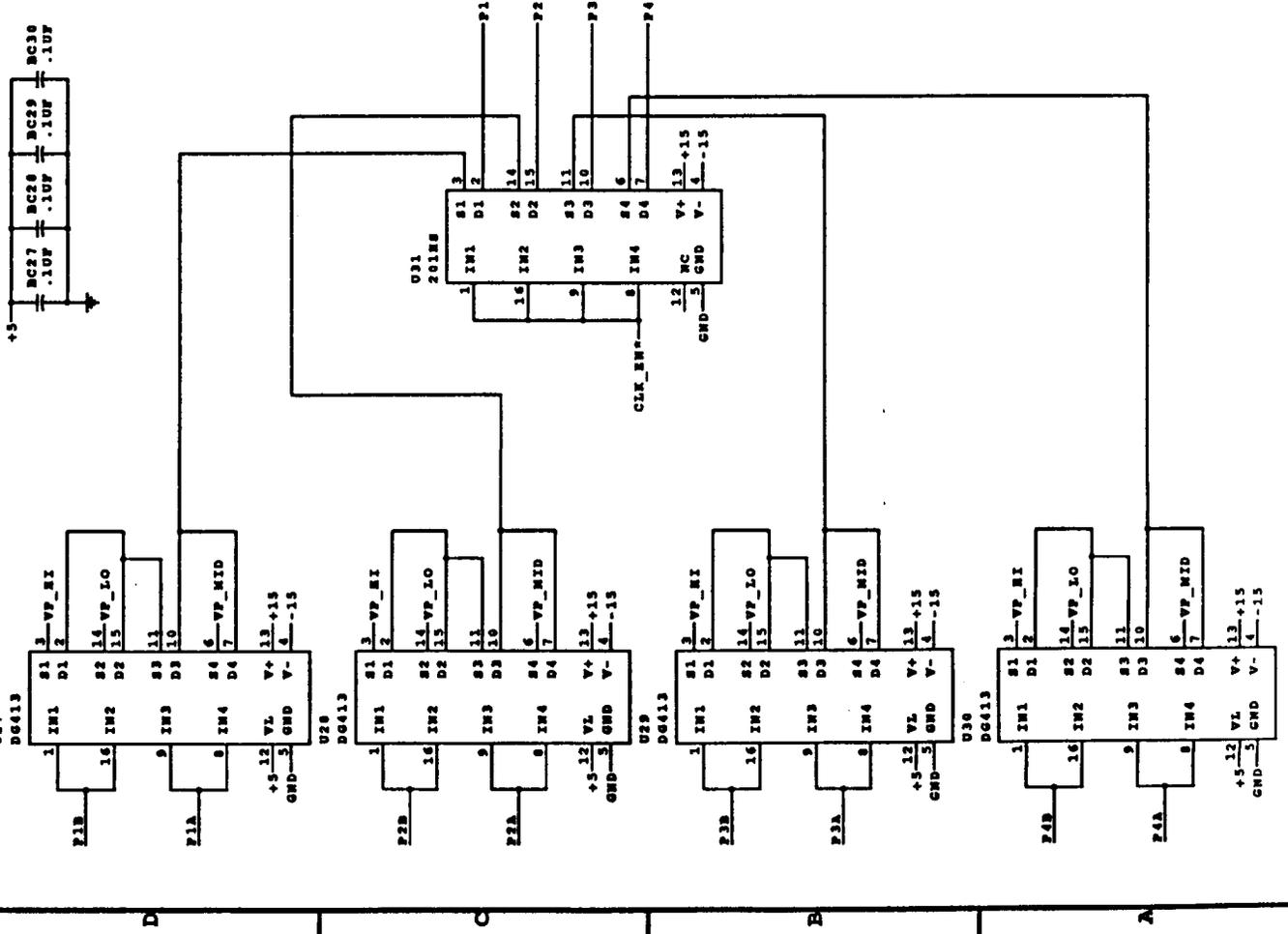
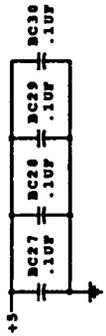
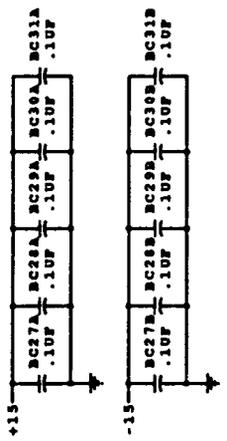
LF 347's

1 2 3 4 5 6 7 8

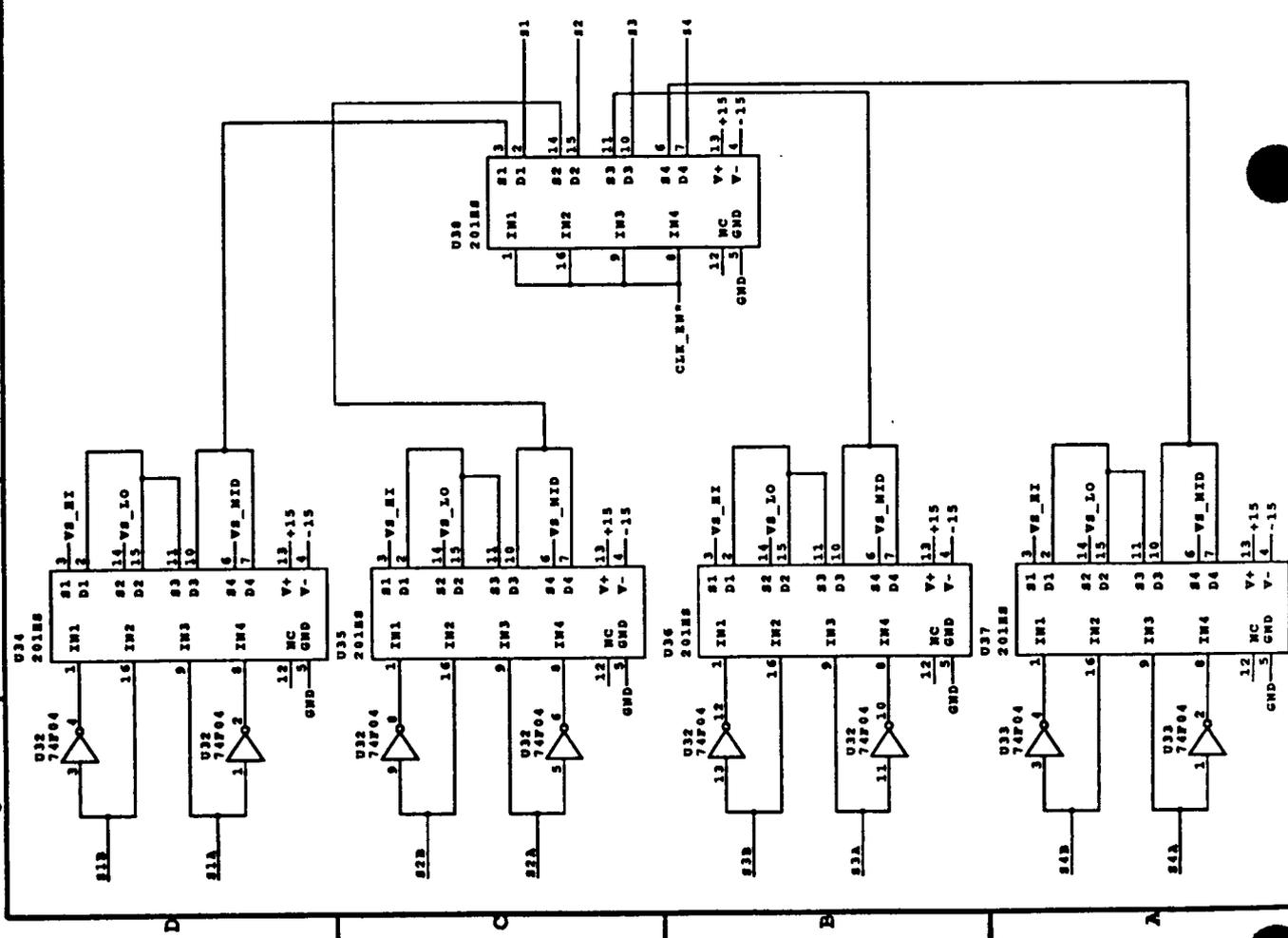
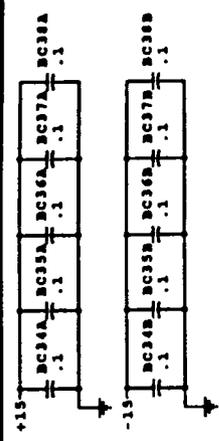




8 7 6 5 4 3 2 1

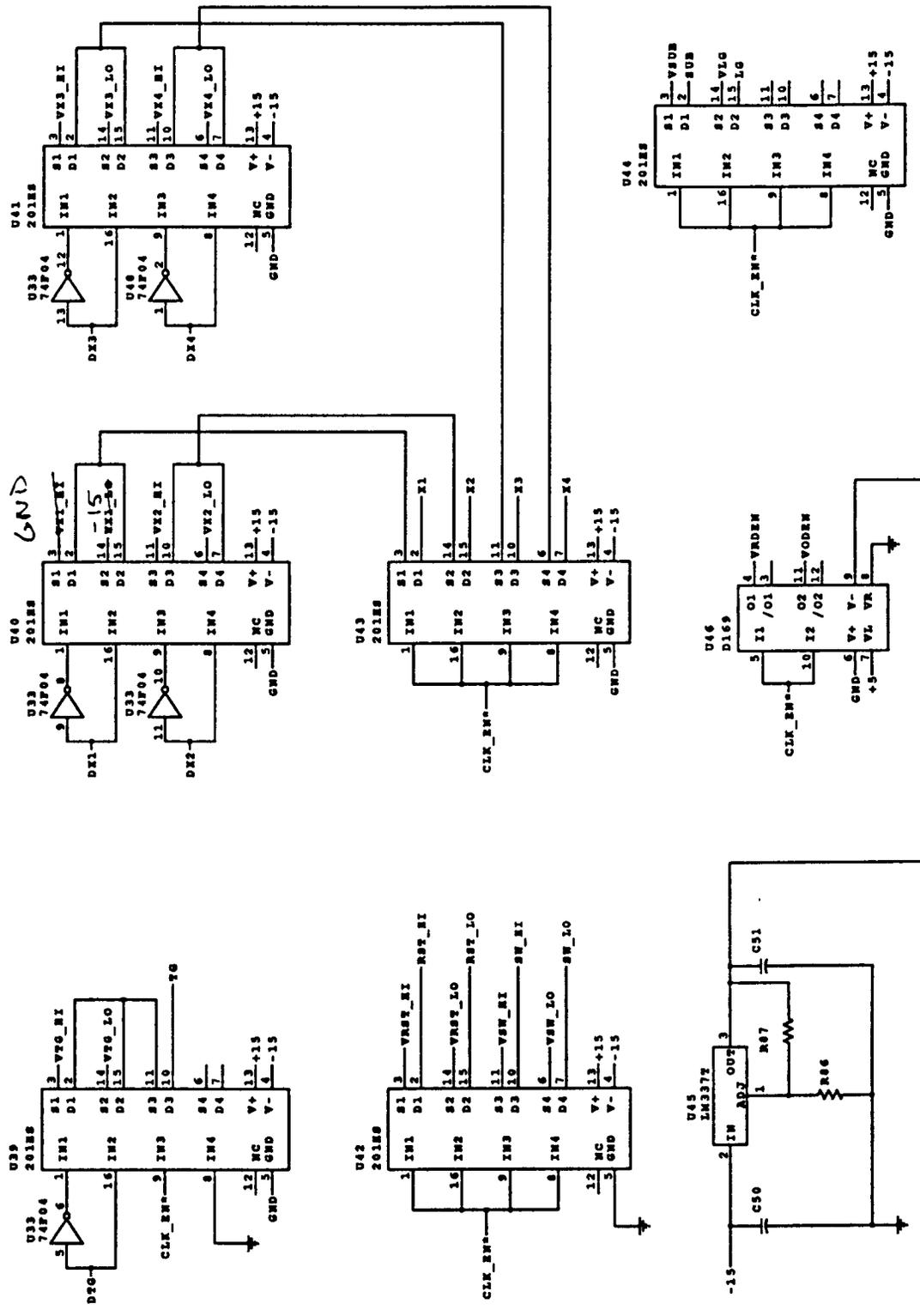


1 2 3 4 5 6 7 8

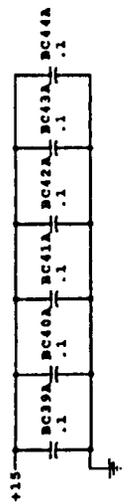
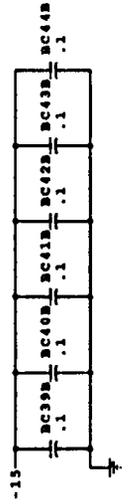


C-2

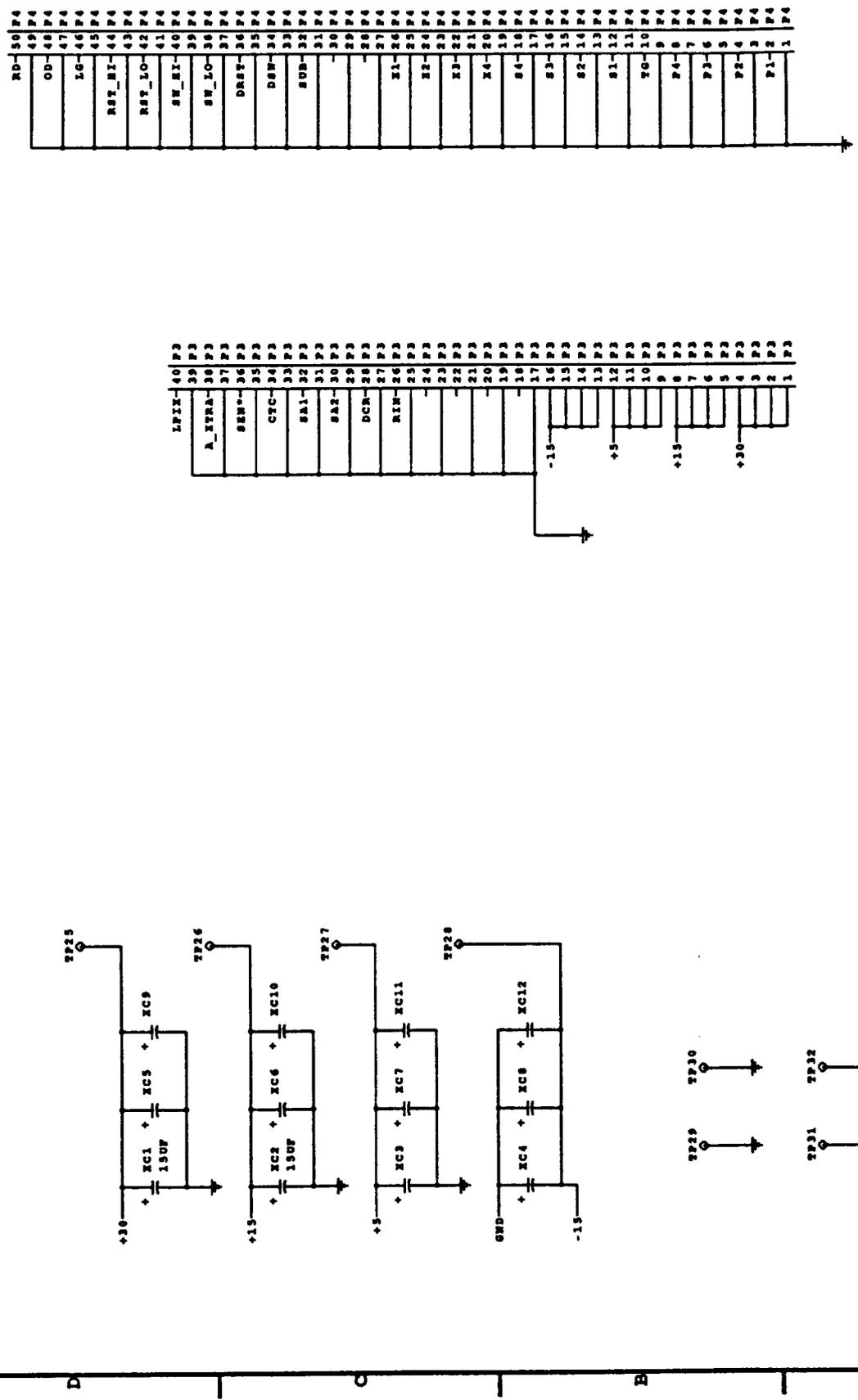
1 2 3 4 5 6 7 8



PHOTOMETRICS LTD.
2010 W. FORBES BLVD. TUCSON, AZ. 85745
DATE: 1/31/91
REV 1A
SHEET 1 OF 9



1 2 3 4 5 6 7 8

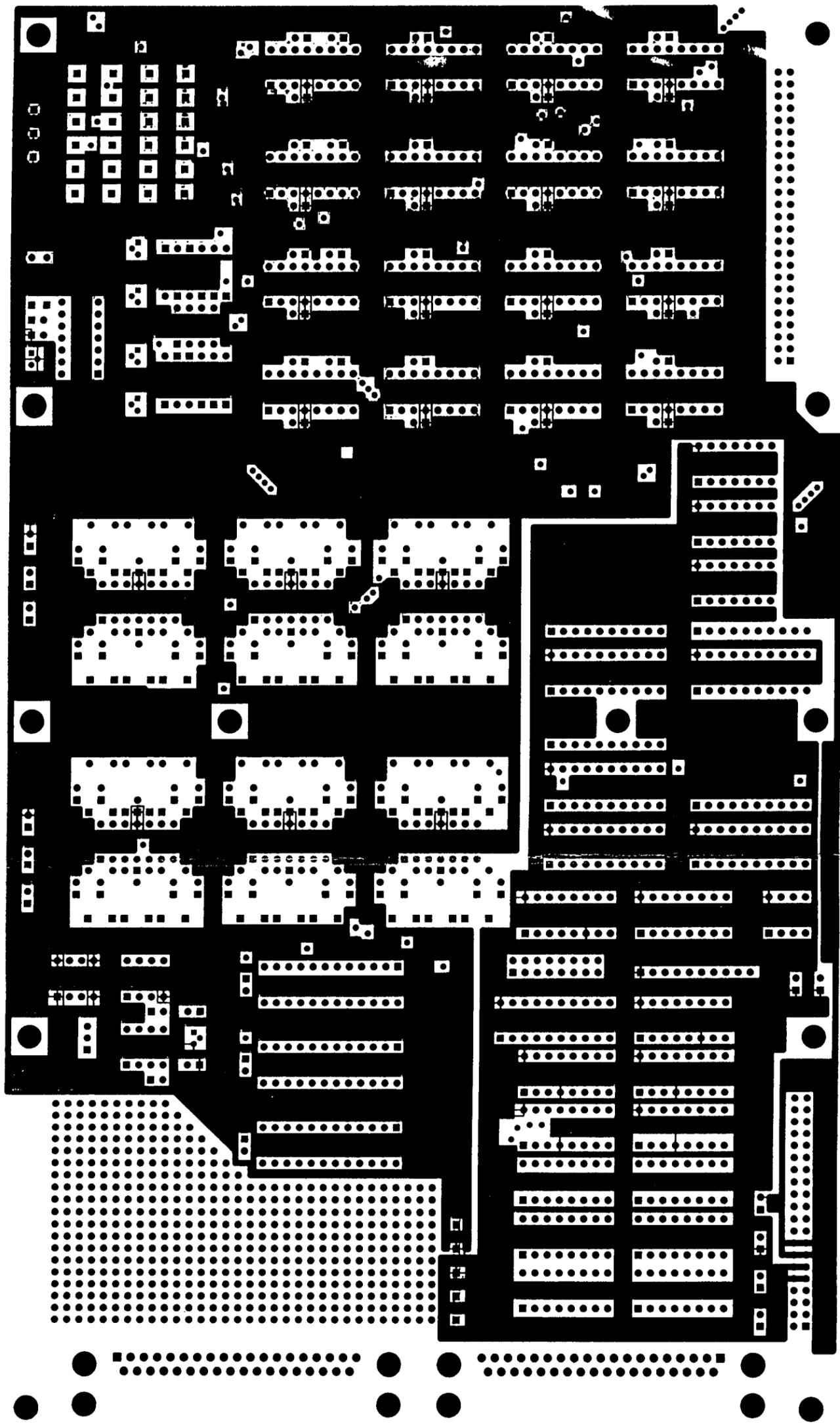


RD-30	24
45	24
OD-48	24
47	24
LG-46	24
45	24
RST_HI-44	24
43	24
RST_LO-42	24
41	24
SW_HI-40	24
39	24
SW_LO-38	24
37	24
DREF-36	24
35	24
D5W-34	24
33	24
SUB-32	24
31	24
30	24
29	24
28	24
27	24
26	24
25	24
24	24
23	24
22	24
21	24
20	24
19	24
18	24
17	24
16	24
15	24
14	24
13	24
12	24
11	24
10	24
9	24
8	24
7	24
6	24
5	24
4	24
3	24
2	24
1	24

LP2X-40	23
39	23
A_XTRA-38	23
37	23
SRB-36	23
35	23
CTC-34	23
33	23
SAL-32	23
31	23
SXZ-30	23
29	23
DCR-28	23
27	23
RIM-26	23
25	23
24	23
23	23
22	23
21	23
20	23
19	23
18	23
17	23
16	23
15	23
14	23
13	23
12	23
11	23
10	23
9	23
8	23
7	23
6	23
5	23
4	23
3	23
2	23
1	23

4.3.7 Clock Generator Card PCB Artwork

MID LAYER 3



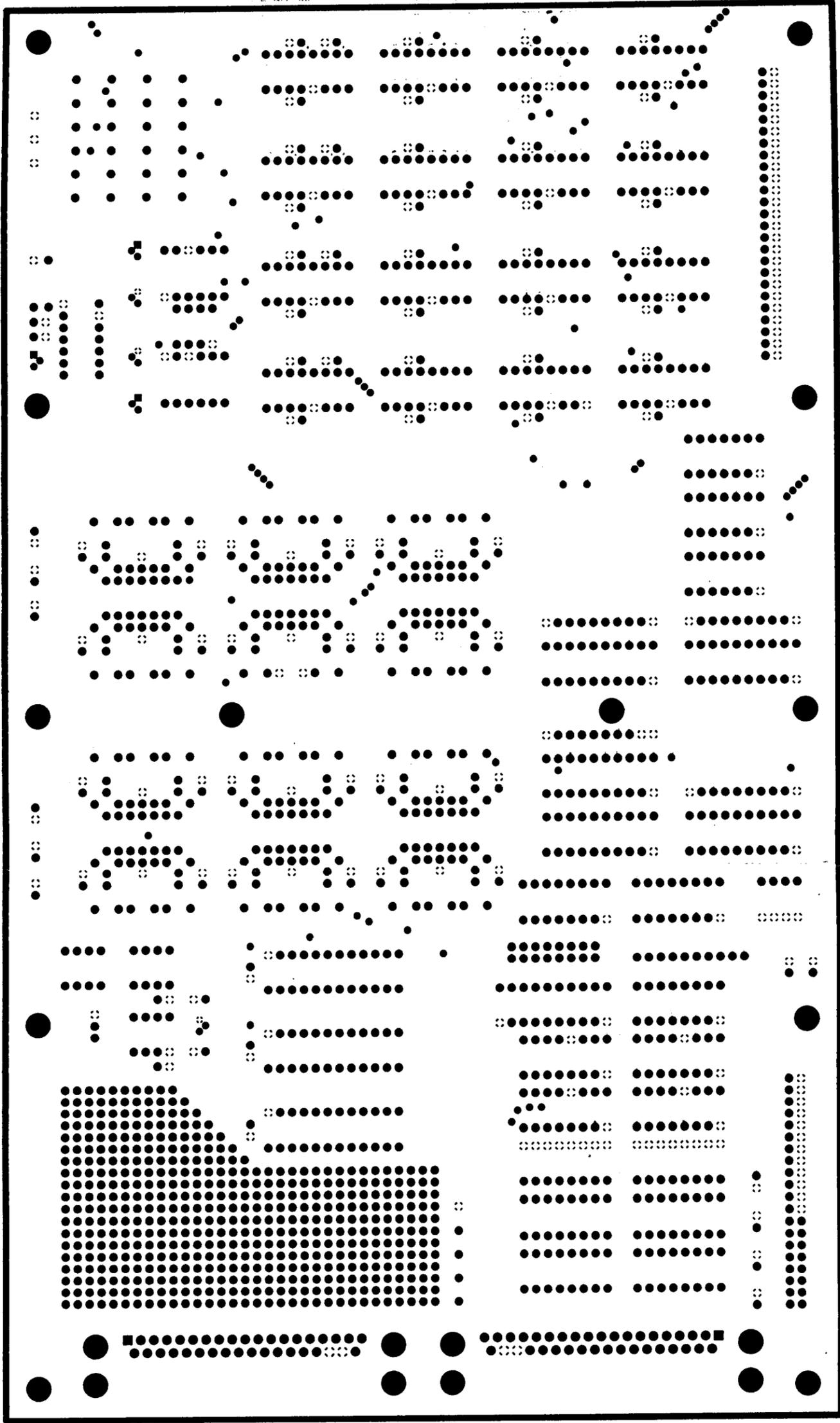
FOLDOUT FRAME

1. 2

FOLDOUT FRAME

2.

GROUND PLANE



FOLDOUT FRAME

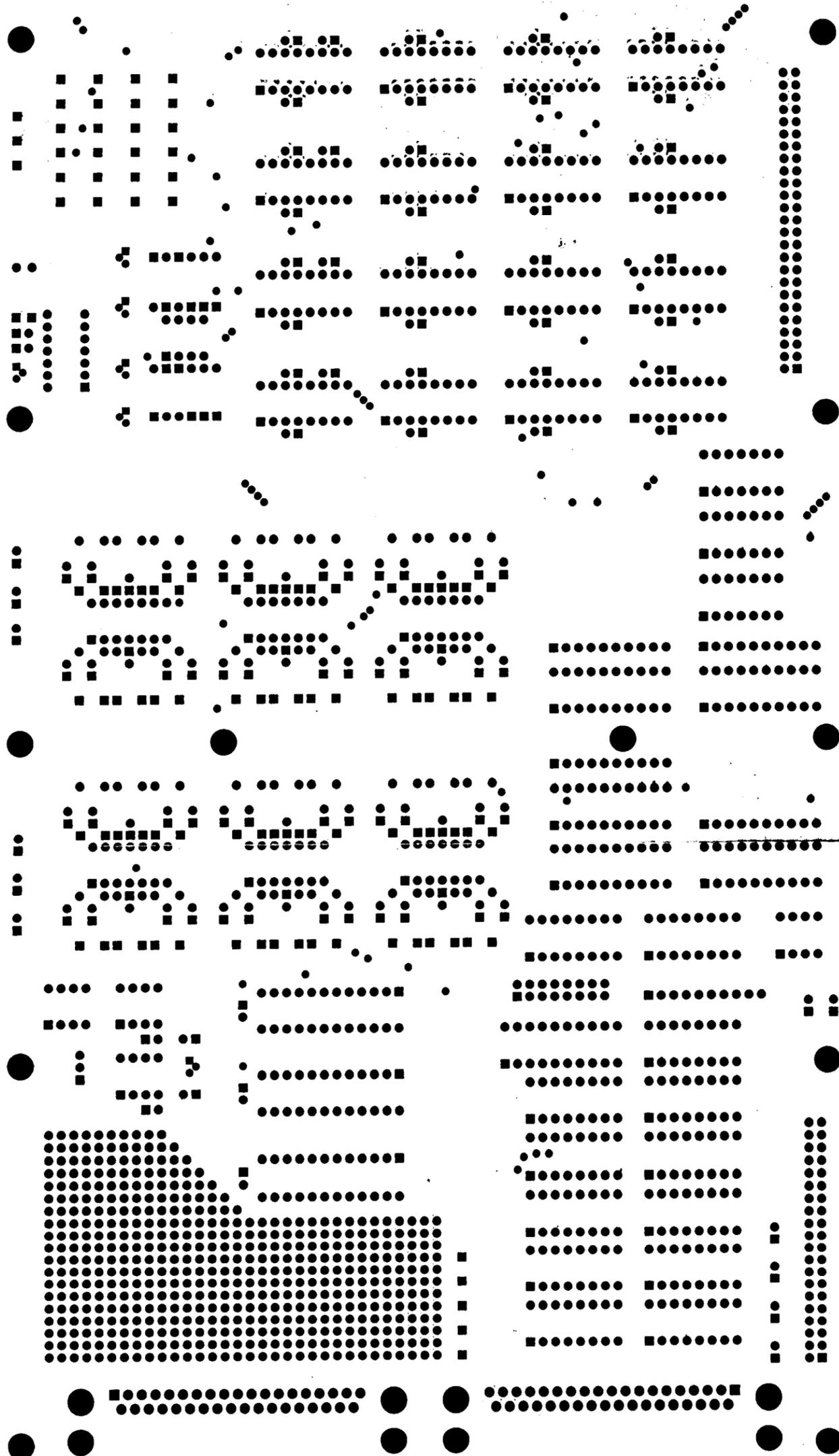
1:

22

FOLDOUT FRAME

2,

SOLDER MASK



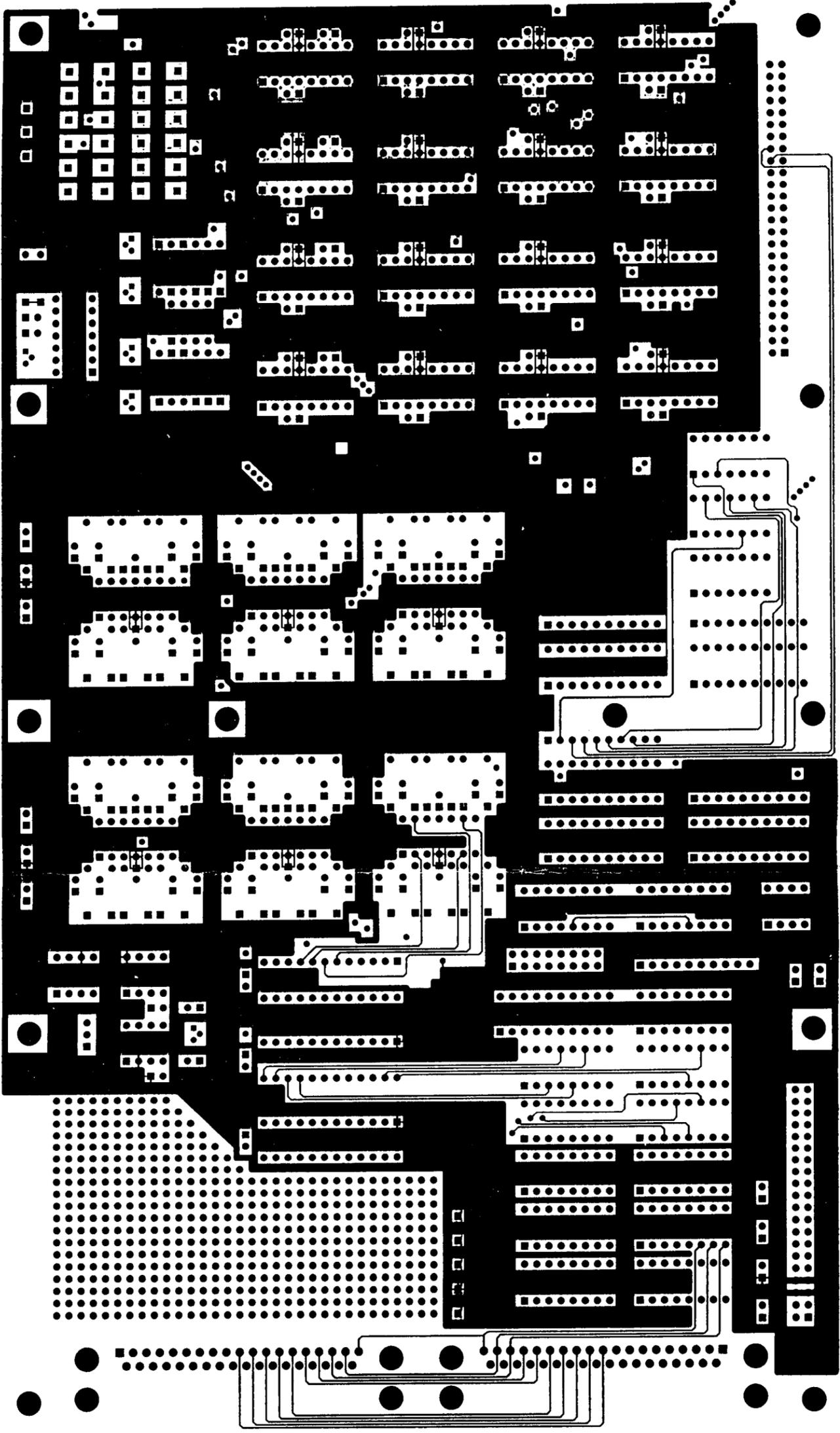
FOLDOUT FRAME

2

FOLDOUT FRAME

23

MID LAYER 2



FOLDOUT FRAME

FOLDOUT FRAME

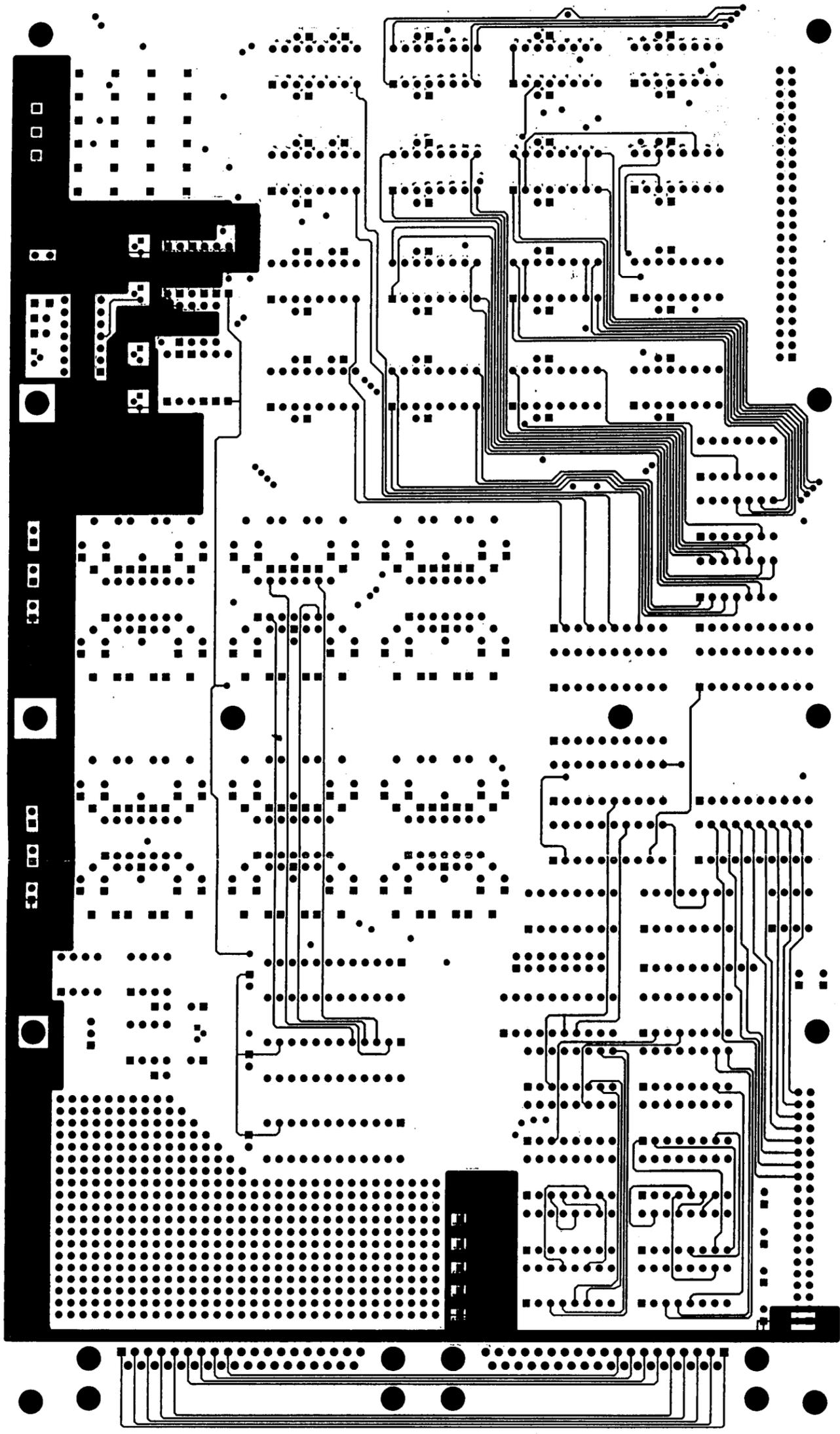
2

ht

1

3

MID LAYER 1



FOLDOUT FRAME

FOLDOUT FRAME

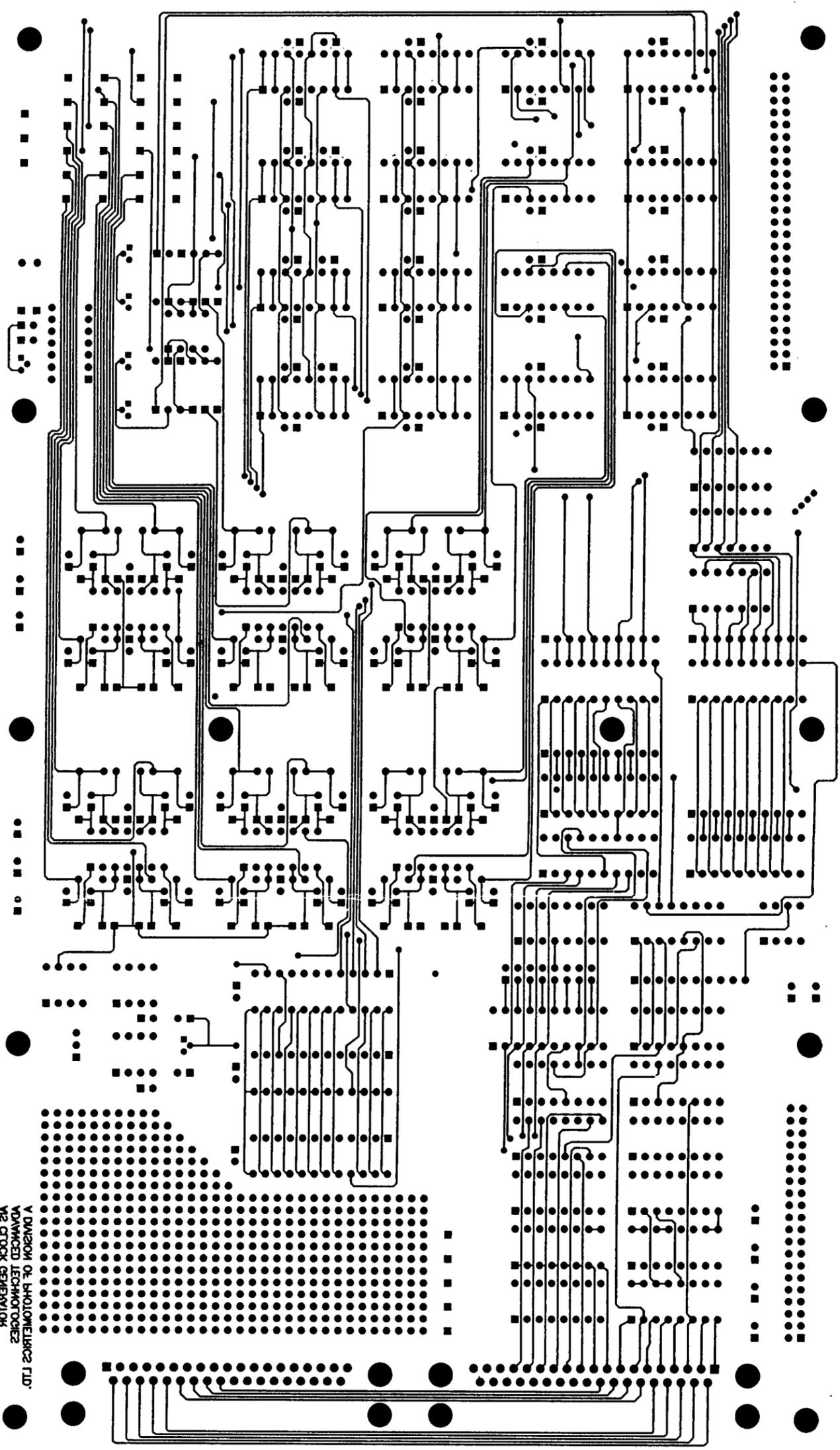
2

1

2

SOLDER SIDE

FEB 1980
A DIVISION OF PHOTOMICRONICS LTD.
ADVANCED TECHNOLOGIES
MS CLOCK GENERATOR

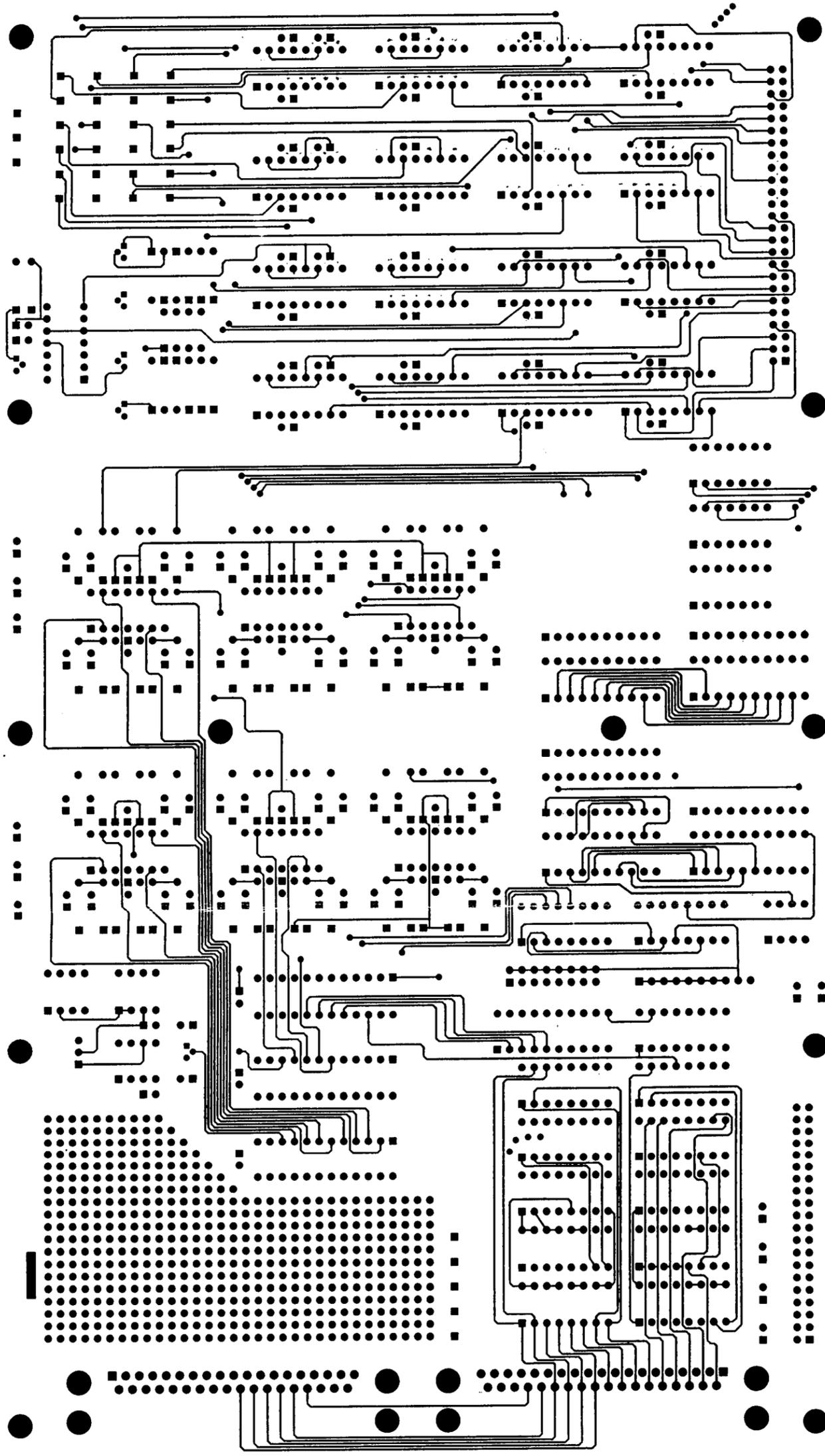


FOLDOUT FRAME

FOLDOUT FRAME

26

COMPONENT SIDE



FOLDOUT FRAME

2.

FOLDOUT FRAME

27

4.4 Analog Processor Card

The analog processor consists of several distinct parts. There is the analog processing circuitry itself, there is temperature regulation circuitry, and there are CCD clock signal buffers. Each part will be discussed in turn below.

4.4.1 CCD Video Processing Circuitry

The CCD video signal enters the clock analog module on pins 19 and 37 of P5, a DB37 that connects to the feed through connector on the side of the camera head.

The signal is passed first to a differential amplifier composed of U1, an AD842 from Analog Devices, and resistors R1, R2, R3, and R4. This differential amplifier eliminates common mode signals between the camera head electronics and the analog processor circuitry. Jumpers J1 and J2 allow one to ground both inputs to this amplifier to test the noise floor of the processor alone. This differential amplifier has a gain equal to $R2/R1$ and $R1=R3$ and $R2=R4$. This gain was set to 1.37.

The signal from the differential amplifier next is fed to the DC restore amplifier, which clamps the reference signal of each pixel to ground level. The gain of this stage is equal to:

$$1 + ((R7 \cdot R7A) / (R7 + R7A)) / R6$$

R7A is not normally needed and is included in the circuit only as a convenient way to adjust the gain of the system. When the STIS AIS1 camera was delivered to NASA, this component was not installed.

After the DC restore section the signal branches, with one half going through a precision inverter. This inverter is constructed using a low drift op-amp and .01% precision low temperature coefficient resistors. The accuracy and stability of this inverter are very important in the overall performance of the circuit.

The DC restored signal and the inverted signal are then fed into the integrator. The integrator is built around U8. JFET transistors Q1 and Q2 are used to selectively connect the video signals to the integrator input. These switches are controlled by TTL lines named SA1 and SA2 on the schematic diagram. Because these switches require a voltage swing of -15 to 0 volts, an analog switch, U5, is used to convert the TTL level control signals to the proper scale.

The integrator includes a DC offset adjustment. In addition to serving as a way to balance the offsets of the data from the two CCD

ports in the system, the offset adjustment circuit also adjusts the video signal to match the -5v to +5v input range on the analog to digital converter. Reference voltage U6 provides 10.0 volts to the top of the voltage divider formed by R13, VR1, and R14. The DC voltage at VR1's wiper contact is fed to the input of the follower formed around U7. The resistive components included in this circuit are of the low temperature coefficient type for stability. The output of the follower is switched across the capacitor by closing an analog switch. When the switch is closed, the output of the integrator is charged with a time constant equal to $(R17 * C6)$. R16 and R17 must be of equal value. In order to assure accuracy to 16 bits, the pre charge time, when the switch is closed, should be at least 12.5 times the time constant $(R21 * C23)$. This period is referred to as 'integrator reset' in parts of this document.

The output of the integrator is connected to the input of the analog to digital converter, U14, an AM30516 hybrid converter module from Analogic. This converter requires approximately 5 usec to perform a conversion. During the conversion, the integrator switches are all open and the charge on the capacitor is held. No sample and hold amplifier is used. The converter does not contain a sample and hold. The conversion takes place after the signal 'command to convert' or 'CTC' is received.

The digital data produced by the conversion is fed into a shift register to prepare for serial transmission. The data are in complementary binary form. A PLD, U13, is used to generate the clock signal required to transmit the serial data stream. The serial data and clock are then fed both to a MC3487 RS422 transmitter and to a pair of fiber optic drivers, with ST type connections. Both transmitters are driven at all times. There is a jumper on the VMEbus interface to determine which interface is being used.

4.4.2 Temperature Regulation Circuitry

Another large portion of the circuitry on the analog card is designed to regulate the temperature of the CAM and, more specifically, the analog to digital converter.

A platinum wire RTD is used to sense the temperature of the analog to digital converter. This RTD is controlled by the current source formed by it and U1a, R80, R81, R82, and VR4. The nominal current of 1 mA through the RTD may be adjusted via VR4.

The voltage drop across the RTD is sensed by U33, an instrumentation amplifier. The gain of the amp is set to 100. The measured temperature is then scaled by the amplifier built from

U32d, R83, R84, and R85 to 0.01v/degree Celsius. The measured temperature is then connected to P6 and then to a meter mounted on the CAM chassis near the fan.

A 'desired temperature' is set by adjusting VR5. The voltage produced is compared to the actual temperature by the differential amplifier formed by U34a, R90, R91, R92, and R93. This 'set point' may be measured at pin 5 of U32. At this location it has a scale of 0.01v per degree Celsius.

The error signal produced by the differential amplifier is then fed to two circuits. One of these, as the difference signal becomes more negative, increases the current through a large resistor used as a heat source. The other, as the difference becomes more positive, increases the voltage applied to a small DC fan that draws air in through the vent located beneath the analog card, around the boards, and out through the fan opening, located beneath the clock card on the outside of the clock/analog module. When adjusted for a temperature approximately 10 degrees Celsius above the ambient temperature, the circuitry will readily stabilize to a point where neither the heater nor fan are running very strongly.

4.4.3 CCD Clock Signal Buffers

All of the CCD clock signals and DC reference voltages that connect from the clock card to the camera head pass through the analog card. They arrive from the clock card on P4. The pinout for P4 is shown below.

Pinout of Analog Processor Card P4

2	parallel clock 1	28	unused
4	parallel clock 2	30	unused
6	parallel clock 3	32	substrate (unused)
8	parallel clock 4	34	reset pulse
10	transfer gate	36	summing well pulse
12	serial clock 1	38	summing well low
14	serial clock 2	40	summing well high
16	serial clock 3	42	reset low
18	serial clock 4	44	reset high
20	extra clock 4	46	last gate
22	extra clock 3	48	output drain
24	extra clock 2	50	reset drain
26	extra clock 1		

The odd numbered contacts on P4 are unused.
Those wires are connected to ground on the clock card.

The serial clocks, parallel clocks, and transfer gate clock are all buffered on the analog card using LM6321N buffers from National Semiconductor. These buffers provide most of the current required to drive the CCD, and they take that current directly from the +15v and -15v supplies on the analog card. The buffer circuits provide two other features. One is the resistive load to ground on the input. This pulls the buffer's output to a near ground potential when the ribbon cable is removed from the P4 connector or when the CCD clock switches have been disconnected by the analog switches on the clock card. Also incorporated into the CCD clock buffer circuitry is an RC filter network that may be used to adjust the CCD clock signal rise time. These resistors and capacitors are installed in sockets allowing for easy replacement should some other time constant be desired. Additionally, damping resistors may be placed on the buffer outputs to further control signal rise time. Parallel clock drivers are capable of driving 0.02 uF loads at 500 kHz. Serial clock drivers are capable of driving 500 pF loads at 1 MHz.

The CCD reset gate and summing well are controlled by analog switches located in the camera head. The TTL signals to drive the switches and the positive and negative rails for the gates are passed through the analog card. The TTL signals are available at test points for diagnostic purposes. The DC rail potentials are connected from the clock card at P4 to the camera head at P5 through a resistor network. One resistor ties the signal to ground, so that the voltage will reach a near ground potential when the source voltages are disconnected on the clock card or when the clock card is not connected to the analog card. Another resistor follows in series; these resistors are meant to serve as a part of an RC filter network with capacitors in the camera head. They are generally not needed or installed.

The CCD reference voltages, the output drain voltage, VOD, the reset drain voltage, VRD, and the last gate voltage, VLG, are also connected to the camera head through resistor networks such as those described in the previous paragraph.

Test points are provided for all the CCD clocks and reference voltages. In the case of the serial and parallel clocks as well as the transfer gate clock, the test point is located after the buffer amplifier.

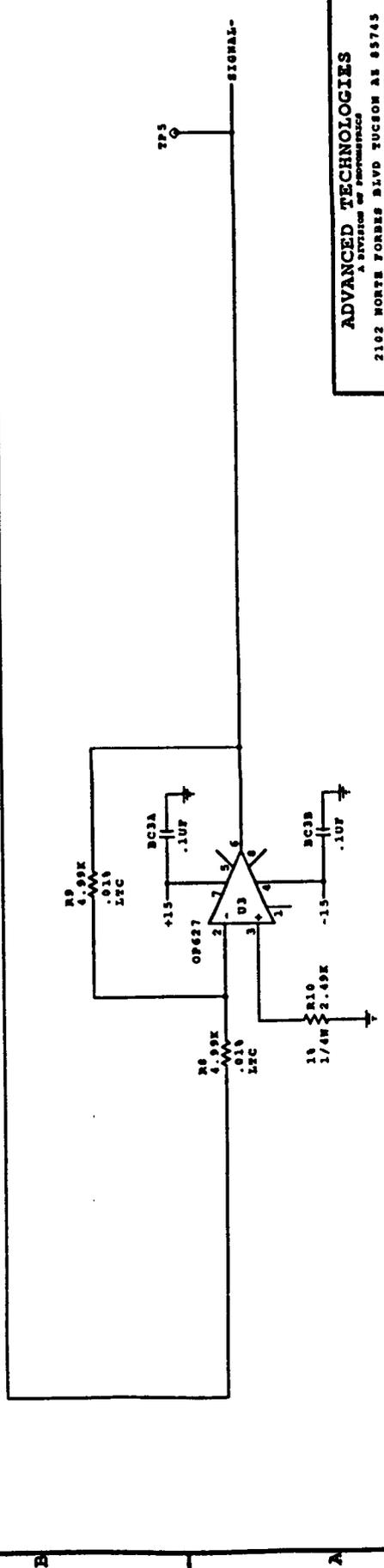
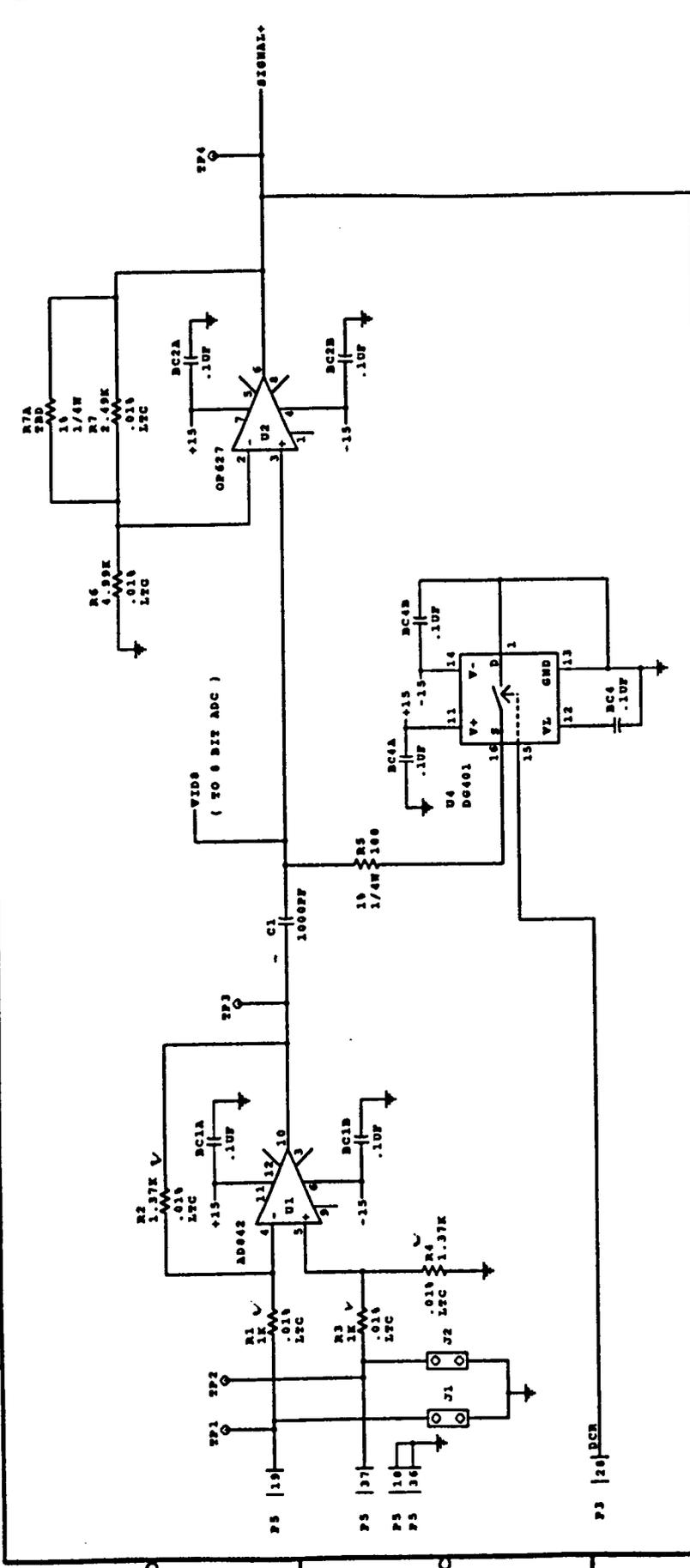
All the CCD clock signals and reference voltages connect to the camera head at P5 on the analog card. This connector plugs into a connector that feeds through the camera head CCD chamber wall. The pinout for this connector is shown below.

Pinout of P5 connector on AIS1 analog card

1	serial clock 1	20	serial clock 2
2	serial clock 3	21	serial clock 4
3	TTL reset	22	TTL summing well
4	extra clock 1	23	extra clock 2
5	extra clock 3	24	extra clock 4
6	parallel clock 1	25	parallel clock 2
7	parallel clock 3	26	parallel clock 4
8	transfer gate	27	
9		28	substrate (unused)
10	last gate	29	output drain
11	reset drain	30	reset gate high
12	reset gate low	31	summing well high
13	summing well low	32	
14		33	-15v
15	-15v	34	+15v
16	+15v	35	+28v
17	+28v	36	ground
18	ground	37	CCD video low
19	CCD video high		

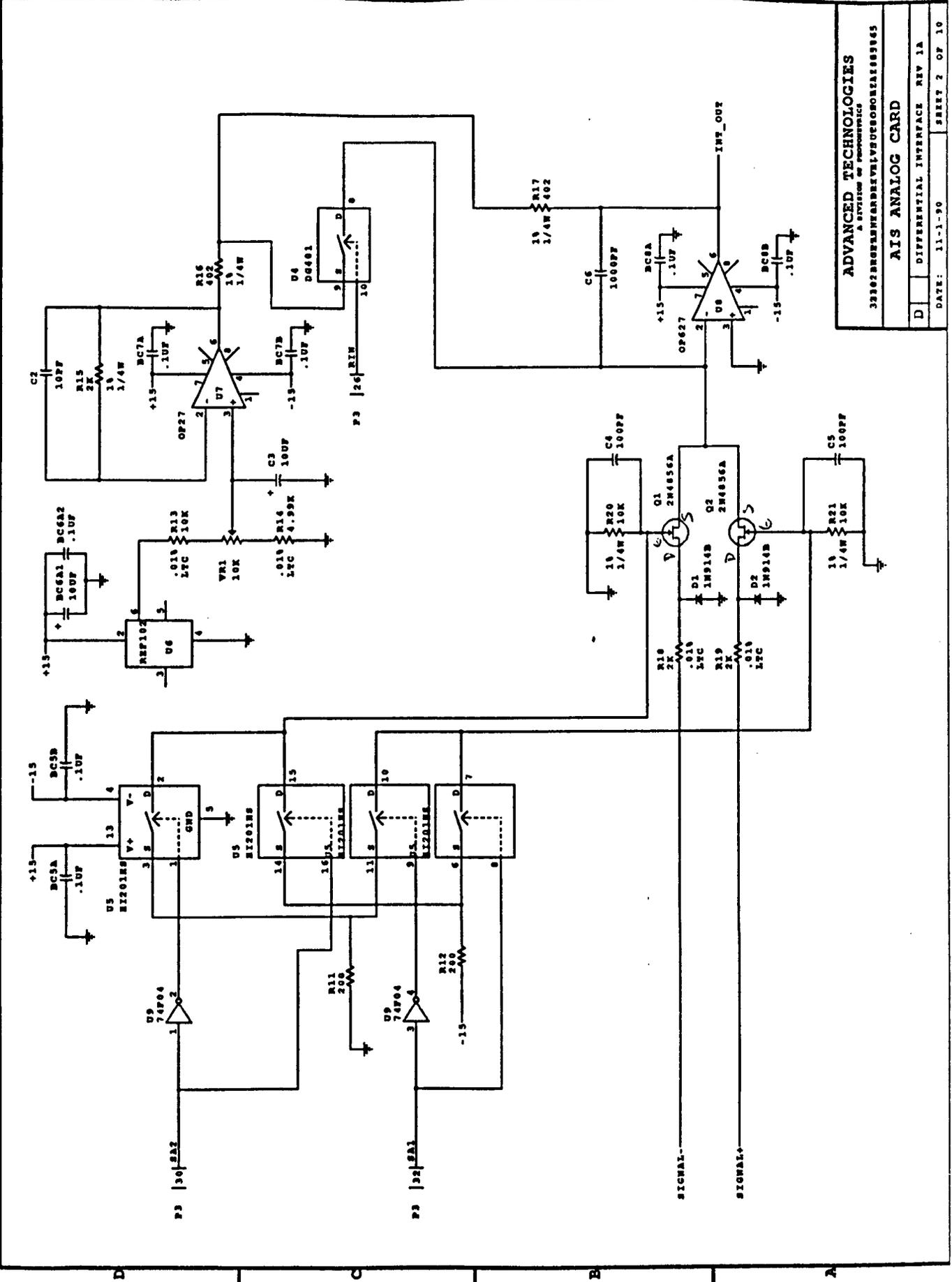
4.4.4 Analog Processor Card Schematic Diagram

1 2 3 4 5 6 7 8



ADVANCED TECHNOLOGIES
 A DIVISION OF PERMASCOR
 2102 MORRIS FORBES BLVD TUCSON AZ 85743
AIS ANALOG CARD
 DC RESTORE CENTER REV 1A
 DATE: 11-1-90 SHEET 1 OF 10

1 2 3 4 5 6 7 8



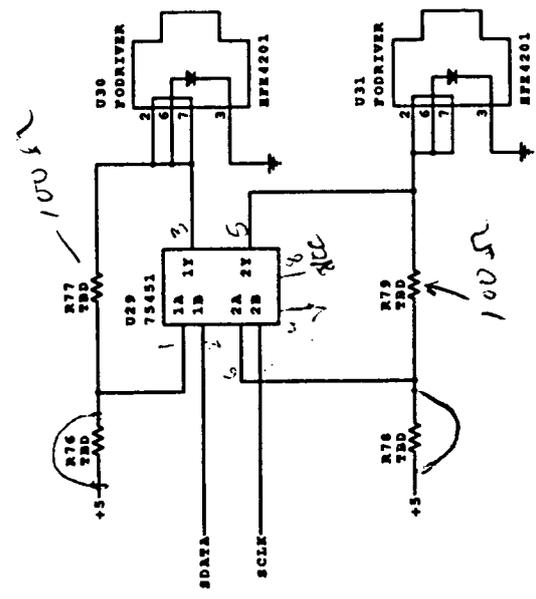
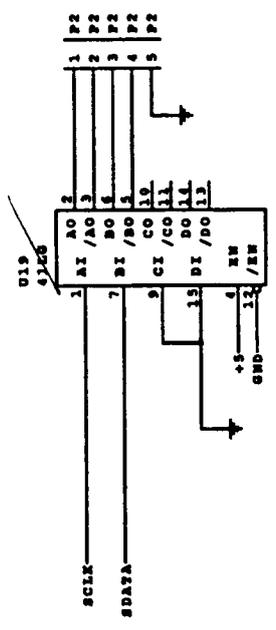
ADVANCED TECHNOLOGIES
 A DIVISION OF PERMUTIC
 32302 BROMHARDT BLVD SUITE 200
 BOSTON, MA 02128
 DATE: 11-1-90
 SHEET 2 OF 10

32302 BROMHARDT BLVD SUITE 200
 BOSTON, MA 02128
 DATE: 11-1-90
 SHEET 2 OF 10

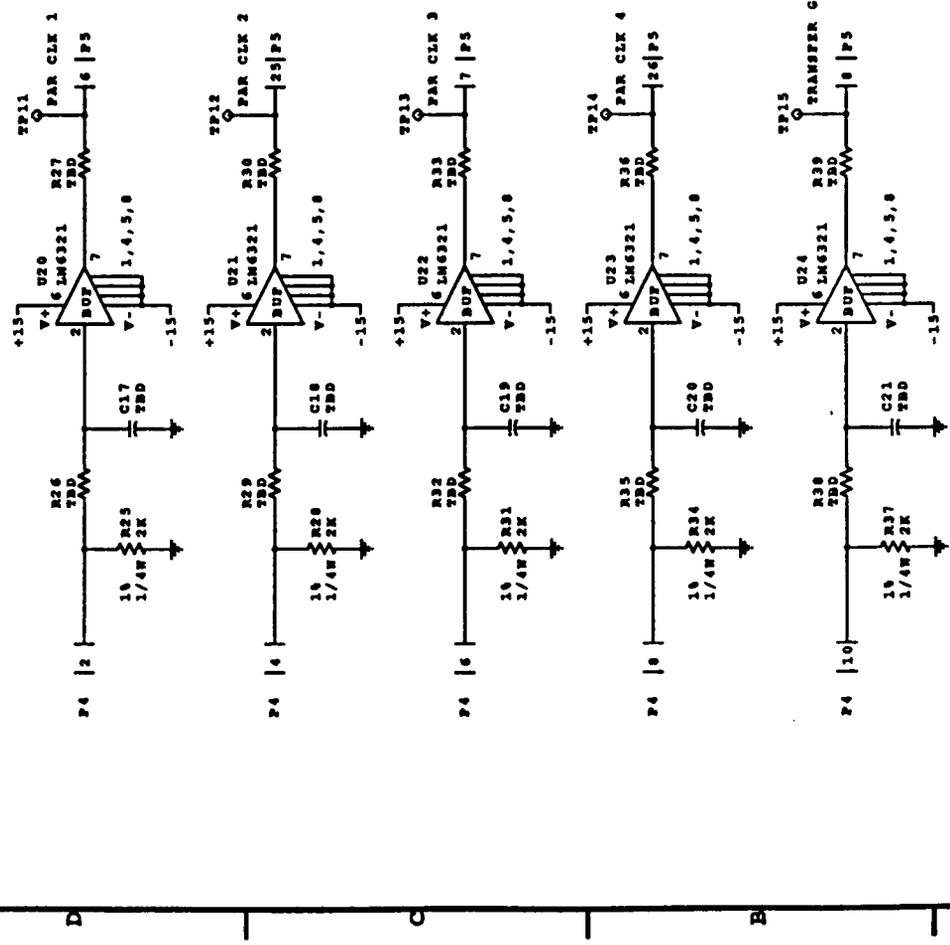
DIFFERENTIAL INTERFACE REV 1A

1 2 3 4 5 6 7 8

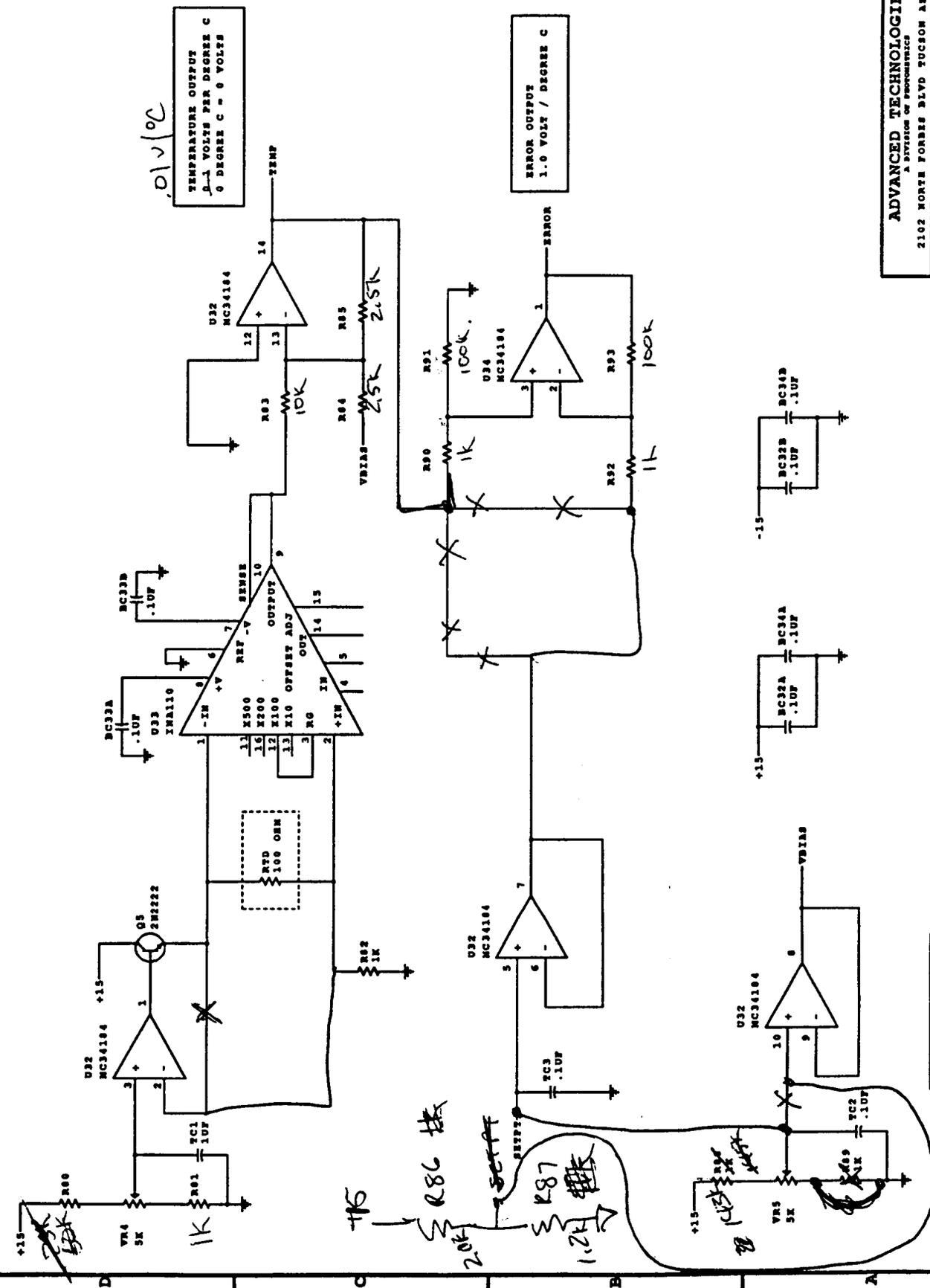
3487



8 7 6 5 4 3 2 1

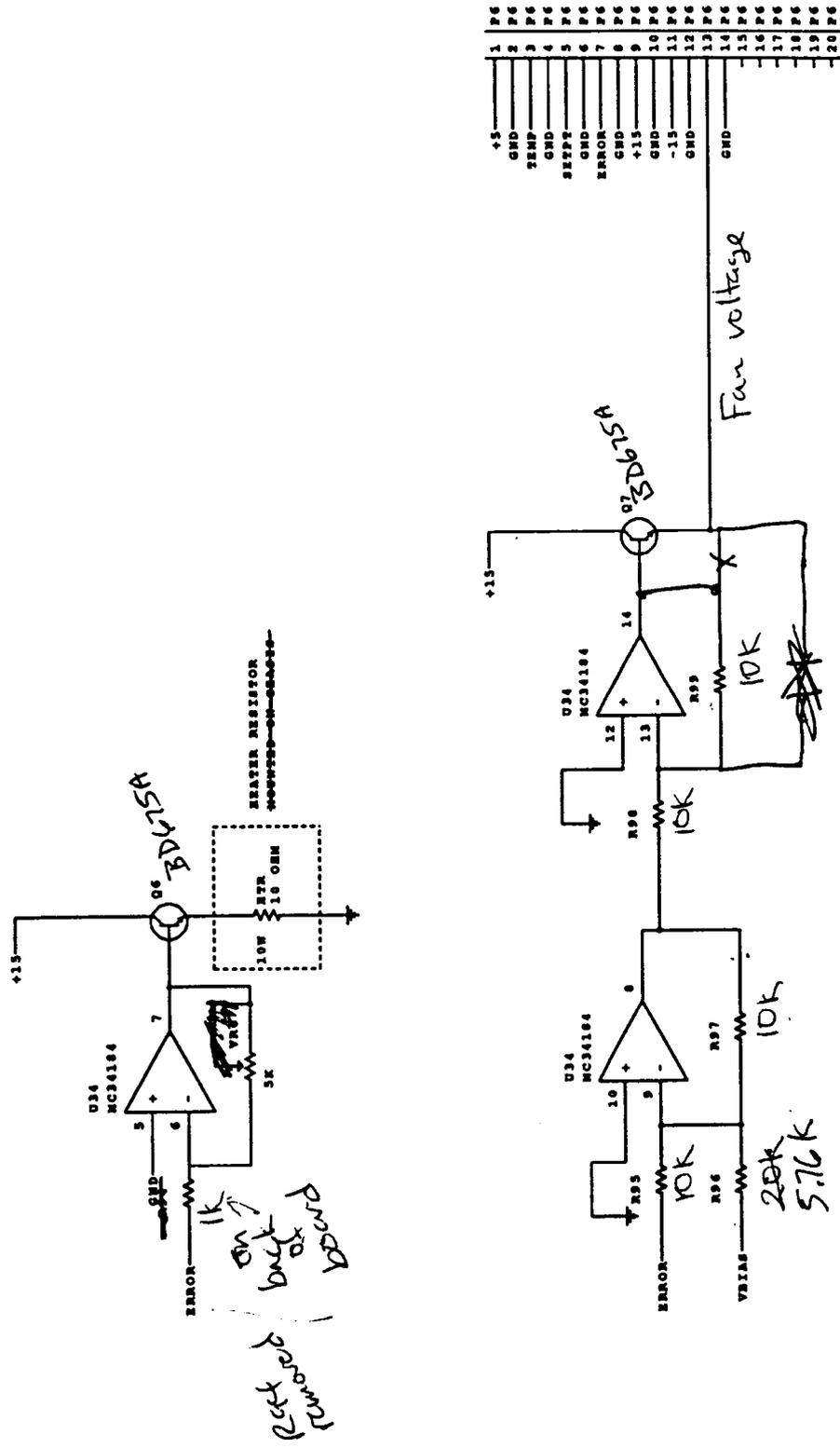


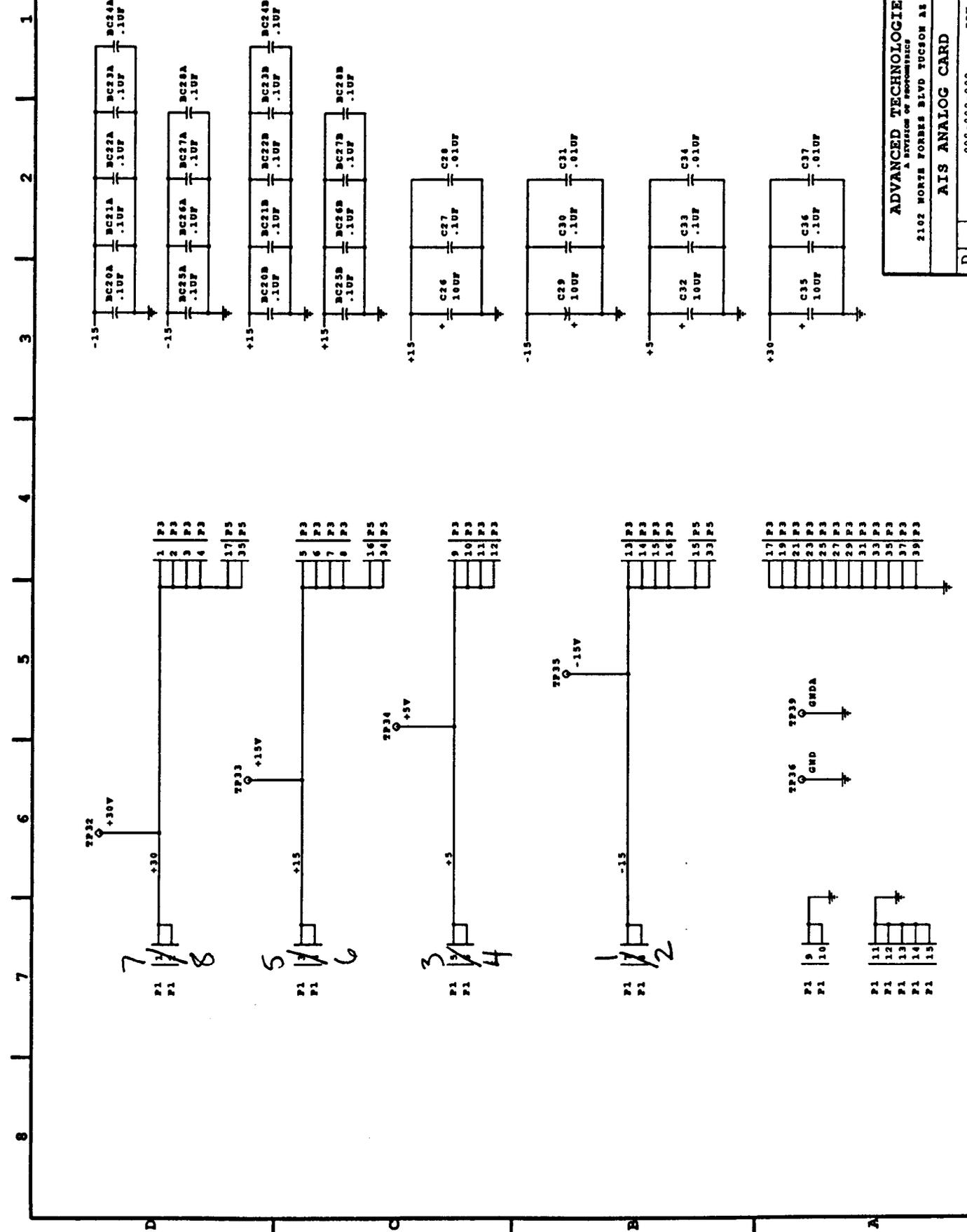
1 2 3 4 5 6 7 8



BIAS VOLTAGE = 9V

1 2 3 4 5 6 7 8





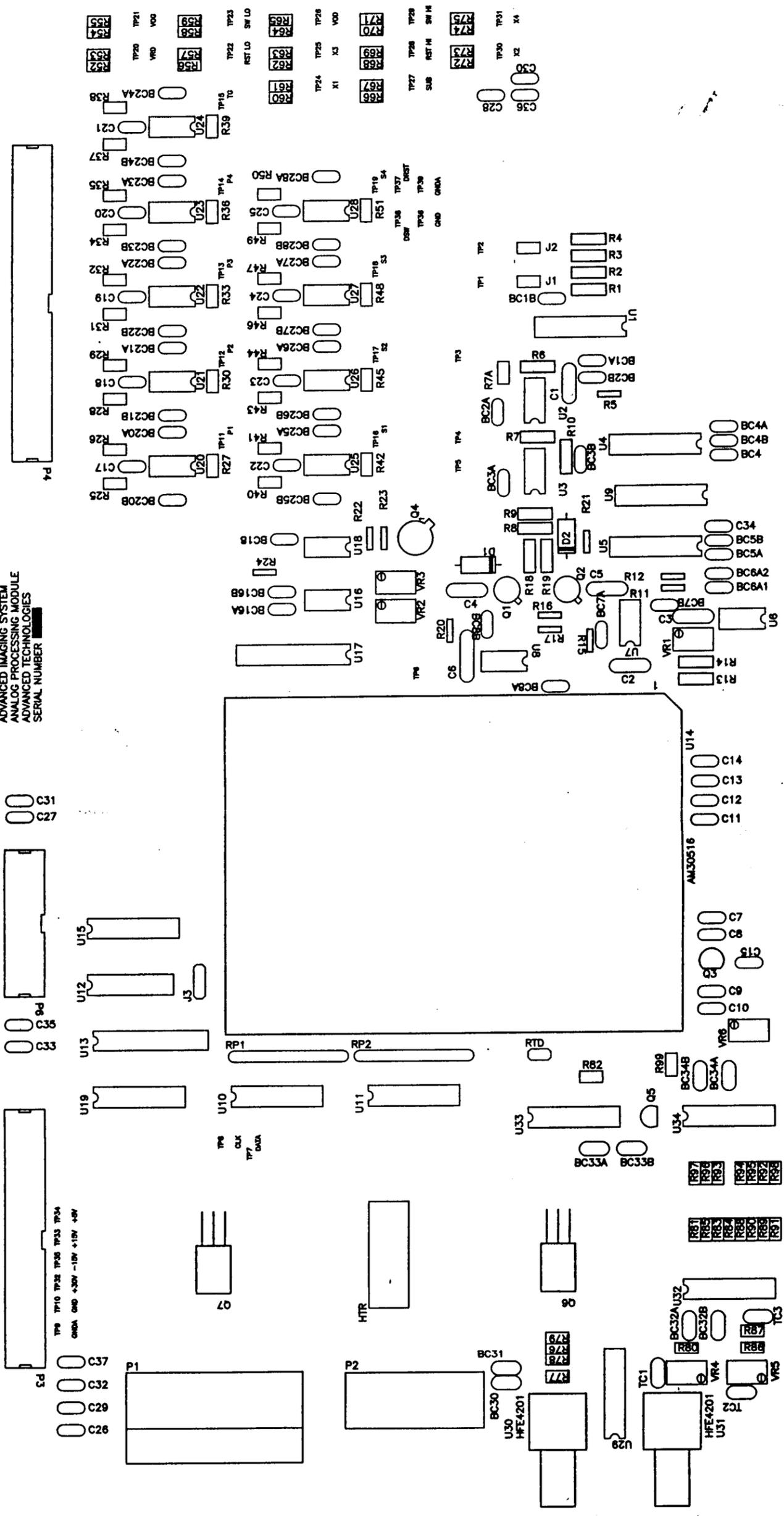
4.4.5 Analog Processor Card PCB Artwork



TOP SILKSCREEN

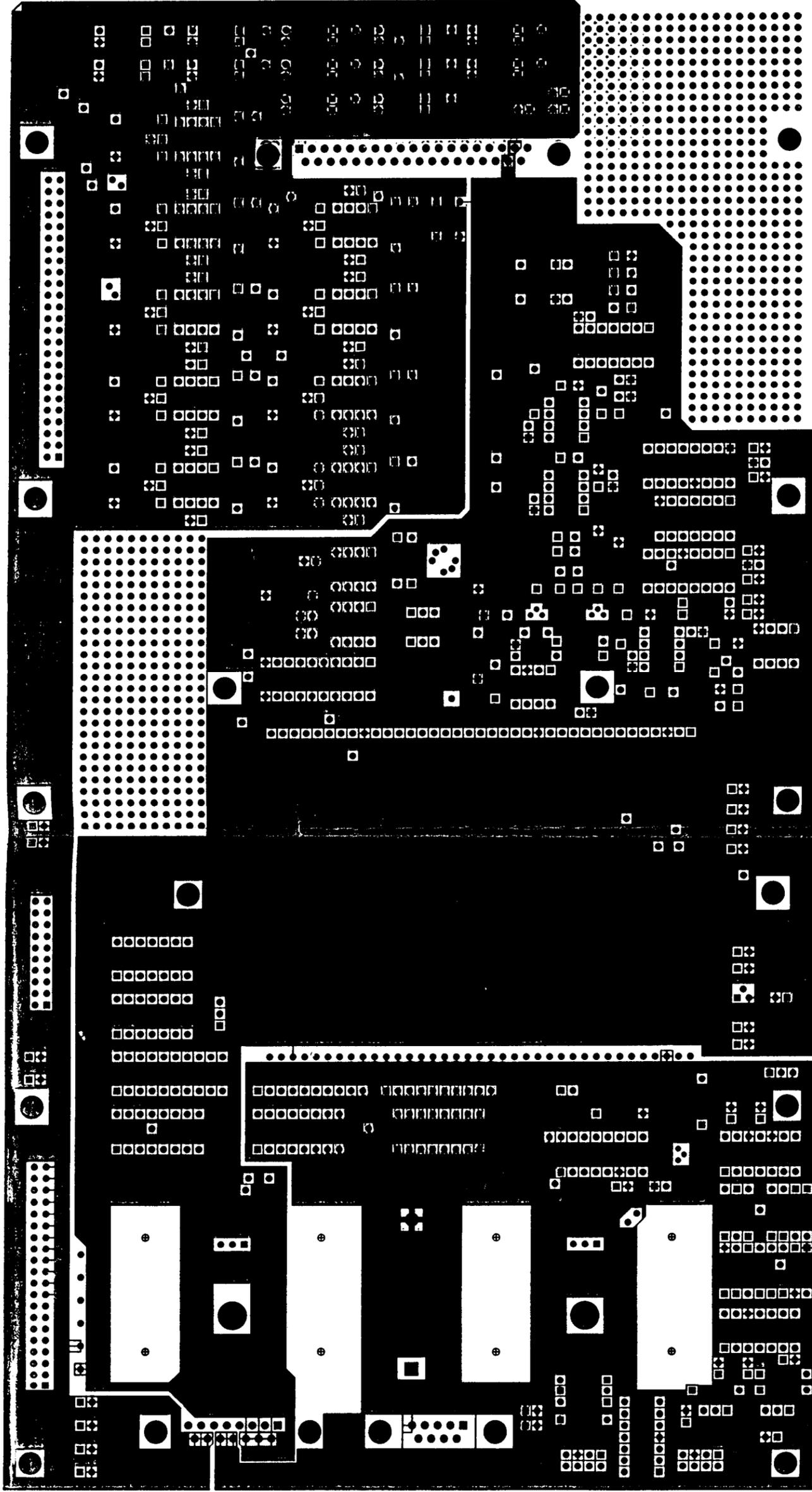


ADVANCED IMAGING SYSTEM
 ANALOG PROCESSING MODULE
 ADVANCED TECHNOLOGIES
 SERIAL NUMBER



FOLDOUT FRAME

COMPONENT SIDE



FOLDOUT FRAME

46

FOLDOUT FRAME

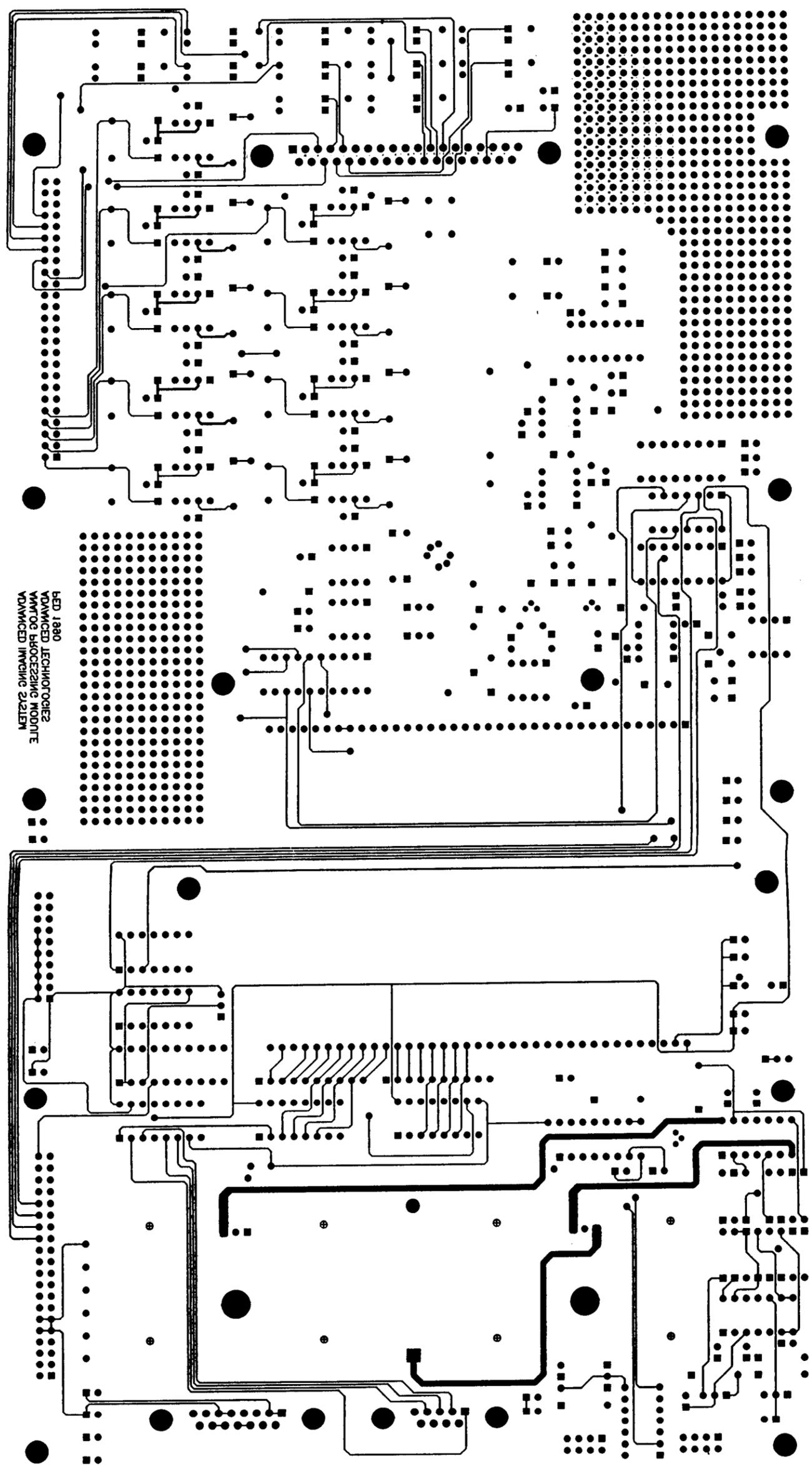
2

2.

FOLDOUT FRAME

SOLDER SIDE

RED 1880
ADVANCED TECHNOLOGIES
ADVANCED PROCESSING MODULE
ADVANCED IMAGING SYSTEM



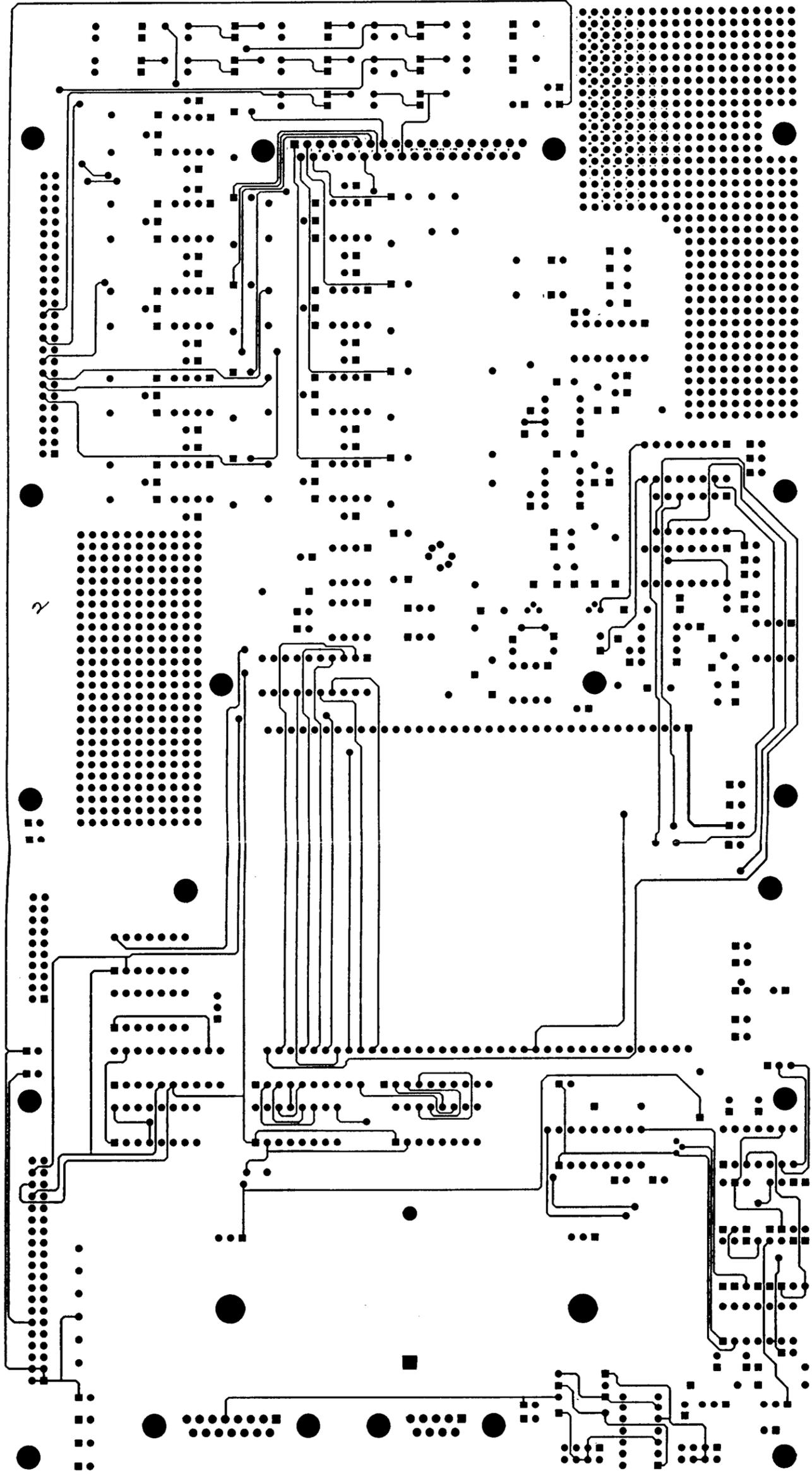
FOLDOUT FRAME

2

FOLDOUT FRAME

48

MID LAYER 1



FOLDOUT FRAME

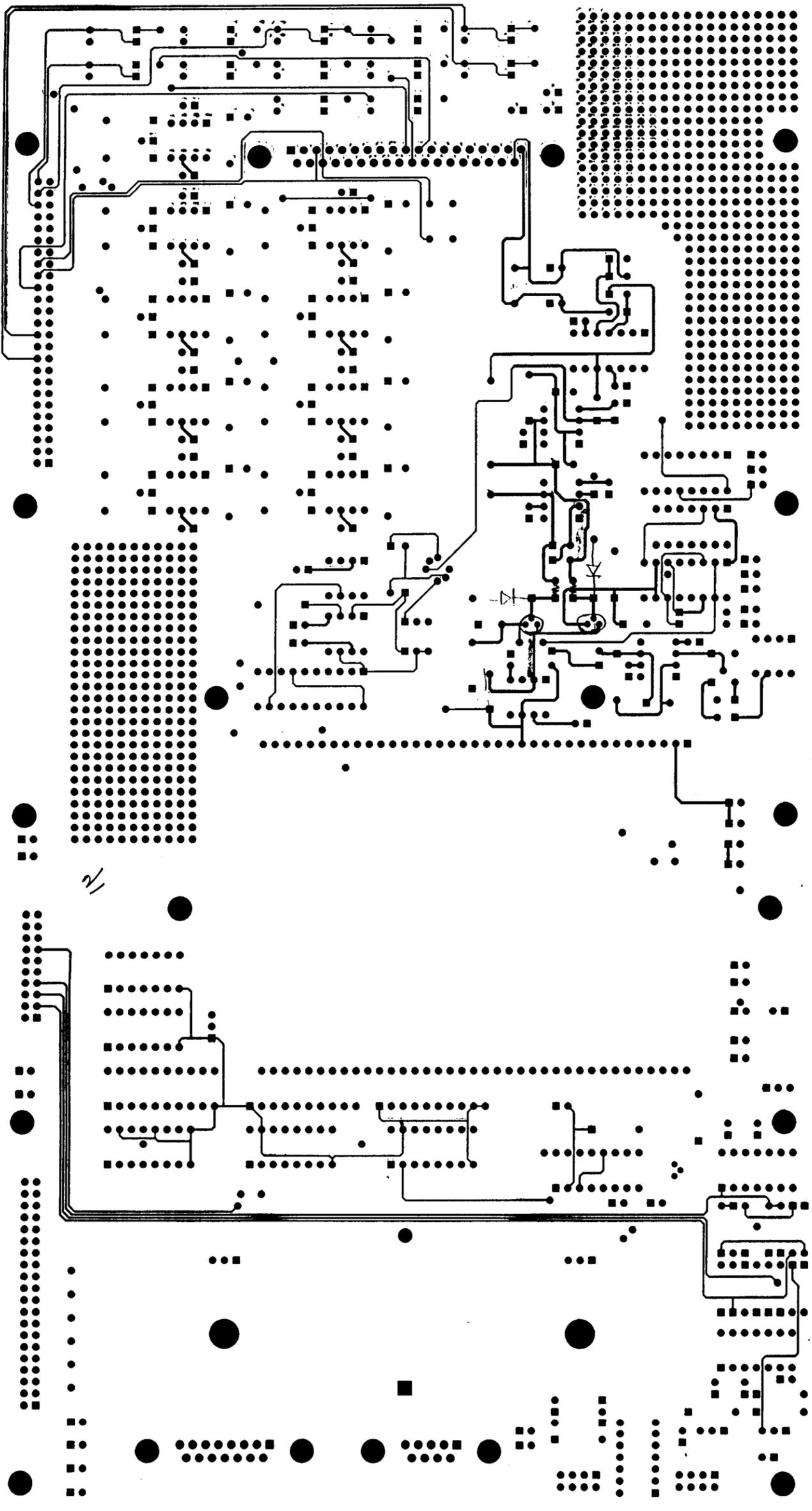
FOLDOUT FRAME

2.

49

1927

MIDLAYER 2

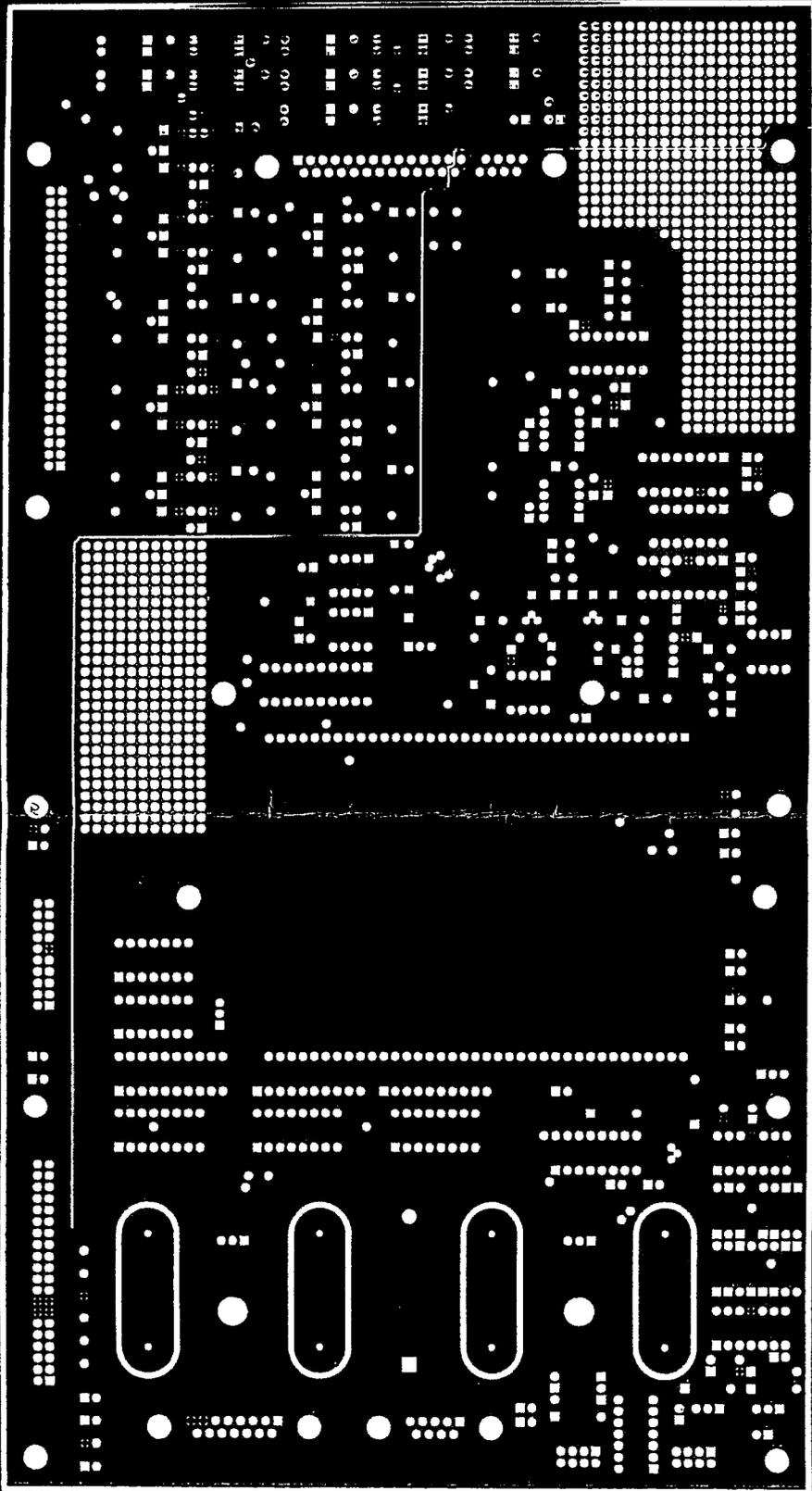


FOLDOUT FRAME

FOLDOUT FRAME 2.

50

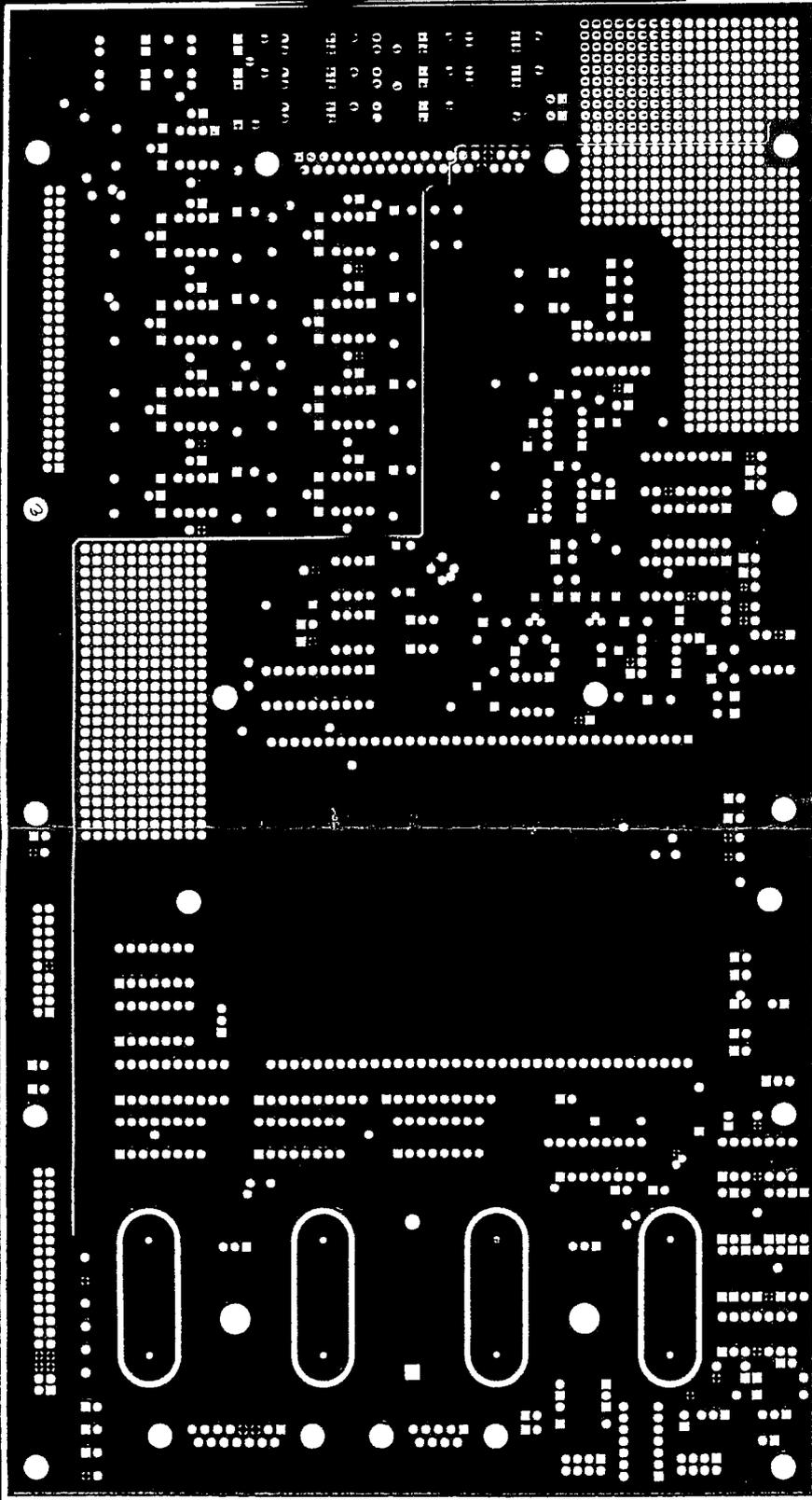
15 POWER PLANE



FOLDOUT FRAME

ADPT FRAM

+15 POWER PLANE



ADJUST FRAME

ADJUST FRAME

4.4.6 Analog Procesor Card Logic Equations

DEVICE AnalogTiming (PAL16R6)

" The PAL program is for SBIR analog card "

PIN

```
  clkIn =      2 (input combinatorial)

  ctc =      5 (input combinatorial)
  en =      6 (input combinatorial) " start transmitting "
  lp =      4 (input combinatorial) " last pixel actived high"
en16 =      3 (input combinatorial)
           " format selected high=16 bits low= 15 bits "
/sclk =      19 (output combinatorial active_low)
/clkout =     12 (output combinatorial active_low)
/cnt[4:0] =   17:13 (output registered active_low)
/sen =      18 (output registered active_low)
;
```

BEGIN

```
ENABLE ( sclk ) ;
ENABLE ( clkout ) ;
```

```
sclk = clkout*/sen+sen*lp;
```

```
IF ( en16 ) THEN
```

```
/sen = (cnt[4]+cnt[3]+cnt[2]+cnt[1]+cnt[0]) *
(/cnt[4]+cnt[2]+cnt[3]+cnt[1]+/cnt[0]) *
(/cnt[4]+cnt[3]+cnt[2]+/cnt[1]+cnt[0]) *
(/cnt[4]+cnt[3]+cnt[2]+cnt[1]+cnt[0]+/lp);
```

```
" Because that the /sen is registered output we need 1 cycle
earlier for the logic equation. The equation is that
IF (0) or (17) or (18) THEN stop sending . and
IF it is last pixel THEN stop on cycle earlier. "
```

ELSE

```
/sen = (cnt[4]+cnt[3]+cnt[2]+cnt[1]+cnt[0]) *
(/cnt[4]+cnt[3]+cnt[2]+cnt[1]+cnt[0]) *
(/cnt[4]+cnt[3]+cnt[2]+cnt[1]+/cnt[0]) *
(/cnt[4]+cnt[3]+cnt[2]+/cnt[1]+cnt[0]) *
(cnt[4]+/cnt[3]+/cnt[2]+/cnt[1]+/cnt[0]+/lp);
```

```
clkout = clkIn*(/cnt[4]+cnt[3]+cnt[2]+/cnt[1]+cnt[0]+/en);
```

```
IF (/en) THEN cnt[4:0] = 0;
```

ELSE BEGIN

```
  CASE (cnt[4:0]) BEGIN
    0) cnt[4:0] = 1;
    1) cnt[4:0] = 2;
    2) cnt[4:0] = 3;
    3) cnt[4:0] = 4;
    4) cnt[4:0] = 5;
    5) cnt[4:0] = 6;
    6) cnt[4:0] = 7;
    7) cnt[4:0] = 8;
    8) cnt[4:0] = 9;
    9) cnt[4:0] = 10;
```

10) cnt [4:0] = 11;
11) cnt [4:0] = 12;
12) cnt [4:0] = 13;
13) cnt [4:0] = 14;
14) cnt [4:0] = 15;
15) cnt [4:0] = 16;
16) cnt [4:0] = 17;
17) cnt [4:0] = 18;
18) cnt [4:0] = 18;
19) cnt [4:0] = 18;
20) cnt [4:0] = 18;
21) cnt [4:0] = 18;
22) cnt [4:0] = 18;
23) cnt [4:0] = 18;
24) cnt [4:0] = 18;
25) cnt [4:0] = 18;
26) cnt [4:0] = 18;
27) cnt [4:0] = 18;
28) cnt [4:0] = 18;
29) cnt [4:0] = 18;
30) cnt [4:0] = 18;
31) cnt [4:0] = 18;

END;

END;

END .

5. Camera Head

The AIS1 camera head contains the CCD imager itself as well as the circuitry necessary to support two channels of video pre-amplification. The camera head is an evacuated chamber to allow the CCD to be more effectively cooled by the liquid nitrogen dewar to which it is connected. The CCD imager easily reaches temperatures below -80°C . The controller module and clock/analog modules are attached to the sides of the camera head. In the case of the clock/analog modules this is to assure that the CCD video signal will not have to travel far and therefore will not be corrupted by noise pickup. In the case of the controller module, it is more a matter of convenience.

5.1 Overview

There are two circuit cards located inside the camera head. One of these, the 'filter card', is used to condition the DC voltages from the clock cards before applying them to the CCD. In most cases this is a simple RC filter network, in some it is an emitter follower type of transistor circuit. The other circuit card is the 'socket card', which holds the CCD itself as well as the video preamplifiers, one for each CCD port read. Additionally, there are a number of jumpers on the socket card which allow the user to select between the amplifiers located at either end of the CCD serial registers. The two cards are connected together by a number of interconnects. The two circuit cards will be described in more detail later in this section.

5.2 Camera Head Connector Pinouts

There are three connectors on the outside of the camera head. Two of these are DB37 style and one is a four pin style. Each of the two 37 pin connectors are used to bring the CCD clock signals and DC power into the camera head for one CCD port. These also bring the CCD video signal out of the chamber and to the analog card. P5 on the analog card is a vertical PC mount connector allowing the analog card inside the clock/analog module to be connected directly to the camera head. If necessary, this connection could be broken and a short cable put in its place. The DB37 connectors naturally have the same pinout as P5 on the analog card. The pinout is shown below.

Pinout of DB37 Connectors on AIS1 Camera Head

1	serial clock 1	20	serial clock 2
2	serial clock 3	21	serial clock 4
3	TTL reset	22	TTL summing well
4	extra clock 1	23	extra clock 2
5	extra clock 3	24	extra clock 4
6	parallel clock 1	25	parallel clock 2
7	parallel clock 3	26	parallel clock 4
8	transfer gate	27	
9		28	substrate (unused)
10	last gate	29	output drain
11	reset drain	30	reset gate high
12	reset gate low	31	summing well high
13	sum well low	32	
14		33	-15v
15	-15v	34	+15v
16	+15v	35	+28v
17	+28v	36	ground
18	ground	37	CCD video low
19	CCD video high		

The third connector on the camera head is used for the CCD temperature control circuitry. One conductor is used to sense the CCD temperature, and another is the ground reference for this signal. The other two are connected to a resistor mounted in the cold block under the CCD to provide heat with which to regulate the CCD temperature. The temperature regulation circuitry will be discussed in greater detail below. The pinout of this connector is shown below.

Temperature Sense Connector

- 1
- 2
- 3
- 4

5.3 Filter Card

All the signals that enter the camera head from the clock/analog modules are connected to the filter card. The filter card supports two clock/analog modules. The circuitry for the two is identical. The signals from the DB37 connectors on the exterior of the

camera head are wired to connectors P6 and P7 on the filter card. The pinout of these connectors is shown below.

Filter Card Signal Input Connectors : P6 and P7

1	n.c.	15	+15V	29	serial 2
2	n.c.	16	+15V	30	parallel 2
3	n.c.	17	+28V	31	serial 3
4	n.c.	18	+28V	32	parallel 1
5	n.c.	19	substrate	33	transfer gate
6	n.c.	20	n.c.	34	parallel 3
7	n.c.	21	clamp	35	sum well high
8	n.c.	22	cur. src.	36	sum well low
9	n.c.	23	VOD	37	reset gate high
10	n.c.	24	VRD	38	reset gate low
11	-15V	25	VLG	39	reset TTL
12	-15V	26	n.c.	40	sum well TTL
13	ground	27	n.c.		
14	ground	28	serial 1		

The serial and parallel clocks, as well as the transfer gate and clamp signals pass through the filter card without encountering any circuitry. They are connected directly to the CCD socket card.

The DC power supply voltages that come into the camera head are filtered on the filter card. Large hermetically sealed capacitors are used to bypass the supplies on each side to their respective grounds.

The CCD reset gate and summing well switches are located on the filter card. The DC rails for these signals are filtered and then applied to the poles of analog switches. The switches for both the reset gate and the summing well on each side are located in a single integrated circuit, U1 or U4, HI-201HS type switches. The lower rail of the reset gate is in fact not tied to the voltage supplied by the clock card. It was found experimentally that the noise figures for the camera could be improved by tying the lower rail of the reset gate switch to the CCD substrate near the CCD output. A wire has been added which accomplishes this.

The substrate signal supplied by the clock card is not used. It was found that it was better to tie the CCD substrate to ground directly.

The CCD output drain voltages are filtered by an RC network and then buffered by an emitter follower. There is additional filtering on the CCD socket card.

The CCD reset drain voltages from the clock card are handled differently. The system was originally designed to provide separate reset drain voltages to the two halves of the CCD. Each of these was to have seen an emitter follower on the filter card. It was found that is best to tie these two points to the same potential. To that end, one of the emitter follower circuits, that built around Q11, has been removed and its output tied instead to the output of the circuit built around Q4.

The unused connections on the CCD, those associated with amplifiers that are not being used, are tied to a 'holding' voltage. This voltage is produced by the voltage divider and emitter follower circuit constructed out of R7, R8, and Q2. Capacitors C16, C17, and C18 are used to stabilize this circuit. The system originally was to have another of these circuits built around Q9, but it was determined that it would be better to tie all the unused CCD connections to the voltage produced by Q2.

The filter card also holds the circuitry used to sense the CCD temperature. A current source draws 1 mA of current through an RTD epoxied to the CCD package. The voltage across the terminals of this RTD is sensed and amplified by U7, an INA101 type instrumentation amplifier from Burr-Brown. The resulting voltage is fed through the camera head wall on the four pin connector that feeds into the controller module and to the temperature/shutter control card.

The signals from the filter card are passed to the CCD socket card via interconnects between the two cards. There are five such interconnects. Two identical pairs include all the signals associated with the CCD itself, and the fifth includes those signals associated with the temperature measurement circuitry. The pinouts of these connectors are shown below.

Filter Card Connectors J1 and J4

1	serial 1
2	parallel 2
3	serial 2
4	parallel 1
5	serial 3
6	parallel 3
7	transfer gate
8	CCD reset
9	unused
10	summing well

Filter Card Connectors J2 and J5

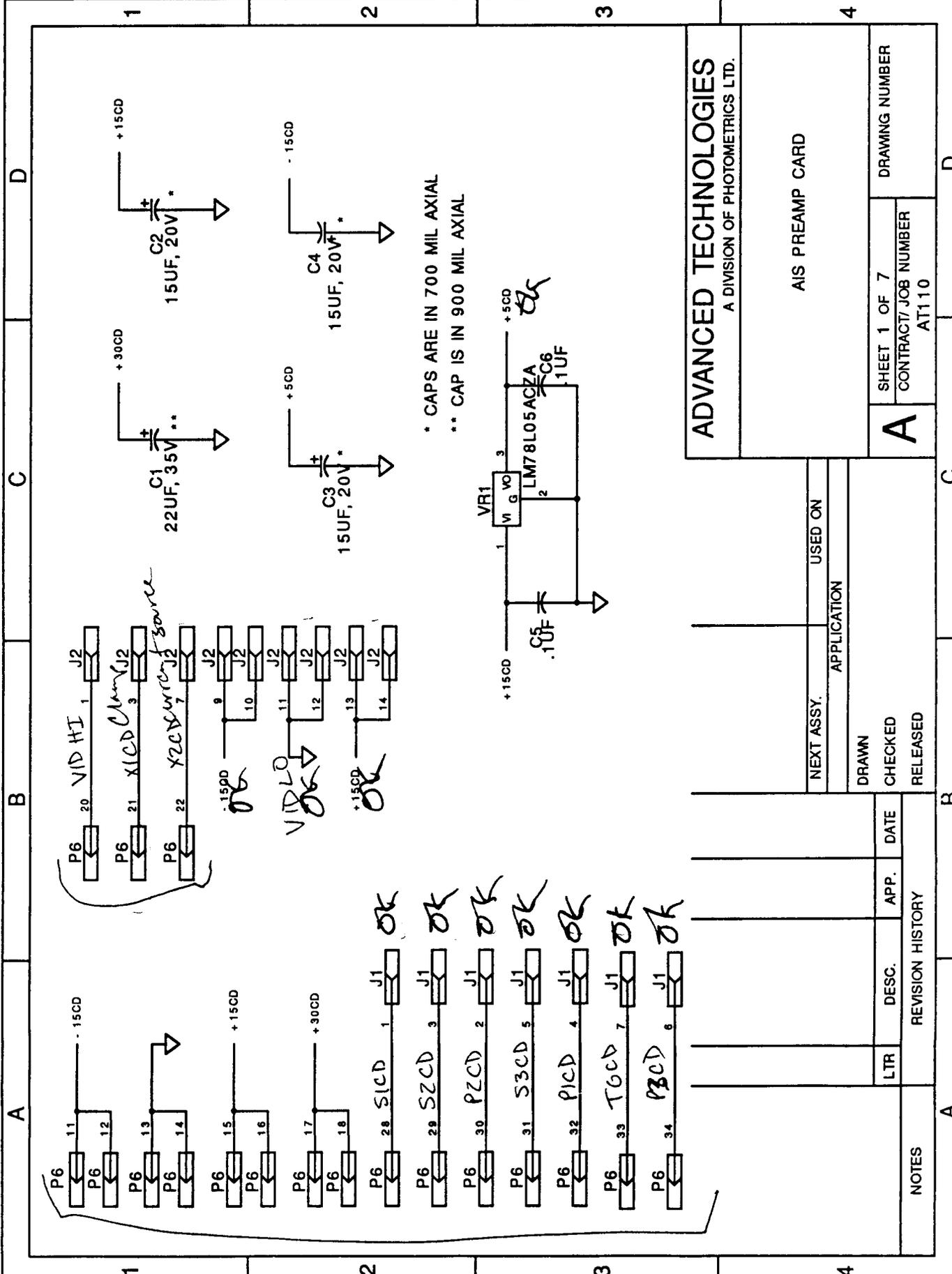
1	unused (formerly CCD video)
2	VOD
3	clamp
4	VRD
5	'holding' voltage
6	VLG
7	current source
8	unused
9	-15V
10	-15V
11	ground
12	ground
13	+15V
14	+15V

The temperature measurement circuitry may be powered by the +15V and -15V supplies from either analog card. Jumpers on the CCD socket card select which will be used. It was found that it made no difference which side was used (which is what you'd hope). The selected power supplies are brought to the filter card via J3. The pinout of this connector is shown below.

Filter Card Connector J3

1	+15V
2	+15V
3	GND
4	GND
5	-15V
6	-15V
7	unused
8	unused
9	RTD+
10	RTD-
11	Heater+
12	Heater-

5.3.1 Filter Card Schematic Diagram



* CAPS ARE IN 700 MIL AXIAL
 ** CAP IS IN 900 MIL AXIAL

ADVANCED TECHNOLOGIES
 A DIVISION OF PHOTOMETRICS LTD.

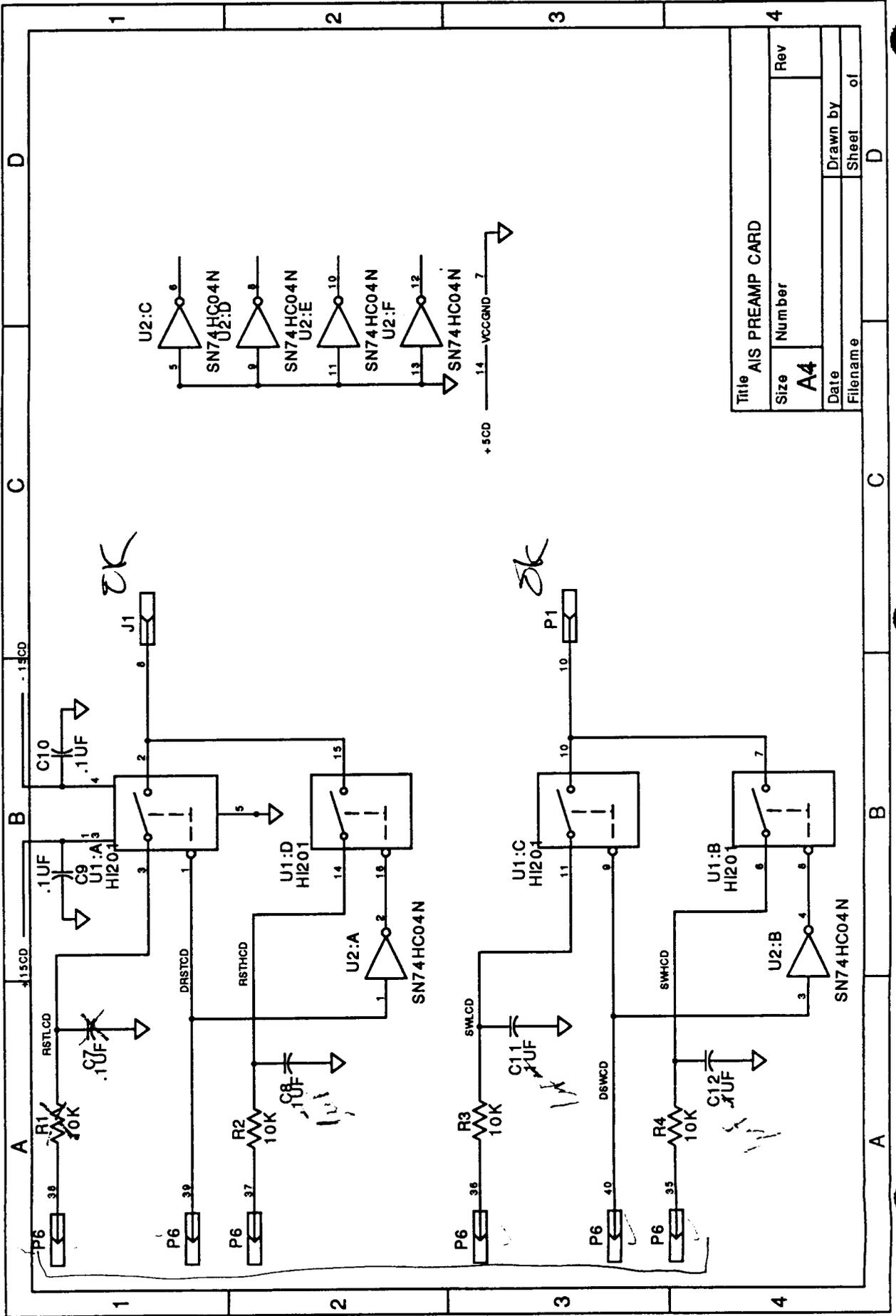
AIS PREAMP CARD

A	SHEET 1 OF 7	DRAWING NUMBER
	CONTRACT/JOB NUMBER AT110	

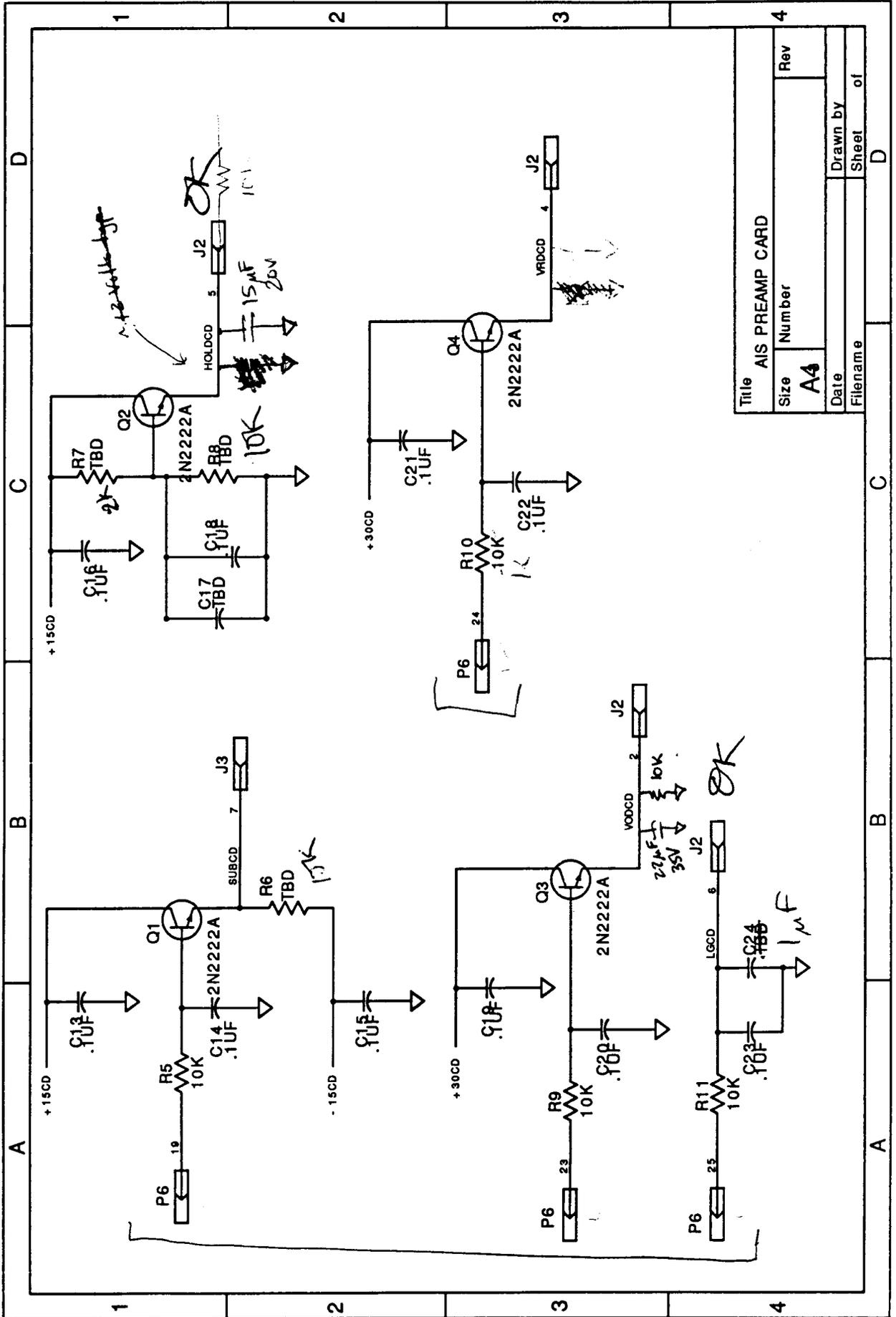
NEXT ASSY.	USED ON
APPLICATION	
DRAWN	CHECKED
RELEASED	RELEASED

REVISION HISTORY		
LTR	DESC.	DATE

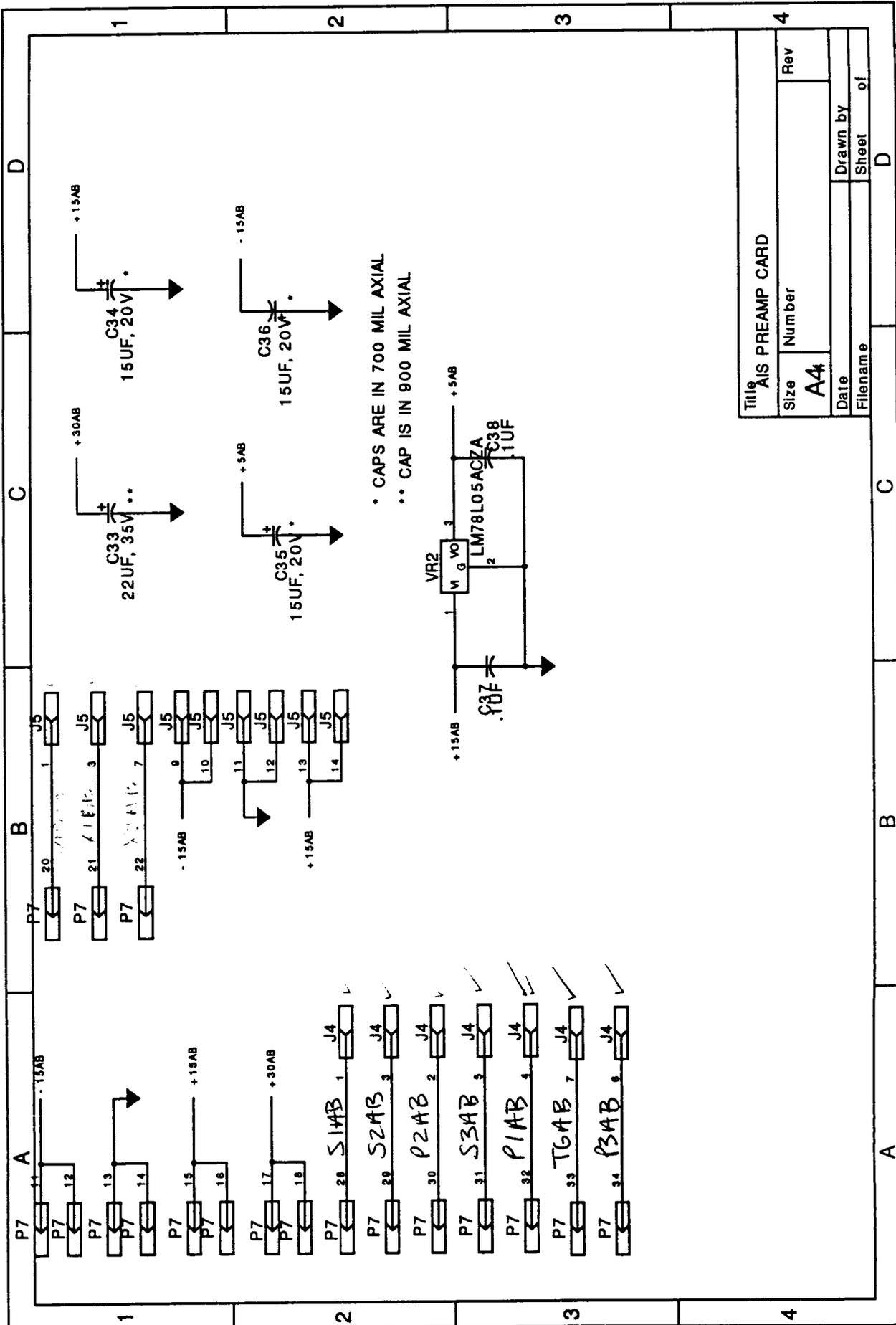
NOTES



Title AIS PREAMP CARD	
Size A4	Number
Date	Rev
Filename	Drawn by
Sheet	of



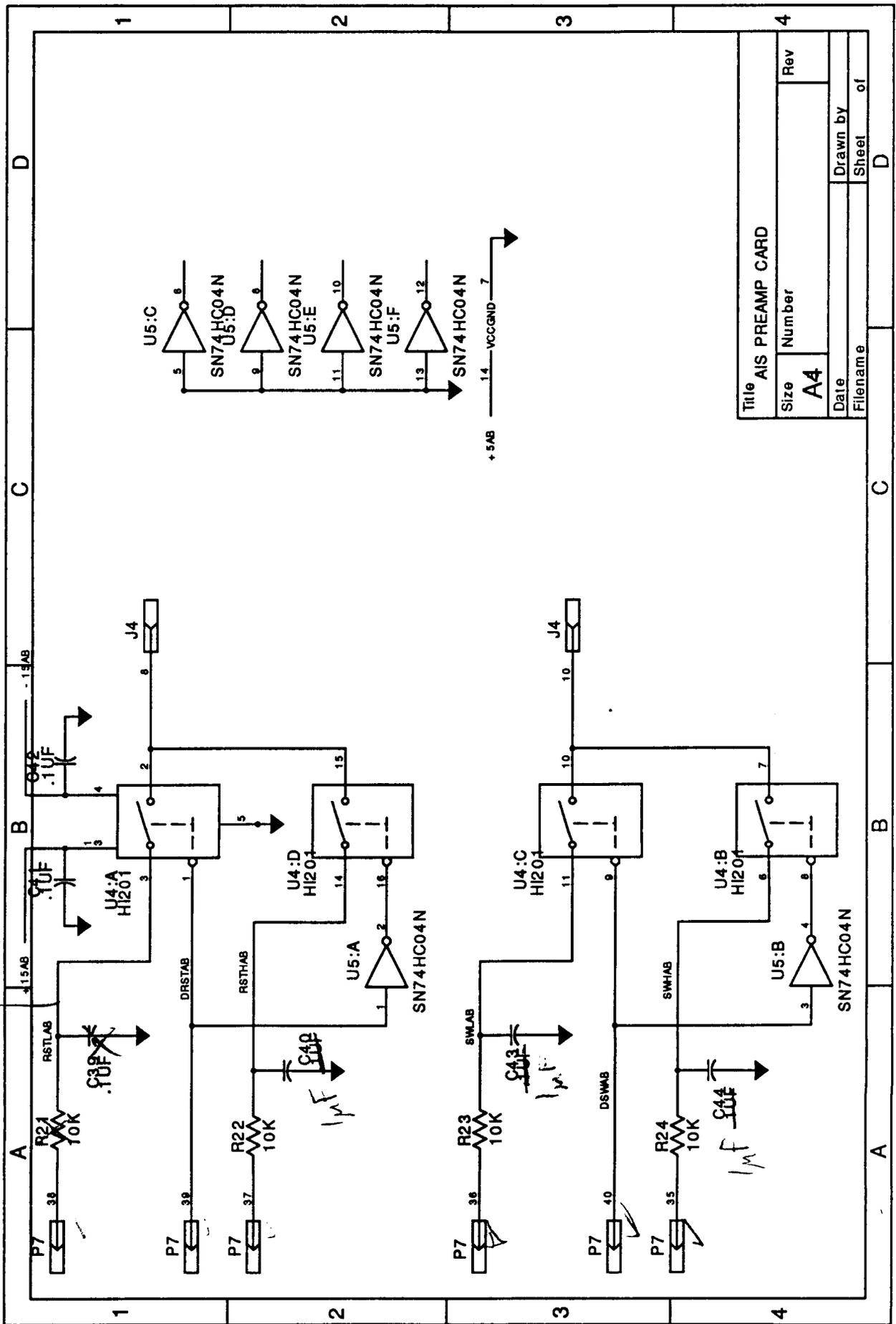
Title		AIS PREAMP CARD	
Size	Number	Rev	
A4			
Date	Drawn by		of
Filename	Sheet		D



* CAPS ARE IN 700 MIL AXIAL
 ** CAP IS IN 900 MIL AXIAL

Title		AIS PREAMP CARD	
Size	Number	Rev	
A4			
Date	Drawn by		Sheet
Filename			of
			D

Handwritten note: Input to J4



Title AIS PREAMP CARD	
Size A4	Number
Date	Rev
Filename	Drawn by
	Sheet of

A

B

C

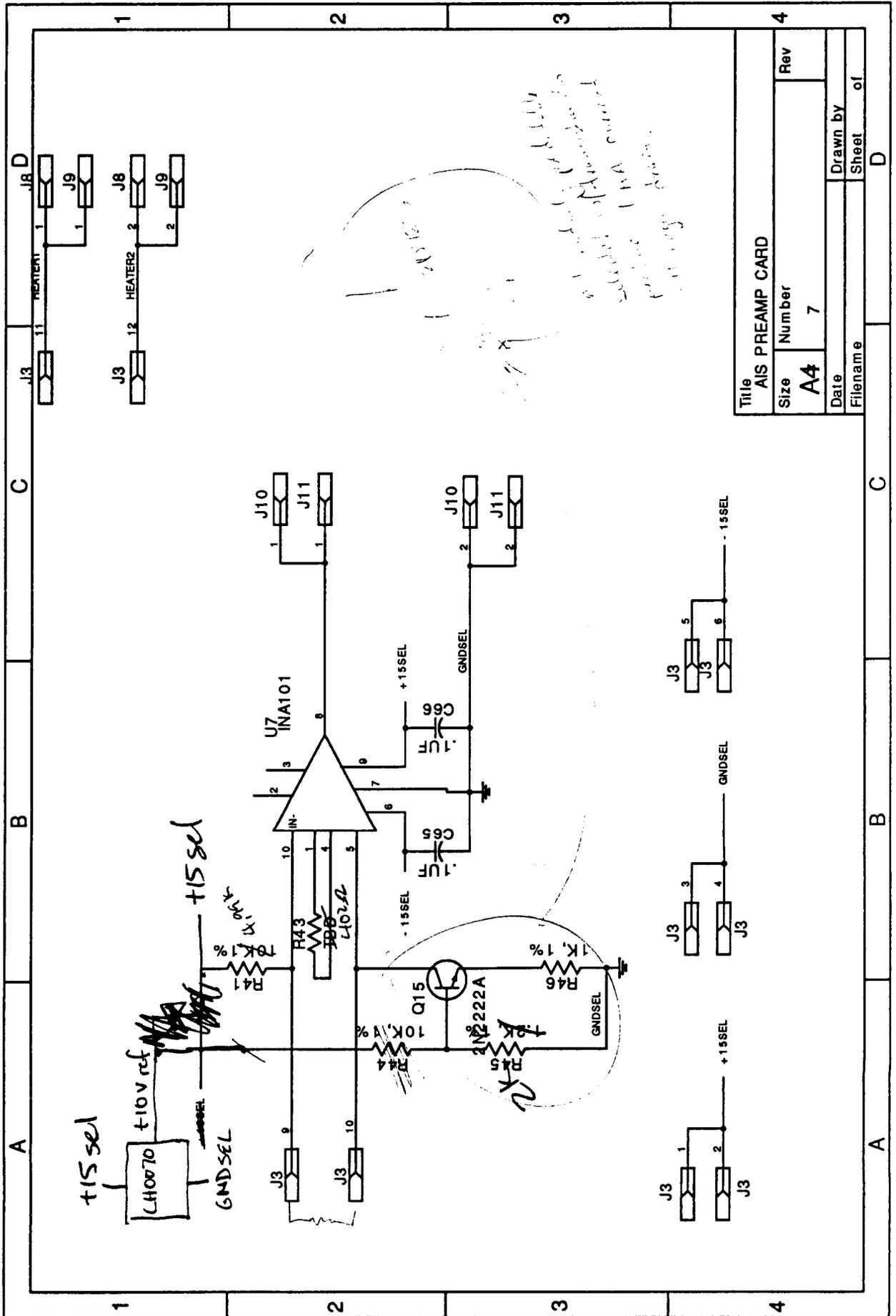
D

1

2

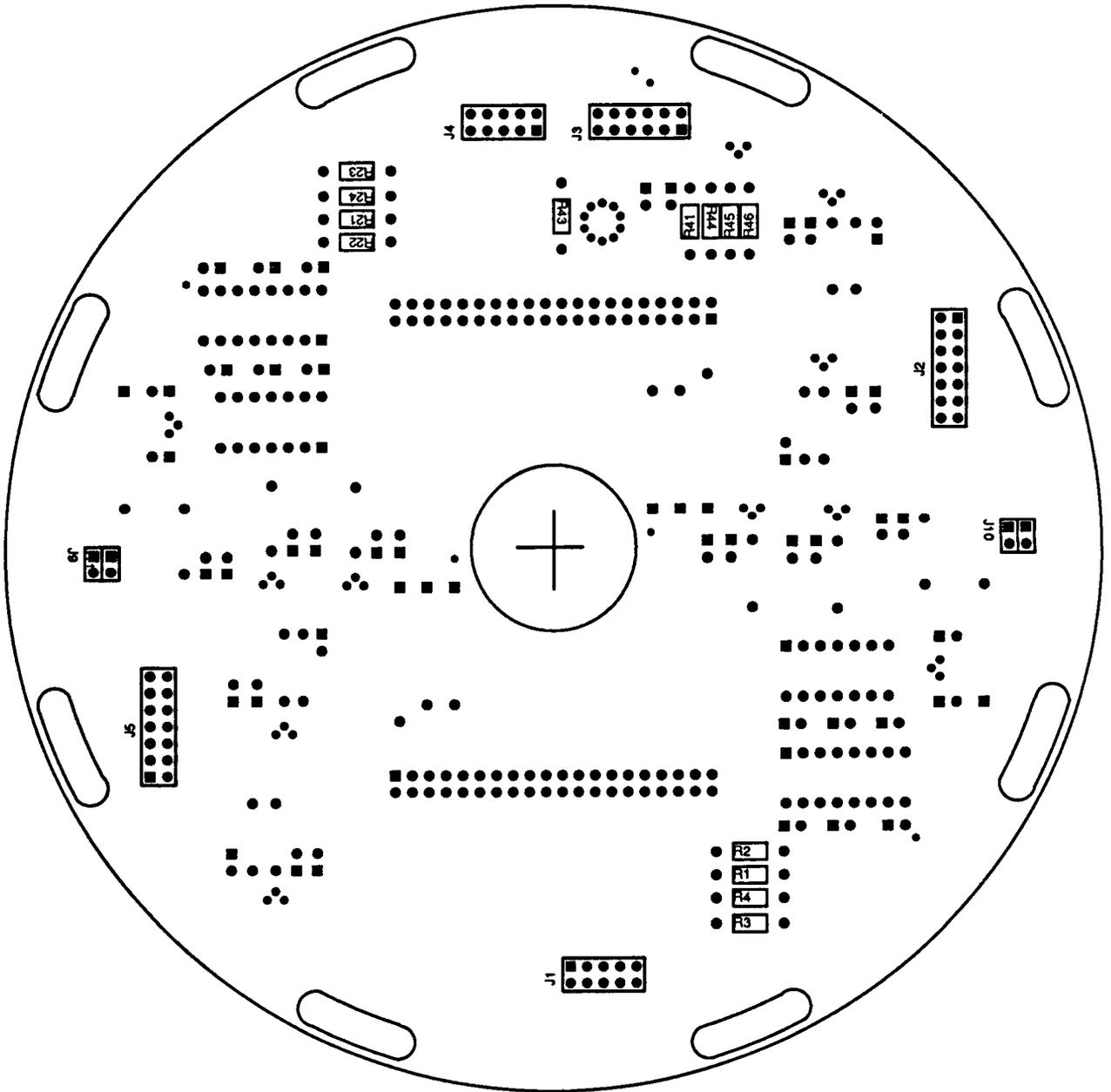
3

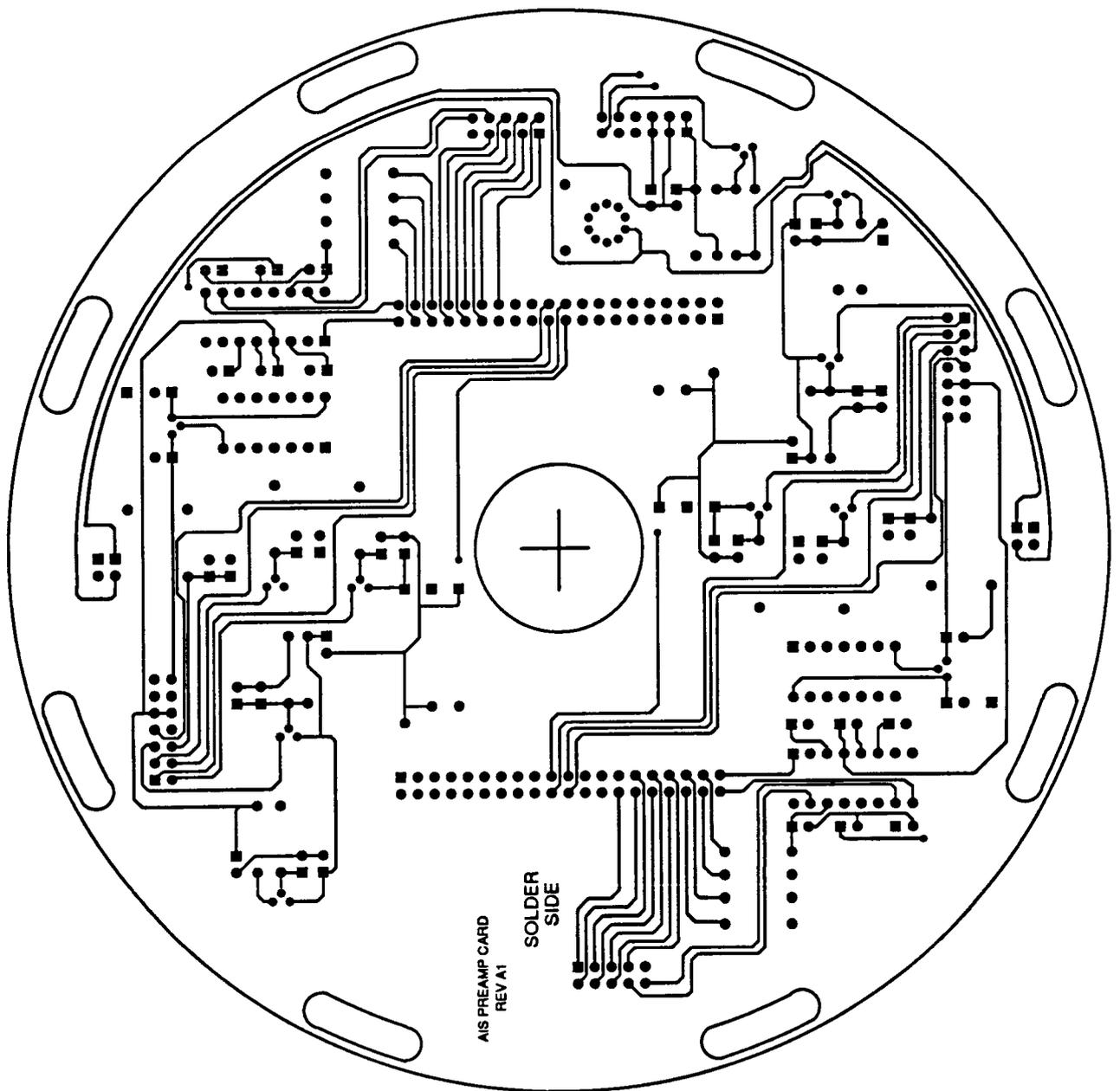
4

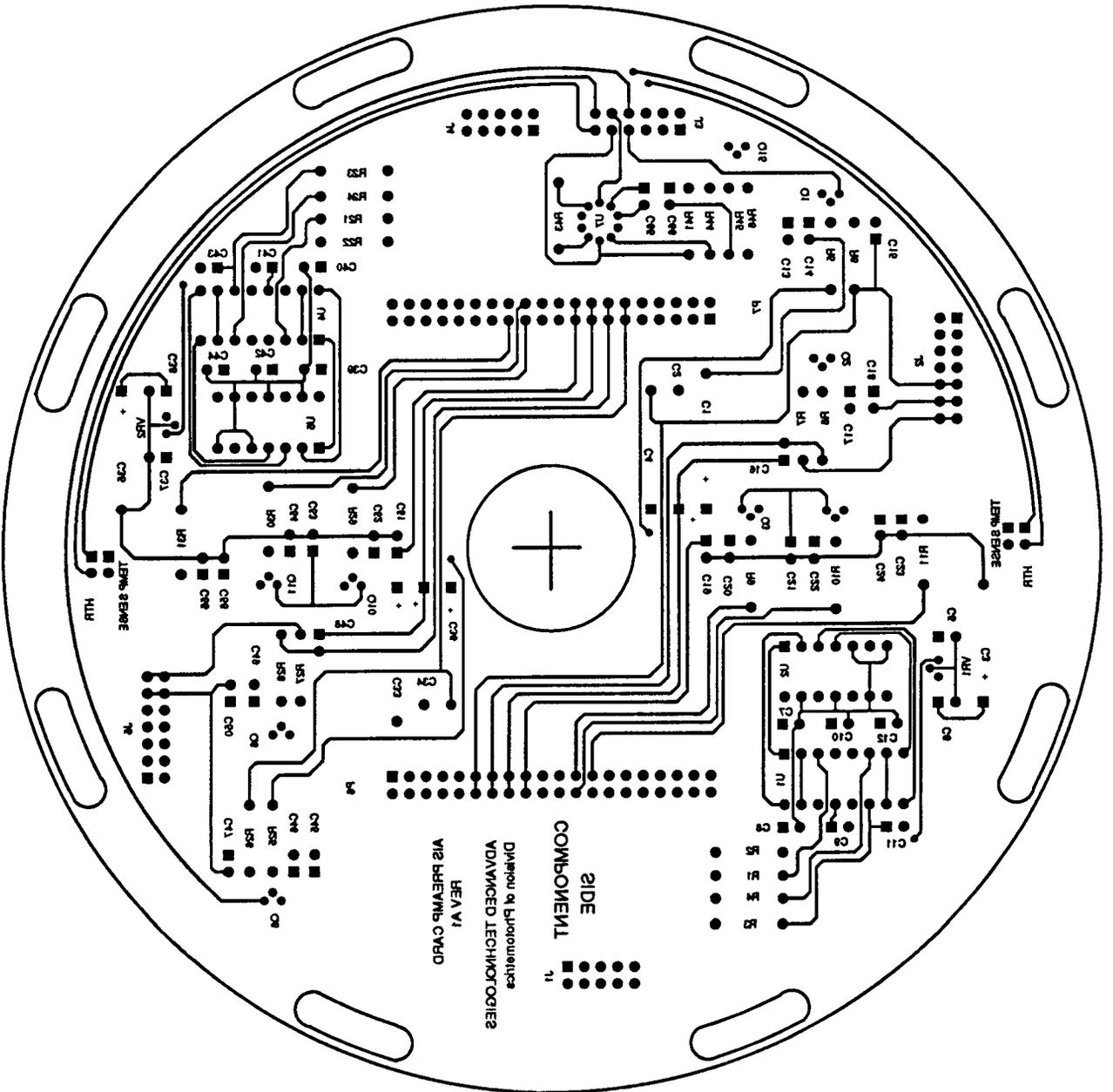


Title		AIS PREAMP CARD	
Size	Number	Date	Rev
A4	7		
Date		Drawn by	
Filename		Sheet	
		of	
		D	

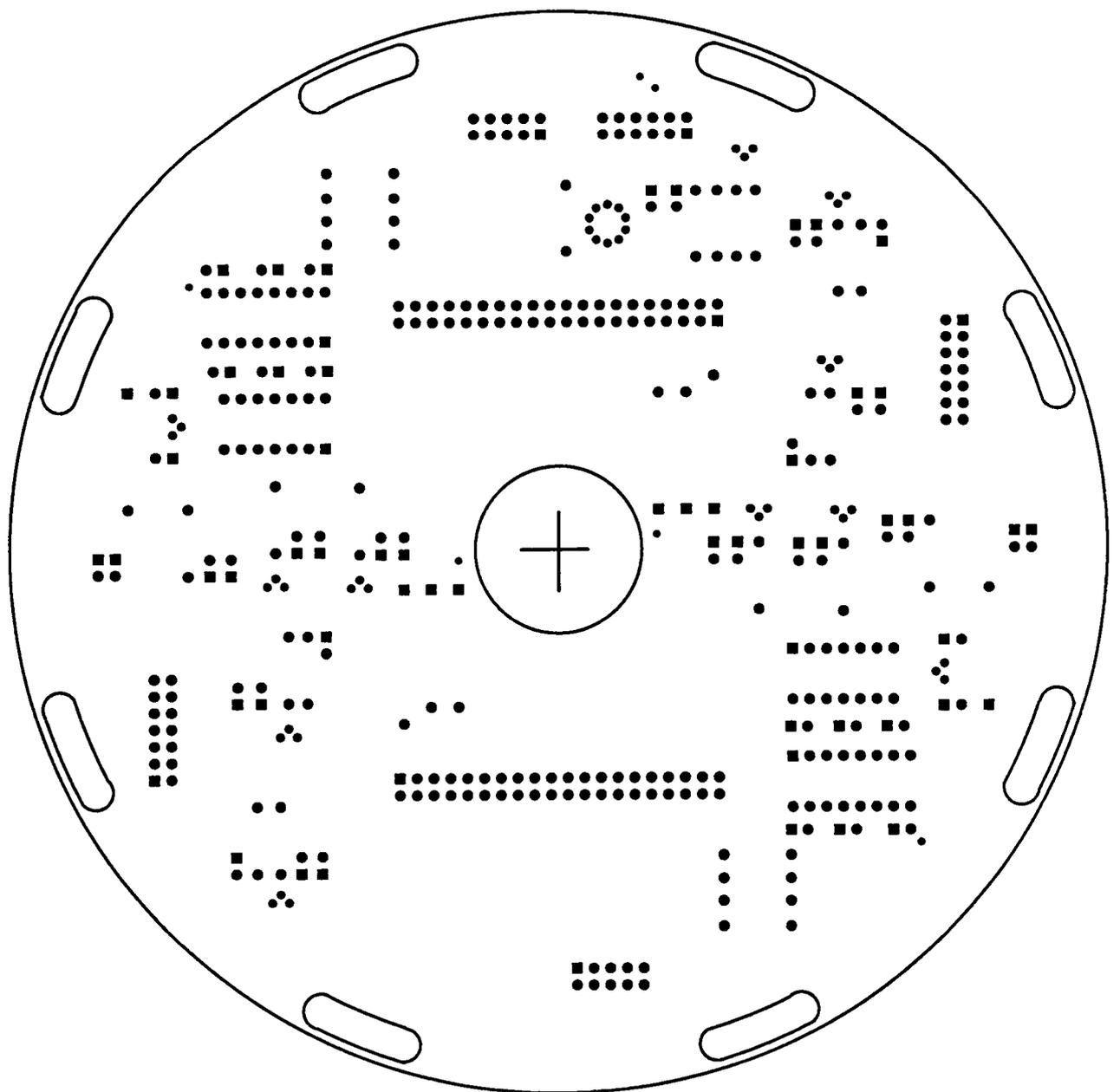
5.3.2 Filter Card PCB Artwork







SIDE
 COMPONENT
 11
 Division of Photometrics
 ADVANCED TECHNOLOGIES
 REV V1
 VIS PREAMP CARD



5.4 CCD Socket Card

The CCD socket card holds the CCD and the video preamplifiers. In addition, it also holds conditioning resistors for the CCD serial and parallel clocks and various bypass components for the CCD reference voltages.

The CCD socket card receives the clock and reference signals from the filter card through connectors P1, P2, P3, P4, and P5. These connectors plug into the connectors J1 through J5 on the filter card and therefore have the same pinout as described for them above.

There are many options available for the configuration of the CCD socket card and these are selected using jumpers. The various options will be discussed in the following sections as appropriate.

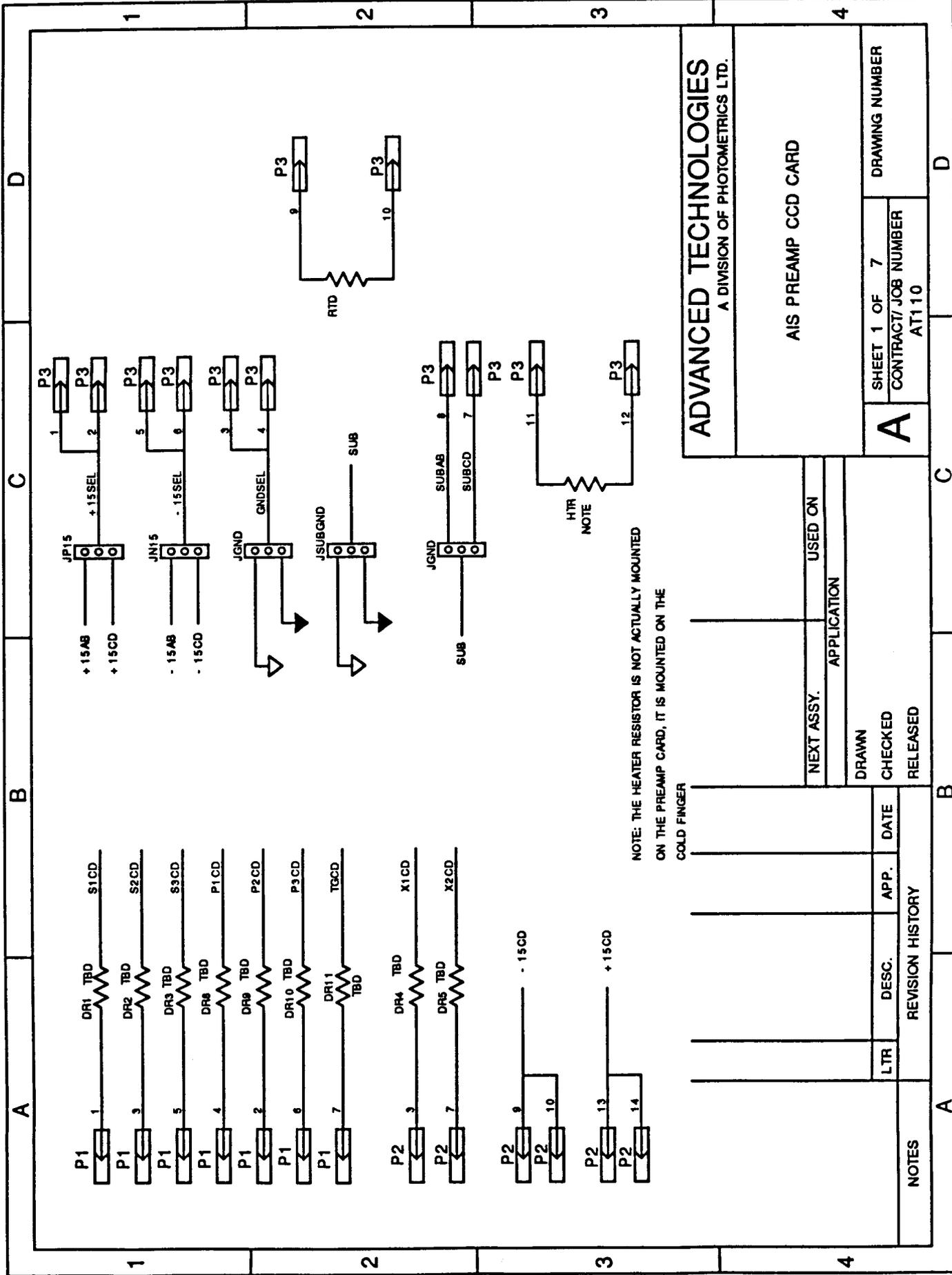
The CCD socket card is designed to hold the Tektronix M745A CCD designed specifically for the STIS project. The CCD is a four port device and the camera is designed to support two of those ports simultaneously. Because the quality of a particular CCD output port cannot be predicted, it was decided to clock the CCD in a 'split parallel' configuration and use one output port on each serial register. To select which end of a serial register is to be used requires the positioning of several jumpers. Jumpers are provided for selecting which VSS signal is attached to the preamplifier, and jumpers are provided for selecting how the various reference voltages, VOD, VRD, and VLG are connected as well as for the reset gate and summing well signals. These signals, at each of the four ports, may be either connected to the active signal associated with that point, or tied to the 'holding' voltage by the placement of the jumpers. To allow a particular unused signal to 'float' simply remove the jumper altogether.

The CCD signal may be processed in a number of ways before reaching the preamplifier. Jumpers J3 and J6 select whether the CCD outputs will see a current source or a simple load resistor, and jumpers J2 and J5 determine whether the current source will be set by a voltage divider located on the CCD socket card or by the programmable voltage provided from the clock card. The simple load resistor was found to provide results equal or better to the other options at this point.

The video signal then passes through C26 or C60, to provide AC coupling of the signal. Transistors Q6 and Q13 are used to clamp the signal to ground during each row shift period. Transistors Q7 and Q14 are used to buffer the signal at this point, and operational amplifiers U3 and U6 serve as the preamplifiers. The gain of these preamplifiers is set to 11.

The video signals were then to pass through connectors P2 and P5, but it was found that they picked up excessive noise through this path, and individual wires were connected from the CCD socket card to the DB37 connectors on the side of the camera head.

5.4.1 CCD Socket Card Schematic Diagram



NOTE: THE HEATER RESISTOR IS NOT ACTUALLY MOUNTED ON THE PREAMP CARD, IT IS MOUNTED ON THE GOLD FINGER

ADVANCED TECHNOLOGIES
A DIVISION OF PHOTOMETRICS LTD.

AIS PREAMP CCD CARD

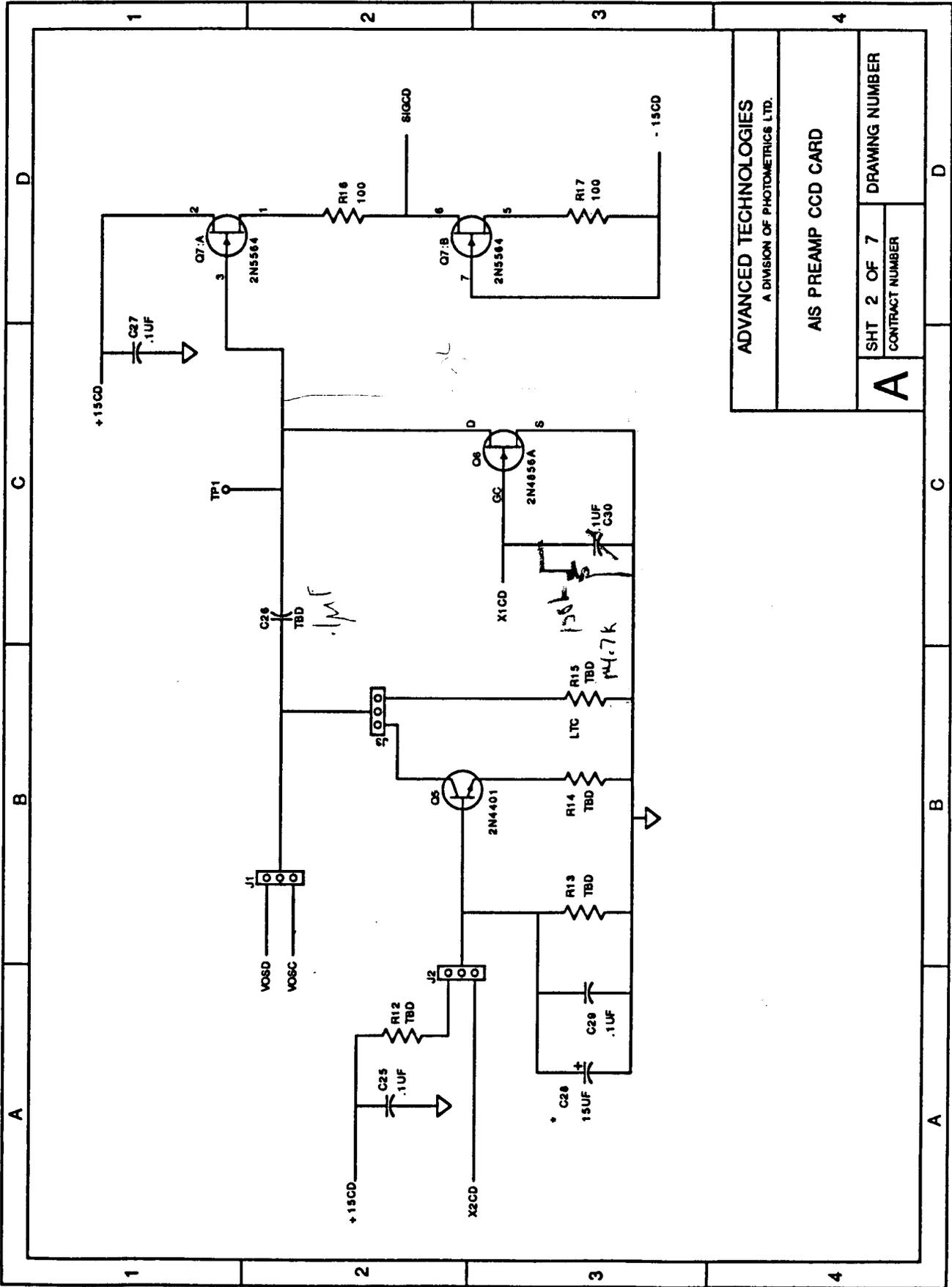
SHEET 1 OF 7
CONTRACT/JOB NUMBER
AT110

USED ON	APPLICATION
DRAWN	CHECKED
RELEASED	

LTR	DESC.	APP.	DATE

REVISION HISTORY

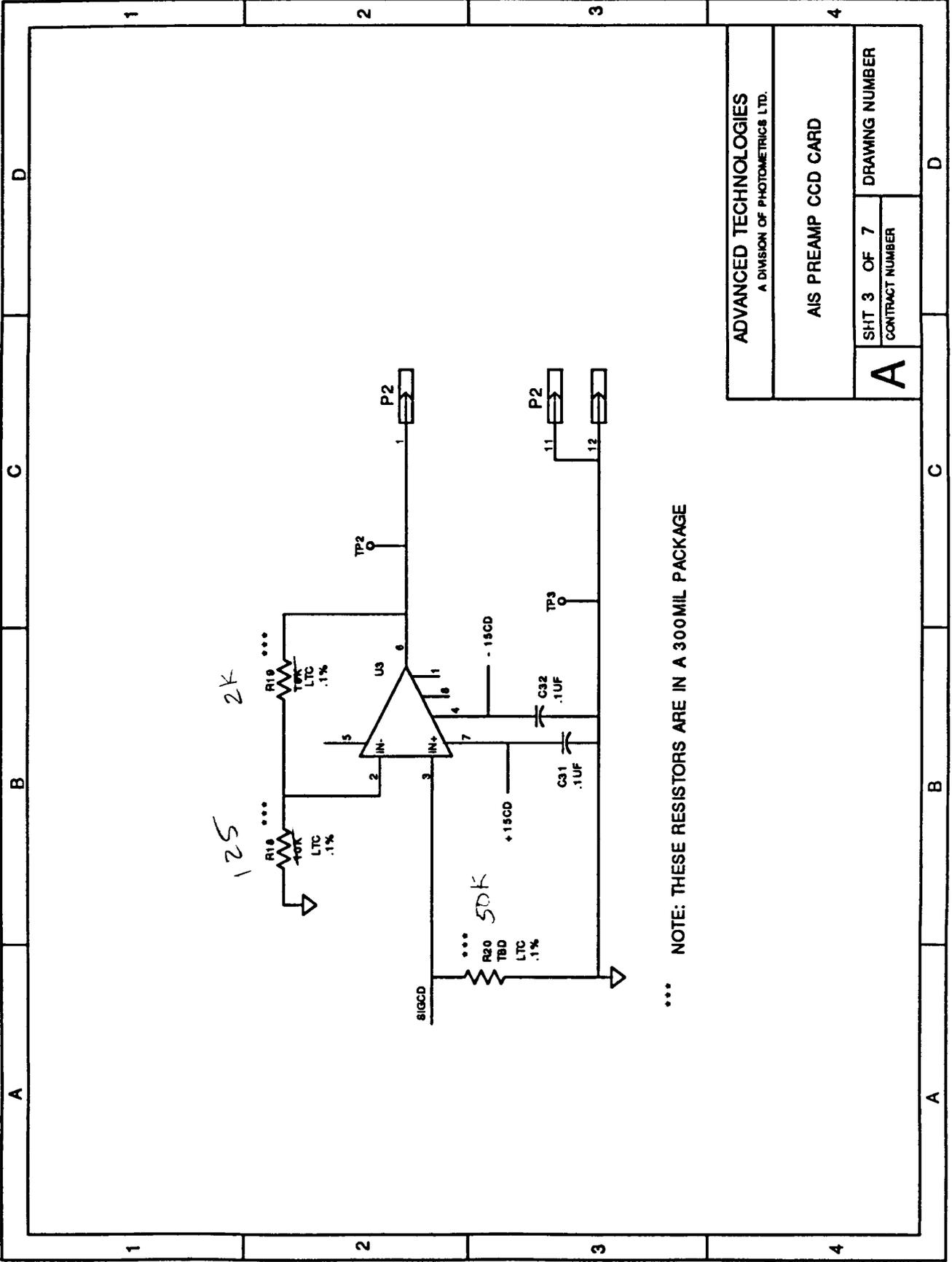
NOTES



ADVANCED TECHNOLOGIES
A DIVISION OF PHOTOMETRICS LTD.

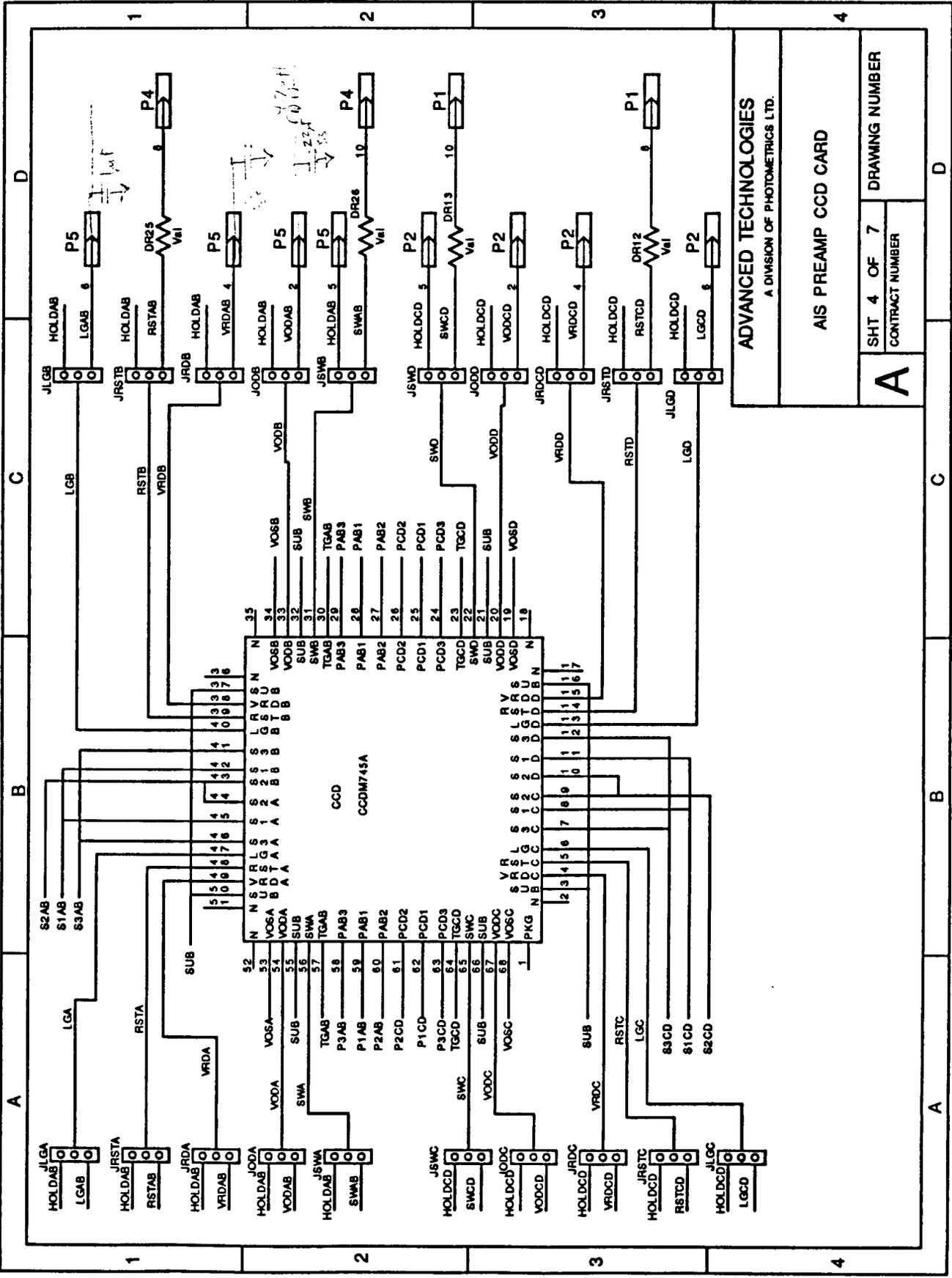
AIS PREAMP CCD CARD

A	SHT 2 OF 7	DRAWING NUMBER
	CONTRACT NUMBER	



*** NOTE: THESE RESISTORS ARE IN A 300MIL PACKAGE

ADVANCED TECHNOLOGIES A DIVISION OF PHOTOMETRICS LTD.	
AIS PREAMP CCD CARD	
A	SHT 3 OF 7
	DRAWING NUMBER CONTRACT NUMBER



ADVANCED TECHNOLOGIES
A DIVISION OF PHOTOMETRICS LTD.

AIS PREAMP CCD CARD

SHT 4 OF 7
DRAWING NUMBER
A
CONTRACT NUMBER

A

B

C

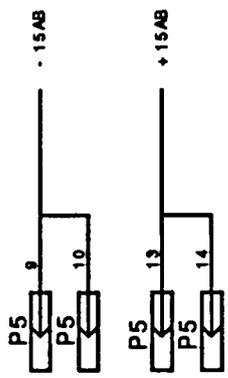
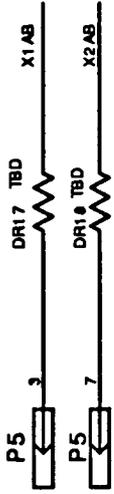
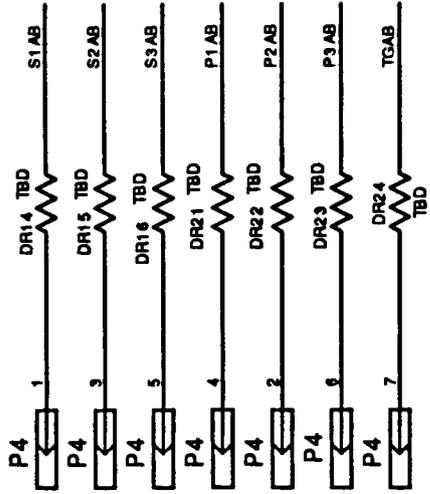
D

1

2

3

4



ADVANCED TECHNOLOGIES
A DIVISION OF PHOTOMETRICS LTD.

AIS PREAMP CCD CARD

SHEET 5 OF 7
CONTRACT/ JOB NUMBER

A

DRAWING NUMBER

NEXT ASSY. USED ON

APPLICATION

DRAWN

CHECKED

RELEASED

LTR	DESC.	APP.	DATE

REVISION HISTORY

NOTES

D

C

B

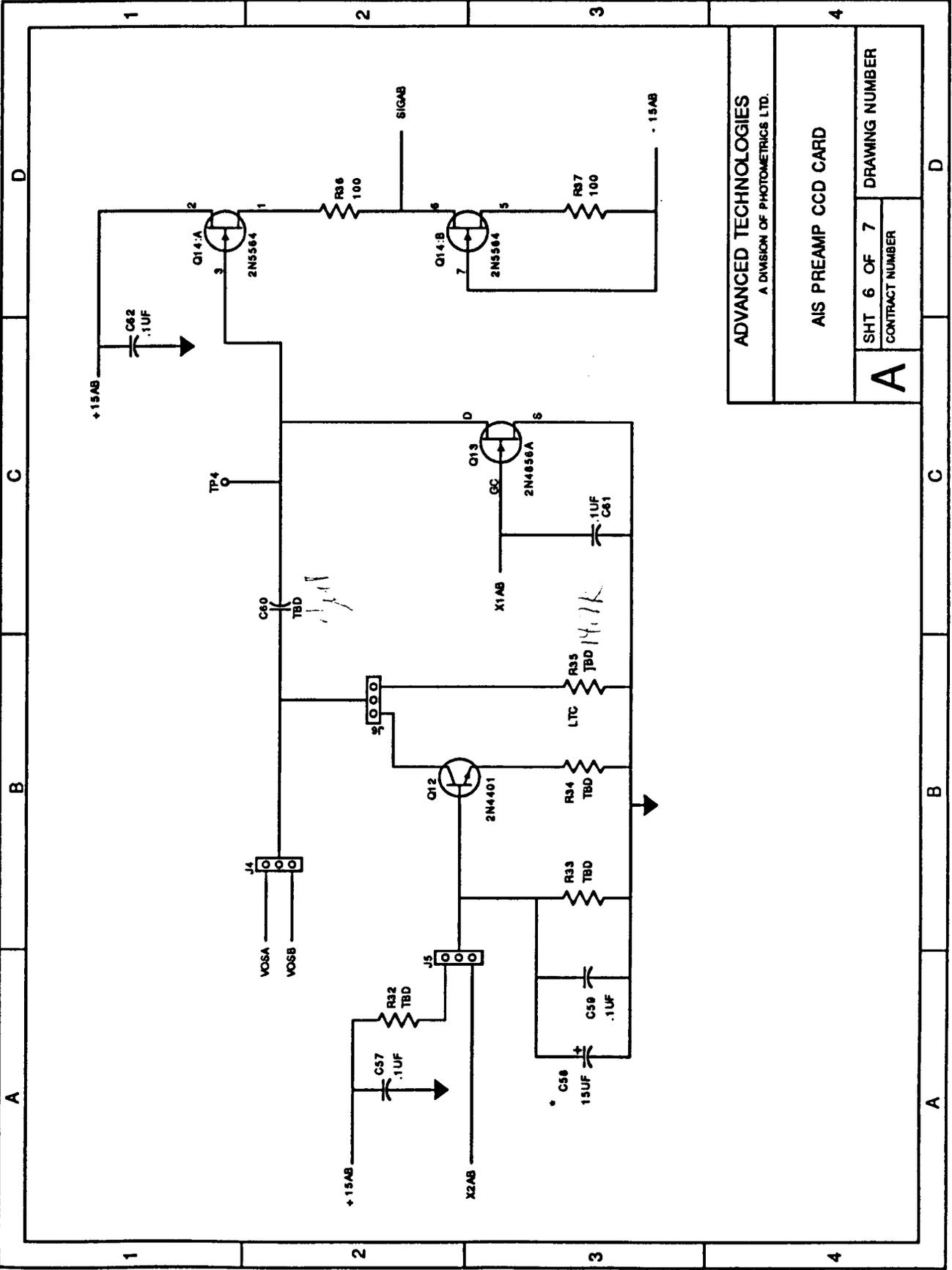
A

1

2

3

4

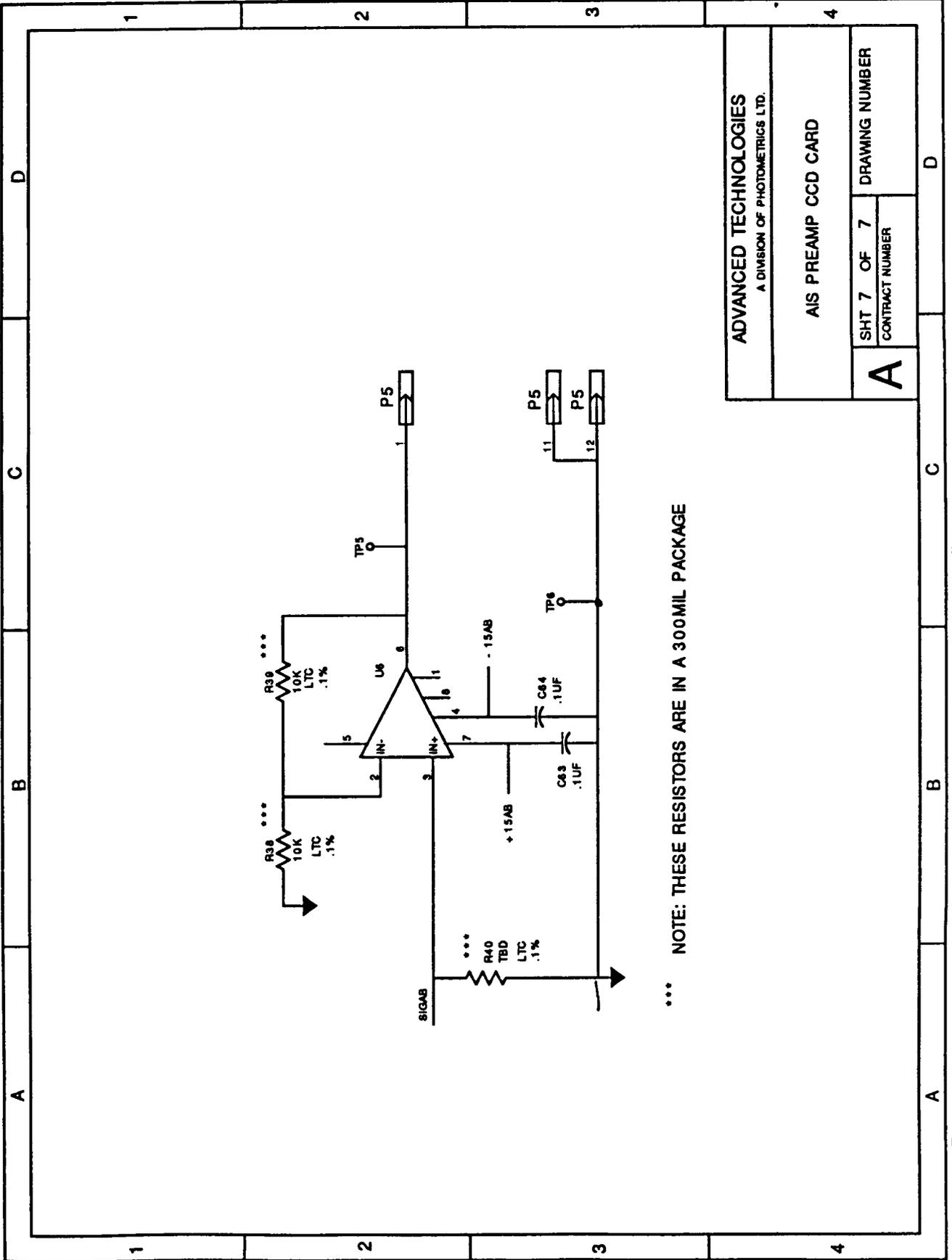


ADVANCED TECHNOLOGIES
 A DIVISION OF PHOTOMETRICS LTD.

AIS PREAMP CCD CARD

SHT 6 OF 7
 DRAWING NUMBER
 CONTRACT NUMBER

A



... NOTE: THESE RESISTORS ARE IN A 300MIL PACKAGE

ADVANCED TECHNOLOGIES
A DIVISION OF PHOTOMETRICS LTD.

AIS PREAMP CCD CARD

A	SHT 7 OF 7	DRAWING NUMBER
	CONTRACT NUMBER	

D

C

B

A

1

2

3

4

1

2

3

4

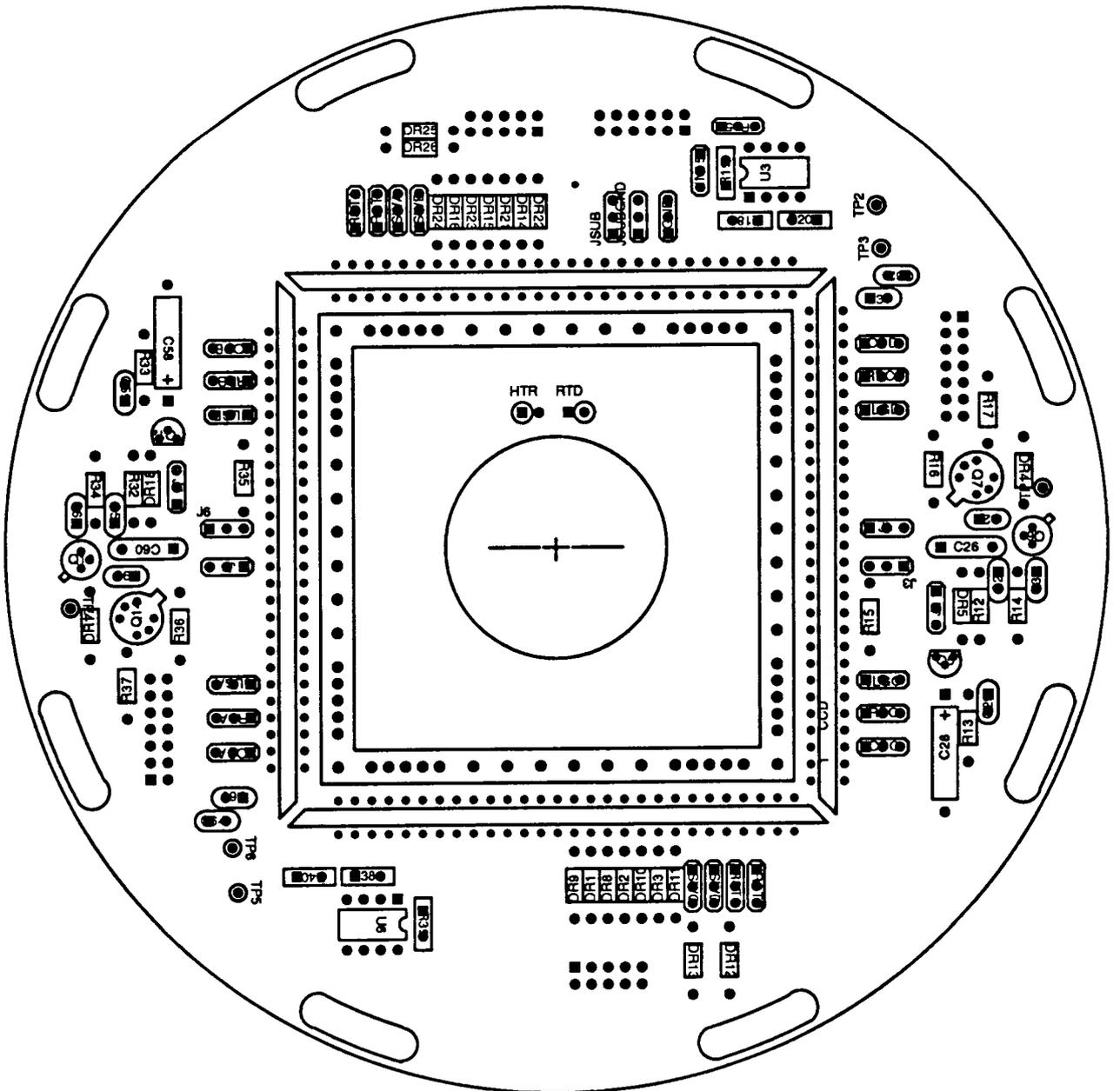
D

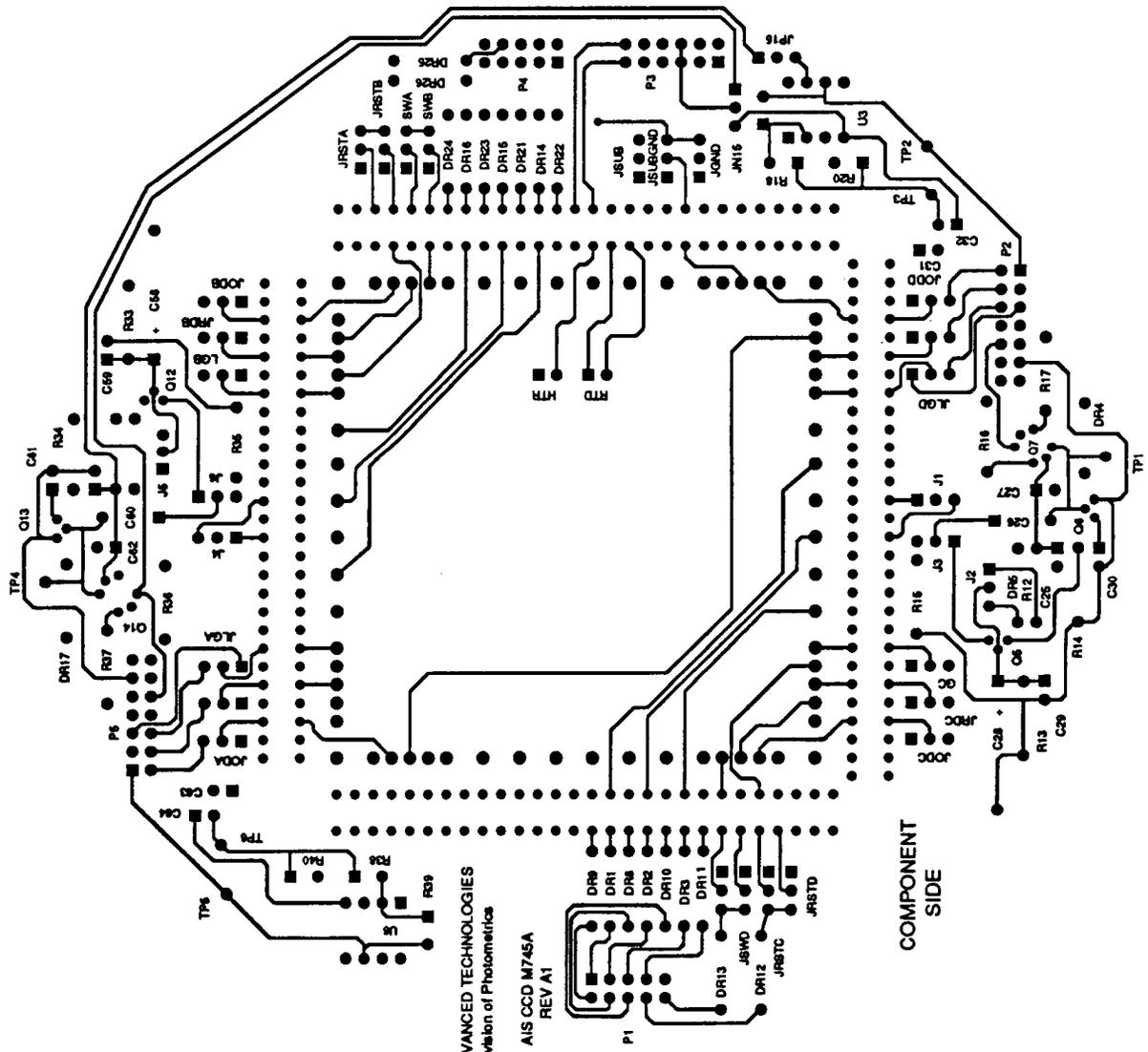
C

B

A

5.4.2 CCD Socket Card PCB Artwork

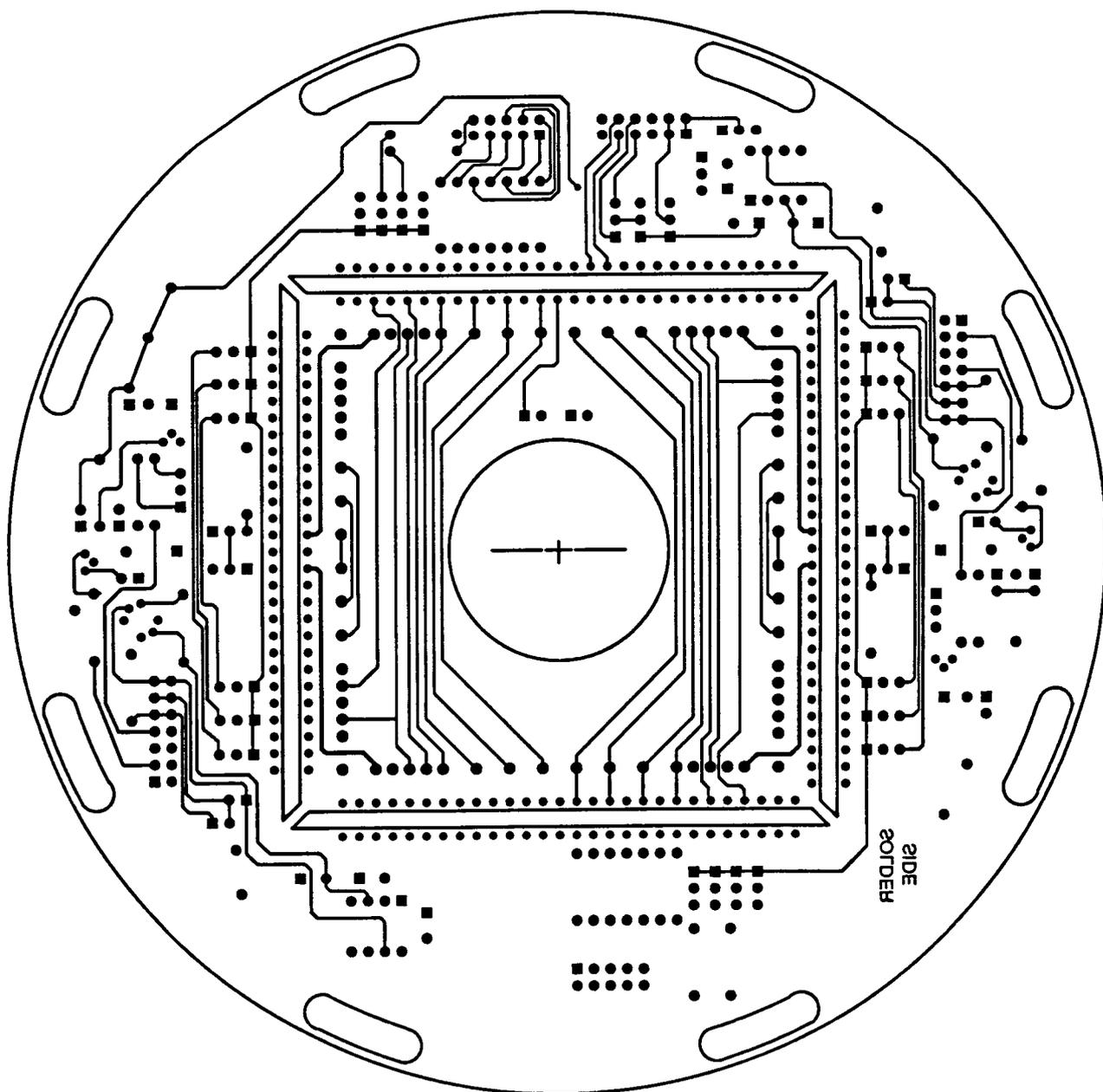


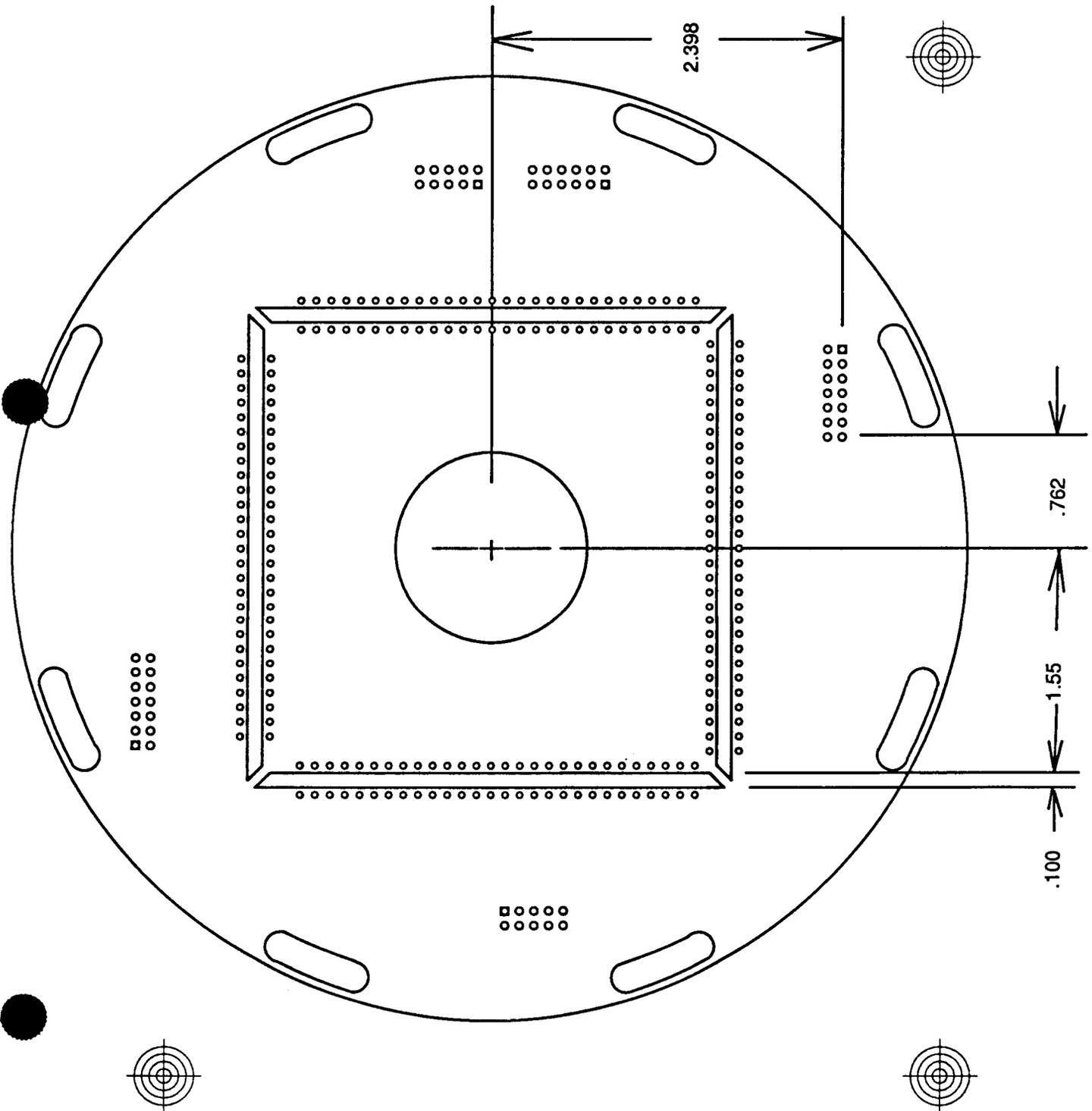


ADVANCED TECHNOLOGIES
Division of Photometrics

AIS CCD M745A
REV A1

COMPONENT
SIDE





5.5 Tektronix M745A CCD

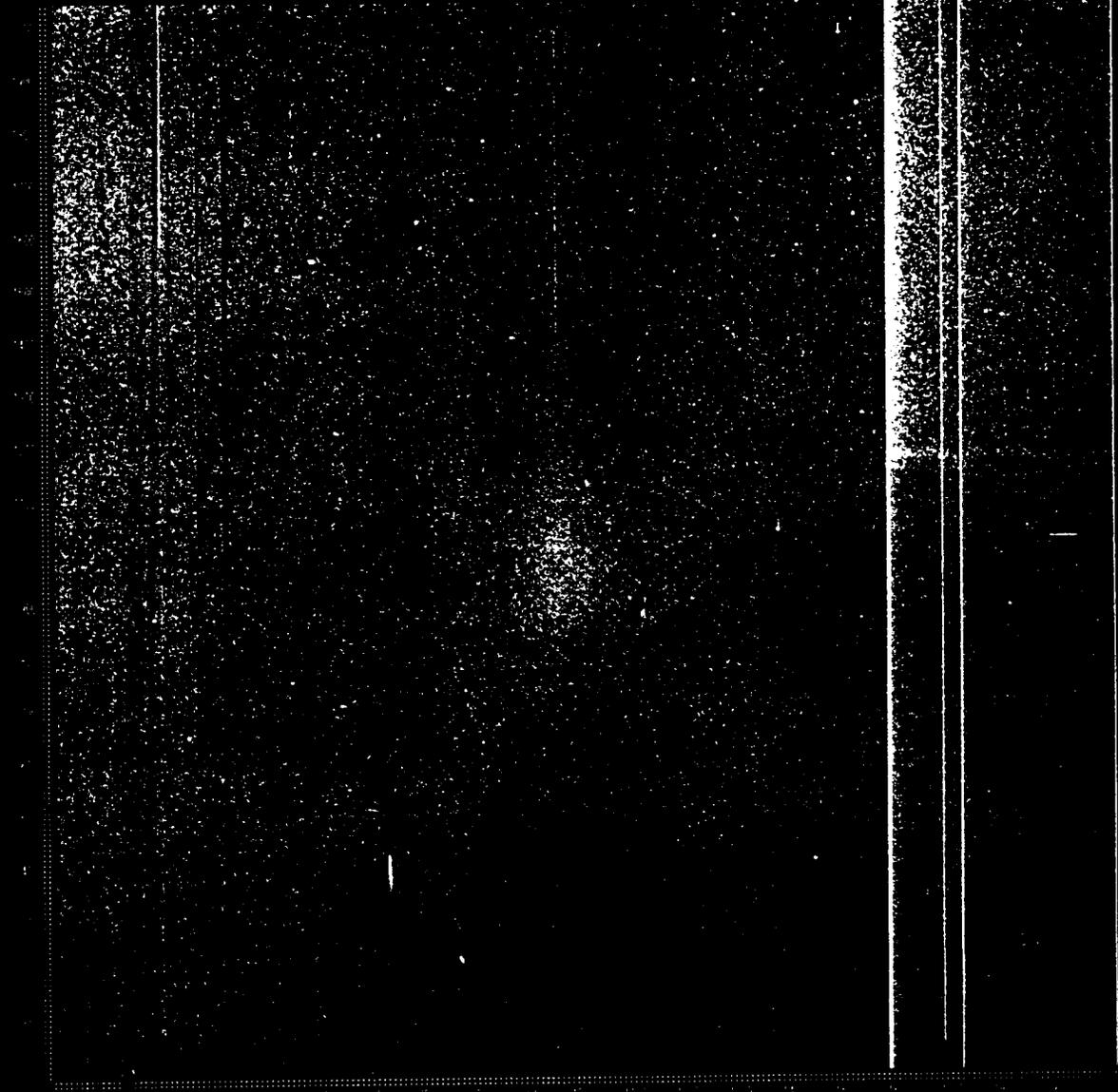
The CCD used in the STIS AIS1 camera system is a 2048 row by 2048 column device designed and fabricated by Tektronix specifically for the STIS project. The device is fabricated using three phase poly techniques in common use for CCD design. The pixel size is $21\mu \times 21\mu$. The serial register extensions contain 20 dummy pixels. The output amplifiers are of the 'lightly-doped drain' type.

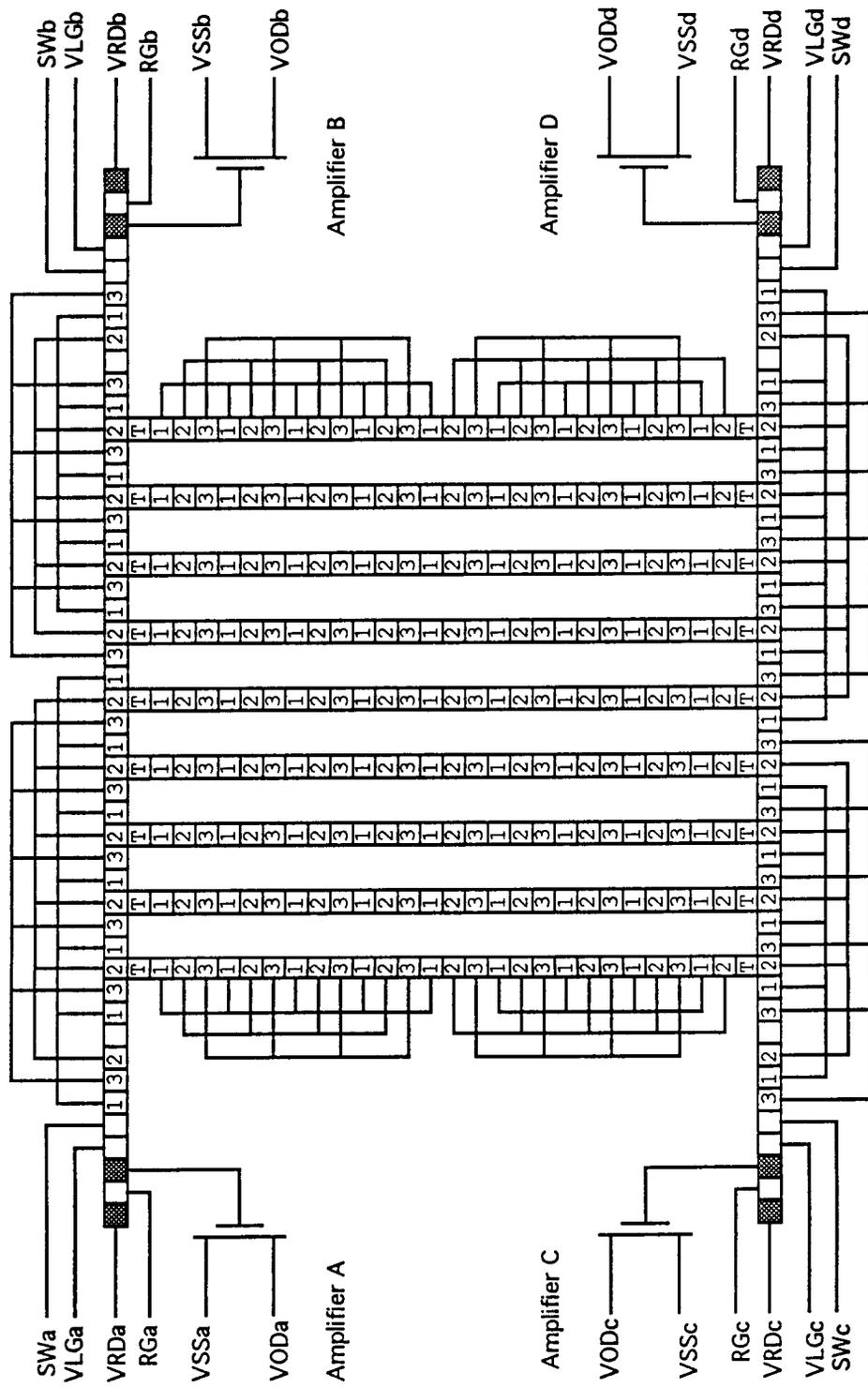


By accepting this product of the National Optical Astronomy Observatories (NOAO), one user declares that the material may not be used to state or imply the endorsement by NOAO of any NOAO product, process, or service.

ORIGINAL PAGE IS
OF POOR QUALITY

PRECEDING PAGE BLANK NOT FILMED





M745A CCD Functional Diagram

2048 Rows by 2048 Columns

20 Serial Extension Pixels per Output

May 22, 1990

M745A Frontside

Pin List

①	Package Ground		③⑤	N/C (GND)	
2.	N/C		36.	N/C	
3.	Chip Ground		37.	Chip Ground	
VRDC	4. VodC	C amplifier	38.	VodB	B amplifier
5.	Reset	C amplifier	39.	Reset	B amplifier
6.	Last Gate	C amplifier	40.	Last Gate	B amplifier
7.	Serial Phase 3C		41.	Serial Phase 3B	
8.	Serial Phase 1C		42.	Serial Phase 1B	
9.	Serial Phase 2C		43.	Serial Phase 2B	
10.	Serial Phase 2D		44.	Serial Phase 2A	
11.	Serial Phase 1D		45.	Serial Phase 1A	
12.	Serial Phase 3D		46.	Serial Phase 3A	
13.	Last Gate	D amplifier	47.	Last Gate	A amplifier
14.	Reset	D amplifier	48.	Reset	A amplifier
VDC	15. VodD	D amplifier	49.	VodA	A amplifier
16.	Chip Ground		50.	Chip Ground	
17.	N/C		51.	N/C	
①⑧	N/C		⑤②	N/C	
VDC	19. VoutD	D amplifier	53.	VoutA	A amplifier
VDA	20. VDDD	D amplifier	54.	VDDA	A amplifier
21.	D amplifier Ground		55.	A amplifier Ground	
22.	Summing Well D amplifier		56.	Summing Well A amplifier	
TC	23. Transfer gate		57.	Transfer gate	
24.	Parallel Phase 3		58.	Parallel Phase 3	
25.	Parallel Phase 1		59.	Parallel Phase 1	
26.	Parallel Phase 2		60.	Parallel Phase 2	
27.	Parallel Phase 2		61.	Parallel Phase 2	
28.	Parallel Phase 1		62.	Parallel Phase 1	
29.	Parallel Phase 3		63.	Parallel Phase 3	
TC	30. Transfer Gate		TC	64. Transfer Gate	
31.	Summing well B amplifier		65.	Summing well C amplifier	
32.	B amplifier Ground		66.	C amplifier Ground	
VDC	33. VDDB	B amplifier	VDC	67. VDDB	C amplifier
34.	VoutB	B amplifier	VDC	68. VoutC	C amplifier

Please note carefully that the pad list and the pin list are not identical!!

○ CORNERS

Static Handling Procedures:

All CCD devices should be stored or transported in conductive material so that all exposed leads are shorted together. CCD devices should NOT be inserted into conventional plastic "snow" or plastic trays of the type used for storage and transportation of other semiconductor devices.

All CCD devices should be placed on a grounded bench surface and the operator should be grounded prior to handling the device. This is done most effectively by having the operator wear a conductive wrist strap.

Whenever handling a CCD circuit, DO NOT WEAR ANY NYLON CLOTHING.

DO NOT insert or remove CCD devices from test sockets with the power applied. Check all of the power supplies to be used for testing CCD devices and be certain that there are no voltage transients present.

When any lead straightening or hand soldering is necessary, provide ground straps for the apparatus used.

Cold chambers using CO₂ for cooling should be equipped with baffles and devices MUST be contained on or in conductive material.

2048 x 2048 Package Configuration (68 Lead Metal Package)

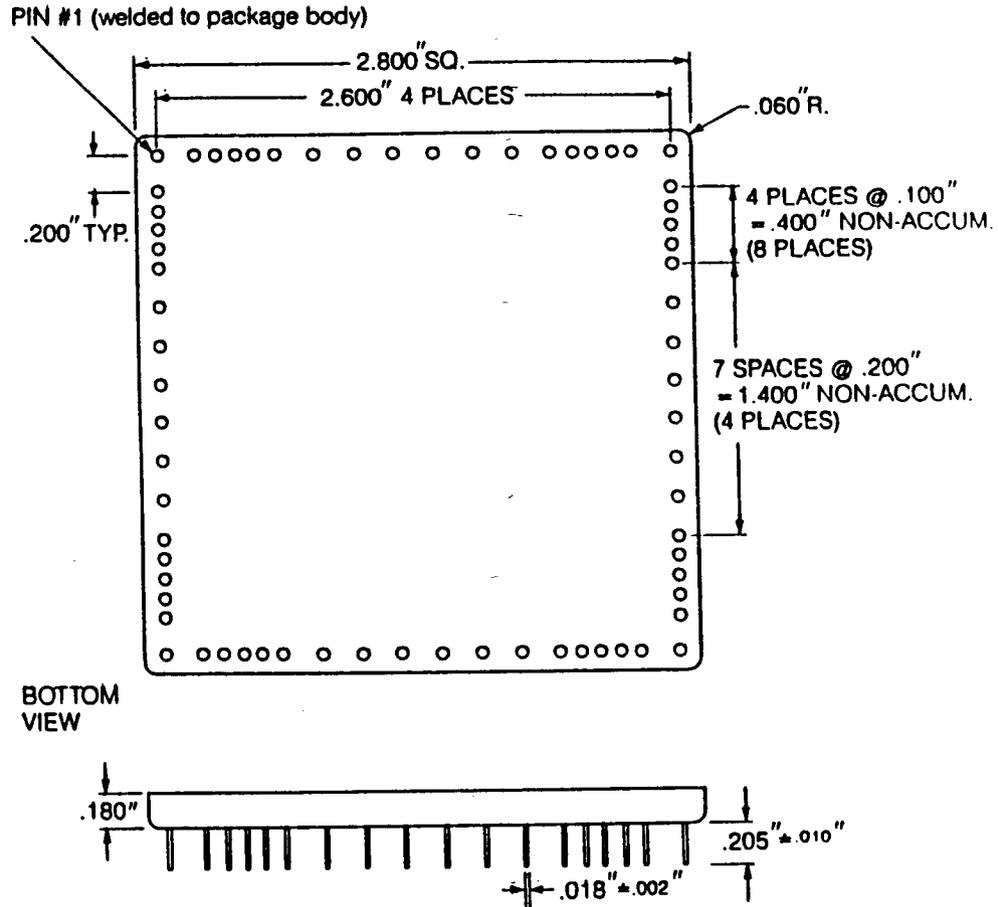


Figure 1

6. VMEbus Interface

6.1 Overview

The AIS1 camera is interfaced to the Sun 4/260 workstation via custom VMEbus interface cards. Two identical cards are used, one for each port of the CCD in use. The VMEbus cards are used for communication with the camera controller and to collect the image data from the clock/analog modules. The cards are modified versions of the Photometrics VME200a camera controller. They are provided with a custom adapter board designed for this camera system that can be used to interface the camera to the workstation via fiber optic cables. The VME200a cards and the adapter boards will be discussed below.

6.2 Connector Pinouts

There are six connectors on the face panel of the VMEbus interface cards. At the top is P6 a DB9 style connector used to connect the CCD data and data clock lines from the clock/analog module to the VMEbus interface. The pinout of this connector is shown below.

VMEbus Adapter P6 Pinout

1	clock +
2	clock -
3	data -
4	data +
5	ground
6	ground for cable shield
7	unused
8	unused
9	unused

P7 is an 'ST' style fiber optic connector used for the CCD data from the clock/analog module and P8 is also an 'ST' style connector used for the data clock.

P9 is an 'ST' type fiber optic connector used for the serial communications transmit channel, and P10 is an 'ST' connector used for the receive portion of the serial communications.

J1 is a female DB9 type connector used for serial communications over wire. The serial communications cable connects

to the 'RS422' connector on the controller module. The pinout of the J1 connector is shown below.

VMEbus Adapter J1 Pinout

1	transmit+
2	transmit-
3	receive+
4	receive-
5	unused
6	unused
7	unused
8	unused
9	unused

6.3 VME200a Data Interface Description

A complete description of the VME200a camera controller card from Photometrics is beyond the scope of this document. In this document, a simple description will be provided with emphasis placed on the modifications made to the card to operate the AIS1 camera system.

The VME200a camera controller is a VMEbus compatible interface card used in conjunction with Photometrics series 200 camera systems. It includes two on board processors, a 68HC11 and a DSP56001 (design ideas borrowed from the development of the AIS1), and up to 32 MB of DRAM memory. The memory on the interface card is dual ported between the VMEbus and the camera data interface. The memory appears as A32/D32 memory to the VMEbus. The cards used in the AIS1 system are populated with 8 MB each.

The 68HC11 processor on the board is used, in this application, as a very smart serial communications controller. All communication with the camera controller are accomplished through the 68HC11 on one of the two interface cards provided. The 68HC11s also perform various housekeeping tasks on the cards including preparing the DRAM array to accept the camera data. On the second card, these are the only tasks the 68HC11 performs.

The DSP56001s have been removed since they were not needed in this system.

The serial data from the camera is converted into parallel form on the interface card and stored into memory as it arrives. FIFO memory devices assure that data will not be lost during DRAM refresh cycles. The 8 MB of memory appear as a single block. Neither the on board 68HC11 nor the VMEbus master have any control over

where the data will be placed except to reset that location to the base address of the card. This is typically done before the camera is passed a command that would cause it to produce image data. See the 'C' Language Interface section of this document for more details concerning the operation of the VMEbus interface card.

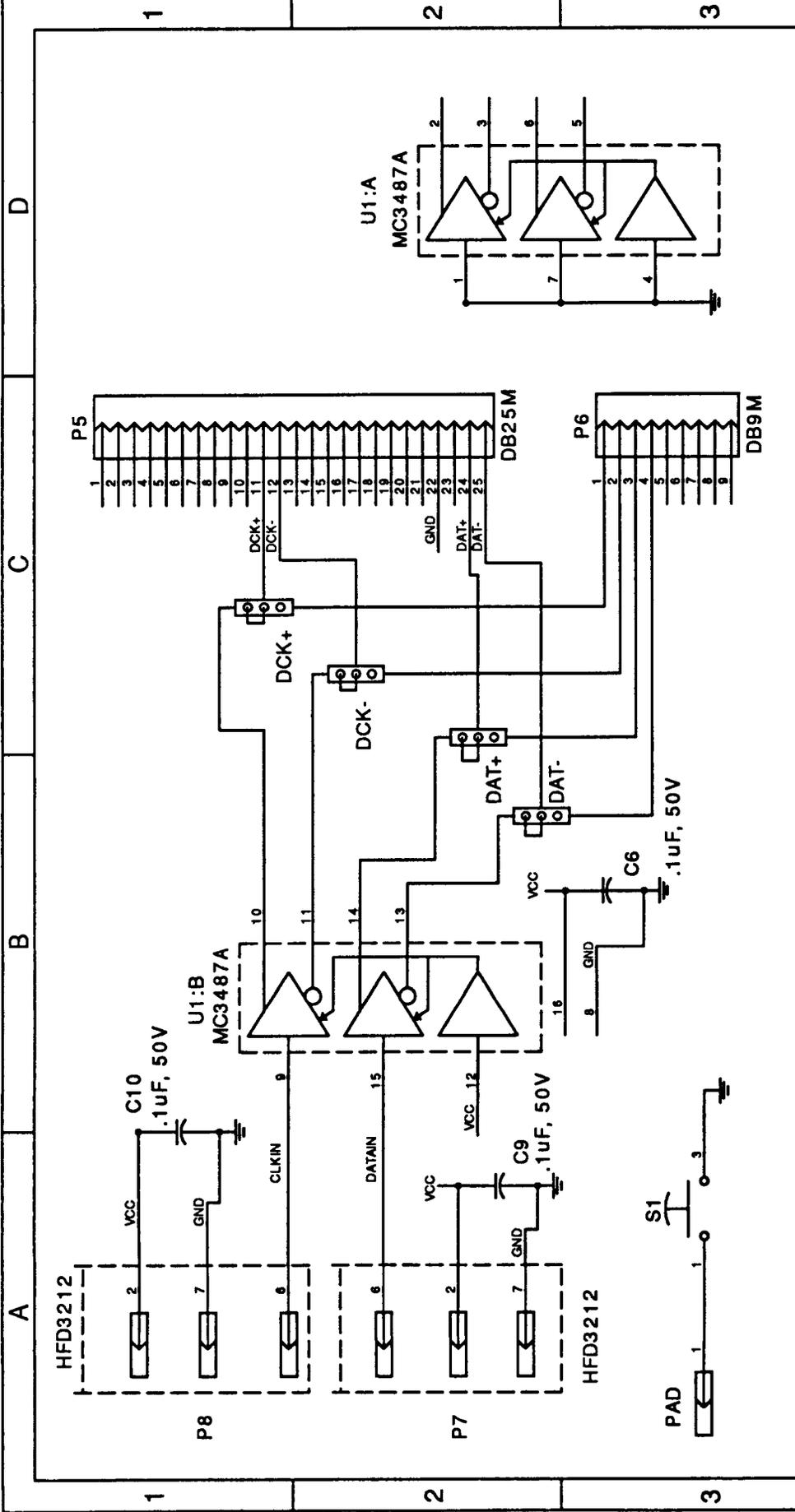
6.4 Fiber Optic Adapter

To accommodate the fiber optic communications and data channels, which are not a standard part of the VME200a, a special adapter card was designed. This card, a simple two sided circuit card, contains the necessary components to convert the differential TTL and RS232 level signals provided and expected by the VME200a into the RS422 and optical signals needed to interface to the camera.

The card bolts into place at the front end of the 6U to 9U VMEbus adapter used to install the VME200a in the Sun workstation.

Complete schematics of the fiber optic adapter card are provided in the following section. The circuit board artwork follows.

6.4.1 VMEbus Fiber Optic Adapter Schematic



ADVANCED TECHNOLOGIES
A DIVISION OF PHOTOMETRICS LTD.

VME ADAPTOR CARD

SHEET 1 OF 2
CONTRACT/ JOB NUMBER

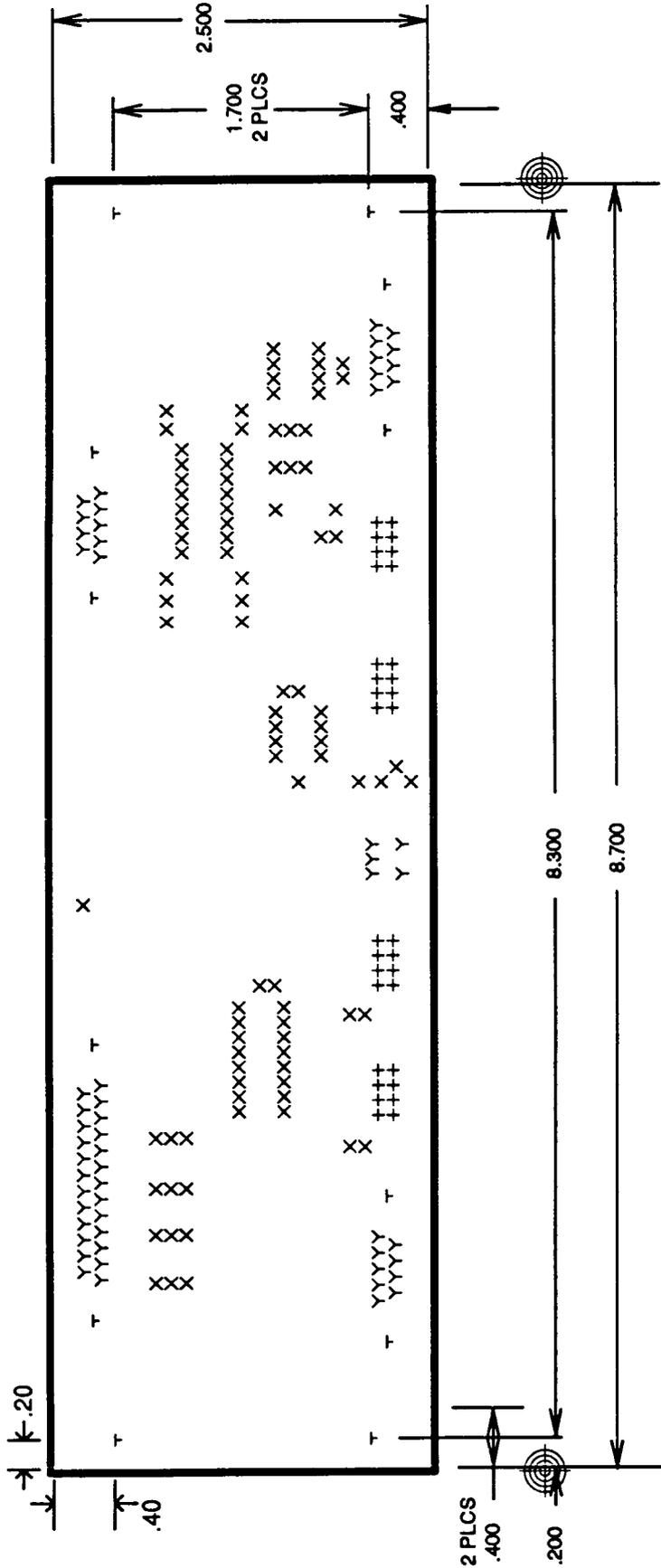
A

NEXT ASSY.	USED ON
APPLICATION	
DRAWN	
CHECKED	
RELEASED	

REVISION HISTORY		DATE
LTR	DESC.	APP.

NOTES

6.4.2 VMEbus Fiber Optic Adapter PCB Artwork



ADVANCED TECHNOLOGIES
A DIVISION OF PHOTOMETRICS LTD.

VME
ADAPTOR CARD

SHEET OF
CONTRACT/JOB NUMBER

DRAWING NUMBER
DRILL

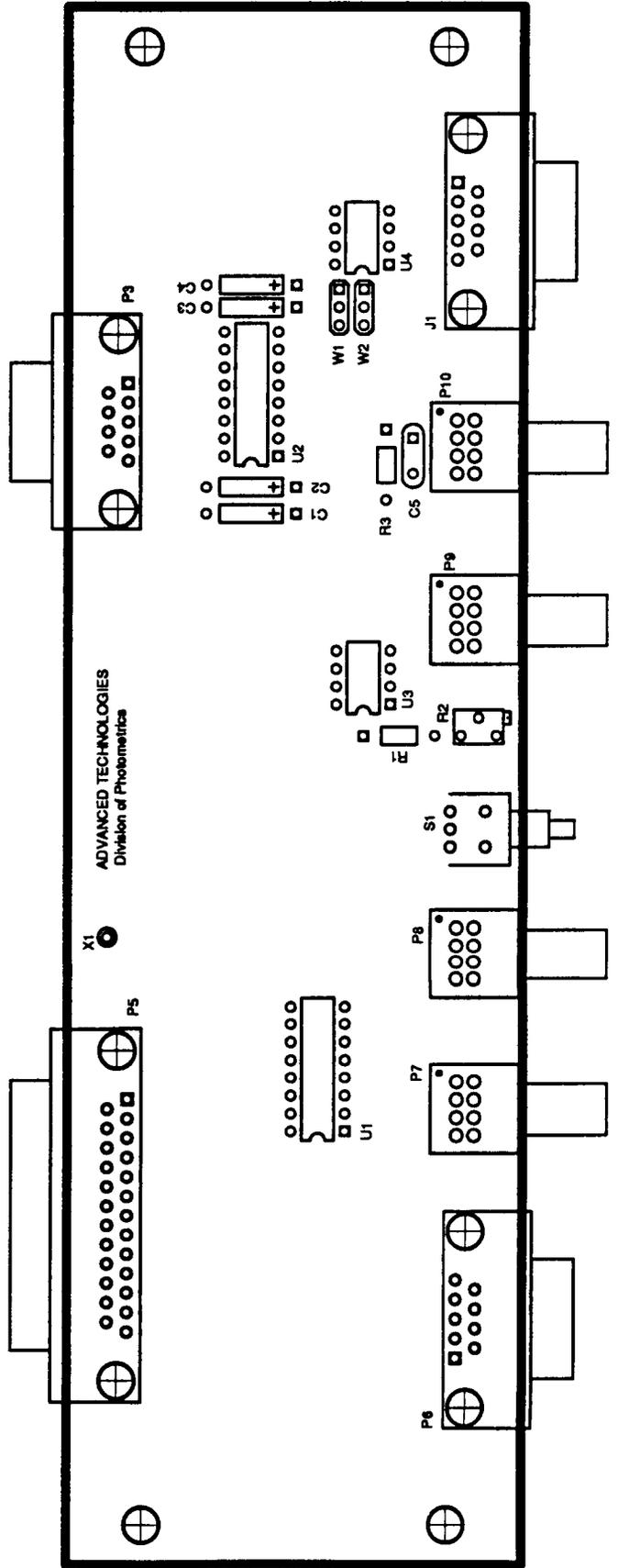
USED ON
APPLICATION

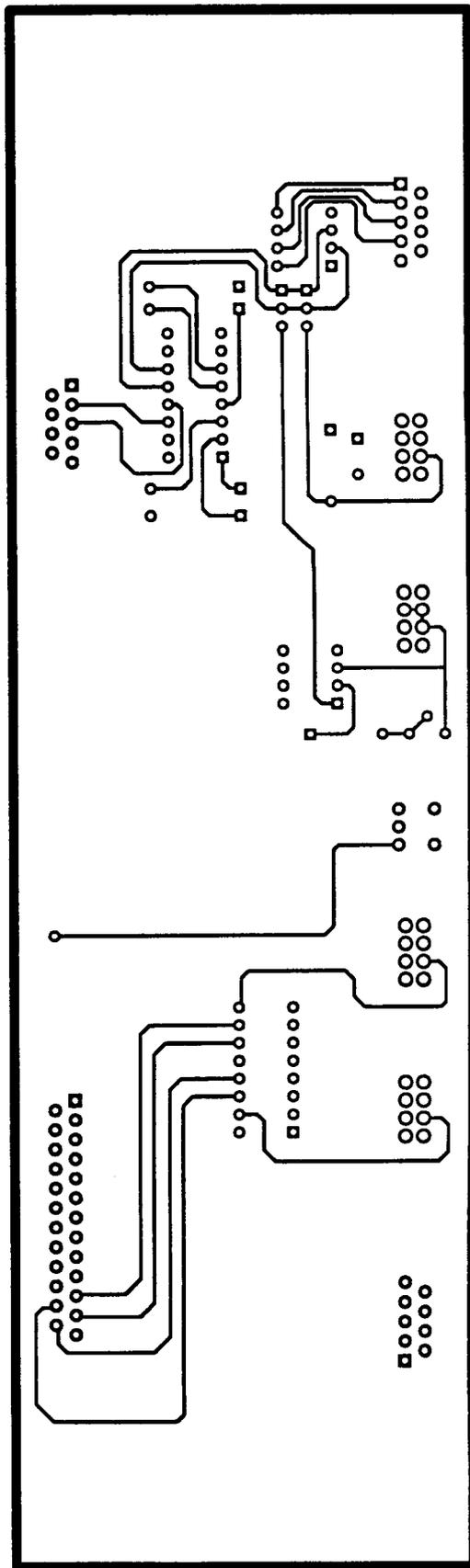
NEXT ASSY.
DRAWN
CHECKED
RELEASED

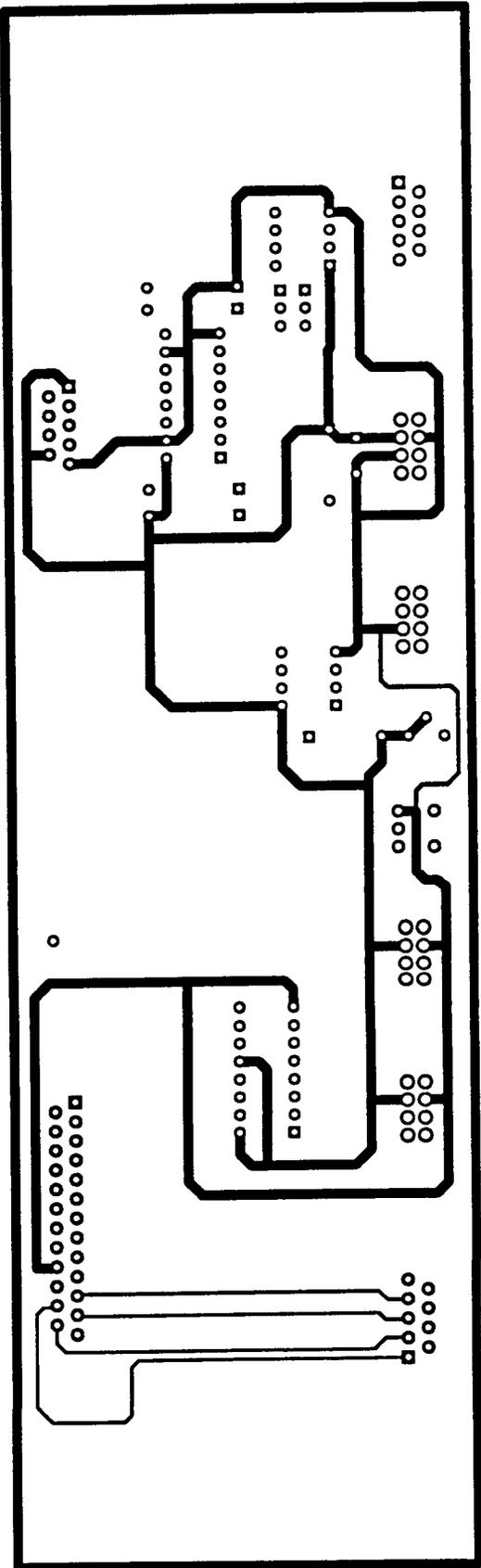
HOLE CHART	
SYM	SIZE
+	.032
X	.038
Y	.043
T	.125

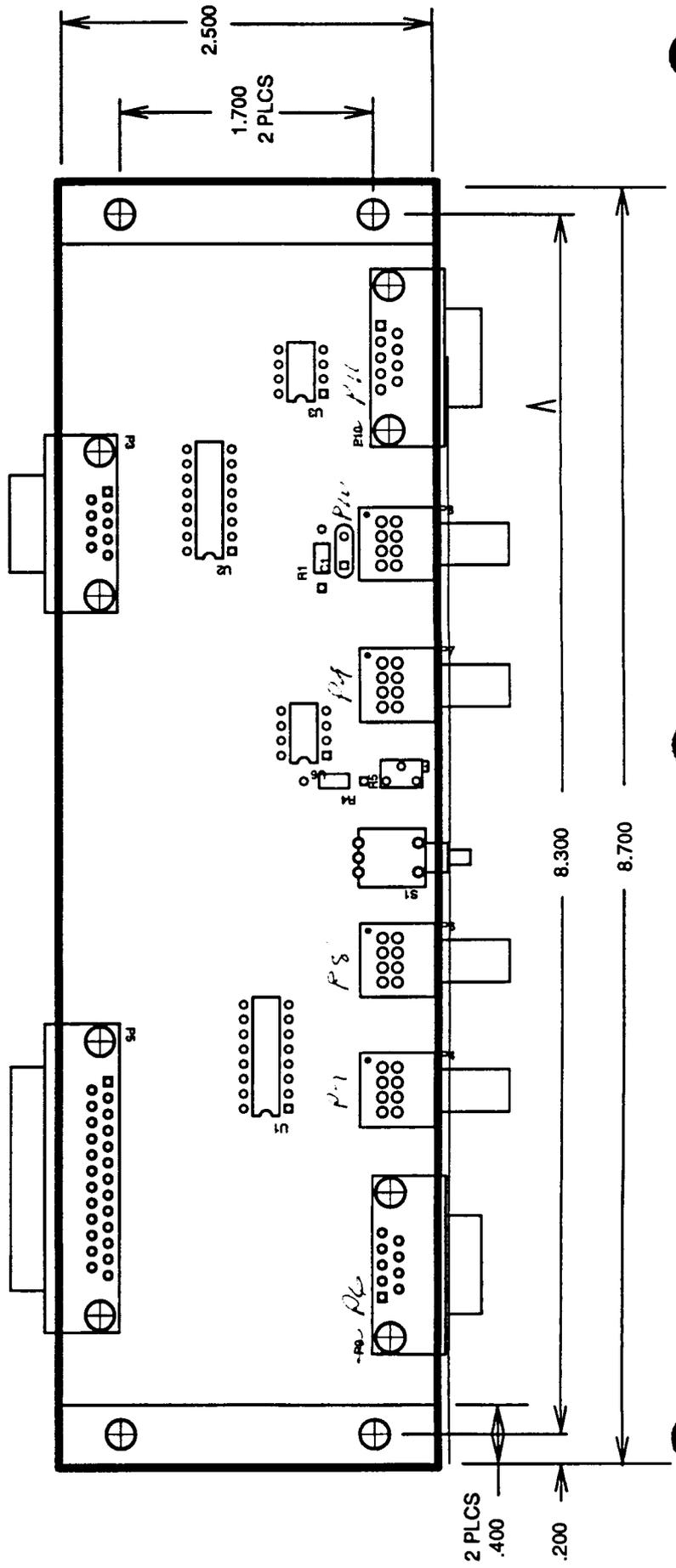
REVISION HISTORY			
LTR.	DESC.	APP.	DATE

- NOTES**
1. LAYER ASSEMBLY ORDER.
 2. ALL DIMENSIONS IN INCHES.
 3. ALL DIMENSIONS IN INCHES.
 4. FINISHED CIRCUIT BOARD THICKNESS TO BE .062 OVERALL.
 5. BASE MATERIAL IS 1 OZ COPPER CLAD FR-4 EPOXY.
 6. DRILL DIA. 1/16" PER UL-A-158A.
 7. ALL DIMENSIONS FROM COMPONENT SIDE WITH WHITE EPOXY INK.
 8. ALL DIMENSIONS FROM COMPONENT SIDE.
 9. SOLDER MASK TOP AND BOTTOM WITH BRASS (BLU).
 10. MINIMUM TO BE .016 INCH.
 11. ALL EXPOSED COPPER TO BE PLATED WITH A MINIMUM OF .002 INCH THICK LEAD CONTAINING 1% TIN.
 12. HOLE THRU HOLE PLATING TO BE .016 INCH COPPER.
 13. HOLE TOLERANCES ± .002 - .004 AFTER PLATING.
 14. TOLERANCES UNLESS OTHERWISE SPECIFIED.
 15. DRILL DIA. .016
 16. ALL HOLES WITHIN .002 THRU .016 INCHES IN DIAMETER TO BE .016 OVERALL.
 17. FINISHED CIRCUIT BOARD THICKNESS TO BE .062 OVERALL.
 18. BASE MATERIAL IS 1 OZ COPPER CLAD FR-4 EPOXY.
 19. DRILL DIA. 1/16" PER UL-A-158A.









8. Sun Microsystems 4/260 Workstation

The Advanced Imaging System CCD based camera includes a workstation class computer for data acquisition, analysis, and storage. This computer is a Sun Microsystems model 4/260. A complete description of this computer is beyond the scope of this document. A brief description will be provided in this section to familiarize the camera system operator with some of its basic features.

The Sun 4/260 CPU is based on a SPARC type processor and is rated at approximately 12.5 million instructions per second. The processor is installed in a 12 slot VMEbus compatible chassis. The CPU is a two card set occupying the left-most position in the chassis.

The system includes an eight bit color display with a resolution of approximately 1100 x 900 pixels. This display adapter is connected to a 19 inch color monitor.

48 MB of memory are installed in the system to allow it to more readily support the 2048 x 2048 pixel images produced by the camera system, which occupy 8 MB of memory per frame.

A 347 MB internal disk drive is included. After the storage required for the UNIX operating system and the IRAF image processing package, approximately 60 MB of storage remains.

An external storage module provides another 550 MB of disk space and a 650 MB removable read/write optical disk for archival storage and data transport.

A low density tape drive is also included. This tape drive supports the DC300 type of media, and provides 60 MB of storage per tape.

The workstation is connected to the camera electronics via two VMEbus interface cards installed in the chassis. These interface cards are described in greater detail elsewhere in this document.

9. System Software Overview

The AIS1 camera system includes custom software programs that run on the various processors in the system. The camera itself contains two processors, each of the VMEbus interface cards includes one, and there is the Sun Microsystems workstation itself. Custom software has been developed for each of these as a part of this project. Each of these programs will be discussed in greater detail in the remainder of this document. This section serves as an introduction to that information.

There are four microprocessors embedded in the camera electronics. Two of these, a 68HC11 microcontroller (MCU) and a DSP56001 digital signal processor (DSP), are contained in the camera controller electronics. The camera controller is the digital timing generation system, typically mounted in a small chassis bolted to the camera head. Also, there is a 68HC11 CPU located on each of the VMEbus interface boards. In the STIS AIS1 camera, there are two VMEbus interface boards, one for each CCD port read. The processors on the two boards execute identical code, but are used somewhat differently in the workstation software.

There are several custom software programs available on the Sun Microsystems 4/260 workstation. There is a set of 'C' language control routines that a programmer might use to control the camera, or as an example of how one might do so. There is a set of utility programs written using the 'C' interface routines, and there is an interface to the Image Reduction and Analysis Facility (IRAF) image processing software.

The different portions of the camera system software are discussed briefly in the following paragraphs of this section of this document and discussed in greater detail in reverse order in the following sections.

9.1 Embedded Software

The AIS1 camera electronics unit contains two digital processors. Each of these has particular functions in the overall operation of the camera system, and there is a high level of interaction between them. Dedicated software has been developed for each processor, and each package will be discussed in turn in this section.

The 68HC11 microcontroller (MCU) performs all communication between the host computer and the camera electronics. It does so over a 9600 baud RS-422 serial data link. The host computer issues

commands to the MCU to set parameters, return status, and perform image acquisition functions. The MCU, in turn, commands the DSP56001 Digital Signal Processor (DSP) to perform whatever tasks are necessary in order to carry out the host's request.

The DSP generates all CCD clock timing and pixel conversion control signals. Functions include clearing the CCD, reading the data off the CCD, and a variety of lower level charge shifting and data acquisition operations. The DSP is also responsible for setting the CCD clock and output amplifier operating point voltages. The DSP performs all operations under the control of the MCU.

These programs are described in some detail in following sections of this document. Sections 5 and 6 contain information regarding the 68HC11 FORTH program and section 7 contains information regarding the DSP program. These more detailed descriptions are of interest to those who would like to create programs to run on the workstation to control and collect data from the camera. It is also of use to those who would like to interface directly to the cameras via an ASCII terminal in order to modify, maintain, or debug the camera software and hardware.

The 68HC11 CPUs located on VMEbus interface execute a software program developed for use in this application and for use in the Advanced Technologies VMEbus interface boards used with other Advanced Technologies cameras. This software is discussed in general terms in section 8 of this document, and the source is listed in section 8a.

9.2 'C' Language Control Library

A set of 'C' language routines is provided in the form of two source code files: ATDcamera.c and ATDcamera.h. The source code in these files serves as a demonstration of how a programmer may control the camera and collect image data using the VMEbus interface cards.

These files are discussed in section 4 of this manual and the source is listed in section 4a.

9.3 UNIX utility Programs

A small set of utility programs is provided for manipulating the camera from the UNIX command line. These utility programs were written using the 'C' language control library, and serve as practical examples to the camera system programmer in addition to their use as camera configuration utilities.

Included are programs to initialize the camera system parameters, to send individual command strings to the camera from

the UNIX command line, and to send text files containing camera commands to the camera. These programs are provided in order to assist in supporting some of the programmable features of the camera through a simple command line interface which would not require programming

AISsay is a program that allows the user to transmit any text string to the camera through the serial link on the VMEbus interface cards.

AISsetup is a program that allows the user to set the various camera parameters and to download ASCII text files containing CCD control voltage settings and CCD timing information to the camera.

AISfile is a program that allows the user to send any arbitrary text file to the camera through the serial interface. These files would typically consist of FORTH commands for the camera to execute or compile.

These programs are discussed in greater detail in section 3 of this document.

9.4 IRAF interface

The camera may be controlled from within the command language of the Image Reduction and Analysis Facility (IRAF) developed at the National Optical Astronomy Observatories. IRAF is a complete data reduction and analysis package used by many astronomers. The camera may be controlled and data acquired using a set of IRAF commands developed at the Steward Observatory.

This interface allows the user to perform simple data acquisition functions, and allows the user to then manipulate and analyze the images acquired using IRAF's various tools. It does not directly support the more advanced features of the camera. No provision is made for manipulating the CCD control voltages or timing from within IRAF, for example, but the command set is complete enough for typical operation of the camera in the field. The user wishing to simply use the AIS1 camera to acquire image data will likely use the IRAF software. Further detail of the IRAF interface software is given in the next section of this document.

10. IRAF Interface

The AIS1 camera can be controlled using the Image Processing and Reduction Facility (IRAF) developed at the National Optical Astronomy Observatories (NOAO) in Tucson, AZ. The camera interface software is based on a model provided by Skip Schaller of Steward Observatory in Tucson, AZ. Mr. Schaller has developed a complete package for interfacing the IRAF command interface to the telescope, instrument, and detector in use. The software package is called *ccdacq*. We acknowledge his efforts and appreciate the time he has spent assisting in the interface of the AIS1 camera to the IRAF package.

The Sun Microsystems 4/260 workstation included in the camera system contains the full IRAF package version 2.9.1. A newer version has since become available, but has not yet been installed. The system runs under the 'Sunview' graphical user interface. In the future this will likely change to the 'Open Look' or 'OSF/Motif' interface.

The Sun workstation runs the SunOS 4.1.1 version of the UNIX operating system. The exact configuration of the operating system and the various user IDs and other re-configurable portions of the operating system will likely change with time. In this document, we will describe the system as it was configured in November of 1992, and will make an effort to note where the description may change.

10.1 Starting the Sun 4/260

If the Sun workstation has been powered down, then it is necessary to restart the machine. The computer should reboot automatically on power-up, but if it should fail to do so the user will see the boot prompt '>' on the display. The user wishing to reboot the machine does so by typing 'b' at the boot prompt. If the boot process is successful, then the user will see a login prompt. At this point the user may login. Various login names are available. The user who does not have her own login name but who wishes to use the camera system, may login as 'iraf' by typing iraf at the login prompt. This should bring up the Sunview user interface and start the IRAF 'cl' command interface. If it does not then type 'sunview' at the new prompt. The exact login process and the appropriate login names may vary as the system is modified. The 'iraf' login was valid when the system was delivered. It may be deleted or modified and others may be added as time passes.

10.2 Running IRAF

A complete description of the IRAF program is far beyond the scope of this document. The user is referred to the IRAF user manuals for a complete description.

After "logging in", the user should type 'ccdacq' to enter the camera control package and type '?' to get a listing of the available commands. The commands will be briefly described below. The user may get a more detailed description of any command by typing 'help *task*' where *task* is the name of the command in question. A complete listing of the various help documents is provided at the end of this chapter.

10.3 *ccdacq* commands

The *ccdacq* IRAF package contains sufficient flexibility to allow the user to acquire images of a variety of types and to attach appropriate header information to them. The basic operation is the same as for any other IRAF package in that there are a variety of parameters associated with each task. These parameters will be discussed in greater detail in later paragraphs. A summary of the commands available in the *ccdacq* package follows. These paragraphs are based on the information in the *ccdacq* help pages available from the command line while running IRAF.

- abort** Abort an exposure in progress. Close the shutter and do not record data. After interrupting the **observe** task, terminate the exposure timer and close the shutter. Flush the detector. No data is recorded. You may **abort** with or without having first done a **pause**.
- comps** Make a series of *imagetype*="comparison" exposures. Call **observe** to make a series of comparison observations, turning on the selected comparison lamp, and opening the shutter with the given time between each chip preparation and chip readout.
- darks** Make a series of *imagetype*="dark" exposures. Calls **observe** to make a series of dark observations, each with the given time between chip preparation and chip readout. The shutter will not open.
- detector** Check the current detector temperature.

- expose** Run an expose cycle on the detector. Use the parameters set with the prepare command. Called as necessary by the higher level functions. Used only for debugging.
- flats** Make a series of imagetype="flat" exposures. Calls **observe** to make a series of flat observations, opening the shutter for the current exposure time between each chip preparation and chip readout.
- instrument**
The instrument command is not associated with the operation of the CCD camera.
- mores** Call **observe** to make a series of observations, using the current default parameters.
- observe** Make an observation, acquiring data from the CCD. **Observe** connects to the detector, instrument, and telescope servers and creates an IRAF image containing the data from the detector. All appropriate information is entered into the header. An exposure of more than ten seconds may be suspended by keyboard interrupt (CTRL-C). You then may use one of the tasks, pause, resume, stop, or abort. Use help to get information on their usage.
- pause** close the shutter and suspend the exposure timer
After interrupting the observe task, suspend the exposure timer and close the shutter. Useful for waiting out passing clouds. May be followed by resume, stop, or abort.
- prepare** Initialize the camera controller with the latest readout parameters. Prepare the interface boards to collect CCD data. Called as necessary by the higher level functions such as **observe**. Used only for debugging.
- readout** Read the data off the CCD. Use the parameters set with the prepare command. Is called as necessary

by the higher level commands such as **observe**.
Used only for debugging.

resume Open the shutter and continue the exposure timer. Used after interrupting the **observe** task and suspending the exposure timer with **pause**. Continue the exposure timer from the point where it was suspended, and reopen the shutter, if it was open originally.

rvshift Run a reverse vertical shift cycle on the detector. Use the parameters set with the prepare command. Used only for debugging.

stop Close the shutter and immediately read out the chip. After interrupting the **observe** task and suspending the exposure timer with **pause**, prematurely terminate the exposure immediately, close the shutter, and readout the chip. Record the data. Useful if you originally over-estimated the exposure time. You may **stop** without doing a **pause** first.

telescope Initiate physical action (if possible) on the telescope and report status to STDOUT. Not a part of the CCD camera control interface.

tests Make a series of test exposures. First tries to delete an image named test. Then calls **observe** to make a series of test observations, opening the shutter with the given time between each chip preparation and chip readout. An image named test is left on the disk. Does not alter the current rootname or sequence parameters.

zeros Make a series of imagetype="zero" exposures. Calls **observe** to make a series of zero observations. The shutter will not open, and the chip will be readout immediately after the prepare cycle, giving an exposure time of zero seconds. The zero exposure is used to determine the bias level. Sometimes also called a bias exposure.

detpars Edit the detector parameters, see discussion below

instrpars Edit the instrument parameters, see discussion below

obspars Edit the observing parameters, see discussion below

telpars Edit the telescope parameters, see discussion below

10.4 ccdacq parameters

As stated above, the action of the tasks in the *ccdacq* package is determined by sets of parameters. These parameters are categorized into four groups: detector parameters, instrument parameters, observing parameters, and telescope parameters.

Detector parameters are associated with the CCD itself and the portion of the CCD to be read.

Instrument parameters are not directly associated with the CCD camera. They have to do with other parts of the instrumentation in use which may be under IRAF control. These might include filters, or other optical and electronic components. They will not be discussed further in this document.

Observing parameters have to do with the nature of the next observations and the files so created. See the detailed discussion below and the on line help for more information.

Telescope parameters have to do with the position and motion of the telescope. They are relevant if the telescope at the sight is under IRAF control. They are not discussed in any greater detail in this document. See the on line help or any documents specific to your telescope for further information.

The various IRAF parameters are edited using the IRAF parameter editing task 'epar'. Detector parameters may be edited by entering the command 'detpars' or 'epar detpars', instrument parameters may be edited by entering 'instrpars' or 'epar instrpars', and so on for the 'obspars' and the 'telpars'. See the IRAF documentation for more details on editing IRAF parameters.

10.4.1 Detector parameters

The detector parameters have to do primarily with the CCD and the image format for the next acquisition. Each will be discussed briefly in turn. These descriptions are essentially the same as those included in the help file for *detpars*, with additional notes regarding their use with this camera system.

- firstcol** : The first pixel to be read out of each row in relation to the beginning of the serial register. The first column on the CCD is considered column 1. If it is desired to read a subregion of the CCD device, then **detcol** may be set to a number other than 1. The CCD in the STIS AIS1 camera has a total serial dimension of 2048 pixels and so **firstcol** can be anything from 1 to 2048.
- lastcol** : The last pixel to be read out of each row in relation to the beginning of the serial register. The value of this parameter must be greater than the value of the **firstcol** parameter. This value may also range from 1 to 2048.
- firstrow** : The first row that the user wishes to read out of the CCDs parallel register. The first row on the CCD is considered row 1. The CCD in the STIS AIS1 camera has a parallel dimension of 2048, but the parallel register is split and the device is read through two ports. Each half of the CCD is clocked simultaneously by the camera controller. Subregions of the device must be centered around the CCDs parallel split. This parameter may range, therefore, from 1 to 1024.
- lastrow** : This parameter represents the last row to be read off the CCDs parallel register. As noted above, the parallel register is split and the device is read out of two ports, and subregions of the device must be centered around the parallel split. Therefore, the value of this parameter, though it may range from 1025 to 2048, must be equal to $(2048 - \text{firstrow})$. If it is not, the camera software will abort the next observation and issue an error message.
- colbin** : The column binning factor. Its value may range from 1 to 65535. Numbers greater than 2048 will not be useful however, as that is the total number of columns on the CCD imager.

- rowbin** : This parameter represents the row binning factor. Its value may range from 1 to 65535. Values greater than 1024 will not be useful however, since that is the maximum number of rows on each half of the CCD imager.
- preflash** : The preflash time in milliseconds. At this time the STIS AIS1 camera does not support a preflash.
- gain** : The gain factor.
- detinfo** : The name of a file to be read and copied into the header. See the help for detpars for a more complete description.
- detcap** : The name of the capabilities file, which describes the devices which can be selected, their physical characteristics, and the protocol used to contact their server. This parameter should typically be set to "ccdacq\$detcap".
- detname** : The device name corresponding to an entry in the capabilities file. The detector name for the STIS AIS1 camera is "AIS1".

10.4.2 Observing parameters

- imagetype** : The type of image being recorded. Choose among: zero, dark, flat, focus, comparison, standard, object. The first two in the list do not open the shutter. All others do.
- exposuretime** : The exposure integration time in seconds. This will be passed to the camera controller through the serial interface prior to acquiring the image.
- objecttitle** : The title to be given the object.

Like the camera controller, the *ccdacq* program treats each image acquisition as a sequence of acquisitions. Two parameters are associated with

this process. The first is the rootname for the files which will be generated and the second is the number of images in the sequence. The files will be given names of the form *rootname1*, *rootname2*, etc.

rootname : The rootname is the name to be given to the IRAF image or images to be recorded onto disk.

sequence : If sequence is not INDEF, this sequence number will be appended to the rootname to form the image name. This parameter will be incremented by one at the end of the task. This IRAF parameter is not to be confused with the camera parameter NUM_IMAGES. When using the camera with IRAF, NUM_IMAGES should be set to 1. This is typically its default value. The *ccdacq* program will handle each image acquisition in the sequence when the camera is used with IRAF.

When focusing the telescope, the *ccdacq* package acquires a series of exposures of the same object in a single CCD frame in order to allow the user to more easily evaluate the variations in focus quality without waiting for the CCD to be read between each acquisition and without comparing multiple files. Three parameters are associated with this feature. One, determines the number of images to be acquired before reading the images off the CCD, the next selects between moving the telescope or reverse shifting the image across the CCD between exposures, and the last determines the number of rows to reverse shift between images if reverse shifting is selected. At this time the AIS1 does not support reverse shifting. This is not a limitation of the camera controller, but has not been coded into the IRAF interface software. At a future date it will be.

nfexpo : This is the number of exposures to take before reading out. In this case, the exposure time is the time for each individual focus exposure. This parameter is used only if *imagetype* is focus.

shtype : Two options exist for acquiring the focus images. The image. If the shift type is "detector", move the image on the detector between focus exposures by reverse shifting the chip. If the shift type is

“telescope”, move the image by moving the telescope. This parameter is used only if imagetype is focus. As noted above, at this time the AIS1 does not support reverse shifting of the image on the CCD and the **shtype** should be set to “telescope”.

nrrows : This is the number of rows to reverse vertical shift between focus exposures. This parameter is used only if imagetype is focus.

Three other parameters are associated with the focus acquisitions. They are described in turn below.

foctype : If the focus type is “telescope”, move the telescope focus between focus exposures. If the focus type is “instrument”, move the instrument focus. This parameter is used only if imagetype is focus.

fstart : This is the starting focus value in a series of focus exposures. It may be any value acceptable to `instrpars.instrfocus` or `telpars.telfocus`. This parameter is used only if imagetype is focus.

fdelta : If non-zero, `fstart` is taken as a numeric value, and `fdelta` is added to it after each focus exposure, to use as the next focus value. If zero, this parameter has no effect. This parameter is used only if imagetype is focus.

pixtype : The data type of the IRAF image to be recorded on disk. Choose among: b, c, u, s, i, l, r, d, x. These correspond to unsigned byte, an IRAF char, unsigned short integer, short integer, integer, long integer, single real, double real, complex, respectively.

observers : The names of the observers to be entered into the image header.

comments : Any comment you wish to be recorded into the image header. One line only.

- comfile** : The name of a file containing comments to be written into the header. Each line of this file is automatically formatted into a FITS COMMENT record. You should not include the COMMENT keyword in this file. Since this file is read at the beginning of the readout, you may edit this file anytime during the exposure.
- obsinfo** : The name of a file to be read and copied into the header.
- command** : Any IRAF task you wish to be executed at the end of the observe task. The first %s in the string is replaced by the image name just recorded. Could be used for post-processing, taping, or just displaying.
- verbose** : Output to STDOUT, messages containing the name of the IRAF image and other information as to what is currently happening. Set to either 'no' or 'yes'.
- debug** : Output to STDERR, debugging information concerning the operation of this task. Set to either 'no' or 'yes'.

11. UNIX Utility Programs

A small set of utility programs are provided for facilitating the use of the camera while its connected to the Sun workstation. These programs may be called from the UNIX command line to communicate with the camera, set parameters, change voltage and timing information, and to operate the camera. No functions are provided for collecting the data from the camera. These programs are meant primarily for use in camera setup and functional testing of the camera electronics. The programs were written using the functions in the 'C' language interface described in the next section of this document.

11.1 *AISsay*

The *AISsay* program is used to issue a single line string of ASCII text to the camera. The text string passed may contain one or more camera commands. The string could define a new camera command. See section 7 of this manual for details on creating new camera commands. Most often, this program will be used to set a camera parameter or adjust a camera voltage during test. This program may be called from within the IRAF environment.

The string is passed on the command line as in :

```
AISsay "OPEN CLOSE READ"
```

which would cause the camera controller to open the shutter, close the shutter, and read the data off the CCD.

11.2 *AISfile*

The *AISfile* program is used to pass an ASCII text file to the camera controller through the serial interface on the VMEbus interface cards. This file may contain any valid camera commands and may include new camera command definitions. This program might commonly be used to set the camera voltages and timing tables to those values stored in a text file. Different text files could be maintained for different CCDs or for different applications. This is a maintenance operation that will not likely be of use to the average operator. If desired, the entire camera software package can be replaced or modified through the use of this utility.

11.3 *AISsetup*

The *AISsetup* program is a simple menu driven program used to set camera parameters and to download files containing voltage and timing information. The program prompts the user for the desired activity and then either opens a file from transmission using an *AISfile*-like function, or prompts the user for parameter values and issues text strings to the camera to set those parameters in much the same way that the *AISsay* program does.

12. 'C' Language Control Library

A set of 'C' language routines is provided to allow programs to be developed which control and collect data from the camera. These routines directly support the most commonly used functions in the camera controller's command set. Other camera functions are indirectly supported via a set of commands used for sending ASCII text strings to the camera through the VMEbus interface.

The 'C' language camera interface consists of a set of routines which may be used in a custom camera control program to operate the camera. The functions may be broken down into four groups. The first of these consists of routines that perform some action on the VMEbus interface card itself, without commanding the camera to do anything. The second set consists of high level functions that cause the VMEbus interface card to issue a command string to the camera controller CPU and wait for a response. The third consists of a set of functions used by those who would like to construct their own command strings and send them to the camera controller. The fourth set consists of low level functions that are called by the functions in each of the three other sets. These deal primarily with the handshaking between the Sun workstation and the CPU on the interface card.

12.1 Function Reference

The following sub-sections of this document discuss the various functions contained in the 'C' language function set. Each sub-section describes one of the sets of functions mentioned above. They are discussed in the following order: Interface Commands, High Level Commands, Communications Functions, and Low Level Commands.

One VMEbus interface card is used for each port read off the CCD. The STIS AIS1 camera uses two CCD ports and therefore, two VMEbus interface cards. The functions in the Interface Command set will likely be called at some point on both interface cards. They accept as their first parameter the base address of the interface card to be affected cast as a character pointer.

12.1.1 Interface Functions

The interface commands are commands which operate not on the camera, but on the VMEbus interface card itself. They include

functions to reset an interface card, to reset an image capture address counter, to initialize the serial to parallel converter, to and to read and write the bits of the "user I/O" port on the interface card, which is as yet unused in this camera system.

The first two functions are used to reset the Interface card, or to halt the microcontroller on the interface card. The first of these might be called at the start of a program, the second is used on for debugging purposes normally.

```
reset (controller)
    reset one of the VME interface boards
    char *controller : address of board to reset
```

```
halt (controller)
    halt the microcontroller on the VMEbus interface
    char *controller : address of board to halt
```

Where the camera data is captured in the cards memory is determined by a counter on the card which may be reset to zero using the function **ag_init** described below. Typically, this function will be called before each image acquisition. If it is not called, then the next image will be collected into the succeeding memory locations. This feature may be used to allow the collection of a series of images into the VMEbus memory before moving the data off the card.

```
ag_init (controller)
    reset the "address generator" PLD on one of the VMEbus interface cards
    char *controller : address of board on which to reset the address
```

The VMEbus interface cards receive the camera data in a serial format and convert that to a parallel form on the card. A programmable logic device controls the conversion.

```
spcon_init (controller)
    initialize the serial to parallel converter on one of the VMEbus interface
    cards.
    char *controller : address of board on which to operate
```

```
spcon_reset (controller)
    reset the serial to parallel converter on one of the VMEbus interface cards
    char *controller : address of board on which to operate
```

The interface cards contain an eight bit "user I/O" port which may be read or written using the following routines. This port could be used as a trigger to external equipment, to control mechanical devices such as filter wheels, etc. As of 12/92, they are not used for any purpose in the STIS AIS1 camera system.

`io_read (controller)`
read the TTL data present on the user I/O port of one of the interface card
char controller : address of board on which to read the I/O port
returns : char the data read on the input pins

`io_write (controller,value)`
write to the output bits of the user I/O port on one of the interface cards
char *controller : address of board on which to write the user I/O port
char value : the value to write

12.1.2 High Level Functions

The high level commands are used to initiate action by the camera controller electronics. They are generally commands that cause the VMEbus interface card to issue an ASCII text string to the camera and wait for a reply. The camera's response is captured as an ASCII text string in the "text input buffer" on the interface card. The user's programs may read this buffer if it is desired to monitor the camera's responses. Many of the commands will also cause the camera to produce image data. This data will be captured by the interface card into the image buffer at the place pointed to by the address counter. Most of the commands here call the `ag_init()` function to reset the address pointer before issuing the camera commands.

`cam_restart (controller)`
Issues a series of command strings to the camera controller which cause it to perform a restart.
char *controller : address of board to use for communication

`set_param (controller,offset,value)`
Set a camera parameter.
char *controller : address of board to use for communication
unsigned short offset : the parameter's parameter table offset
unsigned short value : the new value for the parameter

`get_format (controller)`
get a list of the current settings of the system parameters.
char controller : address of board to use for communication

`set_format (controller)`
initialize the sequencer with the current format params.
char controller : address of board to use for communication

set_exposure_time(controller, exptime)
char *controller: address of board to use for communication

cam_write (controller, address, datum)
write a value to a sequencer address.
char *controller : address of board to use for communication
unsigned short address : the address to write to
unsigned short datum : the datum to write

The camera may perform a continuous clear operation between exposures to eliminate dark current. The camera may perform clock recombination anti blooming during integrations to prevent image degradation due to overexposed areas. The following functions are used to select or deselect these features.

cisc_on (controller)
Turn on continuous clearing of the CCD between image acquisitions.
char *controller : address of the board to use for communication

cisc_off (controller)
Turn off continuous clearing of the CCD between image acquisitions.
char *controller : address of board to use for communication

anti_bloom_on (controller)
enable clock recombination anti-blooming
char *controller : address of board to use for communication

anti_bloom_off (controller)
disable clock recombination anti-blooming
char *controller : address of board to use for communication

The AIS1 camera uses a dual slope integrator based analog to digital conversion scheme on the CCD data. The length of time that the integrator integrates each pixel determines the pixel throughput rate as well as the gain of the electronics. This is why these three gain control functions also affect the camera readout speed. The exact sensitivity of the system will depend on a variety of factors including the CCD output FET operating point voltages. This is why these functions make no reference to an actual gain figure. The cameras gain factor should be measured for an accurate figure.

AIS_gain_lo (controller)
Sets the conversion rate to 40 kpix/sec.
char *controller : address of board to use for communication

AIS_gain_mid (controller)

Sets the conversion rate to 20 kpix/sec.
char *controller : address of board to use for communication

AIS_gain_hi (controller)
Sets the conversion rate to 10 kpix/sec.
char *controller : address of board to use for communication

The CCD temperature is regulated. A program may set and monitor the CCD temperature via the following two functions.

CCD_temp (controller)
query the temperature of the CCD.
char *controller : address of board to use for communication

set_temp (controller,value)
set the desired operating temperature for the CCD.
char *controller : address of board to use for communication
char value : desired temperature

Two functions are provided to control the camera's shutter. Normally, one of the higher level image acquisition commands such as obs(), described below, will be called and the camera controller will take care of the shutter control during the acquisition. These functions are used by the programmer who wishes to control the camera operation at a slightly lower level.

oshut (controller)
Open the shutter on the camera.
char *controller : address of board to use for communication

cshut (controller)
Close the shutter on the camera.
char *controller: address of board to use for communication

A set of low level charge shifting and pixel conversion commands are provided. These include functions to read or discard groups of CCD pixel and columns. Using these routine, the programmer may build custom readout sequences.

pix_bin (controller,npix)
Bin a number of pixels into the CCD summing well.
char *controller: address of board to use for communication
unsigned short npix : the number of pixels to bin

row_bin (controller,nrows)
bin a number of rows into the CCD serial register. One
char *controller: address of board to use for communication
unsigned short nrows : the number of rows to bin

`pix_discard (controller,npix)`
discard a number of pixels in the CCD serial register.
`char *controller`: address of board to use for communication
`unsigned short npix` : the number of pixels to discard

`row_discard (controller,nrows)`
discard a number of rows in the CCD parallel register.
`char *controller`: address of board to use for communication
`unsigned short nrows` : the number of rows to discard

`pix_read (controller,npix)`
read a number of pixels in the CCD serial register.
`char *controller`: address of board to use for communication
`unsigned short npix` : the number of pixels to read

`row_read (controller,nrows)`
read a number of rows in the CCD parallel register.
`char *controller`: address of board to use for communication
`unsigned short nrows` : the number of rows to read

Four functions are provided for clearing the CCD, integrating charge, and reading the CCD image data. They may be used by the programmer to control the CCD image acquisition at a moderately high level. Most often the commands in the following section will be used to allow the camera controller to handle more of the details of image acquisition.

`clear (controller)`
clear all charge off the CCD.
`char controller` : address of board to use for communication

`integrate_dark (controller)`
integrate dark current on the CCD device.
`char controller` : address of board to use for communication

`integrate_light (controller)`
integrate light on the CCD device.
`char *controller` : address of board to use for communication

`readout (controller)`
read the image off the CCD based on the current
`char controller` : address of board to use for communication

Four functions are provided for acquiring images from the camera in a totally self contained fashion. Using these commands allows the programmer the easiest method of image acquisition. The camera controller takes care of the CCD clearing and exposure as well as reading the data off the CCD using the current format parameters.

bias (controller)
generate in image of the CCD with no charge on it.
char *controller : address of board to use for communication

expose (controller,exptime)
this function is redundant to the OBS command below,
char *controller : address of board to use for communication
unsigned long exptime : exposure time, in milliseconds

dark (controller,exptime)
this function is called to generate an image of the CCD
with dark current.
char *controller : address of board to use for communication
unsigned long exptime : exposure time, in milliseconds

obs (controller,exptime)
this command generate an object exposure
char *controller : address of board to use for communication
unsigned long exptime : exposure time, in milliseconds

12.1.3 Communications Functions

These functions are used to send and receive text from the camera. They are use when the 'built-in' commands do not include a command that the programmer would like executed. Normally send_string() is used for this purpose. See the 'C' source code for more information.

send_file (controller,filename)
send an ASCII text file to the camera.
char *controller : address of board to use for communication
char *filename : name of the file to send

send_string (controller, theString)
send an arbitrary string to the camera.
char *controller : address of board to use for communication
char *theString : the string to send

wait_OK (controller)
wait until an 'OK' response has been received from the camera
char *controller: address of board to use for communication

wait_CR_LF (controller)
wait until an carriage return line feed pair has been recieved from the
camera
char *controller: address of board to use for communication

flush_buffer (controller)
char *controller : address of board on which to flush the text input buffer

read_cam_buffer (controller)

char *controller : address of board read the text from
fetch all the characters in the serial input bufer.

send_cam_char (controller,achar)

char *controller : the address of the board to send char through
char achar : the character to send

get_a_char (controller)

char *controller: address of board to fetch the character from
fetch a character from the serial input buffer.

13. Programmable CCD Clock Timing

The AIS1 camera generates the CCD clock timing, that is all sequences of parallel and serial clock edges, and analog processor control signals, with a set of state tables. In doing so, the nature of these sequences may be controlled by placing appropriate information into the tables. Software tools are provided at various levels to allow the user to do so.

13.1 Timing Tables

The camera timing is determined by the variables stored in the following state tables. two tables are provided for each sequence. these include the actual states; i.e. the values to be written into the latches on the clock cards; and a set of "waits" that determine how long the sequencer should pause between states. four sequences are stored :

- Parallel Clock Sequence
- Serial Clock Sequence
- Analog Processing Control Sequence
- Clock Recombination Anti-Blooming Sequence

13.1.1 Parallel Clock Timing

The sequence of the CCD parallel clocks is determined by the entries in the parallel clocking state tables. There are two, one for one half of the CCD and the other for the other half. The two halves of the CCD are clocked toward their respective output ports normally, but may be reversed for full frame readout through either port by changing the entries in the tables or by setting the parallel clocking direction parameters appropriately.

The parallel clock timing tables are 32 entries long. Normally, a CCD clocking sequence can be achieved in 6 or 8 entries. A variable is available for setting the number of states that are actually in use. This variable N_PAR_STATES is referred to by the sequencer, and that number of states are read from the table and written to the latch on the clock card that controls the parallel clocks.

The tables contain 9 bit values. Two bits each are used to control the four parallel clock phases provided. And one bit controls the transfer gate. Two bits are required to control the parallel clocks because tri-level clocking capability is provided.

The FORTH code refers to the parallel clocking tables as CAM0_PAR_STATE and CAM1_PAR_STATE. The entries are accessed by preceding the table name with the desired index. For example,

12 CAM0_PAR_STATE @

will fetch the value of the 12th element in the table named CAM0_PAR_STATE.

Appropriate table entries maybe generated by 'OR-ing' together the following FORTH constants.

```
00 CONSTANT P1_LO
01 CONSTANT P1_MID
02 CONSTANT P1_HI
00 CONSTANT P2_LO
04 CONSTANT P2_MID
08 CONSTANT P2_HI
00 CONSTANT P3_LO
10 CONSTANT P3_MID
20 CONSTANT P3_HI
00 CONSTANT P4_LO
40 CONSTANT P4_MID
80 CONSTANT P4_HI
0 CONSTANT TG_LO
100 CONSTANT TG_HI
```

The FORTH word ' | ' is used to perform a bitwise 'OR' operation on two numbers.

For example, the FORTH command string,

```
P1_LO P2_HI | P3_LO | P4_LO | TG_LO | 3 CAM0_PAR_STATE !
```

will install into position 3 in the paralel clock table associated with CAM0 a state with only parallel clock phase 2 held high. The actual voltage produced for each clock in each of its states is determined by the settings of the CCD clock rail DACs described in greater detail in section 7 of this document.

When moving charge in the parallel direction on the CCD, the camera controller steps through the tables reading the value found at each location and writing that value to the parallel clock control latch on the clock card. If the user value of the CAM0_PDIR or CAM1_PDIR parameter is 0, then the sequemcer starts reading the table at location zero and steps forwards. If one or both of them have a value of 1, then that sequence is read backwards, with the first location being determined by referring to the appropriate table length parameter.

The camera controller pauses between each step in the table. The amount of time that it will pause is determined by the values

stored in another table. The table is referred to as PAR_DELAY and its elements may be accessed in the same way as the state tables. The table contains numbers which represent the number of 100ns 'NOP' instructions the sequencer should execute before writing the next state to the latch. A different delay may therefore be associated with each state in the table. Normally it is not necessary to adjust the delays individually, and a single value may be desired for all the delays. The function SET-PAR_DELAYS may be used to set all the entries in the table to the same value.

To display the contents of the parallel state tables, one may execute the FORTH function SHOW_PAR_STATES, which will produce a formatted ASCII text stream listing the states in the table.

13.1.2 Serial Clock Timing

The sequence of the CCD serial clocks is determined by the entries in the serial clocking state table. The two halves of the CCD are clocked simultaneously using the values in the single table. This is done to assure that the serial clocks run with a minimum of overhead time wasted. No delays are executed by the sequencer when writing the serial clock states to the serial control latches on the clock cards.

The serial clock timing tables are 32 entries long. Normally, a CCD clocking sequence can be achieved in 6 or 8 entries. A variable is available for setting the number of states that are actually in use. This variable N_SER_STATES is referred to by the sequencer, and only that many states are read from the table and written to the latch.

The tables contain 8 bit values. Two bits each are used to control the four serial clock phases provided. Two bits are required to control the parallel clocks because tri-level clocking capability is provided.

The FORTH code refers to the serial clocking table as SER_STATE. The entries are accessed by preceding the table name with the desired index as was shown for the parallel clocking tables above.

As for the parallel clocks, appropriate table entries may be generated by 'OR-ing' together the following predefined FORTH constants.

```
00 CONSTANT S1_LO
01 CONSTANT S1_MID
02 CONSTANT S1_HI
00 CONSTANT S2_LO
04 CONSTANT S2_MID
```

```
08 CONSTANT S2_HI
00 CONSTANT S3_LO
10 CONSTANT S3_MID
20 CONSTANT S3_HI
00 CONSTANT S4_LO
40 CONSTANT S4_MID
80 CONSTANT S4_HI
```

No provision is made for running the serial clocks backwards except by changing the sequence of states in the table.

13.1.3 Analog Processor Control

The analog processor state table is a little different than the parallel or serial state tables. The analog state table determines the sequence in which the various parts of the analog signal processing chain are switched. The analog processor is constructed in the form of a dual slope integrator. There are a very limited number of ways in which this state table may be effectively configured. The primary reason for providing the programmable timing approach for the analog processor is so that entirely different analog processing schemes could be supported by modifying the hardware in the system, while not requiring a change in the software. As long as the analog processing chain is configured as it was delivered, the states in the table will retain essentially the same sequence.

The table is supported in the same way as the parallel and serial timing tables. The table is referred to as ANA_STATE, and the individual elements are referred to by index in the same way.

Of more interest to the programmer is the table of delays between analog processing states. The delays during which the integration takes place are the most important. These two should be the same and will determine the gain of the integrator and therefore of the system. The time during which the integrator is reset should be long enough to assure that the integrating capacitor is fully reset. A minimum length of 6 usec is recommended. For precise adjustment of the analog processor timing it is recommended that the waveforms be examined on an oscilloscope, and the number in the table adjusted until the appropriate timing is achieved.

The analog processor timing is not something that the user will ordinarily need to modify, and there are convenient functions available in the FORTH program to set the integrator values to achieve particular camera operating speeds.

The length of the analog processor control table is fixed at 14. There is a fixed pattern with which they are executed and anyone modifying the table entries must be aware of this. The first two

states are written before the CCD reset pulse goes high and then low again. Then the next two are written. After that state pair, the serial clocks run, shifting the pixels to be read into the summing well. The next five states are then written. Then the summing well signal is pulsed low and then high, pushing the pixel charge into the output of the CCD. Then five more states are output. The sequence then starts over for anymore pixels to be converted.

The FORTH function SHOW-ANA_STATES can be called to examine the contents of the analog control state table.

Entries for the table can be generated by OR-ing together the following FORTH constants.

```

1 CONSTANT RIN      ( integrator reset
0 CONSTANT !RIN
2 CONSTANT DCR      ( D.C. restore
0 CONSTANT !DCR
4 CONSTANT SA2      ( sample period 2
0 CONSTANT !SA2
8 CONSTANT SA1      ( sample period 1
0 CONSTANT !SA1
10 CONSTANT CTC     ( command to convert
0 CONSTANT !CTC
20 CONSTANT SEN     ( send data trigger
0 CONSTANT !SEN
40 CONSTANT 16B     ( use slow speed 16 bit converter )
0 CONSTANT !16B

```

13.1.4 ClockRecombination Anti-Blooming

The AIS1 camera is capable of performing clock recombination anti blooming during image integrations. This anti-blooming is accomplished by switching the parallel clocks while integrating charge. The sequence in which they are clocked to accomplish this is stored in a table named AB_STATE and is indexed in the same way as the other tables discussed in this document. There is also an AB_DELAY table associated with this clocking. It operates the same as the delay table associated with the parallel clocks.

The FORTH function SHOW-AB_STATES may be used to generate a list of the states currently stored in the table.

The same FORTH constants as are OR-ed to create the parallel state table may be used to create the AB_STATE table.

13.2 Filling the Timing Tables

The timing tables do not often need to be modified. They will need modification when a different type of CCD is installed or if the analog processor is modified greatly. If the user wishes to

experiment with CCD clocking schemes to optimize the operation of the CCD then they will need to know how to fill the tables.

A simple terminal may be connected to the camera controller and used to update the tables stored in the controllers EEPROM.

The easiest way to fill a particular entry in a timing table is to use the *AISsay* utility. This utility described elsewhere in this document may be used to transmit a one line text command to the camera controller through the VMEbus interface card. For example, typing

```
AISsay "P1_LO P2_HI | P3_LO | P4_LO | TG_HI | 6 CAM1_PAR_STATE !"
```

at the UNIX command line, will install into position 6 in the parallel clock table associated with CAM1 a state with parallel clock phase 2 and the transfer gate held high.

Typically, the contents of an entire table will need to be modified at once. This is most conveniently done using the *AISfile* utility, also described elsewhere in this document. In this way a simple ASCII text file may be maintained and sent to the camera at once.

The *AISsetup* utility also offers the option of sending a text file to the camera for this purpose. It uses a default filename of "AISTiming", and is primarily intended to allow the user to put the states back in their original configuration easily. If the "AISTiming" file has been modified, then it could contain anything.

13.3 Timing Table Example

DECIMAL

10 N_PAR_STATES !

```

P1_MID P2_LO OR P3_LO OR P4_LO OR TG_LO OR 0 CAM0_PAR_STATE !
P1_MID P2_LO OR P3_HI OR P4_LO OR TG_HI OR 1 CAM0_PAR_STATE !
P1_LO P2_LO OR P3_HI OR P4_LO OR TG_HI OR 2 CAM0_PAR_STATE !
P1_LO P2_MID OR P3_HI OR P4_LO OR TG_HI OR 3 CAM0_PAR_STATE !
P1_LO P2_MID OR P3_LO OR P4_LO OR TG_LO OR 4 CAM0_PAR_STATE !
P1_MID P2_MID OR P3_LO OR P4_LO OR TG_LO OR 5 CAM0_PAR_STATE !
P1_MID P2_LO OR P3_LO OR P4_LO OR TG_LO OR 6 CAM0_PAR_STATE !
P1_MID P2_LO OR P3_LO OR P4_LO OR TG_LO OR 7 CAM0_PAR_STATE !
P1_MID P2_LO OR P3_LO OR P4_LO OR TG_LO OR 8 CAM0_PAR_STATE !
P1_MID P2_LO OR P3_LO OR P4_LO OR TG_LO OR 9 CAM0_PAR_STATE !
    
```

```

P1_MID P2_LO OR P3_LO OR P4_LO OR TG_LO OR 0 CAM1_PAR_STATE !
P1_MID P2_MID OR P3_LO OR P4_LO OR TG_LO OR 1 CAM1_PAR_STATE !
P1_LO P2_MID OR P3_LO OR P4_LO OR TG_LO OR 2 CAM1_PAR_STATE !
P1_LO P2_MID OR P3_HI OR P4_LO OR TG_HI OR 3 CAM1_PAR_STATE !
P1_LO P2_LO OR P3_HI OR P4_LO OR TG_HI OR 4 CAM1_PAR_STATE !
P1_MID P2_LO OR P3_HI OR P4_LO OR TG_HI OR 5 CAM1_PAR_STATE !
P1_MID P2_LO OR P3_LO OR P4_LO OR TG_LO OR 6 CAM1_PAR_STATE !
P1_MID P2_LO OR P3_LO OR P4_LO OR TG_LO OR 7 CAM1_PAR_STATE !
P1_MID P2_LO OR P3_LO OR P4_LO OR TG_LO OR 8 CAM1_PAR_STATE !
P1_MID P2_LO OR P3_LO OR P4_LO OR TG_LO OR 9 CAM1_PAR_STATE !
    
```

```

800 0 PAR_DELAY !
800 1 PAR_DELAY !
800 2 PAR_DELAY !
800 3 PAR_DELAY !
800 4 PAR_DELAY !
800 5 PAR_DELAY !
800 6 PAR_DELAY !
800 7 PAR_DELAY !
800 8 PAR_DELAY !
800 9 PAR_DELAY !
    
```

10 N_SER_STATES !

```

S1_HI S2_HI OR S3_LO OR S4_LO OR 0 SER_STATE !
S1_HI S2_LO OR S3_LO OR S4_LO OR 1 SER_STATE !
S1_HI S2_LO OR S3_HI OR S4_LO OR 2 SER_STATE !
S1_LO S2_LO OR S3_HI OR S4_LO OR 3 SER_STATE !
S1_LO S2_HI OR S3_HI OR S4_LO OR 4 SER_STATE !
S1_LO S2_HI OR S3_LO OR S4_LO OR 5 SER_STATE !
S1_LO S2_HI OR S3_LO OR S4_LO OR 6 SER_STATE !
S1_LO S2_HI OR S3_LO OR S4_LO OR 7 SER_STATE !
S1_LO S2_HI OR S3_LO OR S4_LO OR 8 SER_STATE !
S1_LO S2_HI OR S3_LO OR S4_LO OR 9 SER_STATE !
    
```

14 N_ANA_STATES !

```

RIN DCR OR !SA1 OR !SA2 OR !CTC OR SEN OR 16B OR 0 ANA_STATE !
RIN DCR OR !SA1 OR !SA2 OR !CTC OR !SEN OR 16B OR 1 ANA_STATE !
RIN DCR OR !SA1 OR !SA2 OR !CTC OR SEN OR 16B OR 2 ANA_STATE !
RIN DCR OR !SA1 OR !SA2 OR !CTC OR SEN OR 16B OR 3 ANA_STATE !
RIN DCR OR !SA1 OR !SA2 OR !CTC OR SEN OR 16B OR 4 ANA_STATE !
RIN !DCR OR !SA1 OR !SA2 OR !CTC OR SEN OR 16B OR 5 ANA_STATE !
!RIN !DCR OR !SA1 OR !SA2 OR !CTC OR SEN OR 16B OR 6 ANA_STATE !
!RIN !DCR OR SA1 OR !SA2 OR !CTC OR SEN OR 16B OR 7 ANA_STATE !
!RIN !DCR OR !SA1 OR !SA2 OR !CTC OR SEN OR 16B OR 8 ANA_STATE !
!RIN !DCR OR !SA1 OR SA2 OR !CTC OR SEN OR 16B OR 9 ANA_STATE !
!RIN !DCR OR !SA1 OR !SA2 OR !CTC OR SEN OR 16B OR 10 ANA_STATE !
!RIN !DCR OR !SA1 OR !SA2 OR !CTC OR SEN OR 16B OR 11 ANA_STATE !
!RIN !DCR OR !SA1 OR !SA2 OR CTC OR SEN OR 16B OR 12 ANA_STATE !
!RIN !DCR OR !SA1 OR !SA2 OR !CTC OR SEN OR 16B OR 13 ANA_STATE !
    
```

```

1 0 ANA_DELAY !
1 1 ANA_DELAY !
1 2 ANA_DELAY !
1 3 ANA_DELAY !
    
```

```

1   4 ANA_DELAY !
12  5 ANA_DELAY !
1   6 ANA_DELAY !
( ana_DELAY 7 and 9 are the dual slope integration times )

```

```

157 7 ANA_DELAY !
1   8 ANA_DELAY !
157 9 ANA_DELAY !
1  10 ANA_DELAY !
1  11 ANA_DELAY !
1  12 ANA_DELAY !
50 13 ANA_DELAY !

```

```

10 N_ANTI-BLOOM_STATES !

```

```

P1_LO P2_LO OR P3_LO OR P4_LO OR TG_LO OR 0 ANTI-BLOOM_STATE !
P1_LO P2_LO OR P3_LO OR P4_LO OR TG_LO OR 1 ANTI-BLOOM_STATE !
P1_LO P2_LO OR P3_LO OR P4_LO OR TG_LO OR 2 ANTI-BLOOM_STATE !
P1_LO P2_LO OR P3_LO OR P4_LO OR TG_LO OR 3 ANTI-BLOOM_STATE !
P1_LO P2_LO OR P3_LO OR P4_LO OR TG_LO OR 4 ANTI-BLOOM_STATE !
P1_LO P2_LO OR P3_LO OR P4_LO OR TG_LO OR 5 ANTI-BLOOM_STATE !
P1_LO P2_LO OR P3_LO OR P4_LO OR TG_LO OR 6 ANTI-BLOOM_STATE !
P1_LO P2_LO OR P3_LO OR P4_LO OR TG_LO OR 7 ANTI-BLOOM_STATE !
P1_LO P2_LO OR P3_LO OR P4_LO OR TG_LO OR 8 ANTI-BLOOM_STATE !
P1_LO P2_LO OR P3_LO OR P4_LO OR TG_LO OR 9 ANTI-BLOOM_STATE !

```

```

200 0 ANTI-BLOOM_WAIT !
200 1 ANTI-BLOOM_WAIT !
200 2 ANTI-BLOOM_WAIT !
200 3 ANTI-BLOOM_WAIT !
200 4 ANTI-BLOOM_WAIT !
200 5 ANTI-BLOOM_WAIT !
200 6 ANTI-BLOOM_WAIT !
200 7 ANTI-BLOOM_WAIT !
200 8 ANTI-BLOOM_WAIT !
200 9 ANTI-BLOOM_WAIT !

```

14. Programmable CCD Voltages

The AIS1 camera allows the user or a control program to manipulate the CCD clock voltages through software. All the CCD voltages may be adjusted over a wide range. This allows the voltages to be quickly and predicably varied to either test the operational parameters of a specific CCD device, or to optimize the camera's setup for a particular CCD with known parameters. Since the STIS AIS1 camera is a dual port system, separate voltage adjustments for either portion of the CCD are available where appropriate. Exceptions are noted below. In the development of the system, certain decisions were made regarding which voltages should be programmed and which need not be. These judgements are also discussed below.

14.1 CCD Clock Rails and DC potentials

The CCD clocks are adjusted by a set of eight bit voltage output digital to analog converters (DACs) located on the clock card. The DAC outputs range from 0.0 volts to 5.0 volts, approximately. These voltages are scaled to the range appropriate for the particular clock rail in question. In order to maintain flexibility in the system, most of the voltages may be adjusted over the range of -12.5 volts to +12.5 volts..

14.1.1 CCD Parallel Clocks

The CCD parallel clocks are adjusted by the setting of three DACs. Three voltages are provided in order to support either virtual phase devices (which are becoming increasingly rare) or to optimize the pixel capacity of devices operated in the MPP mode. The three available voltages and their ranges are shown below.

parallel clock low	-12.5/+12.5
parallel clock midrange	-12.5/+12.5
parallel clock high	-12.5/+12.5

In the STIS camera, the Tektronix M745A CCD is typically operated in the MPP mode and clock phases 1 and 2 are switched between low and midrange while phase 3, under which the MPP implant resides, is switched from low to high. In this way the upper rail of phase 3 may be independently controlled to optimize the CCD's "full well capacity". The actual clocking of the various phases depends on the CCD timing information, which is also programmable and discussed elsewhere. Separate clock voltages for the parallel to serial transfer gate are provided and their ranges are shown below.

transfer gate low	-12.5/+12.5
transfer gate high	-12.5/+12.5

Tri-level clocking is not implemented on the transfer gate. It is not deemed necessary.

14.1.2 CCD Serial Clocks and Suming Well

The CCD serial clocks also support tri-level clocking, but here it is primarily for use with virtual phase devices, since MPP mode is not usually an option on the serial register. The available voltages and their ranges are shown below.

serial clock low	-12.5/+12.5
serial clock midrange	-12.5/+12.5
serial clock high	-12.5/+12.5

Separate voltages are available for the summing well, although in most cases it is clocked at the same potentials as the serial register. Tri-level clocking is not supported on this gate. These voltages and their ranges are shown below.

summing well low	-12.5/+12.5
summing well high	-12.5/+12.5

14.1.3 CCD Reset Gate and Last Gate

The CCD reset gate is supported with a pair of voltages for high and low. The switch which actually clocks the CCD reset gate is located in the camera head for maximum immunity to noise and potentially damaging voltage spikes. The CCD reset gate is a very sensitive point on the device and very susceptible to noise pickup. A programmable voltage for the lower rail of the reset gate is provided on the clock card and fed into the camera head, but it was found that for optimum noise performance, it was best to tie this potential to a very clean ground. Therefore, adjusting the reset gate low voltage has no affect whatsoever on the system.

reset gate low	0.0/+15.0
reset gate high	0.0/+15.0

The CCD serial register is terminated by a single gate that acts as a barrier between the summing well and the output. This gate is known by a variety of names such as "serial transfer gate", "output gate", and "last gate". In the hardware and software of the AIS1 camera, this gate is

named the "last gate" and is adjustable over a reduced range. Whereas the serial and parallel clocks may be adjusted from -12.5V to +12.5 volts, the last gate may be adjusted from -5.0V to +5.0V. This design decision was made for two reasons. The first is that it was not thought that the last gate would require potentials outside of this range, and the second was to allow a finer adjustment of the potential with only 0.04V change per DAC unit. The last gate potential range is shown below.

last gate	-5.0/+5.0
-----------	-----------

14.1.4 CCD Output FET Potentials

Two DC potentials that can have a very dramatic effect on the operation of the CCD output amplifier are the reset FET and output FET drain voltages. They determine the "operating point" of the amplifier and have a very great effect on gain and linearity of the amplifier. These two voltages need to be raised to greater potentials than the other signals in the system and are therefore generated a little differently. The voltage out of the DAC is buffered by an operational amplifier operating off a single sided supply of approx. +28 VDC. The design of the circuit allows the voltages to be raised to +20 and +25 V respectively. This allows the operation of all CCDs in widespread use today including those with "lightly doped drain" output amplifiers. The STIS 2048 CCD is such a device.

Separate reset drain (VRD) and output drain (VOD) voltages are provided for the two amplifiers on the CCD, but experience shows that there should be little or no potential between the various VRD voltages used on a single CCD. For this reason, although both clock cards have the circuitry necessary for producing the VRD signal and the software refers to VRD potentials for both amplifiers, only one voltage is actually applied to both amplifiers. It is the VRD signal produced on the clock card for amplifier 1 which is used. Adjusting this voltage will affect the operation of both amplifiers, while adjusting the other will have no effect on the system. The signal names and their ranges are shown below.

CCD reset drain (VRD)	+5.0/+20.0
CCD output drain (VOD)	+5.0/+25.0

14.1.5 Other Clocks and Potentials

The AIS1 socket card provides a variety of options, one of which is the use of a "clamp" circuit which discharges the AC coupling capacitor used in the very first stage of amplification off the CCD. The use of this circuit is selected by installation of the appropriate components into the

circuit board. The camera was built to include this circuit since it has been proven to be effective. As this option was not originally planned for, there was no signal provided to operate the FET switch which is used to connect the capacitor to ground. Therefore, one of the extra clock switches provided on the clock card was used to toggle the switch which requires a signal that swings from somewhat less than -10.0 volts to 0 volts, with the higher voltage closing the switch. The voltages for this switch do not need to be adjusted and in doing so incorrectly (i.e. >0.6 volts) can result in damage to the FET. For this reason, the rails were hard wired to -15.0 V and ground. Therefore, although extra switch 1 is used for this circuit, the programmable voltages associated with it are not, and adjusting them has no affect on the camera. In some other configuration, these voltages may be used for some other purpose, and are therefore listed below along with their ranges.

(clamp signal)		
extra clock 2 high	-12.5/+12.5	(unused)
extra clock 2 low	-12.5/+12.5	(unused)

Another option provided by the AIS1 socket card is the selection of either a simple load resistor or a current source as the load seen by the CCD output FET. There are some times when a current source can provide a slightly higher gain and therefore lower noise in the CCD readout. The AIS1 camera provides a programmable current by providing a programmable voltage which is applied to the base of an emitter follower circuit. The emitter load, a resistor, converts this voltage to a current through the transistor's collector which is connected to the CCD output FET's source. The CCD load for each side of the camera may be independently selected by jumpers J3 and J6 on the socket card. This selection determines whether the CCD output drives the load resistor or the current source. Another jumper selection, jumpers J2 and J5, determine whether the current sources will have a fixed current or be controlled by the programmable voltage. The current source was not part of the original design of the camera electronics and represents another use of one of the extra clocks that were designed into the system. The current source transistor's base is tied to extra clock 2. The range of the programmable voltage is shown below.

(current source)		
extra clock 2 high	-12.5/+12.5	
extra clock 2 low	-12.5/+12.5	

At the time that the AIS1 camera design was first being implemented, it was decided that all CCD clock rails and DC potentials should be programmable. This philosophy was carried through the later parts of the design and was implemented to the extent deemed advantageous to the best operation of the CCD. One voltage which was provided as a programmed voltage but not used as such in the final system is the CCD substrate. A programmable voltage was provided for this potential, and is connected to the camera head feedthrough connector, but is not used in the circuit. It was learned through experimentally that this potential should be tied to a good ground point in the circuit, and the CCD substrate actually provides the center point for the "star" grounding system in the final AIS1 camera configuration. The voltage and it's range is shown below pimarily for completeness. Adjusting the "substrate" voltage has no affect whatsoever on the system.

substrate -5.0/+5.0

There are two additional clocked signals available which are unused in the STIS AIS1 camera system, but which could be of some use with a different CCD or some other circuitry. They are provided for future use and changing their values will have no affect on the system as it stands. These voltages are listed below.

extra switch 3 low	-12.5/+12.5
extra switch 3 high	-12.5/+12.5
extra switch 4 low	-12.5/+12.5
extra switch 4 high	-12.5/+12.5

Table 14.1 - AIS1 Camera Programmable Voltages

voltage	min	max	comment
parallel clock low	-12.5	+12.5	
parallel clock midrange	-12.5	12.5	
parallel clock high	-12.5	12.5	
transfer gate low	-12.5	12.5	
transfer gate high	-12.5	12.5	
serial clock low	-12.5	12.5	
serial clock midrange	-12.5	12.5	
serial clock high	-12.5	12.5	
summing well low	-12.5	12.5	
summing well hig	-12.5	+12.5	
reset gate low	0	+15.0	(unused)
reset gate high	0	15.0	
last gate	-5.0	5.0	
reset drain	+5.0	20.0	
output drain	+5.0	25.0	
substrate	-5.0	5.0	(unused)
extra switch 1 low	-12.5	12.5	(unused)(clamp)
extra switch 1 high	-12.5	12.5	(unused)(clamp)
extra switch 2 low	-12.5	12.5	(unused)(current source)
extra switch 2 high	-12.5	12.5	(unused)(current source)
extra switch 3 low	-12.5	12.5	(unused)
extra switch 3 high	-12.5	12.5	(unused)
extra switch 4 low	-12.5	12.5	(unused)
extra switch 4 high	-12.5	12.5	(unused)

14.2 CCD Protection Circuitry

Since the various programable voltages on the clock cards are generated by digital to analog converters whose power-up status is indeterminate, it was decided that an effort should be made to assure that the voltages when randomly assigned, would not in any way damage the CCD device. Two different approaches were taken in the design of the system to make this assurance. The first is that of limiting the range of the various voltages so that there could not be excessive potential across any of the gates. The CCD substrate was therefore limited to $-5/+5$ volts so that there could be no more than 17.5 volts from any parallel or serial clock rail and the CCD substrate. With the substrate grounded, as in the final camera configuration, there can be no more than 12.5 volts between any one of these clocks and the substrate and no more than 25 volts between any two clock gates. These potentials are considered safe.

Additionally, circuitry was provided for essentially disconnecting the clocking signals from the CCD. The clock signals and DC potentials are passed through FET analog switches before the signal buffers. By opening these switches, the buffers are disconnected from the clock potentials and their inputs see only a resistive load to ground. In this way all the CCD clock potentials are tied to a near ground potential. These switches are open by default at power up and must be closed by the DSP software. They may be opened at any time through the software and, if desired, the CCD may be safely removed from its socket without powering down the system.

Catastrophic failure of the clock card circuitry or unpredictable behavior by the DSP program should not harm the CCD due to the limits placed on the programable voltage ranges discussed above and the limited range of the power supply voltages in the system.

14.3 Software Interface for Programmable Voltages

The programmable CCD voltages in the AIS1 camera system are supported in different ways at all levels in the camera software. The DACs that are used to produce the voltages are mapped into the DSPs address space, the FORTH interpreter handles the different sets of voltages for the various clock cards in the system, and the user may adjust the voltages either from a terminal or through one of the utility programs available on the sun workstation. No direct manipulation of the voltages is provided in the IRAF interface software, but the user may manipulate them from within the IRAF shell by calling one of the utility programs as an external command.

Each level of software associated with the programmable clock voltages will be discussed in turn below.

14.3.1 DSP Control of Programmable Voltages

The clock cards in the AIS1 camera system are mapped into the DSP sequencer's address space as a set of registers. Twenty four of these registers are the eight bit DACs that set the various potentials. The DSP sets the clock voltages simply by enabling the particular clock card in question and writing to those locations. The clock cards are mapped into the upper portion of the DSPs "Y" memory space. The DACs occupy the following locations:

Table 14.2 - DSP Voltage Mnemonics and Addresses

clock name	DSP mnemonic	DSP address (hex)
Parallel low	par_lo	ff08
parallel midrange	par_mid	ff09
parallel high	par_hi	ff0a
serial low	ser_lo	ff0b
serial midrange	ser_mid	ff0c
serial high	ser_hi	ff0d
transfer gate low	tg_lo	ff0e
transfer gate high	tg_hi	ff0f
summing well low	sw_lo	ff10
summing well high	sw_hi	ff11
CCD substrate	sub	ff12
last gate	lg	ff13
reset gate low	rst_lo	ff14
reset gate high	rst_hi	ff15
reset drain	vrd	ff16

output drain	vod	ff17
extra switch 1 low	x1_lo	ff18
extra switch 1 high	x1_hi	ff19
extra switch 2 low	x2_lo	ff1a
extra switch 2 high	x2_hi	ff1b
extra switch 3 low	x3_lo	ff1c
extra switch 3 high	x3_hi	ff1d
extra switch 4 low	x4_lo	ff1e
extra switch 4 high	x4_hi	ff1f

The DSP does not keep track of the voltage settings nor does it have any way of measuring the actual voltage that is produced. It simply sets the DACs to the values requested by the controller, the 68HC11, when it is requested to do so.

14.3.2 FORTH Interface for Programmable Voltages

The controller, the 68HC11, is responsible for keeping track of the desired clock voltage settings and for initializing them when so requested. The 68HC11 only records the value that the user desires to have written to the DAC, it does not have any way of knowing what that voltage will actually be, nor does it have any way of measuring the voltage produced. Since the voltages are controlled by eight bit DACs, the legitimate values for their settings range from 0 to 255. A setting of 0 will produce the voltage at the bottom of the voltage's range and a setting of 255 will produce a voltage at the top of the range. A setting of 128 will produce a voltage at half range, which is approximately 0.0 volts for many of the signals in the system including the parallel and serial clocks, the transfer gate and the summing well. The 25 volt range of these clocks and the 255 different settings provide a convenient 0.10 volt approximate step for each increment of the DAC value.

A separate table of DAC settings is maintained for each clock card in the system. Since the STIS camera is a two channel system, there are two tables of DAC settings in the FORTH dictionary. These tables are "remembered" along with the rest of the dictionary from session to session if desired and a set of standard DAC settings may be maintained this way in the camera controller. New settings may be made at any time by changing the table entry and issuing the command to initialize the clock card with the current settings. The tables and the associated commands are discussed below.

The voltage tables for each clock card in the system are stored as separate tables, allowing the software to be easily expanded to include support for more CCD readout channels. The tables are essentially indexed arrays, and any voltage in the table may be addressed by index and table

name, but it is often easier to associate a specific name with the voltage in question. Both methods of addressing the table elements are supported.

The table of DAC settings for channel 0 is referred to as CAM0_VOLT, and the elements of the table may be read and written by index as in:

```
(value) (index) CAM0_VOLT C!
```

to set a new value and

```
(index) CAM0_VOLT C@
```

to retrieve the value of an element. Note that since the values are eight bits in size, the C@ and C! commands are used to store and retrieve the values. The table of DAC settings for channel 1 is referred to as CAM1_VOLT and is addressed in a similar way. The order of the voltages in the tables is the same as the order in which they occupy the DSP memory space.

Since it may be difficult to remember the correct sequence of the values, individual FORTH definitions have been defined which index into the array the correct amount to access the various values, and mnemonic names have been assigned to them to make the tables more easily accessible to humans. The FORTH definitions simply index the array. An example of one of these definitions is shown below.

```
: CAM0_PAR_LO 0 CAM0_VOLT ;
```

This definition exists solely to allow the user or a control program to access the voltage setting by name instead of by index. Setting the value of the parallel low voltage for channel 0 can be accomplished as shown below.

```
(value) CAM0_PAR_LO C!
```

Such definitions exist for all the voltages in the system and their names are listed below.

Table 14.3 - AIS1 Channel 0 FORTH Voltage Definitions

voltage	FORTH name	comment
parallel low	CAM0_PAR_LO	
parallel midrange	CAM0_PAR_MID	
parallel high	CAM0_PAR_HI	
serial low	CAM0_SER_LO	
serial midrange	CAM0_SER_MID	
serial high	CAM0_SER_HI	
transfer gate low	CAM0_TG_LO	
transfer gate high	CAM0_TG_HI	
summing well low	CAM0_SW_LO	
summing well high	CAM0_SW_HI	
CCD substrate	CAM0_SUB	(unused)
last gate	CAM0_LAST	
reset gate low	CAM0_RST_LO	(unused)
reset gate high	CAM0_RST_HI	
output drain	CAM0_VOD	
reset drain	CAM0_VRD	(unused)
extra switch 1 low	CAM0_X1_LO	(current source)
extra switch 1 high	CAM0_X1_HI	(current source)
extra switch 2 low	CAM0_X2_LO	(unused)
extra switch 2 high	CAM0_X2_HI	(unused)
extra switch 3 low	CAM0_X3_LO	(unused)
extra switch 3 high	CAM0_X3_HI	(unused)
extra switch 4 low	CAM0_X4_LO	(unused)
extra switch 4 high	CAM0_X4_HI	(unused)

Table 14.4 - AIS1 Channel 1 FORTH Voltage Definitions

voltage	FORTH name	comment
parallel low	CAM1_PAR_LO	
parallel midrange	CAM1_PAR_MID	
parallel high	CAM1_PAR_HI	
serial low	CAM1_SER_LO	
serial midrange	CAM1_SER_MID	
serial high	CAM1_SER_HI	
transfer gate low	CAM1_TG_LO	
transfer gate high	CAM1_TG_HI	
summing well low	CAM1_SW_LO	
summing well high	CAM1_SW_HI	
CCD substrate	CAM1_SUB	(unused)
last gate	CAM1_LAST	
reset gate low	CAM1_RST_LO	(unused)
reset gate high	CAM1_RST_HI	
output drain	CAM1_VOD	
reset drain	CAM1_VRD	(both channels)
extra switch 1 low	CAM1_X1_LO	(current source)
extra switch 1 high	CAM1_X1_HI	(current source)
extra switch 2 low	CAM1_X2_LO	(unused)
extra switch 2 high	CAM1_X2_HI	(unused)
extra switch 3 low	CAM1_X3_LO	(unused)
extra switch 3 high	CAM1_X3_HI	(unused)
extra switch 4 low	CAM1_X4_LO	(unused)
extra switch 4 high	CAM1_X4_HI	(unused)

14.4 Setting the CCD Potentials

The programmable voltages may be manipulated in a variety of ways. Since the camera contains the FORTH interpreter and has the on board EEPROM for storing the FORTH dictionary, maintenance setup and diagnostics may be performed with a simple RS-422 terminal connected to the serial port on the controller. All camera operations may be performed through this interface, but, of course, no image data may be collected. This method of interacting with the camera may be useful for setting the clock voltages to new values if the desired values are known (i.e. when installing a different but previously tested CCD). The voltages may also be adjusted using one of the utility programs provided on the Sun workstation. The "AISsetup" program may be used to send the default DAC settings file "AISvolts" to the camera through the serial interface on the VMEbus interface card. The "sendfile" program may be used to send any arbitrary text file to the camera through that interface, and the file could contain a list of DAC settings. Either of these programs sends a simple ASCII text file to the camera. This file consists of FORTH commands and may include anything that the 68HC11 can compile or execute. As far as setting camera voltages is concerned, the files should contain lines such as that shown below.

```
250 CAM0_VOD C!  
225 CAM0_VRD C!  
INIT-VOLTS
```

The first two lines set the FORTH dictionary table entries for those two voltages to new values. The last line instructs the 68HC11 to initialize the clock voltages and actually change the DAC settings on the clock card. The text file may adjust whichever clock voltages are desired. If the file does not contain the INIT-VOLTS instruction, or some other instruction which will cause INIT-VOLTS to be executed, then the table entries will be changed but the CCD voltages will not. If it is desired to make these new values the defaults, then the instruction STORE should be placed in the file after the values have been updated. It is not necessary to initialize the DACs in order to store the new values as defaults.

Additionally, the *AISsay* program can be used to manipulate a single clock voltage. This program is used to issue a single line of text to the camera. It is called from the UNIX command line in the following way.

```
AISsay <string>
```

As an example :

AISsay "134 CAM1_LAST C! INIT-VOLTS"

would set the last gate DAC value for channel 1 to 134 and initialize the camera voltages. And the command

AISsay "134 CAM1_LAST C! STORE"

would set the last gate voltage and store as the default.

14.5 Programmable Voltage Example File

DECIMAL
40 CAM0_PAR_LO C!
120 CAM0_PAR_MID C!
149 CAM0_PAR_HI C!
87 CAM0_SER_LO C!
128 CAM0_SER_MID C!
169 CAM0_SER_HI C!
76 CAM0_TG_LO C!
180 CAM0_TG_HI C!
82 CAM0_SW_LO C!
169 CAM0_SW_HI C!
128 CAM0_SUB C!
100 CAM0_LAST C!
0 CAM0_RST_LO C!
203 CAM0_RST_HI C!
220 CAM0_VOD C!
180 CAM0_VRD C!
0 CAM0_X1_LO C!
128 CAM0_X1_HI C!
0 CAM0_X2_LO C!
149 CAM0_X2_HI C!
0 CAM0_X3_LO C!
255 CAM0_X3_HI C!
0 CAM0_X4_LO C!
255 CAM0_X4_HI C!

40 CAM1_PAR_LO C!
190 CAM1_PAR_MID C!
149 CAM1_PAR_HI C!
87 CAM1_SER_LO C!
128 CAM1_SER_MID C!
169 CAM1_SER_HI C!
76 CAM1_TG_LO C!
180 CAM1_TG_HI C!
82 CAM1_SW_LO C!
169 CAM1_SW_HI C!
128 CAM1_SUB C!
102 CAM1_LAST C!
0 CAM1_RST_LO C!
203 CAM1_RST_HI C!
221 CAM1_VOD C!
180 CAM1_VRD C!
0 CAM1_X1_LO C!
128 CAM1_X1_HI C!
0 CAM1_X2_LO C!
149 CAM1_X2_HI C!
0 CAM1_X3_LO C!
255 CAM1_X3_HI C!
0 CAM1_X4_LO C!
255 CAM1_X4_HI C!

DECIMAL

15. Camera Configuration Parameters

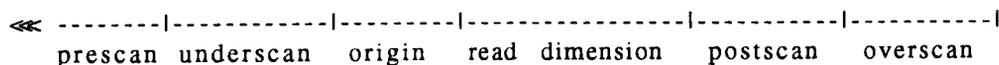
The operation of the AIS1 camera is controlled by a set of variables. These variables are stored in a "parameter table" and may be referenced by index or name. The parameters may be separated into several groups, each relating to a different aspect of camera operation.

15.1 Format Parameters

The first group of parameters are considered 'format' parameters. These parameters control the area on the CCD imager which will be read during the next image acquisition cycle. The format parameters may be set to any 16 bit value. They are considered unsigned numbers and range from 0 to 65535. No type of error checking is performed to assure that the current parameters match the user's CCD or that valid image data will be obtained. Complete flexibility within the scope allowed by the geometry implied by the parameters is pursued instead. It is up to the user to assure that the parameters in use match his or her desires. Upon a system reset the format parameters will be returned to the value stored in the EEPROM memory. The format parameters may be stored there at any time using the STORE command which stores the entire FORTH dictionary.

15.1.1 Serial Read Parameters

The serial register is read out based on a set of format parameters which imply the following geometry:



The readout operations are performed from left to right. The pixels in the prescan, origin, and postscan are discarded. The pixels in the underscan, the read dimension and the overscan are read off the CCD imager. The discarded pixels are in units of physical pixels and are not affected by the binning factor. The others are read off the chip and binning is performed. The same binning factor is used on the pixels in the underscan, read dimension, and overscan.

Typically,

prescan + {underscan * binning factor} = serial extension length
and

$\text{origin} + \{\text{binning factor} * \text{read dimension}\} + \text{postscan} \geq \text{CCD serial dimension,}$

but this convention is not enforced. Not all users will care to use the underscan and overscan features. In this case the prescan, underscan, and overscan parameters may be set to zero and the following equation will be used.

$\text{origin} + \{\text{binning factor} * \text{read dimension}\} + \text{postscan} = \text{total CCD serial length}$

No effort is made to enforce this convention. It is simply that, a convenient convention.

0 CCD_SER CCD Serial Dimension

This parameter represents the total length of the CCD serial register. This includes any pixels in the serial extensions on either end of the register. This parameter is used by the DSP as the number of pixels to discard when it clears the serial register. It is irrelevant to those rows which are actually read.

1 BIN_SER Serial Binning Factor :

The serial binning factor is the number of pixels in the serial register which will be shifted for each pixel read. By binning pixels the image resolution is sacrificed for higher signal to noise ratio. In low light applications or where the image data is one dimensional this may be used to great benefit.

2 PRE_SER Serial Prescan

The serial prescan is the number of pixels to discard before performing the serial underscan. The prescan is in units of physical pixels and is not affected by the binning factor. No image data is produced.

3 UNDER_SER Serial Under Scan

The serial underscan is the number of pixels to read after the serial prescan. This parameter represents the number of binned pixels to read. One data point is produced for each unit of under scan. Binning is performed.

4 ORG_SER Serial Read Origin

This number represents the number of pixels to be discarded after performing the serial under scan. This parameter is in units of

physical pixels and is unaffected by the binning factor. No image data is produced.

5 READ_SER Serial Read Dimension

The value of this parameter represents the number of pixels to read after the serial origin and before the postscan. These pixels are represented in units of binned pixels. One data point is acquired for each unit of read dimension. Binning is performed.

6 POST_SER Serial Postscan

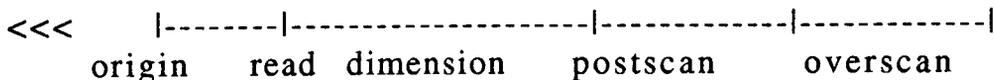
The value of this parameter represents the number of pixels to discard after the read is performed. These pixels are represented in units of physical pixels and are not affected by binning factor. No image data is produced.

7 OVER_SER Serial Overscan

The value of this parameter represents the number of data points to be taken after the postscan is performed. These pixels are represented in units of binned pixels. One data point is acquired for each unit of serial overscan. Binning is performed.

4.1.2 Parallel Read Parameters

The parallel register is read out based on a set of format parameters which imply the following geometry:



The readout operations are performed from left to right. The rows in the origin and post scan are discarded. The rows in the read dimension and the overscan are read off the CCD imager. The discarded rows are in units of physical rows and are not affected by the binning factor. The others are read off the chip and binning is performed. The same binning factor is used on the rows in the read dimension and overscan.

8 CCD_PAR CCD Parallel Dimension

This parameter represents the total length of the CCD parallel register. This parameter is it clears the parallel register. It is irrelevant to those rows which are actually read.

9 BIN_PAR Parallel Binning Factor

The parallel binning factor is the number of rows in the parallel register which will be shifted for each row read. By binning rows the image resolution is sacrificed for higher signal to noise ratio. In low light applications or where the image data is one dimensional this may be used to great benefit.

1 0 ORG_PAR Parallel Read Origin

This number represents the number of rows to be discarded before performing the parallel read. This parameter is in units of physical rows and is unaffected by the binning factor. No image data is produced.

1 1 READ_PAR Parallel Read Dimension

The value of this parameter represents the number of rows to read after the parallel origin and before the postscan. These rows are represented in units of binned rows. Binning is performed. Image data is produced.

1 2 POST_PAR Parallel Postscan

The value of this parameter represents the number of rows to discard after the read is performed. These rows are represented in units of physical rows and are not affected by binning factor. No image data is produced.

1 3 OVER_PAR Parallel Overscan

The value of this parameter represents the number of rows to be read after the postscan is performed. These rows are represented in units of binned rows. Binning is performed. Image data is produced.

15.2 Exposure Parameters

Exposure parameters include shutter delays, exposure time, and the number of times the CCD should be cleared prior to the exposure.

Two parameters are used when opening and closing the shutter. Since the shutter takes a certain amount of time to open, and it would be undesirable to begin to time the exposure or read out the CCD before the shutter motion had stopped, these two parameters are provided to allow the user to set the delay which will take place after the camera controller opens or closes the shutter. The values are stored as 16 bit numbers representing milliseconds. Delay times may therefore vary from 0 to 65.535 seconds. If your shutter takes

longer than this to open or close, you will need to write custom open and close routines.

2 0 ODELAY Shutter Open Delay

The shutter open delay is the time, in milliseconds, that the camera controller will pause after opening the shutter to be sure that it is fully open. Typical values for small aperture shutters are around 8 to 10 milliseconds. The value of the shutter open delay may be as great as 65535 milliseconds.

2 1 CDELAY Shutter Close Delay

The shutter close delay is the time, in milliseconds, that the camera controller will pause after closing the shutter before continuing. This is to be sure that we do not begin shifting the charge on the CCD until the shutter is closed. Typical times for small aperture shutters are around 10 to 20 milliseconds. The value of the shutter close delay may be as great as 65535 milliseconds.

2 2 EXP_TIME_LO lower 16 bits exposure time

2 3 EXP_TIME_HI upper 16 bits exposure time

The exposure time for image acquisitions is stored as a 32 bit value in two 16 bit parameters. The value represents the exposure time in milliseconds. The exposure time may therefore be set from 0 milliseconds to just over 7 weeks. This should be sufficient to cover most applications.

2 4 NUM_CLEARS number of clears

Certain camera commands will clear the CCD as part of their operation. These commands are typically high level image acquisition commands. The number of times that the CCD is cleared in these circumstances is controlled by the value of the following parameter. A typical value for this parameter is two, but it may range from 0 to 65535. A value of zero may be of use under some circumstances where it is not desirable to clear the imager at all. A value of 1 is acceptable under most conditions where a cleared CCD is desired. The default value typically assigned is 2 is conservative. Values greater than 2 are likely to be useful only under unusual circumstances.

15.3 Acquisition Sequence Parameters

The camera may be operated in a sequenced acquisition mode. Each high level image acquisition command actually acquires a sequence of images based on the values of these parameters. Continuous clearing of the CCD may be performed between images.

Clock recombination anti-blooming may be performed during exposures. The camera may acquire images in a frame transfer mode. The camera readout may be performed at either of two speeds.

2 5 NUM_IMAGES number of images

The NUM_IMAGES parameter determines the number of images that will be acquired in each acquisition sequence.

26 IM_DELAY_LO lower 16 bits of image delay

27 IM_DELAY_HI upper 16 bits of image delay

The camera controller will pause between images in the sequence for the number of milliseconds represented by the value of IM_DELAY parameter. The parameter is represent as two 16 portions to allow for delays of greater than 65.535 seconds. The delay is executed after the first image in the sequence. If the number of images is set to 1, the camera controller will still execute the delay before responding with the 'OK' prompt.

3 0 CCLEAR Continuous Clear Flag

The value of this parameter determines whiether the camera will perform continuous clearing of the CCDbetween exposures.

- 0 = continuous clearing disabled
- 1 = continuous clearing enabled

3 1 ANTI-BLOOM Anti-Blooming Flag

The value of this parameter will determine whether the camera will perform clock recombination anti-blooming during the exposures.

- 0 = anti-blooming disabled
- 1 = anti-blooming enabled

16. Camera Controller FORTH Software

16.1 FORTH Command Interpreter

All communication between the camera and the host computer consist of ASCII text strings. All commands from the host to the camera consist of numeric data and FORTH commands. The MCU's FORTH interpreter processes these strings and call the appropriate routines. Some of the commands require operands to be passed previously. A selection of static parameters reduces the number of operands that need to be passed. Parameters may be stored as system defaults. A series of commands may be issued to the camera by sending a series of parameters and commands separated by spaces and followed by a carriage return. A maximum of 256 characters may be sent at any time including the carriage return. The strings issued to the camera may include command definitions, and these commands may be stored as part of the default command set.

The microcontroller used in the AIS1 camera includes an eight kilobyte ROM that contains a FORTH-83 compliant FORTH interpreter. All the software written for the MCU is written either in FORTH or 68HC11 assembler instructions.

The MCU program may be considered to be layers of code with increasing degrees of complexity. Each layer is built out of functions contained in the preceding layers. The camera's FORTH dictionary may be selectively removed from interactive access, effectively removing selected functions. In this way, a programmer can take advantage of the built-in FORTH interpreter for processing the command strings, but limit the number and, more importantly, type of functions available to the user.

At the bottom level is the New Micros Max-FORTH F83 compliant dictionary and the extensions that have been added by the MCU's manufacturer. The next level consists of MCU specific and camera specific I/O routines. These routines are used to manipulate the MCU's control registers and the hardware in the system. These include functions that read and set the CCD temperature, read the case temperature, open and close the shutter, initialize and communicate with the DSP, set the camera gain, set and return parameter list values, and perform other low level operations.

A variety of functions that perform charge shifting and pixel conversion exist at the intermediate level. All of these operations include action by the DSP. Out of these functions the user may build

customized charge shifting and pixel readout sequences. Under normal circumstances these functions will be called indirectly through the short form command interpreter described below.

There is also a set of high level image acquisition commands. There are functions for acquiring all the standard types of images from the camera, including bias frames, dark frames, and object exposures. The elemental parts of these high level image acquisition routines are also available, including functions to clear the CCD, expose it to light or integrate dark current, and to read the charge off the CCD. Under normal circumstances the user will call the highest level functions rather than re-creating them from their parts. Most often these functions will be called via the short form commands.

A short form command interpreter is provided to minimize the overhead associated with the serial transmission of commands from the host computer to the MCU. All the functions necessary for normal operation of the camera are available.

16.2 FORTH Command Descriptions

The Following sections describe the various FORTH commands in some detail. They are presented in order of complexity with the highest level commands listed first. The 'short form commands' are listed first, for details on their operation, please see the descriptions of the functions they call in the following sections.

16.2.1 Short Form Commands

A set of short form commands is provided that is particularly useful to those very familiar with the command set or for programmed hardware interfaces, such as the Advanced Technologies VMEbus interface. All commands in the set are abbreviated to three letters in order to maximize the transfer rate of comands between the host computer and the camera controller.

CXX	COLD	restart the FORTH interpreter
CIN	INIT	initialize the camera
ST@	TEMP!	CCD temperature short form
STC	CCD-TEMP?	CCD temperature long form
ST!	TEMP!	set the CCD temperature
SCF	CISC-OFF	disable continuous clearing
SCT	CISC-ON	enable continuous clearing
SBF	CRAB-OFF	disable anti-blooming
SBT	CRAB-ON	enable anti-blooming
SSO	OPEN	open the shutter
SSC	CLOSE	close the shutter

SIV	INIT-VOLTS	initialize CAM voltages
SIF	INIT-FORMAT	initialize format parameters
SIS	INIT-STATES	initialize timing parameters
SPD	SET-PAR_DELAYS	set the parallel delays
FF?	FORMAT?	fetch the format parameters
FSF	INIT-FORMAT	initialize format parameters
LPB	PIX-BIN	bin N pixels
LRB	ROW-BIN	bin N rows
LPC	PIX-CLR	discard N pixels
LRC	ROW-CLR	discard N rows
LPR	PIX-READ	read N pixels
LRR	ROW-READ	read N rows
LCW	CAM-WRITE	write a value to a camera address
IIL	EXPOSE_CCD	integrate light
IID	INTEGRATE_DARK	integrate dark
IRD	READ	read the image off the CCD
ICL	CLEAR	clear the CCD of all charge
IAB	BIAS	acquire bias
IAD	DARK	acquire dark
IAL	OBS	acquire light
ACE	ALL-CLKS-EN	enable all the CCD clocks
ACD	ALL-CLKS-DIS	disable all CCD clocks

16.2.2 High Level Image Acquisition Routines

There is a set of high level image acquisition routines which are used to initiate a complete CCD exposure and readout sequence. These routines are described below.

READ

Read the image data off the CCD. The CCD is read out based on the current format parameters.

CLEAR

Clear the entire CCD array, NUM_CLEARS times. All charge is cleared off the CCD.

BIAS

Acquire a bias frame, NUM_IMAGES times. The shutter is closed, CISC stopped, the CCD is cleared, the image data is read off the chip based on the current format parameters,

continuous clearing of the chip is begun, if enabled, and then a delay of IM_DELAY milliseconds is executed. This entire sequence is executed NUM_IMAGES times.

DARK

Acquire a dark reference frame, num_images times. The shutter is closed, CISC stopped, the CCD is cleared, dark current is integrated for EXP_TIME milliseconds, the image data is read off the chip based on the current format parameters, continuous clearing of the chip is begun, if enabled, and then a delay of IM_DELAY milliseconds is executed. This entire sequence is executed NUM_IMAGES times. If enabled, clock recombination anti-blooming is performed during dark frame integration.

OBS

Acquire a light frame, num_images times. The shutter is closed, CISC stopped, the CCD is cleared, the shutter is opened, light is integrated for EXP_TIME milliseconds, the shutter is closed, the image data is read off the chip based on the current format parameters, continuous clearing of the chip is begun, if enabled, and then a delay of IM_DELAY milliseconds is executed. This entire sequence is executed NUM_IMAGES times. If enabled, clock recombination anti-blooming is performed during light integration.

16.2.3 Low Level Image Acquisition Routines

There is a set of low level image acquisition routines. A small amount of additional overhead is associated with a read sequence based on low level routines, due to increased participation by the 68HC11, but very complicated read sequences can be constructed. Not all programmers will need to use these routines, as normal operational modes are fully supported by the high level image acquisition routines. Low level image acquisition routines include the following:

PIX-BIN <N> PIX-BIN

Bin N pixels in the serial direction. N serial shifts are performed. Binning factor is not used. N is an integer from 0 to 65535.

ROW-BIN <N>ROW-BIN

Bin N rows in the parallel direction. N parallel shifts are performed. Binning ifactor is not used. N is an integer from 0 to 65535.

PIX-CLR <N>PIX-CLR

Discard N pixels in the serial register. Binning factor is not used. N is an integer from 0 to 65535.

ROW-CLR <N>ROW-CLEAR

Discard N rows in the parallel register. Binning factor is not used. N is an integer from 0 to 65535.

PIX-READ <N>PIX-READ

Read N pixels out of the serial register. Binning is performed. N is an integer from 0 to 65535.

ROW-READ <N>ROW-READ

Read N rows off the CCD. Binning is performed. N is an integer from 0 to 65535.

16.2.4 Shutter Control

Two routines exist for manipulating the camera's shuuter. Their purpose is self-evident. There are shutter delay parameters which determine how long the controller will wait for the shutte blades to actually finish moving. See the parameter descriptions for more information.

OPEN

Open the camera's shutter.

CLOSE

Close the camera's shutter.

16.2.5 Temperature Measurement

TEMP?

Reports the temperature of the CCD in an unformatted ASCII string of the form:

-85<CR>

TEMP!

<N>TEMP!

Set the desired CCD temperature to a new value. N is the desired temperature. N is an integer that may vary from -200 to +55.

CCD-TEMP?

The CCD temperature is reported as an ASCII text string of the form :

<CR> CCD temperature = -125 <CR>

TEMP-DIFF?

Reports the difference between the temperature of the CCD and the desired temperature in an unformatted ASCII string of the form:

-85<CR>

16.2.6 Format commands

These two words may be used to set and examine the format parameters. In each case the format parameters are passed as a simple character stream.

FP! <M><N>FP!

Set the value of a format parameter. N is the parameter index in the parameter table, and M is the new value for the parameter.

FP@

<N>FP@

Fetch and print a parameter value. N is the parameter index in the parameter table.

INIT-FORMAT

Initialize the format parameters. The current format parameter values are passed to the DSP sequencer for use during subsequent image acquisitions. This function, or one of its short forms 'SIT' or 'FSF' must be called for

parameter changes to take effect. If the 'FP!' command is used to change the parameters, then this command is unnecessary, as it is included in 'FP!'.

FORMAT?

Fetch the current format parameters. The current format parameters are reported in a single text string in the following order:

```
CCD_SER  
BIN_SER  
PRE_SER  
UNDER_SER  
ORG_SER  
READ_SER  
POST_SER  
OVER_SER  
CCD_PAR  
BIN_PAR  
ORG_PAR  
READ_PAR  
POST_PAR  
OVER_PAR
```

They are reported on a single line.

16.2.7 CCD voltages

Two routines exist for handling the CCD voltages. The first prints the voltages in a table. The second initializes the clock cards with the new voltages. Simply changing the values in the voltage tables does not affect the actual voltages applied to the CCD. Calling INIT-VOLTS sets all the voltages to the current table values.

SHOW-VOLTS

This function is used to generate a listing of the current CCD clock voltage DAC settings. The voltages themselves cannot be read by the system. This function lists the eight bit values that are to be written to the DACs during an INIT-VOLTS or SIV operation. The actual voltages depend on the way the various parts of the hardware are

configured. If you need this function, you probably know how the numbers relate to the voltages.

INIT-VOLTS

Initialize the CCD voltages. The current values of the clock voltages are downloaded to the DSP, and written to the DACs on the clock card. This function, or its short form **SIV** should be called after changing the voltage table entries in order to have the new settings take effect. See also **SIV**.

16.2.8 CCD timing

The timing of the clock waveforms applied to the CCD is programmable in this camera system. The desired CCD clock timing is stored in tables. These functions are used to examine those timing tables. **INIT-STATES** is used to initialize the DSP sequencer with the current table entries. For a more detailed description, please see the chapter describing the programmable timing.

INIT-STATES

Initialize timing states. The current timing tables are downloaded to the DSP sequencer for use as clock timing in subsequent CCD readouts. This command, or its short form '**SIT**', must be called after changing the state tables in order for the changes to take effect. See also **SIT**.

SET-PAR_DELAYS

<N> SPD or <N> SET-PAR_DELAYS

Set the parallel delay times to a new value. The AIS cameras support different delay times between the various stages of a parallel shift. This feature is usually unused. This function allows one to set all the delays to a single value. It is convenient. The new values will not take effect until a **SIF** or **INIT-FORMAT** command has been received.

SHOW-PAR_STATES

This function is used to generate a listing of the timing states currently in the table associated with charge shifting in the parallel direction. This is performed the

same way in both the slow and the fast modes. This function is used by the user who wishes to modify the timing states to change the way the parallel clocks are run. The average user has no need to view or modify these tables.

SET-PAR_DELAYS

This function is used to set the parallel clock delay values to some value. The camera software supports different values for parallel delay between the steps in the parallel shift, but these are usually set to the same value

SHOW-ANTI-BLOOM_STATES

This function is used to generate a listing of the timing states currently in the table associated with clock recombination anti-blooming.

SHOW-SER_STATES

This function is used to generate a listing of the timing states currently in the table associated with the shifting of charge in the serial register during a slow mode bin, discard, or read operation. This table contains the serial clock timing only. This function is for use by the user who is modifying the camera's timing and needs to see the current settings.

SHOW-ANA_STATES

This functions are used to generate a listing of the states currently in the tables associated with pixel readout in the slow mode. The signals represented here are generated at time intervals determined by the entries in the SLOW_DELAY table. These functions are of interest to the user who is changing the CCD and conversion timing. The average user has no need to see these tables.

16.2.9 Camera Speed

The AIS1 readout speed is adjustable from approx 40kHz downward. The exact speed depends on some timing variables, most importantly the integration time of the dual slope integrator. The SPEED may be adjusted over a much wide range and. As the

ANALOG TO DIGITAL convertors architecture is that of a dual slope integrator, the gain of the system is directly affected by the integration time. The integration time is determined by the values of ANA_DELAYs 7 and 9. Several words are provided here to set the camera's slow readout rate to convenient values.

ITIME!

<N> ITIME!

This function sets the integration time for the slow speed analog processor. It expects a 16 bit number on the stack with a value ranging from 1 -> 65535. This value is used by the DSP as a counter for the integration delay. The DSP delays 100 ns for every unit in the integration time. Integrations may therefore vary from 100ns to 6.5535 ms. In practice, there is approx 350 ns overhead in the delay time and 100ns integration times are not possible. Additionally, if the time is set too short, the converter will not have finished converting the previous pixel when the new command to convert comes along. Integration times as short as 1 us should work fine. Extremely long integrations will saturate the converter.

SPEED!

<N> SPEED!

This function is used to set the readout speed to a particular value. It expects the new value for the integration time on the stack. It is used primarily as a convenience for the definition of the functions that follow.

40KHZ, 35KHZ, 30KHZ, 20KHZ, 15KHZ, 10KHZ, 5KHZ

These functions are used to set the camera's readout speed. They set the readout mode to slow, and set the analog delays appropriately to establish the requested readout speed. The speeds that result are fairly obvious from the function names.

16.2.10 Continuous Clearing of the CCD

It is often desirable to continuously clear charge off the CCD between image acquisitions. If there is no shutter in the system or if

the temperature of the CCD is fairly high this is very desirable. In a cryogenic camera, this is not as important since the dark current is typically very low. The 'CISC' terminology comes from the Photometrics CC200 cameras, where it stood for 'Clear Image and Storage Continuously'. Continuous clearing of the camera is performed by the DSP. A parameter is maintained that may be polled at any time to determine if it should do so.

START-CISC

Begin clearing the CCD continuously. Does not affect the status of the CCLEAR parameter.

STOP-CISC

Stop the continuous clearing of the CCD. Does not affect the status of the CCLEAR parameter.

CISC?

Examine CCLEAR parameter and begin to continuously clear the CCD if it is true.

CISC-ON

Enable continuous clearing of the CCD between frames.

CISC-OFF

Disable continuous clearing of the CCD between frames.

16.2.11 Clock Recombination Anti-Blooming

Clock recombination anti-blooming is controlled by the DSP. It may be told to begin the process through a Host Command at any time that it is not executing any other host command. It will continue until the 68HC11 tells it to stop. The 68HC11 will call this function during exposures if the ANTI-BLOOM parameter is set to true.

START-CRAB

Begin clock recombination anti-blooming. Does not affect the status of the ANTI-BLOOM parameter.

STOP-CRAB

Stop clock recombination anti-blooming. Does not affect the status of the ANTI-BLOOM parameter.

CRAB?

Examine the ANTI-BLOOM parameter and begin clock recombination anti-blooming if it is true.

CRAB-ON

Set the ANTI-BLOOM parameter to true.

CRAB-OFF

Set the ANTI-BLOOM parameter to false, and stop clock recombination anti-blooming if it is in progress.

16.2.12 CCD Image Integration

These routines are used to manipulate the exposure counter, to initiate exposures, terminate exposures, and perform complete light or dark current integrations.

EXP_LEFT@

This function is used to query the time remaining in the current exposure sequence. It may be called at any time.

EXP_LEFT!

This function is used to set the remaining exposure time to some value. It is used internally by the exposure commands and is not normally needed by the user. Programmers may use this function in custom exposure commands if so desired.

EXP-START

This function is used to begin an exposure sequence. The EXP_TIME parameter is copied into the EXP_LEFT variable, the shutter is opened, and the exposure timer is started. Control is returned to the host immediately, without waiting for the exposure to complete.

EXP-PAUSE

This function is used to pause an exposure in progress. The shutter is closed and the exposure timer is stopped. The CCD is not read out and the remaining exposure time is maintained so that the exposure may be completed by issuing the **EXP-RESUME** command discussed below.

EXP-RESUME

This function is used to resume an exposure that has been interrupted via the EXP-PAUSE command. The shutter is reopened and the exposure timer is restarted where it left off. If the EXP_LEFT variable has been changed in the interim, that is the exposure time that will follow.

EXP-ABORT

This function is used to abort an exposure in progress. The shutter is closed and the exposure counter is stopped. The EXP_LEFT variable is zeroed and continuous clearing of the CCD is begun if the CISC parameter is true. The CCD is not read out. No image data is generated. If it is desired to read the data at this point it may be done via the READ command.

EXP-LEFT?

This function is used to query the time remaining in the current exposure cycle.

EXP-WAIT

This function is used to wait for the completion of the current exposure cycle.

EXP-STOP

This function closes the shutter, and stops clock recombination anti-blooming. Used internally by the camera, it is not needed by the typical user.

EXPOSE_CCD

This function is used to perform a complete exposure cycle on the CCD. Calls EXP-START, EXP-WAIT, and EXP-STOP defined above.

INTEGRATE-LIGHT

Integrate light onto the CCD for EXP-TIME milliseconds. The shutter is opened, the exposure delay is performed, and the shutter is closed. Clock recombination anti-blooming is performed during the exposure if it is enabled.

INTEGRATE-DARK

Integrate dark current onto the CCD for EXP-TIME milliseconds. the shutter is assumed to be closed already. Clock recombination anti-blooming is performed during the exposure if it is enabled.

16.2.13 Time Measurement

These routines are used to manipulate the system timer, a free running counter incremented every millisecond, and to measure the passing of time. These may be useful to the programmer who is constructing new image acquisition sequences or any other function that requires the accurate measurement of time.

TIME@

This function fetches the cameras current timer value and places it on the FORTH stack as a double length number. This function may be of some use to the user who wishes to create custom FORTH definitions, but is otherwise unnecessary. It is used internally by the system software.

TIME! <D> TIME!

This function expects a double length number on the FORTH stack and sets the system timer to that value. This function may be of some use to the user who wishes to create custom FORTH definitions, but is otherwise unnecessary. It is used internally by the system software.

STOP-TIMER

This function will stop the cameras free running timer. It is not usually necessary to call this function. Useful for debug.

START-TIMER

This function starts the camera's free running timer. It is called at system startup by the INIT routine and need not usually be called by the user or a user program unless STOP-TIMER has been called. If the timer is not running, any function which calls M-WAIT or MS-WAIT will hang the system indefinitely.

MS-WAIT <D> MS-WAIT

This function causes the system to pause for the number of milliseconds specified by the double length number passed on the FORTH stack. It may be of use to one who desires to create custom FORTH definitions. Since it accepts a double length number, very long delays may be generated. See also M-WAIT.

M-WAIT <N> M-WAIT

This function causes the system to pause for the number of milliseconds specified by the 16 bit value passed. This function may be of use to one who desires to create custom FORTH definitions. The number N ranges from 0 to 65535, and the delay may therefore range from 0 to 65.535 seconds. Extremely short delays may be noticeably affected by the small amount of overhead associated with the execution of the task.

16.2.14 System Maintenance

The 'system maintenance' routines are used to perform low level initialization of the camera controller as well as to manipulate the FORTH dictionary. They are not needed normally. They are used by the system internally, and are available to the programmer.

COLD

The camera system software is restarted. This is essentially a software reset.

INIT

Camera initialization function. All parts of the camera are initialized or re-initialized. It is necessary to call this function before using any of the other camera control functions.

VERSION

This function is used to query the camera about the software version that is currently running. Used primarily during software development. Reports the version and a date as an ASCII text string.

EEPROT

This function is used to protect the EEPROM on the controller card from write accesses. It is used internally by the STORE function to protect the EEPROM after it has been written. It is not normally needed by the user.

EEUNPROT

This function is used to unprotect the EEPROM on the controller card. This allows the EEPROM locations to be written over. This function is used internally by the STORE function before writing the EEPROM. It is not normally used by the user.

EE-!

$\langle N \rangle \langle M \rangle EE-!$

This function is used to write a value into the EEPROM memory. It accepts two 16 bit numbers on the stack and operates just like the normal FORTH '!' command. N is the data and M is the address.

EE-C!

$\langle N \rangle \langle M \rangle EE-C!$

This function is used to write a value into the EEPROM memory. It accepts two numbers on the stack and operates just like the normal FORTH 'C!' command. N is the 8 bit data and M is the 16 bit address.

STORE

This function is used to store the current state of the camera's FORTH dictionary as the default. This includes all the format parameters, voltage and timing tables and any variables. Also, any user defined functions which the user desires to include in the default dictionary are saved.

RESTOR

This function may be used to restore the dictionary from the EEPROM at any time. Primarily used for debugging, this command is not needed in normal operation of the camera. The camera performs a RESTOR command on power-up reset or in response to a CXX command.

CUT-DICT

This function is used to cut the FORTH dictionary at any point so that lower level functions will not be accessible to the user or host computer. The primary use of this is to prevent the user from changing parameter or timing information that should not be changed. The system manager might find it useful to cut the dictionary at a fairly high level once all the parameters and timing are at the desired state. The function returns two very important pieces of information: the address of the dictionary link that was broken and the value that should be placed there to restore the dictionary.

BRIDGE-DICT

This function is similar to CUT-DICT except it allows to continue the dictionary at any point. This can be used to disable access to a set of words
'WORD2' WORD1 BRIDGE-DICT

disables all words below WORD2 including WORD1.

16.2.15 DSP Software Support

The DSP program is developed on a PC, Macintosh, or Sun platform and the '.LOD' file generated is downloaded to the camera using this function.

DNLD

download a new DSP sequencer program to the camera.

DSP-BOOT

reset the DSP sequencer in it's bootstrap mode

DSP-DUMP

This function dumps DSP program code from the FORTH dictionary to the DSP sequencer.

DSP-OK?

check that the DSP is operating correctly and responding to host commands. Used internally by a variety of camera control functions to check status before proceeding.

INIT-DSP

This function initializes the DSP sequencer. The DSP is reset and program code is transferred to it. The DSP-OK? function is then used to determine if the DSP is responding correctly. If not, an error message is generated.

DSP8@

fetch 8 bit data values from the DSP56001 sequencer. The datum is left on the FORTH stack when it has been fetched.

DSP16@

fetch 16 bit data values from the DSP. The datum is left on the FORTH stack when it has been fetched.

DSP8! <N>DSP8!
pass 8 bit data values to the DSP. N is the datum.

DSP16! <N>DSP16!
pass 16 bit data values to the DSP. N is the datum.

16.2.16 Test Functions

A small set of test functions are provided for hardware debug. Most are repetitive words that are useful for testing small subsets of the clock and analog cards. Useful for debugging the camera electronics. Not needed in normal operation.

RREAD or READS

read the CCD repeatedly until a character is received through the serial communications link.

RCLEAR

clear the CCD repeatedly until a character is received through the serial communications link.

RBIAS

acquire bias frames repeatedly until a character is received through the serial communications link.

ROBS

acquire exposures repeatedly until a character is received through the serial communications link.

RDARK

acquire dark frames repeatedly until a character is received through the serial communications link.

16.3 Creating Custom FORTH Definitions

It is possible to create custom FORTH commands for the AIS1 camera if it is found that the basic command set is not sufficient for a given application. This is a very simple process.

The AIS1 camera controller contains a F68HC11 microcontroller from New Micros Inc located in Dallas Texas. It is a custom version of Motorola's standard MC68HC11A8 microcontroller which New Micros has had manufactured with a FORTH interpreter in the on chip ROM. The camera controller expects FORTH commands through the serial port. All communication with the camera controller during operation of the camera consists of ASCII text strings passed from the host computer to the camera. These strings are interpreted as FORTH commands.

The text strings passed may create new FORTH commands by the normal means of adding words (functions) to the FORTH command dictionary. Definitions are ASCII strings of the following format:

```
: NAME      (FORTH commands) ;
```

The definition begins with the receipt of the colon character. NAME is the name of the new definition. The FORTH commands are a string of ASCII text which tell the interpreter what to execute when the function NAME is called. The definition is terminated by the semi-colon character.

For example:

```
: MYWORD    OPEN CLOSE READ ;
```

defines a new word named MYWORD that opens the shutter, closes the shutter and reads the data off the CCD. All the words called in the FORTH commands section of the definition must already exist in the

FORTH dictionary or an error message will result and the complitaion will be terminated.

This rather simplistic description is intended only for the user who desires to create simple commands. A much more complete description of FORTH and FORTH definitions can be found in such useful refernces as :

MAX-FORTH Reference Manual
New Micros Inc.
1601 Chalk Hill Road
Dallas Texas 75212
1-214-339-2204

Starting FORTH
Leo Brodie, FORTH Inc.
Prentice-Hall, Inc., Englewood Cliffs, N.J.
ISBN 0-13-842930-8

Custom words may be defined through any of the normal means of communication with the camera. An RS-422 terminal connected to the camera's serial port for example, or an ASCII text file containing new definitions may be downloaded to the camrea via the "sendfile" program from the UNIX command line, or the "say" program may be used from the UNIX command line to issue a new definition which can fit on a single line.

Additionally, custom FORTH words may be added to the camera's default dictionary by storing them in the EEPROM located on the controller card. This is also a very simple process and is described in the following section of this document.

16.4 Saving and Restoring the FORTH Dictionary

The FORTH dictionary may be saved at any time by the user or a control program by issuing the STORE command. This command copies the current dictionary, stored in the 68HC11's RAM memory, into the EEPROM memory. At startup, the contents of the EEPROM are copied into the RAM, returning the camera to the state it was in when STORE was last called. All the camera system parametrs and timing tables are restored along with the standard dictionary of FORTH commands and any commands that the user had defined previous to issuing the STORE command. The user is free to store whatever functions she would like in the EEPROM.

17. DSP56001 Sequencer Software

The DSP software consists primarily of a set of routines and associated subroutines required to set CCD clock rail voltages and output amplifier operating point voltages, to shift charge on the CCD or CCDs, to manipulate CCD clock lines and analog processor control signals, and to perform analog to digital conversion of the pixel data. The DSP routines may be divided into three categories: System configuration, system test, and CCD control and pixel conversion.

System control routines are used to set CCD clock voltages and output FET voltages as well as establishing format parameters for CCD readout. Low level functions to read and write DSP memory locations allow the user to directly manipulate the camera hardware.

System test functions provide facilities for testing and debugging the camera system hardware. A simple test function is provided which allows the controller or host computer to test the DSP's proper operation. Additional functions are provided which allow the user to generate specific patterns of data and or voltages at various points on the clock generator modules for diagnostic test.

Charge shifting and conversion functions range in complexity from simple primitives such as shifting one pixel in the serial register to fully contained functions including clearing and readout of the CCD as well as "time delay integration" operation. By combining primitive commands the user may construct complex readout sequences not directly supported in the higher level operations.

DSP operations are flexible enough to provide for a wide range of camera configurations. A high level command set combined with a set of low level commands allow for a variety of modes of interaction with the sequencer. The casual user may rely only on the built in command sequences to acquire image data, while a programmer familiar with the system hardware and software may access the hardware directly through the DSP's low level commands to generate custom CCD control sequences.

17.1 68HC11 Host Interface

The 68HC11 passes commands to the DSP through the DSP's command vector register, CVR, the value written there determines which of the host commands will be executed. There are a total of 28 command vectors which may be passed the first 26 correspond to the DSP's normal exception processing including external interrupts, software interrupts, and other interrupts associated with the dsp's various peripherals. Through the CVR, we may force recognition of

any of these exceptions. Some of those are implemented in this code. In addition, there are 12 command vectors reserved for use by the host. It is through these command vectors that we normally force execution of dsp routines. The routines which we may cause the dsp to execute include the following:

17.2 DSP Functions

<u>HV#</u>	<u>FUNCTION</u>	<u>DESCRIPTION</u>
6	say_ok	respond to 6811 as test none
7	cam-write	write to camera address addr data 16 bit
8	cam_read	read from camera address
9	volt_test	test clock voltage dac's
10	cam_ctl!	write to cam control latch
11	init-volts	set cam clock voltages 24 8 bit voltage settings
12	init-states	set camera readout timing
19	init-format	set readout parameters xx 16 bit format parameters
20	read	read the ccd none
21	clear	clear the ccd 1 8 bit number of clears
22	pix-bin	shift pixels to summing well 1 16 bit # of pixels
23	row-bin	shift rows into ser reg 1 16 bit # of rows
24	pix-discard	clear pixels in ser reg 1 16 bit # of pixels
25	row-discard	clear rows 1 16 bit # of rows
26	pix-read	read pixels 1 16 bit # of pixels
27	row-read	read rows 1 16 bit # of rows
30	anti-bloom	perform anti-blooming

10.3 DSP Parameters

<u>address</u>	<u>parameter</u>	<u>description</u>
x:0	ccd_ser	serial register length
x:1	bin_ser	serial binning number
x:2	pre_ser	serial register extension length
x:3	under_ser	serial register underscan
x:4	org_ser	serial prescan length
x:5	read_ser	serial read length
x:6	post_ser	serial postscan length
x:7	over_ser	serial register overscan
x:8	ccd_par	parallel register length
x:9	bin_p	parallel binning number
x:10	org_par	parallel prescan length
x:11	read_par	parallel read length
x:12	post_par	parallel postscan length
x:13	over_par	parallel register overscan
x:14	pdir_cam0	parallel shift direction for CAM0
x:15	pdir_cam1	parallel shift direction for CAM1

17.4 Creating Custom DSP56001 Software

The programmer wishing to make changes to the DSP56001 software may do so using the tools provided. The DSP software must be in its fully assembled form, that of the Motorola ".lod" file such as that produced by the DSP assembler. This assembler is available on several platforms, IBM compatible, Macintosh, and Sun workstation. The assembled code is stored in the 68HC11's FORTH dictionary using the DNLD command and may be stored in the EEPROM by issuing the STORE command. The DSP is initialized in its usual way by executing the DSP-BOOT and DSP-DUMP commands, which reset the DSP in bootstrap mode, and dump the code to the DSP through the host interface.

18. System Performance

Readout noise and Gain:

speed	gain (e-/ADU)	noise (e-)
20 kHz	0.57	3.6
10 kHz	0.21	2.5
5 kHz	0.09	2.4

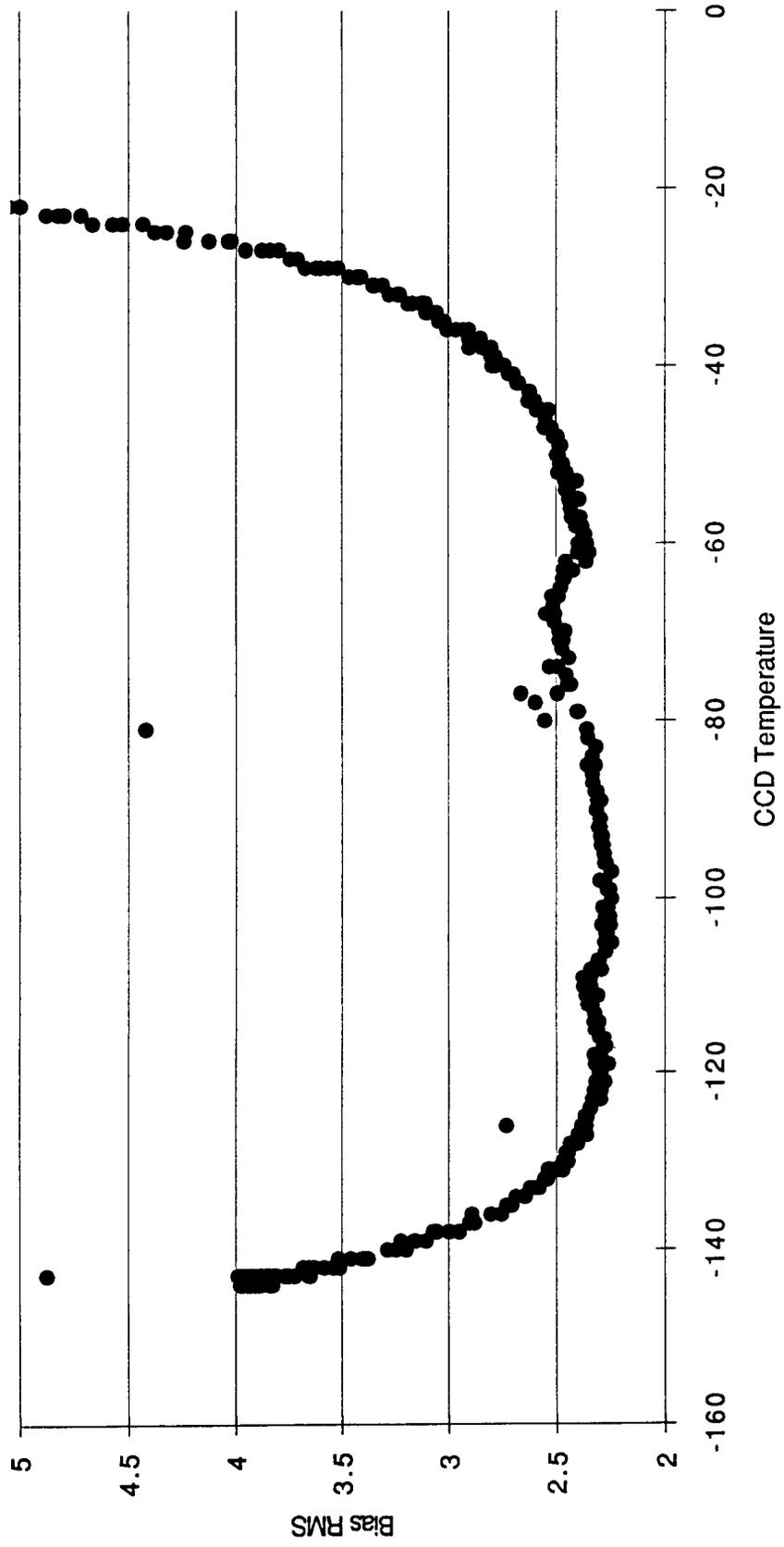
Readout noise was affected by CCD temperature. The optimum performance was achieved between -60C and -100C. At lower or higher CCD temperature the noise increased rapidly (see graph Bias RMS vs Temperature).

CTE was good at all temperatures. CTE was at least 0.99999 in both serial and parallel registers

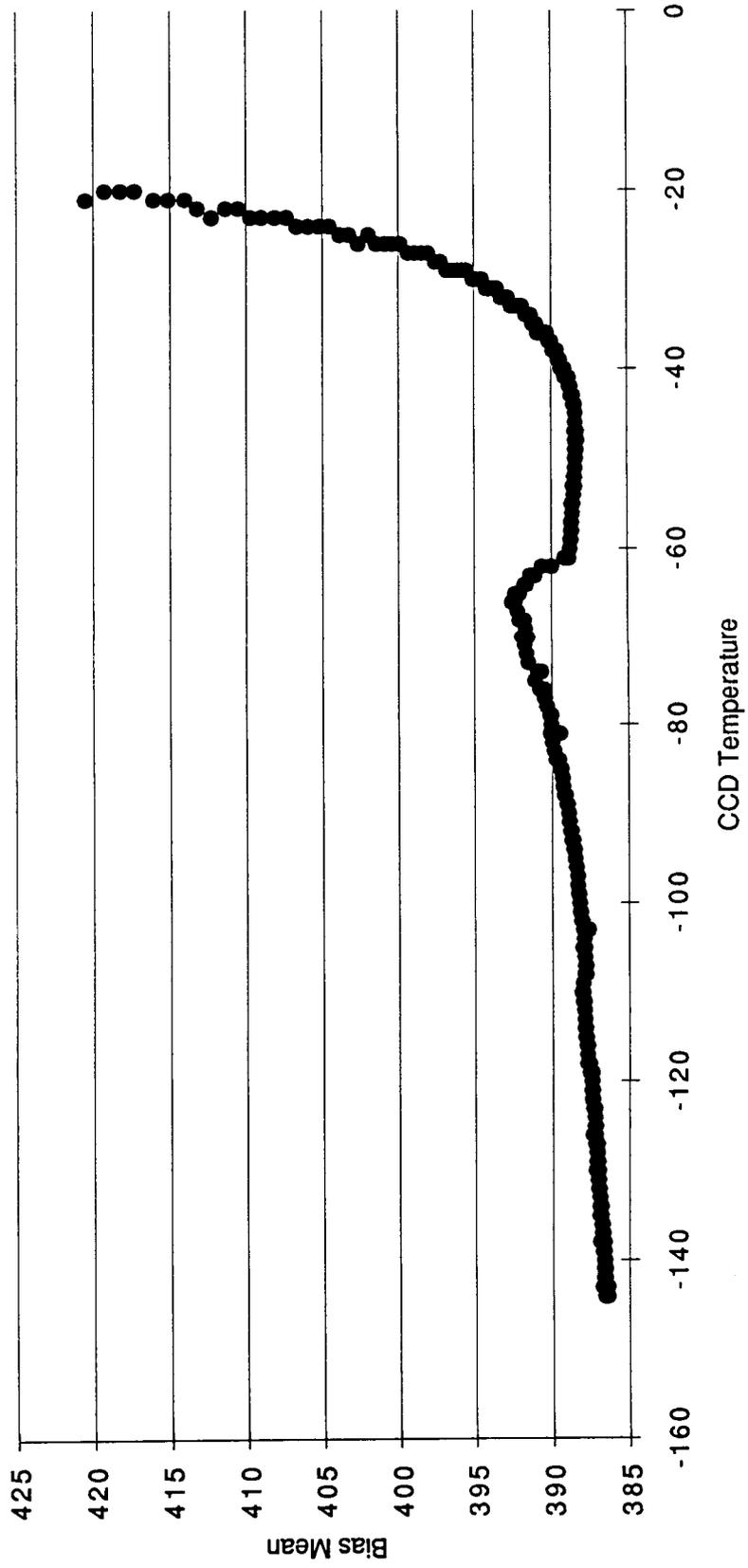
The bias and gain stability was excellent as is demonstrated in the graph "Bias Stability vs Time".

Bias RMS vs Temperature 9/15/92

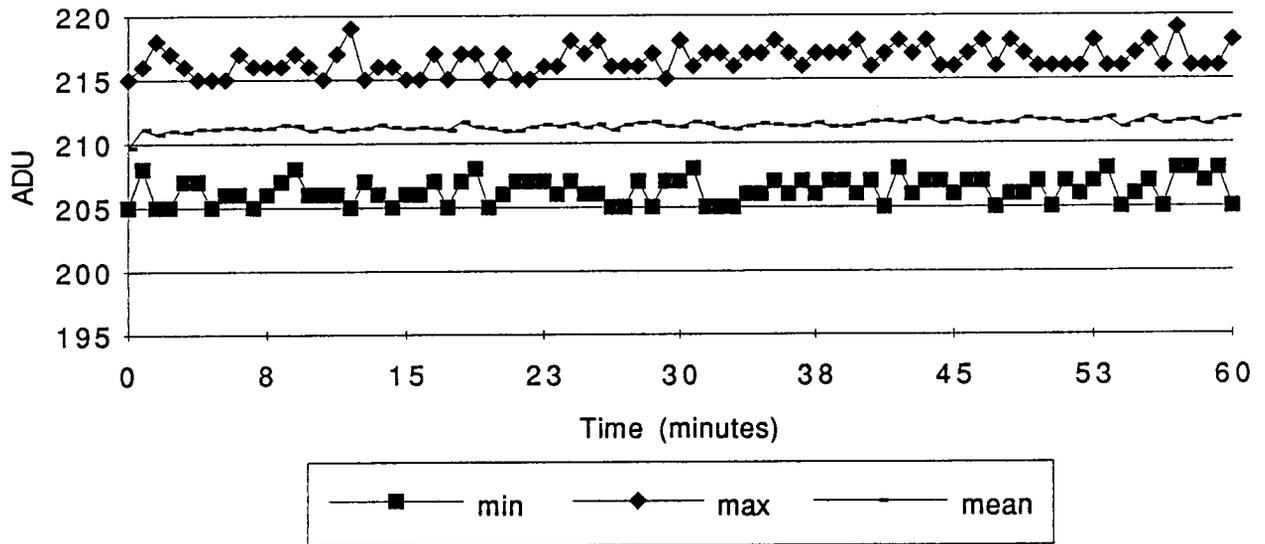
AIS1 Channel 1 M745A CCD #1340AN-10-01 Output Amplifier C



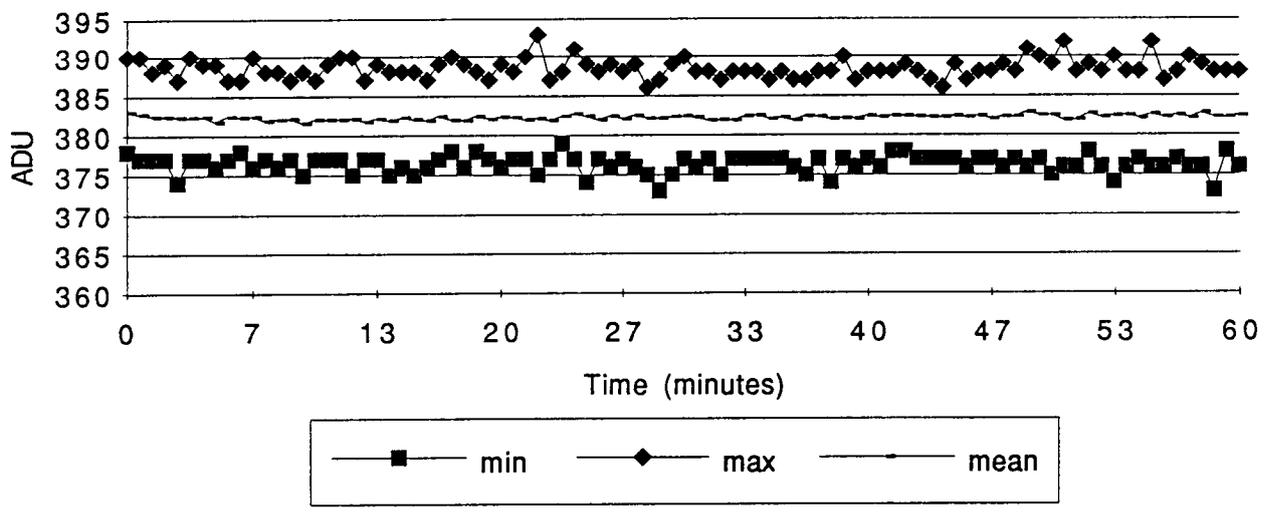
Bias Mean vs. Temperature 9/15/92
AIS1 Channel 1 M745A CCD#1340AN10-01 Output Amplifier C



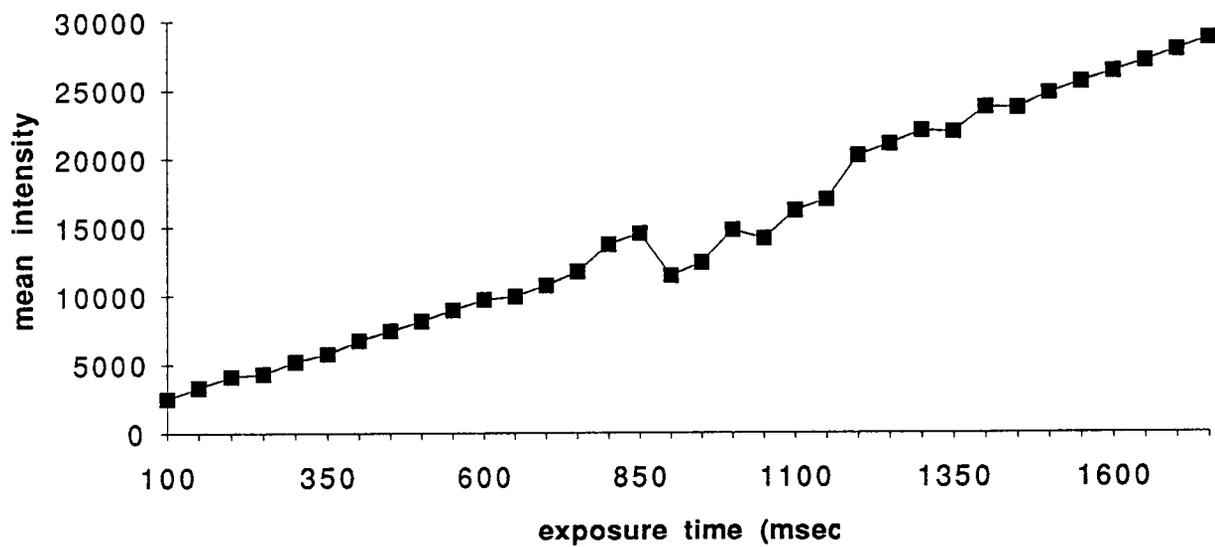
Bias Stability AIS1 Channel 0 CCD Output Amplifier B



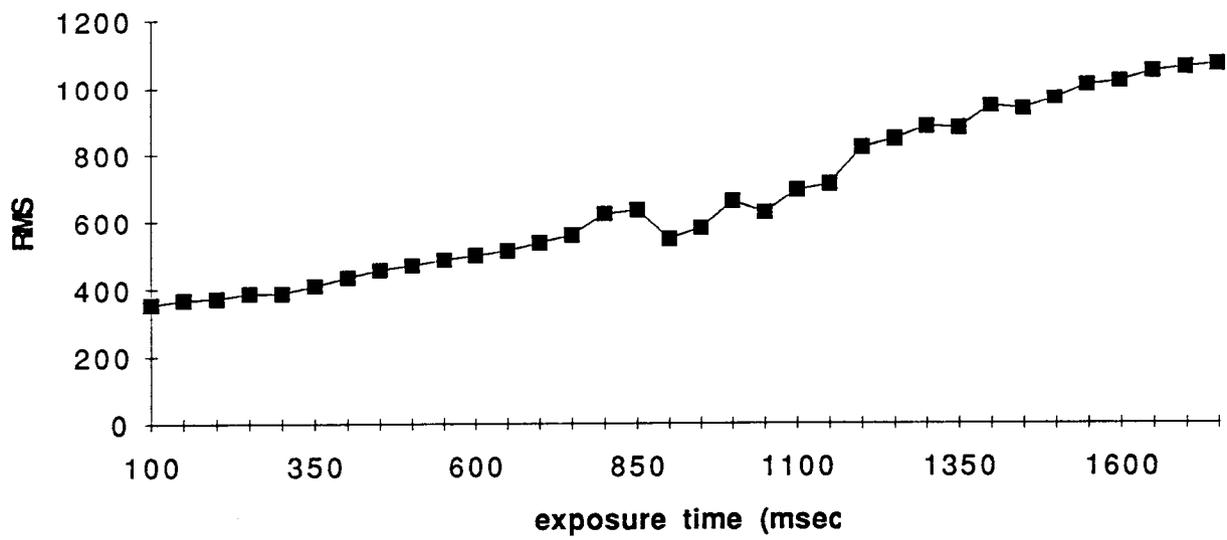
Bias Stability vs Time AIS1 Channel 1 CCD Output
Amplifier C



linearity plot of AIS1 channel 1 #1 9/13/92 3P



linearity chart 2 for lin1chan1 RMS vs expti



Appendix A: Camera Controller FORTH Source Code

```
( COLD )
HEX
100 TIB !
100 TIB 2+ !
200 DP !
```

```
: VERSION CR CR ." AIS revision 5.7 9/18/92 " CR ;
```

```
( * ***** )
( * )
( * AIS.FOR  forth code for control of the AIS AND AIS2 camera systems )
( * )
( *      copyright 1992 Advanced Technologies )
( *      Division of Photometrics ltd. )
( * )
( * )
( * )
( * HISTORY : revision 5.7 9/18/92 )
( *      added some of the features and comments from AIS2 )
( * )
( *      revision 5.6 5/9/92 )
( *      first attempt at background exposures )
( *      sped up the dsp response checks )
( *      replaced "M-WAIT" with "0 MS-WAIT" )
( * )
( *      revision 5.5 4/13/92 )
( *      fixed long standing (though unobserved) bug )
( *      in DSP16! wherein large numbers were handled wrong )
( * )
( *      revision 5.4 3/17/92 )
( *      minor changes to a variety of things )
( * )
( *      revision 5.3 3/10/92 )
( *      Changed timer functions to interrupts. )
( * )
( *      revision 5.2 1/10/92 )
( *      Added RTI for temp control )
( *      New state table structure )
( *      Complete reorganization )
( *      No longer backward compatible )
( * )
( * ***** )
```

```
( * some useful constants
```

```
0 CONSTANT FALSE
1 CONSTANT TRUE
```

```
( * some useful temporary storage locations
```

```
VARIABLE TEMPO
VARIABLE TEMP1
```

```
( * ***** )
( )
(          68HC11 Registers
(          -----
( )
( * declaration of constants representing 68hc11 control registers and ports
( * not all are used in this code,
( * but all are made available for user commands
HEX
B000 CONSTANT PORTA      ( * I/O PORT A DATA REGISTER

( B002 CONSTANT PIOC      ( * PARALLEL I/O CONTROL REGISTER
( B003 CONSTANT PORTC    ( * I/O PORT C DATA REGISTER
( B004 CONSTANT PORTB    ( * OUTPUT PORT B DATA REGISTER
( B005 CONSTANT PORTCL   ( * ALTERNATE LATCHED PORT C

( B007 CONSTANT DDRC     ( * DATA DIRECTION REGISTER FOR PORT C
( B008 CONSTANT PORTD   ( * I/O PORT D DATA REGISTER
( B009 CONSTANT DDRD     ( * DATA DIRECTION REGISTER FOR PORT D
```

```

( B00A CONSTANT PORTE      ( * INPUT PORT E
( B00E CONSTANT CFORC     ( * COMPARE FORCE REGISTER
B00C CONSTANT OC1M       ( * OUTPUT COMPARE 1 ACTION MASK REGISTER
B00D CONSTANT OC1D       ( * OUTPUT COMPARE 1 ACTION DATA REGISTER
B00E CONSTANT TCNT       ( * TIMER COUNTER REGISTER 16 BIT
( B010 CONSTANT TIC1      ( * INPUT CAPTURE 1 REGISTER 16 BIT
( B012 CONSTANT TIC2      ( * INPUT CAPTURE 2 REGISTER 16 BIT
( B014 CONSTANT TIC3      ( * INPUT CAPTURE 3 REGISTER 16 BIT
B016 CONSTANT TOC1       ( * OUTPUT COMPARE 1 REGISTER 16 BIT
B018 CONSTANT TOC2       ( * OUTPUT COMPARE 2 REGISTER 16 BIT
B01A CONSTANT TOC3       ( * OUTPUT COMPARE 3 REGISTER 16 BIT
B01C CONSTANT TOC4       ( * OUTPUT COMPARE 4 REGISTER 16 BIT
( B01E CONSTANT TOC5      ( * OUTPUT COMPARE 5 REGISTER 16 BIT
B020 CONSTANT TCTL1      ( * TIMER CONTROL REGISTER 1
B021 CONSTANT TCTL2      ( * TIMER CONTROL REGISTER 2
B022 CONSTANT TMSK1      ( * TIMER INTERRUPT MASK REGISTER 1
B023 CONSTANT TFLG1      ( * TIMER INTERRUPT FLAG REGISTER 1
B024 CONSTANT TMSK2      ( * TIMER INTERRUPT MASK REGISTER 2
B025 CONSTANT TFLG2      ( * TIMER INTERRUPT FLAG REGISTER 2
B026 CONSTANT PACTL      ( * PULSE ACCUMULATOR CONTROL REGISTER
B027 CONSTANT PACNT      ( * PULSE ACCUMULATOR COUNT REGISTER
B028 CONSTANT SPCR       ( * SERIAL PERIPHERAL INTERFACE CONTROL REGISTER
B029 CONSTANT SPSR       ( * SERIAL PERIPHERAL INTERFACE STATUS REGISTER
B02A CONSTANT SPDR       ( * SERIAL PERIPHERAL INTERFACE DATA REGISTER

( B02B CONSTANT BAUD      ( * SERIAL COMMUNICATIONS INTERFACE BAUD RATE REGISTER
( B02C CONSTANT SCCR1     ( * SERIAL COMMUNICATIONS INTERFACE CONTROL REGISTER 1
( B02D CONSTANT SCCR2     ( * SERIAL COMMUNICATIONS INTERFACE CONTROL REGISTER 2
( B02E CONSTANT SCSR      ( * SERIAL COMMUNICATIONS INTERFACE STATUS REGISTER
( B02F CONSTANT SCDR      ( * SERIAL COMMUNICATIONS INTERFACE DATA REGISTER
B030 CONSTANT ADCTL      ( * ANALOG TO DIGITAL CONVERTER CONTROL REGISTER
B031 CONSTANT ADR1       ( * ANALOG TO DIGITAL CONVERTER RESULT REGISTER 1
B032 CONSTANT ADR2       ( * ANALOG TO DIGITAL CONVERTER RESULT REGISTER 2
B033 CONSTANT ADR3       ( * ANALOG TO DIGITAL CONVERTER RESULT REGISTER 3
B034 CONSTANT ADR4       ( * ANALOG TO DIGITAL CONVERTER RESULT REGISTER 4

( B035 CONSTANT BPROT     (

B039 CONSTANT OPTION     ( * SYSTEM CONFIGURATION OPTIONS
( B03A CONSTANT COPRST    ( * ARM/RESET COP TIMER CIRCUITRY
( B03B CONSTANT PPROG     ( * INTERNAL EEPROM PROGRAMMING CONTROL REGISTER
( B03C CONSTANT HPRIO     ( * HIGHEST PRIORITY I-BIT INT AND MISC
( B03D CONSTANT INITREG   ( * RAM AND I/O MAPPING REGISTER
B03F CONSTANT CONFIG     ( * COP, ROM, AND EEPROM ENABLES

```

```

( B7EC CONSTANT XIRO_LINK
( B7E9 CONSTANT IRQ_LINK
B7E6 CONSTANT RTI_LINK
( B7E3 CONSTANT IC1_LINK
( B7E0 CONSTANT IC2_LINK
( B7DD CONSTANT IC3_LINK
( B7DA CONSTANT OC1_LINK
( B7D7 CONSTANT OC2_LINK
( B7D4 CONSTANT OC3_LINK
B7D1 CONSTANT OC4_LINK
( B7CE CONSTANT OC5_LINK
( B7CB CONSTANT TOI_LINK
( B7C8 CONSTANT PAOVI_LINK
( B7C5 CONSTANT PAII_LINK
( B7C2 CONSTANT SPIE_LINK

```

```

( * *****

```

Bit Manipulations

```

( * useful words for performing bit manipulations
( * these words operate on the number currently on the top of the stack.
HEX
( * MASK-X leaves true if bit x set else false
( : MASK-15 8000 AND ;
( : MASK-14 4000 AND ;

```

```
( : MASK-13      2000 AND ;
( : MASK-12      1000 AND ;
( : MASK-11      0800 AND ;
( : MASK-10      0400 AND ;
( : MASK-9       0200 AND ;
( : MASK-8       0100 AND ;
: MASK-7        0080 AND ;
: MASK-6        0040 AND ;
: MASK-5        0020 AND ;
: MASK-4        0010 AND ;
: MASK-3        0008 AND ;
: MASK-2        0004 AND ;
: MASK-1        0002 AND ;
: MASK-0        0001 AND ;
```

```
( * T-X makes certain bit x is true regardless of current state
```

```
( : T-15        8000 OR ;
( : T-14        4000 OR ;
( : T-13        2000 OR ;
( : T-12        1000 OR ;
( : T-11        0800 OR ;
( : T-10        0400 OR ;
( : T-9         0200 OR ;
( : T-8         0100 OR ;
: T-7          0080 OR ;
: T-6          0040 OR ;
: T-5          0020 OR ;
: T-4          0010 OR ;
: T-3          0008 OR ;
: T-2          0004 OR ;
: T-1          0002 OR ;
: T-0          0001 OR ;
```

```
( * F-X makes certain bit x is false regardless of current state
```

```
( : F-15        7FFF AND ;
( : F-14        BFFF AND ;
( : F-13        DFFF AND ;
( : F-12        EFFF AND ;
( : F-11        F7FF AND ;
( : F-10        FBFF AND ;
( : F-9         FDFD AND ;
( : F-8         FEFF AND ;
: F-7          FF7F AND ;
: F-6          FFBF AND ;
: F-5          FFDF AND ;
: F-4          FFEF AND ;
: F-3          FFF7 AND ;
: F-2          FFFB AND ;
: F-1          FFFD AND ;
: F-0          FPFE AND ;
```

```
( * *****)
```

```
(
(
(          EEPROM Memory
(          -----
```

```
( * the following words deal with the off chip EEPROM
( * where we store the default system dictionary
( * eeprot and eeunprot are used to write protect the EEPROM
( * there are some funny numbers in here which are hard coded
( * and assume that the eeprom starts at $6000, which it does
( * right at the moment.
( * the addresses we want to write to, from the chips point
( * of view, are $5555 and $2AAA. these addresses are not
( * available for our EEPROM, but the bottom 14 bits must be
( * representing these addresses, so we will set the top bit
( * which the eeprom does not even see and write to it at
( * $D555 and $AAAA .
```

```
HEX
```

0-4

```

CODE-SUB EEPROT
  CE C,   D5 C,   55 C,   ( * LDX   #$D555  1ST EPROM ADDRESS
  96 C,   AA C,           ( * LDAA   #$AA    1ST DATA PATTERN
  A7 C,   00 C,           ( * STAA   0,X    SEND AN AA TO $5555
  CE C,   AA C,   AA C,   ( * LDX   #$AAAA  2ND EPROM ADDRESS
  96 C,   55 C,           ( * LDAA   #$55    2ND DATA PATTERN
  A7 C,   00 C,           ( * STAA   0,X    SEND A 55 TO $2AAA
  CE C,   D5 C,   55 C,   ( * LDX   #$D555  3RD EPROM ADDRESS
  96 C,   A0 C,           ( * LDAA   #$A0    3RD DATA PATTERN
  A7 C,   00 C,           ( * STAA   0,X    SEND AN A0 TO $5555
  39 C,           ( * RTS                    RETURN

```

END-CODE

```

CODE-SUB EEUNPROT
  CE C,   D5 C,   55 C,   ( * LDX   #$D555  1ST EPROM ADDRESS
  96 C,   AA C,           ( * LDAA   #$AA    1ST DATA PATTERN
  A7 C,   00 C,           ( * STAA   0,X    SEND AN AA TO $D555
  CE C,   AA C,   AA C,   ( * LDX   #$AAAA  2ND EPROM ADDRESS
  96 C,   55 C,           ( * LDAA   #$55    2ND DATA PATTERN
  A7 C,   00 C,           ( * STAA   0,X    SEND A 55 TO $AAAA
  CE C,   D5 C,   55 C,   ( * LDX   #$D555  3RD EPROM ADDRESS
  96 C,   80 C,           ( * LDAA   #$80    3RD DATA PATTERN
  A7 C,   00 C,           ( * STAA   0,X    SEND AN 80 TO $D555
  CE C,   D5 C,   55 C,   ( * LDX   #$D555  4TH EPROM ADDRESS
  96 C,   AA C,           ( * LDAA   #$AA    4TH DATA PATTERN
  A7 C,   00 C,           ( * STAA   0,X    SEND AN AA TO $D555
  CE C,   AA C,   AA C,   ( * LDX   #$AAAA  5TH EPROM ADDRESS
  96 C,   55 C,           ( * LDAA   #$55    5TH DATA PATTERN
  A7 C,   00 C,           ( * STAA   0,X    SEND A 55 TO $AAAA
  CE C,   D5 C,   55 C,   ( * LDX   #$D555  6TH EPROM ADDRESS
  96 C,   20 C,           ( * LDAA   #$20    6TH DATA PATTERN
  A7 C,   00 C,           ( * STAA   0,X    SEND A 20 TO $D555
  39 C,           ( * RTS                    RETURN

```

END-CODE

```

( EPROM represents where in EEPROM to start storing the dictionary )
( we want to reserve some space for autostart routines etc. so we
( start at 256 bytes above the actual start of the EEPROM or $6100 )

```

```

6000 CONSTANT EPROM
6104 CONSTANT EEDICT-START
200 CONSTANT DICT-START

```

```

: EE-! 2DUP ! BEGIN 2DUP @ = UNTIL DROP DROP ;
: EE-C! 2DUP C! BEGIN 2DUP C@ = UNTIL DROP DROP ;

```

```

( the forth dictionary is stored in eeprom memory, but is run out )
( of ram. at any time the user may use the following word "store" )
( to move the current dictionary to eeprom. this in and of itself )
( is not all that useful. the next word "restore" will move the
( dictionary from eeprom to ram, and restore the state of the
( forth system to exactly where you were when you issued the store)
( command. this is also of limited use, because next time you
( power up the system, restore is not available in ram to to be
( called. another word, "astart!" defined at the end of this file)
( is used to tell the 6811 what word we would like executed on the)
( next restart. "restore" can be used as the autostart routine.
( other words which include restore may also be defined as auto-
( start words, but the routine name is hard coded and this file
( would have to be edited. a detailed discussion of the whole
( autostart process can be found in the new micros manuals.

```

```

( STORE store the current state of the forth machine
( and the user dictionary to external EEPROM

```

```

VARIABLE PROMLOC
VARIABLE PROMCOUNT

```

```

HEX
: STORE
  0 PROMCOUNT !
  EEUNPROT

```

```

EPROM 100 + PROMLOC !
HERE PROMLOC @ EE-!
PROMLOC @ 2+ PROMLOC !
DICT-START PROMLOC @ EE-!
PROMLOC @ 2+ PROMLOC !

CR ." storing user dictionary ... " CR
HERE DICT-START DO
I C@ DUP PROMLOC @ C@ = IF DROP
      ELSE PROMLOC @ EE-C! PROMCOUNT @ 1+ PROMCOUNT !
      THEN
PROMLOC @ 1+ PROMLOC !
LOOP

." storing FORTH variables ... " CR

84 6 DO
I C@ PROMLOC @ EE-C!
PROMLOC @ 1+ PROMLOC !
LOOP

EEPROT
CR ." store complete. " CR
." dictionary length: " HERE 200 - U. CR
." bytes updated: " PROMCOUNT @ U. CR
EEPROT ;

```

```

HEX
: RESTOR

```

```

HEX
( CR ." restoring user dictionary " CR
6104      ( * START ADDR. OF DICTIONARY IN EEPROM
200      ( * START ADDR. OF DICTIONARY IN RAM
6100 @ 6102 @      ( * GET END AND START OF RAM DICT
-      ( * SUBTRACT TO GET DICTIONARY LENGTH
CMOVE      ( * MOVE DICTIONARY

( ." restoring FORTH variables " CR
6100 @      ( * GET THE RAM DICTIONARY END ADDRESS
6102 @      ( * GET THE RAM DICTIONARY START ADDRESS
-      ( * NOW WE'VE GOT THE DICT LENGTH
6104 +      ( * + EEDICT OFFSET = ADDR OF USER AREA IN EEPROM
6 84 CMOVE      ( * MOVE USER AREA

CR ." restore complete " CR
VERSION CR
;

```

```

( *****
(
(          FORTH Dictionary Manipulation
(          -----
(
( these words allow to cut or bridge the dictionary. they will return
( an address and data. if this data is stored under the address, the
( dictionary will recover to the original form

( CUT-DICT must be called with the name's compilation address of the
( first word to be cut off. Attention: also the forth words will not be
( accessible anymore.
( To cut all words below TEST: ' TEST CUT
(                               ^ this is the TICK

: CUT-DICT 2- DUP CR ." ADDRESS: " . DUP @ ." DATA: " . 0 SWAP ! ;

( BRIDGE-DICT is similar to CUT-DICT except it allows to continue
( the dictionary at any point. This can be used to disable access
( to a set of words
( ' WORD2 ' WORD1 BRIDGE-DICT disables all words below WORD2 including WORD1.
( ^-----^--these are TICKS
( Attention: make sure WORD2 is higher in the wordlist (further
( away from TASK) than WORD1

```

```
: BRIDGE-DICT 2- @ SWAP 2- DUP CR ." ADDRESS: " . DUP @ ." DATA: " . ! ;
```

```
( ***** )
(
(           Analog To Digital Converter
( -----
( the 68hc11 includes an 8 channel, multiplexed input successive
( approximation analog to digital converter with sample and hold.
( each conversion is accomplished in 32 e clock cycles, or about
( 16 usec. the converter has two modes of operation, single channel
( mode or multiplexed input mode. in each of these modes the system
( has two modes of operation: single cycle or scanning. in single
( cycle mode a single conversion cycle is performed leaving four
( results. in scanning mode the converter performs continuously,
( overwriting the data in adr1 with data from the fifth conversion,
( and the data in the second with results from the sixth, and so on.
(
( note1 : in either mode a conversion cycle consists of four
( conversions. in single channel mode this means that
( four conversions will be performed on the input
( channel before the conversion complete flag will be
( set, with results being stored in adr1-adr4.
(
( note2 : any writes to the control register, adctl, will
( initiate a conversion cycle. writing to adctl is
( the only way to initiate a conversion. all words
( below which end with "adctl c!" actually start a
( conversion cycle.
(
( note3 : to use the adc the adpu bit in the option register
( must be set. it usually is. to check it use the
( forth word adc-on? defined below, and the word
( adc-on to turn it on if its not.
(
( : ADC-ON?  OPTION C@ MASK-7 ;          ( * true if analog to digital converter
(                                           ( * system is turned on, false otherwise
(
( : ADC-ON   OPTION C@ T-7 OPTION C!    ( * turns converter on.
( 1 0 MS-WAIT ;                        ( * a 100 usec delay is required
(                                           ( * before using the converter.
( : ADC-OFF  OPTION C@ F-7 OPTION C! ;
(
( : SET-SCAN  ADCTL C@ T-5 ADCTL C! ; ( * set continuous conversions
( : SET-NOSCAN ADCTL C@ F-5 ADCTL C! ; ( * set single conversion cycle
(
( : SET-SINGLE ADCTL C@ F-4 ADCTL C! ; ( * SETS SINGLE CHANNEL MODE
(
( : SET-CH0   SET-SINGLE ADCTL C@ F-0 F-1 F-2 F-3 ADCTL C! ;
( : SET-CH1   SET-SINGLE ADCTL C@ T-0 F-1 F-2 F-3 ADCTL C! ;
( : SET-CH2   SET-SINGLE ADCTL C@ F-0 T-1 F-2 F-3 ADCTL C! ;
( : SET-CH3   SET-SINGLE ADCTL C@ T-0 T-1 F-2 F-3 ADCTL C! ;
( : SET-CH4   SET-SINGLE ADCTL C@ F-0 F-1 F-2 T-3 ADCTL C! ;
( : SET-CH5   SET-SINGLE ADCTL C@ T-0 F-1 F-2 T-3 ADCTL C! ;
( : SET-CH6   SET-SINGLE ADCTL C@ F-0 T-1 F-2 T-3 ADCTL C! ;
( : SET-CH7   SET-SINGLE ADCTL C@ T-0 T-1 F-2 T-3 ADCTL C! ;
(
( : SET-MULT  ADCTL C@ T-4 ADCTL C! ; ( * PUTS ADC IN MULTIPLEXED INPUT MODE
(
( : SET-GRP1  SET-MULT ADCTL C@ F-3 F-2 ADCTL C! ; ( * USE INPUT GROUP1, CH0-3
( : SET-GRP2  SET-MULT ADCTL C@ F-3 F-2 ADCTL C! ; ( * USE INPUT GROUP2, CH4-7
(
( : ADC-DONE? ADCTL C@ MASK-0 ; ( * TRUE IF A CONVERSION SEQUENCE IS COMPLETE
```



```

19 C, 09 C, ( DEY
18 C, ED C, 00 C, ( STD 0,Y
FC C, TIME , ( LDD TIME
18 C, 09 C, ( DEY
18 C, 09 C, ( DEY
18 C, ED C, 00 C, ( STD 0,Y
0E C, ( CLI
39 C, ( RTS

```

END-CODE

```

CODE-SUB TIME!
0F C, ( SEI
18 C, EC C, 00 C, ( LDD 0,Y
18 C, 08 C, ( INY
18 C, 08 C, ( INY
FD C, TIME , ( STD TIME
18 C, EC C, 00 C, ( LDD 0,Y
18 C, 08 C, ( INY
18 C, 08 C, ( INY
FD C, TIME 2+ , ( STD TIME+2
0E C, ( CLI
39 C, ( RTS

```

END-CODE

(the following are used by the exposure routines defined near the end of this file.
(they are defined here so as to be accessible to the timer routine immediately below.

```

2VARIABLE EXP_LEFT
VARIABLE EXPOSING

```

(assembler routines to fetch and store EXP_LEFT are required to avoid collisions
(with the timer routine itself, without interrupting it significantly.

HEX

```

CODE-SUB EXP_LEFT@
0F C, ( SEI
FC C, EXP_LEFT 2+ , ( LDD EXP_LEFT+2
18 C, 09 C, ( DEY
18 C, 09 C, ( DEY
18 C, ED C, 00 C, ( STD 0,Y
FC C, EXP_LEFT , ( LDD EXP_LEFT
18 C, 09 C, ( DEY
18 C, 09 C, ( DEY
18 C, ED C, 00 C, ( STD 0,Y
0E C, ( CLI
39 C, ( RTS

```

END-CODE

```

CODE-SUB EXP_LEFT!
0F C, ( SEI
18 C, EC C, 00 C, ( LDD 0,Y
18 C, 08 C, ( INY
18 C, 08 C, ( INY
FD C, EXP_LEFT , ( STD EXP_LEFT
18 C, EC C, 00 C, ( LDD 0,Y
18 C, 08 C, ( INY
18 C, 08 C, ( INY
FD C, EXP_LEFT 2+ , ( STD EXP_LEFT+2
0E C, ( CLI
39 C, ( RTS

```

END-CODE

(all this routine needs to do is increment the time and set the next interrupt
(to occur one millisecond from now. Someday soon we might want to check if it's
(time to perform housekeeping tasks and do some safety checks. Is the backplate
(too hot? Is the CCD too cold? or too hot?? Stuff like that.

```
( now we're attempting to do background exposures by decrementing a counter in this
( interrupt service routine. We can check it at any time to see if the exposure is
( done. we can do other things while the exposure proceeds.
```

```
( we are using the output compare number 4 because 1 is reserved for other uses in the
( 6811, and the pins associated with oc2 and oc3 are used for shutter control.
```

```
HEX
```

```
CODE TIMER
```

```
0F C,      (      sei          *  disable other interrupts
36 C, 10 C, (      ldaa  #$10
B7 C, TFLG1, (      staa  tflg1 *  clear interrupt flag
FC C, TIME 2+, (      ldd   time+2 *  fetch the lower half of time
C3 C, 1, (      addd  #1 *  increment it,
FD C, TIME 2+, (      std   time+2 *  store it,
24 C, 09 C, (      bcc   nocarry *  if it didn't overflow, then don't
FC C, TIME, (      ldd   time *  fetch the upper half
C3 C, 0001, (      addd  #1 *  increment it,
FD C, TIME, (      std   time *  and store it,
( nocarry *  just continue.

FC C, TOC4, (      ldd   toc4 *  fetch the last compare value
C3 C, 07D0, (      addd  #2000 *  add 2000 to it
FD C, TOC4, (      std   toc4 *  store as next compare value

B6 C, EXPOSING 1+, (      ldaa  EXPOSING *  fetch exposure flag
27 C, 14 C, (      beq   nexp *  if not exposing leave
FC C, EXP_LEFT 2+, (      ldd   expleft+2 *  fetch lower half of counter
83 C, 1, (      subd  #1 *  subtract one from it
FD C, EXP_LEFT 2+, (      std   expleft+2 *  store it
24 C, 09 C, (      bcc   nexp *  if no borrow, then don't
FC C, EXP_LEFT, (      ldd   exp_left *  fetch top half
83 C, 1, (      subd  #1 *  decrement it
FD C, EXP_LEFT, (      STD   EXP_LEFT *  and store it
( nexp

FC C, EXP_LEFT, (      ldd   exp_left *  If top half of exp left
26 C, 0E C, (      bne   not_done *  is not equal to zero,
FC C, EXP_LEFT 2+, (      ldd   exp_left+2 *  or the bottom half is not,
26 C, 09 C, (      bne   not_done *  were not done.
( *  If we are done,
7F C, EXPOSING 1+, (      clr   exposing *  clear exposure flag,
CE C, PORTA, (      ldx  #porta *  point at port A
1D C, 0 C, 60 C, (      bclr  0,x #560 *  and close the shutters.
( not_done

0E C,      (      cli          *  enable other interrupts
3B C,      (      rti          *  and leave.
```

```
END-CODE
```

```
( install the jump to our TIMER function at the appropriate address )
```

```
HEX
```

```
CODE-SUB CLEAR-CC-MASKS
```

```
86 C, 00 C,      ( LDAA #0 )
06 C,           ( TAP )
39 C,           ( RTS )
```

```
END-CODE
```

```
: STOP-TIMER      0 TMSK1 C! ( TURN OFF THE INTERRUPT )
                  0 TCTL1 C! ( STOP THE COMPARE OUTPUT ON PA4 ) ;
```

```
: INSTALL-TIMER
```

```
STOP-TIMER
7E DUP OC4_LINK C@ = IF DROP ELSE OC4_LINK EEC! THEN
[ ' TIMER @ >> FF AND ] LITERAL
DUP OC4_LINK 1+ C@ = IF DROP ELSE OC4_LINK 1+ EEC! THEN
[ ' TIMER @ FF AND ] LITERAL
DUP OC4_LINK 2+ C@ = IF DROP ELSE OC4_LINK 2+ EEC! THEN ;
```

```

: INIT-TIMER
  STOP-TIMER
  CLEAR-CC-MASKS
  0 0 TIME!           ( ZERO THE TIME VARIABLE )
  3000 TOC4 !        ( START WITH COMPARE AT TCNT = $8000 )
    04 TCTL1 C!     ( SET TO TOGGLE PA4 ON EACH COMPARE )
    00 OC1M C!
    00 OC1D C!       ( OUTPUT COMPARE 1 DOES NOTHING )
    FF TFLG1 C! ;   ( CLEAR ANY AND ALL COMPARE FLAGS )

```

```

HEX
: START-TIMER
  STOP-TIMER
  INIT-TIMER
  10 TMSK1 C! ;     ( ENABLE THE INTERRUPT )

```

```

(
  Time Measurement
(

```

```

( MS-WAIT expects a 32 bit number on the stack.
( It waits that many milliseconds.

```

```

: MS-WAIT ( D - )
  2DUP 0= SWAP 0= AND NOT
  IF TIME@ D+ BEGIN 2DUP TIME@ DU< UNTIL
  THEN 2DROP
  ;

```

```

( M-WAIT expects a 16 BIT number on the stack.
( it waits that many milliseconds.

```

```

: M-WAIT ( W - )
  0 MS-WAIT ;

```

```

( TEMPERATURE CONTROL LOOP SOFTWARE

```

```

( the general scheme is as follows ...
( a real time interrupt is initiated which interrupts the 6811 every 32.77 ms
( each time the interrupt is received, the 6811 fetches and increments the
( tmp_cnt variable storing the result and comparing it to the max_tmp_cnt
( variable. if less than, the 6811 simply returns, if equal or greater, which
( should not occur, the 6811 checks the temperature against the desired temp.
( if less than desired, the 6811 fetches the heater value and increments it. if
( the measured temp is greater than the desired, the 6811 decrements the heater value.
( if the measured temp equals the desired, no action is taken.

```

```

HEX
VARIABLE ACT_TMP      ( actual temp, most recently measured temperature
VARIABLE DES_TMP      ( desired temperature
VARIABLE TMP_CNT      ( temperature loop counter
VARIABLE MAX_CNT      ( how many times to loop before checking temperature
VARIABLE DAC_VAL      ( value most recently written to the temp control DAC
VARIABLE DAC_INC      ( value to add or subtract from dac_val when nec.
B200 CONSTANT TMP_DAC ( location of temperature control DAC

```

```

CODE-SUB TEMP-CTL
0F C,      (      sei           * mask other interrupts
36 C, 40 C, (      ldaa  #$40
B7 C, B025 , (      staa  $b025      * clear RTI flag

B6 C, TMP_CNT 1+ , (      ldaa  tmp_cnt    * fetch current count into acc a
4C C,      (      inca           * increment it
E7 C, TMP_CNT 1+ , (      staa  tmp_cnt    * and store a copy in tmp_cnt
B1 C, MAX_CNT 1+ , (      cmpa  max_cnt    * then compare to max count
25 C, 70 C,      (      blo   exit      * if max > current, exit

```

```

4F C,          (      clra          * zero accumulator a
B7 C, TMP_CNT 1+ , (      staa  tmp_cnt      * and reset counter

B6 C, B039 ,   (      ldaa  $b039   * get option register into acca
84 C, BF C,   (      anda  #$bf    * make sure csel bit is clear
8A C, 80 C,   (      oraa  #$80    * make sure adpu bit is set
B7 C, B039 ,   (      staa  $b039   * store that control register
86 C, 02 C,   (      ldaa  #2     *
B7 C, B030 ,   (      staa  $b030   * start 4 conversions on channel two

( adwait
B6 C, B030 ,   (      ldaa  $b030   * get adc control reg into acc a
84 C, 80 C,   (      anda  #$80    * conversion complete?
27 C, F9 C,   (      beq   adwait   * if not set, go check again

18 C, 3C C,   (      pshy          * push y onto stack
18 C, CE C, 0 , (      ldy   #0           * clear reg y
F6 C, B031 ,   (      ldab  $b031   * get 1st adc result
18 C, 3A C,   (      aby          * add b to y
F6 C, B032 ,   (      ldab  $b032   * get 2nd adc result
18 C, 3A C,   (      aby          * add b to y
F6 C, B033 ,   (      ldab  $b033   * get 3rd adc result
18 C, 3A C,   (      aby          * add b to y
F6 C, B034 ,   (      ldab  $b034   * get 4th adc result
18 C, 3A C,   (      aby          * add b to y
18 C, 9F C,   (      xgdy          * move y to double acc
18 C, 38 C,   (      puly          * restore reg y
04 C,         (      lsr   #2     * divide acc by 2
04 C,         (      lsr   #2     * divide by 2 again, accd = avg

F7 C, ACT_TMP 1+ , (      stab  act_tmp      * store as current temperature

F1 C, DES_TMP 1+ , (      cmpb  des_tmp      * compare to desired
27 C, 2E C,   (      beq   exit      * if the temp is right, just leave

22 C, 16 C,   (      bhi   its_hi     * if actual > desired, it's too high

( * if the temperature is too low and the current dac setting
( * is more than ff - dac_inc, then set the dac to $ff.

( its_lo
86 C, FF C,   (      ldaa  #$ff
B0 C, DAC_INC 1+ , (      suba  dac_inc
B1 C, DAC_VAL 1+ , (      cmpa  dac_val
22 C, 04 C,   (      bhi   inc_it      * if dac_val < {ff - dac_inc} , increment
86 C, FF C,   (      ldaa  #$ff
20 C, 18 C,   (      bra   store_it   * otherwise, set dac_val to $ff

( * if current dac vcal is less than ff - dac_inc , then
( * increment the dac_val by dac_inc

( inc_it
B6 C, DAC_VAL 1+ , (      ldaa  dac_val      * get current dac value
BB C, DAC_INC 1+ , (      adda  dac_inc      * increment dac_val by dac_inc
20 C, 10 C,   (      bra   store_it   * store it and leave

( * if the temperature is too high and the current dac setting
( * is greater than increment value, then decrement dac val
( * by increment value, otherwise decrement by one.

( its_hi
B6 C, DAC_VAL 1+ , (      ldaa  dac_val
27 C, 11 C,   (      beq   exit      * if dac_val = 0, just leave
B1 C, DAC_INC 1+ , (      cmpa  dac_inc      * if dac_val > dac_inc
22 C, 03 C,   (      bhi   dec_lots    * ... dec dac_val by dac_inc
4A C,         (      deca          * decrement by one
20 C, 03 C,   (      bra   store_it   * store it and leave

( dec_lots
B0 C, DAC_INC 1+ , (      suba  dac_inc      * decrement dac_val by dac_inc

( store_it
B7 C, DAC_VAL 1+ , (      staa  dac_val      * store the value for future reference

```

```

B7 C, TMP_DAC 1+ , (      staa tmp_dac      * write it to the dac
                   {
0E C,              { exit cli          * unmask interrupts
3B C,              {      rti
END-CODE          ( end of TEMP-CTL definition )

```

DECIMAL

```

: INIT-TEMP      0 ACT_TMP ! 150 DES_TMP ! 0 TMP_CNT ! 5 MAX_CNT !
                 0 DAC_VAL ! 10 DAC_INC ! 0 TMP_DAC ! ;

```

HEX

```

( the address to which the 6811 will jump when it gets the interrupt )
( install the jump to our function at that address )

```

```

: STOP-TEMP-CTL
  TMSK2 C@ F-6 TMSK2 C! ;

: START-TEMP-CTL
  STOP-TEMP-CTL
  CLEAR-CC-MASKS
  INIT-TEMP
  PACTL C@ T-1 T-0 PACTL C! ( set rti to 32.77 milliseconds )
  TMSK2 C@ T-6 TMSK2 C! ( enable the interrupt )
  ;

: INSTALL-TEMP-CTL
  STOP-TEMP-CTL
  STOP-TIMER
  7E DUP RTI_LINK C@ = IF DROP ELSE RTI_LINK EEC! THEN
  [ ' TEMP-CTL @ >< FF AND ] LITERAL
  DUP RTI_LINK 1+ C@ = IF DROP ELSE RTI_LINK 1+ EEC! THEN
  [ ' TEMP-CTL @ FF AND ] LITERAL
  DUP RTI_LINK 2+ C@ = IF DROP ELSE RTI_LINK 2+ EEC! THEN
  ;

```

DECIMAL

```

( : CHK-TMP      ACT_TMP @ 200 - ." TEMP = " . ." DEG C " CR ;
( : WATCH-TMP   BEGIN CHK-TMP 2000 0 MS-WAIT ?TERMINAL UNTIL ;

( : CHK-CNT     TMP_CNT @ U. CR ;
( : WATCH-CNT  BEGIN CHK-CNT 2000 0 MS-WAIT ?TERMINAL UNTIL ;

( : CHK-DAC     DAC_VAL @ U. CR ;
( : WATCH-DAC  BEGIN CHK-DAC 2000 0 MS-WAIT ?TERMINAL UNTIL ;

( : CHK-RTI     CR
(               ACT_TMP @ ." ACT_TMP = " U. CR
(               DES_TMP @ ." DES_TMP = " U. CR
(               TMP_CNT @ ." TMP_CNT = " U. CR
(               MAX_CNT @ ." MAX_CNT = " U. CR
(               DAC_VAL @ ." DAC_VAL = " U. CR ;
(

: TEMP!        200 + DES_TMP ! ;

: TEMP?        ACT_TMP @ 200 - . CR ;

: CCD-TEMP?    ACT_TMP @ 200 - ." CCD temperature = " . ." deg C" CR ;

: TEMP-DIFF?   DES_TMP @ ACT_TMP @ - U. CR ;

```

```

( * *****)
( DSP RELATED FUNCTIONS
(
( DEFINITIONS OF CONSTANTS REPRESENTING DSP HOST PORT LOCATIONS

HEX

C000 CONSTANT ICR      ( * INTERRUPT CONTROL REGISTER
C001 CONSTANT CVR      ( * COMMAND VECTOR REGISTER
C002 CONSTANT ISR      ( * INTERRUPT STATUS REGISTER
C003 CONSTANT IVR      ( * INTERRUPT VECTOR REGISTER
( * C004 NOT USED
C005 CONSTANT RXH      ( * RECEIVE REGISTER HIGH BYTE
C006 CONSTANT RXM      ( * RECEIVE REGISTER MIDDLE BYTE
C007 CONSTANT RXL      ( * RECEIVE REGISTER LOW BYTE

C005 CONSTANT TXH      ( * TRANSMIT REGISTER HIGH BYTE
C006 CONSTANT TXM      ( * TRANSMIT REGISTER MIDDLE BYTE
C007 CONSTANT TXL      ( * TRANSMIT REGISTER LOW BYTE

( * SHOW-REGS simply fetches and prints the values in the DSP registers
(      this is useful for debugging, but serves no real purpose
( : SHOW-REGS
(      CR
(      ICR C@ ." ICR = " . CR
(      ISR C@ ." ISR = " . CR
(      CVR C@ ." CVR = " . CR
(      RXH C@ ." RXH = " . CR
(      RXM C@ ." RXM = " . CR
(      RXL C@ ." RXL = " . CR ;
(

( DSPCTL IS THE DSP CONTROL LATCH WHICH THE 6811 MAY WRITE TO IN ORDER
(      TO AFFECT THE DSP'S OPERATION.

C100 CONSTANT DSPCTL

( : DSP-RESET
(      FE DSPCTL C! ;          ( * DSP RESET LINE LOW
(      FF DSPCTL C! ;          ( * DSP RESET LINE HIGH

( : DSP-IRQA
(      FB DSPCTL C! ;          ( * IRQA LOW
(      FF DSPCTL C! ;          ( * IRQA HIGH
(

( : DSP-IRQB
(      FD DSPCTL C! ;          ( * IRQB LOW
(      FF DSPCTL C! ;          ( * IRQB HIGH
(

: DSP-BOOT
FF DSPCTL C! ;          ( * EVERYTHING HIGH
FC DSPCTL C! ;          ( * DSP RESET LOW  IRQB LOW
FD DSPCTL C! ;          ( * DSP RESET HIGH IRQB LOW
FF DSPCTL C! ;          ( * DSP RESET HIGH IRQB HIGH

: CLR-TXH-TXM  00 DUP TXH C! TXM C! ;

( ROUTINES FOR INTERPRETING THE HOST REGISTERS
: RXDF?      ISR C@ MASK-0 ;      ( * LEAVES TRUE IF RXDF BIT SET
: TXDE?      ISR C@ MASK-1 ;      ( * LEAVES TRUE IF TXDE BIT SET
( : HF2?      ISR C@ MASK-3 ;      ( * TRUE IF HF2 SET
( : HF3?      ISR C@ MASK-4 ;      ( * TRUE IF HF3 SET
( : HREQ?      ISR C@ MASK-7 ;      ( * TRUE IF DSP IS ASSERTING HREQ
: T-HF0      ICR C@ T-3 ICR C! ;   ( * SETS HF0 TO TRUE
: F-HF0      ICR C@ F-3 ICR C! ;   ( * SETS HF0 TO FALSE

```

```

: T-HF1      ICR C@ T-4 ICR C! ;    ( * SETS HF1 TO TRUE
: F-HF1      ICR C@ F-4 ICR C! ;    ( * SETS HF1 TO FALSE

```

```

( when the DSP is executing a host command, it sets hf2 in the ISR to a 1 )

```

```

( DSP-BUSY checks that flag
: DSP-BUSY?  ISR C@ MASK-3 ;

```

```

( DSP-WAIT  waits until the DSP is no longer busy )
: DSP-WAIT  BEGIN ISR C@ MASK-3 0= UNTIL ;

```

```
VARIABLE DSP_OK
```

```

: HV! ( VECTOR# - )
      DUP 20 < IF T-7 CVR C!
          ELSE ." ERROR : command vector too large "
          THEN ;

```

```

( DSP8@, DSP16@, DSP24@, DSP8!, DSP16!, and DSP24!

```

```

( these words are used to fetch and store data from and to the DSP
( some checking is performed to see that the data registers are ready

```

```

( read from DSP - DSP8@  fetches 8 bits from rxl reg
(                 DSP16@ fetches 16 bits from DSP rxm and rxl registers
(                 DSP24@ fetches 24 bits from DSP rxh, rxm, and axl registers

```

```

( write to DSP - DSP8!  write 8 bits to DSP txl register
(                DSP16! write 16 bits to DSP txm and txl registers
(                DSP24! write 24 bits to DSP txh, txm and txl registers

```

```
HEX
```

```

: REPORT-DSP-ERROR
  ." ERROR : DSP not responding " CR ;

```

```

: WAIT-DSP@
  FALSE DSP_OK !    ( guilty until proven innocent )
  200 0 DO RXDF?
    IF TRUE DSP_OK ! LEAVE
    THEN
  LOOP ;

```

```

: DSP8@ ( - B )
  WAIT-DSP@
  DSP_OK @ IF RXL C@
    ELSE REPORT-DSP-ERROR
    THEN ;

```

```

: DSP16@ ( - W )
  WAIT-DSP@
  DSP_OK @ IF RXM C@ 100 * RXL C@ +
    ELSE REPORT-DSP-ERROR
    THEN ;

```

```

( : DSP24@ ( - D )
(   WAIT-DSP@
(     DSP_OK @ IF RXH C@ RXM C@ 100 * RXL C@ + SWAP ;
(       ELSE REPORT-DSP-ERROR
(         THEN ;

```

```

: WAIT-DSP!
  FALSE DSP_OK !    ( guilty until proven innocent )

```

```

200 0
DO TXDE?
  IF TRUE DSP_OK ! LEAVE
  THEN
  LOOP ;

```

```

: DSP8! ( B - )
  WAIT-DSP!
  DSP_OK @ IF 0 TXH C! 0 TXM C! TXL C!
            ELSE REPORT-DSP-ERROR
            THEN ;

```

```

: DSP16! ( W - )
  WAIT-DSP!
  DSP_OK @ IF 0 TXH C! DUP >< TXM C! TXL C!
            ELSE REPORT-DSP-ERROR
            THEN ;

```

```

( : DSP24! ( D - )
  (   WAIT-DSP!
  (   DSP_OK @ IF TXH C! DUP >< TXM C! TXL C! ;
  (   ELSE REPORT-DSP-ERROR
  (   THEN ;

```

```

( ***** )
(
(           DSP Software Support
(           -----
(
( routines for getting files from the pc to the dsp
(
( dsp code is downloaded from the host computer and stored in eeprom.
( two buffers are stored, one is 256 bytes long and the other is of variable
( length. the 100 byte buffer is typically used to store a bootstrap program
( to load the dsp's external program memory from the larger buffer.
(

```

HEX

```

( XLATE ACCEPTS AN ASCII VALUE ON THE STACK AND TRANSLATES IT INTO THE HEX VALUE)
( WHICH IT REPRESENTS. FOR INSTANCE: THE VALUE 41 WILL BE REPLACED BY 50A)

```

```

: XLATE ( ASCII-VALUE - HEX-VALUE
  DUP DUP
  2F > SWAP 40 < AND
  IF 30 -
  ELSE DUP DUP 40 > SWAP 47 < AND
  IF 37 -
  ELSE
  CR ." ERROR : DNLD failure, illegal character recieved " CR
  THEN
  THEN ;

```

```

HEX
VARIABLE END-ADDR
VARIABLE BOT-BUF
VARIABLE TOP-BUF
HERE END-ADDR !
2 ALLOT
HERE BOT-BUF !
1200 ALLOT
HERE TOP-BUF !

```

```

: DNLD
  CR ." expecting DSP code file in MOTOROLA *.LOD format ... " CR
  KEY DUP EMIT DUP 0D = IF 0A EMIT THEN ( drop first _
  BEGIN
  KEY DUP EMIT DUP 0D = IF 0A EMIT THEN 5F = ( drop characters until _ )

```

```

UNTIL
BEGIN
  KEY DUP EMIT DUP OD = IF OA EMIT THEN OD = ( drop chars to line end )
UNTIL

TOP-BUF @ BOT-BUF @
DO KEY DUP 5F = ( * IF CHAR = "_" )
IF EMIT I 1- END-ADDR ! LEAVE ( * STORE COUNT AS END ADDR LEAVE )
ELSE DUP 20 = ( * IF ITS A SPACE
  IF EMIT R> 1- >R ( * PRINT IT BUT DON'T COUNT IT
  ELSE DUP OD = ( * IF ITS A CR
    IF EMIT OA EMIT R> 1- >R ( * DO CR-LF DON'T COUNT IT
    ELSE DUP XLATE 10 * SWAP EMIT ( * ELSE XLATE MULT BY $10
      KEY DUP XLATE ROT + I C! EMIT
    THEN
  THEN
  THEN
LOOP
BEGIN ( drop rest to prevent that
  KEY DUP EMIT DUP OD = IF OA EMIT THEN OD = ( it will be executed
UNTIL
CR ." Download completed " ;

```

```

( DSP-DUMP DUMPS DSP CODE FROM ONE OF THE BUFFERS INTO THE DSP DATA REGISTERS
( IT IS ASSUMED THAT THE DSP IS EXPECTING THE DATA.
: DSP-DUMP
  BOT-BUF @ 2- END-ADDR @ =
  IF CR ." no DSP code in buffer -- DSP-DUMP aborted " CR
  ELSE
    END-ADDR @ BOT-BUF @ DO
    I C@ TXH C!
    I 1+ C@ TXM C!
    I 2+ C@ TXL C!
    3 +LOOP
    T-HF0
    10 0 MS-WAIT F-HF0 10 0 MS-WAIT
  THEN ;

```

```

( VARIABLE CHK_SUM
( : CHKSUM
  BOT-BUF @ 2- END-ADDR @ =
  IF ." no DSP code in buffer "
  ELSE
    0 CHK_SUM !
    END-ADDR @ 1+
    BOT-BUF @
    DO I C@ CHK_SUM @ + CHK_SUM !
    LOOP
    ." DSP chksum: " CHK_SUM @ U.
  THEN ;
(
(
( *****
(
( Configuration Tables
(
( All CCD format parameters, CAM voltages, and timing information is
( stored in arrays. Access to these array elements is provided via
( either the array index or parameter name.
(
( The following word, TABLE, is used to define a new state table.
( It accepts the length of the array on the stac. This length is in words.
(
( Items in that table may then be accessed in the following manner :
(
( " N TABLE_NAME @ " will return the value of the nth item in the table
(

```


Typically,

$$\text{prescan} + (\text{underscan} * \text{binning factor}) = \text{serial extension length}$$

and

$$\text{origin} + (\text{binning factor} * \text{read dimension}) + \text{postscan} \geq \text{CCD serial dimension},$$

but this convention is not enforced. Not all users will care to use the underscan and overscan features. In this case the prescan, underscan, and overscan parameters may be set to zero and the following equation will be used.

$$\text{origin} + (\text{binning factor} * \text{read dimension}) + \text{postscan} = \text{total CCD serial length}$$

No effort is made to enforce this convention. It is simply that, a convenient convention.

- 0 CCD_SER CCD Serial Dimension :
- This parameter represents the total length of the CCD serial register. This includes any pixels in the serial extensions on either end of the register. This parameter is used by the DSP as the number of pixels to discard when it clears the serial register. It is irrelevant to those rows which are actually read.
- 1 BIN_SER Serial Binning Factor :
- The serial binning factor is the number of pixels in the serial register which will be shifted for each pixel read. By binning pixels the image resolution is sacrificed for higher signal to noise ratio. In low light applications or where the image data is one dimensional this may be used to great benefit.
- 2 PRE_SER Serial Prescan
- The serial prescan is the number of pixels to discard before performing the serial underscan. The prescan is in units of physical pixels and is not affected by the binning factor. No image data is produced.
- 3 UNDER_SER Serial Under Scan
- The serial undrescan is the number of pixels to read after the serial prescan. This parameter represents the number of binned pixels to read. One data point is produced for each unit of under scan. Binning is performed.
- 4 ORG_SER Serial Read Origin
- This number represents the number of pixels to be discarded after performing the serial under scan. This parameter is in units of physical pixels and is unaffected by the binning factor. No image data is produced.
- 5 READ_SER Serial Read Dimension
- The value of this parameter represents the number of pixels to read after the serial origin and before the postscan. These pixels are represented in units of binned pixels. One data point is acquired for each unit of read dimension. Binning is performed.
- 6 POST_SER Serial Postscan
- The value of this parameter represents the number of pixels to discard after the read is performed. These pixels are represented in units of physical pixels and are not affected by binning factor. No image data is produced.
- 7 OVER_SER Serial Overscan
- The value of this parameter represents the number of data points to be taken after the postscan is performed. These pixels are represented in units of binned pixels. One data point is acquired for each unit of serial overscan. Binning is performed.

Parallel Read Parameters

The parallel register is read out based on a set of format parameters which imply the following geometry:

```
<<< |-----|-----|-----|-----|
      origin   read dimension   postscan   overscan
```

The readout operations are performed from left to right. The rows in the origin and post scan are discarded. The rows in the the read dimension and the overscan are read off the CCD imager. The discarded rows are in units of physical rows and are not affected by the binning factor. The others are read off the chip and binning is performed. The same binning factor is used on the rows in the read dimension and overscan.

- 8 CCD_PAR CCD Parallel Dimension
This parameter represents the total length of the CCD parallel register. This parameter is it clears the parallel register. It is irrelevant to those rows which are actually read.
- 9 BIN_PAR Parallel Binning Factor
The parallel binning factor is the number of rows in the parallel register which will be shifted for each row read. By binning rows the image resolution is sacrificed for higher signal to noise ratio. In low light applications or where the image data is one dimensional this may be used to great benefit.
- 10 ORG_PAR Parallel Read Origin
This number represents the number of rows to be discarded before performing the parallel read. This parameter is in units of physical rows and is unaffected by the binning factor. No image data is produced.
- 11 READ_PAR Parallel Read Dimension
The value of this parameter represents the number of rows to read after the parallel origin and before the postscan. These rows are represented in units of binned rows. Binning is performed. Image data is produced.
- 12 POST_PAR Parallel Postscan
The value of this parameter represents the number of rows to discard after the read is performed. These rows are represented in units of physical rows and are not affected by binning factor. No image data is produced.
- 13 OVER_PAR Parallel Overscan
The value of this parameter represents the number of rows to be read after the postscan is performed. These rows are represented in units of binned rows. Binning is performed. Image data is produced.
- 14 PAR_DELAY Parallel Clock Delay Time

Exposure Parameters

Two parameters are used when opening and closing the shutter. Since the shutter takes a certain amount of time to open, and it would be undesirable to begin to time the exposure or read out the CCD before the shutter motion had stopped, these two parameters are provided to allow the user to set the delay which will take place after the camera controller opens or closes the shutter. The values are stored as 16 bit numbers representing milliseconds. Delay times may therefore

vary from 0 to 65535 seconds. If your shutter takes longer than this to open or close, you will need to write custom open and close routines.

20 ODELAY time, in milliseconds, required for the shutter to open
21 CDELAY time, in milliseconds, required for the shutter to close

The exposure time for image acquisitions is stored as a 32 bit value in two 16 bit parameters. The value represents the exposure time in milliseconds. The exposure time may therefore be set from 0 milliseconds to just over 7 weeks. This should be sufficient to cover most applications.

22 EXP_TIME_LO lower 16 bits of 32 bit exposure time
23 EXP_TIME_HI higher 16 bits of 32 bit exposure time

Certain camera commands will clear the CCD as part of thier operation. These commands are typically high level image acquisition commands. The number of times that the CCD is cleared in these circumstances is controlled by the value of the following parameter. A typical value for this parameter is two, but it may range from 0 to 65535. A value of zero may be of use under some circumstances where it is not desirable to clear the imager at all. A value of 1 is acceptable under most conditions where a cleared CCD is desired. The default value typically assigned is 2 is conservative. Values greater than 2 are likely to be useful only under unusual circumstances.

24 NUM_CLEARS number of times to clear CCD per clear cycle

Acquisition Sequence Parameters

The camera may operate in a sequenced acquisition mode. Each high level image acquisition command actually acquires a sequence of images based on the values of these parameters. Continuous clearing of the CCD may be performed between images. Clock recombination anti-blooming may be performed during exposures. The camera may acquire images in a frame transfer mode. The camera readout may be performed at either of two speeds.

25 NUM_IMAGES number of images to acquire per image acquisition cycle

26 IM_DELAY_LO lower 16 bits of 32 bit delay between images in cycle
27 IM_DELAY_HI higher 16 bits of 32 bit delay between images in cycle

The default values of 1 for NUM_IMAGES and 0 for both IM_DELAY_HI and IM_DELAY_LO will produce 1 image per high level command with no additional delay.

30 CCLEAR Continuous Clear Flag
The value of this parameter determines whether the camera will perform continuous clearing of the CCD between exposures.

0 = continuous clearing disabled
1 = continuous clearing enabled

31 ANTI-BLOOM Clock Recombination Anti-Blooming Flag
The value of this parameter will determine whether the camera will perform clock recombination anti-blooming during the exposures.

0 = anti-blooming disabled
1 = anti-blooming enabled

32 ARCH CCD architecture
The value of this parameter determines whether the camera will perform a 'shift image to storage' operation before reading the image off the CCD.

0 = full frame
1 = frame transfer

```
( 33 SPEED          Camera Readout Speed
(
( The value of this parameter determines at what speed the
( camera will readout the image data. The actual speed of
( the two options will depend on camera hardware and software
( configuration.
(
( 0 = slow
( 1 = fast
(
(
```

DECIMAL

64 TABLE PARAM

```
( parameter name definitions
( we can access the variables either by index or name
DECIMAL
( parameters 0 -> 29 are shared with the ATC5 cameras )

: CCD_SER      0 PARAM ; ( total length of serial register, including extension
: BIN_SER      1 PARAM ; ( serial binning factor
: PRE_SER      2 PARAM ;
: UNDER_SER   3 PARAM ;
: ORG_SER      4 PARAM ; ( serial read origin, should include extension
: READ_SER     5 PARAM ; ( serial read dimension
: POST_SER     6 PARAM ; ( serial postscan, pixels to discard after the read
: OVER_SER     7 PARAM ; ( serial overscan

: CCD_PAR      8 PARAM ; ( total length of parallel register
: BIN_PAR      9 PARAM ; ( parallel binning factor
: ORG_PAR     10 PARAM ; ( parallel read origin
: READ_PAR    11 PARAM ; ( parallel read dimension
: POST_PAR    12 PARAM ; ( parallel postscan, rows to discard after the read
: OVER_PAR    13 PARAM ; ( parallel overscan
( : PAR_DELAY  14 PARAM ; ( parallel clock delay time

( parameter locations 15 through 19 unused for now )

: ODELAY      20 PARAM ; ( time, in milliseconds, required for the shutter to open
: CDELAY      21 PARAM ; ( time, in milliseconds, required for the shutter to close

: EXP_TIME_LO 22 PARAM ; ( exposure time as a double word for exposure and integrate
: EXP_TIME_HI 23 PARAM ;

: NUM_CLEARS  24 PARAM ; ( number of clears to perform when clear is called
: NUM_IMAGES  25 PARAM ;

: IM_DELAY_LO 26 PARAM ; ( delay between multiple image acquisitions, as a double word )
: IM_DELAY_HI 27 PARAM ;

( parameter locations 28 through 29 unused for now )

: CCLEAR      30 PARAM ; ( continuous clear flag
: ANTI-BLOOM  31 PARAM ; ( clock recombination anti-blooming flag

: ARCH        32 PARAM ;
: SPEED       33 PARAM ;

( : SL_DEL1    34 PARAM ; ( slow readout delay 1 from shift to summing well )
( : SL_DEL2    35 PARAM ; ( slow read delay 2 from summing well to adc start pulse )

( params > 50 are specific to the camera this software is installed in
( there are currently no ATC5 specific parameters )
( AIS camera specific parameters )
```

```
( note : serial direction control is disabled as the camera has been limited to only
one port on either side. )
```

```
: CAM0_PDIR    51 PARAM ; ( parallel direction for CAM0 0 = forwards, 1 = backwards
```

```
: CAM1_PDIR    53 PARAM ; ( parallel direction for CAM1 0 = forwards, 1 = backwards
```

```
0 CONSTANT FORWARD
```

```
1 CONSTANT REVERSE
```

```
( DEFAULT PARAMETERS FOR THE AIS1 CAMERA
```

```
2048 CCD_SER !
```

```
 1 BIN_SER !
```

```
15 PRE_SER !
```

```
0 UNDER_SER !
```

```
0 ORG_SER !
```

```
2048 READ_SER !
```

```
15 POST_SER !
```

```
0 OVER_SER !
```

```
1024 CCD_PAR !
```

```
 1 BIN_PAR !
```

```
0 ORG_PAR !
```

```
1024 READ_PAR !
```

```
0 POST_PAR !
```

```
0 OVER_PAR !
```

```
20 ODELAY !
```

```
100 CDELAY !
```

```
0 EXP_TIME_HI !
```

```
200 EXP_TIME_LO !
```

```
2 NUM_CLEARS !
```

```
1 NUM_IMAGES !
```

```
0 IM_DELAY_HI !
```

```
50 IM_DELAY_LO !
```

```
0 CCLEAR !
```

```
0 ANTI-BLOOM !
```

```
0 ARCH !
```

```
0 SPEED !
```

```
( print a formatted list of current parameter set
```

```
( : SHOW-PARAMS
```

```
( CR
```

```
( ." CCD_SER    = " CCD_SER    @ 5 U.R CR
```

```
( ." BIN_SER    = " BIN_SER    @ 5 U.R CR
```

```
( ." PRE_SER    = " PRE_SER    @ 5 U.R CR
```

```
( ." UNDER_SER  = " UNDER_SER @ 5 U.R CR
```

```
( ." ORG_SER    = " ORG_SER    @ 5 U.R CR
```

```
( ." READ_SER   = " READ_SER   @ 5 U.R CR
```

```
( ." POST_SER   = " POST_SER   @ 5 U.R CR
```

```
( ." OVER_SER   = " OVER_SER   @ 5 U.R CR
```

```
( CR
```

```
( ." CCD_PAR    = " CCD_PAR    @ 5 U.R CR
```

```
( ." BIN_PAR    = " BIN_PAR    @ 5 U.R CR
```

```
( ." ORG_PAR    = " ORG_PAR    @ 5 U.R CR
```

```
( ." READ_PAR   = " READ_PAR   @ 5 U.R CR
```

```
( ." POST_PAR   = " POST_PAR   @ 5 U.R CR
```

```
( ." OVER_PAR   = " OVER_PAR   @ 5 U.R CR
```

```
( ." PAR_DELAY  = " PAR_DELAY  @ 5 U.R CR
```

```
( CR
```

```
( ." press any key to see next page of parameter list ..." BEGIN ?TERMINAL UNTIL CR
```

```
(
```

```
(
```

```
( ." ODELAY     = " ODELAY @ 5 U.R CR
```

```
( ." CDELAY     = " CDELAY @ 5 U.R CR
```

```
( CR
```

```
( ." EXP_TIME_HI = " EXP_TIME_HI @ 5 U.R CR
```

```
( ." EXP_TIME_LO = " EXP_TIME_LO @ 5 U.R CR
```

```
( CR
```



```

: CAM0_RST_LO      12 CAM0_VOLT ;
: CAM0_RST_HI      13 CAM0_VOLT ;
: CAM0_VRD         14 CAM0_VOLT ;
: CAM0_VOD         15 CAM0_VOLT ;
: CAM0_X1_LO       16 CAM0_VOLT ;
: CAM0_X1_HI       17 CAM0_VOLT ;
: CAM0_X2_LO       18 CAM0_VOLT ;
: CAM0_X2_HI       19 CAM0_VOLT ;
: CAM0_X3_LO       20 CAM0_VOLT ;
: CAM0_X3_HI       21 CAM0_VOLT ;
: CAM0_X4_LO       22 CAM0_VOLT ;
: CAM0_X4_HI       23 CAM0_VOLT ;

```

```

: THEN FOR CLOCK/ANALOG MODULE 1

```

```

NUM_VOLTS TABLE CAM1_VOLT

```

```

: CAM1_PAR_LO      0 CAM1_VOLT ;
: CAM1_PAR_MID     1 CAM1_VOLT ;
: CAM1_PAR_HI      2 CAM1_VOLT ;
: CAM1_SER_LO      3 CAM1_VOLT ;
: CAM1_SER_MID     4 CAM1_VOLT ;
: CAM1_SER_HI      5 CAM1_VOLT ;
: CAM1_TG_LO       6 CAM1_VOLT ;
: CAM1_TG_HI       7 CAM1_VOLT ;
: CAM1_SW_LO       8 CAM1_VOLT ;
: CAM1_SW_HI       9 CAM1_VOLT ;
: CAM1_SUB         10 CAM1_VOLT ;
: CAM1_LAST        11 CAM1_VOLT ;
: CAM1_RST_LO      12 CAM1_VOLT ;
: CAM1_RST_HI      13 CAM1_VOLT ;
: CAM1_VRD         14 CAM1_VOLT ;
: CAM1_VOD         15 CAM1_VOLT ;
: CAM1_X1_LO       16 CAM1_VOLT ;
: CAM1_X1_HI       17 CAM1_VOLT ;
: CAM1_X2_LO       18 CAM1_VOLT ;
: CAM1_X2_HI       19 CAM1_VOLT ;
: CAM1_X3_LO       20 CAM1_VOLT ;
: CAM1_X3_HI       21 CAM1_VOLT ;
: CAM1_X4_LO       22 CAM1_VOLT ;
: CAM1_X4_HI       23 CAM1_VOLT ;

```

```

( SHOW-VOLTS SEND FORMATTED TABLE OF CLOCK VOLTAGES TO HOST COMPUTER
: SHOW-VOLTS ( -
CR
( CAM0_PAR_LO C@ ." CAM0_PAR_LO = " 2 U.R CR
( CAM0_PAR_MID C@ ." CAM0_PAR_MID = " 2 U.R CR
( CAM0_PAR_HI C@ ." CAM0_PAR_HI = " 2 U.R CR
( CAM0_SER_LO C@ ." CAM0_SER_LO = " 2 U.R CR
( CAM0_SER_MID C@ ." CAM0_SER_MID = " 2 U.R CR
( CAM0_SER_HI C@ ." CAM0_SER_HI = " 2 U.R CR
( CAM0_TG_LO C@ ." CAM0_TG_LO = " 2 U.R CR
( CAM0_TG_HI C@ ." CAM0_TG_HI = " 2 U.R CR
( CAM0_SW_LO C@ ." CAM0_SW_LO = " 2 U.R CR
( CAM0_SW_HI C@ ." CAM0_SW_HI = " 2 U.R CR
( CAM0_SUB C@ ." CAM0_SUB = " 2 U.R CR
( CAM0_LAST C@ ." CAM0_LAST = " 2 U.R CR
( CAM0_RST_LO C@ ." CAM0_RST_LO = " 2 U.R CR
( CAM0_RST_HI C@ ." CAM0_RST_HI = " 2 U.R CR
( CAM0_VOD C@ ." CAM0_VOD = " 2 U.R CR
( CAM0_VRD C@ ." CAM0_VRD = " 2 U.R CR
( CAM0_X1_LO C@ ." CAM0_X1_LO = " 2 U.R CR
( CAM0_X1_HI C@ ." CAM0_X1_HI = " 2 U.R CR
( CAM0_X2_LO C@ ." CAM0_X2_LO = " 2 U.R CR
( CAM0_X2_HI C@ ." CAM0_X2_HI = " 2 U.R CR
( CAM0_X3_LO C@ ." CAM0_X3_LO = " 2 U.R CR
( CAM0_X3_HI C@ ." CAM0_X3_HI = " 2 U.R CR
( CAM0_X4_LO C@ ." CAM0_X4_LO = " 2 U.R CR
( CAM0_X4_HI C@ ." CAM0_X4_HI = " 2 U.R CR
CR
( CAM1_PAR_LO C@ ." CAM1_PAR_LO = " 2 U.R CR

```



```

DO I DUP CAM1_PAR_STATE @ SWAP
  ." CAM1_PAR_STATE" . ." = " 4 .R CR
LOOP CR
N_PAR_STATES @ 0
DO I DUP PAR_DELAY @ SWAP
  ." PAR_DELAY" . ." = " 4 .R CR
LOOP ;

```

```

: SET-PAR_DELAYS ( W - ) { sets all parallel delayss to the value W }
  N_PAR_STATES @ 0 DO DUP I PAR_DELAY ! LOOP ;

```

```

VARIABLE N_SER_STATES ( * CAM0_N_SER_STATES, NUMBER OF SERIAL STATES IN USE

```

```

TABLE_LEN TABLE SER_STATE
: SHOW-SER_STATES
  HEX CR
  N_SER_STATES @ 0
  DO I DUP SER_STATE @ SWAP
    ." SER_STATE" . ." = " 4 .R CR
  LOOP ;

```

```

VARIABLE N_ANA_STATES ( * N_PAR_STATES, NUMBER OF ANALOG STATES IN USE
TABLE_LEN TABLE ANA_STATE
TABLE_LEN TABLE ANA_DELAY

```

```

: SHOW-ANA_STATES
  HEX CR
  N_ANA_STATES @ 0
  DO I DUP ANA_STATE @ SWAP
    ." ANA_STATE" . ." = " 4 .R
    I DUP ANA_DELAY @ SWAP
    ." ANA_DELAY" . ." = " 4 .R CR
  LOOP ;

```

```

( AB_STATES are used in clock recombination anti-blooming. They are values to
( be written to the parallel clock control latch during such operation.
( AB_DELAYS represent the amount of time to pause between each state.

```

```

VARIABLE N_ANTI-BLOOM_STATES ( * N_ANTI-BLOOM_STATES = THE NUMBER OF ANTI-BLOOM STATES IN USE
16 TABLE ANTI-BLOOM_STATE
16 TABLE ANTI-BLOOM_WAIT

```

```

: SHOW-ANTI-BLOOM_STATES
  HEX CR
  N_ANTI-BLOOM_STATES @ 0
  DO I DUP ANTI-BLOOM_STATE @ SWAP
    ." ANTI-BLOOM_STATE" . ." = " 4 .R CR
    I DUP ANTI-BLOOM_WAIT @ SWAP
    ." ANTI-BLOOM_WAIT " . ." = " 4 .R CR
  LOOP ;

```

```

(
( State Definitions
( -----

```

```

( constants for the parallel timing states are defined below
HEX
00 CONSTANT P1_LO
01 CONSTANT P1_MID
02 CONSTANT P1_HI
00 CONSTANT P2_LO
04 CONSTANT P2_MID
08 CONSTANT P2_HI
00 CONSTANT P3_LO

```

```

10 CONSTANT P3_MID
20 CONSTANT P3_HI
00 CONSTANT P4_LO
40 CONSTANT P4_MID
80 CONSTANT P4_HI

```

```

( constants for transfer gate )
 0 CONSTANT TG_LO
100 CONSTANT TG_HI

```

```

( constants for the serial timing states are defined below
00 CONSTANT S1_LO
01 CONSTANT S1_MID
02 CONSTANT S1_HI
00 CONSTANT S2_LO
04 CONSTANT S2_MID
08 CONSTANT S2_HI
00 CONSTANT S3_LO
10 CONSTANT S3_MID
20 CONSTANT S3_HI
00 CONSTANT S4_LO
40 CONSTANT S4_MID
80 CONSTANT S4_HI

```

```

( constants for the analog timing states are defined below
HEX

```

```

 1 CONSTANT RIN ( integrator reset
 0 CONSTANT !RIN

 2 CONSTANT DCR ( D.C. restore
 0 CONSTANT !DCR

 4 CONSTANT SA2 ( sample period 2
 0 CONSTANT !SA2

 8 CONSTANT SA1 ( sample period 1
 0 CONSTANT !SA1

10 CONSTANT CTC ( command to convert
 0 CONSTANT !CTC

20 CONSTANT SEN ( send data trigger
 0 CONSTANT !SEN

40 CONSTANT 16B ( use slow speed 16 bit converter )
 0 CONSTANT !16B

```

```

( *****
( AIS CAMERA CONTROL FUNCTIONS
(
(
(
( *****

```

```

( *****
( the 68HC11 passes commands to the DSP through the DSP's command vector )
( register, CVR, the value written there determines which of the host commands )
( will be executed. There are a total of 28 command vectors which may be passed )
( the first 26 correspond to the DSP's normal exception processing including )
( external interrupts, software interrupts, and other interrupts associated with )
( the dsp's various peripherals. Through the CVR, we may force recognition of )
( any of these exceptions. Some of those are implemented in this code. )
( in addition, there are 12 command vectors reserved for use by the host. It is )
( through these command vectors that we normally force execution of dsp routines. )
( the routines which we may cause the dsp to execute include the following: )

```

FUNCTION	DESCRIPTION	PARAMETERS
say_ok	respond to 6811 as test	none
cam-write	write to camera address	addr data 16 bit
cam_read	read from camera address	
volt_test	test clock voltage dac's	
cam_ctl!	write to cam control latch	
init-volts	set cam clock voltages	24 8 bit voltage settings
init-states	set camera readout timing	
init-format	set readout parameters	xx 16 bit format parameters
read	read the ccd	none
clear	clear the ccd	1 8 bit number of clears
pix-bin	shift pixels to summing well	1 16 bit # of pixels
row-bin	shift rows into ser reg	1 16 bit # of rows
pix-discard	clear pixels in ser reg	1 16 bit # of pixels
row-discard	clear rows	1 16 bit # of pixels
pix-read	read pixels	1 16 bit # of rows
row-read	read rows	1 16 bit # of rows
shift_is	shift image to storage (frame transfer only)	
anti-bloom	perform clock recombination anti-blooming	

(VALUES TO WRITE TO THE CVR TO INITIATE THE DSP FUNCTIONS)

HEX

06 CONSTANT CMD_SAY_OK
 07 CONSTANT CMD_CAM_WRITE
 08 CONSTANT CMD_CAM_READ

(THE FOLLOWING four ARE USED BY AIS CAMERAS ONLY)

09 CONSTANT CMD_VOLT_TEST
 0A CONSTANT CMD_CAM_CTL!
 0E CONSTANT CMD_INIT_VOLTS
 0C CONSTANT CMD_INIT_STATES

(12 CONSTANT CMD_SHADE USED BY ATC5 ONLY

13 CONSTANT CMD_FORMAT
 14 CONSTANT CMD_READ
 15 CONSTANT CMD_CLEAR
 16 CONSTANT CMD_PIX_BIN
 17 CONSTANT CMD_ROW_BIN
 18 CONSTANT CMD_PIX_DISCARD
 19 CONSTANT CMD_ROW_DISCARD
 1A CONSTANT CMD_PIX_READ
 1B CONSTANT CMD_ROW_READ

1D CONSTANT CMD_SHIFT_IS
 1E CONSTANT CMD_ANTI_BLOOM

(there are several commands which will cause the DSP to perform
 (some operation repetitively until told by the 68hc11 to stop.
 (Host Flag 1 in the DSP registers is used as a boolean by the DSP
 (after the completion of each cycle to determine if it should
 (continue. Two functions currently use this flag : continuous
 (clearing of the CCD and clock recombination anti-blooming.
 (Others could be so configured.
 (The following word is used to stop any of these operations.

: CEASE-AND-DESIST F-HF1 ;

(The following word is used to encourage the DSP to continue

: PLEASE-CONTINUE T-HF1 ;

VARIABLE DSP_REV (STORAGE FOR DSP REVISION NUMBER)

HEX

: DSP-OK? CEASE-AND-DESIST
 CMD_SAY_OK HV!

```

DSP16@ 4F =
IF
    DSP16@ 4B = IF DSP16@ DSP_REV ! TRUE DSP_OK !
                ELSE FALSE DSP_OK !
                THEN
ELSE FALSE DSP_OK !
THEN
DSP_OK @ IF
    ELSE REPORT-DSP-ERROR
    THEN ;

( *****
(
    CAM-READ and CAM-WRITE
(
( Low-level commands for reading and writing DSP addresses from the FORTH
( program.
(

0 CONSTANT X_ADDR
1 CONSTANT Y_ADDR

( CAM-READ    read a value from a DSP address )
: CAM-READ    ( space address - datum )
    DSP-OK? DSP_OK @
    IF DSP-WAIT CEASE-AND-DESIST CMD_CAM_READ HV!
        DSP16!          ( pass the address )
        DSP16!          ( pass the address space : 0 = X 1 = Y )
        DSP16@          ( get the datum )
        DSP_OK @ IF DSP-WAIT
            ELSE ." ERROR : DSP error condition persists " CR
            THEN
    THEN ;

( CAM-WRITE    write a value to a DSP address )
: CAM-WRITE    ( datum space address - )
    DSP-OK? DSP_OK @
    IF DSP-WAIT CEASE-AND-DESIST CMD_CAM_WRITE HV!
        DSP16!          ( pass the address )
        DSP16!          ( pass the address space : 0 = x 1 = y )
        DSP16!          ( pass the data )
        DSP_OK @ IF DSP-WAIT
            ELSE ." ERROR : DSP error condition persists " CR
            THEN
    THEN ;

( *****
(
    CCD control registers
( -----
(
( The CCD is controlled by the DSP through the clock card. The clock
( card is a memory mapped device. By writing to the various locations
( on the clock card the DSP controls the entire CCD readout. Several of
( the most important locations on the clock card are defined below as
( the addresses at which the DSP accesses them.
( Each is described in some small detail below, alongwith thire definitions.

( AIS AND AIS2 ONLY
HEX
FF00 CONSTANT ID_LATCH    ( LOCATION, IN ALL CAMS, OF CAM ID LATCH
FF20 CONSTANT SER_LATCH  ( LOCATION OF CAM LATCH FOR SERIAL CLOCK CONTROL
FF24 CONSTANT PAR_LATCH  ( LOCATION OF CAM LATCH FOR PARALLEL CLOCK CONTROL
FF28 CONSTANT OUT_LATCH  ( LOCATION OF CAM LATCH FOR RESET, SUMMING WELL,
                          ( AND TRANSFER GATE CONTROL
FF2C CONSTANT ANA_LATCH  ( LOCATION OF CAM LATCH FOR ANALOG PROCESSING CONTROL
FF30 CONSTANT CTL_LATCH  ( LOCATION OF CAM LATCH FOR CAM SETUP

```



```
: CAM1-ONLY   CAM-DISABLE-ALL CAM1 CAM-ENABLE ;
```

```
( ***** )
( CAM CONTROL REGISTERS
; EACH CAM CONTAINS AN 8 BIT CONTROL REGISTER
(
( BIT 0   CLOCK ENABLE           CLOCKS THROUGH TO CCD  ACTIVE HIGH
( BIT 1   ANALOG CONTROL ENABLE
( BIT 2   PARALLEL CLOCK ENABLE
( BIT 3   SERIAL CLOCK ENABLE
( BIT 4   OTHER CLOCKS ANABLE
( BIT 5   NOT USED
( BIT 6   NOT USED
( BIT 7   NOT USED
```

```
( a storage location for current value in control latch for each CAM
```

```
VARIABLE CTL_CAM0
VARIABLE CTL_CAM1
```

```
( VARIABLE CTL_CAM2
( VARIABLE CTL_CAM3
( VARIABLE CTL_CAM4
( VARIABLE CTL_CAM5
( VARIABLE CTL_CAM6
( VARIABLE CTL_CAM7
```

```
( CAM-CTL! stores a value in the camera control latch of the
( currently selected CAM. expects the value on the stack
```

```
: CAM-CTL!
    DSP-WAIT
    CLR-TXH-TXM
    CMD_CAM_CTL! HV!      ( call function
    DSP16!                ( pass data
    DSP-WAIT ;
```

```
(          ENABLE-CCD-CLOCKS and  DISABLE-CCD-CLOCKS
( -----
( These words are used to open and close the switches which allow the
( CCD clocks and voltages to be connected to the buffers or wires which
( connect them to the CCD. When the switches are open, pull down resistors
( pull the potentials to near 0 volts.
(
```

```
( the following word is used to enable the clock signals in each CAM
( it expects the CAM ID { CAM0 - CAM7 } to be on the stack
```

```
: CLKS-EN
    CAM_ID @ TEMPO !
    DUP CAM0 =
    IF
        CAM0-ONLY CTL_CAM0 @ T-0 DUP CTL_CAM0 !
    ELSE
        DUP CAM1 =
        IF
            CAM1-ONLY CTL_CAM1 @ T-0 DUP CTL_CAM1 !
        THEN
    THEN
    CAM-CTL!
    DROP TEMPO @ CAM_ID ! CAM-ID! ;
```

```
( the following word is used to disable the clock signals in each CAM
( it expects the CAM ID { CAM0 - CAM7 } to be on the stack
```

```
: CLKS-DIS
    CAM_ID @ TEMPO !
    DUP CAM0 =
    IF
        CAM0-ONLY CTL_CAM0 @ F-0 DUP CTL_CAM0 !
    ELSE
```

```

    DUP CAM1 =
    IF
        CAM1-ONLY CTL_CAM1 @ F-0 DUP CTL_CAM1 !
    THEN
    THEN
    CAM-CTL!
    DROP TEMPO @ CAM_ID ! CAM-ID! ;

```

```

( the following words are used to enable and disable the different latches
( in each cam. they expect the cam id ( cam0 - cam7 ) to be on the stack.
: ANALOG-EN

```

```

    CAM_ID @ TEMPO !
    DUP CAM0 =
    IF
        CAM0-ONLY CTL_CAM0 @ F-1 DUP CTL_CAM0 !
    ELSE
        DUP CAM1 =
        IF
            CAM1-ONLY CTL_CAM1 @ F-1 DUP CTL_CAM1 !
        THEN
    THEN
    CAM-CTL!
    DROP TEMPO @ CAM_ID ! CAM-ID! ;

```

```

: ANALOG-DIS

```

```

    CAM_ID @ TEMPO !
    DUP CAM0 =
    IF
        CAM0-ONLY CTL_CAM0 @ T-1 DUP CTL_CAM0 !
    ELSE
        DUP CAM1 =
        IF
            CAM1-ONLY CTL_CAM1 @ T-1 DUP CTL_CAM1 !
        THEN
    THEN
    CAM-CTL!
    DROP TEMPO @ CAM_ID ! CAM-ID! ;

```

```

(           ENABLE-PARALLELS and DISABLE-PARALLELS
(           -----

```

```

( These words are used to bit enable and disable the output of the
( parallel clock control latch.
(
( These words also set the Frame ENable and Line ENable signals to
( their active low states allowing interface boards that monitor
( these signals to collect the data while the parallels are not running.
( This low level mode of operation is useful for test purposes.
(

```

```

: PCLKS-EN

```

```

    CAM_ID @ TEMPO !
    DUP CAM0 =
    IF
        CAM0-ONLY CTL_CAM0 @ F-2 DUP CTL_CAM0 !
    ELSE
        DUP CAM1 =
        IF
            CAM1-ONLY CTL_CAM1 @ F-2 DUP CTL_CAM1 !
        THEN
    THEN
    CAM-CTL!
    DROP TEMPO @ CAM_ID ! CAM-ID! ;

```

```

: PCLKS-DIS

```

```

    CAM_ID @ TEMPO !
    DUP CAM0 =
    IF
        CAM0-ONLY CTL_CAM0 @ T-2 DUP CTL_CAM0 !
    ELSE

```

```

    DUP CAM1 =
    IF
        CAM1-ONLY CTL_CAM1 @ T-2 DUP CTL_CAM1 !
    THEN
    THEN
    CAM-CTL!
    DROP TEMPO @ CAM_ID ! CAM-ID! ;

(
    ENABLE-SERIALS and DISABLE-SERIALS
(
    -----
(
( These words are used to bit enable and disable the output of the
( SERIAL clock control latch.
(

: SCLKS-EN
    CAM_ID @ TEMPO !
    DUP CAM0 =
    IF
        CAM0-ONLY CTL_CAM0 @ F-3 DUP CTL_CAM0 !
    ELSE
        DUP CAM1 =
        IF
            CAM1-ONLY CTL_CAM1 @ F-3 DUP CTL_CAM1 !
        THEN
    THEN
    CAM-CTL!
    DROP TEMPO @ CAM_ID ! CAM-ID! ;

: SCLKS-DIS
    CAM_ID @ TEMPO !
    DUP CAM0 =
    IF
        CAM0-ONLY CTL_CAM0 @ T-3 DUP CTL_CAM0 !
    ELSE
        DUP CAM1 =
        IF
            CAM1-ONLY CTL_CAM1 @ T-3 DUP CTL_CAM1 !
        THEN
    THEN
    CAM-CTL!
    DROP TEMPO @ CAM_ID ! CAM-ID! ;

: OCLKS-EN
    CAM_ID @ TEMPO !
    DUP CAM0 =
    IF
        CAM0-ONLY CTL_CAM0 @ F-4 DUP CTL_CAM0 !
    ELSE
        DUP CAM1 =
        IF
            CAM1-ONLY CTL_CAM1 @ F-4 DUP CTL_CAM1 !
        THEN
    THEN
    CAM-CTL!
    DROP TEMPO @ CAM_ID ! CAM-ID! ;

: OCLKS-DIS
    CAM_ID @ TEMPO !
    DUP CAM0 =
    IF
        CAM0-ONLY CTL_CAM0 @ T-4 DUP CTL_CAM0 !
    ELSE
        DUP CAM1 =
        IF
            CAM1-ONLY CTL_CAM1 @ T-4 DUP CTL_CAM1 !
        THEN
    THEN
    CAM-CTL!
    DROP TEMPO @ CAM_ID ! CAM-ID! ;

```

```

: ALL-CLKS-EN      ( enable all clock signals on currently selected CAM )
  O1 CAM-CTL! ;

: ALL-CLKS-DIS    ( disable all clock signals on currently selected CAM )
  FE CAM-CTL! ;

```

```

( ***** )

```

```

          Format commands
          -----

```

```

          FORMAT!, FORMAT?

```

```

These two words may be used to set and examine the format parameters in
a fairly low-level way. In each case the format parameters are passed as a
simple character stream. The order of the parameters is easy to remember.

```

```

( FORMAT sets the format parameters to the values found on the stack )

```

```

( FORMAT SETS THE FORMAT PARAMETERS TO THE VALUES FOUND ON THE STACK )

```

```

DECIMAL

```

```

: FORMAT!
  DEPTH 14 <
  IF
    CR ." ERROR : INSUFFICIENT STACK ENTRIES " CR
  ELSE
    CCD_SER ! BIN_SER ! PRE_SER ! UNDER_SER ! ORG_SER !
    READ_SER ! POST_SER ! OVER_SER !

    CCD_PAR ! BIN_PAR ! ORG_PAR ! READ_PAR ! POST_PAR ! OVER_PAR !
  THEN ;

```

```

( FORMAT? SIMPLY ECHOES THE FORMAT PARAMTER VALUES IN ORDER )

```

```

: FORMAT?
  CR
  CCD_SER @ U. BIN_SER @ U. PRE_SER @ U. UNDER_SER @ U.
  ORG_SER @ U. READ_SER @ U. POST_SER @ U. OVER_SER @ U.

  CCD_PAR @ U. BIN_PAR @ U.
  ORG_PAR @ U. READ_PAR @ U. POST_PAR @ U. OVER_PAR @ U. PAR_DELAY @ U.
  ARCH @ U.
  CR ;

```

```

( ***** )

```

```

          Camera Initialization
          -----

```

```

          INIT-DSP, INIT-VOLTS, INIT-STATES, INIT-FORMT, INIT-ALL

```

```

( A variety of initialization routines are required to get the camera hardware
( ready to acquire image data. All of them are executed by calling one routine
( 'INIT' defined near the end of this document. The INIT routine performs a
( number of other tasks as well. These routines are used when it is necessary
( to reinitialize some part of the hardware due to a configuration change. Much
( camera configuration information is not valid until initialized. Clock card
( voltages, for instance, are not set to the values stored in the VOLT table until
( the INIT-VOLTS function is performed. Similiarly, the readout format parameters
( due not take effect until the INIT-FORMAT function is called, and the timing
( information stored in the various timing tables is not used until INIT-STATES is
( called. Nothing can be accomplished at all if the DSP is not operating. The
( INIT-DSP function is provided for the purpose of initializing the DSP program.
( This is useful in development work where new DSP code has been downloaded.
( The INIT-ALL function is a convenient way to just re-initialize everything
( having to do with the DSP sequencer.

```

```

HEX

```

```

: INIT-DSP
  DSP-BOOT
  ICR C@ 0 =
  ISR C@ 6 = AND
  CVR C@ 12 = AND
  IF      ( if boot was successful ...
    DSP-DUMP      ( ... dump code to DSP.
    DSP-OK?      ( ... check DSP status
  ELSE
    FALSE DSP_OK !
    CR ." ERROR : DSP bootstrap failed " CR
  THEN ;

( SETS THE CLOCK BOARD VOLTAGES)
: INIT-VOLTS ( - )
  DSP-OK? DSP_OK @
  IF CLR-TXH-TXM
    CAMO-ONLY
    CMD_INIT_VOLTS HV!
    NUM_VOLTS 0 DO I CAMO_VOLT C@ DSP8! LOOP
    ." CAM0 voltages initialized " CR
  DSP-OK? DSP_OK @
  IF CAM1-ONLY
    CMD_INIT_VOLTS HV!
    NUM_VOLTS 0 DO I CAM1_VOLT C@ DSP8! LOOP
    ." CAM1 voltages initialized " CR
  DSP-WAIT
  ELSE ." ERROR : INIT-VOLTS terminated, DSP error " CR
  THEN
  ELSE ." ERROR : INIT-VOLTS terminated, DSP error " CR
  THEN ;

( INIT-STATES DOWNLOADS THE CURRENT SET OF PAR_STATE, SER_STATE,
  AND ANA_STATE VALUES TO THE DSP
: INIT-STATES ( - )
  DSP-OK? DSP_OK @
  IF CMD_INIT_STATES HV!
    CLR-TXH-TXM

    N_PAR_STATES @ DUP DSP16! 0 DO I CAMO_PAR_STATE @ DSP16! LOOP
    N_PAR_STATES @          0 DO I CAM1_PAR_STATE @ DSP16! LOOP
    N_PAR_STATES @          0 DO I PAR_DELAY @ DSP16! LOOP
    N_SER_STATES @ DUP DSP16! 0 DO I SER_STATE @ DSP16! LOOP
    N_ANA_STATES @ DUP DSP16! 0 DO I ANA_STATE @ DSP16! I ANA_DELAY @ DSP16! LOOP
    N_ANTI-BLOOM_STATES @ DUP DSP16!
    0 DO I ANTI-BLOOM_STATE @ DSP16! I ANTI-BLOOM_WAIT @ DSP16! LOOP
    DSP-WAIT
  ELSE ." ERROR : INIT-STATES terminated, DSP error " CR
  THEN ;

: INIT-FORMAT
  DSP_OK @ 0=
  IF CR ." DSP ERROR : INIT-FORMAT aborted " CR
  ELSE
    DSP-WAIT
    CLR-TXH-TXM
    CEASE-AND-DESIST
    CMD_FORMAT HV!
    CCD_SER @ DSP16!   BIN_SER @ DSP16!   PRE_SER @ DSP16!   UNDER_SER @ DSP16!
    ORG_SER @ DSP16!   READ_SER @ DSP16!   POST_SER @ DSP16!   OVER_SER @ DSP16!
    CCD_PAR @ DSP16!   BIN_PAR @ DSP16!   ORG_PAR @ DSP16!   READ_PAR @ DSP16!
    POST_PAR @ DSP16!   OVER_PAR @ DSP16!
    CAMO_PDIR @ DSP16!   CAM1_PDIR @ DSP16!
    DSP-WAIT
  THEN ;

```

```
( INITIALIZE EVERYTHING
: INIT-ALL
  INIT-DSP
  INIT-VOLTS
  INIT-STATES
  INIT-FORMAT
;
```

```
( ***** )
```

```
Camera Speed
```

```
ITIME!, 40KHZ, 35KHZ, 30KHZ, 25 KHZ, 20KHZ, 15KHZ, 10KHZ
```

```
( The AIS1 READOUT SPEEDlow speed is adjustable from approx 40kHz downward.
( The exact speed depends on some timing variables, most importantly the integration
( time of the dual slope integrator. The SPEED may be adjusted over a much wide
( range and. As the ANALOG TO DIGITAL convertors architecture is that of a dual
( slope integrator, the gain of thr system is directly affected by the integration
( time. The integration time is determined by the values of ANA_DELAYS 7 and 9.
( Several words are provided here to set the cameras slow readout rate to convenient
( values.
```

```
DECIMAL
```

```
( The following word sets the integration time for the slow speed analog processor.
( It expects a 16 bit number on the stack with a value ranging from 1 -> 65535.
( This value is used by the DSP as a counter for the integration delay. The DSP
( delays 100 ns for every unit in the integration time. Integrations may therefore
( vary from 100ns to 6.5535 ms. In practice, there is approx 350 ns overhead in the
( delay time and 100ns integration times are not possible. Additionally, if the
( time is set too short, the converter will not have finished converting the previous
( pixel when the new command to convert comes along. Integration times as short as
( 1 us should work fine. Extremely long integrations will saturate the converter.
```

```
: ITIME! ( W - ) ( expects the desired integration time on the stack )
  DUP 7 SLOW_DELAY !
  9 SLOW_DELAY !
  INIT-STATES ;
```

```
( SPEED! is used to set the slow speed to a particular value. It expects the
( new value for the integration time on the stack. It is used primarily as a
( convenience for the definition of the words that follow.
```

```
: SPEED! ( W - )
  1 0 ANA_DELAY !
  1 1 ANA_DELAY !
  1 2 ANA_DELAY !
  1 3 ANA_DELAY !
  1 4 ANA_DELAY !
  12 5 ANA_DELAY !
  1 6 ANA_DELAY !
  1 8 ANA_DELAY !
  1 10 ANA_DELAY !
  1 11 ANA_DELAY !
  1 12 ANA_DELAY !
  50 13 ANA_DELAY !
  DUP
  ( STACK ) 7 ANA_DELAY !
  ( STACK ) 9 ANA_DELAY !
  INIT-STATES ;
```

```
( The following words set the camera readout speed to some convenient values.
```

```
DECIMAL
: 40KHZ 38 SPEED! ;
: 35KHZ 56 SPEED! ;
: 30KHZ 90 SPEED! ;
: 20KHZ 163 SPEED! ;
```

```
: 15KHZ 245 SPEED! ;
: 10KHZ 411 SPEED! ;
: 5KHZ 910 SPEED! ;
```

```
( * ***** )
( WORDS FOR SHUTTER CONTROL
( ***** )
( ***** )
( The following words are used for shutter control in AIS1 cameras
```

```
: OPEN1 PORTA C@ T-5 PORTA C! ;
: OPEN2 PORTA C@ T-6 PORTA C! ;
```

```
: CLOSE1 PORTA C@ F-5 PORTA C! ;
: CLOSE2 PORTA C@ F-6 PORTA C! ;
```

```
: OPEN
    OPEN1 OPEN2
    ODELAY @ 0 MS-WAIT ;
```

```
: CLOSE
    CLOSE1 CLOSE2
    CDELAY @ 0 MS-WAIT ;
```

```
( The following words are used for shutter control
( in ATC5 and AIS2 cameras.
```

```
( : OPEN PORTA C@ T-6 PORTA C! ODELAY @ 0 MS-WAIT ;
```

```
( : CLOSE PORTA C@ F-6 PORTA C! CDELAY @ 0 MS-WAIT ;
```

```
( ***** )
( System commands
```

```
( ***** )
( ***** )
```

```
( Continuous Clearing of the CCD
( -----
```

```
( START-CISC, STOP-CISC, CISC?, CISC-ON, and CISC-OFF
```

```
( This set of words is used in conjunction with continuous clearing of the CCD
( CCD between readouts. It is often desirable to continuously clear charge off
( the CCD between image acquisitions. If there is no shutter in the system or if
( the temperature of the CCD is fairly high this is very desirable. In a cryogenic
( camera, this is not as important as the dark current is typically very low.
```

```
( The 'CISC' terminology comes from the Photometrics CC200 where it stood for
( 'Clear Image and Storage Continuously'.
```

```
( Continuous clearing of the camera is controlled by the DSP. If it is in its idle
( state waiting for a host command and it detects Host Flag 0 in the Host Interface
( as a '1', then the DSP clears a row off the CCD. The flag is used in a reverse
( logical sense, i.e. a '0' enables CISC and a '1' disables it.
```

```
( The 68HC11 controls this feature simply by setting and clearing this flag at
( any time. A parameter is maintained which may be polled at any time to determine
( if it should do so.
```

```
: START-CISC F-HF0 ;
```

```
: STOP-CISC T-HF0 ;
```

{ the following word checks the parameter and either enables or disables CISC

```
: CISC?      CCLEAR @ IF START-CISC
              ELSE STOP-CISC
              THEN ;
```

{ the following words both set the flag and the parameter }

```
: CISC-ON    TRUE  CCLEAR ! START-CISC ;
: CISC-OFF   FALSE CCLEAR ! STOP-CISC ;
```

{ ***** }

Clock Recombination Anti-Blooming

START-CRAB, STOP-CRAB, CRAB?, CRAB-ON, and CRAB-OFF

{ This set of words is used in conjunction with continuous clearing of the CCD
{ CCD between readouts. It is often desirable to continuously clear charge off
{ the CCD between image acquisitions. If there is no shutter in the system or if
{ the temperature of the CCD is fairly high this is very desirable. In a cryogenic
{ camera, this is not as important as the dark current is typically very low.

{ Clock recombination anti-blooming is controlled by the DSP. It may be told to
{ begin the process through a Host Command at any time that it is not executing
{ any other host command. It will continue until the 68HC11 tells it to stop
{ by clearing HF1. (see CEASE-AND-DESIST above)

{ The 68HC11 will call this function during exposures if the ANTI-BLOOM
{ parameter is set to true.

```
: START-CRAB  CEASE-AND-DESIST
              DSP-WAIT
              CMD_ANTI_BLOOM HV! ;
: STOP-CRAB   CEASE-AND-DESIST ;
```

{ the following word checks the parameter and either enables or disables CRAB

```
: CRAB?      ANTI-BLOOM @ IF START-CRAB ELSE STOP-CRAB THEN ;
```

{ The following words set or clear the ANTI-BLOOM parameter.
{ They DO NOT start the anti-blooming they simply set the camera up so that it
{ will be performed during the next exposure.

```
: CRAB-ON    TRUE  ANTI-BLOOM ! ;
: CRAB-OFF   FALSE ANTI-BLOOM ! CEASE-AND-DESIST ;
```

{ ***** }

{ the exposure time desired by the user is in the parameter table above.
{ It is stored as an upper and lower half, each 16 bits wide.

{ the variable exp_left used below is defined above, so as to be accessible
{ to the timer interrupt routines. It is decremented by the millisecond
{ timer when the exp flag is set. The EXPOSING is defined above as well.

```
: EXP-START
  CRAB?
  FALSE EXPOSING !
  EXP_TIME_LO @ EXP_TIME_HI @ 0= SWAP 0= AND
```


sequence based on low level routines, due to increased participation by the 68HC11, but very complicated read sequences can be constructed. Not all users will wish to use these routines, as normal operational modes are fully supported by the high level image acquisition routines.

Low level image acquisition routines include the following:

```

PIX-BIN      shift N serial pixels into the summing well
ROW-BIN      shift N parallel rows into the serial register
PIX-CLR      clear N pixels from serial register
ROW-CLR      clear N rows off parallel register
PIX-READ     read N pixels from serial register
ROW-READ     read N rows from parallel register

```

These low level dsp routines can be used to build more complex functions by combining calls to them into forth words and storing these in the dictionary.

DSP-16CMD is used by a number routines that take that particular form. Only works for functions that pass a 16 bit value to the dsp. Implemented here to save on dictionary space. Note that not all the dsp commands are issued this way. This word expects the value and the command number on the stack. Returns nothing.

```

: DSP-16CMD      ( W W - )
  DSP-WAIT CEASE-AND-DESIST      ( store the command number from the stack )
  HV!
  DSP16!
  DSP-WAIT ;

```

```

( PIX-BIN      shift w serial pixels into the summing well )
: PIX-BIN      ( W - )
  CMD_PIX_BIN DSP-16CMD ;

```

```

( ROW-BIN      shift w parallel rows into the serial register )
: ROW-BIN      ( W - )
  CMD_ROW_BIN DSP-16CMD ;

```

```

( PIX-CLR      clear w pixels from serial register )
: PIX-CLR      ( W - )
  CMD_PIX_DISCARD DSP-16CMD ;

```

```

( ROW-CLR      clear w rows off parallel register )
: ROW-CLR      ( W - )
  CMD_ROW_DISCARD DSP-16CMD ;

```

```

( PIX-READ     read w pixels from serial register )
: PIX-READ     ( W - )
  CMD_PIX_READ DSP-16CMD ;

```

```

( ROW-READ     read w rows from parallel register )
: ROW-READ     ( W - )
  CMD_ROW_READ DSP-16CMD ;

```

*****)

High Level Image Acquisition Routines

```

( SHIFT-IS     shift image portion of frame transfer device to storage )
( CLEAR        clear the entire ccd array, num_clears times
( READ         read out the CCD based on format,
(              storage area only in a frame transfer device

```

```

( BIAS      acquire a bias frame, num_images times
( DARK      acquire a dark reference frame, num_images times
( OBS       acquire a light frame, num_images times

(
( SHIFT_IS  shift image portion of frame transfer device to storage )
(           Not yet implemented in the AIS1 camera.
( : SHIFT_IS  ( - )
(           DSP-WAIT CEASE-AND-DESIST CMD_SHIFT_IS HV! DSP-WAIT ;
(

( CLEAR    clear the entire ccd array, num_clears times
( All charge is cleared off the CCD. The DSP is told to stop
( any current action, the 68HC11 waits until the DSP does so,
( and then passes the clear command to the DSP. The 68HC11 then
( waits for the DSP to finish clearing the CCD.
( This process is repeated NUM_CLEARS times. NUM_CLEARS is a
( a parameter the value of which may vary from 0 -> 65535.

: CLEAR
  NUM_CLEARS @ 0= NOT
  IF NUM_CLEARS @ 0
    DO DSP-WAIT
      CEASE-AND-DESIST
      CMD_CLEAR HV!
      DSP-WAIT LOOP
  THEN
;

( READ    read the image data off the CCD
( The DSP is told to stop any current action, the 68HC11 waits until
( the DSP does so, and then passes the READ command to the DSP. The
( 68HC11 then waits for the DSP to finish reading the image off the CCD.
( The CCD is read out the CCD based on the current format parameters.
( Only the storage area is read in a frame transfer device.

: READ
  DSP-WAIT
  CEASE-AND-DESIST
  CMD_READ HV!
  DSP-WAIT
;

( BIAS    acquire a bias frame, num_images times
( The shutter is closed, CISC stopped, the CCD is cleared, the
( image data is read off the chip based on the current format parameters,
( continuous clearing of the chip is begun, if enabled, and then a
( delay of IM_DELAY milliseconds is executed.
( This entire sequence is executed NUM_IMAGES times.
: BIAS    ( - )
  NUM_IMAGES @ 0 DO
    CLOSE
    T-HFO
    CLEAR
    READ
    CISC?
    IM_DELAY_LO @ IM_DELAY_HI @ MS-WAIT
  LOOP
;

( DARK    acquire a dark reference frame, num_images times
( The shutter is closed, CISC stopped, the CCD is cleared, dark
( current is integrated for EXP_TIME milliseconds, the image data
( is read off the chip based on the current format parameters,
( continuous clearing of the chip is begun, if enabled, and then a
( delay of IM_DELAY milliseconds is executed.
( This entire sequence is executed NUM_IMAGES times.
( If enabled, clock recombination anti blooming is performed during

```

```

dark frame integration.
: DARK      ( - )
            NUM_IMAGES @ 0 DO
            CLOSE
            T-HFO
            CLEAR
            INTEGRATE_DARK
            READ
            CISC?
            IM_DELAY_LO @ IM_DELAY_HI @ MS-WAIT
            LOOP
            ;

: OBS      acquire a light frame, num_images times
:          The shutter is closed, CISC stopped, the CCD is cleared, the
:          shutter is opened, light is integrated for EXP_TIME milliseconds,
:          the shutter is closed, the image data is read off the chip based
:          on the current format parameters, continuous clearing of the chip
:          is begun, if enabled, and then a delay of IM_DELAY milliseconds
:          is executed.
:          This entire sequence is executed NUM_IMAGES times.
:          If enabled, clock recombination anti-blooming is performed during
:          light integration.

: OBS      ( - )
            NUM_IMAGES @ 0 DO
            CLOSE
            STOP-CISC
            CLEAR
            EXPOSE_CCD
            READ
            CISC?
            IM_DELAY_LO @ IM_DELAY_HI @ MS-WAIT
            LOOP
            ;

( ***** )
(
(           Test Functions
(           -----
(
(   A small set of test functions are provided for hardware debug.
( Most are repetitive words which are useful for testing for testing small
( subsets of the clock and analog cards.
(
(   RREAD      read the CCD until terminal break
(   RCLEAR     clear the CCD until terminal break
(   RBIAS     clear, then read the CCD until terminal break
(   ROBS      execute a light acquisition sequence until terminal break
(   RDARK     execute a dark acquisition sequence until terminal break
(   VOLT-TEST test all clock card voltages
(   LATCH-TEST test all clock card latches
(
(
(
: ROBS      BEGIN OBS      ?TERMINAL UNTIL ;
: RDARK     BEGIN DARK     ?TERMINAL UNTIL ;
: RREAD     BEGIN READ     ?TERMINAL UNTIL ;
: READS     BEGIN READ     ?TERMINAL UNTIL ;
: RCLEAR    BEGIN CLEAR   ?TERMINAL UNTIL ;
: RBIAS     BEGIN BIAS    ?TERMINAL UNTIL ;

```

```
( ***** )
(
(           Startup Initialization
(           -----
(
( The 68HC11 performs certain functions on startup.
(
( Version number is echoed, housekeeping takes place, an attempt is made to
( initialize the DSP. If successful, the clock card is initialized through the
( DSP. If not an error message is returned, and you might as well give up on
( acquiring any image data or doing much else useful. A hardware failure is
( the only likely cause of such difficulty. A sign on message is generated.
(
```

HEX

```
: INIT
6104 200 6100 @ 6102 @ - CMOVE
6100 @ 6102 @ - 6104 + 6 84 CMOVE

STOP-TEMP-CTL
DECIMAL
VERSION
FALSE EXPOSING !
INSTALL-TIMER INSTALL-TEMP-CTL
START-TIMER START-TEMP-CTL
0 CAM_ID C!
20 ODELAY ! 20 CDELAY !
INIT-DSP
FALSE DSP_OK ! ( guilty until proven innocent )
DSP-OK?
DSP_OK @
IF
." DSP ok revision : " DSP_REV @ U. CR
CISC-OFF
INIT-STATES
INIT-VOLTS
INIT-FORMAT
ALL-CLKS-EN
CAM-ID!
ELSE
." DSP NOT RESPONDING ... camera control functions not available " CR
THEN
." AIS camera initialized " CR
;
```

```
( ***** )
( ASTART!
HEX
EPROM CONSTANT ASTART
```

```
: ASTART!
EEUNPROT ( * UNPROTECT THE EEPROM
A44A DUP ASTART EE-! ( * STORE THE AUTOSTART PATTERN
[ ' RESTOR ] LITERAL CFA ( * GET THE CFA OF OUR AUTOSTART WORD
DICT-START - ( * DICT OFFSET OF AUTOSTART WORD
EEDICT-START + ( * LOCATION OF AUTOSTART WORD IN EEPROM)
DUP ASTART 2+ EE-! ( * STORE AFTER THE AUTOSTART PATTERN
EEPROT ( * PROTECT THE EEPROM
CR ." autostart sequence stored " CR ;
```

```
( ***** )
(           Short Form Commands
(           -----
(
( A set of short form commands is provided which is particularly useful
( to those very familiar with the command set or for programmed hardware
( interfaces, such as the Advanced Technologies VMEbus interface. All commands
( in the set are abbreviated to three letters in order to maximize the transfer
( rate of comands between the host computer and the camera controller.
(
```

```

( Commands are grouped by classes.
  The class identifier, F, L, S, I, or C, is the first character in the name.
)
:
: C :
: S :
( F : commands that manipulate format parameters
( L : lowest level image acquisition routines
( I :
( E :
)
)

: ^ ! ;
: ST STORE ASTART! ;

( P! stores a parameter
( it expects the value and parameter on the stack
: P!
  DEPTH DUP 2 <
  IF
    CR ." needs the value and parameter on stack "
    0 DO DROP LOOP
  ELSE
    DROP
    ( STACK ) ( STACK ) PARAM ! INIT-FORMAT
  THEN
;

( C: Controller commands )
: CXX COLD ; ( restart the 68HC11 FORTH interpreter )
: CIN INIT ; ( initialize the camera )

( S: System commands )
: ST@ TEMP! ; ( CCD temperature short form)
: STC CCD-TEMP? ; ( CCD temperature long form)
: ST! TEMP! ; ( set the CCD temperature )

: SCF CISC-OFF ; ( disable continuous clearing of the CCD )
: SCT CISC-ON ; ( enable continuous clearing of the CCD )

: SBF CRAB-OFF ; ( disable clock recombination anti-blooming )
: SBT CRAB-ON ; ( enable clock recombination anti-blooming )

: SSO 200 0 EXP_LEFT! OPEN ; ( open the shutter )
: SSC CLOSE ; ( close the shutter )

( these system commands are specific to AIS and AIS2 cameras
: SIV INIT-VOLTS ; ( initialize CAM voltages )
: SIF INIT-FORMAT ; ( initialize format parameters )
: SIS INIT-STATES ; ( initialize timing parameters )
: SPD SET-PAR_DELAYS ; ( set the parallel delay times to a new value )

( F: Format or parameter-related commands )
: FP! PARAM ! ; ( store a parameter )
: FP@ PARAM @ U. ; ( fetch a parameter )
: FF? FORMAT? ; ( fetch the format parameters in an ordered list )
: FSF INIT-FORMAT ; ( initialize format parameters, redundant to SIF)

( L: Low-level commands )
: LPB PIX-BIN ; ( bin N pixels )
: LRB ROW-BIN ; ( bin N rows )
: LPC PIX-CLR ; ( discard N pixels )
: LRC ROW-CLR ; ( discard N rows )
: LPR PIX-READ ; ( read N pixels )
: LRR ROW-READ ; ( read N rows )
: LCW CAM-WRITE ; ( write a value to a camera address )
: LCR CAM-READ ; ( read a value from a camera address )

( I: Image-oriented or data producing commands )
: IIL EXPOSE_CCD ; ( integrate light )
: IID INTEGRATE_DARK ; ( integrate dark )

```

```

: IRD READ ;          ( rad the image off the CCD based on current format )
: ICL CLEAR ;        ( clear the CCD of all charge )

: IAB BIAS ;         ( acquire bias )
: IAD DARK ;        ( acquire dark )
: IAL OBS ;         ( acquire light )

```

(special ones for AIS style cameras

```

: CEA CAM-ENABLE-ALL ;
: CE0 CAM0 CAM-ENABLE ;
: CE1 CAM1 CAM-ENABLE ;

: CDA CAM-DISABLE-ALL ;
: CD0 CAM0 CAM-DISABLE ;
: CD1 CAM1 CAM-DISABLE ;

: ACE ALL-CLKS-EN ;
: ACD ALL-CLKS-DIS ;

```

HERE U.

DECIMAL

0 0 TIME!

0 0 EXP_LEFT!
FALSE EXPOSING !

```

0 ACT_TMP !
150 DES_TMP !
0 TMP_CNT !
5 MAX_CNT !
0 DAC_VAL !
10 DAC_INC !

```

FALSE DSP_OK !

```

2048 CCD_SER !
1 BIN_SER !
20 PRE_SER !
0 UNDER_SER !
0 ORG_SER !
2048 READ_SER !
20 POST_SER !
0 OVER_SER !

```

```

2048 CCD_PAR !
1 BIN_PAR !
0 ORG_PAR !
2048 READ_PAR !
0 POST_PAR !
0 OVER_PAR !
200 PAR_DELAY !

```

```

100 ODELAY !
100 CDELAY !

```

```

200 EXP_TIME_LO !
0 EXP_TIME_HI !

```

```

2 NUM_CLEARS !
1 NUM_IMAGES !

```

```

50 IM_DELAY_LO !
0 IM_DELAY_HI !

```

```

FALSE CCLEAR !
FALSE ANTI-BLOOM !

```

```
0 ARCH      !  
0 SPEED     !  
  
0 CAMO_PDIR !  
0 CAM1_PDIR !  
0 DSP_REV  !  
0 CAM_ID   C!  
0 CTL_CAM0 C!  
0 CTL_CAM1 C!
```

STORE

Appendix B: Camera Controller DSP56001 Source Code

```

;; *****
;;*
;;*
;;*      aisDSP5.asm      12-1-91
;;*
;;*
;;*
;;*      DSP56001 Assembler source
;;*      for the Advanced Imaging System
;;*
;;*
;;*      Copyright 1990
;;*      Advanced Technologies Inc.
;;*      Peter Doherty
;;*
;;*
;;*      In order to operate both channels and read out through both channels,
;;*      certain limitations are placed on the read format. All subarrays are to
;;*      be centered on the CCD.
;;*      That simplifies things greatly, and the only issue now is direction of
;;*      charge transfer.
;;*
;;*
define      revision      '#>57'

;;*
;;*      Include Files are Used to DEFINE and EQUate constants
;;*      to be used in this source file.
;;*
nolist
include    'aisequ57.h'      ; equates for i/o locations
list
include    'aisdsp57.h'      ; equates for read data storage

include    'aisdsp57.mac'    ; macro definitions

;; definitions of camera pointers

define out_store 'x:(r3)'
define id_latch  'y:(r4)'
define ser_latch 'y:(r5)'
define out_latch 'y:(r6)'
define ana_latch 'y:(r7)'

;; Interrupt Service routine table
;;      This "Table" actually contains executable code.
;;      See DS56001 User's Manual chapter 8 for a discussion of
;;      exception processing.

org      p:$0000

;; *****
;; interrupt routine table
;; this "vector table" actually holds executable code
;;

jmp      startup      ;;; cmd #0 : reset vector
nop
nop      ;;; cmd #1 : stack error
nop
nop      ;;; cmd #2 : trace interrupt
nop
nop      ;;; cmd #3 : software interrupt
nop
nop      ;;; cmd #4 : irq*
nop
nop      ;;; cmd #5 : irqb*
jsr      sayOK        ;;; cmd #6 : DSP "OK" routine for system test
jsr      cam_write    ;;; cmd #7 : write host data to DSP address

```

```

jsr    cam_read      ;; cmd #8 : fetch data from DSP address for host
jsr    volt_test     ;; cmd #9 :
jsr    cam_ctl       ;; cmd #A : write to CAM control latch
jsr    init_volts    ;; cmd #B :
jsr    init_states   ;; cmd #C :
jsr    mytest        ;; cmd #D : a word I use for testing stuff
nop
nop                ;; cmd #E :
nop
nop                ;; cmd #F :
nop
nop                ;; cmd #10 :
nop
nop                ;; cmd #11 :

```

;; Host interrupt routine table

```

nop
nop                ;; cmd #12: used as SHADE by ATC5 cameras

jsr    init_format   ;; cmd #13: initialize readout format
jsr    read          ;; cmd #14: read the CCD
jsr    clear         ;; cmd #15: clear the ccd
jsr    pix_bin       ;; cmd #16: bin N pixels
jsr    row_bin       ;; cmd #17: bin N rows
jsr    pix_discard   ;; cmd #18: discard N pixels
jsr    row_discard   ;; cmd #19: discard N rows
jsr    pix_read      ;; cmd #1a: read N pixels
jsr    row_read      ;; cmd #1b: read N rows
nop
nop                ;; cmd #1c: unused
jsr    shift_IS      ;; cmd #1d: shift image to storage (unimplemented)
jsr    abloom        ;; cmd #1e: perform clock recomb. anti-blooming
nop
nop                ;; cmd #1f:

```

;; *****

```

; * startup
; * entry from dsp reset
startup
memfill
move    #$200,r0
jset    #3,x:hsr,init ;; if HF0 is set, go initialize
jclr    #hrdf,x:hsr,memfill ;; else if no new host data, loop
movep   x:hrx,a1      ;; else get the host data
movem   a1,p:(r0)+    ;; store program word
jmp     memfill

init
movec   #$0200,SR     ;; min int level set to 2
movep   #0,x:bcr      ;; all external mem access
bset    #10,x:ipr     ;; Host int set to ...
bset    #11,x:ipr     ;; priority level 2
bset    #0,x:psc      ;; set port b as Host port

;; we use portc2/sclk as enable for camera diff drivers
;; so we need to initialize it properly here.
;; actually it doesn't make any difference.

bset    #2,x:pcd      ;; cam_en low = camera enabled
movep   #$4,x:pcddr   ;; set pc2/sclk as output
movep   #$1f8,x:pcc   ;; set sci pins as i/o, ssi as ssi

move    #i_latch,r4
move    #o_store,r3
move    #s_latch,r5
move    #o_latch,r6
move    #a_latch,r7

clr     a
move    a1,x:initialized ;; we haven't been initialized yet

```



```

        next_ana
_sw     sw_lo           ;; summing well
        sw_hi
        do             #5,_pixel      ;; five states after summing well
        next_ana
_pixel nop
_ser_read
        rts

```

```

;; *****
; ser_clear
;;     clears the serial register
;;
;;     destroys          A
;;
;;     expects           nothing
;;
;;     calls              ser_discard
;;
;;     depends on        nothing
;;
;;     includes           nothing

```

```

ser_clear
        jclr          #format,x:initialized,_ser_clear
        move          x:ccd_ser,A
        jsr           ser_discard
_ser_clear
        rts

```

```

;; *****
; ser_discard  discard A1 pixels from serial register
;;
;;
;;     destroys          r0
;;
;;     expects           A1 = number of pixels to discard
;;
;;     calls              nothing
;;
;;     depends on        nothing
;;
;;     includes           rst_hi, rst_lo, sw_hi, sw_lo, ser_shift
;;

```

```

ser_discard
        jclr          #states,x:initialized,_ser_discard
        tst           A
        jeq           _ser_discard
        rst_hi
        sw_hi
        do             A1,_ser_dis     ;; shift pixels
_ser_dis ser_shift
        sw_lo
        sw_hi           ;; clear out the summing well
;;     rst_lo           ;; don't bother going low
_ser_discard

```

```

        rts

;; *****
; par_read      read A1 rows off the CCD
par_read
        jclr      #format,x:initialized,_par_read
        tst       A                ;; if no rows to read...
        jeq       _par_read        ;; .. then do nothing.
        do        A1,_par_read
        move      #>1,a
        jsr       par_bin          ;; bin one row
        move      x:pre_ser,a
        jsr       ser_discard      ;; serial pre-prescan
        move      x:under_ser,a
        jsr       ser_read         ;; serial underscan
        move      x:org_ser,a
        jsr       ser_discard      ;; serial prescan
        move      x:read_ser,a
        jsr       ser_read         ;; serial read
        move      x:post_ser,a
        jsr       ser_discard      ;; serial postscan
        move      x:over_ser,a
        jsr       ser_read         ;; serial overscan
        nop
_par_read
        rts

;; *****
; par_discard   discard the number of rows in acc A off the current ccd
;
;
; destroys r0
;
; expects A1 = number of rows to discard
;
; calls par_shift, ser_clear
;
; depends B,x0
;
par_discard
        tst       a                ;; if there's nothing to do...
        jeq       _clear          ;; ... do nothing.

;;
all_cams                ;; enable all CAMs
        move      #$aaaa,r0
        move      r0,ser_latch    ;; set all clocks to high
        rst_hi
        jsr       par_shift       ;; shift parallels
        jsr       ser_clear       ;; clear the serial register
        jsr       ser_idle        ;; put serials back to idle
        rst_lo
        nop
_clear
        rts                ;; return

```



```

        move      (r1)-                ;; point at last entry
        move      #$ffff,n1           ;; increment = -1
_init_cam1

        move      #par_wait0,r2       ;; r2 points at parallel delay table
        move      #p_latch,r7        ;; use r7 as pointer, must restore later

        do        x:n_par_states,_par_loop

;; move the next state for cam0
        cam0_only
        jclr     #tg_state_bit,x:(r0),_cam0_tg_lo ;; if the next state doesn't
        tg_hi    ;; include transfer gate high..
        jmp     _cam0_next_p         ;;.. don't set it high.
_cam0_tg_lo
        tg_lo
_cam0_next_p
        move     x:(r0)+n0,a1
        move     a1,y:(r7)           ;; move a state, r7 points at latch

;; move the next state for cam1
        cam1_only
        jclr     #tg_state_bit,x:(r1),_cam1_tg_lo ;; if the next state doesn't
        tg_hi    ;; include transfer gate high..
        jmp     _cam1_next_p         ;;.. don't set it high.
_cam1_tg_lo
        tg_lo
_cam1_next_p
        move     x:(r1)+n1,a1
        move     a1,y:(r7)           ;; move a state

        rep     x:(r2)+              ;; do a wait
        nop
_par_loop
        nop
_par_shift
        all_cams
        move     #a_latch,r7         ;; restore a_latch ptr
        clamp_lo ;; turn off the preamp clamp

        rep     #100                 ;; and hold it that way
        nop                               ;; for a long while

        rts

;; *****
;; cmd 7
;; cam_write accept data from host
;; and write it to a DSP address
;; expects address, then address space, then data
;;
;; destroys a, r0
;;
;; expects          nothing
;;
;; calls            host2a
;;
;; depends          nothing
;;
;; includes         nothing
;;
cam_write
        busy
        jsr     host2a                ;; get the address
        move     a1,r1
        jsr     host2a                ;; get address space 0=X, 1=Y
        move     a,b
        jsr     host2a                ;; get the data
        tst     b                     ;; test address space ID
        jeq     _writeX              ;; if space = 1

```

```

        move    a1,y:(r1)                ;; ... write to Y ram
_writeX  jmp     _cam_write              ;; ... then return
        ;; else
        move    a1,x:(r1)                ;; ... write to X ram
_cam_write
        rti                               ;; return
    
```

;; *****

```

; cmd 8
;
; cam_read  get data for host from DSP address
;           expects address, then address space, returns data
;
; destroys a, r1
;
; expects  nothing
;
; calls   a2host, host2a
;
; depends nothing
;
; includes nothing
;
cam_read
    busy
    jsr    host2a                ;; get the address
    move   a1,r1
    jsr    host2a                ;; get address space 0=X, 1=Y
    tst    a                      ;; test address space ID
    jeq    _readX                ;; if space = 1
    move   y:(r1),a1             ;; ... read from Y ram
    jmp    _cam_read            ;; ... then return
_readX   ;; else
        move   x:(r1),a1         ;; ... read from X ram
_cam_read
    jsr    a2host                ;; send data to host
    rti                          ;; return
    
```

;; *****

```

; cmd 9 : volt_test
;
; test the DAC's on the current CAM
;
; destroys a, r0, x1
;
; expects
;
; calls
;
; depends
;
; includes
;
volt_test
;
; clr      a
; move    #>1,x1
; move    #$fff,x0
;;_test1  move    #par_lo,r0      ;; point at first address
; rep     #24
; move    a1,y:(r0)+            ;; write to seq addresses
; add     x1,a                  ;; increment a
; rep     x0
; nop
;;_volt_test
; jmp     _test1                ;; go do it some more
; rti
    
```

;; THERE'S NO WAY OUT OF HERE !!!!

;; *****

```

; cmd 0x0A :   cam_ctl
;;
;;   write to the CAM control latch of current CAM
;;
;;   destroys  a, r0, x1
;;
;;   expects
;;
;;   calls
;;
;;   depends
;;
;;   includes
;;
cam_ctl
  move    #c_latch,r0      ;; point at control latch
  jsr    host2a           ;; get the datum
  move    a1,y:(r0)       ;; store it
  rti

;; *****
;; cmd 0x0B :   init_volts
;;
;;   gets the ccd voltages from the Host and writes them to the DACs.
;;   see aisdsp.h for the order in which they must arrive.
;;
;;   destroys  r1
;;
;;   expects  nothing
;;
;;   calls    host2a
;;
;;   depends  r1
;;
;;   includes nothing
;;

init_volts
  busy
  movep   #f, x:bcr        ;; setup wait states
  move    #par_lo,r1      ;; point r1 at par_lo
  do      #n_volts,_volts ;; do loop n_volts times
  jsr    host2a
  move    a1,y:(r1)+      ;; set DAC, increment DAC ptr

_volts
  movep   #0,x:bcr        ;; back to 0 wait states

  all_cams
  bset   #voltages,x:initialized
  rti

;; *****
;; HV3:   init_states
;;
;;   gets parrallel states and waits, serial states,
;;   and analog states and waits from Host. stores them
;;   in x memory
;;
;;   destroys  r1
;;
;;   expects  nothing
;;
;;   calls    host2a
;;
;;   depends  r1
;;
;;   includes nothing
;;

```

```

init_states
    busy

    jsr    host2a                ;; get the number of Parallel states
    move   a1,x:n_par_states
    move   a1,b0

    move   #cam0_par_state0,r1   ;; get the states for cam0
    do     b0,_cam0
    jsr    host2a
    move   a1,x:(r1)+
_cam0

    move   #cam1_par_state0,r1   ;; get the states for cam1
    do     b0,_cam1
    jsr    host2a
    move   a1,x:(r1)+
_cam1

    move   #par_wait0,r1        ;; get the parallel clock delays
    do     b0,_par_waits
    jsr    host2a
    move   a1,x:(r1)+
_par_waits

    jsr    host2a                ;; get the number of serial states
    move   a1,x:n_ser_states
    move   #ser_state0,r1
    do     a1,_ser_init          ;; get the serial states
    jsr    host2a
    move   a1,x:(r1)+          ;; ... one at a time.
_ser_init

    jsr    host2a                ;; get the number of analog states
    move   a1,x:n_ana_states
    move   #ana_state0,r1
    jsr    state_loop

    jsr    host2a                ;; get the number of ab states
    move   a1,x:n_ab_states
    move   #ab_state0,r1
    jsr    state_loop

    bset   #states,x:initialized  ;; states are initialized now
    rti

;; *****
;; cmd 12 :  shade
;;
;; used by ATC5 cameras, not implemented in AIS or AIS2 cameras
shade
    rti

;; *****
;; cmd 13 :  init_format
;;
;; acquire a new set of readout data from Host
;;
;; including:
;;
;; ccd_ser ; serial register length
;; bin_ser ; serial binning number

```

```

;;      pre_ser  ; serial register extension length
;;      under_ser ; serial register underscan
;;      org_ser  ; serial prescan length
;;      read_ser ; serial read length
;;      post_ser ; serial postscan length
;;      over_ser ; serial register overscan

;;      ccd_par  ; parallel register length
;;      bin_par  ; parallel binning number
;;      org_par  ; parallel prescan length
;;      read_par ; parallel read length
;;      post_par ; parallel postscan length
;;      over_par ; parallel register overscan
;;
;;      pdir_CAM0 ; parallel shift direction for CAM0
;;      pdir_CAM1 ; parallel shift direction for CAM1
;;
;;      n_fmt_params      equ      18
;;
;;
;;      destroys  r1
;;
;;      expects   nothing
;;
;;      calls     host2a
;;
;;      depends   r1
;;
;;      includes  nothing

init_format
    busy

    move    #ccd_ser,r1
    do      #n_fmt_params,_format
    jsr     host2a
    move    a1,x:(r1)+

_format
    bset   #format,x:initialized
    rti

;; *****
;; cmd 14:  read
;;
;;      reads the ccd out depending on the format parameters
;;
;;      destroys  A
;;
;;      expects   nothing
;;
;;      calls     par_discard, ser_clear, pbin, ser_discard, ser_read
;;
;;      depends   nothing
;;
;;      includes  nothing
;;

read
    busy                                ;; set dsp busy flag

    jsr     par_idle                      ;; put parallels into idle state
    jsr     ser_idle                      ;; put serials into idle state

    jcir   #format,x:initialized,_read

    move    x:org_par,a                  ;; parallel prescan
    jsr     par_discard

    jsr     ser_clear                    ;; clear ser reg again

```

```

        move    x:read_par,a           ;; read rows
        jsr    par_read

        move    x:post_par,a          ;; parallel postscan
        jsr    par_discard

_read   rti
    
```

```

;; *****
; cmd 15 : clear
;;
;; clear the ccd imager
;;
;; destroys A
;;
;; expects nothing
;;
;; calls par_dis
;;
;; depends nothing
;;
;; includes nothing
;;
    
```

```

clear   busy

        jsr    par_idle           ;; put parallels into idle state
        jsr    ser_idle          ;; put serials into idle state

        jclr   #format,x:initialized,_clear
        move   x:ccd_par,A
        jsr    par_discard

_clear  rti
    
```

```

;; *****
; cmd 16 : pix_bin
;;
;; bin N pixels
;;
;; destroys a1
;;
;; expects nothing
;;
;; calls host2a
;;
;; depends nothing
;;
;; includes sw_hi,ser_shift
;;
    
```

```

pix_bin busy
        jclr   #states,x:initialized,_pix_bin

        jsr    host2a
        do     a1,_pix_bin
        ser_shift

_pix_bin rti
    
```

```
;; *****  
; cmd 17 : row_bin  
;;  
;; shift N rows  
;;  
;; destroys a1,A1  
;;  
;; expects nothing  
;;  
;; calls host2a, par_shift  
;;  
;; depends nothing  
;;  
;; includes nothing  
;;  
;;
```

```
row_bin  
    busy  
    jsr    host2a  
    jsr    par_shift  
_row_bin  
    rti
```

```
;; *****  
; cmd 18 : pix_discard  
;;  
;; discard N pixels  
;;  
;; destroys A  
;;  
;; expects nothing  
;;  
;; calls host2a, ser_discard  
;;  
;; depends nothing  
;;  
;; includes nothing  
;;  
;;
```

```
pix_discard  
    busy  
    jsr    host2a  
    jsr    ser_discard  
_pix_discard  
    rti
```

```
;; *****  
; cmd 19 : row_discard  
;;  
;; discard N rows  
;;  
;; destroys A  
;;  
;; expects nothing  
;;  
;; calls par_discard, ser_clear  
;;  
;; depends nothing  
;;  
;; includes nothing  
;;  
;;
```

```
row_discard  
    busy  
    jsr    host2a  
    jsr    par_discard  
_row_discard  
    rti
```

```

;; *****
; cmd 1a : pix_read
;;
;; read N pixels
;;
;; destroys A
;;
;; expects nothing
;;
;; calls host2a, ser_read
;;
;; depends nothing
;;
;; includes nothing
;;
pix_read
    busy
    jsr    host2a
    jsr    ser_read
_pix_read
    rti

```

```

;; *****
; cmd 1B : row_read
;;
;; read N rows
;;
;; destroys
;;
;; expects nothing
;;
;; calls host2a, par_bin, ser_discard, ser_read
;;
;; depends
;;
;; includes
;;
row_read
    busy
    jsr    host2a
    jsr    par_read
_row_read
    rti

```

```

;; *****
; cmd 1c : unused

```

```

;; *****
; cmd 1d : shift_IS
;;
;; shift image to storage on a frame transfer device
;;
;; unimplemented
shift_IS
    rti

```

```

;; *****
; cmd 0x1e : abloom perform clock recombination anti blooming
;;
;; destroys

```

```

;;
;; expects
;;
;; calls
;;
;; depends
;;
;; includes
;;
;;
abloom
    jcir    #states,x:initialized,_abloom
    jcir    #voltages,x:initialized,_abloom
    all_cams
    move    #p_latch,r1
_ab_prep
    move    #ab_state0,r0
    do      x:n_ab_states,_ab_loop
    move    x:(r0)+,x0          ;; ab state 0 to x0
    move    x0,y:(r1)          ;; then to parallel clock latch
    rep     x:(r0)+
    nop
_ab_loop
    jset    #continue,x:hsr,_ab_prep    ;; do clock recomb. anti-blooming?
;;
no_cams

_abloom
    rti

```

```

;; *****
;; *****

```

```

;* state_loop
; assumes x0 holds the number of states and waits to get
;;
;; destroys r1
;;
;; expects a1 = number of state/wait pairs to get
;;         r1 = pointer to state/wait table
;;
;; calls host2a
;;
;; depends r1
;;
;; includes nothing
;;
state_loop
    do      a1,_state_loop
    jsr     host2a          ;; get a state
    move    a1,x:(r1)+
    jsr     host2a          ;; get a wait time
    move    a1,x:(r1)+
_state_loop
    rts

```

```

;; *****
;
; host2a
;;         Waits for data from the Host
;;         Moves the data into a
;;
;; destroys a
;;
;; expects nothing
;;
;; calls nothing
;;
;; depends nothing
;;
host2a

```

```

        clr      a
        jclr    #hrdf,x:hsr,host2a
        movep   x:hrx,a1      ;; get the data into a1
        rts

;; *****
;; a2host
;;           Moves data from register a1 to the Host
;;
;; destroys  nothing
;;
;; expects  a1 to hold useful data
;;
;; calls    nothing
;;
;; depends  nothing
;;
;; note : this is not presently used, but this is how you'd do it.
;;
a2host
        jclr    #htde,x:hsr,a2host
        movep   a1,x:htx
        rts

;; *****
;; par_idle
;;           sets the parallel clocks to the idle state
;;
;; destroys  x0,r0,a
;;
;; expects
;;
;; calls    nothing
;;
;; depends  nothing
;;

par_idle
        jclr    #states,x:initialized,_par_idle
        move    #p_latch,r1

        cam0_only      ;; enable cam0
        move    #cam0_par_state0,r0 ;; r0 points at parallel state table
        rep     x:n_par_states
        move    (r0)+      ;; r0 now points at place PAST table
        move    (r0)-      ;; r0 now points to last place IN table
        nop
        move    x:(r0),a1   ;; put that state into a
        move    a1,y:(r1)   ;; write the idle state to cam0

        cam1_only      ;; enable cam1 only
        move    #cam1_par_state0,r0 ;; r0 points at parallel state table
        rep     x:n_par_states
        move    (r0)+      ;; r0 now points to place PAST table
        move    (r0)-      ;; r0 now points to last place IN table
        nop
        move    x:(r0),a1   ;; move that state to a
        move    a1,y:(r1)   ;; write the idle state to cam1

        all_cams
_par_idle
        rts

;; *****
;; ser_idle
;;           sets the serial clocks to the idle state
;;
;; destroys  x0,r0,a
;;

```

```
;; expects
;;
;; calls nothing
;;
;; depends nothing
;;

ser_idle
    jclr    #states,x:initialized,_ser_idle
    all_cams
    move    #ser_state0,r0
    rep     x:n_ser_states
    move    (r0)+
    move    (r0)-
    move    x:(r0),a
    move    a1,ser_latch

_ser_idle
    rts

;; *****
; cmd 6 sayOK respond to 6811 for test purposes
;; allows 6811 to test if DSP is operating properly
sayOK
    clr     a
    move    #$4f,a1
    jsr    a2host           ;; say 'O'
    move    #$4b,a1
    jsr    a2host           ;; say 'K'
    move    revision,a1
    jsr    a2host           ;; pass revision number
    rti
```

```
;; aisdsp51.mac
;; macro definitions for use with aisdsp51.asm
```

```
host_int_enable MACRO
    bset    #2,x:hcr
ENDM
```

```
host_int_disable MACRO
    bclr   #2,x:hcr
ENDM
```

```
;; NOTE :
;; in all of these macros it is assumed that r3 points to out_store,
;; the internal memory location we use to store the last value written
;; to the output latch on the clock card.
```

```
;; rst_hi
;;
;; set the CCD reset gate high
rst_hi MACRO
    bset    #rst_bit,out_store
    move    out_store,y0
    move    y0,out_latch
ENDM
```

```
;; rst_lo
;;
;; set the CCD reset gate low
rst_lo MACRO
    bclr   #rst_bit,out_store
    move    out_store,y0
    move    y0,out_latch
ENDM
```

```
;; sw_hi
;;
;; set the CCD summing well high
sw_hi MACRO
    bset    #sw_bit,out_store
    move    out_store,y0
    move    y0,out_latch
ENDM
```

```
;; sw_lo
;;
;; set the CCD summing well low
sw_lo MACRO
    bclr   #sw_bit,out_store
    move    out_store,y0
    move    y0,out_latch
ENDM
```

```
;; tg_hi
;;
;; set the CCD transfer gate high
tg_hi MACRO
    bset    #tg_bit,out_store
    move    out_store,y0
    move    y0,out_latch
ENDM
```

```
;; tg_lo
;;
;; set the CCD transfer gate low
tg_lo MACRO
    bclr   #tg_bit,out_store
```

```

        move    out_store,y0
        move    y0,out_latch
        ENDM

;; clamp_lo
;;
;; set the preamp clamp bit low (clamp off)
clamp_lo    MACRO
        bclr    #clamp_bit,out_store
        move    out_store,y0
        move    y0,out_latch
        ENDM

;; clamp_hi
;;
;; set the preamp clamp bit high (clamp on)
clamp_hi    MACRO
        bset    #clamp_bit,out_store
        move    out_store,y0
        move    y0,out_latch
        ENDM

;; ser_shift
;;
;; shift one serial pixel
;; operates the same on all cams simultaneously
ser_shift    MACRO
        move    #ser_state0,r0
        nop
        move    x:(r0)+,a
        do      x:n_ser_states,_s_shift
        move    x:(r0)+,a  a,ser_latch
_s_shift
        nop
        ENDM

;; sbin
;;
;; bin one serial superpixel
sbin    MACRO
        do      x:bin_ser,_s_bin_loop
        ser_shift
_s_bin_loop
        ENDM

;; next Analog state macro
;; destroys      A
;; expects r1 points at next analog state
;;
next_ana    MACRO
        move    x:(r1)+,A1          ;; astate in
        move    A1,ana_latch        ;; analog state out
        rep    x:(r1)+              ;; perform await
        nop
        ENDM

;; busy flag for host
busy    MACRO
        bset    #3,x:hcr            ;; set the DSP busy flag
        ENDM

```

```
;; macros for selecting clock/analog modules
```

```
all_cams      MACRO
               move #>$0,y0
               move y0,id_latch
               ENDM
```

```
no_cams       MACRO
               move #>$ff,y0
               move y0,id_latch
               ENDM
```

```
cam0_only     MACRO
               move #>$fe,y0
               move y0,id_latch
               ENDM
```

```
cam1_only     MACRO
               move #>$fd,y0
               move y0,id_latch
               ENDM
```

```
;;cam2_only   MACRO
;;             move #>$fb,y0
;;             move y0,id_latch
;;             ENDM
;;
```

```
;;cam3_only   MACRO
;;             move #>$f7,y0
;;             move y0,id_latch
;;             ENDM
;;
```

```
;;cam4_only   MACRO
;;             move #>$ef,y0
;;             move y0,id_latch
;;             ENDM
;;
```

```
;;cam5_only   MACRO
;;             move #>$df,y0
;;             move y0,id_latch
;;             ENDM
;;
```

```
;;cam6_only   MACRO
;;             move #>$bf,y0
;;             move y0,id_latch
;;             ENDM
;;
```

```
;;cam7_only   MACRO
;;             move #>$7f,y0
;;             move y0,id_latch
;;             ENDM
;;
```

```
;; end of macro definitions
```

```

; motorola standard i/o equates.
;
;*****
;
;   equates for dsp56000 i/o registers and ports
;
;*****
;-----
;
;   equates for i/o port programming
;
;-----
;
;   register addresses
bcr equ    $fffe      ; port a bus control register
pbc equ    $ffe0      ; port b control register
pbddr equ  $ffe2      ; port b data direction register
pbd equ    $ffe4      ; port b data register
pcc equ    $ffe1      ; port c control register
pcddr equ  $ffe3      ; port c data direction register
pcd equ    $ffe5      ; port c data register

;-----
;
;   equates for host interface
;
;-----
;
;   register addresses
hcr equ    $ffe8      ; host control register
hsr equ    $ffe9      ; host status register
hrx equ    $ffeb      ; host receive data register
htx equ    $ffeb      ; host transmit data register

;   host control register bit flags
hrie equ    0          ; host receive interrupt enable
htie equ    1          ; host transmit interrupt enable
hcie equ    2          ; host command interrupt enable
hf2 equ    3          ; host flag 2
hf3 equ    4          ; host flag 3

;   host status register bit flags
hrdf equ    0          ; host receive data full
htde equ    1          ; host transmit data empty
hcp equ    2          ; host command pending
hf equ    $18         ; host flag mask
hf0 equ    3          ; host flag 0
hf1 equ    4          ; host flag 1
dma equ    7          ; dma status

;-----
;
;   equates for serial communications interface (sci)
;
;-----
;
;   register addresses
srxl equ    $fff4      ; sci receive data register (low)
srxm equ    $fff5      ; sci receive data register (middle)
srxh equ    $fff6      ; sci receive data register (high)
stxl equ    $fff4      ; sci transmit data register (low)
stxm equ    $fff5      ; sci transmit data register (middle)
stxh equ    $fff6      ; sci transmit data register (high)
stxa equ    $fff3      ; sci transmit data address register
scr equ    $fff0      ; sci control register
ssr equ    $fff1      ; sci status register
sccr equ    $fff2      ; sci clock control register

```

```

; sci control register bit flags
wds equ    $3          ; word select mask
wds0 equ    0          ; word select 0
wds1 equ    1          ; word select 1
wds2 equ    2          ; word select 2
sbk equ    4          ; send break
wake equ    5          ; wake-up mode select
rwi equ    6          ; receiver wake-up enable
woms equ    7          ; wired-or mode select
re equ     8          ; receiver enable
te equ     9          ; transmitter enable
ilie equ   10         ; idle line interrupt enable
rie equ   11         ; receive interrupt enable
tie equ   12         ; transmit interrupt enable
tmie equ   13         ; timer interrupt enable

; sci status register bit flags
trne equ    0          ; transmitter empty
tdre equ    1          ; transmit data register empty
rdrf equ    2          ; receive data register full
idle equ    3          ; idle line
or equ     4          ; overrun error
pe equ     5          ; parity error
fe equ     6          ; framing error
r8 equ     7          ; received bit 8

; sci clock control register bit flags
cd equ     $fff        ; clock divider mask
cod equ    12         ; clock out divider
scp equ    13         ; clock prescaler
rcm equ    14         ; receive clock source
tcm equ    15         ; transmit clock source

-----
;
; equates for synchronous serial interface (ssi)
;
-----

; register addresses
rx equ     $ffef      ; serial receive data register
tx equ     $ffef      ; serial transmit data register
cra equ     $ffec      ; ssi control register a
crb equ     $ffed      ; ssi control register b
str equ     $ffee      ; ssi status register
tsr equ     $ffee      ; ssi time slot register

; ssi control register a bit flags
pm equ     $ff        ; prescale modulus select mask
dc equ     $1f00      ; frame rate divider control mask
wl equ     $6000      ; word length control mask
wl0 equ    13         ; word length control 0
wl1 equ    14         ; word length control 1
psr equ    15         ; prescaler range

; ssi control register b bit flags
of equ     $3          ; serial output flag mask
of0 equ    0          ; serial output flag 0
of1 equ    1          ; serial output flag 1
scd equ    $1c        ; serial control direction mask
scd0 equ   2          ; serial control 0 direction
scd1 equ   3          ; serial control 1 direction
scd2 equ   4          ; serial control 2 direction
sckd equ   5          ; clock source direction
fsl equ    8          ; frame sync length
syn equ    9          ; sync/async control
gck equ   10         ; gated clock control

```

```

mod equ 11 ; mode select
ste equ 12 ; ssi transmit enable
sre equ 13 ; ssi receive enable
stie equ 14 ; ssi transmit interrupt enable
srie equ 15 ; ssi receive interrupt enable

; ssi status register bit flags

if equ $2 ; serial input flag mask
if0 equ 0 ; serial input flag 0
if1 equ 1 ; serial input flag 1
tfs equ 2 ; transmit frame sync
rfs equ 3 ; receive frame sync
tue equ 4 ; transmitter underrun error
roe equ 5 ; receiver overrun error
tde equ 6 ; transmit data register empty
rdf equ 7 ; receive data register full

;-----
;
; equates for exception processing
;-----

; register addresses

ipr equ $ffff ; interrupt priority register

; interrupt priority register bit flags

ial equ $7 ; irqa mode mask
ial0 equ 0 ; irqa mode interrupt priority level (low)
ial1 equ 1 ; irqa mode interrupt priority level (high)
ial2 equ 2 ; irqa mode trigger mode
ibl equ $38 ; irqb mode mask
ibl0 equ 3 ; irqb mode interrupt priority level (low)
ibl1 equ 4 ; irqb mode interrupt priority level (high)
ibl2 equ 5 ; irqb mode trigger mode
hpl equ $c00 ; host interrupt priority level mask
hpl0 equ 10 ; host interrupt priority level mask (low)
hpl1 equ 11 ; host interrupt priority level mask (high)
;; ssl equ $3000 ; ssi interrupt priority level mask
ssl0 equ 12 ; ssi interrupt priority level mask (low)
ssl1 equ 13 ; ssi interrupt priority level mask (high)
scl equ $c000 ; sci interrupt priority level mask
scl0 equ 14 ; sci interrupt priority level mask (low)
scl1 equ 15 ; sci interrupt priority level mask (high)

```

```

; aisdsp.h
; an equate file with definitions for the
; variable storage locations used in aisdsp.asm
;
    ORG    x:0

ccd_ser      ds    1    ; serial register length
bin_ser      ds    1    ; serial binning number
pre_ser      ds    1    ; serial register extension length
under_ser    ds    1    ; serial register underscan
org_ser      ds    1    ; serial prescan length
read_ser     ds    1    ; serial read length
post_ser     ds    1    ; serial postscan length
over_ser     ds    1    ; serial register overscan

ccd_par      ds    1    ; parallel register length
bin_par      ds    1    ; parallel binning number
org_par      ds    1    ; parallel prescan length
read_par     ds    1    ; parallel read length
post_par     ds    1    ; parallel postscan length
over_par     ds    1    ; parallel register overscan

pdir_cam0    ds    1    ; parallel shift direction for CAM0
pdir_cam1    ds    1    ; parallel shift direction for CAM1
n_fmt_params equ    16

    ORG    x:20

temp0        ds    1    ; a useful, very temporary storage location

;; the following location "initialized" is a set of bits that we can
;; test to see if various things have been initialized. We would not
;; want to manipulate parallel clocks or do a readout if the states
;; have not been initialized, for instance, we would not want to turn
;; the clocks on if they have not been initialized, nor would we
;; want to do a readout if the format has not been initialized.
;; bit 0 = states    bit 1 = format    bit 2 = voltages
;; bits will be set to a one when the related parameters are initialized
;; and will be cleared otherwise.

initialized   ds    1

states        equ    0
format        equ    1
voltages      equ    2

; storage for our state counts,
;; they must be here so we can use direct addressing
;
n_par_states  ds    1
n_ser_states  ds    1
n_ana_states  ds    1
n_ab_states   ds    1
o_store       ds    1    ; storage for output latch data

;
; parallel shift control constant storage locations
;
    ORG    x:$40

cam0_par_state0 ds    1
cam0_par_state1 ds    1
cam0_par_state2 ds    1
cam0_par_state3 ds    1
cam0_par_state4 ds    1
cam0_par_state5 ds    1
cam0_par_state6 ds    1
cam0_par_state7 ds    1
cam0_par_state8 ds    1

```

```
cam0_par_state9      ds    1
cam1_par_state0      ds    1
cam1_par_state1      ds    1
cam1_par_state2      ds    1
cam1_par_state3      ds    1
cam1_par_state4      ds    1
cam1_par_state5      ds    1
cam1_par_state6      ds    1
cam1_par_state7      ds    1
cam1_par_state8      ds    1
cam1_par_state9      ds    1
```

```
par_wait0            ds    1
par_wait1            ds    1
par_wait2            ds    1
par_wait3            ds    1
par_wait4            ds    1
par_wait5            ds    1
par_wait6            ds    1
par_wait7            ds    1
par_wait8            ds    1
par_wait9            ds    1
```

```
;  
;
```

```
; serial shift control constant storage locations
```

```
;
```

```
;; for now there will be NO serial wait states performed !!
```

```
ORG    x:$80
ser_state0          ds    1
ser_state1          ds    1
ser_state2          ds    1
ser_state3          ds    1
ser_state4          ds    1
ser_state5          ds    1
ser_state6          ds    1
ser_state7          ds    1
ser_state8          ds    1
ser_state9          ds    1
```

```
;  
;
```

```
; analog control constant storage locations
```

```
;
```

```
ORG    x:$a0
ana_state0          ds    1
ana_wait0           ds    1
ana_state1          ds    1
ana_wait1           ds    1
ana_state2          ds    1
ana_wait2           ds    1
ana_state3          ds    1
ana_wait3           ds    1
ana_state4          ds    1
ana_wait4           ds    1
ana_state5          ds    1
ana_wait5           ds    1
ana_state6          ds    1
ana_wait6           ds    1
ana_state7          ds    1
ana_wait7           ds    1
ana_state8          ds    1
ana_wait8           ds    1
ana_state9          ds    1
ana_wait9           ds    1
```

```
;; ab_states and ab_waits are constants used in  
;; clock recombination anti-blooming mode
```

```
;  
    ORG    x:$c0  
ab_state0 ds    1  
ab_wait0  ds    1  
ab_state1 ds    1  
ab_wait1  ds    1  
ab_state2 ds    1  
ab_wait2  ds    1  
ab_state3 ds    1  
ab_wait3  ds    1  
  
;; equates for off chip locations, including latches and dacs  
;;  
;; equates for clock board dacs  
n_volts   equ    24  
par_lo    equ    $ff08  
par_mid   equ    $ff09  
par_hi    equ    $ff0a  
ser_lo    equ    $ff0b  
ser_mid   equ    $ff0c  
ser_hi    equ    $ff0d  
tg_lo     equ    $ff0e  
tg_hi     equ    $ff0f  
sw_lo     equ    $ff10  
sw_hi     equ    $ff11  
sub       equ    $ff12  
lg        equ    $ff13  
rst_lo    equ    $ff14  
rst_hi    equ    $ff15  
vrd       equ    $ff16  
vod       equ    $ff17  
x1_lo     equ    $ff18  
x1_hi     equ    $ff19  
x2_lo     equ    $ff1a  
x2_hi     equ    $ff1b  
x3_lo     equ    $ff1c  
x3_hi     equ    $ff1d  
x4_lo     equ    $ff1e  
x4_hi     equ    $ff1f  
  
;; equates for clock board latches  
i_latch   equ    $ff00  
s_latch   equ    $ff20  
p_latch   equ    $ff24  
o_latch   equ    $ff28  
a_latch   equ    $ff2c  
c_latch   equ    $ff30  
  
; equate for bit numbers used in bitset commands  
; bits in output control latch  
tg_bit    equ    0  
sw_bit    equ    1  
rst_bit   equ    2  
clamp_bit equ    3  
extra2_bit equ    4  
extra3_bit equ    5  
extra4_bit equ    6  
  
;; equates for host control flags used to enable  
;; continuous clear and clock recombination  
;; anti-blooming  
cont_clear equ    3    ;; hf0  
continue   equ    4    ;; hf1  
  
;; equate for host flag 2  
;; used to tell the host when the DSP is busy  
bsy       equ    3    ;; hf2  
  
;; equate for the eighth bit in the parallel  
;; states. This bit tells the DSP when to
```

```
;; set the transfer gate high  
tg_state_bit equ 8  
clamp_state_bit equ 9
```

```
;; par_group is the number of parallel shifts  
;; to perform as a group in parallel discards  
par_group equ 10
```

Appendix C: VMEbus Interface Board Source Code

HEX
7400 TIB !
200 TIB 2+ !
7600 DP !

```
( ATDvme.FOR forth code to run on the ATD VME interface board )  
(  
( c Copyright 1991 )  
( Advanced Technologies Inc. )  
( P. Doherty, H. Meyer )  
( )  
( REV. 4 modified for three-letter command abbreviations )  
( )  
( REV. 3 changed the way the address switches work )  
( )  
( REV. 2 fixed up serial text data transfers )  
( added error checking )  
( added separate error register )  
( changed SEND_STRING function to one character at a time )  
( )  
( )  
( REV. 1 modified ATCVMER17.for vme code for ATC5 cameras )  
( so that it will work with both the ATC5 cameras and )  
( AIS style cameras running rev 5 or higher )  
( )  
( * ***** )
```

0 CONSTANT FALSE
1 CONSTANT TRUE

```
( * useful words for performing bit manipulations )  
( * these words operate on the number currently on the top of the stack.
```

```
( * MASK-X leaves true if bit x set else false
```

```
: MASK-7 0080 AND ;  
: MASK-6 0040 AND ;  
: MASK-5 0020 AND ;  
: MASK-4 0010 AND ;  
: MASK-3 0008 AND ;  
: MASK-2 0004 AND ;  
: MASK-1 0002 AND ;  
: MASK-0 0001 AND ;
```

```
( * T-X makes certain bit x is true regardless of current state )
```

```
: T-7 0080 OR ;  
: T-6 0040 OR ;  
: T-5 0020 OR ;  
: T-4 0010 OR ;  
: T-3 0008 OR ;  
: T-2 0004 OR ;  
: T-1 0002 OR ;  
: T-0 0001 OR ;
```

```
( * F-X makes certain bit x is false regardless of current state )
```

```
: F-7 FF7F AND ;  
: F-6 FFBF AND ;  
: F-5 FFDF AND ;  
: F-4 FFEF AND ;  
: F-3 FFF7 AND ;  
: F-2 FFFB AND ;  
: F-1 FFFD AND ;  
: F-0 FFFE AND ;
```

```
( ***** INTERRUPT READ ***** )  
( the following variables belong to the interrupt read routines )
```

```
B02D CONSTANT SCCR2 ( SCI control register 2 )  
B02E CONSTANT SCSR2 ( SCI status register 2 )  
B02F CONSTANT SCDR ( the read / write register )
```

```
B7BF CONSTANT SCI_INT_LINK ( SCI interrupt link )
```

```
VARIABLE WRITE_POINTER ( variables for read buffer )  
VARIABLE READ_POINTER  
VARIABLE BUFFER_BOTTOM  
VARIABLE BUFFER_TOP  
VARIABLE BUFFER_LENGTH
```

```
VARIABLE CR_FLAG  
VARIABLE QM_FLAG  
VARIABLE OK_FLAG
```

HERE BUFFER_BOTTOM !

FE ALLOT

HERE BUFFER_TOP !

(WRITE-BUFFER writes into textbuffer. It accepts characters until the buffer is full.)
(This routine is called by the interrupt read word)
(expects value on stack)

: WRITE-BUFFER

```
WRITE_POINTER @
DUP 1+ DUP BUFFER_TOP @ = IF BUFFER_LENGTH @ - THEN READ_POINTER @ = NOT
IF
C! WRITE_POINTER @ 1+          ( write data into buffer and advance pointer )
DUP BUFFER_TOP @ = IF BUFFER_LENGTH @ - THEN WRITE_POINTER ! ( wrap if buffer end )
ELSE
DROP DROP                      ( ignore data if buffer is full )
THEN
```

;

(READ-BUFFER reads from text buffer. If buffer is empty a zero is returned)
(It does not wait for the next available character if buffer is empty)
(If READ_POINTER @ and WRITE_POINTER @ are equal, the buffer is empty)
(returns value on stack)

: READ-BUFFER

```
READ_POINTER @
DUP ( 1+ DUP BUFFER_TOP @ = IF BUFFER_LENGTH @ - THEN ) WRITE_POINTER @ = NOT
IF
C@ READ_POINTER @ 1+          ( read data from buffer and advance pointer )
DUP BUFFER_TOP @ = IF BUFFER_LENGTH @ - THEN READ_POINTER ! ( wrap if buffer end )

ELSE
DROP 0                        ( return a zero if buffer is empty )
THEN
```

;

: CLEAR-FLAGS

```
0 OK_FLAG !                    ( initialize flags )
0 CR_FLAG !
0 QM_FLAG !
```

;

(INT-RECEIVE is an interrupt driven routine, called after receiving serial data)
(It scans the incoming datastream for the occurrences of ' OK', ' ?', CRLF.)
(The flags QM_FLAG, OK_FLAG and CR_FLAG indicate that the corresponding words were)
(detected. As the flags are internally also used for the detecting process, they must)
(be checked for a certain value: QM_FLAG = 2)
(OK_FLAG = 3)
(CR_FLAG = 2 are the values, when the string is detected)
(Zero values for the flags indicates that the string has not occurred.)
(Any other value is not a valid flag.)
(Once a flag is set, it will stay set. The checking routine must reset the flags.)

CODE INT-RECEIVE

(by the way: the following three modules are independend and can be
(copied for other applications. But dont forget to add the RTS then

(first we read the value from the interface and put it on the forth data stack

```
36 C,      ( PUSHA      * SECURE REGISTERS
37 C,      ( PUSHB
3C C,      ( PUSHX
86 C, 00 C, ( LDAA #0    * FORTH WANTS 16 BIT-> SO CLEAR MSB
F6 C, SCSR2 , ( LDAB SCSR2 * CLEAR INTERRUPT REQUEST
F6 C, SCDR , ( LDAB SCDR  * GET DATA FROM INTERFACE
18 C, 09 C, ( DEY        * PUT DATA ON FORTH DATA STACK
18 C, 09 C, ( DEY
18 C, ED C, 00 C, ( STD 0,Y
```

(then we check for all the flags
(assumes testvalue on FORTH data stack - BUT it will NOT remove it

```
F6 C, QM_FLAG 1+ , ( qmtest  ldab qmflag  * is QM_FLAG =0 ?
26 C, 0E_C,      (      bne qmtest
18 C, EC C, 00 C, (      ldd 0,y    * copy value on data stack but dont remove it
C1 C, 20 C,      (      cmpb #' '   * is it the first testvalue ?
26 C, 29 C,      (      bne oktest * go to next test
```

```

C6 C, 01 C, ( ldab #1 * set QM_FLAG to 1
F7 C, QM_FLAG 1+ , ( stab qmflag
20 C, 22 C, ( bra oktest * go to next test
C1 C, 01 C, ( qmltest cmpb #1 * is QM_FLAG =1 ?
26 C, 1E C, ( bne oktest
18 C, EC C, 00 C, ( ldd 0,y * copy value on data stack but dont remove it
C1 C, 3F C, ( cmpb #'?' * is it the second testvalue ?
26 C, 07 C, ( bne qmlnext * test for another space (leading spaces?)
C6 C, 02 C, ( ldab #2 * set QM_FLAG to 2
F7 C, QM_FLAG 1+ , ( stab qmflag
20 C, 10 C, ( bra oktest * go to next test
C1 C, 20 C, ( qmlnext cmpb #' '
26 C, 07 C, ( bne qmzero
C6 C, 01 C, ( ldab #1 * set QM_FLAG to 1
F7 C, QM_FLAG 1+ , ( stab qmflag
20 C, 05 C, ( bra oktest * go to next test
C6 C, 00 C, ( qmzero ldab #0 * reset flag
F7 C, QM_FLAG 1+ , ( stab qmflag
F6 C, OK_FLAG 1+ , ( oktest ldab okflag * is OK_FLAG =0 ?
26 C, 0E C, ( bne okltest
18 C, EC C, 00 C, ( ldd 0,y * copy value on data stack but dont remove it
C1 C, 4F C, ( cmpb #'0' * is it the first testvalue ?
26 C, 36 C, ( bne crtest * go to next test
C6 C, 01 C, ( ldab #1 * set QM_FLAG to 1
F7 C, OK_FLAG 1+ , ( stab okflag
20 C, 2F C, ( bra crtest * go to next test
C1 C, 01 C, ( okltest cmpb #1 * is OK_FLAG =1 ?
26 C, 14 C, ( bne ok2test
18 C, EC C, 00 C, ( ldd 0,y * copy value on data stack but dont remove it
C1 C, 4B C, ( cmpb #'K' * is it the second testvalue ?
26 C, 07 C, ( bne oklnext * test for another space (leading spaces?)
C6 C, 02 C, ( ldab #2 * set QM_FLAG to 2
F7 C, OK_FLAG 1+ , ( stab okflag
20 C, 1D C, ( bra crtest * go to next test
C1 C, 4F C, ( oklnext cmpb #'0'
26 C, 14 C, ( bne okzero
20 C, 17 C, ( bra crtest * go to next test
C1 C, 02 C, ( ok2test cmpb #2 * is OK_FLAG =2 ?
26 C, 13 C, ( bne crtest
18 C, EC C, 00 C, ( ldd 0,y * copy value on data stack but dont remove it
C1 C, 0D C, ( cmpb #'CR' * is it the third testvalue ?
26 C, 07 C, ( bne okzero * go to next test
C6 C, 03 C, ( ldab #3 * set OK_FLAG to 3
F7 C, OK_FLAG 1+ , ( stab okflag
20 C, 05 C, ( bra crtest * go to next test
C6 C, 00 C, ( okzero ldab #0 * reset flag
F7 C, OK_FLAG 1+ , ( stab okflag
F6 C, CR_FLAG 1+ , ( crtest ldab crflag * is CR_FLAG =0 ?
26 C, 0E C, ( bne crltest
18 C, EC C, 00 C, ( ldd 0,y * copy value on data stack but dont remove it
C1 C, 0D C, ( cmpb #SOD * is it the first testvalue ?
26 C, 1E C, ( bne exit * go to exit
C6 C, 01 C, ( ldab #1 * set QM_FLAG to 1
F7 C, CR_FLAG 1+ , ( stab crflag
20 C, 17 C, ( bra exit * go to next test
C1 C, 01 C, ( crltest cmpb #1 * is CR_FLAG =1 ?
26 C, 13 C, ( bne exit
18 C, EC C, 00 C, ( ldd 0,y * copy value on data stack but dont remove it
C1 C, 0A C, ( cmpb #SOA * is it the first testvalue ?
26 C, 07 C, ( bne crzero * go to exit
C6 C, 02 C, ( ldab #2 * set CR_FLAG to 2
F7 C, CR_FLAG 1+ , ( stab crflag
20 C, 05 C, ( bra exit * go to next test
C6 C, 00 C, ( crzero ldab #0 * reset flag
F7 C, CR_FLAG 1+ , ( stab crflag
01 C, ( nop * jump address for 'exit'

```

(then we write it into the FIFO
 (assumes data on the FORTH data stack.

```

FE C, WRITE_POINTER , ( ldx writepoint
FC C, WRITE_POINTER , ( ldd writepoint * load write pointer
C3 C, 00 C, 01 C, ( addd #1 * inc write pointer
1A C, B3 C, BUFFER_TOP , ( cpd buffertop
26 C, 03 C, ( bne label * no correction
B3 C, BUFFER_LENGTH , ( subd buflen * subtract buffer length
1A C, B3 C, READ_POINTER , ( label cpd readpoint * is read pointer ahead?
27 C, 12 C, ( beq full
36 C, ( psha * preserve address for later use
37 C, ( pshb
18 C, EC C, 00 C, ( ldd 0,y * get data from data stack
18 C, 08 C, ( iny
18 C, 08 C, ( iny

```

```

E7 C, 00 C,      (      stab 0,x      * store data (LSB only)
33 C,           (      pulb
32 C,           (      pula      * restore incremented pointer
FD C, WRITE_POINTER , (      std  writepoint
20 C, 07 C,     (      bra  exit
18 C, EC C, 00 C, ( full ldd 0,y      * take data, but dont use them
18 C, 08 C,     (      iny
18 C, 08 C,     (      iny
38 C,           ( exit pulx      * restore registers
33 C,           (      pulb
32 C,           (      pula
3B C,           (      rti

```

END-CODE

```

: KEY-BUF
  BEGIN
  READ_POINTER @ WRITE_POINTER @ - NOT
  UNTIL
  READ-BUFFER
  ;

```

```

CODE-SUB CLEAR-CC-MASKS
  86 C, 00 C,      ( LDAA # 0 )
  06 C,           ( TAP )
  39 C,           ( RTS )
END-CODE

```

```

: INIT-BUFFER
  BUFFER_BOTTOM @ DUP WRITE_POINTER ! READ_POINTER ! ( initialize buffer )
  BUFFER_TOP @ BUFFER_BOTTOM @ - BUFFER_LENGTH !
  0 OK_FLAG ! ( initialize flags )
  0 CR_FLAG !
  0 QM_FLAG !
  ;

```

```

: START-IRQ
  SCI_INT_LINK C@ 7E - NOT
  IF
  7E SCI_INT_LINK EEC!
  THEN ( JMP code )
  [ ' INT-RECEIVE @ > FF AND ] LITERAL DUP SCI_INT_LINK 1+ C@ -
  IF
  DROP
  ELSE
  SCI_INT_LINK 1+ EEC! ( hi byte )
  THEN
  [ ' INT-RECEIVE @ FF AND ] LITERAL DUP SCI_INT_LINK 2+ C@ -
  IF
  DROP
  ELSE
  SCI_INT_LINK 2+ EEC! ( lo byte )
  THEN
  CLEAR-CC-MASKS
  INIT-BUFFER ( initialize buffer )
  SCCR2 C@ T-5 SCCR2 C! ( enable interrupt )
  ;

: STOP-IRQ
  SCCR2 C@ F-5 SCCR2 C! ( disable interrupt )
  ; ( links may stay as they are )

```

```

( ***** TIMER WORDS ***** )
( the following are some generally useful words for timing things.)
( the 68hc11 has an on board counter/timer which runs off the )
( system clock. we can use this timer for timing things like )
( exposures. it is a 16 bit counter running at 2 mhz.

```

```

( twait waits a given number of timer ticks )
HEX
CODE-SUB T-WAIT
  18 C, EC C, 00 C, ( LDD 0,Y )
  18 C, 08 C,      ( INY )
  18 C, 08 C,      ( INY )
  F3 C, B00E ,    ( ADDD TCNT )
  1A C, B3 C, B00E , ( LOOP CPD TCNT )
  24 C, FA C,      ( BCC LOOP )
  39 C,
END-CODE

```

```
( M-WAIT expects a number on the stack. it waits that many milli- )
( seconds. the number 1870, which you might expect to be )
( 2000, is adjusted downwards to account for the overhead )
( involved in getting in and out of the routine. it was )
( found by experiment, and may need adjustment. )
```

```
: M-WAIT ( W - ) ( WAIT W MILLISECONDS )
  0 DO 74E T-WAIT LOOP ;
```

```
: MS-WAIT ( D - ) ( WAIT D MILLISECONDS, D IS A DOUBLE )
  DUP 0= IF DROP ELSE
  0 DO FFFF M-WAIT 1 M-WAIT LOOP THEN
  DUP 0= IF DROP ELSE M-WAIT THEN ;
```

```
( ***** EEPROM WORDS ***** )
( The forth dictionary is stored in eeprom memory, but is run out )
( of ram. At any time the user may use the following word "store" )
( to move the current dictionary to eeprom. This, in and of itself, )
( is not all that useful. The next word, "RESTOR", will move the )
( dictionary from eeprom to ram, and restore the state of the )
( forth system to exactly where you were when you issued the store )
( command. This is also of limited use, because next time you )
( power up the system, RESTOR is not available in ram to to be )
( called. Another word, "ASTART!" defined at the end of this file )
( is used to tell the 6811 what word we would like executed on the )
( next restart. "RESTOR" can be used as the autostart routine. )
( Other words which include restore may also be defined as auto- )
( start words, but the routine name is hard coded and this file )
( would have to be edited. A detailed discussion of the whole )
( autostart process can be found in the New MICROS manuals. )
```

```
( eeprom represents where in eeprom to start storing the dictionary )
( we want to reserve some space for autostart routines etc so we )
( start at 256 bytes above the actual start of the eeprom or $1200 )
```

```
1100 CONSTANT EPROM
1204 CONSTANT EEDICT-START
7600 CONSTANT DICT-START
```

```
: EE-! 2DUP ! BEGIN 2DUP @ = UNTIL DROP DROP ;
: EE-C! 2DUP C! BEGIN 2DUP C@ = UNTIL DROP DROP ;
```

```
CODE-SUB EEPROT
  CE C, 55 C, 55 C, ( LDX #$5555 - 1ST EPROM ADDRESS )
  86 C, AA C, ( LDAA #$AA - 1ST DATA PATTERN )
  A7 C, 00 C, ( STAA 0,X - SEND AN AA TO $5555 )
  CE C, 2A C, AA C, ( LDX #$2AAA - 2ND EPROM ADDRESS )
  86 C, 55 C, ( LDAA #$55 - 2ND DATA PATTERN )
  A7 C, 00 C, ( STAA 0,X - SEND A 55 TO $2AAA )
  CE C, 55 C, 55 C, ( LDX #$5555 - 3RD EPROM ADDRESS )
  86 C, A0 C, ( LDAA #$A0 - 3RD DATA PATTERN )
  A7 C, 00 C, ( STAA 0,X - SEND AN A0 TO $5555 )
  39 C, ( RTS - RETURN )
END-CODE
```

```
CODE-SUB EEUNPROT
  CE C, 55 C, 55 C, ( LDX #$5555 - 1ST EPROM ADDRESS )
  86 C, AA C, ( LDAA #$AA - 1ST DATA PATTERN )
  A7 C, 00 C, ( STAA 0,X - SEND AN AA TO $5555 )
  CE C, 2A C, AA C, ( LDX #$2AAA - 2ND EPROM ADDRESS )
  86 C, 55 C, ( LDAA #$55 - 2ND DATA PATTERN )
  A7 C, 00 C, ( STAA 0,X - SEND A 55 TO $2AAA )
  CE C, 55 C, 55 C, ( LDX #$5555 - 3RD EPROM ADDRESS )
  86 C, 80 C, ( LDAA #$80 - 3RD DATA PATTERN )
  A7 C, 00 C, ( STAA 0,X - SEND AN 80 TO $5555 )
  CE C, 55 C, 55 C, ( LDX #$5555 - 4TH EPROM ADDRESS )
  86 C, AA C, ( LDAA #$AA - 4TH DATA PATTERN )
  A7 C, 00 C, ( STAA 0,X - SEND AN AA TO $5555 )
  CE C, 2A C, AA C, ( LDX #$2AAA - 5TH EPROM ADDRESS )
  86 C, 55 C, ( LDAA #$55 - 5TH DATA PATTERN )
  A7 C, 00 C, ( STAA 0,X - SEND A 55 TO $2AAA )
  CE C, 55 C, 55 C, ( LDX #$5555 - 6TH EPROM ADDRESS )
  86 C, 20 C, ( LDAA #$20 - 6TH DATA PATTERN )
  A7 C, 00 C, ( STAA 0,X - SEND A 20 TO $5555 )
```

39 C,
END-CODE

(RTS - RETURN)

VARIABLE PROMLOC
VARIABLE PROMCOUNT

HEX

```
: STORE
  0 PROMCOUNT !
  EEUNPROT
  EPROM 100 + PROMLOC !
  HERE PROMLOC @ EE-!
  PROMLOC @ 2+ PROMLOC !
  DICT-START PROMLOC @ EE-!
  PROMLOC @ 2+ PROMLOC !

  CR ." STORING DICTIONARY " CR
  HERE DICT-START DO
  I C@ DUP PROMLOC @ C@ - IF DROP
    ELSE PROMLOC @ EE-C! PROMCOUNT @ 1+ PROMCOUNT !
    THEN

  PROMLOC @ 1+ PROMLOC !
  LOOP

  ." STORING FORTH USER AREA " CR

  84 6 DO
  I C@ PROMLOC @ EE-C!
  PROMLOC @ 1+ PROMLOC !
  LOOP

  EEPROT
  CR ." STORE COMPLETE " CR
  ." dictionary length: " HERE 7600 - U. CR
  ." bytes updated: " PROMCOUNT @ U. CR
  EEPROT ;
```

```
: RESTOR
  ( CR ." RESTORING THE DICTIONARY " CR )
  1204 ( start of dictionary in EEPROM )
  7600 ( start of dictionary in RAM )
  1202 @ 1202 @ ( fetch dictionary end and start )
  - ( compute length )
  CMOVE ( move it to RAM )

  ( ." RESTORING FORTH USER AREA " CR )
  1200 @ ( fetch dictionary end address )
  1202 @ ( fetch dictionary start address )
  - ( compute length )
  1204 + ( add eeprom dictionary offset )
  ( which points us to the eeprom user area )
  6 84 CMOVE ( move it into ram )

  ( CR ." RESTORE STATE COMPLETE " CR )
  ;
```

(***** MAILBOX WORDS *****)
(DEFINITIONS OF PT-VSI MAILBOXES)

HEX
D200 CONSTANT MB0
D201 CONSTANT MB1
D202 CONSTANT MB2
D203 CONSTANT MB3
D204 CONSTANT MB4
D205 CONSTANT MB5
D206 CONSTANT MB6
D207 CONSTANT MB7
D208 CONSTANT MB8
D209 CONSTANT MB9
D20A CONSTANT MB10
D20B CONSTANT MB11
D20C CONSTANT MB12
D20D CONSTANT MB13
D20E CONSTANT MB14
D20F CONSTANT MB15

D218 CONSTANT MB24
D219 CONSTANT MB25
D21A CONSTANT MB26
D21D CONSTANT MB29

CODE-SUB MB@

```

0F C,      ( SEI          - do not disturb (disable int.) )
CE C, B00A , ( LDX PORTE - PUT ADDRESS OF PORT E INTO X )
A6 C, 00 C, ( LDAA 0,X   - READ PORT E )
84 C, 80 C, ( ANDA #$80 - MASK OUT ALL BUT BIT 7 )
26 C, FA C, ( BNE #$FA  - IF NOT 0 GO BACK TO LDAA )
18 C, EC C, 00 C, ( LDD 0,Y   - GET ADDRESS OF MAILBOX )
18 C, 08 C, ( INY       - VALUE ON THE STACK IS 16 BITS )
18 C, 08 C, ( INY       - SO INCREMENT THE SP TWICE )
8F C,      ( XGDX      - PUT ADDRESS INTO INDEX REG X )
3C C,      ( PSHX      - SAVE ADDRESS FOR FUTURE USE )
A6 C, 00 C, ( LDAA 0,X   - READ MAILBOX, IGNORE THE VALUE )
CE C, B00A , ( LDX PORTE - GET ADDRESS OF PORT E AGAIN )
A6 C, 00 C, ( LDAA 0,X   - READ PORT E )
84 C, 80 C, ( ANDA #$80 - MASK OUT ALL BUT BIT 7 )
27 C, FA C, ( BEQ #$FA  - IF NOT 1 GO BACK TO LDAA )
38 C,      ( PULX      - GET THE MAILBOX ADDRESS AGAIN )
A6 C, 00 C, ( LDAA 0,X   - READ THE MAILBOX AGAIN )
16 C,      ( TAB       - TRANSFER A TO B TO ORIENT FOR STACK )
4F C,      ( CLRA      - CLEAR ACCUMULATOR A )
18 C, 09 C, ( DEY       - VALUE TO BE PUT ON THE STACK IS 16 )
18 C, 09 C, ( DEY       - BITS SO DECREMENT THE SP TWICE )
18 C, ED C, 00 C, ( STD 0,Y   - STORE THE DATA ON THE STACK )
0E C,      ( CLI       - interrupts enabled
39 C,      ( RTS          - RETURN )

```

END-CODE

CODE-SUB MB!

```

0F C,      ( SEI          - do not disturb (disable int.) )
CE C, B00A , ( LDX PORTE - PUT ADDRESS OF PORT E INTO X )
A6 C, 00 C, ( LDAA 0,X   - READ PORT E )
84 C, 80 C, ( ANDA #$80 - MASK OUT ALL BUT BIT 7 )
26 C, FA C, ( BNE #$FA  - IF NOT 0 GO BACK TO LDAA )
18 C, EC C, 00 C, ( LDD 0,Y   - GET ADDRESS OF THE MAILBOX )
18 C, 08 C, ( INY       - VALUE ON THE STACK IS 16 BITS )
18 C, 08 C, ( INY       - SO INCREMENT THE SP TWICE )
8F C,      ( XGDX      - PUT THE ADDRESS INTO INDEX REG X )
18 C, EC C, 00 C, ( LDD 0,Y   - GET DATA TO PUT INTO THE MAILBOX )
18 C, 08 C, ( INY       - VALUE ON THE STACK IS 16 BITS )
18 C, 08 C, ( INY       - SO INCREMENT THE SP TWICE )
E7 C, 00 C, ( STAB 0,X  - WRITE DATA TO THE MAILBOX )
0E C,      ( CLI       - interrupts enabled
39 C,      ( RTS          - RETURN )

```

END-CODE

(***** status LEDs *****)
(LIGHT THE LED'S ONE AT A TIME INDICATING LIFE)

D700 CONSTANT LIGHTS
20 CONSTANT MAXCOUNT
80 CONSTANT MAXVAL

VARIABLE LCOUNT
VARIABLE LIGHTVAL

: INITLEDS

```

0 LIGHTS C!
0 LCOUNT !
1 LIGHTVAL C! ;

```

: HEARTBEAT

```

LIGHTVAL C@
DUP LIGHTS C!
LCOUNT @ DUP MAXCOUNT -
IF DROP 0 LCOUNT !
  DUP MAXVAL -
  IF DROP 1 LIGHTVAL C!
  ELSE 2 * LIGHTVAL C!
  THEN
ELSE 1+ LCOUNT ! DROP
THEN ;

```

(***** VMEbus Interface and PTVSI CHIP *****)

```
( Setting the address of the ATDVME interface
(
( Definitions of the switches are as follows :
(
(
(   1  2  3  4  5  6  7  8  9 10 11 12
(
(   MSB ----- LSB -----
(           ADDRESS                               first two digits in address of DRAM
(
(           DRAM ARRAY SIZE _____ size of DRAM 00 = 4M, 01 = 8M
(                                           10 = 16M, 11 = 32M
(
(           ADDRESS SPACE _____ "off" = 24 bit, "on" = 32 bit
(
(           NO-START OPTION _____ "off" = do not start
(
(
( Switches 1 through 8 are used for the first two digits
( in the address of the VME board. This address represents
( the address of the first location in the DRAM interface.
( The controller portion of the board is accessed immediatly
( above the DRAM array.
(
( Switches 9 and 10 determine how much memory space the
( board will occupy. Presumably this is how much RAM actually
( exists on the board, but that is not necessary.
(
( Switch 11 which address space the board will respond
( to and hence which digits in the 32 bit address are set
( according to the switches.
(
( Switch 12 is used as a "go / no-go" option. If the
( board is reset with this bit set "off", the on-board
( microcontroller will initialize the bus interface, but will
( not run the camera control loop. It will instead look to the
( serial interface for FORTH commands.
(
```

HEX
(many of the registers in the bus uinterface chip are initialized the
(same no matter what configuration.

```
: INIT-PTVSI
  0 MB0 MB!   0 MB1 MB!   0 MB2 MB!
  0 MB3 MB!   0 MB4 MB!   0 MB5 MB!
  0 MB6 MB!   0 MB7 MB!   0 MB8 MB!
  0 MB9 MB!   0 MB10 MB!  0 MB11 MB!
  0 MB12 MB!  0 MB13 MB!  0 MB14 MB!
  0 MB15 MB!  0 D210 MB!  0 D211 MB!
  0 D212 MB!  0 D213 MB!  0 D214 MB!
  0 D215 MB!  9 D216 MB!  34 D217 MB!
  5 D21B MB! ;
```

(The bits on the DIPSWITCH and on PORTE are inverted, so we
(always end up either inverting them or using them in a way that
(may seem backwards. Keep that in mind.

D600 CONSTANT DIPSWITCH
B00A CONSTANT PORTE

(Temporary storage locations for what will ultimately be written
(into registers 25 and 29 of the PT-VSI bus interface chip.
(Temporary storage required because we cannot read back from
(the PT-VSI and we may need to set bits in these registers
(as we initialize the bus interface.

VARIABLE T25
VARIABLE T29

(report impossible addressing combination to the user
: ADDR-ERROR

```

BEGIN                ( forever ...
  00 LIGHTS C!      (   turn lights on
  400 M-WAIT        (   wait 1 second
  FF LIGHTS C!      (   turn lights off
  400 M-WAIT        (   wait 1 second
  FALSE            (   leave false
UNTIL ;             ( go do it again

```

```

( setup bus interface for 24 bit address space
: SETUP-24

```

```

FF MB24 MB!
DIPSWITCH C@ FF XOR      ( fetch address bits and invert them
DUP 4 / DUP T25 C! T29 C! ( right shift by two store temp values
T29 C@ F0 AND T29 C!     ( zero bottom four bits of T29
40 * MB26 MB!           ( left shift addr by 6 and store in reg 26

PORTE C@ FF XOR          ( fetch control bits and invert them
OC AND                   ( mask off memory size bits
DUP 0 = IF
  T25 C@ T-4 T25 C!      ( ... set bit 4 in T25
  THEN
DUP 4 = IF
  T25 C@ T-5 T25 C!      ( if 8 meg ...
  T29 C@ 0 OR T29 C!     ( ... set bit 1 in T29
  THEN
DUP 8 = IF
  ADDR-ERROR            ( if 16 meg ...
  THEN                  ( ... that's a fatal error in 24 bit space
  C = IF
  ADDR-ERROR            ( if 32 meg ..
  THEN                  ( ... that's also fatal.

T25 C@ MB25 MB!         ( store the temp values to mailboxes 25, 29
T29 C@ MB29 MB!

90 D21F MB! ;          ( call it 24 bit mode and enable DRAM

```

```

( setup bus interface for 32 bit address space
: SETUP-32

```

```

DIPSWITCH C@ FF XOR      ( fetch address bits and invert them
DUP 4 / MB24 MB!         ( right shift by two and store

40 * DUP T25 C! T29 C!   ( left shift by 6 and store temp values
T29 C@ F0 AND T29 C!     ( zero bottom four bits of T29
0 MB26 MB!              ( zero MB26

PORTE C@ FF XOR          ( fetch control bits and invert them
OC AND                   ( mask off memory size bits
DUP 0 = IF
  T25 C@ T-4 T25 C!      ( ... set bit 4 in T25
  THEN
DUP 4 = IF
  T25 C@ T-5 T25 C!      ( if 8 Meg ...
  T29 C@ 01 OR T29 C!    ( ... set bit 5 in T25
  THEN
DUP 8 = IF
  T25 C@ T-6 T25 C!      ( if 16 Meg ...
  T29 C@ 03 OR T29 C!    ( ... set bits 0 and 1 in T29
  THEN
  C = IF
  T25 C@ T-7 T25 C!      ( if 32 Meg ..
  T29 C@ 07 OR T29 C!    ( ... set bit 7 in T25
  THEN                  ( ... set bits 0,1,2 in T29

T25 C@ MB25 MB!         ( store the temp values to mailboxes 25, 29
T29 C@ MB29 MB!

80 D21F MB! ;          ( call it 32 bit mode and enable DRAM

```

```

( GO_FOR_IT will be checked later to
( determine if we should enter command loop

```

```

VARIABLE GO_FOR_IT

```

```

: SETUP

```

```

INIT-PTVSI
PORTE C@ 01 AND
IF FALSE GO FOR IT !    ( if PORTE bit 0 = "off" then we will not start

```

```

ELSE TRUE GO_FOR_IT !      ( if PORTE bit 0 = "on" then we will
THEN
PORTE C@ 02 AND            ( check address space bit
  IF SETUP-24             ( if PORTE bit 1 = "off" then 24 bit space
  ELSE SETUP-32           ( if PORTE bit 1 = "on" then 32 bit space
  THEN
C0 D21E MB! ;             ( enable dram array select pin

```

```
( ***** COMMUNICATION ***** )
```

```
( the general command process for this card is as follows:
( all commands are prefixed with their parameters.
( the 6811 waits for data or commands by polling MB0
```

```
( MB0 is used as the command status register for master slave communication
```

```
( bit definitions :
```

```
(
( 76543210
( |||||+----- CRDY    command ready
( |||||+----- MSDF    master slave data flag
( ||||+----- BSY     busy
( |||+-----
( ||+-----
( |+-----
( |-----
( +-----
(
(

```

```
( MB1 and MB2 are the MSDR master/slave data register
( Date may be of two types. Either parameter data to be placed on
( the stack for later use by a command. Or command number data, a number
( representing a command the master wishes executed.
( Both types of data is assumed to be 16 bit with the upper portion
( in MB2 and the lower portion in MB1.
```

```
( MB3 contains the count of characters recieved from the camera
( available for the host.
( the count is updated after every transfer
```

```
( MB4 is used for the text transfer from the VME board to the host
```

```
( MB5 is used as the ERROR STATUS REGISTER.
( the local slave sets bits in this register
( when it recognizes errors
```

```
( errors returned include :
( 76543210
( |||||+----- CMD-ERR   : command number out of range
( |||||+----- STACK-ERR : stack underflow condition
( ||||+----- LOST-DATA  : buffer overflow during text transfer to host
( |||+-----
( ||+-----
( |+-----
( |-----
( +----- QM-ERR : error flag caused by a question mark
```

```
( Please note : It is requested that the host cpu not write to the
( error status register. A command is provided for
( resetting the register to 0. If that command fails
( the board is not functioning properly and as a last
( resort a software reset is recommended.
```

```
( MB6 is used as the status word for slave master communication.
( The master sets the SMDR flag to indicate it is ready to accept data from slave.
( The slave provides a character in MB4 and the count of available characters in MB3.
( It then resets the SMDR.
```

```
(
( 76543210
( |||||+----- SMDR    : slave master data request
( |||||+-----
( ||||+-----
( |||+-----
( ||+-----
( |+-----
( |-----
( +-----

```

```
(      |+-----  
(      +-----
```

```
( BSY sets the busy flag in MB0 )  
: BSY 4 MB0 MB! ;
```

```
( RDY clears MB0 )  
: RDY 0 MB0 MB! ;
```

```
( MSDF? checks the MSDF flag in MB0 to see if data has been placed in MB1 )  
: MSDF? MB0 MB@ 2 = ;
```

```
( CRDY? checks the CRDY flag in MB0 to see if a command is ready to be executed )  
: CRDY? MB0 MB@ 1 = ;
```

```
( SMDR? is issued by the host when it is ready to accept text )  
: SMDR? MB6 MB@ 1 AND 1 = ;
```

```
( SMRDY is the ready flag for the slave to master transfer )  
: SMRDY 0 MB6 MB! ;
```

```
( * COMMANDS TO SET THE ERROR STATUS REGISTER BITS )
```

```
( CMD-ERR sets the command range error flag  
( indicating that the VME board thinks that the  
( command number passed with the last CRDY flag was invalid  
HEX  
: CMD-ERR MB5 MB@ 01 OR MB5 MB! ;
```

```
( STACK-ERR sets the stack underflow condition flag  
( indicating that the VME board has noticed that it does not  
( have adequate data on the stack to perform a requested  
( operation. NOTE : the VME board does not attempt to  
( execute the command, and leaves the stack untouched.  
HEX  
: STACK-ERR MB5 MB@ 02 OR MB5 MB! ;
```

```
( QM-ERR sets the question mark error flag  
( indicating that the camera has returned a question mark and  
( does not recognize a command that has been issued )  
HEX  
: QM-ERR MB5 MB@ 80 OR MB5 MB! ;
```

```
( LOST-DATA sets a flag indicating that an overflow occurred during text transfer  
( from slave to host. If this flag is set, the host was not fast enough  
( to get all data from slave  
HEX  
: LOST-DATA MB5 MB@ 40 OR MB5 MB! ;
```

```
( BUFFER-COUNT gets the number of characters stored in the buffer )  
( Leaves count value on stack )  
: BUFFER-COUNT  
WRITE_POINTER @ READ_POINTER @ -  
DUP 0 < IF BUFFER_LENGTH @ + THEN  
DUP BUFFER_LENGTH @ 1 - = IF LOST-DATA THEN  
;
```

```
( SERVE-HOST checks if host is ready to accept data. If it is ready, the data are  
( transferred via MB4 and the count of the remaining data is stored in MB3
```

```
: SERVE-HOST  
SMDR? ( check if host is ready for data  
IF  
BUFFER-COUNT ( if data are available in buffer  
DUP 0 = NOT  
IF  
READ-BUFFER MB4 MB! ( store data in mailbox  
1 - ( decrement BUFFER-COUNT  
THEN  
MB3 MB! ( store count in mailbox  
SMRDY ( issue the ready flag  
ELSE
```

```
    BUFFER-COUNT
    MB3 MB!           ( store count in mailbox
THEN ;
```

```
: WAIT-MSDF
  BEGIN
  SERVE-HOST
  MSDF?
  UNTIL ;
```

```
( ***** COMMAND EXECUTION ***** )
( DOCMD execute the command corresponding to the number )
( retrieved from the host in the mailbox )
```

```
CODE-SUB DOCMD
 18 C, EC C, 00 C, ( LDD 0,Y get the command number )
 18 C, 08 C,      ( INY      the value on the stack is 16 bits )
 18 C, 08 C,      ( INY      so inCRement the sp twice )
 05 C,           ( LSLD     multiply the number by two )
 C3 C, 7000 ,    ( ADDD CTAB add the command table start address )
 8F C,           ( XGDX     put the address into the index reg )
 EC C, 00 C,     ( LDD 0,X load the cfa into the d accumulator )
 BD C, ATO4 ,    ( JSR ATO4 jump to the command table routine )
 39 C,           ( RTS      return )
END-CODE
```

```
HEX
```

```
( WAIT-CR-LF WAITS UNTIL A CARRIAGE RETURN LINEFEED PAIR HAS BEEN RECEIVED )
```

```
: WAIT-CR-LF
  BEGIN
  CR_FLAG @ 2 = DUP IF 0 CR_FLAG ! THEN
  UNTIL
  ;
```

```
( HEX
: WAIT-OK ( wait until an " 'ok' cr lf " has been received )
  BEGIN
  OK_FLAG @ 3 = DUP IF 0 OK_FLAG ! THEN ( check if OK occurred
  QM_FLAG @ 2 = OR ( or the QM flag was set
  SERVE-HOST
  UNTIL
  WAIT-CR-LF
  ;
```

```
( ***** commands ***** )
```

```
( ***** functions called by the host to operate this board ***** )
```

```
( AG-INIT initialize the "address generator" PLD which controls
( the DMA writes of camera data into the dual ported buffer
( Calling this function performs no useful function other
( than assuring that the next image data will begin at address 0
HEX
D400 CONSTANT AG1
D401 CONSTANT AG2
: AG-INIT 0 AG2 C! 3 AG2 C! ;
```

```
( SPCON-RESET resets the " serial to parallel converter pld
( which identifies the bursts of pixel clocks on the AIS style
( interface cards. performs no useful function whatsoever on
( the ATDVME1 parallel data interface cards
HEX
D500 CONSTANT SPCON
D800 CONSTANT ISPCON
```

```

: SPCON-RESET    FF ISPCON C! ;
: SPCON-INIT     BSY SPCON C! RDY ;

( I/O Port routines :
( The board contains a parallel I/O port accessible to the host
( through a set of commands.
( The 6811's built in I/O port PORT A is used
HEX
B000 CONSTANT PORTA

( IO-READ  read the 3 input pins on the user io port, bits 0-2 )
: IO-READ
    PORTA C@ 07 AND MB2 MB!
    WAIT-OK
    ;

( IO-WRITE  write the 4 output pins of the user io port, bits 3-7 )
(          expects the value to be on the stack
: IO-WRITE
    DEPTH 1 <
    IF STACK-ERR
    ELSE 78 AND PORTA C! WAIT-OK
    THEN ;

( IO-PULSE  pulse pin 1 of the user io port for x milliseconds )
(          expects the number of milliseconds to be on the stack )
: IO-PULSE
    DEPTH 1 <
    IF STACK-ERR
    ELSE 08 PORTA C! M-WAIT 00 PORTA C! WAIT-OK
    THEN ;

( CAM-HALT  stops the command loop
(          it is only a dummy word as the code is detected before we get here
(          the dummy word is needed to have an entry in the CMD table
: CAM-HALT ;

( ERROR-RESET resets the error flags
: ERROR-RESET 0 MB5 MB! ;

: FLUSH-BUFFER                ( empties the buffer )
    STOP-IRQ
    MB5 MB@ F-6 MB5 MB!      ( reset any old LOST-DATA bit )
    START-IRQ ;

: NOT-DEFINED CMD-ERR ;

( ***** functions called by the host to operate the camera ***** )
HEX
: CAM-RESTART
    ." CXX " OD EMIT
    FFF M-WAIT
    FLUSH-BUFFER
    ." CIN " OD EMIT
    WAIT-OK
    ;

: SET-PARAM  ( set the value of a camera control parameter )
    DEPTH 2 <
    IF STACK-ERR
    ELSE
        U. U. ." FP! " OD EMIT
        WAIT-OK
    THEN ;

: CISC-ON      ( begin continuous clear option )
    ." SCT" OD EMIT
    WAIT-OK
    ;

: CISC-OFF     ( terminate continuous clear option )

```

```

." SCF" OD EMIT
WAIT-OK
;

: GAIN-LO      ( set camera gain to low )
." SGL" OD EMIT
WAIT-OK
;

: GAIN-HI      ( set camera gain to high )
." SGH" OD EMIT
WAIT-OK
;

: OFFSET!      ( write value to CCD signal offset DAC )
DEPTH 1 <
IF STACK-ERR
ELSE
    U. ." SCO" OD EMIT
    WAIT-OK
THEN ;

: SHADE        ( produce test pattern )
." IAS" OD EMIT
WAIT-OK
;

: OSHUT        ( open the shutter )
." SSO" OD EMIT
WAIT-OK
;

: CSHUT        ( close the shutter )
." SSC" OD EMIT
WAIT-OK
;

: PIX-BIN      ( shift N pixels into the summing well or output )
DEPTH 1 <
IF STACK-ERR
ELSE
    U. ." LPB" OD EMIT
    WAIT-OK
THEN ;

: ROW-BIN      ( shift N rows into the serial register )
DEPTH 1 <
IF STACK-ERR
ELSE
    U. ." LRB" OD EMIT
    WAIT-OK
THEN ;

: PIX-DISCARD  ( discard N pixels off the CCD )
DEPTH 1 <
IF STACK-ERR
ELSE
    U. ." LPC" OD EMIT
    WAIT-OK
THEN ;

: ROW-DISCARD  ( discard N rows off the CCD )
DEPTH 1 <
IF STACK-ERR
ELSE
    U. ." LRC" OD EMIT
    WAIT-OK
THEN ;

: PIX-READ     ( read N pixels off the CCD )
DEPTH 1 <
IF STACK-ERR

```

```

ELSE
  U. ." LPR" OD EMIT
  WAIT-OK
THEN ;

: ROW-READ      ( read N rows off the CCD )
  DEPTH 1 <
  IF STACK-ERR
  ELSE
    U. ." LRR" OD EMIT
    WAIT-OK
  THEN ;

: CAM-WRITE
  DEPTH 1 <
  IF STACK-ERR
  ELSE
    U. U. ." LCW" OD EMIT
    WAIT-OK
  THEN ;

: CLEAR        ( clear the ccd )
  ." ICL" OD EMIT
  WAIT-OK
;

: READ         ( read out the ccd )
  ." IRD" OD EMIT
  WAIT-OK
;

: BIAS         ( close, clear, read )
  ." IAB" OD EMIT
  WAIT-OK
;

: XPOSE        ( close, clear, integrate )
  ." IIL" OD EMIT
  WAIT-OK
;

: DARK         ( CLOSE, CLEAR, INTEGRATE, READ )
  ." IAD" OD EMIT
  WAIT-OK
;

: OBS          ( CLOSE, CLEAR, OPEN, INTEGRATE, CLOSE, READ )
  ." IAL" OD EMIT
  WAIT-OK
;

: TEMPO@      ( produce formatted string reporting temperature of CCD )
  ." STB" OD EMIT
  WAIT-OK
;

: TEMPl@      ( produce formatted string reporting temperature of CASE )
  ." STC" OD EMIT
  WAIT-OK
;

: FORMAT?     ( produce formatted table of camera parameters )
  ." FF?" OD EMIT
  WAIT-OK
;

: SET-FORMAT   ( initialize camera with current format parameters )
  ." FSF" OD EMIT
  WAIT-OK
;

( ANTI-BLOOM      these are to be replaced by a parameter )
: ANTI-BLOOM-OFF
  ." SBF" OD EMIT
  WAIT-OK
;

: ANTI-BLOOM-ON

```

```
." SBT" OD EMIT
WAIT-OK
;
```

```
: INTEGRATE-CCD ( stop clearing, integrate charge on the CCD )
." IID" OD EMIT
WAIT-OK
;
```

```
: CAM-FAST ( set camera speed to fast )
." SSF" OD EMIT
WAIT-OK
;
```

```
: CAM-SLOW ( set camera speed to slow )
." SSS" OD EMIT
WAIT-OK
;
```

```
: FRAME-TRANSFER
." SFT" OD EMIT
WAIT-OK
;
```

```
: FULL-FRAME
." SFF" OD EMIT
WAIT-OK
;
```

```
: SET-TEMP
DEPTH 1 <
IF STACK-ERR
ELSE
    U. ." ST! " OD EMIT
    WAIT-OK
THEN ;
```

VARIABLE TEMP_POINTER

```
: SEND-STRING ( get a character from the vme master and send to the camera )
DEPTH 1 <
IF STACK-ERR
ELSE
    WRITE_POINTER @ TEMP_POINTER ! ( store old pointer )
    EMIT
    BEGIN
    WRITE_POINTER @ TEMP_POINTER @ = NOT ( wait until pointer advances )
    UNTIL
THEN ;
```

(BUILD BUILD A JUMP TABLE OF COMMANDS IN RAM)
7000 CONSTANT CTAB

```
: BUILD
( functions related to initialization and operation of this board )
[ ' AG-INIT ] LITERAL CFA CTAB ! ( COMMAND OFFSET 0x00 )
[ ' SPCON-INIT ] LITERAL CFA CTAB 2 + ! ( COMMAND OFFSET 0x01 )
[ ' SPCON-RESET ] LITERAL CFA CTAB 4 + ! ( COMMAND OFFSET 0x02 )

[ ' IO-READ ] LITERAL CFA CTAB 6 + ! ( COMMAND OFFSET 0x03 )
[ ' IO-WRITE ] LITERAL CFA CTAB 8 + ! ( COMMAND OFFSET 0x04 )
[ ' IO-PULSE ] LITERAL CFA CTAB A + ! ( COMMAND OFFSET 0x05 )

[ ' CAM-HALT ] LITERAL CFA CTAB C + ! ( COMMAND OFFSET 0x06 )
[ ' ERROR-RESET ] LITERAL CFA CTAB E + ! ( COMMAND OFFSET 0x07 )
[ ' FLUSH-BUFFER ] LITERAL CFA CTAB 10 + ! ( COMMAND OFFSET 0x08 )

[ ' NOT-DEFINED ] LITERAL CFA CTAB 12 + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 14 + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 16 + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 18 + ! ( COMMAND not defined )
```

```
[ ' NOT-DEFINED ] LITERAL CFA CTAB 1A + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 1C + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 1E + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 20 + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 22 + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 24 + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 26 + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 28 + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 2A + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 2C + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 2E + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 30 + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 32 + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 34 + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 36 + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 38 + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 3A + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 3C + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 3E + ! ( COMMAND not defined )
```

```
( functions related to camera operation )
[ ' CAM-RESTART ] LITERAL CFA CTAB 40 + ! ( COMMAND OFFSET 0x20 )
[ ' SET-PARAM ] LITERAL CFA CTAB 42 + ! ( COMMAND OFFSET 0x21 )
[ ' CISC-ON ] LITERAL CFA CTAB 44 + ! ( COMMAND OFFSET 0x22 )
[ ' CISC-OFF ] LITERAL CFA CTAB 46 + ! ( COMMAND OFFSET 0x23 )
[ ' GAIN-HI ] LITERAL CFA CTAB 48 + ! ( COMMAND OFFSET 0x24 )
[ ' GAIN-LO ] LITERAL CFA CTAB 4A + ! ( COMMAND OFFSET 0x25 )
[ ' OFFSET! ] LITERAL CFA CTAB 4C + ! ( COMMAND OFFSET 0x26 )
[ ' SHADE ] LITERAL CFA CTAB 4E + ! ( COMMAND OFFSET 0x27 )
[ ' OSHUT ] LITERAL CFA CTAB 50 + ! ( COMMAND OFFSET 0x28 )
[ ' CSHUT ] LITERAL CFA CTAB 52 + ! ( COMMAND OFFSET 0x29 )
[ ' PIX-BIN ] LITERAL CFA CTAB 54 + ! ( COMMAND OFFSET 0x2A )
[ ' ROW-BIN ] LITERAL CFA CTAB 56 + ! ( COMMAND OFFSET 0x2B )
[ ' PIX-DISCARD ] LITERAL CFA CTAB 58 + ! ( COMMAND OFFSET 0x2C )
[ ' ROW-DISCARD ] LITERAL CFA CTAB 5A + ! ( COMMAND OFFSET 0x2D )
[ ' PIX-READ ] LITERAL CFA CTAB 5C + ! ( COMMAND OFFSET 0x2E )
[ ' ROW-READ ] LITERAL CFA CTAB 5E + ! ( COMMAND OFFSET 0x2F )
[ ' CAM-WRITE ] LITERAL CFA CTAB 60 + ! ( COMMAND OFFSET 0x30 )
[ ' CLEAR ] LITERAL CFA CTAB 62 + ! ( COMMAND OFFSET 0x31 )
[ ' READ ] LITERAL CFA CTAB 64 + ! ( COMMAND OFFSET 0x32 )
[ ' BIAS ] LITERAL CFA CTAB 66 + ! ( COMMAND OFFSET 0x33 )
[ ' XPOSE ] LITERAL CFA CTAB 68 + ! ( COMMAND OFFSET 0x34 )
[ ' DARK ] LITERAL CFA CTAB 6A + ! ( COMMAND OFFSET 0x35 )
[ ' OBS ] LITERAL CFA CTAB 6C + ! ( COMMAND OFFSET 0x36 )
[ ' SEND-STRING ] LITERAL CFA CTAB 6E + ! ( COMMAND OFFSET 0x37 )
[ ' TEMPO@ ] LITERAL CFA CTAB 70 + ! ( COMMAND OFFSET 0x38 )
[ ' TEMP1@ ] LITERAL CFA CTAB 72 + ! ( COMMAND OFFSET 0x39 )
[ ' FORMAT? ] LITERAL CFA CTAB 74 + ! ( COMMAND OFFSET 0x3A )
[ ' SET-FORMAT ] LITERAL CFA CTAB 76 + ! ( COMMAND OFFSET 0x3B )
[ ' ANTI-BLOOM-OFF ] LITERAL CFA CTAB 78 + ! ( COMMAND OFFSET 0x3C )
[ ' ANTI-BLOOM-ON ] LITERAL CFA CTAB 7A + ! ( COMMAND OFFSET 0x3D )
[ ' INTEGRATE-CCD ] LITERAL CFA CTAB 7C + ! ( COMMAND OFFSET 0x3E )
[ ' CAM-FAST ] LITERAL CFA CTAB 7E + ! ( COMMAND OFFSET 0x3F )
[ ' CAM-SLOW ] LITERAL CFA CTAB 80 + ! ( COMMAND OFFSET 0x40 )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 82 + ! ( COMMAND OFFSET 0x41 )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 84 + ! ( COMMAND OFFSET 0x42 )
[ ' SET-TEMP ] LITERAL CFA CTAB 86 + ! ( COMMAND OFFSET 0x43 )
[ ' FRAME-TRANSFER ] LITERAL CFA CTAB 88 + ! ( COMMAND OFFSET 0x44 )
[ ' FULL-FRAME ] LITERAL CFA CTAB 8A + ! ( COMMAND OFFSET 0x45 )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 8C + ! ( COMMAND not defined )
[ ' NOT-DEFINED ] LITERAL CFA CTAB 8E + ! ( COMMAND not defined )
```

```
( -----> if you extend this list, be sure to update the MAX_CMD in the following line
47 CONSTANT MAX_CMD
```

```
( POLL-CMD wait for a command in mailbox 0
( if nothing light an led
( if MSDF put it on the stack
( if CRDY fetch command number and place on the stack
( repeat until a command is found )
( POLL-CMD also checks if host is willing to accept return data
( it will place the count of the available characters in MB3 and the
( value of the next data in MB4
```

```
( MBO is polled until a command is found to be ready,
( if MSDF, the data is fetched from MB1 and placed on the stack
( When CRDY is true, the command is fetched from MB1, placed on the stack,
( and the loop is exited, returning to RUN which will interpret the command.
( It is assumed that the parameters needed for the proper execution of the command
( will be found on the stack, if not, we will Crash. For now, no error checking
( is performed to be sure that the command number found is legit. This safety
( function will be added later.
```

```

: POLL-CMD
BEGIN
  MSDF? IF ( if data...
    MB2 MB@ 100 * ( ... fetch the top half
    MB1 MB@ + ( ... add in the bottom half, leave on stack
    RDY 0 ( ... and go ready, but don't leave loop
  ELSE ( if no data ...
    CRDY? IF ( if there is a command ...
      MB1 MB@
      DUP MAX_CMD > ( check if cmd exceeds range
      IF
        DROP ( drop CMD if not valid
        CMD-ERR ( issue error flag
        BSY ( satisfy that host waits for the BSY flag
        100 M-WAIT ( wait a while
        RDY 0 ( pretend to be ready and stay in loop
      ELSE
        1 ( if valid, keep it and leave loop to execute
      THEN
    ELSE ( if no data or command
      HEARTBEAT 0 ( ... do LEDs, stay in loop
    THEN
  THEN
SERVE-HOST
UNTIL
BSY ;

```

```

HEX
OC CONSTANT STOP ( the value of CAM-HALT )

```

```

HEX
: MAIN
  1204 7600 1200 @ 1202 @ - CMOVE ( restore dictionary
  1200 @ 1202 @ - 1204 + 6 84 CMOVE ( restore FORTH user area
  SETUP ( initialize bus interface
  BUILD ( build the command table
  INITLEDS ( initialize the lights
  DECIMAL ( decimal mode please
  GO FOR IT @ ( is it OK to start ?
  IF START-IRQ ( if so, start serial receive interrupt
  BEGIN ( enter main command interpreter loop
    POLL-CMD ( wait for a command
    DUP STOP - ( check to see if it's cam-halt
    IF TRUE ( if so, leave TRUE to exit loop at UNTIL
    ELSE CLEAR-FLAGS ( clear OK QM and CR flags {just to be safe}
      DUP 0 SWAP -
      LIGHTS C! ( show command number on LED panel
      DOCMD ( execute command
      QM FLAG @ 2 - ( did we get a question mark from the camera ?
      IF QM-ERR ( if so, QM_FLAG is set, so terminate with error flag
      THEN
      RDY ( command is done, we are ready
      FALSE ( leave FALSE so we stay in loop at UNTIL
    THEN
  UNTIL ( repeat until told to halt
  STOP-IRQ ( stop serial receive interrupt routine
THEN ;

```

```

( ASTART! store the autostart sequence and the cfa
( of the autostart word in ext. eprom )

```

```

( to use a different word on startup, just replace the word restor
( in the following definition with the name of whatever you would
( like the 6811 to do on power up.

```

```

HEX
400 CONSTANT ASTART

```

```

: ASTART!
  EEUNPROT
  A44A DUP ASTART EE-!
  [ ' MAIN ] LITERAL CFA
  DICT-START -
  EEDICT-START +

```

DUP ASTART 2+ EE-!
EPROT
CR ." AUTOSTART SEQUENCE STORED " CR ;

Appendix D: 'C' Language Interface Source Code

```

/***** */
/* */
/* ATDcam.c */
/* */
/* Subroutines for Advanced Technologies VME interface. */
/* */
/* P Doherty, HJ Meyer, A Gale */
/* */
/* */
/* Rev. 2.0 4/2/92 */
/* lots of little changes, some major ones to the */
/* IRAF interface code. */
/* added lots more comments. */
/* */
/* Rev. 1.0 1/20/92 */
/* finally extracted this from cam.c */
/* provides support for all ATD VME functions */
/* including all necessary to run either the */
/* ATC5 cameras or the AIS or AIS2 */
/* */
/* */
/* */
/***** */

#include <stdio.h>
#include <errno.h>

#include <fcntl.h>

#include <sys/file.h>
#include <sys/mman.h>
#include <sys/types.h>

/* #include <ctype.h> */
/* #include <string.h> */

/*
#include "cam.h"
#include "camhdw.h"
*/

#include "ATDcamera.h"

#ifdef BIT3
#define BTS_MODEL 466
#include "btsio.h"

#define TRUE 1
#define FALSE 0

char *map_mem ();
char *map_bit3();

#define PAGE_SHFT 16
#define PAGE_SIZE (1 << PAGE_SHFT)
#define PAGE_MASK (PAGE_SHFT - 1)

#define MEMPAGE 0x1000
#define MAILPAGE 0x1200

int gBit3fd; /* bit3 file descriptor */
struct bts reg *gNodeIO; /* pointer to the bit3 I/O registers */
unsigned short gMemPage = MEMPAGE;
unsigned short gMailPage = MAILPAGE;
#endif

char *camera_base;
/* extern struct system_params camera_params; */

/* PTVSI mailboxes */

extern unsigned char *mailbox0;
extern unsigned char *mailbox1;
extern unsigned char *mailbox2;
extern unsigned char *mailbox3;
extern unsigned char *mailbox4;
extern unsigned char *mailbox5;
extern unsigned char *mailbox6;
extern unsigned char *mailbox7;
extern unsigned char *mailbox8;

```

```
extern unsigned char *mailbox9;
extern unsigned char *mailbox10;
extern unsigned char *mailbox11;
extern unsigned char *mailbox12;
extern unsigned char *mailbox13;
extern unsigned char *mailbox14;
extern unsigned char *mailbox15;
```

```
/* ***** */
/* ***** */
/*
/*      VME interface oriented commands          */
/*
/* These commands are used to operate on the VME interface */
/* locally. No action is taken by the camera controller. */
/*
/*
/* ***** */
/* ***** */
```

```
/* reset is used to reset the VME interface board. */
/* It is good practice to do so as your application */
/* first initializes. It assures that the interface */
/* is in a known state. */
/* camera_base is the base address of the command */
/* portion of the VME interface. */
/* This function is identical for ATDVME boards and */
/* VME200A boards. */
```

```
void reset ()
{
```

```
#ifdef BIT3
    gNodeIO->rem_page = MAILPAGE;
#endif
```

```
    *((char *)camera_base + 0x3d) = CMD_RESET;
    usleep (3000000);
```

```
    return;
}
```

```
/* halt tells the on board microcontroller to cease */
/* interpretation of commands from the VME master and */
/* to begin interpreting FORTH commands coming in */
/* through the serial port on the camera connector. */
/* This function is useful only for test purposes, and */
/* will lead to confusion if called while connected */
/* to the camera. */
```

```
void halt ()
```

```
{
    issue_command (CMD_HALT);
    return;
}
```

```
/* ag_init is used to reset the "address generator" PLD */
/* This PLD controls the DRAM write of camera data. */
/* Sets the start address for subsequent camera data */
/* to the first location on the VME interface board. */
/* This function should be called before an image */
/* acquisition command like dark or obs. If a */
/* series of images is to be acquired into DRAM before */
/* reading the data, then ag_init should be called */
/* only before the first acquisition. */
```

```
void ag_init()
```

```
{
    issue_command (CMD_AG_INIT);
    return;
}
```

```

/* spcon_init is used to initialize the serial to parallel */
/*   convertor control PLD an VME200A type boards with */
/*   serial data interface. Performs no useful or */
/*   harmful function if called on a ATDVME interface. */
/*   The value passed is determined by the type and rate */
/*   of the pixel data clock signal. The value used */
/*   here, 7E, is for 40 kHz readouts. For a complete */
/*   explanation of how this works, see the VME200a */
/*   theory of operation. */

```

```

void spcon_init()
{
    pass_char_param(0x3f);
    issue_command (CMD_SPCON_INIT);
    return;
}

```

```

/* spcon_reset is used to reset the serial to parallel */
/*   convertor control PLD an VME200A type boards with */
/*   serial data interface. Performs no useful or */
/*   harmful function if called on a ATDVME interface. */

```

```

void spcon_reset()
{
    issue_command (CMD_SPCON_RESET);
    return;
}

```

```

/* The ATDVme interface includes an I/O port whose function */
/*   remains uncommitted. Acces to this port is granted to */
/*   the user through the following set of fuctions. */
/* */

```

```

/* io_read is used to read the TTL data present on the */
/*   input bits of the user I/O port. */
/*   Returns the value found. */

```

```

char io_read ()
{
    unsigned char value;

#ifdef BIT3
    gNodeIO->rem_page = MAILPAGE;
#endif
    issue_command (CMD_IORD);
    wait_cmd_done ();
    value = *mailbox1;
    return (value);
}

```

```

/* io_write is used to write to the output bits of the */
/*   user I/O port. */

```

```

void io_write (value)
    char value;
{
    pass_char_param (value);
    issue_command (CMD_IOWR);
    wait_cmd_done ();
    return;
}

```

```

/* io_pulse is used to pulse one of the bits on the user */
/*   I/O port for a given number of deci-seconds */
/*   This is useful for shutter drivers or filter wheels */
/*   This function will be changed to accept bit */
/*   selection and millisecond timings. */

```

```

void io_pulse (deciseecs)
    unsigned short deciseecs;

```

```
{
pass_short_param (decisecs);
issue_command (CMD_IOPLS);
wait_cmd_done ();
return;
}
```

```
/* ***** */
/* ***** */
/* ***** */
/* Camera Oriented Commands */
/* ***** */
/* ***** */
```

```
/* cam_restart issues a series of command strings to the */
/* camera controller. These operations perform a */
/* warm restart of the camera controller. This is */
/* purely a software reset. No hardware reset of the */
/* camera controller is available through the VME */
/* interface board. */
```

```
void cam_restart ()
{
issue_command(CMD_CAM_RESTART);
wait_cmd_done ();
read_cam_buffer ();
return;
}
```

```
/* set_param is used to set a camera parameter. Parameters */
/* are static variables associated with camera */
/* operation and CCD readout. Parameters are described */
/* in detail elsewhere. */
```

```
void set_param (offset, value)
unsigned short offset;
unsigned short value;
{
pass_short_param (offset);
pass_short_param (value);
cam_command (CMD_SET_PARAM);

return;
}
```

```
/* ATD cameras are capable of performing continuous clear */
/* operations on the CCD while the camera is idle. The */
/* following two functions are used to turn this on and off.*/
```

```
void cisc_on ()
{
cam_command (CMD_CISCON);
return;
}
```

```
void cisc_off ()
{
cam_command (CMD_CISCOFF);
return;
}
```

```
/* anti_bloom_on is used to enable clock recombination */
/* anti-blooming. After this function called, CRAB */
/* will be performed during exposures. */
/* THIS FUNCTION IS NOT FULLY IMPLEMENTED IN THE ATC5 */
/* CAMERAS AND WILL PRODUCE UNDESIRABLE EFFECTS. */
```

```
void anti_bloom_on()
```

```

    {
    cam_command(CMD_ANTI_BLOOM_ON);
    return;
    }

/* anti_bloom_off this function is used to disable the      */
/* clock recombination anti-blooming mode of operation */
/* CRAB will not be performed during exposures after      */
/* this function is called.                               */
void anti_bloom_off()
{
    cam_command(CMD_ANTI_BLOOM_OFF);
    return;
}

/* The ATC5 and AIS2 cameras are capable of running at two */
/* different speeds. In the ATC5 camera, the hardware that */
/* is required may or may not have been purchased. If not, */
/* setting the camera to low speed will produce invalid    */
/* data. The AIS camera does not support this option at all.*/
void cam_fast()
{
    cam_command(CMD_CAM_FAST);
    return;
}

void cam_slow()
{
    cam_command(CMD_CAM_SLOW);
    return;
}

/* The gain of all the ATD cameras may be adjusted by the */
/* user.                                                    */
/* The ATC5 cameras have a gain switch allowing the      */
/* camera to be operated in two gain modes. The exact   */
/* value of the gain in either mode is determined by the */
/* adjustments made to the camera during integration and  */
/* test.                                                  */
/* The AIS cameras have a dual slope integrator based   */
/* architecture in the analog processing, and the camera */
/* gain may be adjusted nearly continuously by changing the */
/* dual slope integrators integration time. Three standard */
/* gain modes are provided here.                          */

void ATC5_gain_hi()
{
    cam_command (CMD_GAIN_HI);
    return;
}

void ATC5_gain_lo()
{
    cam_command (CMD_GAIN_LO);
    return;
}

void AIS_gain_lo()
{
}

void AIS_gain_mid()
{
}

void AIS_gain_hi()
{
}

```

```

/* cam_offset sets the camera's bias offset to a new value */
/* cam_offset requires one parameter: the new value to */
/* be written to the offset DAC in the camera head. */
/* This value may range from 0 to 255. */
/* This function operates on ATC5 cameras and on AIS2 */
/* cameras, but performs no function on an AIS. */

```

```

void cam_offset(value)
char value;
{
    pass_char_param (value);
    cam_command (CMD_OFFSET);
    return;
}

```

```

/* CCD_temp() is used to query the temperature of the CCD. */
/* In the ATC5, the CCD is cooled by a thermo- */
/* electric cooler located in the vacuum chamber. */
/* In the AIS cameras, the CCD is cooled by a tank */
/* of liquid nitrogen located in the dewar. */

```

```

void CCD_temp()
{
    cam_command(CMD_TEMP0);
    return;
}

```

```

/* case_temp() is used to query the temperature of the back */
/* side of the CCD chamber, which is heated by the */
/* thermoelectric cooler. The temperature is */
/* returned as a formatted ASCII text string. */

```

```

void case_temp()
{
    cam_command(CMD_TEMP1);
    return;
}

```

```

/* set_temp() is used to set the desired operating point for */
/* the CCD. A signed 8 bit value is expected. For */
/* practical purposes, the temperature of an ATC5 */
/* camera will not stabilize more than approx. 65 */
/* degrees below ambient. An AIS, with a nitrogen */
/* dewar, should reach temperatures below -80 deg C.*/

```

```

void temp_set(value)
char value;
{
    pass_s_char_param(value);
    cam_command(CMD_SET_TEMP);
    return;
}

```

```

/* oshut opens the shutter on the camera. The camera */
/* controller will pause for ODELAY milliseconds */
/* after and then return. */

```

```

void oshut ()
{
    cam_command (CMD_OSHUT);
    return;
}

```

```

/* cshut closes the shutter on the camera. The camera */
/* controller will pause for CDELAY milliseconds */
/* after and then return. */

```

```

void cshut ()
{
    cam_command (CMD_CSHUT);
    return;
}

```

```
}
```

```
/* pix_bin is used to request that the camera controller bin */  
/* a number of pixels into the CCD summing well. One */  
/* parameter is required: npix, the number of pixels */  
/* to bin. Simply put, npix CCD serial shift operations */  
/* are performed. */
```

```
void pix_bin (npix)  
    unsigned short npix;  
{  
    pass_short_param (npix);  
    cam_command (CMD_PIX_BIN);  
    return;  
}
```

```
/* row_bin is used to request that the camera controller bin */  
/* a number of rows into the CCD serial register. One */  
/* parameter is required: nrows, the number of rows to */  
/* bin. Simply put, nrows CCD parallel shift operations */  
/* are performed. */
```

```
void row_bin (nrows)  
    unsigned short nrows;  
{  
    pass_short_param (nrows);  
    cam_command (CMD_ROW_BIN);  
    return;  
}
```

```
/* pix_discard is used to request that the camera controller */  
/* discard a number of pixels in the CCD serial register. */  
/* One parameter is required: npix, the number of pixels */  
/* to discard. */
```

```
void pix_discard (npix)  
    unsigned short npix;  
{  
    pass_short_param (npix);  
    cam_command (CMD_PIX_DISCARD);  
    return;  
}
```

```
/* row_discard is used to request that the camera controller */  
/* discard a number of rows in the CCD parallel register. */  
/* One parameter is required: nrows, the number of rows */  
/* to discard. */
```

```
void row_discard (nrows)  
    unsigned short nrows;  
{  
    pass_short_param (nrows);  
    cam_command (CMD_ROW_DISCARD);  
    return;  
}
```

```
/* pix_read is used to request that the camera controller */  
/* read a number of pixels in the CCD serial register. */  
/* One parameter is required: npix, the number of pixels */  
/* to read. */
```

```
void pix_read (npix)  
    unsigned short npix;  
{  
    pass_short_param (npix);  
    cam_command (CMD_PIX_READ);  
    return;  
}
```

```
/* row_read is used to request that the camera controller */
/* read a number of rows in the CCD parallel register. */
/* One parameter is required: nrows, the number of rows */
/* to read. */
```

```
void row_read (nrows)
  unsigned short nrows;
  {
    pass_short_param (nrows);
    cam_command (CMD_ROW_READ);
    return;
  }
```

```
/* cam_write is used to request that the camera controller */
/* write a value to a sequencer address. Such addresses */
/* might be the camera control latches or any stored */
/* parameters. This is an extremely low level operation, */
/* and should only be performed if one knows exactly what */
/* one is doing. This command may be eliminated in the */
/* near future. */
```

```
void cam_write (address,datum)
  unsigned short address;
  unsigned short datum;
  {
    pass_short_param (address);
    pass_short_param (datum);
    cam_command(CMD_CAM_WRITE);
    return;
  }
```

```
/* get_format is used to get an unformatted list of the */
/* current settings of the system parameters. */
```

```
void get_format()
  {
    cam_command(CMD_FORMAT);
    return;
  }
```

```
/* set_format is used to initialize the camera controller */
/* sequencer with the current format params. When */
/* parameters are set with set_param, the sequencer */
/* is NOT updated. A subsequent set_format command is */
/* required to assure that the sequencer will use the */
/* updated format. */
```

```
void set_format()
  {
    cam_command(CMD_SET_FORMAT);
    return;
  }
```

```
/* clear is used to request that the camera controller */
/* clear all charge off the CCD. The CCD is actually */
/* cleared NUM_CLEARS times. */
```

```
void clear()
  {
    cam_command (CMD_CLEAR);
    return;
  }
```

```
/* integrate_dark is used in a low level control mode to */
/* integrate dark current on the CCD device. */
/* The shutter .... */
```

```
void integrate_dark()
  {
    cam_command(CMD_INTEGRATE_CCD);
  }
```

```

return;
}

/* integrate_light is used in a low level control mode to */
/* integrate light on the CCD device. */
/* The shutter .... */

void integrate_light()
{
    cam_command(CMD_INTEGRATE_CCD);
    return;
}

/* shade is used to request that the camera controller */
/* generate the test data stream. The amount of data and */
/* its form will depend on the current status of the */
/* format parameters. READ_PAR rows will be produced */
/* with READ_SER pixels in each row. Each pixel in the */
/* row will have the same value. The value will increment */
/* by one each row. The first row will have the value of */
/* ORG_PAR. Multiple shade patterns may be produced. The */
/* number of shades produced is equal to the current value */
/* of the NUM_IMAGES parameter. */

void shade ()
{
    ag_init();
    cam_command (CMD_SHADE);
    return;
}

/* readout is used to request that the camera controller */
/* read the image off the CCD based on the current */
/* format parameters. The CCD will actually be read */
/* NUM_IMAGES times. */
/* In this example we call ag_init() to point the data */
/* pointer to the start of the DRAM array, but this is not */
/* necessary, and would be undesirable if you desired to */
/* "stack" several images in the DRAM before fetching the */
/* data. */

void readout ()
{
#ifdef AIS
    spcon_reset ();
    spcon_init();
#endif
    ag_init ();
    cam_command (CMD_READ);
    return;
}

/* bias is used to generate in image of the CCD with no */
/* charge on it. The shutter is closed, the CCD is */
/* cleared NUM_CLEARS times, and the image is read out. */
/* This complete cycle is performed NUM_IMAGES times. */
/* In this example we call ag_init() to point the data */
/* pointer to the start of the DRAM array, but this is not */
/* necessary, and would be undesirable if you desired to */
/* "stack" several images in the DRAM before fetching the */
/* data. */

void bias()
{
#ifdef AIS
    spcon_reset ();
    spcon_init();
#endif
    ag_init ();
    cam_command (CMD_BIAS);
    return;
}

/* expose */
/* this function is redundant to the OBS command below, */

```

```

/* and exists only for some obsolete application code */
/* which expects it. */
/* In this example we call ag_init() to point the data */
/* pointer to the start of the DRAM array, but this is not */
/* necessary, and would be undesirable if you desired to */
/* "stack" several images in the DRAM before fetching the */
/* data. */

```

```

void expose (exptime)
unsigned long exptime;
{
    ag_init();
    set_exposure_time (exptime);
    cam_command (CMD_OBS);
    return;
}

```

```

/* dark this function is called to generate an image of the CCD */
/* with dark current. The shutter is closed, the CCD is */
/* cleared, dark current is allowed to build for exptime */
/* milliseconds, and the chip is read out. This entire */
/* cycle is repeated NUM_IMAGES times. */
/* In this example we call ag_init() to point the data */
/* pointer to the start of the DRAM array, but this is not */
/* necessary, and would be undesirable if you desired to */
/* "stack" several images in the DRAM before fetching the */
/* data. */

```

```

void dark (exptime)
unsigned long exptime;
{
    set_exposure_time (exptime);

```

```

#ifdef AIS
    spcon_reset ();
    spcon_init();
#endif
    ag_init ();
    cam_command (CMD_DARK);
    return;
}

```

```

/* obs this command is used to generate an object exposure */
/* image. The shutter is closed, the CCD is cleared, the */
/* shutter is opened, light is accumulated for exptime */
/* milliseconds, the shutter is closed, and the image is */
/* read out. This complete cycle is repeated NUM_IMAGES */
/* times. */
/* In this example we call ag_init() to point the data */
/* pointer to the start of the DRAM array, but this is not */
/* necessary, and would be undesirable if you desired to */
/* "stack" several images in the DRAM before fetching the */
/* data. */

```

```

void obs (exptime)
unsigned long exptime;
{
    set_exposure_time (exptime);

```

```

#ifdef AIS
    spcon_reset ();
    spcon_init();
#endif
    ag_init();
    cam_command (CMD_OBS);
    return;
}

```

```

void send_cam_char (achar)
char achar;
{
    pass_char_param (achar);
    cam_command(CMD_SEND_STRING);
    return;
}

```

```

/* ***** */
/* Text transfer commands */
/* ***** */

/* send_file is used to send an ASCII text file to the */
/* camera. The file is assumed to consist of FORTH */
void send_file (filename)
char *filename;
{
FILE *theFile;
int compiling;
char thisChar;

flush_buffer(); /* empty the serial text buffer on the board */

if ((theFile = fopen (filename, "r")) >= 0)
{
thisChar = 0x00; /* we haven't got any characters yet ... */
compiling = 0; /* ... and we're not compiling yet */

while( thisChar != EOF )
{
/* fscanf(theFile,"%c",&thisChar); /* read a char from the file
*/

thisChar = getc(theFile);

if ( thisChar == 0x0a) /* if its a line feed... */
thisChar = 0x0d; /* ... make it a carriage return */

if ( thisChar != EOF ) /* if not end of file ... */
send_cam_char (thisChar); /* ... send it to the camera */

if ( thisChar == ':' ) /* if the char is a colon ... */
compiling = 1; /* ... then we're compiling. */

if ( thisChar == ';' ) /* if the char is a semi-colon ... */
compiling = 0; /* ... we're not compiling anymore. */

if( thisChar == 0x0d ) /* if the char is a carriage return ... */
{
if (compiling) /* ... and we're compiling ... */
wait_CR_LF (); /* ... just wait for CR LF */
else
{
/* ... and we're not compiling ... */
wait_OK(); /* ... wait for an 'OK' ... */
wait_CR_LF(); /* ... and then a CR LF sequence. */
}
}
}
}
else
fprintf (stderr, "Error opening %s", filename);
return;
}

```

```

/* send_string is used to send an arbitrary string to */
/* the camera. This is useful in a variety of ways. */
/* Any command may be issued this way. A carriage */
/* return is appended to the end of the string. */
/* The camera should respond with an 'OK', but this */
/* function does not wait for it. */

```

```

void send_string (theString)
char *theString;
{
char aChar;

aChar = 0x00;
while (aChar != 0x0D)
{
aChar = *theString++;

if (aChar != 0x00)

```

```

        {
            send_cam_char(aChar);
        }
        else
        {
            aChar = 0x0D;
            send_cam_char(aChar);
        }
    }
    return;
}

```

```

/* wait_OK is used to wait until an 'OK' response has been */
/* received from the camera */

```

```

void wait_OK ()
{
    int    done;
    char   theChar;

    done = 0;
    while (done=0)
    {
        theChar = get_a_char();
#ifdef VERBOSE
        fprintf ( stderr,"%c",theChar);
#endif
        if ( theChar == 'O' )
        {
            theChar = get_a_char();
#ifdef VERBOSE
            fprintf ( stderr,"%c",theChar);
#endif
            if ( theChar == 'K' )
                done = 1 ;
        }
    }
}

```

```

/* wait_CR_LF is used to wait until an carriage return */
/* line feed pair has been recieved from the camera */

```

```

void wait_CR_LF ()
{
    int    done;
    char   theChar;

    done = 0;
    while (done=0)
    {
        theChar = get_a_char();
#ifdef VERBOSE
        fprintf ( stderr,"%c",theChar);
#endif
        if ( theChar == 0x0d )
        {
            theChar = get_a_char();
#ifdef VERBOSE
            fprintf ( stderr,"%c",theChar);
#endif
            if ( theChar == 0x0a )
                done = 1 ;
        }
    }
}

```

```

/* ***** */
/* read buffer functions */
/* ***** */

```

```

void flush buffer()

```

```

    {
    issue_command(CMD_FLUSH_BUFFER);
    wait_cmd_done();
    return;
    }

/* read_cam_buffer is used to fetch all the characters in */
/* the serial input bufer. This is useful for */
/* monitoring the camera's responses to some commands. */

void read_cam_buffer()
{
    unsigned char count;
    unsigned char thechar;

#ifdef BIT3
    gNodeIO->rem_page = MAILPAGE;
#endif
    count = *mailbox3;
    while ( count != 0 )
    {
        thechar = get_a_char() ;
        if (thechar != 0xff)
#ifdef VERBOSE
            fprintf ( stderr,"%c",thechar);
#endif
        count = *mailbox3;
    }
    return;
}

/* get_a_char is used to fetch a character from the serial */
/* input buffer. This function includes a time out. */

char get_a_char()
{
    unsigned char myChar;
    int i;

#ifdef BIT3
    gNodeIO->rem_page = MAILPAGE;
#endif

    *mailbox6 = SMDR ; /* tell 6811 we're ready for serial data */
    i=0;

    while ( (*mailbox6 == SMDR) && (i < 10000010) )
        i++;

    if (i<10000000) {
        myChar = *mailbox4;
    }
    else {
        printf("%s/n", "Timeout while reading from controller.");
        myChar = 0xff;
    }
    return(myChar);
}

/* ***** */
/* parameter transfer functions */
/* ***** */

/* pass_short_param is used to pass function parameters */
/* to the on board microcontroller before issuing */
/* commands. This function is used to pass parameters */
/* of the short integer type. */

void pass_short_param (theshort)
    unsigned short theshort;
{
    unsigned char low_byte;
    unsigned char high_byte;
}

```

```

#ifdef BIT3
    gNodeIO->rem_page = MAILPAGE;
#endif
    wait_cmd_done();

    /* Load the high byte of the datum into mailbox 2*/
    high_byte = (unsigned char)(theshort >> 8);
    *mailbox2 = high_byte;

    /* Load the low byte of the datum into mailbox 1*/
    low_byte = (unsigned char)(theshort & 0x00ff);
    *mailbox1 = low_byte;

    set_msdf();
    wait_cmd_done();
    return;
}

/* pass_char_param is used to pass function parameters to the */
/* on board microcontroller before issuing commands. */
/* This function is used to pass parameters of the */
/* char type. All parameters are unsigned. */

void pass_char_param (thechar)
    unsigned char thechar;
{
#ifdef BIT3
    gNodeIO->rem_page = MAILPAGE;
#endif
    wait_cmd_done();

    /* clear the high byte in mailbox 2*/
    *mailbox2 = 0;

    *mailbox1 = thechar;

    set_msdf();
    wait_cmd_done();
    return;
}

/* pass_s_char_param is used to pass function parameters */
/* to the on board microcontroller before issuing */
/* commands. */
/* This function is used to pass parameters of the */
/* char type. All parameters are signed. */

void pass_s_char_param (thechar)
    char thechar;
{
#ifdef BIT3
    gNodeIO->rem_page = MAILPAGE;
#endif
    wait_cmd_done();

    /* clear the high byte in mailbox 2*/
    *mailbox2 = 0;

    *mailbox1 = thechar;

    set_msdf();
    wait_cmd_done();
    return;
}

/* ***** */
/* functions used to issue commands to the controller */
/* ***** */

/* cam_command issues command, waits for completion, checks */
/* for command errors, and echoes buffer contents. */

void cam_command(cmd)
    char cmd;
{
    char status;

    wait cmd done();
}

```

```

    issue_command(cmd);
    wait_cmd_done();
    check_error(cmd);
    read_cam_buffer();
    return;
}

/* q_cam_command issues command, waits for completion, checks */
/* for command errors, and flushes serial input buffer. */

void q_cam_command(cmd)
    char cmd;
    {
    char status;

    wait_cmd_done();
    issue_command(cmd);
    wait_cmd_done();
    check_error(cmd);
    flush_buffer();
    return;
}

/* issue_command waits for the board to complete execution*/
/* Of the last function called then writes the new */
/* command number to mailbox1 and sets the command */
/* ready flag in mailbox0. This function does not wait */
/* until the camera and interface are done with the */
/* execution of the command. */

void issue_command (cmd)
    char cmd;
    {
    char status;

#ifdef BIT3
    gNodeIO->rem_page = MAILPAGE;
#endif
    wait_cmd_done();
    *mailbox1 = cmd;
    *mailbox0 = CRDY;
    return;
}

/* ***** */
/* flag manipulation functions */
/* ***** */

/* set_msdf sets the master to slave data full bit in */
/* the status register. */

void set_msdf()
    {
#ifdef BIT3
    gNodeIO->rem_page = MAILPAGE;
#endif
    *mailbox0 = MSDF;
    return;
}

/* wait_cmd done waits until the 6811 has completed the */
/* last function called. When done, then 6811 will */
/* clear mailbox 0 */
void wait_cmd_done()
    {
    char status;

#ifdef BIT3

```

```

    gNodeIO->rem_page = MAILPAGE;
#endif
    status = *mailbox0;
    while ( status != 0 )
    {
        status = *mailbox0;
    }

    return;
}

/* ***** */
/* error handling function */
/* ***** */

void check_error(cmd)
char cmd;
{
    if (*mailbox5 == 0)
        return;
    else
    {
        printf("\nError in command %x, error code %x\n", cmd, *mailbox5);
        issue_command(CMD_ERROR_RESET);
        wait_cmd_done();
        if (*mailbox5 != 0)
            printf("%s\n", "Fatal error: can't reset error flag. Propose reset.");
    }
    return;
}

```

```

/* ***** */
/* Higher level commands built from the command set */
/* and provided as examples. Used internally by ATD test */
/* software. */
/* ***** */

```

```

/* cam_init is used in quickview to initialize the camera */
/* parameter set with the current values in a struct */
/* called cparams. */

```

```

/*
void caminit (cparams)
struct system_params *cparams;
{
    unsigned short tempval;
    static int first_time = 1;
    /*
    /* Write camera system parameters */
    set_param(CCDSER, cparams->sdim);
    set_param(CCDPAR, cparams->pdim);
    set_param(BINSER, cparams->sbin);
    set_param(BINPAR, cparams->pbin);
    set_param(ORGSER, cparams->sorg + cparams->spre);
    set_param(ORGPARG, cparams->porg + cparams->ppre);
    set_param(READSER, cparams->srden);
    set_param(READPAR, cparams->prden);
    /*
    tempval = cparams->sdim -
    ( cparams->sorg + ( cparams->sbin * cparams->srden))
    + cparams->spost ;
    /*
    if ( tempval < 0x7fff )
    set_param(POSTSER, tempval);
    else
    set_param(POSTSER, 0);
    /*
    tempval = cparams->pdim -
    ( cparams->porg + ( cparams->pbin * cparams->prden))
    + cparams->ppost ;
    /*
    if ( tempval < 0x7fff )
    set param(POSTPAR, tempval);
    /*

```

```

/* else
/*     set_param(POSTSER,0);
/*
/* set_param(PRESER,0);
/* set_param(UNDERSER,0);
/* set_param(OVERSER,0);
/* set_param(OVERPAR,0);
/* set_param(PARDELAY,cparams->pdelay);
/*
/* set_param(ODELAY,cparams->odelay);
/* set_param(CDELAY,cparams->cdelay);
/*
/* if(cparams->cisc)
/*     cisc_on();
/* else
/*     cisc_off();
/*
/* set_param(EXPL,200);
/* set_param(EXPH,0);
/* set_param(NUMCLEARS,2);
/* set_param(NUMIMAGES,1);
/* set_param(IM_WAIT_LO,50);
/* set_param(IM_WAIT_HI,0);
/*
/* set_param(GSPFLAG,0);
/* set_param(GSP_CAP_WIN,0);
/* set_param(GSP_CAP_X,0);
/* set_param(GSP_CAP_Y,0);
/* set_param(GSP_DIS_WIN,0);
/*
/*
/*
/* set_format();
/*#ifdef AIS
/*     send_file("AISinit");
/*#endif AIS
/*
/* read_cam_buffer();
/*
/* return;
/* }
/*
*/

```

```

void set_exposure_time(exptime)
unsigned long exptime;
{
    unsigned short lotime;
    unsigned short hitime;

    lotime = (unsigned short) exptime;
    set_param(EXPL, lotime);

    hitime = (unsigned short) (exptime >> 16);

    set_param(EXPH, hitime);

    return;
}

```

```

/* rexpose accepts two parameters the exposure time and the */
/*     number of exposures desired. */

```

```

void rexpose (exptime,count)
unsigned long exptime;
int count;
{
    int i ;

    set_exposure_time(exptime);
    for (i=0; i<count; i++)
    {
        ag_init();
        cam_command(CMD_OBS);
    }
    return;
}

```

```
void rbias(count)
  int count;
  {
  int i;

  for(i=0;i<count;i++)
    bias();
  return;
  }
```

```
void robs(exptime,count)
  unsigned long exptime;
  int count;
  {
  int i;

  for(i=0;i<count;i++)
    obs(exptime);
  return;
  }
```

```
/* AIS cameras have additional flexibility not available in the ATC5. */
/* Clock voltages and clock sequence are programmable. */
/* The following functions are available to initialize these to the */
/* desired values. This would normally only be done as a maintenance */
/* function or in the integration of an new CCD imager. */
/* It is assumed that the user has retained the file naming convention */
/* wherein timing information is stored in a file named AISTiming and */
/* clock voltage settings are retained in a file named AISvolts. */
/* If you change the name of the files, you've got to change this code */
/* */
```

```
#ifdef AIS
```

```
/* AIS_init_volts send the file that sets the clock rail voltages */
```

```
void AIS_init_volts()
  {
  send_file("AISvolts");
  return;
  }
```

```
/* AIS_init_timing send the file that sets the timing parameters */
```

```
void AIS_init_timing()
  {
  send_file("AISTiming");
  return;
  }
```

```
#endif AIS
```

```
/* ***** */
/* dummy routines just to keep the rest of quickview happy */
/* ***** */
```

```
void rdark(exptime,count)
  unsigned long exptime;
  char count;
  {
  }
```

```
void trig_arm(cmd)
  char cmd;
  {
  }
```

```
void focus(exptime)
  unsigned long exptime;
  {
  }
```

```
void send stop()
```

```

    {
    }

void nexpose()
{
}

void trig_disarm()
{
}

void camwait()
{
}

void slavewait()
{
}

/* ***** */
/* test functions, used for low level debugging at ATD only. */
/* ***** */

void get_mbs()
{
    unsigned char  i;

#ifdef BIT3
    gNodeIO->rem_page = MAILPAGE;
#endif
    fprintf(stderr, " mailbox1 = %x \n", *mailbox0);
    fprintf(stderr, " mailbox0 = %x \n", *mailbox1);
    fprintf(stderr, " mailbox2 = %x \n", *mailbox2);
    fprintf(stderr, " mailbox3 = %x \n", *mailbox3);
    fprintf(stderr, " mailbox4 = %x \n", *mailbox4);
    fprintf(stderr, " mailbox5 = %x \n", *mailbox5);
    fprintf(stderr, " mailbox6 = %x \n", *mailbox6);
    fprintf(stderr, " mailbox7 = %x \n", *mailbox7);
    fprintf(stderr, " mailbox8 = %x \n", *mailbox8);
    fprintf(stderr, " mailbox9 = %x \n", *mailbox9);
    fprintf(stderr, " mailbox10 = %x \n", *mailbox10);
    fprintf(stderr, " mailbox11 = %x \n", *mailbox11);
    fprintf(stderr, " mailbox12 = %x \n", *mailbox12);
    fprintf(stderr, " mailbox13 = %x \n", *mailbox13);
}

/* open_devices.c */
/*
/*      A 'C' language program that open's the necessary device drivers
/*      for communicating with either an ATDVME1 VMEbus interface or a VME200a
/*      interface.  Used when controlling ATC5 or AIS cameras under UNIX.
/*
/*      Used at Advanced Technologies in Sun workstations with or without the
/*      Bit3 Sbus to VMEbus adaptor
/*
/*      History:
/*      revision 1.0      1/22/92 ped
/*                        extracted as necessary from ccd.c
/*                        thoroughly untested
*/

void *ram_base;          /* pointer to mmaped image buffer ram */
int ram_pbase;          /* the physical starting address of external memory */
int ram_len;            /* the length, in bytes, of external memory */

char *camera_base;     /* pointer to mmaped controller */
int cam_pbase;         /* the physical starting address of controller */
int cam_len;           /* the length, in bytes, of the controller interface */

unsigned char *mailbox0;
unsigned char *mailbox1;
unsigned char *mailbox2;
unsigned char *mailbox3;
unsigned char *mailbox4;

```

```
unsigned char *mailbox5;
unsigned char *mailbox6;
unsigned char *mailbox7;
unsigned char *mailbox8;
unsigned char *mailbox9;
unsigned char *mailbox10;
unsigned char *mailbox11;
unsigned char *mailbox12;
unsigned char *mailbox13;
unsigned char *mailbox14;
unsigned char *mailbox15;
```

```
char *map_VME_devices ();
void ATD_init_devices();
```

```
void ATD_init_devices ()
{
```

```
    /* NOTE : hard coded addresses for the VMEbus interface board ! */
```

```
    cam_pbase = 0x10800000;    /* physical address of controller portion */
    cam_len   = 0x200;        /* length, in bytes, of controller portion */

    ram_pbase = 0x10000000;    /* physical address of DRAM portion */
    ram_len   = 0x800000;     /* length of DRAM portion */
```

```
#ifdef BIT3
    init_bit3 ();
#endif BIT3
```

```
    /* Map the controller device into user address space */
```

```
#ifdef BIT3
    camera_base = (char *) map_bit3 ();
#else BIT3
    camera_base = (char *) map_VME_devices (cam_pbase, cam_len);
#endif BIT3
```

```
    /* assign values to the mailbox pointers */
```

```
mailbox0 = (unsigned char *) camera_base + 1;
mailbox1 = (unsigned char *) camera_base + 3;
mailbox2 = (unsigned char *) camera_base + 5;
mailbox3 = (unsigned char *) camera_base + 7;
mailbox4 = (unsigned char *) camera_base + 9;
mailbox5 = (unsigned char *) camera_base + 11;
mailbox6 = (unsigned char *) camera_base + 13;
mailbox7 = (unsigned char *) camera_base + 15;
mailbox8 = (unsigned char *) camera_base + 17;
mailbox9 = (unsigned char *) camera_base + 19;
mailbox10 = (unsigned char *) camera_base + 21;
mailbox11 = (unsigned char *) camera_base + 23;
mailbox12 = (unsigned char *) camera_base + 25;
mailbox13 = (unsigned char *) camera_base + 27;
mailbox14 = (unsigned char *) camera_base + 29;
mailbox15 = (unsigned char *) camera_base + 31;
```

```
    /* Map DRAM memory device into user address space */
```

```
#ifdef BIT3
    ram_base = camera_base;
#else BIT3
    ram_base = (char *) map_VME_devices (ram_pbase, ram_len);
#endif BIT3
}
```

```
/* map_VME_devices mmaps devices into virtual address space */
/*
/*   accepts two parameters :
/*   addr   physical address of VME device
/*   len    length, in bytes, of VME device
/*
/*   called when no bit3 Sbus to VME adaptor is used
/*
/*
```

```
char *map_VME_devices (addr, len)
int addr;
int len;
```



```

/* ***** */
int setup (io_p)
struct bts_reg *io_p;
{
    u_char      data;

    if (io_p->loc_status & NO_POWER)
    {
        bit3_check (io_p);
        return FALSE;
    }
    data = io_p->rem_cmd;
    io_p->loc_cmd = CLEAR_ERR;
    io_p->rem_cmd = USE_PAGE;
    io_p->rem_page = 0;
    return (bit3_check (io_p));
}

/* ***** */
int bit3_check (io_p)
struct bts_reg *io_p;
{
    int data;

    data = io_p->loc_status;
    if (data & (ERROR_MASK | NO_POWER))
    {
        printf ("Status error 0x%2.2x:\n", data);
        if (data & NO_POWER)
        {
            printf ("\tRemote chassis off or cable disconnected.\n");
        }
        else
        {
            if (data & ERR_PARITY)
                printf ("\tInterface parity error.\n");
            if (data & ERR_BERR)
                printf ("\tRemote bus error.\n");
            if (data & ERR_TIMEOUT)
                printf ("\tInterface timeout.\n");
        }
        return (FALSE);
    }
    return (TRUE);
}

/* ***** */
char *map_bit3 ()
{
    char *baseMemAddr = (char *) mmap ((char *)NULL, PAGE_SIZE,
    PROT_READ | PROT_WRITE, MAP_SHARED, gBit3fd, BT_RRAM);

    if (baseMemAddr == (char *)-1)
    {
        fprintf (stderr, "Failed to mmap memory.\n");
        exit (-1);
    }
#ifdef VERBOSE
    fprintf (stderr, "\tMemory mmap returned %x\n", (int)baseMemAddr);
#endif
    return (baseMemAddr);
}

#endif BIT3

/* ***** */
/* ***** */
/* ***** */
/* IRAF interface commands */
/* ***** */

```

```

/* These commands are used by the IRAF interface      */
/* program detlocal.c                                */
/* They are not used in Quickview.                  */
/* created 1/22/92 ped                               */
/* ***** */
/* ***** */

```

```

void ATD_irafr_defaults();
void ATD_irafr_format();
void ATD_irafr_initialize();

```

```

void ATD_irafr_defaults()
{
    set_param ( CCDSER, 2048 ); /* total length of serial register, including
extension */
    set_param ( BINSER, 1 ); /* serial binning factor
*/
    set_param ( PRESER, 0 ); /* serial pre-prescan
*/
    set_param ( UNDERSER, 0 ); /* serial underscan
*/
    set_param ( ORGSER, 0 ); /* serial read origin, should include extension
*/
    set_param ( READSER, 2048 ); /* serial read dimension
*/
    set_param ( POSTSER, 0 ); /* serial postscan, pixels to discard after the
read */
    set_param ( OVERSER, 0 ); /* serial overscan, pixels to read after overscan
*/
    set_param ( CCDPAR, 2048 ); /* total length of parallel register
*/
    set_param ( BINPAR, 1 ); /* parallel binning factor
*/
    set_param ( ORGPAR, 0 ); /* parallel read origin
*/
    set_param ( READPAR, 2048 ); /* parallel read dimension
*/
    set_param ( POSTPAR, 0 ); /* parallel postscan, rows to discard after the
read */
    set_param ( OVERPAR, 0 ); /* parallel overscan, rows to read after the
postscan */
    set_param ( PARDELAY, 100 ); /* parallel clock delay time
*/
    set_param ( ODELAY, 20 ); /* ODELAY is the delay for the shutter to open
fully */
    set_param ( CDELAY, 20 ); /* CDELAY is the delay for the shutter to close
fully */
    set_param ( EXPL, 200 ); /* exposure time as a double word
*/
    set_param ( EXPH, 0 );
    set_param ( NUMCLEARS, 1 ); /* the number of clears to perform when clearing
*/
    set_param ( CISCFLAG, 0 ); /* continuous clear flag
*/
    set_param ( ANTIBLOOM, 0 ); /* clock recombination anti-blooming flag
*/
    /* set_param ( CAMO_SDIR, 0 ); /* serial dir for CAM0 0 = forwards, 1 =
backwards */
    /* set_param ( CAMO_PDIR, 0 ); /* parallel dir for CAM0 0 = forwards, 1 =
backwards */
    /* set_param ( CAM1_SDIR, 0 ); /* serial dir for CAM1 0 = forwards, 1 =
backwards */
    /* set_param ( CAM1_PDIR, 0 ); /* parallel dir for CAM1 0 = forwards, 1 =
backwards */
}

```

```

void ATD_irafr_format (ser_bin, ser_pre, ser_under, ser_org, ser_read, ser_post,
ser_over, par_bin, par_org, par_read)

long ser_bin, ser_pre, ser_under, ser_org;
long ser_read, ser_post, ser_over;
long par_bin, par_org, par_read;
{

```

```

    set_param (BINSER, (unsigned short) ser_bin );
    set_param (PRESER, (unsigned short) ser_pre );
    set_param (UNDERSER, (unsigned short) ser_under );
    set_param (ORGSER, (unsigned short) ser_org );
    set_param (READSER, (unsigned short) ser_read );
    set_param (POSTSER, (unsigned short) ser_post );
    set_param (OVERSER, (unsigned short) ser_over );
    set_param (BINPAR, (unsigned short) par_bin );
    set_param (ORGPARG, (unsigned short) par_org );
    set_param (READPAR, (unsigned short) par_read );

    set_param (OVERPAR,0);
}

void ATD_1raf_initialize()
{
    printf(stderr,"opening VMEbus device drivers...\n");
    ATD_init_devices ();
#ifdef VERBOSE
    fprintf (stderr,"performing restart of camera controller ...\n");
#endif
    reset ();
#ifdef VERBOSE
    fprintf (stderr,"initializing camera ...\n");
#endif
    cam_restart ();
#ifdef VERBOSE
    fprintf (stderr,"setting all defaults ... \n");
#endif
    ATD_1raf_defaults();
    set_format();
}

static char *ATD_data_ptr;

void ATD_init_data_ptr()
{
    ATD_data_ptr = (char *)ram_base;
}

char *ATD_get_data_ptr()
{
    return (ATD_data_ptr);
}

void ATD_set_data_ptr(ptr)
char *ptr;
{
    ATD_data_ptr = ptr;
}

```

```

/*****/
/*                                     */
/*  ATDcam.h                           */
/*                                     */
/*  Definitions for ATDcam.c           */
/*                                     */
/*  History:   Rev 1.0  4/3/92         */
/*            Extracted from cam.h for use in IRAF interface */
/*                                     */
/*  Copyright (c) 1992 Advanced Technologies Division of Photometrics Ltd. */
/*                                     */
/*****/

```

```

/* functions */
void caminit ();
void reset ();
void halt ();

void cam_restart();
char io_read ();
void io_write ();
void io_pulse ();
void set_param ();
void ciscon ();
void ciscoff ();
void gain_lo();
void gain_hi();
void offset();
void shade ();
void oshut ();
void cshut ();
void pix_bin();
void row_bin();
void pix_discard();
void row_discard();
void pix_read();
void row_read();
void cam_write();
void clear();
void readout();
void bias ();
void xpose ();
void dark ();
void obs ();

void pass_short_param ();
void pass_char_param ();

void pass_s_char_param();

void q_cam_command();

void cam_command ();
void issue_command ();

void check_error();
void set_crdy();
void set_msdf();

void wait_cmd_done ();
void wait_smdf();

void get_mbs();

void fill_mbs();
void get_mb0();

void temp0 ();
void temp1 ();
void get_format ();
void set_format ();
void anti_bloom_off();
void anti_bloom_on();
void integrate_ccd();
void cam_fast ();
void cam_slow();

void wait_empty();
void wait_OK();
void wait_CR_LF();

void set_exposure_time();

```

```
char get_a_char();
void send_file();

void flush_buffer();
void read_cam_buffer();
```

```
/******
/* The 68HC11 controller command set. */
/******
/* VME board commands */
```

```
#define CMD_AG_INIT      0x0000 /* */
#define CMD_SPCON_INIT  0x0001 /* */
#define CMD_SPCON_RESET 0x0002 /* */

#define CMD_IORD         0x0003 /* */
#define CMD_IOWR         0x0004 /* */
#define CMD_IOPLS        0x0005 /* */

#define CMD_HALT         0x0006 /* */
#define CMD_ERROR_RESET 0x0007 /* */
#define CMD_FLUSH_BUFFER 0x0008 /* */
```

```
/******
/* Camera commands */
```

```
#define CMD_CAM_RESTART 0x0020 /* */
#define CMD_SET_PARAM   0x0021 /* */
#define CMD_CISCON      0x0022 /* */
#define CMD_CISCOFF     0x0023 /* */
#define CMD_GAIN_HI     0x0024 /* */
#define CMD_GAIN_LO     0x0025 /* */
#define CMD_OFFSET      0x0026 /* */
#define CMD_SHADE       0x0027 /* */
#define CMD_OSHUT       0x0028 /* */
#define CMD_CSHUT       0x0029 /* */
#define CMD_PIX_BIN     0x002A /* */
#define CMD_ROW_BIN     0x002B /* */
#define CMD_PIX_DISCARD 0x002C /* */
#define CMD_ROW_DISCARD 0x002D /* */
#define CMD_PIX_READ    0x002E /* */
#define CMD_ROW_READ    0x002F /* */
#define CMD_CAM_WRITE   0x0030 /* */
#define CMD_CLEAR       0x0031 /* */
#define CMD_READ        0x0032 /* */
#define CMD_BIAS        0x0033 /* */
#define CMD_XPOSE       0x0034 /* */
#define CMD_DARK        0x0035 /* */
#define CMD_OBS         0x0036 /* */
#define CMD_SEND_STRING 0x0037 /* */
#define CMD_TEMP0       0x0038 /* */
#define CMD_TEMP1       0x0039 /* */
#define CMD_FORMAT      0x003A /* */
#define CMD_SET_FORMAT  0x003B /* */
#define CMD_ANTI_BLOOM_OFF 0x003C /* */
#define CMD_ANTI_BLOOM_ON 0x003D /* */
#define CMD_INTEGRATE_CCD 0x003E /* */
#define CMD_CAM_FAST    0x003F /* */
#define CMD_CAM_SLOW    0x0040 /* */
```

```
#define CMD_CAM_ENABLE 0x0041
#define CMD_CAM_DISABLE 0x0042
#define CMD_SET_TEMP   0x0043
#define CMD_FRAME_TRANSFER 0x0044
#define CMD_FULL_FRAME 0x0045
```

```
/******
/* Constants */
```

```
#define CMD_STOP 0x00FF /* */
#define CMD_RESET 0x00AA /* */
```

```
/* parameter offsets as far as the camera is concerned */
```

```
#define CCDSER 0 /* length of serial register, incl. extension */
#define BINSER 1 /* serial binning factor */
#define PRESER 2 /* serial prescan */
#define UNDERSER 3 /* serial underscan */
#define ORGSER 4 /* serial read origin */
#define READSER 5 /* serial read dimension */
```

```

#define POSTSER      6  /* serial postscan          */
#define OVERSER      7  /* serial overscan         */
#define CCDPAR       8  /* length of parallel register */
#define BINPAR       9  /* parallel binning factor   */
#define ORGPARG      10 /* parallel read origin      */
#define READPAR      11 /* parallel read dimension    */
#define POSTPAR      12 /* parallel postscan         */
#define OVERPAR      13 /* parallel overscan         */
#define PARDELAY     14 /* parallel clock delay time  */

/* parameter locations 15 through 19 unused for now */

#define ODELAY       20 /* shutter open delay [m]    */
#define CDELAY       21 /* shutter close delay [ms]  */
#define EXPL         22 /* exposure time              */
#define EXPH         23
#define NUMCLEARS    24 /* number of clears to execute */
#define NUMIMAGES    25 /* number of images to acquire */
#define IM_WAIT_LO   26 /* delay between image acquisitions */
#define IM_WAIT_HI   27

/* parameter locations 28 through 29 unused for now */

#define CISCFLAG     30 /* continuous clear flag     */
#define ANTIBLOOM    31 /* clock recombination anti-blooming flag */
#define ARCH         32 /* ccd architecture          */
                        0 = full frame, 1 = frame transfer */
#define SPEED        33 /* camera readout speed 0 = slow, 1 = fast */
#define SLDEL1       34 /* slow readout delay 1      */
                        from shift to summing well */
#define SLDEL2       35 /* slow readout delay 2      */
                        from summing well to adc start pulse */
#define CAMDAC_DEFAULT 36 /* default value for offset dac */

/* parameter locations 37 through 39 unused for now */

#define GSPFLAG      40 /* graphics system processor flag */
#define GSP_CAP_WIN  41 /* gsp capture window          */
#define GSP_CAP_X    42 /* gsp capture window x location */
#define GSP_CAP_Y    43 /* gsp capture window y location */
#define GSP_DIS_WIN  44 /* gsp display window         */

/* parameter locations 45 through 49 unused for now */

/* the following parameters are used to set the shift directions on a AIS or
   AIS2 camera system. 0 = forwards, 1 = backwards. */
#define CAM0_SDIR    50 /* serial direction for CAM0 */
#define CAM0_PDIR    51 /* parallel direction for CAM0 */
#define CAM1_SDIR    52 /* serial direction for CAM1 */
#define CAM1_PDIR    53 /* parallel direction for CAM1 */

/* Offsets into the PTVSI mailbox registers */

#define MB0 1
#define MB1 3
#define MB2 5
#define MB3 7
#define MB4 9
#define MB5 11
#define MB6 13
#define MB7 15
#define MB8 17
#define MB9 19
#define MB10 21
#define MB11 23
#define MB12 25
#define MB13 27
#define MB14 29
#define MB15 31

/* Offset to the PTVSI reset location */

#define RESET 0x3D;

/* Bit masks used for interpreting the status mailbox (mailbox 0) */

#define CRDY 0x01
#define MSDF 0x02
#define BSF 0x04

```

```
/* Bit masks for interpreting mailbox 6 */  
#define SMDR 0x01
```

Appendix E: UNIX Utility Source Code

```
/* AISSay - a simple program used to send any old text string to the AIS cameras
   through the serial port on the VME interface. May also be used with
   ATC5 or ATC6 cameras. */
```

```
#include <stdio.h>
```

```
main (argc,argv)
```

```
    int argc;
    char *argv[];
```

```
    {
    if(argc != 2)
        {
            fprintf(stderr, "\nUsage: AISSay string\n");
            exit();
        }
    }
```

```
    ATD_init_devices();
    send_string(argv[1]);
    wait_OK();
    wait_CR_LF();
    }
```

```
/* AISsend - a simple program used to send any old text file to the AIS cameras
   through the serial port on the VME interface. May also be used with
   ATC5 or ATC6 cameras. */
```

```
#include <stdio.h>
```

```
main (argc,argv)
    int argc;
    char *argv[];

    {
    if(argc != 2)
        {
        fprintf(stderr, "\nUsage: AISsend filename\n");
        exit();
        }

    ATD_init_devices();
    send_file(argv[1]);

    }
```

```

/* AISsetup - used to initialize some of the more advanced
   features of the AIS camera system, including the
   programmable clock voltages and the programmable timing
   states. Meant for use by the system maintenance person
   more than for the casual user.
*/

#include <stdio.h>
void ask_default();

main ()
{
    unsigned int selection;
    int done=0;

    ATD_init_devices();

    while(!done)
    {
        printf("\n\n");
        printf("          AIS setup procedure  \n\n");
        printf("  Please select one of the following operations\n\n");
        printf("    0. reset and initialize camera\n");
        printf("    1. initialize clock voltages\n");
        printf("    2. initialize timing information\n");
        printf("    3. initialize parameters\n");
        printf("    4. exit this program\n");
        printf("\n\nplease enter your selection...");

        scanf("%u",&selection);

        if((selection < 0) || (selection > 4))
            printf("ERROR: invalid selection.\n");

        if(selection == 0)
        {
            reset();          /* reset the VMEbus interface  */
            cam_restart();    /* re-initialize the camera */
        }

        if(selection == 1)
        {
            send_file("AISvolts");
            ask_default();
        }

        if(selection == 2)
        {
            send_file("AISTiming");
            ask_default();
        }

        if (selection == 3)
            printf("\nSorry ... function not yet supported.\n");

        if (selection == 4)
            done = 1;
    }
    exit(0);
}

```