NASA Contractor Report No. 185114

Film Annotation System For A Space Experiment

MILETUS ASSOCIATES, INC.
Albuquerque, New Mexico

July 1989

Prepared for
National Aeronautics and Space Administration
Lewis Research Center
Contract NAS 3-25055

Table of Contents

PRECEDING PAGE BLANK NOT FILMED

Table of Contents, cont'd.

Section 1
Summary

The following document has been prepared to provide the user with operating instructions for the Isothermal Dendritic Growth Experiment (IDGE) Film Annotation System (FAS). This annotation system is a microprocessor interface to a modified 35mm Nikon camera.

This microprocessor system has been manufactured on a single STD BUSS interface card. The interface card has been designed in such a way as to allow it to be used in either a stand alone application with minimum features or installed in an STD computer with maximum features available.

If the FAS card is installed in an STD computer system it has the ability to take commands from and communicate status information to the STD computer. If the FAS card is set up in a stand alone configuration, it will print time (starting from zero), day (0-9), camera ID (0-3), and frame count (0-255).

The Nikon camera has been modified to print a single row of up to 28 5x7 dot matrix characters across the bottom of the 35mm frame. A single bar of seven LEDs and coherent glass fiber bundle has been installed in the pressure plate of the film back to accomplish this annotation. In addition, an optical encoder has been installed in the motor driver to provide column demands to the FAS card.

Section 2
Introduction

Miletus has designed a FAS for use by NASA on the Isothermal Dendritic Growth Experiment (IDGE).

The systems capabilities include operation as a manual controller of a 35mm camera under the direction of the main CPU, controlling the STD BUSS, and operation as an intelligent intervalometer precisely controlling the photo taking sequence. In either case, all photos are annotated with alpha-numeric data including GMT (to 0.01 seconds resolution), camera ID number, frame count and text data derived from the STD BUSS CPU. The system consists of three parts:

.) The Camera Modifications to add an LED printing matrix and an optical encoder to detect film motion.

.) An intelligent STD BUSS peripheral card providing an interface between the CPU controlling the STD BUSS and the camera itself.

.) High level language, (Turbo Pascal) software subroutines, (source code) to implement all functions from within the main IDGE software.

Each of these items is discussed in detail in the following sections.

Section 3
Camera Modifications

Miletus has developed an LED printing matrix consisting of a linear array of seven LEDs and a coherent fiber optic bundle installed in the pressure plate of a 35mm Camera. This allows data to be printed directly on the film as each photo is taken. The print technique is similar to that used in a dot-matrix computer printer except that it is the media (film) instead of the print head that moves. That is, the data is printed on the film as the motor driver advances it to the next frame. Using this method, it is possible to print any character that can be defined by a 5X7 dot-matrix. Reliability is enhanced by using such a simple matrix, (only eight wires are required). In this system, 28 alpha-numeric digits are printed on each frame.

In addition to the LED matrix it is also necessary to install an optical encoder to allow sensing of film motion. This is done using an optical sensor designed for reflective sensing. This device incorporates an infrared LED and a photo-transistor in a single package. A disk is mounted (usually to the motor drive main drive gear) in the camera to provide a surface upon which the sensor can be focused. The disk consists of "spokes" of silvered or flat black material such that pulses are generated as the motor turns and the film is advanced.

The following table describes the sequence of events occurring for one photograph and annotation operation:

	FAS CPU CARD:	CAMERA:
1.)	Send trigger to camera	---
2.)	---	Camera opens shutter and sends "X" switch pulse
3.)	FAS recognizes "X" switch and freezes data for annotation.	---
4.)	---	Camera finishes exposing film, closes shutter and begins transporting film.
5.)	Annotation begins when encoder pulses are received, and finishes when all data is printed.	---
6.)	---	Camera stops transporting film and is ready for next frame.

Miletus has performed modifications of this nature to many different models of 35mm SLR Cameras while constantly aiming to avoid interference with either the electro-mechanical operation of the camera or the photo/optical capabilities. Since a non-contact motion sensor is used there is no affect on the mechanical operation of the camera. Also, since the LED printing matrix is independent of the camera shutter and lens the camera/lens settings have no effect on the LED intensity nor do the LED intensity settings affect the camera.

Note: Under normal operation the FAS CPU Card acts as an intervalometer and, as such, is the primary source of camera trigger pulses. However, the camera may also be triggered by STD CPU or by the trigger on the camera itself. Consequently, the FAS Card has been designed to annotate the film no matter which source of triggering is utilized. However, a cable waving change must be made to enable annotation when triggering the camera with its own shutter release control. This change involves removing the wire from J3 pin 12 and connecting it to J3 pin 19.

Note: If the FAS card is utilized in the standalone mode, jumper W1 must be connected to +5v. See Appendix D, drawing #3594.

Section 4
FAS Hardware Description

4.0 General

Each FAS card contains all the STD BUSS communication capability, the data storage and time keeping functions, and the camera interface and control circuitry required to annotate one 35mm SLR camera under the control of the STD BUSS CPU.

The hardware consists of the following parts:

a) Microprocessor and support circuitry (NSC800 microprocessor)
b) Real Time Clock (RTC) and crystal oscillator (ICM 7170)
c) Camera interface and control circuitry (PAL Device)
d) STD BUSS Interface and decoding logic (PAL Device)

4.1 Microprocessor Support Circuitry

The microprocessor and support circuitry consists of an eight bit CPU with 2K bytes of RAM and 8K bytes of EPROM. Erasable Programmable Logic Devices, (EPLD) are used to implement the necessary memory and I/O decoding functions required to operate the CPU. A hardware reset of the FAS CPU is available to the STD BUSS CPU by writing to a dedicated STD BUSS I/O address. Additionally, a watchdog timer is provided to initiate a FAS CPU reset should certain critical I/O operations fail to be detected within a certain allotted amount of time.

4.2 Real Time Clock (RTC)

The RTC has a resolution of 0.01 second and uses a standard crystal oscillator with a trimmer capacitor to adjust the accuracy. Using this trimmer the oscillator is set to the exact desired frequency (within the accuracy of our measuring equipment; ±1/2 PPM) at the operating temperature to be used. The expected frequency deviation should be less than ±10 PPM over the range (30 degrees C ±10) To maintain a drift of under two seconds per day only requires an accuracy of ±23 PPM. The clock is also settable from the STD BUSS only.

Upon power up, the RTC is loaded with all zeros and starts counting from there. It could be possible to use this feature as an elapsed time counter for use on other projects at a later date.

The ones of days are maintained by the FAS CPU. Upon power up the ones of days are set to zero and may be set to a value between 0 and 9 through the use of the STD CPU.

4.3 Camera Interface Circuitry

The camera interface circuitry allows the FAS CPU to trigger the camera, take a picture, detect an "X" switch closure from the camera, and (using the pulses provided by the optical encoder installed in the camera motor driver), annotates the film. The FAS circuitry allows the LED matrix intensity to be adjusted for proper exposure. This is accomplished using wire jumpers that can easily be changed in the field.

Additionally, this circuitry contains a hardware and software frame counter. The hardware frame counter may only be accessed by STD BUSS CPU. The software frame counter, however, is accessed by both the FAS CPU and the STD CPU. This feature ensures that the current frame count contained in the hardware frame counter cannot be disturbed by the FAS CPU should a malfunction occur. The hardware frame counter may be reset to zero by the STD CPU. It should be noted that the hardware frame count is used for reference only and the software frame count is printed on the film.

The camera may be triggered by the STD BUSS CPU or the FAS CPU; the STD BUSS CPU may disable the FAS CPU from performing this function. This allows the STD BUSS CPU to take over camera control and detect a problem on the FAS CPU Board.

The "X" switch is de-bounced for detection and used by the FAS CPU to freeze data at the time the film is exposed. The LED Head will then print this data as the film is advanced. This signal is also provided as an output to the IDGE system flash unit.

4.4 STD BUSS Interface

The STD BUSS interface decoding is performed using an EPLD in order to minimize board surface area used for this function. Two "types" of STD BUSS ports exist on the FAS Board, those that are independent of the FAS CPU and those used for control and communication of the FAS CPU by the STD BUSS CPU.

The BUSS ports independent of the FAS CPU consists of the following:

- FAS CPU HARD RESET (Write 00 to Port 108H)
- HARDWARE FRAME COUNTER RESET (Write 02 to Port 108H)
- READ HARDWARE FRAME COUNTER (Read from 107H)
- STD BUSS TRIGGER TO CAMERA (Write 04 to 108H)
- FAS CPU CAMERA TRIGGER ENABLE/DISABLE (Write 08 to 108H)

The BUSS ports used for control and communication of the FAS CPU are:

- FAS Camera Interface STATUS to STD (Port 104H)
- FAS Interval warning to STD (Port 103H Bit 0)
- FAS PHOTO confirmation bit to STD (Port 102 Bit 0)
- DATA to FAS (from STD) (Port 101H)
- DATA to STD (from FAS) (Port 100H)
- FAS READY FOR DATA (Port 105H Bit 0)
- STD READY FOR DATA (Port 106H Bit 0)

4.4.1 FAS Camera Interface Status to STD

The FAS is capable of informing the STD CPU of certain conditions. Some of these are derived from the camera interface and some from the FAS CPU itself. A brief description of each follows.

From the Camera Interface:

- Camera trigger received but no "X" switch detection. This error will occur and be detected by the FAS CPU during every photograph. However, this detection is normal and due to the delay between the time the camera is triggered by the FAS and the time the camera actually takes the photo. This error is reported by the FAS for a very brief time and will probably be undetectable by the STD CPU. Consequently, the only time this error should be interpreted as a failure is when the error persists for a prolonged period of time (longer than 500ms).

- "X" switch received but no encoder pulses detected. This error is reported when the FAS detects an "X" switch but does not detect encoder pulses. This error is reported in the same manner as discussed above.

- Encoder pulses received but terminated prior to a print complete. This error occurs when the FAS receives an insufficient number of encoder pulses to complete a print cycle. This error is also reported in the same fashion as the previous two errors discussed.

From FAS CPU:

- Watchdog timer timeout occurred. Should the FAS CPU be reset by the watchdog timer, this bit will inform the STD BUSS CPU that this occurred. When the status port is read onto the STD BUSS, this bit is reset. If a failure occurs that causes continuous timeout resets, this bit would continue to report each one.

- FAS EPROM Check Sum Pass/Fail.

- FAS RAM Test Pass/Fail.

- FAS CPU Self-Test Pass/Fail.

The three above items represent the results of the FAS Card self test function described in Section 5.19.

- FAS CPU Annotation Mode Bits:

These bits represent the current annotation mode (A, B, or C) as discussed in Section 4.6.

The eight bit port to be used for reporting these status/error codes is allocated as follows:

(Port Address 104 Hex)

Bits 0-1: Camera Interface
00 = Camera Hardware OK
01 = Trigger sent but no "X" switch
10 = "X" switch but no encoder pulses
11 = Encoder pulses but no print complete

Bits 2-4: FAS Card Self Test
000 = ALl tests pass
001 = EPROM check sum fail
010 = RAM check fail
100 = CPU check fail
(other codes would reflect multiple failures)

Bit 5: Time-out reset occurred flip flop

Bits 6-7: FAS Annotation Mode Bits
00 = Mode A
01 = Mode B
10 = Mode C
11 = Unused at this time

4.4.2 FAS Interval Warning to STD

A warning is issued to the STD CPU one second before a trigger is sent to the camera. This warning bit is removed when the camera is triggered.

Note: This warning is only reported for photo intervals greater than one second.

4.4.3 Photo Confirmation Bit to STD

This bit is reset to a zero when an interval warning is issued, when read by the STD CPU, or when a camera trigger is issued. It is set to a one when a print complete is detected. The presence of this bit is therefore an indication that the requested photo was taken and data annotation performed. A successful photograph would be indicated by the confirmation bit going high. If it doesn't, the status bits will contain the failure code.

4.4.4 Data and Ready Ports

Data is transferred from one CPU to another using four ports, two for data and two for handshake. The data ports can be written to by one CPU and read from by the other. The ready "ports" are actually a flip-flop reflecting the status of the data port. It is read by either CPU but is set or reset only under specific conditions as described below.

When a data port is written to, the corresponding ready bit is automatically set to a one. The intended target of this data sees this and takes the data. This action resets the ready bit to a zero telling the sender to send another byte by writing to that data port. This action can continue until all data has been transferred.

(FAS Data to STD Port 100H Read) (Data Ready FAS Port 106H Bit 0)
(STD Data to FAS Port 101H Write) (Data Ready STD Port 105H Bit 0)

4.5 Communication Protocol

The communication Protocol used on the STD BUSS to control the FAS functions is similar to that used to control a computer peripheral such as a printer. Each control function has an ASCII control character sequence assigned to it and the data blocks transferred have a fixed format adhered to by both the STD BUSS CPU and the FAS CPU.

The intent here is simple: To keep the hardware interface, (number of I/O ports used on the STD BUSS) simple while not limiting functionality. Also, as needs change functions can be added by changing only the software and using existing communication ports.

The Data Ready Flip-Flop implementation also allows for relatively high-speed data communication without requiring elaborate hardware timing circuitry.

Section 5
FAS Software Description

5.0 General

The software used to implement the FAS functions consists primarily of two elements. The firmware existing in the EPROM on the FAS CPU Card itself and the Turbo Pascal procedures supplied for inclusion into the larger IDGE software.

The firmware performs all functions required by the FAS CPU Card hardware creating an intelligent camera control and annotation system. This system is largely autonomous to the STD BUSS CPU once the FAS CPU has been given the data and control commands necessary to perform the camera intervalometer and annotation functions. The STD BUSS CPU can determine the status and health of the FAS CPU at any time by reading certain ports as discussed above. Should it be required to override the FAS CPU Card's control of the camera, this is possible through the STD BUSS.

The Turbo Pascal procedures are supplied with commented source code to allow integration directly into the larger IDGE Software.

These procedures allow the following control functions and data transfer to take place.

5.1 FAS CPU Camera Trigger Enable/Disable

(Set Bit 3 of Port 108H to Enable)
(Reset Bit 3 of Port 108H to Disable)

This procedure allows the STD BUSS CPU to disconnect the FAS CPU from the camera trigger circuitry. The hardware used to implement this function is independent of the FAS CPU and allows the STD BUSS to override the FAS camera control should a problem develop during a mission.

5.2 STD CPU Camera Trigger Command

(Set Bit 2 of Port 108H and Reset Bit 2 of Port 108H after 10ms)

This causes the camera to trigger immediately and under direct control of the STD BUSS CPU. This is available at all times regardless of the state of the disconnect hardware described above or the FAS CPU itself.

5.3 FAS CPU Reset Command

(Reset Bit 0 of Port 108H and Set Bit 0 of Port 108H after 10ms)

This causes a hardware reset of the CPU on the FAS card and is available as a means of attempting recovery from a failure.

5.4 Read Hardware Frame Counter

(Read from Port 107H)

This procedure returns the eight bit contents of the frame counter on the FAS Card. This count is a hardware function of the camera interface circuitry and is not affected by the state of the FAS CPU.

5.5 Reset Hardware Frame Counter

(Set Bit 1 of Port 108H and Reset Bit 1 of Port 108 after 1ms)

This function will clear the frame counter to zero under the control of the STD BUSS CPU directly.

5.6 SET FAS Annotation MODE

The FAS has three modes under which data annotation is performed. These modes (A, B, and C) are briefly described below.

(Write "^M" to Port 101H followed by an ASCII "A", "B", "C")

- MODE A: In this mode all 28 digits to be annotated onto the film are supplied by the STD BUSS CPU. The characters can be any of the standard ASCII characters defined by the ASCII 20 (hex) through 5F (hex). This includes all uppercase letters, all numbers and most punctuation symbols. The annotation is:

  tttttttttttttttttttttttttttt t = text

- MODE B: In this mode the FAS CPU supplies camera ID, frame number, and GMT. The STD BUSS CPU provides 16 characters of text The format is:

  CFFFttttttttttttttttHHMMSSSS

  C = Camera ID
  FFF = Frame Number
  t = Text
  HH = Hours
  MM = Minutes

SSSS = Seconds of 0.01s

- MODE C: In this mode the FAS CPU provides GMT and the STD CPU will provide 19 text characters. The format is: tttttttttttttttttttDHHMMSSSS

  t = Text
  D = Day (1's digit of day of month counter)
  HH = Hours
  MM = Minutes
  SSSS = Seconds to 0.01s

  The text data is sent using a separate procedure described below. The procedure discussed here will only SET the mode.

5.7 Send Last Photo "X" Switch Data to STD BUSS CPU

(Write "^N" to Port 101H and Read 28 bytes from Port 100H)

Each time a photo is taken the "X" switch is used to freeze the data for use later to perform the annotation when the film is advanced. This data is held in memory until the next photo is taken. The STD BUSS CPU can request this data at any time up to .01 seconds prior to the next photo interval. During this .01 second window the FAS CPU will not honor a request for data from the last photo. Instead, the request is noted but not acted upon until after the impending photo is taken. Therefore, in this case, the data reported will reflect the photo just taken.

5.8 Receive Text from STD BUSS CPU

(Write "^O" to Port 101H followed by 28 ASCII characters)

This procedure allows a string of character data to be sent to the FAS CPU. The data is interpreted as text data to be printed according to the current mode as described in section 5.6 above. If more characters are sent than needed, the extra data is ignored.

5.9 Set GMT Time

To set clock: Write "^P" followed by 5 bytes of binary data in the following format: day (0-9), hours (0-23), minutes (0-59), seconds (0-59), hundreds of seconds (0-99).

To start clock running: Write "^G" to Port 101H.

Using this procedure the STD BUSS CPU can set the RTC on the FAS Board. A string of data will define the current date and time to be loaded into the clock, but the clock will not be started until a separate "GO" command is sent. In this way more than one FAS Card can be set to the same time.

5.10 "GO" Command to Start Clock

(Write "^G" to Port 101H)

As discussed above, this procedure will start the RTC counting after time has been loaded.

5.11 Set Intervalometer Parameters

Write "^Q" to Port 101H followed by the number of photos (1 byte binary, 0-250). Next, send two bytes of binary (MSB first) photo interval, 0-14400.

This allows the STD BUSS CPU to give the FAS CPU information required to perform a photo sequence. This information includes the quantity of pictures to be taken and the time interval between pictures. The quantity will range from 1 to 250 and the time interval from 0.25 to 3600 seconds in increments of 0.25 seconds.

The accuracy of this interval is $\pm$0.01 second and the time annotated onto the film is the time at which the "X" switch was received.

5.12 "GO" Command to Start Photo Sequence

(Write "^R" to Port 101)

This procedure causes the photo sequence to begin by taking one picture immediately and the next photo one "time interval" later and continuing until the entire "quantity" of pictures is taken.

5.13 FAS Camera Interface STATUS Code

Using this procedure the STD BUSS CPU is able to determine the status and health of the FAS CPU and camera interface circuitry by reading the FAS status bits from port 104H. This port is a dedicated port assigned to a unique STD BUSS I/O address and is available to the STD BUSS CPU at any time, independent of the FAS CPU. The FAS CPU will simply update the data in this port as the status of the hardware changes. Some of these status bits are only updated after a FAS self test command. This is summarized below: See Section 4.4.1.

- Camera Error Code: (updated as camera status changes)
  (Read from Port 104H)

  a) Camera trigger sent but no "X" switch received.
  b) "X" switch received but no encoder pulses received.
  c) Encoder pulses received but terminated before a print complete.

- FAS CPU Status Bits

  a) Timeout reset has occurred:

This item is reported independent of the FAS CPU and software. This will indicate a probable failure of the FAS CPU and as such will only be indicating the hardware attempting to reset the CPU and all hardware with the exception of the hardware frame counter.

.

  b) Self Test Result Bits:

    - EPROM check sum verify/fail
    - RAM test pass/fail
    - CPU test pass/fail
      (See Section 4.4.1 for error codes.)

These tests are performed upon power up and request, (by STD CPU), and results are reported back.

5.14 Cancel Current Photo Sequence

(Write "^S" to Port 101H)

Using this procedure the STD BUSS CPU can stop the current photo taking session at any time.

5.15 Interval Pre-Warning Flag

(Read from Port 103H bit 0 only)

By reading this dedicated STD BUSS I/O port the STD BUSS CPU is able to detect (one second ahead of time) when a photo is about to be taken. See Section 4.4.2.

5.16 Photo Confirmation to STD

(Read from Port 102H bit 0 only)

By reading this dedicated STD BUSS I/O Port, the STD BUSS CPU can determine if an expected photo was actually taken. This port should be reset to a zero one quarter second prior to a photo, when a photo is triggered, or when the port is read by the STD CPU. This bit is set to a one if a photo is successfully taken. If the photo is unsuccessful, the STD BUSS CPU should read the camera status bits to determine the probable cause of the failure. See section 4.4.1.

5.17 Photo Count This Sequence

(Write "^X" to Port 101H and read 1 byte binary from 100H)

The FAS is capable of reporting to the STD BUSS CPU how many photos of the current sequence have been taken. This count will increment every time the camera is triggered, up to the total quantity of photos required for the current sequence.

5.18 Perform FAS Communication Port Test

(Write "^Y" to Port 101H)

This procedure invoke the test routines on the FAS board to exercise all the communication hardware between the FAS and the STD BUSS CPU's.

5.19 Request FAS Hardware Self Test

(To perform for all tests, write "^T" to Port 101H followed by an ASCII A)

Using this procedure the STD BUSS CPU can request the FAS to perform the following checks:

a) EPROM check/sum verify. (Write "^T" followed by an ASCII E to Port 101H)

b) RAM test with all ones, all zeros, and alternating ones and zeros. (Write "^T" followed by an ASCII R to Port 101H)

c) CPU self check routines. (Write "^T" followed by an ASCII C to Port 101H)

The results of these tests are made available to the STD BUSS CPU in the above mentioned STATUS port. (See sections 4.4.1 and 5.13.)

5.20 Report Annotation Mode to STD

(Write an "^L" to Port 101H and Read one byte of ASCII from Port 100H)

The STD BUSS CPU can ask the FAS CPU which mode the annotator is currently in (A, B, or C). This software procedure performs a request using the data/ready ports (STD to FAS) and receives an answer using the data/ready ports (FAS to STD). The FAS card status port, a dedicated STD BUSS I/O Port, also contains this mode code.

5.21 Report Time Left to Next Scheduled Photo

(Write a "^V" to Port 101H followed by two reads from Port 100H)
(Read two bytes binary MSB first)

The STD BUSS CPU can access the intervalometer timer in the FAS CPU by requesting this item. The FAS will return the number of increments left until the next scheduled photo as a binary number 0 to 14,400. This would represent the time left in .25 second increments, (i.e. 3600 1/4 second intervals = 14,400).

5.22 Set Software Frame Counter

(Write "^H" to Port 101H followed by one byte binary frame count 0-250)

This procedure allows the STD CPU to set the software frame counter to any value between 0 and 250.

1.0 INTRODUCTION

The following test procedure is meant to provide a means of verifying the functional operation of the film annotation controller card (FAS) and camera. Each test outlined within this procedure is designed to test a specific portion of the FAS hardware as well as the command sequences associated with that hardware.

These test procedures are not meant to be a means of trouble shooting the FAS card but rather they are meant to provide documentation as to the functionality of each FAS card. After each FAS card has undergone preliminary testing and it is believed to work properly it will be subjected to the test procedures contained within this document to verify complete functionality.

2.0 TEST SET-UP

For all of the test procedures outlined in this document it is assumed that the person performing the tests has obtained the software, hardware and test equipment outlined in the next two sections. In addition, it is assumed the the individual executing the test has already installed the FAS card in the STD computer, has applied power, and has run the FASTEST.EXE test program provided by Miletus Associates.

2.1) Hardware & Software: The following is a list of the hardware necessary to perform the tests contained in this document:

1. Video monitor and keyboard
2. Prolog computer card # 7890A-05
3. Prolog computer card # 7350C-01
4. Prolog computer card # 7717-01
5. Prolog computer card # 7391A-01 & 7390-02
6. Prolog computer card cage
7. Prolog compatible power supply
8. FAS controller card
9. Modified 35mm camera
10. Camera connection cable, # 8-1094
11. FASTEST.EXE test program
12. Prolog extender card.

2.2) Test Equipment: The following is a list of test equipment necessary to perform the tests contained in this test specification:

1. Hewlett Packard logic analyzer model 1631D (or equivalent machine capable of capturing and analyzing the microprocessor BUSS cycles of the National Semiconductor NSC-800 CPU running at eight Mhz)

2. Tektronix 100 Mhz oscilloscope model 2236 (or equivalent)

3. Frequency Counter with a display accuracy of 10 PPM or better used to measure 1Hz clock counter ticks (Tektronix Scope 2236 with option 001)

3.0 TEST PROCEDURES

3.1 TEST FAS CPU RESET COMMAND

SETUP: Issue the "F" command and select mode "A". Issue the "M" command. Note: the mode changes to "A". Issue the "C" command. Followed by the "M" again.

RESULT: After the mode is changed to "A", a reset command will restore the mode to "B". The last "M" command will show Mode = B to pass this test.

3.2 TEST FAS STATUS CODE

SETUP: Exit the FASSTD test program by pressing "Z". Turn off the power to the STD computer and wait at least ten seconds. Next, apply power to the STD computer and run the FASTEST test program. Finally, issue the "M" command.

RESULT: After issuing the "M" command, the mode status should equal "B", the card status should equal zero, and the camera status should equal zero. If this is not the result this test has failed.

3.3 PERFORM FAS COMMUNICATION TEST

SETUP: Issue the "R" command from the FASTEST program.

RESULT: After waiting at least four seconds note the FAS error massage value located in the lower right-hand side of the computer screen. If the returned error code equals zero the communication port test has passed. Any other response constitutes a communication port failure.

3.4 PERFORM FAS HARDWARE TEST

SETUP: Issue the "S" command and select "A" as a response to the computer prompt.

RESULT: After waiting at least five seconds issue the "M" command to read the FAS status latch. If the FAS card error code result equals zero the hardware test has passed. Any other response constitutes a hardware failure.

3.5 TEST STD CAMERA TRIGGER COMMAND

SETUP: Connect FAS camera cable and camera to the FAS camera card. Apply power to the camera and Issue the "B" command.

RESULT: If the camera triggers once for each time the "B" command is sent, the test has passed.

3.6 TEST HARDWARE FRAME COUNTER

SETUP: First, send the "D" command to read the contents of the hardware frame counter. Next, issue the "B" command to trigger the camera. Lastly, issue the "D" command a second time and note the current count value. (Carefully repeat this test several times to insure proper operation.)

RESULT: Each time the camera is triggered the frame count should increase by one count and only one count. If after repeating this test twenty times the frame count increases by exactly twenty, then this test has been passed.

3.7 TEST HARDWARE FRAME COUNTER RESET

SETUP: Issue the "E" command followed by the "D" command. If the returned value of the frame counter is zero then issue the "B" command followed by the "D" command.

RESULT: If after performing this test the frame counter value is one then this test has been passed.

3.8 TEST THE SEND DATA TO FAS COMMAND

SETUP: Type the "H" command and enter "ABCDEFGHIJKLMNOP".

RESULT: After waiting at least two seconds, note the result of the FAS error code. If the error code is zero then this test has passed.

3.9 TEST THE SEND LAST "X" SWITCH DATA

SETUP: Send the "B" command followed by the "G" command.

RESULT: If the data received from the FAS is as follows, this test has passed. "CFFFABCDEFGHIJKLMNOPHHMMSSSS"

C = Camera ID, FFF = Frame count, HHMMSSSS = Time

NOTE! Camera ID, Frame count, and Time are undefined at this point in the test; therefore, it should not concern the operator that these fields are meaningless. However, what is important is that the text matches: "ABCDEFGHIJKLMNOP". (This data was entered in step 3.8)

3.10 TEST THE ANNOTATION MODE COMMAND

SETUP: Issue the "F" command and enter "A" in response to the computer prompt. Enter the "H" command and type in a unique combination of data. Next, enter the "B" command to trigger followed by the "G" command. Check the data that is returned from the FAS and insure that it is identical to the data that was just sent. Repeat this test with the exception of entering a "C" in response to the computer prompt and realizing that the data returned will be comprised of the text sent followed by day and time data which at this point are undefined.

RESULT: If the data returned from the FAS is identical to the data sent to the FAS this test has passed.

3.11 TEST FAS REAL TIME CLOCK

SETUP: Issue the "F" command and enter mode "C". Send the "I" command and enter "123450000" and press return. Next, issue the "J" command to start the clock. Finally, type the "B" command followed by the "G" command.

RESULT: Verify that the time returned is equal to the time sent plus the time between the issuing of "J" command and the sending of the "B" command. If it is found that the time is correct within reasonable tolerances then this test has passed.

3.12 TEST FAS INTERVALOMETER

SETUP: First, setup the analyzer in the timer mode to start and stop on a fetch from address 0B77 hex. Second, Send the "K" command and enter a request to take thirty pictures on an interval of one (.25 seconds). Third, issue the "L" command to start the photo sequence and verify that the camera is being triggered.

RESULT: Verify that the analyzer's measured time is 250 ms which will constitute a passing of this test.

Note: If an analyzer is not available, a stop watch can be used to give an approximate time. This test should actually be done by the manufacturer and fine tuned to factory (manufacturer) specifications. The stop watch will give a ball park time which is all that is necessary since adjustments cannot be made at the user level.

3.13 TEST THE CANCEL PHOTO SEQUENCE COMMAND

SETUP: Issue the "L" command and before the camera has completed it's thirty photographs enter the "N" command.

RESULT: Observe whether the camera stops triggering promptly after the "N" command is issued. If this behavior is observed then this test has passed.

3.14 TEST THE INTERVAL PRE-WARNING FLAG

SETUP: Issue the "K" command and enter 250 photos at 16 intervals (every 4 seconds). Issue the "L" followed by the "Y" command.

RESULT: When the time remaining until the next scheduled photo counts down below 5, the interval pre-warning flag will change to say "Photo Pending." When the count goes to zero, which is usually not seen on the screen, the camera will click and the test program reports a "Successful Photo Taken" message. If this occurs, then the test has been passed. Issue the "N" and "Y" to stop this test.

3.15 TEST THE PHOTO CONFORMATION BIT

NOTE: It is recommended that the following test be performed by the manufacturer ONLY.

SETUP: Turn off the power. Disconnect jumper (W3) or take out pin 25 of the camera Berg connector located at J3 on the FAS card and after power up, issue the "B" command. Next, observe the state of the photo confirmation, command "P". If the photo confirmation status is "No Photo Taken" then turn off power and reconnect the encoder pulses. Then turn on power and issue the "B" command a second time. This time a successful photo confirmation status should be observed.

RESULT: If the operator observes the above conditions this test has passed. (Note: reconnect jumper (W3)

3.16 TEST THE PHOTO SEQUENCE COUNT

SETUP: Start the camera by taking a photo sequence and then issue the "Q" command repeatedly.

RESULT: The operator should observe the sequence photo counter "Q" to be incrementing. If this observation is made this test has passed. Use command "N" to stop the photo sequence.

3.17 TEST THE REPORT ANNOTATION MODE COMMAND

SETUP: First issue the "F" command and respond "A" to the computer prompt. Next, issue the "T" command followed by the "M" command. Repeat for modes "B" and "C".

RESULT: If the annotation mode that is reported in response to the "T" and "M" commands is "A", "B", "C" then this test has passed.

3.18 TEST THE REPORT TIME TO NEXT PHOTO COMMAND

SETUP: Start the camera taking photos on two second intervals. Next, issue the "U" command repeatedly and verify that the time to next photo count is decreasing.

RESULT: If it is found that the time to next photo count is decreasing then this test has been passed. Use command "N" to stop the photo sequence.

3.19 TEST PHOTO EXPOSURE TIMES

NOTE: It is recommended that the following test be performed by the manufacturer ONLY.

SETUP: Attach channel one of the oscilloscope to pin 60 or Test Point One (TP1) of the PNT EPLD (U3). The exposure time "E" 16us, has already been pre set by the jumper between pins 5 and 12 of U4. Start a photo sequence. Make sure that the positive going pulse width is approximately 16us.

RESULT: A 16us + 1 - 2us positive pulse width should be observed.

3.20 TEST REAL TIME CLOCK CALIBRATION

NOTE: It is recommended that the following test be performed by the manufacturer ONLY.

SETUP: Issue the "V" command to set the FAS real time clock into the 1Hz interrupt mode. After issuing this command the FAS will remain in the 1Hz interrupt mode for three minutes or until any other key is pressed. Next, connect the frequency counter to pin 22 of U9.

RESULT: Verify that the frequency measurement is 1HZ +/- 10 PPM. However, the operator should not be concerned if the frequency count jumps +/- 30 PPM due to the sampling time of the frequency counter.

3.21 PERFORM FILM TEST

NOTE: It is recommended that the following test be performed by the manufacturer ONLY.

SETUP: Load the camera with approximately thirty frames of film and take photos with fixed data in all of the print modes.

RESULT: Insure that the annotated data appears correctly on the film without any scratches caused by the annotation head. Also insure that the camera is advancing film correctly.

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)
{$R-}     {Range checking off}
{$B+}     {Boolean complete evaluation on}
{$S+}     {Stack checking on}
{$I+}     {I/O checking on}
{$N-}     {No numeric coprocessor}
{$M 65500,16384,655360} {Turbo 3 default stack and heap}

(***********************************************************)
(***********************************************************)
(***                                                     ***)
(***          FAS AND STD UNIT PROCEDURES                ***)
(***                 REVISION 2.0                        ***)
(***                                                     ***)
(***               BY: TOM CAVALLI                       ***)
(***                  JUNE 1988                          ***)
(***            LAST MODIFIED:13 MARCH 1989              ***)
(***            FILENAME: FASSTD20.TP5                   ***)
(***            COMPILES: FASSTD20.TPU                   ***)
(***                                                     ***)
(*** 9 DEC 88: Load_the_FAS_software_frame_count(VAR     ***)
(***           FAS_ERROR_CODE, NEW_FRAME_COUNT :BYTE);   ***)
(***           was added.                                ***)
(***                                                     ***)
(***14 DEC 88: Modified FAS Hardware Self Test to state***)
(***           when the test has succesfully finished.   ***)
(***                                                     ***)
(***16 DEC 88: Converted the FASSTD procedures into a    ***)
(***           unit.  The private GLOBAL variable,       ***)
(***           FAS_TRIGGER_ENABLE_SAVE, allows those     ***)
(***           procedures which set the FAS control      ***)
(***           bits to function with out passing the     ***)
(***           current trigger enabled/disabled value    ***)
(***                                                     ***)
(***********************************************************)
(***********************************************************)
```

(* Miletus Associates, Inc.   STD to FAS Unit Procedures *)

(* The following TURBO PASCAL (V5) procedures support the operation of the Miletus Film Annotation System (FAS) camera card.

There are twenty two procedure calls and various utility procedures that will activate the FAS card. Each procedure call contains the parameters necessary for correct operation. These parameters provide the communication path between the STD and FAS. A list of the these parameters can be found the FASTEST.TP4 source code. For instance, an error code parameter is passed between each procedure and reports the error status. This error code should be checked by the user's program to verify successful completion of the called procedure. In PASCAL,
IF A_FAS_Error_Has_Occured(FAS_ERROR_CODE) is TRUE then the called procedure has failed.

The user can NOT call any of the utility procedures which follow EXCEPT:

```
A_FAS_Error_Occured
Get_Error_Code_Number
Get_Error_Code_Caller
FAS_Power_Up
```

Upon power up the user must first call the Fas_Power_Up procedure which will initialize the FAS: the FAS Annotation mode equals 'B', and the FAS camera trigger is enabled.

The FASTEST.TP5 test program serves as an example where the uses clause contains FASSTD20 so that a screen menu program can test the FAS. Although the FASTEST.TP5 program only displays the error code after each procedure call, the user must perform an error code check. *)

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

UNIT FASSTD20;
INTERFACE
uses crt;   (* for the DELAY() procedure *)

(******************** TYPE DECLARATIONS *******************)

TYPE
     STRING_4    = STRING[4];  (*FOR THE COMMUNICATION TEST*)
     STRING_10   = STRING[10]; (* FOR CONTROL CODE + GMT   *)
     STRING_9    = STRING[9];  (* FOR GMT  'DHHMMSSSS'     *)
     STRING_28   = STRING[28]; (* FOR TEXT ANNOTATION      *)
     STRING_29   = STRING[29]; (* FOR CONTROL CODE + TEXT  *)
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*      The following FUNCTIONS and PROCEDURES may be used*)
(* by the host program.                                  *)

FUNCTION  A_FAS_Error_Occured(VAR FAS_ERROR_CODE : BYTE)
          : BOOLEAN;

PROCEDURE Get_Error_Code_Number(VAR
                    FAS_ERROR_CODE        : BYTE;
                    VAR
                    FAS_ERROR_NUMBER      : BYTE);

PROCEDURE Get_Error_Code_Caller(VAR
                   FAS_ERROR_CODE        : BYTE;
                   VAR
                   FAS_ERROR_CALLER      : BYTE);

PROCEDURE FAS_Power_Up(VAR
                FAS_ERROR_CODE     : BYTE);

PROCEDURE Set_FAS_Camera_Trigger_Control(VAR
                    FAS_ERROR_CODE     : BYTE;
                    FAS_TRIGGER_ENABLE : BOOLEAN);

PROCEDURE STD_CPU_Trigger_Camera_Command(VAR
                    FAS_ERROR_CODE     : BYTE);

PROCEDURE FAS_CPU_Reset_Command(VAR
                      FAS_ERROR_CODE     : BYTE);

PROCEDURE Read_Hardware_Frame_Counter(VAR
                 FAS_ERROR_CODE,
                 FAS_HARDWARE_FRAME_COUNTER : BYTE);

PROCEDURE Reset_Hardware_Frame_Counter(VAR
                    FAS_ERROR_CODE     : BYTE);

PROCEDURE Set_FAS_Annotation_Mode(VAR
                  FAS_ERROR_CODE : BYTE;
                  VAR
                  NEW_FAS_ANNOTATION_MODE : CHAR);

PROCEDURE Get_Last_Photo_Data(VAR
                FAS_ERROR_CODE            : BYTE;
                VAR
                LAST_PHOTO_X_SWITCH_DATA :  STRING_28);

PROCEDURE Send_Text_To_FAS(VAR
                  FAS_ERROR_CODE  : BYTE;
                  VAR
                  TEXT_TO_FAS     : STRING_28);

PROCEDURE Set_GMT(VAR
```

```
                  FAS_ERROR_CODE     : BYTE;
                  VAR
                  NEW_GMT            : STRING_9);

PROCEDURE Start_FAS_Clock(VAR
                          FAS_ERROR_CODE  : BYTE);

PROCEDURE Set_Intervalometer(VAR
                        FAS_ERROR_CODE              : BYTE;
                        VAR
                        NUMBER_OF_PHOTOS_TO_TAKE : BYTE;
                        VAR
                        INTERVAL_BETWEEN_PHOTOS   : INTEGER);

PROCEDURE Start_Photo_Sequence(VAR
                               FAS_ERROR_CODE  : BYTE);

PROCEDURE Get_FAS_Status(VAR
                         FAS_ERROR_CODE,
                         FAS_STATUS_CODE : BYTE);
PROCEDURE Cancel_Photo_Sequence(VAR
                                FAS_ERROR_CODE  : BYTE);

PROCEDURE Get_Photo_Pre_Warning_Flag(VAR
                        FAS_ERROR_CODE              : BYTE;
                        VAR
                        INTERVAL_PRE_WARNING_FLAG : BOOLEAN);

PROCEDURE Get_Photo_Confirmation_Flag(VAR
                        FAS_ERROR_CODE              : BYTE;
                        VAR
                        PHOTO_CONFIRMATION_FLAG   : BOOLEAN);

PROCEDURE Get_Photo_Count(VAR
                          FAS_ERROR_CODE         : BYTE;
                          VAR
                          NUMBER_OF_PHOTOS_TAKEN : BYTE);

PROCEDURE Perform_FAS_Communication_Test(VAR FAS_ERROR_CODE
                                             :BYTE   );

PROCEDURE Perform_FAS_Hardware_Test(VAR
                          FAS_ERROR_CODE     : BYTE;
                          VAR
                          FAS_HARDWARE_TEST  : CHAR;
                          VAR
                          FAS_HARDWARE_TEST_FLAG : BOOLEAN);

PROCEDURE Report_Annotation_Mode(VAR
                            FAS_ERROR_CODE            : BYTE;
                            VAR
                            REPORTED_ANNOTATION_MODE : CHAR);
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

PROCEDURE Report_Time_To_Next_Photo(VAR
                     FAS_ERROR_CODE             : BYTE;
                     VAR
                     REPORTED_TIME_TO_NEXT_PHOTO : INTEGER);

PROCEDURE Load_the_FAS_software_frame_count(VAR
                                        FAS_ERROR_CODE,
                                        NEW_FRAME_COUNT
                                                :BYTE);

PROCEDURE FAS_Clock_Calibration(VAR
                           FAS_ERROR_CODE    : BYTE);
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*      The above PROCEDURES are implementated below.  In *)
(* addition to all the required FAS PASCAL PROCEDURES     *)
(* described in section 4.1 through 4.22, there are       *)
(* private utility PROCEDURES which perform I/O between   *)
(* the FAS and STD, and there is one private GLOBAL       *)
(* variable, called FAS_TRIGGER_ENABLE_SAVE, which allows *)
(* the host STD program to no longer pass the             *)
(* FAS_TRIGGER_ENABLE flag to every procedure which sets  *)
(* the FAS control bits.                                  *)
(*      PROCEDURE 4.1, Set_FAS_Camera_Trigger_Control,    *)
(* enables/disables the FAS trigger and defines the value *)
(* of the private global variable.                        *)

IMPLEMENTATION

VAR
     FAS_TRIGGER_ENABLE_SAVE        : BOOLEAN;
```

(* Miletus Associates, Inc.   STD to FAS Unit Procedures *)

```
(*    The following twelve utility procedures perform the
necessary input and output between the FAS and the
Miletus FAS PASCAL PROCEDURES.
     The user is encouraged to use
the function A_FAS_Error_Occured(FAS_ERROR_CODE) to verify
that each procedure performed without error.  And, use the
two error decoding procedures, Get_Error_Code_Number  and
Get_Error_Code_Caller to aid in writing an error recovery
routine.
     The first two utility routines perform the I/O by
defining an address and data byte parameter.  Valid FAS
addresses consists of the following:
      Address         Use
       $100           FAS data to STD                read only
       $101           STD data to FAS               write only
       $102           FAS photo confirmation         read only
       $103           FAS interval pre-warning       read only
       $104           FAS status                     read only
       $105           Handshake                      read only
       $106           no use
       $107           Hardware frame counter         read only
       $108           Control bits                  write only
*)


(***************** GETS FAS HARDWARD DATA *****************)
PROCEDURE Get_FAS_Hardware_Data(VAR
                                FAS_ADDRESS    : INTEGER;
                                VAR
                                DATA_READ      : BYTE);
BEGIN

     DATA_READ   :=  PORT[FAS_ADDRESS];

END;


(**************** WRITES BYTE TO FAS DATA INPUT ***********)
PROCEDURE Write_Data_Byte_To_FAS(VAR
                                 DATA_BYTE    : BYTE);

BEGIN

    PORT[$101]  := DATA_BYTE;

END;
```

(*   The following three utilities are used for error detection. The error code parameter is encoded to include the called procedure and error number by using the first two significant digits for the procedure numbered 00 thru 21. "00" defines the power-up procedure. "01" thru "21" defines the paragraph under section 4 which describes each procedure. The units digit contains the error code which goes as:

FAS_ERROR_NUMBER	MEANING
0	No error has occured
1	The FAS is not ready to receive data.
2	The FAS has not acknowledged the receipt of the data just sent.
3	The FAS has data ready for the STD before data was requested.
4	The FAS has no data for the STD after data was requested.
5	Improper parameter format.
6	Invalid mode, test selection, or annotation text character.
7	FAS failed to send correct status
8	FAS failed to send correct data

Numbers 1, 2, 3, and 4 pertain mainly to the handshake bits between the STD and FAS. An unknown glich may cause the FAS to hang-up and a RESET, FAS403.PAS, may be the only remedy here. On numbers 5 and 6 read the comments of each procedure to learn what the expected format, mode, test and text should be. Finally numbers 7 and 8 explain where the communication test has failed, FAS418.PAS.
*)

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(************* CHECKS IF A FAS ERROR OCCURED *************)
FUNCTION  A_FAS_Error_Occured(VAR FAS_ERROR_CODE : BYTE)
          : BOOLEAN;
BEGIN
     IF FAS_ERROR_CODE = (FAS_ERROR_CODE DIV 10) * 10 THEN
           A_FAS_ERROR_OCCURED := FALSE
       ELSE
           A_FAS_ERROR_OCCURED := TRUE;
END;


(************* GETS THE FAS ERROR CODE NUMBER; 0 TO 9 *****)
PROCEDURE Get_Error_Code_Number(VAR
                              FAS_ERROR_CODE        : BYTE;
                              VAR
                              FAS_ERROR_NUMBER      : BYTE);
BEGIN
     FAS_ERROR_NUMBER := FAS_ERROR_CODE  -
                               ((FAS_ERROR_CODE DIV 10)*10);
END;


(************* GETS THE FAS ERROR CODE CALLER; 00 TO 21 ***)
PROCEDURE Get_Error_Code_Caller(VAR
                             FAS_ERROR_CODE        : BYTE;
                             VAR
                             FAS_ERROR_CALLER      : BYTE);
BEGIN
     FAS_ERROR_CALLER := FAS_ERROR_CODE DIV 10;

END;
```

```
(************* CHECK THE FAS HANDSHAKE BITS **************)
PROCEDURE FAS_Handshake_Test(VAR
                             HANDSHAKE_MASK      : BYTE;
                             VAR
                             EXPECTED_BYTE_VALUE : BYTE;
                             VAR
                             HANDSHAKE_READY     : BOOLEAN);
CONST
     HANDSHAKE_ADDRESS   : INTEGER = $105;
VAR
     HANDSHAKE_BYTE      : BYTE;
     TRY_COUNTER         : BYTE;

BEGIN
     TRY_COUNTER  := 0;
     HANDSHAKE_BYTE := 255; (* to initialize the variable *)
     REPEAT
           BEGIN

                Get_FAS_Hardware_Data(HANDSHAKE_ADDRESS,
                                      HANDSHAKE_BYTE);
                HANDSHAKE_READY := (HANDSHAKE_BYTE AND
                                    HANDSHAKE_MASK)
                                  = EXPECTED_BYTE_VALUE;
                DELAY( 23 * TRY_COUNTER );
                  (* 1035ms maximum delay *)
                TRY_COUNTER := TRY_COUNTER + 1;
           END;
     UNTIL      HANDSHAKE_READY
          OR
                (TRY_COUNTER >= 10);
END;

(************* CHECK IF THE FAS IS SENDING DATA ***********)
FUNCTION FAS_Is_Already_Sending_Data(VAR
                                     FAS_ERROR_CODE :BYTE )
         : BOOLEAN;
CONST
     HANDSHAKE_MASK      : BYTE  = $01; (* 00000001   Do  *)
     EXPECTED_BYTE_VALUE : BYTE  = 0;   (* Do=0, DATA CLEAR*)
     HANDSHAKE_READY     : BOOLEAN = FALSE;

BEGIN
     FAS_Handshake_Test(HANDSHAKE_MASK,
                        EXPECTED_BYTE_VALUE,
                        HANDSHAKE_READY);
     FAS_Is_Already_Sending_Data  := NOT HANDSHAKE_READY;
     IF NOT HANDSHAKE_READY
       THEN
           FAS_ERROR_CODE := FAS_ERROR_CODE + 3;

END;
```

```
(***************** SETS FAS CONTROL BITS *****************)
PROCEDURE Set_FAS_Control_Bits(VAR
                               DATA_BYTE           : BYTE);
(* Plus the Global: FAS_TRIGGER_ENABLE_SAVE        : BOOLEAN *)

BEGIN
     DATA_BYTE    := DATA_BYTE AND $07;  (* DATA XXXXXXXX *)
                                         (* $07  00000111 *)
                                         (*  =   00000XXX *)
     IF FAS_TRIGGER_ENABLE_SAVE THEN
             DATA_BYTE := DATA_BYTE +8;  (* ADD  00001000 *)

     PORT[$108]   := DATA_BYTE;          (* SEND 0000XXXX *)

END;
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*       THIS PROCEDURE MUST BE CALLED UPON POWER UP      *)
(* The FAS camera trigger will be enabled.                *)
(* The FAS powers up in the 'B' annotation mode.          *)

(***************** FAS POWER UP ***************************)
PROCEDURE FAS_Power_Up(VAR
                              FAS_ERROR_CODE     : BYTE);
(* Plus the Global: FAS_TRIGGER_ENABLE_SAVE      : BOOLEAN *)

CONST
     FAS_POWER_UP_CODE     : BYTE   = $09;  (*  00001001    *)
     HANDSHAKE_MASK        : BYTE   = $01;  (* 00000001  Do *)
     EXPECTED_BYTE_VALUE   : BYTE   = 0;    (* WANT Do=0    *)
     FAS_ADDRESS           : INTEGER=$100;  (* READ SO Do=0 *)
     FAS_STATUS_CODE_ADDRESS  : INTEGER  = $104;
VAR
     TRY_COUNTER           : BYTE;          (* FOR TIMEOUT  *)
     DATA_READ             : BYTE;          (* IGNORED      *)
     HANDSHAKE_READY       : BOOLEAN;       (* TRUE IF SO   *)

BEGIN
     FAS_ERROR_CODE           := 0;     (* NO ERRORS YET    *)
     FAS_TRIGGER_ENABLE_SAVE := TRUE; (*ALLOWS FAS CONTROL*)
     TRY_COUNTER              := 0;
        (* INITIALIZES FAS CONTROL BITS *)
     Set_FAS_Control_Bits(FAS_POWER_UP_CODE);

        (* RESETS THE LATCH STATUS ON THE FAS    *)
     Get_FAS_Hardware_Data(FAS_STATUS_CODE_ADDRESS,
                           DATA_READ);

        (* RESETS AND VERIFIES Do=0 FOR FAS HANDSHAKE *)
     Get_FAS_Hardware_Data(FAS_ADDRESS, DATA_READ);
     FAS_Handshake_Test(HANDSHAKE_MASK,
                        EXPECTED_BYTE_VALUE,
                        HANDSHAKE_READY);
     IF NOT HANDSHAKE_READY THEN
                FAS_ERROR_CODE := 3;

END;
```

```
(***************** SEND CODE OR DATA TO FAS ***************)
PROCEDURE Send_Byte_To_FAS(VAR
                           FAS_ERROR_CODE   : BYTE;
                           VAR
                           DATA_TO_FAS      : BYTE);

CONST
     HANDSHAKE_MASK        : BYTE   = 2;     (* AND BIT MASK   *)
     EXPECTED_BYTE_VALUE : BYTE   = 0;  (*D1=0, FAS IS READY*)
     ZEROS                 : BYTE   = 0;     (*CLEAR FAS INPUT*)
VAR
     HANDSHAKE_READY       : BOOLEAN;        (* TEST RESULTS   *)

BEGIN
                (* IS FAS READY TO RECEIVE DATA? *)
     FAS_Handshake_Test(HANDSHAKE_MASK,
                        EXPECTED_BYTE_VALUE,
                        HANDSHAKE_READY);
     IF NOT HANDSHAKE_READY THEN        (* FAS IS NOT READY *)
                       FAS_ERROR_CODE := FAS_ERROR_CODE + 1
       ELSE
                (* FAS IS READY SO SEND DATA NOW *)
         BEGIN
              Write_Data_Byte_To_FAS(DATA_TO_FAS);
                           (* SENDS DATE BYTE TO FAS AND
                              SETS HANDSHAKE BIT#1 (D1=1)
                              ADDRESS 105H TO TRUE         *)

                (* WAIT FOR FAS TO ACCEPT DATA   *)
              FAS_Handshake_Test(HANDSHAKE_MASK,
                                 EXPECTED_BYTE_VALUE,
                                 HANDSHAKE_READY);

                (* CHECK IF FAS HAS ACCEPTED DATA *)
             IF NOT HANDSHAKE_READY THEN
                  BEGIN
                       FAS_ERROR_CODE := FAS_ERROR_CODE + 2;
                       Write_Data_Byte_To_FAS(ZEROS);
                  END;
         END;
END;
```

```
(************** SENDS A STRING OF DATA TO THE FAS *********)
PROCEDURE Send_String_To_FAS(VAR
                             FAS_ERROR_CODE    :BYTE;
                             VAR
                             STRING_TO_FAS     :STRING_29);
VAR
     COUNTER             : BYTE;
     BYTE_VALUE_OF_STRING : BYTE;

BEGIN
     COUNTER      := 1;
     WHILE  (COUNTER <= LENGTH(STRING_TO_FAS))
          AND
            NOT A_FAS_ERROR_OCCURED(FAS_ERROR_CODE)  DO
       BEGIN
            BYTE_VALUE_OF_STRING := ORD(
                                    STRING_TO_FAS[COUNTER]);
            Send_Byte_To_FAS(FAS_ERROR_CODE,
                             BYTE_VALUE_OF_STRING);
           COUNTER   := COUNTER + 1;
      END;
END;
```

```
(*************** GET A DATA BYTE FROM THE FAS *************)
PROCEDURE Get_Byte_From_FAS(VAR
                            FAS_ERROR_CODE : BYTE;
                            VAR
                            DATA_READ      : BYTE );
CONST
     HANDSHAKE_MASK      : BYTE = $01;    (* 00000001 Do    *)
     EXPECTED_BYTE_VALUE : BYTE = $01;    (* WANT Do=1      *)
     FAS_ADDRESS         : INTEGER=$100; (* FAS DATA OUT   *)
VAR
     HANDSHAKE_READY     : BOOLEAN;

BEGIN
        (* DOES THE FAS HAVE DATA READY TO SEND *)
     HANDSHAKE_READY  := FALSE;
     FAS_Handshake_Test(HANDSHAKE_MASK,
                        EXPECTED_BYTE_VALUE,
                        HANDSHAKE_READY);
     IF NOT HANDSHAKE_READY THEN
                 FAS_ERROR_CODE := FAS_ERROR_CODE + 4;
     IF NOT A_FAS_ERROR_OCCURED(FAS_ERROR_CODE) THEN
                (* FAS NOW HAS DATA READY *)
               Get_FAS_Hardware_Data(FAS_ADDRESS,
                                      DATA_READ);
                            (* WILL SET Do=0 AT 105H *)
END;
```

```
(****** 4.1   FAS CPU CAMERA TRIGGER ENABLE/DISABLE *******)

PROCEDURE Set_FAS_Camera_Trigger_Control(VAR
                              FAS_ERROR_CODE     : BYTE;
                              FAS_TRIGGER_ENABLE : BOOLEAN);
(* Plus the Global: FAS_TRIGGER_ENABLE_SAVE      : BOOLEAN *)

CONST
     NORMAL_CONTROL_BYTE   : BYTE = $09;  (* 00001001 *)

BEGIN

     FAS_ERROR_CODE          := 010;

     FAS_TRIGGER_ENABLE_SAVE  := FAS_TRIGGER_ENABLE;

     Set_FAS_Control_Bits(NORMAL_CONTROL_BYTE);

END;
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*      Calling this procedure will cause the FAS to      *)
(* override software and trigger the camera.               *)

(****** 4.2   STD CPU CAMERA TRIGGER COMMAND ***************)
PROCEDURE STD_CPU_Trigger_Camera_Command(VAR
                              FAS_ERROR_CODE     : BYTE);

CONST
     NORMAL_CONTROL_BYTE    : BYTE = $09;   (* 00001001 *)
     TRIGGER_BYTE           : BYTE = $0D;   (* 00001101 *)

BEGIN
     FAS_ERROR_CODE  := 020;
     Set_FAS_Control_Bits(TRIGGER_BYTE);

     DELAY(20);                    (* PULSE WIDTH IS 20 mS *)
     Set_FAS_Control_Bits(NORMAL_CONTROL_BYTE);

     DELAY(20);                    (* BETWEEN PULSES *)

     Set_FAS_Control_Bits(TRIGGER_BYTE);

     DELAY(20);                    (* PULSE WIDTH IS 20 mS *)
     Set_FAS_Control_Bits(NORMAL_CONTROL_BYTE);

END;
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*      This procedure will reset the FAS CPU.  The user   *)
(* must be aware that the FAS annotation mode now equals   *)
(* 'B' and that the FAS camera trigger is enabled.         *)

(****** 4.3   FAS CPU RESET COMMAND ************************)

PROCEDURE FAS_CPU_Reset_Command(VAR
                                FAS_ERROR_CODE     : BYTE);

CONST
     NORMAL_CONTROL_BYTE    : BYTE = $09;  (* 00001001 *)
     RESET_CPU_BYTE         : BYTE = $08;  (* 00001000 *)

BEGIN
         (* The FAS is RESET first  *)
     FAS_ERROR_CODE  := 030;
     Set_FAS_Control_Bits(RESET_CPU_BYTE);

     DELAY(10);               (* PULSE WIDTH IS 10 mS *)

     Set_FAS_Control_Bits(NORMAL_CONTROL_BYTE);

          (* then the FAS is power up reset.   *)
   FAS_POWER_UP(FAS_ERROR_CODE);

END;
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*      The hardware frame counter is read from the FAS   *)
(* card and the value is returned in the variable         *)
(* FAS_HARDWARE_FRAME_COUNTER                             *)

(****** 4.4   READ HARDWARE FRAME COUNTER *****************)
PROCEDURE Read_Hardware_Frame_Counter(VAR
                         FAS_ERROR_CODE,
                         FAS_HARDWARE_FRAME_COUNTER : BYTE);
CONST
     FAS_FRAME_COUNTER_ADDRESS  : INTEGER  = $107;

BEGIN
     FAS_ERROR_CODE   := 040;
     Get_FAS_Hardware_Data(FAS_FRAME_COUNTER_ADDRESS,
                           FAS_HARDWARE_FRAME_COUNTER);


END;
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*      Calling this procedure cause the hardware frame    *)
(* counter to be reset                                     *)

(****** 4.5   RESET HARDWARE FRAME COUNTER ****************)
PROCEDURE Reset_Hardware_Frame_Counter(VAR
                              FAS_ERROR_CODE     : BYTE);

CONST
     NORMAL_CONTROL_BYTE        : BYTE = $09;  (* 00001001 *)
     RESET_FRAME_COUNT_BYTE     : BYTE = $0B;  (* 00001011 *)
VAR
     RESET_SOFTWARE_FRAME_CODE  : BYTE;

BEGIN
     FAS_ERROR_CODE  := 050;
     Set_FAS_Control_Bits(RESET_FRAME_COUNT_BYTE);

     DELAY(10);                 (*PULSE WIDTH IS 10 mS *)

     Set_FAS_Control_Bits(NORMAL_CONTROL_BYTE);

     RESET_SOFTWARE_FRAME_CODE  :=ORD(^W);
     Send_Byte_To_FAS(FAS_ERROR_CODE,
                      RESET_SOFTWARE_FRAME_CODE);


END;
```

```
(*      The user's program will set the variable           *)
(* NEW_FAS_ANNOTATION_MODE and call this procedure which   *)
(* will verify that the mode is A, B, or C and update the *)
(* FAS.                                                    *)

(****** 4.6   SET FAS ANNOTATION MODE *********************)
PROCEDURE Set_FAS_Annotation_Mode(VAR
                            FAS_ERROR_CODE : BYTE;
                            VAR
                            NEW_FAS_ANNOTATION_MODE : CHAR);
VAR
     SET_ANNOTATION_CODE   : BYTE;
     BYTE_VALUE_OF_MODE    : BYTE;

BEGIN
     FAS_ERROR_CODE                   := 060;
        (* VERIFY MODE IS A, B, OR C *)
     IF NEW_FAS_ANNOTATION_MODE IN ['A','B','C'] THEN
        BEGIN
            SET_ANNOTATION_CODE        := ORD(^M);
            Send_Byte_To_FAS(FAS_ERROR_CODE,
                             SET_ANNOTATION_CODE);
            IF NOT A_FAS_ERROR_OCCURED(FAS_ERROR_CODE) THEN
               BEGIN
                    BYTE_VALUE_OF_MODE   :=
                               ORD(NEW_FAS_ANNOTATION_MODE);
                    Send_Byte_To_FAS(FAS_ERROR_CODE,
                                     BYTE_VALUE_OF_MODE);
               END;
        END
      ELSE
       FAS_ERROR_CODE              := 066;

END;
```

```
(****** 4.7   GET LAST PHOTO "X" SWITCH DATA **************)
PROCEDURE Get_Last_Photo_Data(VAR
                       FAS_ERROR_CODE            : BYTE;
                       VAR
                       LAST_PHOTO_X_SWITCH_DATA : STRING_28);
VAR
     GET_TIME_CODE      : BYTE;
     DATA_READ          : BYTE;
     COUNTER            : BYTE;
     LAST_DATA          : STRING_28;

BEGIN
     FAS_ERROR_CODE      := 070;
     GET_TIME_CODE       := ORD(^N);
     IF NOT FAS_Is_Already_Sending_Data(FAS_ERROR_CODE)
       THEN
        BEGIN
         Send_Byte_To_FAS(FAS_ERROR_CODE,
                          GET_TIME_CODE);
         IF NOT A_FAS_Error_Occured(FAS_ERROR_CODE) THEN
           BEGIN
             COUNTER  := 0;
             LAST_DATA  := '                            ';
             REPEAT
               BEGIN
                 COUNTER := COUNTER + 1;
                 Get_Byte_From_FAS(FAS_ERROR_CODE,
                                   DATA_READ);
                 IF NOT A_FAS_Error_Occured(FAS_ERROR_CODE)
                 THEN LAST_DATA[COUNTER] := CHAR(DATA_READ);
               END;
             UNTIL   (COUNTER >= 28)
                  OR
                     A_FAS_Error_Occured(FAS_ERROR_CODE);
           END;
        END;
     IF NOT A_FAS_ERROR_OCCURED(FAS_ERROR_CODE)
       THEN
           LAST_PHOTO_X_SWITCH_DATA := LAST_DATA;

END;
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*      A text string, TEXT_TO_FAS, is sent to the FAS    *)
(*  unless the text string contains undefined characters  *)
(*  outside the range of 20h to 5Ah, ' ' TO 'Z'.
(****** 4.8   SEND ANNOTATION TEXT TO FAS *****************)
PROCEDURE Send_Text_To_FAS(VAR
                           FAS_ERROR_CODE   : BYTE;
                           VAR
                           TEXT_TO_FAS      : STRING_28);
VAR
     COUNTER                : BYTE;
     STRING_TO_FAS          : STRING_29;

BEGIN
     FAS_ERROR_CODE          := 080;
     COUNTER                 := 0;
     REPEAT
                COUNTER    := COUNTER + 1;
                IF ( (TEXT_TO_FAS[COUNTER]) < ' ' )
                  OR
                   ( (TEXT_TO_FAS[COUNTER]) > 'Z' )
                  THEN
                         FAS_ERROR_CODE := 086;

      UNTIL ( COUNTER >= LENGTH(TEXT_TO_FAS) )
         OR
            A_FAS_ERROR_OCCURED(FAS_ERROR_CODE);
     IF NOT A_FAS_ERROR_OCCURED(FAS_ERROR_CODE)
        THEN
            BEGIN
                 STRING_TO_FAS     := ^O + TEXT_TO_FAS +
                              '                            ';
                              (*  WILL SEND CONTROL CODE  *)
                              (* FOLLOWED BY THE TEXT THEN*)
                              (* PAD WITH SPACES TO       *)
                              (* MAKE STRING_TO_FAS       *)
                              (* EXACTLY 29 CHARACTERS    *)
                  Send_String_To_FAS(FAS_ERROR_CODE,
                                     STRING_TO_FAS);
            END;
END;
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*      The variable NEW_GMT which is set by the user's    *)
(* program is error checked and sent to the FAS.           *)

(****** 4.9   SET GMT *************************************)
PROCEDURE Set_GMT(VAR
                  FAS_ERROR_CODE      : BYTE;
                  VAR
                  NEW_GMT             : STRING_9);
VAR                                  (* DHHMMSSSS *)
     STRING_TO_FAS      : STRING_29;
     BINARY_VALUE       : BYTE;
     COUNTER            : BYTE;
     TENS               : BYTE;
     UNITS              : BYTE;
     ADAPTED_NEW_GMT    : STRING_10;
     TENS_DIGIT         : BYTE;
     UNITS_DIGIT        : BYTE;
     DECIMAL_VALUE      : BYTE;

BEGIN
     FAS_ERROR_CODE        := 090;
     STRING_TO_FAS         := ^P + '     ';

    (* CONVERT '0D', 'HH', 'MM', 'SS', AND 'SS' TO BINARY *)
    (* AND MOVE INTO STRING_TO_FAS POSITION; UPON ERROR, *)
    (* SET FAS_ERROR_CODE AND EXIT.                       *)
     ADAPTED_NEW_GMT     := '0' + NEW_GMT;
     COUNTER   := 1;
     REPEAT
           BEGIN
                COUNTER   := COUNTER + 1;  (*NEXT BYTE     *)
                TENS      := COUNTER * 2 - 3;
                UNITS     := COUNTER * 2 - 2;
                TENS_DIGIT := BYTE(ADAPTED_NEW_GMT[TENS])
                            - $30;
                UNITS_DIGIT:= BYTE(ADAPTED_NEW_GMT[UNITS])
                            - $30;
                IF (TENS_DIGIT > 9) OR (UNITS_DIGIT >9)
                  THEN
                      FAS_ERROR_CODE := 095
                  ELSE
                      BEGIN
                           DECIMAL_VALUE := TENS_DIGIT * 10
                                          + UNITS_DIGIT;
                           CASE COUNTER OF
                           3 : IF DECIMAL_VALUE > 23 THEN
                                  FAS_ERROR_CODE := 095;
                           4 : IF DECIMAL_VALUE > 59 THEN
                                  FAS_ERROR_CODE := 095;
                           5 : IF DECIMAL_VALUE > 59 THEN
                                  FAS_ERROR_CODE := 095;
                           END;
```

```
                    END;
                IF NOT A_FAS_ERROR_OCCURED(FAS_ERROR_CODE)
                  THEN
                    STRING_TO_FAS[COUNTER] :=
                      CHAR(TENS_DIGIT * 10 + UNITS_DIGIT);
          END;
     UNTIL    (COUNTER >= 6) OR (FAS_ERROR_CODE = 095);
     IF NOT A_FAS_ERROR_OCCURED(FAS_ERROR_CODE)
       THEN
          Send_String_To_FAS(FAS_ERROR_CODE,
                             STRING_TO_FAS);

END;
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*     This procedure starts the GMT clock running       *)

(****** 4.10  "GO" COMMAND TO START THE CLOCK *************)
PROCEDURE Start_FAS_Clock(VAR
                          FAS_ERROR_CODE  : BYTE);
VAR
     START_CLOCK_CODE  : BYTE;

BEGIN
     FAS_ERROR_CODE              := 100;
     START_CLOCK_CODE            := ORD(^G);
     Send_Byte_To_FAS(FAS_ERROR_CODE,
                      START_CLOCK_CODE);


END;
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*      The two variables, NUMBER_OF_PHOTOS_TO_TAKE and    *)
(* INTERVAL_BETWEEN_PHOTOS, set by the user's program are *)
(* error check and sent to the FAS.                        *)

(****** 4.11  SET INTERVALOMETER PARAMETERS ***************)
PROCEDURE Set_Intervalometer(VAR
                        FAS_ERROR_CODE           : BYTE;
                        VAR
                        NUMBER_OF_PHOTOS_TO_TAKE : BYTE;
                        VAR
                        INTERVAL_BETWEEN_PHOTOS  : INTEGER);
VAR
     SEND_INTER_CODE   : BYTE;
     HIGH_BYTE         : BYTE;
     LOW_BYTE          : BYTE;

BEGIN
     FAS_ERROR_CODE    := 110;
     SEND_INTER_CODE   := ORD(^Q);
     IF  (NUMBER_OF_PHOTOS_TO_TAKE > 250)
       OR
         (NUMBER_OF_PHOTOS_TO_TAKE < 1 )
       OR
         (INTERVAL_BETWEEN_PHOTOS > 14400)
       OR
         (INTERVAL_BETWEEN_PHOTOS < 1)
       THEN
           FAS_ERROR_CODE  := 115;
     HIGH_BYTE         := BYTE(HI(INTERVAL_BETWEEN_PHOTOS));
     LOW_BYTE          := BYTE(INTERVAL_BETWEEN_PHOTOS);

     IF NOT A_FAS_ERROR_OCCURED(FAS_ERROR_CODE)
       THEN
           Send_Byte_To_FAS(FAS_ERROR_CODE,
                            SEND_INTER_CODE);
     IF NOT A_FAS_ERROR_OCCURED(FAS_ERROR_CODE)
       THEN
           Send_Byte_To_FAS(FAS_ERROR_CODE,
                             NUMBER_OF_PHOTOS_TO_TAKE);
     IF NOT A_FAS_ERROR_OCCURED(FAS_ERROR_CODE)
       THEN
           Send_Byte_To_FAS(FAS_ERROR_CODE,
                            HIGH_BYTE);
     IF NOT A_FAS_ERROR_OCCURED(FAS_ERROR_CODE)
       THEN
           Send_Byte_To_FAS(FAS_ERROR_CODE,
                            LOW_BYTE);

END;
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*      The FAS will begin taking automatic pictures       *)

(****** 4.12  "GO" COMMAND TO START A PHOTO SEQUENCE*******)
PROCEDURE Start_Photo_Sequence(VAR
                               FAS_ERROR_CODE  : BYTE);
VAR
     START_PHOTO_SEQUENCE_CODE  : BYTE;

BEGIN
     FAS_ERROR_CODE             := 120;
     START_PHOTO_SEQUENCE_CODE  := ORD(^R);
     Send_Byte_To_FAS(FAS_ERROR_CODE,
                      START_PHOTO_SEQUENCE_CODE);

END;
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*      The status of the FAS is returned in the variable *)
(* named FAS_STATUS_CODE.                                  *)

(****** 4.13  FAS STATUS CODE TO STD **********************)
PROCEDURE Get_FAS_Status(VAR
                         FAS_ERROR_CODE,
                         FAS_STATUS_CODE : BYTE);
CONST
     FAS_STATUS_CODE_ADDRESS  : INTEGER  = $104;

BEGIN
     FAS_ERROR_CODE   := 130;
     Get_FAS_Hardware_Data(FAS_STATUS_CODE_ADDRESS,
                           FAS_STATUS_CODE);

END;
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*      This procedure will stop automatic pictures       *)

(****** 4.14  CANCEL CURRENT PHOTO SEQUENCE ***************)
PROCEDURE Cancel_Photo_Sequence(VAR
                                FAS_ERROR_CODE   : BYTE);
VAR
     CANCEL_PHOTO_SEQUENCE_CODE  : BYTE;

BEGIN
     FAS_ERROR_CODE              := 140;
     CANCEL_PHOTO_SEQUENCE_CODE  := ORD(^S);
     Send_Byte_To_FAS(FAS_ERROR_CODE,
                          CANCEL_PHOTO_SEQUENCE_CODE);


END;
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*      The TRUE value of boolean variable                  *)
(* INTERVAL_PRE_WARNING_FLAG  indicates when a picture is *)
(* about to occur.                                          *)

(****** 4.15  INTERVAL PRE-WARNING FLAG *******************)
PROCEDURE Get_Photo_Pre_Warning_Flag(VAR
                        FAS_ERROR_CODE            : BYTE;
                        VAR
                        INTERVAL_PRE_WARNING_FLAG : BOOLEAN);

CONST
     FAS_PRE_WARNING_ADDRESS  : INTEGER  = $103;

VAR
     PRE_WARNING_DATA_BYTE    : BYTE;

BEGIN
     FAS_ERROR_CODE    := 150;
     Get_FAS_Hardware_Data(FAS_PRE_WARNING_ADDRESS,
                           PRE_WARNING_DATA_BYTE);

     INTERVAL_PRE_WARNING_FLAG  :=
               (PRE_WARNING_DATA_BYTE AND $01) = $01;

END;
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*      The TRUE value the variable                         *)
(*  PHOTO_CONFIRMATION_FLAG indicated that a photo was     *)
(* taken.                                                  *)

(****** 4.16  PHOTO CONFIRMATION TO STD *******************)
PROCEDURE Get_Photo_Confirmation_Flag(VAR
                        FAS_ERROR_CODE            : BYTE;
                        VAR
                        PHOTO_CONFIRMATION_FLAG  : BOOLEAN);
CONST
     FAS_PRE_WARNING_ADDRESS  : INTEGER  = $102;

VAR
     PHOTO_CONF_DATA_BYTE    : BYTE;

BEGIN
     FAS_ERROR_CODE   := 160;
     Get_FAS_Hardware_Data(FAS_PRE_WARNING_ADDRESS,
                           PHOTO_CONF_DATA_BYTE);

     PHOTO_CONFIRMATION_FLAG  :=
                (PHOTO_CONF_DATA_BYTE AND $01) = $01;

END;
```

```
(*      The NUMBER_OF_PHOTOS_TAKEN sofar is reported       *)

(****** 4.17  PHOTO COUNT IN THIS SEQUENCE ****************)
PROCEDURE Get_Photo_Count(VAR
                          FAS_ERROR_CODE          : BYTE;
                          VAR
                          NUMBER_OF_PHOTOS_TAKEN : BYTE);
VAR
     PHOTO_COUNT_CODE    : BYTE;
     DATA_READ           : BYTE;

BEGIN
     FAS_ERROR_CODE     := 170;
     PHOTO_COUNT_CODE   := ORD(^X);
     IF NOT FAS_Is_Already_Sending_Data(FAS_ERROR_CODE)
       THEN
           Send_Byte_To_FAS(FAS_ERROR_CODE,
                            PHOTO_COUNT_CODE);
     IF NOT A_FAS_Error_Occured(FAS_ERROR_CODE)
       THEN
           Get_Byte_From_FAS(FAS_ERROR_CODE,
                             DATA_READ);
     IF NOT A_FAS_Error_Occured(FAS_ERROR_CODE)
       THEN
           NUMBER_OF_PHOTOS_TAKEN  := DATA_READ;

END;
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*      Four bytes, FFh, 00h, AAh, and 55h, are sent to    *)
(* the FAS and echoed back first on the status byte then   *)
(* on the data byte read addresses.  Any failure causes    *)
(* an error reported by FAS_ERROR_CODE.                    *)

(****** 4.18  PERFORM FAS COMMUNICATION PORT TEST *********)
PROCEDURE Perform_FAS_Communication_Test(VAR FAS_ERROR_CODE
                                                   :BYTE  );

CONST
     FAS_STATUS_ADDRESS          :INTEGER  = $104;
VAR
     COMMUNICATION_TEST_CODE     : BYTE;
     TEST_CASES                  : STRING_4;
     TEST_CHAR                   : BYTE;
     COUNTER                     : BYTE;
     FAS_STATUS_CODE             : BYTE;
     DATA_READ                   : BYTE;

BEGIN
     FAS_ERROR_CODE               := 180;
     COMMUNICATION_TEST_CODE      := ORD(^Y);
     Send_Byte_To_FAS(FAS_ERROR_CODE,
                      COMMUNICATION_TEST_CODE);
       (* BEGIN SENDING TEST PATTERNS *)
     TEST_CASES   := CHAR($FF)+CHAR($00)+CHAR($AA)+CHAR($55);
     COUNTER      := 0;
     IF NOT A_FAS_ERROR_OCCURED(FAS_ERROR_CODE)
       THEN
           REPEAT
                 COUNTER    := COUNTER + 1;
                 TEST_CHAR := BYTE(TEST_CASES[COUNTER]);
                 Send_Byte_To_FAS(FAS_ERROR_CODE,
                                  TEST_CHAR);
                 IF NOT A_FAS_ERROR_OCCURED(FAS_ERROR_CODE)
                   THEN
                       BEGIN
                            DELAY(100);   (* WAIT FOR FAS *)
                            Get_FAS_Hardware_Data(
                                         FAS_STATUS_ADDRESS,
                                         FAS_STATUS_CODE);
                            IF ((TEST_CHAR AND $DF) <>
                                  (FAS_STATUS_CODE AND $DF))
                              THEN
                                  FAS_ERROR_CODE :=187
                            ELSE BEGIN
                            Get_Byte_From_FAS(FAS_ERROR_CODE,
                                                   DATA_READ);
                            IF NOT A_FAS_ERROR_OCCURED(
                                              FAS_ERROR_CODE)
                              AND
                               (TEST_CHAR <> DATA_READ)
```

```
                                THEN
                                    FAS_ERROR_CODE := 188
                            END;
                          END;
           UNTIL
                (COUNTER >= 4)
            OR
                A_FAS_ERROR_OCCURED(FAS_ERROR_CODE);

END;
```

```
(*      This procedure causes the FAS to perform the test *)
(* specified by the FAS_HARDWARE_TEST variable set by the *)
(* user's program:                                         *)
(*   FAS_HARDWARE_TEST value       TEST PREFORMED          *)
(*          A                      All tests               *)
(*          C                      CPU self check only     *)
(*          E                      EPROM check sum verify *)
(*          R                      RAM tested              *)
(*                                                         *)
(*     The FAS_HARDWARE_TEST_FLAG returns a true value    *)
(* when the hardware test is finished. The results of     *)
(* these tests are available from the status byte reported*)
(* by procedure FAS413.                                    *)
(*    IF the FAS_HARDWARE_TEST_FLAG returns a false value *)
(* then the FAS_ERROR_CODE must be examined to determine  *)
(* where the test failed.                                  *)

(****** 4.19  PERFORM FAS HARDWARE SELF TEST **************)
PROCEDURE Perform_FAS_Hardware_Test(VAR
                          FAS_ERROR_CODE      : BYTE;
                          VAR
                          FAS_HARDWARE_TEST   : CHAR;
                          VAR
                          FAS_HARDWARE_TEST_FLAG : BOOLEAN);
VAR
     SELF_TEST_CODE          : BYTE;
     BYTE_VALUE_OF_TEST      : BYTE;
     DATA_READ               : BYTE;

BEGIN
     FAS_ERROR_CODE               := 190;
     FAS_HARDWARE_TEST_FLAG       := FALSE;
     SELF_TEST_CODE               := ORD(^T);

        (* VERIFY TEST IS A, C, E,OR R *)
     IF FAS_HARDWARE_TEST IN ['A','C','E','R']
       THEN
         BEGIN
             IF NOT FAS_Is_Already_Sending_Data(FAS_ERROR_CODE)
              THEN
                 Send_Byte_To_FAS(FAS_ERROR_CODE,
                                  SELF_TEST_CODE);
             IF NOT A_FAS_ERROR_OCCURED(FAS_ERROR_CODE)
              THEN
               BEGIN
                  BYTE_VALUE_OF_TEST  :=
                                   ORD(FAS_HARDWARE_TEST);
                  Send_Byte_To_FAS(FAS_ERROR_CODE,
                                   BYTE_VALUE_OF_TEST);
                  IF NOT A_FAS_ERROR_OCCURED(FAS_ERROR_CODE)
                   THEN
                      Get_Byte_From_FAS(FAS_ERROR_CODE,
```

```
                                                DATA_READ);
                        IF NOT A_FAS_Error_Occured(FAS_ERROR_CODE)
                         THEN
                             FAS_HARDWARE_TEST_FLAG := TRUE;
                 END;
            END
         ELSE
           FAS_ERROR_CODE  := 196;
END;
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*      This procedure sets the variable                    *)
(* REPORTED_ANNOTATION_MODE to the current FAS's value      *)

(****** 4.20  REPORT ANNOTATION MODE TO STD ***************)
PROCEDURE Report_Annotation_Mode(VAR
                           FAS_ERROR_CODE             : BYTE;
                           VAR
                           REPORTED_ANNOTATION_MODE : CHAR);
VAR
     REPORT_MODE_CODE    : BYTE;
     DATA_READ           : BYTE;

BEGIN
     FAS_ERROR_CODE     := 200;
     REPORT_MODE_CODE   := ORD(^U);
     IF NOT FAS_Is_Already_Sending_Data(FAS_ERROR_CODE)
       THEN
          Send_Byte_To_FAS(FAS_ERROR_CODE,
                           REPORT_MODE_CODE);
     IF NOT A_FAS_Error_Occured(FAS_ERROR_CODE)
       THEN
          Get_Byte_From_FAS(FAS_ERROR_CODE,
                            DATA_READ);
     IF NOT A_FAS_Error_Occured(FAS_ERROR_CODE)
       THEN
          REPORTED_ANNOTATION_MODE := CHAR(DATA_READ);

END;
```

```
(*      The time (in intervals of 1/4 seconds) before the *)
(* next automatic photo is reported in the variable       *)
(* named REPORTED_TIME_TO_NEXT_PHOTO.                     *)

(****** 4.21  REPORT TIME LEFT TO NEXT SCHEDULED PHOTO ****)
PROCEDURE Report_Time_To_Next_Photo(VAR
                     FAS_ERROR_CODE              : BYTE;
                     VAR
                     REPORTED_TIME_TO_NEXT_PHOTO : INTEGER);
VAR
     TIME_LEFT_CODE      : BYTE;
     HIGH_BYTE           : BYTE;
     LOW_BYTE            : BYTE;

BEGIN
     FAS_ERROR_CODE     := 210;
     TIME_LEFT_CODE     := ORD(^V);
     IF NOT FAS_Is_Already_Sending_Data(FAS_ERROR_CODE)
       THEN
           Send_Byte_To_FAS(FAS_ERROR_CODE,
                            TIME_LEFT_CODE);
     IF NOT A_FAS_Error_Occured(FAS_ERROR_CODE)
       THEN
           Get_Byte_From_FAS(FAS_ERROR_CODE,
                             HIGH_BYTE);
     IF NOT A_FAS_Error_Occured(FAS_ERROR_CODE)
       THEN
           Get_Byte_From_FAS(FAS_ERROR_CODE,
                             LOW_BYTE);
     IF NOT A_FAS_Error_Occured(FAS_ERROR_CODE)
       THEN
           REPORTED_TIME_TO_NEXT_PHOTO :=
                               HIGH_BYTE * 256 + LOW_BYTE;

END;
```

```
(*******4.22  Load the FAS software frame counter *********)

PROCEDURE Load_the_FAS_software_frame_count(VAR
                                              FAS_ERROR_CODE,
                                              NEW_FRAME_COUNT
                                                      :BYTE);

VAR
     STRING_TO_FAS     : STRING_29;

BEGIN
     FAS_ERROR_CODE               := 220;
     IF (NEW_FRAME_COUNT >= 0 )
       AND
        (NEW_FRAME_COUNT <= 250 ) THEN
         BEGIN
              STRING_TO_FAS := ^H + CHAR(NEW_FRAME_COUNT);
              Send_String_to_FAS(FAS_ERROR_CODE,
                                 STRING_TO_FAS);
         END
       ELSE
         FAS_ERROR_CODE     := 225;

END;
```

```
(* Miletus Associates, Inc.   STD to FAS  Unit Procedures *)

(*      The FAS clock calibration procedure should be      *)
(* commented out after testing is completed so that no     *)
(* accidental call will cause problems.                    *)

(***********  FAS CLOCK CALIBRATION  **********************)
PROCEDURE FAS_Clock_Calibration(VAR
                                FAS_ERROR_CODE    : BYTE);
VAR
     CALIBRATE_CODE                : BYTE;
     DELAY_COUNTER                 : INTEGER;

BEGIN
     FAS_ERROR_CODE                := 250;

     (* expand this comment to then END after testing  *)

     CALIBRATE_CODE                := ORD(^I);
     Send_Byte_To_FAS(FAS_ERROR_CODE,
                      CALIBRATE_CODE);
     DELAY_COUNTER  := 0;
     REPEAT
          DELAY(10);  (* WAIT 10mS *)
          IF KEYPRESSED
            THEN
                DELAY_COUNTER := 18001
            ELSE
                DELAY_COUNTER := DELAY_COUNTER + 1;
     UNTIL
          ( DELAY_COUNTER > 18000 );

     FAS_CPU_Reset_Command(FAS_ERROR_CODE);

     (* Place closing comment marker here after testing. *)

END;
END.
```

```
(* Miletus Associates, Inc.        FAS Testing PROCEDURES *)
{$R-}     {Range checking off}
{$B+}     {Boolean complete evaluation on}
{$S+}     {Stack checking on}
{$I+}     {I/O checking on}
{$N-}     {No numeric coprocessor}
{$M 65500,16384,655360} {Turbo 3 default stack and heap}

(***********************************************************)
(***********************************************************)
(***                                                    ***)
(***          FAS AND STD TEST PROCEDURES               ***)
(***                REVISION 2.0                        ***)
(***                                                    ***)
(***              BY: TOM CAVALLI                       ***)
(***                 JUNE 1988                          ***)
(***           LAST MODIFIED:13 MARCH 1989              ***)
(***           FILENAME: FASTEST.TP5                    ***)
(***                                                    ***)
(***                                                    ***)
(*** 13 MARCH 89: CHANGED INITVAR TO SHOW POWERUP       ***)
(***              FAS_TRIGGER_ENABLE := TRUE            ***)
(***                                                    ***)
(*** 13 MARCH 89: CHANGED NUMBER_OF_PHOTOS_TO_TAKE      ***)
(***              TO BE ONLY A BYTE TYPE VARIABLE       ***)
(***********************************************************)
(***********************************************************)


PROGRAM FASTEST;

Uses                  Crt,  FASSTD20;
```

```
(*********SUGGESTED FAS PASCAL TYPE DECLARATIONS***********)
     (***      All type declarations are defined in     ***)
     (*** the unit FASSTD20.                            ***)

(*****SUGGESTED FAS PASCAL GLOBAL VARIABLE DECLARATION*****)
     (***      THE FOLLOWING PASCAL PROCEDURES REQUIRE   **)
     (*** PARAMETERS THAT EITHER CHANGE THE FAS OR       **)
     (*** RETURN THE REQUESTED VALUE.                    **)
     (*** THE IDENTIFIERS LISTED HERE MAY BE USED AS     **)
     (*** THE PARAMETERS FOR THOSE PROCEDURE CALLS.      **)
     (***      Of course these variables can be defined  **)
     (*** and used as local variables.  They have been   **)
     (*** listed here for reference.                     **)
VAR                                                (* WHO   *)
                                                   (*DEFINES*)
     FAS_ERROR_CODE                 : BYTE;        (*FAS    *)
     FAS_ERROR_NUMBER               : BYTE;        (*FAS    *)
     FAS_ERROR_CALLER               : BYTE;        (*FAS    *)
     FAS_TRIGGER_ENABLE             : BOOLEAN;     (*    STD*)
     FAS_HARDWARE_FRAME_COUNTER     : BYTE;        (*FAS    *)
     NEW_FAS_ANNOTATION_MODE        : CHAR;        (*    STD*)
     LAST_PHOTO_X_SWITCH_DATA       : STRING_28;   (*FAS    *)
     TEXT_TO_FAS                    : STRING_28;   (*    STD*)
     NEW_GMT                        : STRING_9;    (*    STD*)
     NUMBER_OF_PHOTOS_TO_TAKE       : BYTE;        (*    STD*)
     INTERVAL_BETWEEN_PHOTOS        : INTEGER;     (*    STD*)
     FAS_STATUS_CODE                : BYTE;        (*FAS    *)
     INTERVAL_PRE_WARNING_FLAG      : BOOLEAN;     (*FAS    *)
     PHOTO_CONFIRMATION_FLAG        : BOOLEAN;     (*FAS    *)
     NUMBER_OF_PHOTOS_TAKEN         : BYTE;        (*FAS    *)
     NEW_FRAME_COUNT                : BYTE;        (*    STD*)
     FAS_HARDWARE_TEST              : CHAR;        (*    STD*)
     FAS_HARDWARE_TEST_FLAG         : BOOLEAN;     (*FAS    *)
     REPORTED_ANNOTATION_MODE       : CHAR;        (*FAS    *)
     REPORTED_TIME_TO_NEXT_PHOTO    : INTEGER;     (*FAS    *)

(**********FASTEST TURBO PASCAL DECLARATIONS***************)
     KEY           : CHAR;         (* ONE KEYBOARD CHAR       *)
     MENUKEY       : CHAR;         (* MENU SELECTION          *)
     DYNAMIC       : BOOLEAN;      (* CONTINOUS STATUS CHECK *)
```

```
(***************CLEAR AND PAINT SCREEN***************)
PROCEDURE PAINTSCREEN;
BEGIN
     CLRSCR;
     WRITELN('  MILETUS ASSOCIATES, INC.     ',
             ' F A S   T O   S T D     TEST PROGRAM');
     WRITELN('A  4.1   FAS CPU CAMERA TRIGGER ENABLED =');
     WRITELN('B  4.2   STD CPU CAMERA TRIGGER COMMAND');
     WRITELN('C  4.3   FAS CPU RESET COMMAND');
     WRITELN('D  4.4   READ HARDWARE FRAME COUNTER =');
     WRITELN('E  4.5   RESET HARDWARE FRAME COUNTER        ',
             'W  4.22  LOAD FRAME COUNT =' );
     WRITELN('F  4.6   SET FAS ANNOTATION MODE =');
     WRITELN('G  4.7   SEND LAST PHOTO "X" SWITCH     =');
     WRITELN('H  4.8   SEND TEXT TO FAS               =');
     WRITELN('I  4.9   SET GMT =');
     WRITELN('J  4.10  START CLOCK');
     WRITELN('K  4.11  SET INTERVALOMETER PARAMETERS.',
             ' 000 PHOTOS  AT 00000 INTERVALS');
     WRITELN('L  4.12  START PHOTO SEQUENCE');
     WRITELN('M  4.13  FAS STATUS CODE TO STD:',
             ' MODE =   , RESET =   ,  CARD =   , CAMERA =');
     WRITELN('N  4.14  CANCEL CURRENT PHOTO SEQUENCE');
     WRITELN('O  4.15  INTERVAL PRE-WARNING FLAG =');
     WRITELN('P  4.16  PHOTO CONFIRMATION TO STD =');
     WRITELN('Q  4.17  PHOTO COUNT THIS SEQUENCE =');
     WRITELN('R  4.18  PERFORM FAS COMMUNICATION PORT',
             ' TEST');
     WRITELN('S  4.19  REQUEST FAS HARDWARE SELF TEST =');
     WRITELN('T  4.20  REPORT ANNOTATION MODE TO STD =');
     WRITELN('U  4.21  REPORT TIME REMAINING TILL NEXT',
             ' SCHEDULED PHOTO =');
     WRITELN('    LAST FAS ROUTINE CALLED =',
             '        FAS_ERROR_CODE =');
     WRITELN('ENTER YOUR LETTER SELECTION =',
             '        (Z TO EXIT, Y STATIC , V TO CALIB.)');
     END;
```

```
(***********INITIALIZE THE VARIABLES*********)
PROCEDURE INITVAR;
BEGIN

     DYNAMIC                        := FALSE;

     MENUKEY                        := '*';

          (* NEW_FAS_ANNOTATION_MODE is set to the power *)
          (* up value.  All other variables under the    *)
          (* control of the STD, simulated by this menu  *)
          (* program, are set to what the operator enters*)
          (* in responsed to the menu item selected.     *)

     NEW_FAS_ANNOTATION_MODE        := 'B';

      (* 13 MARCH 89: IN ADDITION, FAS_TRIGGER_ENABLE*)
      (* MUST BE SET TO THE POWER UP ENABLED VALUE   *)
      (* IN ORDER FOR THE FAS TEST MENU TO SHOW TRUE *)

     FAS_TRIGGER_ENABLE             := TRUE;

END;
```

```
(******************CHANGE FAS TRIGGER************)

PROCEDURE CHFTRIG;

BEGIN

     GOTOXY(43,2);

     WRITE('ENTER:  T or F  = ');

     REPEAT UNTIL KEYPRESSED;

     KEY   := READKEY;

     FAS_TRIGGER_ENABLE := ( KEY = 'T' );

(* CALL FAS PROCEDURE HERE*)
   Set_FAS_Camera_Trigger_Control(FAS_ERROR_CODE,
                                  FAS_TRIGGER_ENABLE);

END;
```

```
(***************STD CPU CAMERA TRIGGER COMMAND*************)

PROCEDURE CHSTRIG;

BEGIN

(* CALL FAS PROCEDURE HERE*)
   STD_CPU_Trigger_Camera_Command(FAS_ERROR_CODE);

END;
```

```
(****************FAS CPU RESET COMMAND*********************)

PROCEDURE CHFRESET;

BEGIN

(* CALL FAS PROCEDURE HERE*)
   FAS_CPU_Reset_Command(FAS_ERROR_CODE);


(* DOES THIS RESET THE ANNOTATION MODE TO B? and          *)
(* ENABLE THE FAS CAMERA CONTROL?  YES, IT does. So:      *)

   NEW_FAS_ANNOTATION_MODE  := 'B';

END;
```

```
(****************READ HARDWARE FRAME COUNTER***************)

PROCEDURE CHFFCNT;

BEGIN

(* CALL FAS PROCEDURE HERE*)
   Read_Hardware_Frame_Counter(FAS_ERROR_CODE,
                               FAS_HARDWARE_FRAME_COUNTER);

     GOTOXY(40,5);

     CLREOL;

     WRITE(FAS_HARDWARE_FRAME_COUNTER);

END;
```

```
(*****************RESET HARDWARE FRAME COUNTER*************)

PROCEDURE CHFFRESET;

BEGIN

(* CALL FAS PROCEDURE HERE*)
   Reset_Hardware_Frame_Counter(FAS_ERROR_CODE);

END;
```

```
(*****************SET FAS ANNOTATION MODE******************)

PROCEDURE CHFMODE;

BEGIN

     GOTOXY(36,7);

     WRITE('ENTER: A, B, or C');

     REPEAT UNTIL KEYPRESSED;

     KEY   := READKEY;

     GOTOXY(36,7);

     WRITE(KEY,'                  ');

     NEW_FAS_ANNOTATION_MODE := KEY;

(* CALL FAS PROCEDURE HERE*)
   Set_FAS_Annotation_Mode(FAS_ERROR_CODE,
                           NEW_FAS_ANNOTATION_MODE);

END;
```

```
(*****************SEND LAST PHOTO "X" SWITCH***************)

PROCEDURE CHFX;

BEGIN

(* CALL FAS PROCEDURE HERE*)
  Get_Last_Photo_Data(FAS_ERROR_CODE,
                       LAST_PHOTO_X_SWITCH_DATA);


   GOTOXY(43,8);

   CLREOL;

   WRITE(LAST_PHOTO_X_SWITCH_DATA);

END;
```

```
(*****************SEND TEXT TO STD BUSS CPU***********)

PROCEDURE CHFTEXT;
VAR
     CHAR_POSITION   : BYTE;

BEGIN
     GOTOXY(43,9);
     CLREOL;
     CASE NEW_FAS_ANNOTATION_MODE OF
          'A'      : TEXT_TO_FAS :=
                        '1234567890123456789012345678';
          'B'      : TEXT_TO_FAS := '1234567890123456';
          'C'      : TEXT_TO_FAS := '1234567890123456789';
     END;
     WRITE(TEXT_TO_FAS);
     GOTOXY(43,9);
     CHAR_POSITION   := 0;
     REPEAT
            BEGIN
                 REPEAT UNTIL KEYPRESSED;
                 KEY   := READKEY;
                 CHAR_POSITION := CHAR_POSITION + 1;
                 CASE KEY OF
        (* CR *)      ^M    :  CHAR_POSITION := 30;
        (* BS *)      ^H    :  CHAR_POSITION :=
                                    CHAR_POSITION - 2;
                   ' '..'z':  TEXT_TO_FAS[CHAR_POSITION] :=
                                                  KEY;
                   END;
                   IF CHAR_POSITION >= 0
                      THEN
                          WRITE(KEY)
                      ELSE
                          CHAR_POSITION := 0;
            END;
       UNTIL  CHAR_POSITION >= LENGTH(TEXT_TO_FAS);

(* CALL FAS PROCEDURE HERE*)
   Send_Text_To_FAS(FAS_ERROR_CODE, TEXT_TO_FAS);

END;
```

```
(* Miletus Associates, Inc.        FAS Testing PROCEDURES *)

(*****************SET GMT TIME******************************)

PROCEDURE CHFGMT;

BEGIN

     GOTOXY(20,10);

     CLREOL;

     WRITE('DHHMMSSSS');

     GOTOXY(20,10);

     READLN(NEW_GMT);

(* CALL FAS PROCEDURE HERE*)
   Set_GMT(FAS_ERROR_CODE, NEW_GMT);

END;
```

```
(****************START CLOCK*****************************)

PROCEDURE CHFGOCLK;

BEGIN

(* CALL FAS PROCEDURE HERE*)
   Start_FAS_Clock(FAS_ERROR_CODE);

END;
```

```
(*****************SET INTERVALOMETER PARAMETERS************)

PROCEDURE CHFINTER;

VAR
     NUM_TEMP        :INTEGER;

BEGIN

     NUM_TEMP := 300;  (* SOME NUMBER > 255, A MAX BYTE VALUE*)

     WHILE ((NUM_TEMP < 0)
           OR
            (NUM_TEMP >255)) DO

     BEGIN

          GOTOXY(41,12);

          WRITE('   ');

          GOTOXY(41,12);

          READLN(NUM_TEMP);

     END;

     NUMBER_OF_PHOTOS_TO_TAKE  := BYTE(NUM_TEMP);

     GOTOXY(56,12);

     WRITE('     ');

     GOTOXY(56,12);

     READLN(INTERVAL_BETWEEN_PHOTOS);

(* CALL FAS PROCEDURE HERE*)
   Set_Intervalometer(FAS_ERROR_CODE,
                      NUMBER_OF_PHOTOS_TO_TAKE,
                      INTERVAL_BETWEEN_PHOTOS);

END;
```

```
(*****************START PHOTO SEQUENCE*********************)

PROCEDURE CHFGOPHOTO;

BEGIN

(* CALL FAS PROCEDURE HERE*)
   Start_Photo_Sequence(FAS_ERROR_CODE);

END;
```

```
(*****************FAS STATUS CODE TO STD******************)

PROCEDURE CHFSTATUS;

VAR
     REAL_STATUS    : BYTE;
     TEMP           : BYTE;   (* USED IN MATH *)

BEGIN

(* CALL FAS PROCEDURE HERE*)
   Get_FAS_Status(FAS_ERROR_CODE, FAS_STATUS_CODE);

     TEMP           := FAS_STATUS_CODE;
     REAL_STATUS    := TEMP DIV 64;
     GOTOXY(34,14);
     CLREOL;
     WRITE('MODE = ');
       CASE REAL_STATUS OF
          0  :  WRITE('A, ');
          1  :  WRITE('B, ');
          2  :  WRITE('C, ');
          3  :  WRITE('?, ');
       END;
     TEMP           := TEMP - (REAL_STATUS * 64);
     REAL_STATUS    := TEMP DIV 32;
     TEMP           := TEMP - (REAL_STATUS * 32);
     WRITE('RESET = ',REAL_STATUS,', ');
     REAL_STATUS    := TEMP DIV 4;
     WRITE('CARD = ',REAL_STATUS,', ');
     TEMP           := TEMP - (REAL_STATUS * 4);
     WRITE('CAMERA = ',TEMP);

END;
```

```
(* Miletus Associates, Inc.          FAS Testing PROCEDURES *)

(*****************CANCEL CURRENT PHOTO SEQUENCE************)

PROCEDURE CHFSTOPPHOTO;

BEGIN

(* CALL FAS PROCEDURE HERE*)
   Cancel_Photo_Sequence(FAS_ERROR_CODE);

END;
```

```
(*****************INTERVAL PRE-WARNING FLAG*****************)

PROCEDURE CHFPREPHOTO;

BEGIN

(* CALL FAS PROCEDURE HERE*)
   Get_Photo_Pre_Warning_Flag(FAS_ERROR_CODE,
                              INTERVAL_PRE_WARNING_FLAG);

   GOTOXY(38,16);

   CLREOL;

   IF NOT INTERVAL_PRE_WARNING_FLAG THEN WRITE('NO ');

   WRITE('PHOTO PENDING   ');

END;
```

```
(* Miletus Associates, Inc.          FAS Testing PROCEDURES *)

(****************PHOTO CONFIRMATION TO STD****************)

PROCEDURE CHFCONPHOTO;

BEGIN

(* CALL FAS PROCEDURE HERE*)
   Get_Photo_Confirmation_Flag(FAS_ERROR_CODE,
                               PHOTO_CONFIRMATION_FLAG);

   GOTOXY(38,17);

   CLREOL;

   IF PHOTO_CONFIRMATION_FLAG THEN

             WRITE('SUCCESSFUL PHOTO TAKEN')

             ELSE

             WRITE('NO PHOTO TAKEN        ');

END;
```

```
(*****************PHOTO COUNT THIS SEQUENCE****************)

PROCEDURE CHFPHOTOCNT;

BEGIN

(* CALL FAS PROCEDURE*)
   Get_Photo_Count(FAS_ERROR_CODE, NUMBER_OF_PHOTOS_TAKEN);

   GOTOXY(38,18);

   WRITE('        ');

   GOTOXY(38,18);

   WRITE(NUMBER_OF_PHOTOS_TAKEN);

END;
```

```
(* Miletus Associates, Inc.        FAS Testing PROCEDURES *)

(*****************PERFORM FAS COMM PORT TEST***************)

PROCEDURE CHFPORTTEST;

BEGIN

(* CALL FAS PROCEDURE HERE*)
   Perform_FAS_Communication_Test(FAS_ERROR_CODE);

END;
```

```
(*****************REQUEST FAS HARDWARE SELF TEST************)

PROCEDURE CHFHARDTEST;

BEGIN
   GOTOXY(43,20);
   CLREOL;
   WRITE('ENTER A, E, R, OR C =');
   REPEAT UNTIL KEYPRESSED;
   KEY   := READKEY;
   GOTOXY(43,20);
   CLREOL;
   WRITE(KEY,' has ');
   FAS_HARDWARE_TEST  := KEY;

(* CALL FAS PROCEDURE HERE*)
   Perform_FAS_Hardware_Test(FAS_ERROR_CODE,
                             FAS_HARDWARE_TEST,
                             FAS_HARDWARE_TEST_FLAG);

  IF FAS_HARDWARE_TEST_FLAG
    THEN
        WRITE(' F I N I S H E D')
    ELSE
        WRITE(' E R R O R E D  O U T');
END;
```

```
(*****************REPORT ANNOTATION MODE TO STD*************)

PROCEDURE CHFRPTMODE;

BEGIN

(* CALL FAS PROCEDURE HERE*)
   Report_Annotation_Mode(FAS_ERROR_CODE,
                                REPORTED_ANNOTATION_MODE);

   GOTOXY(42,21);

   CLREOL;

   WRITE(REPORTED_ANNOTATION_MODE);

END;
```

```
(*****************REPORT TIME LEFT TO NEXT PHOTO***********)

PROCEDURE CHFNXPHOTO;

BEGIN

(* CALL FAS PROCEDURE HERE*)
   Report_Time_To_Next_Photo(FAS_ERROR_CODE,
                             REPORTED_TIME_TO_NEXT_PHOTO);

   GOTOXY(60,22);

   CLREOL;

   WRITE(REPORTED_TIME_TO_NEXT_PHOTO);

END;
```

C-2

```
(****************BEGIN CALIBRATION OF FAS CLOCK***********)

(*  WARNING: THE CALIBRATION PROCEDURE BELOW PUTS THE FAS *)
(*  INTO THE MODE NECESSARY TO ADJUST THE CAPACITOR WHICH *)
(*  REGULATES THE CRYSTAL'S FREQUENCY.  THE FAS CAN DO NO *)
(*  OTHER FUNCTION ONCE SET IN THIS MODE; ONLY A FAS CPU  *)
(*  RESET (4.3) WILL RETURN THE FAS TO PROPER OPERATION.  *)
(*       After a 3 minute timeout or key press, the       *)
(*  PROCEDURE FAS_Clock_Calibration will do a RESET, 4.1. *)
(*  THEN THIS PROCEDURE WILL INITVAR AND DO A FAS POWER UP*)
PROCEDURE CHFCAL;

VAR
     CALIBRATE_CODE     : BYTE;
     DELAY_COUNTER      : INTEGER;
BEGIN
     CLRSCR;
     GOTOXY(20,8);
     WRITE('C L O C K   C A L I B R A T I O N   M O D E');
     GOTOXY(17,12);
     WRITELN('The FAS will be reset, so all variables are lost!');
     GOTOXY(20,14);
     WRITELN('For the next 3 minutes, the FAS card can be');
     GOTOXY(18,16);
     WRITELN('calibrated.  Press any key to terminate earlier.');

(* CALL FAS PROCEDURE HERE *)
   FAS_Clock_Calibration(FAS_ERROR_CODE);

      PAINTSCREEN;
      INITVAR;
                       (***FAS POWER UP***)
(* CALL FAS PROCEDURE HERE*)
   FAS_POWER_UP(FAS_ERROR_CODE);

END;
```

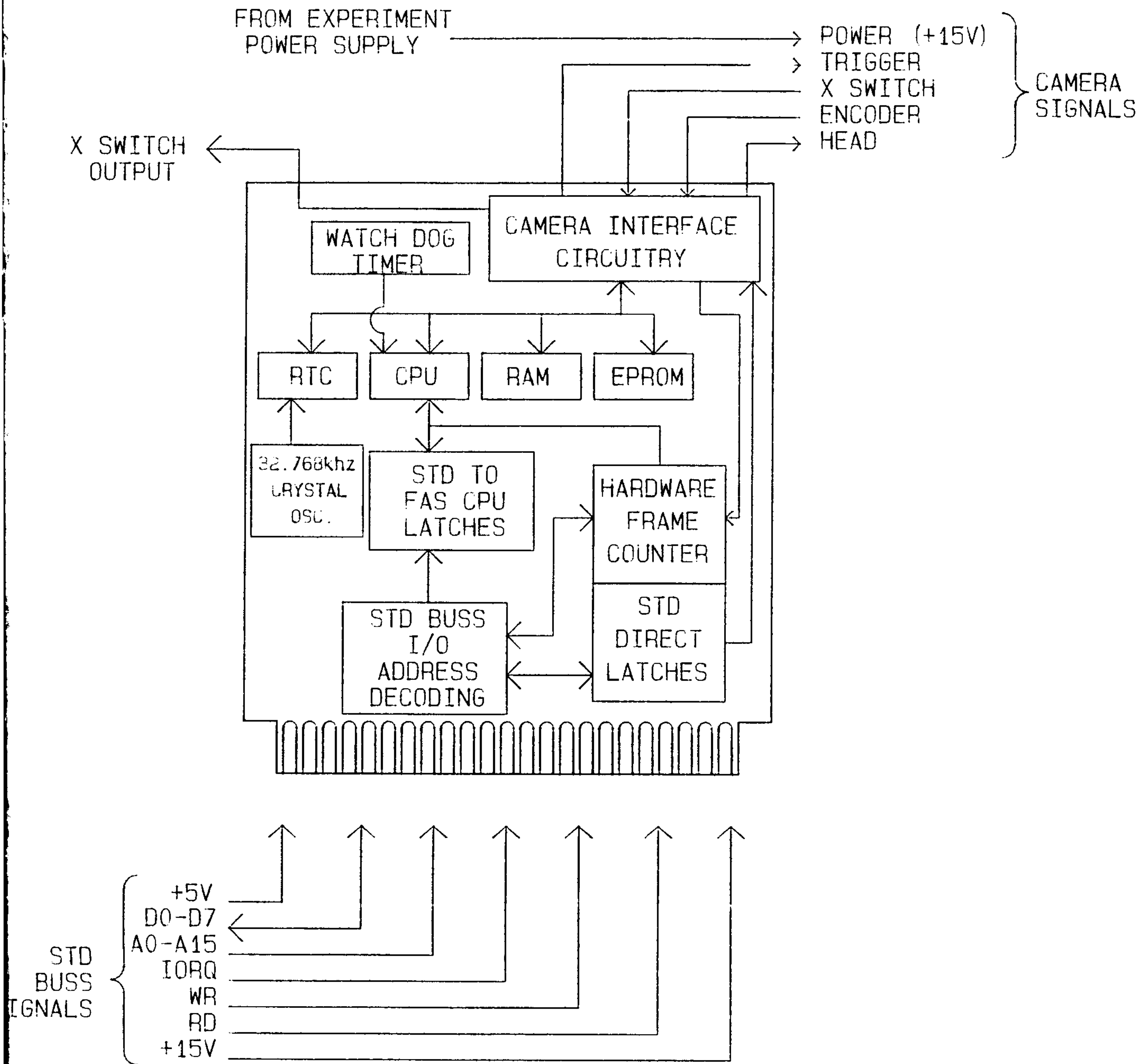
```
(****************CHANGE THE FRAME COUNT******************)

PROCEDURE CHFFRAMECNT;

VAR
     NUM_TEMP        :INTEGER;

BEGIN

     NUM_TEMP := 300;  (* SOME NUMBER > 255, A MAX BYTE VALUE*)

     WHILE ((NUM_TEMP < 0)
           OR
            (NUM_TEMP >255)) DO

        BEGIN

          GOTOXY(73,6);

          CLREOL;

          READLN(NUM_TEMP);

        END;

NEW_FRAME_COUNT  := BYTE(NUM_TEMP);

(* CALL FAS PROCEDURE HERE*)
   Load_the_FAS_software_frame_count(FAS_ERROR_CODE,
                                     NEW_FRAME_COUNT);

END;
```

```
(*****************TOGGLE DYNAMIC/STATIC MENU FLAG**********)

PROCEDURE DYNAMIC_STATIC_TOGGLE;

BEGIN

     DYNAMIC := NOT DYNAMIC;

     GOTOXY(50,24);

     IF DYNAMIC THEN WRITE('DYNAMIC')

                ELSE WRITE('STATIC ');

END;
```

```
(*****************UPDATED SCREEN****************************)

PROCEDURE UPDATE_SCREEN;
VAR
           TEMP   : BYTE;
BEGIN
           GOTOXY(43,2);
           CLREOL;
           WRITE(FAS_TRIGGER_ENABLE,'    ');
           GOTOXY(53,23);
           CLREOL;
           Get_Error_Code_Number(FAS_ERROR_CODE, TEMP);
           WRITE(TEMP);
           Get_Error_Code_Caller(FAS_ERROR_CODE, TEMP);
           GOTOXY(31,23);
           WRITE('4.',TEMP,' ');
END;
```

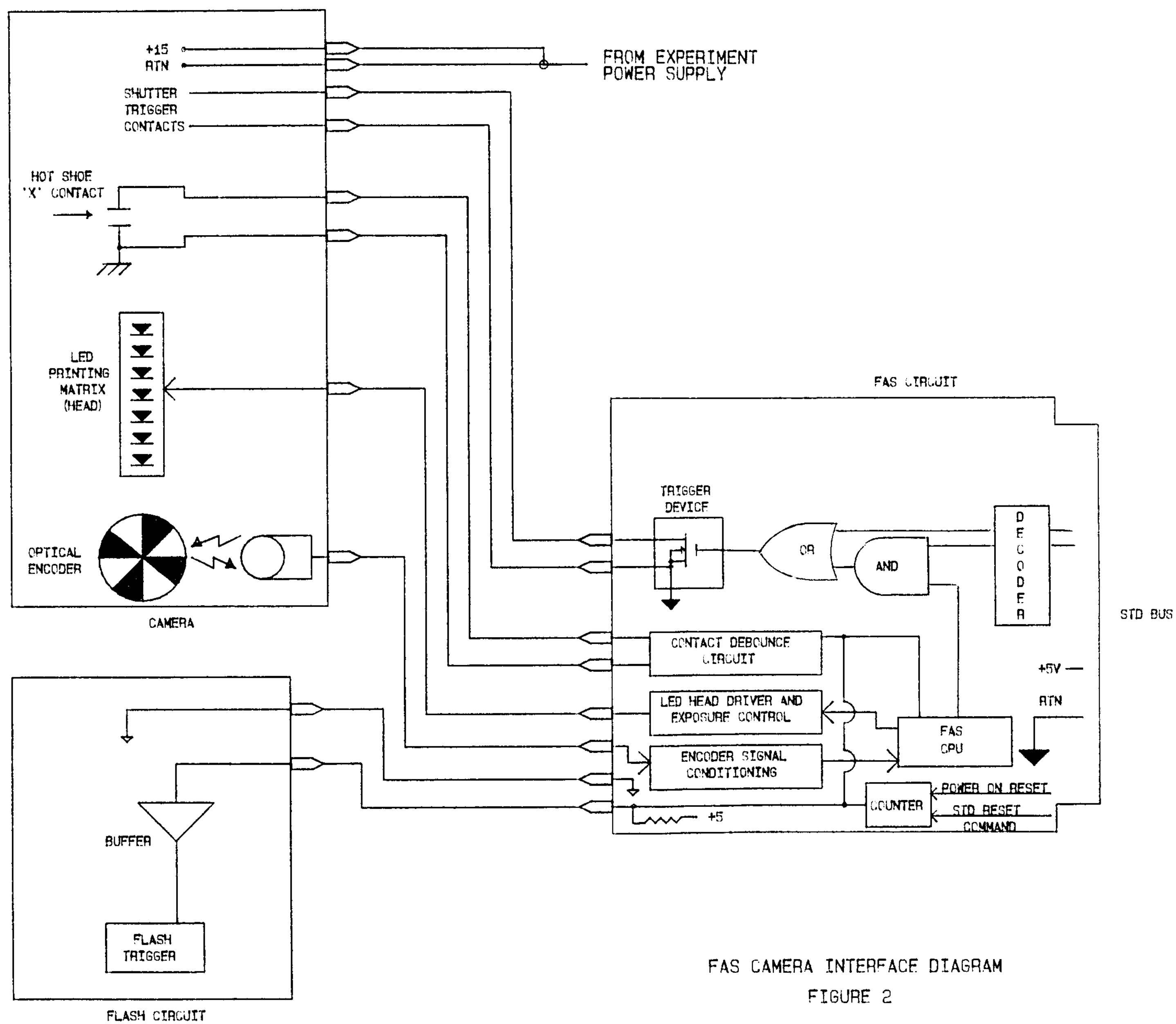
```
(*************TURBO PASCAL MAIN PROGRAM*****************)

                     (***INITIALIZE***)
BEGIN
     TEXTBACKGROUND(BLUE);
     TEXTCOLOR(WHITE);
     PAINTSCREEN;
     INITVAR;
                     (***FAS POWER UP***)
(* CALL FAS PROCEDURE HERE*)
   FAS_POWER_UP(FAS_ERROR_CODE);

                      (***MENU LOOP***)
  WHILE MENUKEY<>'Z' DO
       BEGIN
           UPDATE_SCREEN;
           GOTOXY(31,24);
           WRITE('  ');
           GOTOXY(31,24);
           IF DYNAMIC THEN
                           DELAY(500)
                ELSE
                           REPEAT UNTIL KEYPRESSED;
           IF KEYPRESSED THEN
                              MENUKEY  := READKEY
                ELSE
                              MENUKEY := '*';
           WRITE(MENUKEY);
           CASE MENUKEY OF
               'A'   : CHFTRIG;
               'B'   : CHSTRIG;
               'C'   : CHFRESET;
               'D'   : CHFFCNT;
               'E'   : CHFFRESET;
               'F'   : CHFMODE;
               'G'   : CHFX;
               'H'   : CHFTEXT;
               'I'   : CHFGMT;
               'J'   : CHFGOCLK;
               'K'   : CHFINTER;
               'L'   : CHFGOPHOTO;
               'M'   : CHFSTATUS;
               'N'   : CHFSTOPPHOTO;
               'O'   : CHFPREPHOTO;
               'P'   : CHFCONPHOTO;
               'Q'   : CHFPHOTOCNT;
               'R'   : CHFPORTTEST;
               'S'   : CHFHARDTEST;
               'T'   : CHFRPTMODE;
               'U'   : CHFNXPHOTO;
               'V'   : CHFCAL;
               'W'   : CHFFRAMECNT;
```

```
          'Y'   : DYNAMIC_STATIC_TOGGLE;
      END;
```



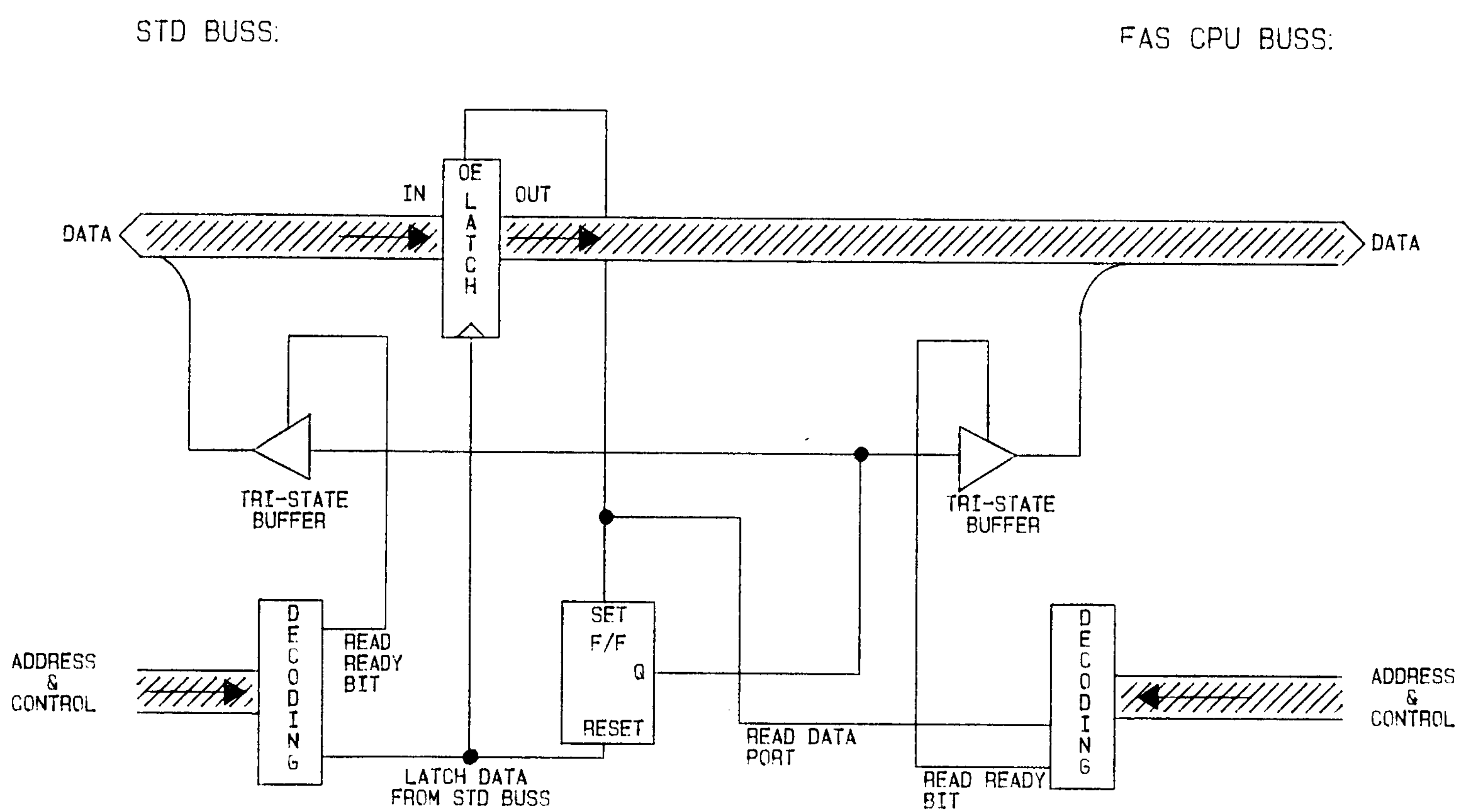
FAS CARD BLOCK DIAGRAM

FIGURE 1

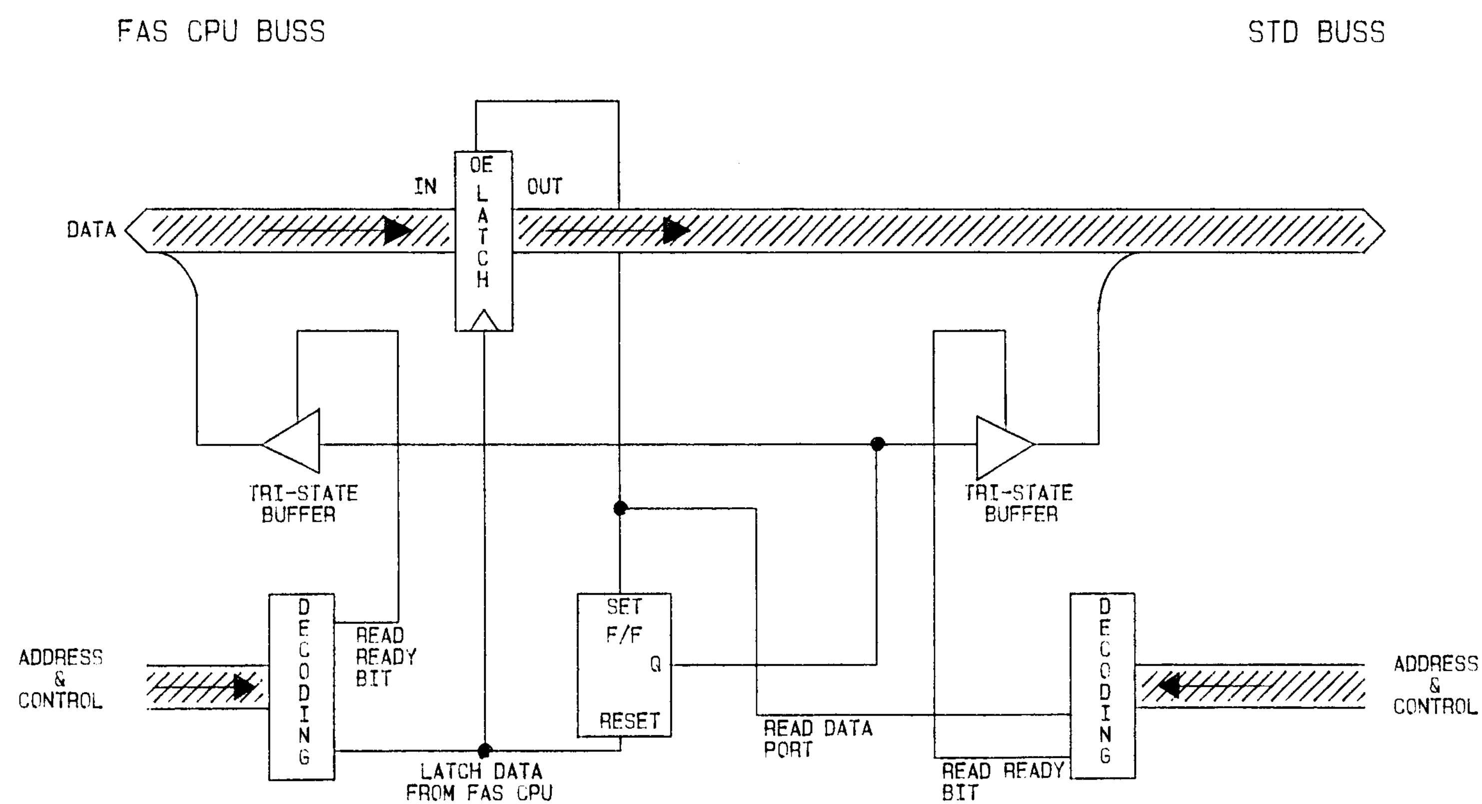


FAS CAMERA INTERFACE DIAGRAM

FIGURE 2



DATA AND READY PORTS BLOCK DIAGRAM
STD BUSS TO FAS CPU
FIGURE 3



DATA AND READY PORTS BLOCK DIAGRAM
FAS CPU TO STD BUSS

FIGURE 4

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

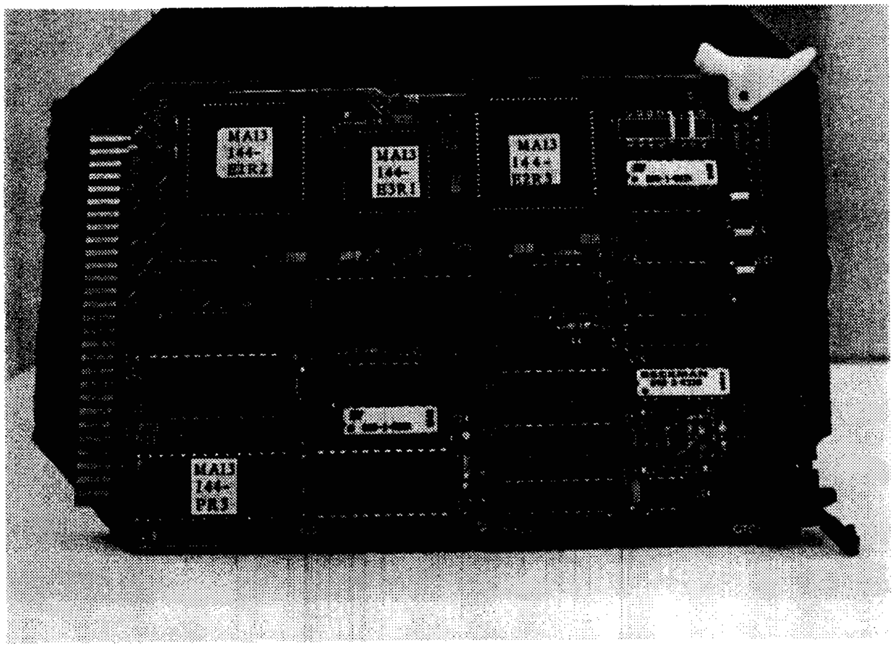


Figure 5. Component Side of P.C. Board

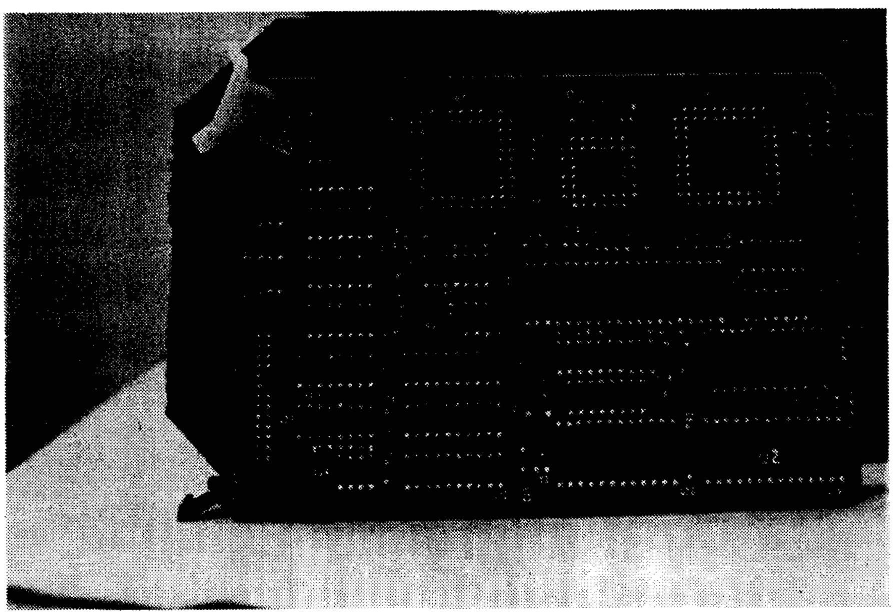


Figure 6. Solder Side of P.C. Board

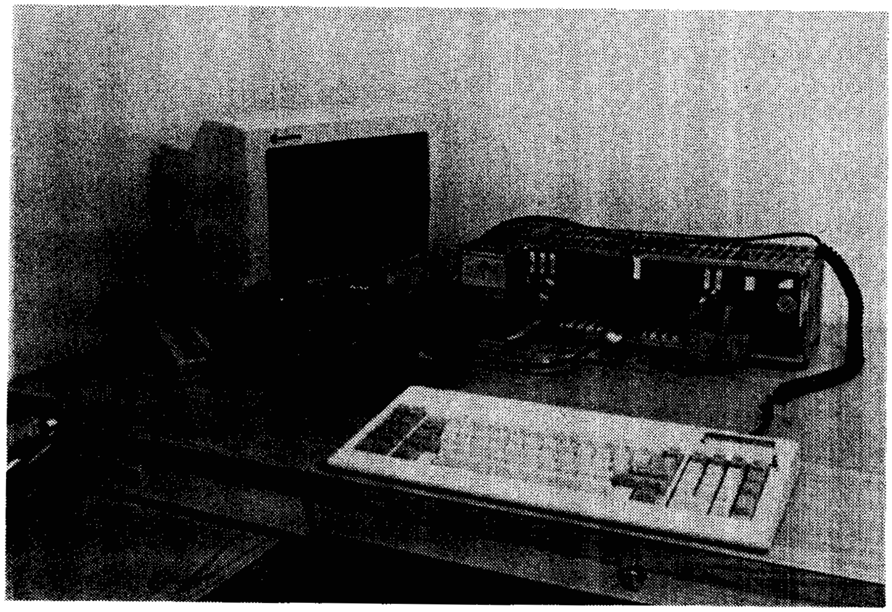


Figure 7. Test Set Up

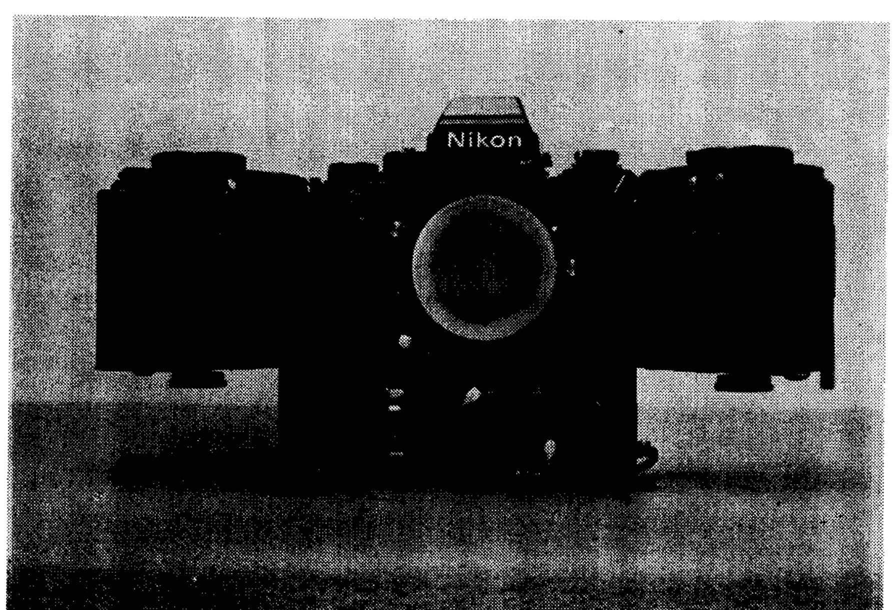


Figure 8. Nikon F3 Camera (Front View)

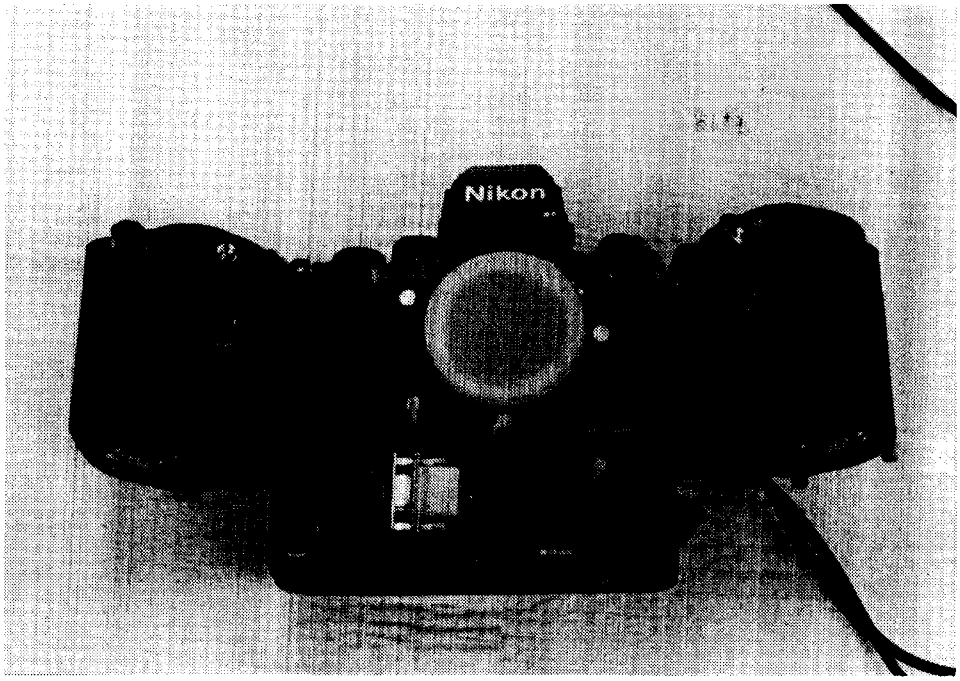


Figure 9. Nikon F3 Front Cable Attachment

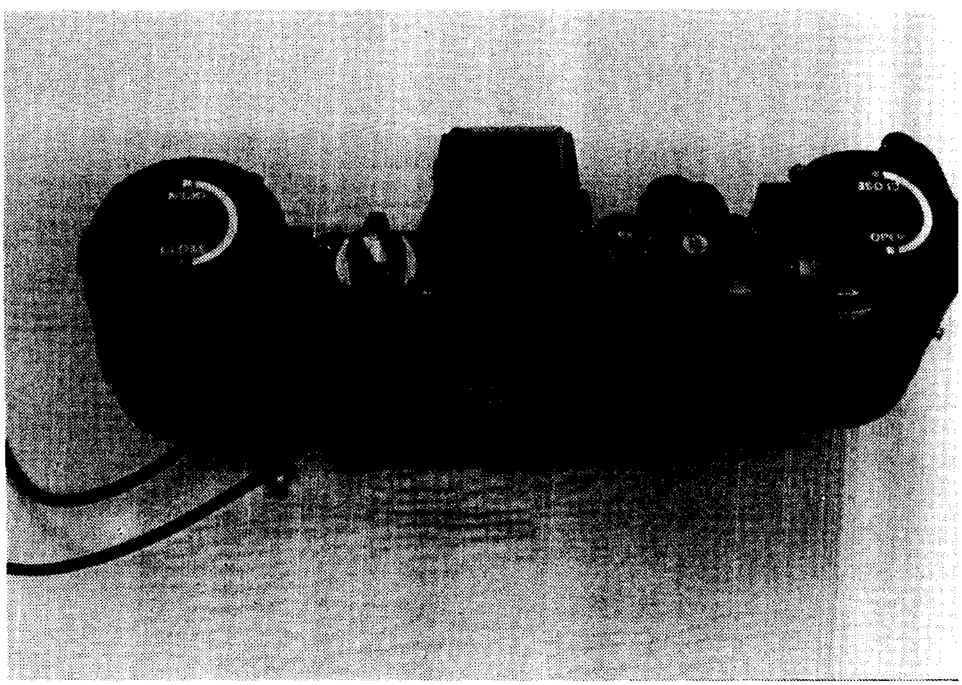


Figure 10. Nikon F3 Rear Cable Attachment

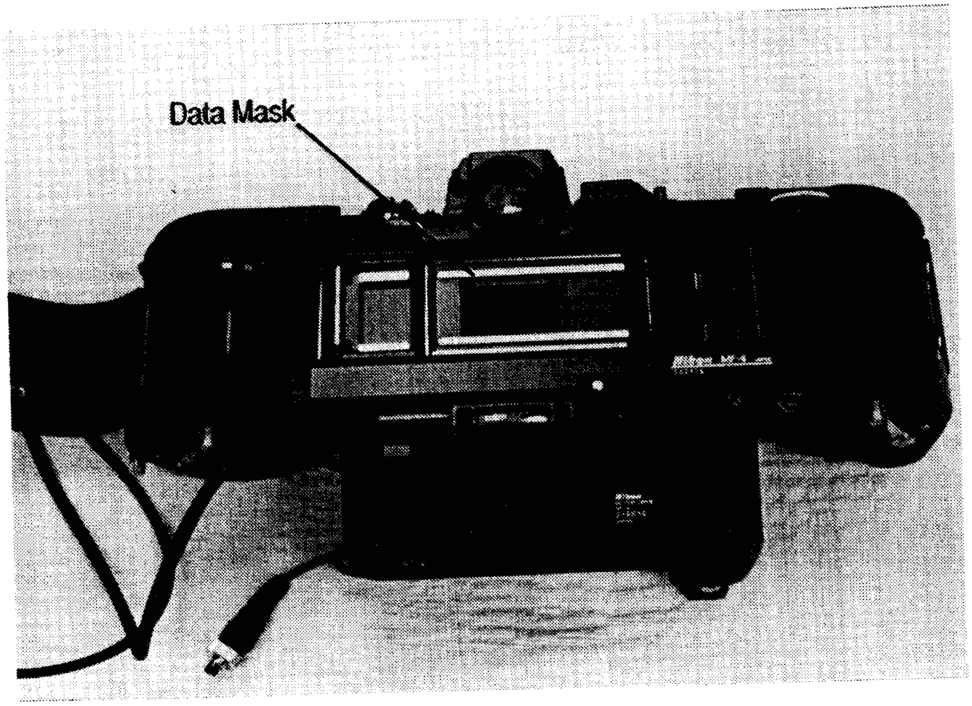


Figure 11. Nikon F3 Data Mask

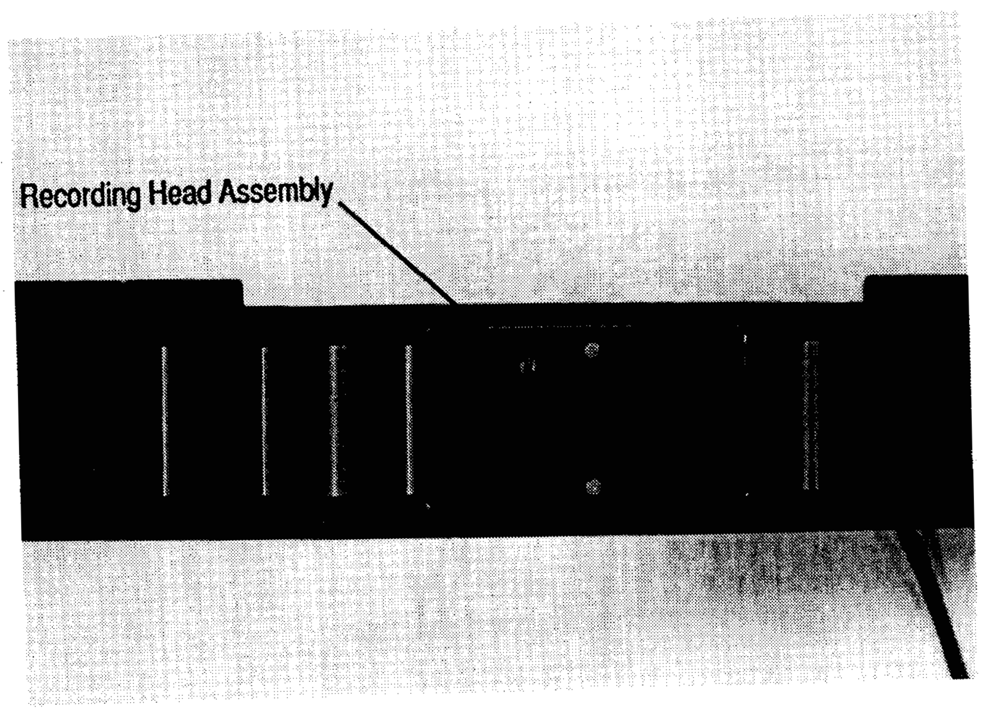
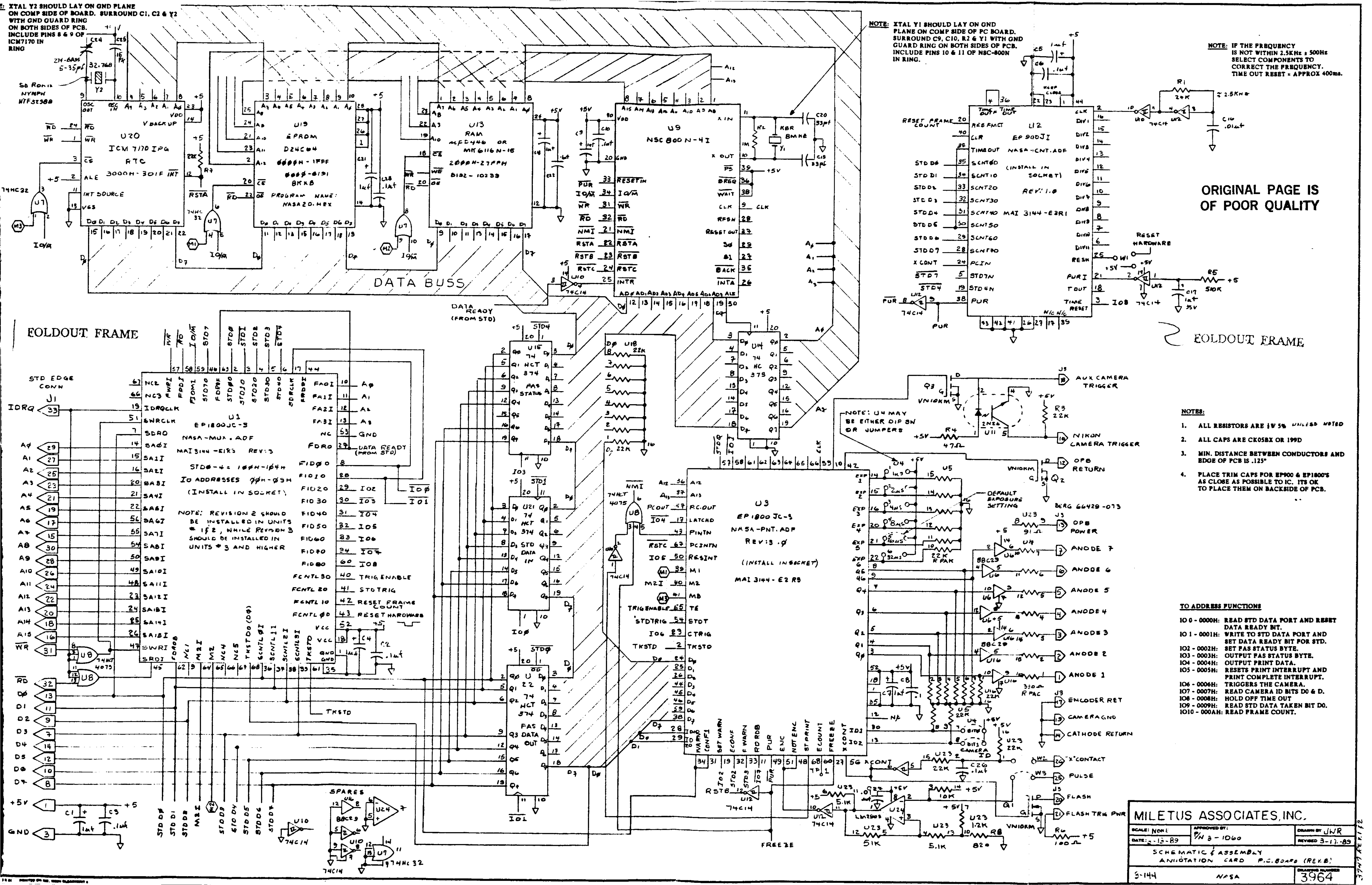


Figure 12. RHA

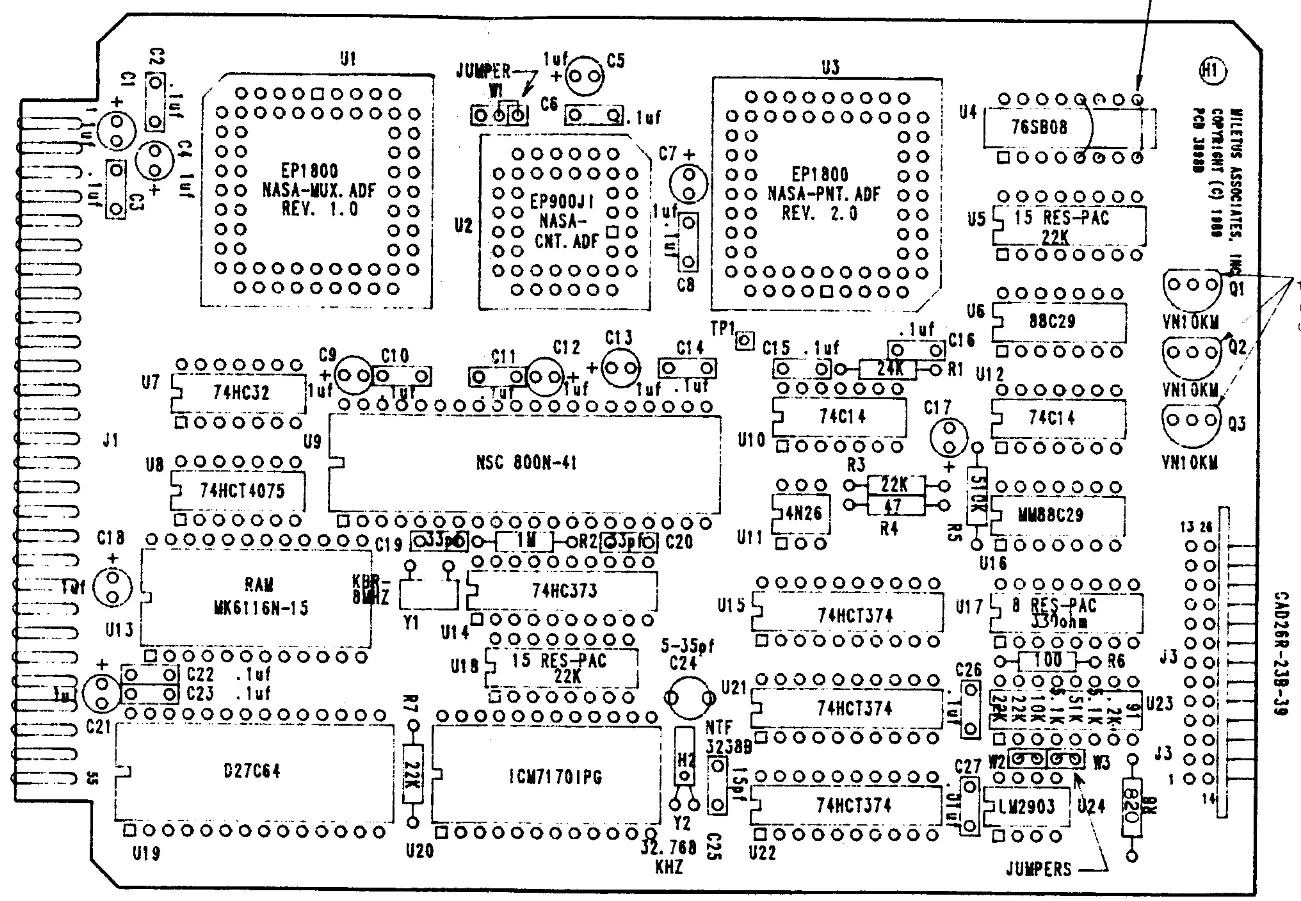


NOTE: XTAL Y2 SHOULD LAY ON GND PLANE ON COMP SIDE OF BOARD. SURROUND C1, C2 & Y2 WITH GND GUARD RING ON BOTH SIDES OF PCB. INCLUDE PINS 8 & 9 OF ICM7170 IN RING
NOTE: XTAL Y1 SHOULD LAY ON GND PLANE ON COMP SIDE OF PC BOARD. SURROUND C9, C10, R2 & Y1 WITH GND GUARD RING ON BOTH SIDES OF PCB. INCLUDE PINS 10 & 11 OF NSC-800N IN RING.
NOTE: IF THE FREQUENCY IS NOT WITHIN 2.5KHz ± 500Hz SELECT COMPONENTS TO CORRECT THE FREQUENCY. TIME OUT RESET = APPROX 400ms.
ORIGINAL PAGE IS OF POOR QUALITY
FOLDOUT FRAME
DATA BUSS
NOTES:
1. ALL RESISTORS ARE ¼W 5% UNLESS NOTED
2. ALL CAPS ARE CK05BX OR 199D
3. MIN. DISTANCE BETWEEN CONDUCTORS AND EDGE OF PCB IS .125"
4. PLACE TRIM CAPS FOR EP900 & EP1800'S AS CLOSE AS POSSIBLE TO IC. ITS OK TO PLACE THEM ON BACKSIDE OF PCB.
TO ADDRESS FUNCTIONS
IO 0 - 0000H: READ STD DATA PORT AND RESET DATA READY BIT.
IO 1 - 0001H: WRITE TO STD DATA PORT AND SET DATA READY BIT FOR STD.
IO2 - 0002H: SET FAS STATUS BYTE.
IO3 - 0003H: OUTPUT FAS STATUS BYTE.
IO4 - 0004H: OUTPUT PRINT DATA.
IO5 - 0005H: RESETS PRINT INTERRUPT AND PRINT COMPLETE INTERRUPT.
IO6 - 0006H: TRIGGERS THE CAMERA.
IO7 - 0007H: READ CAMERA ID BITS D0 & D.
IO8 - 0008H: HOLD OFF TIME OUT
IO9 - 0009H: READ STD DATA TAKEN BIT D0.
IO10 - 000AH: READ FRAME COUNT.
FOLDOUT FRAME
MILETUS ASSOCIATES, INC.
SCALE: NONE
DATE: 2-13-89
APPROVED BY: P/N 3-1060
DRAWN BY: JWR
REVISED 3-17-89
SCHEMATIC & ASSEMBLY ANNOTATION CARD P.C. BOARD (REV. B)
3-144
NASA
DRAWING NUMBER 3964

NOTES:

1. Use 800° solder tip.
2. All leads must be cut to not less than .015 and not more than .025. Use 26 awg wire for spacing.
3. All components are to be installed with leads cut to lenth then soldered, using tape to hold parts in place. No copper is to be exposed where lead has been cut.
4. Install W1, W2, W3 during assembly as per drawing.
5. Install sleeved buss wire #24 on U4 - 5 + 12, 7 + 10.

ORIGINAL PAGE IS OF POOR QUALITY



FAS ANNOTATION   MILETUS ASSOCIATES, INC.   TECH DESIGN, INC.
PCB 3898 REV - B   03/14/89   TD189064
LAYER 1 - COMPONENT SIDE   01/10/89

FOLDOUT FRAME

FOLDOUT FRAME

MILETUS ASSOCIATES, INC.	
SCALE: 2:1 DATE: 3-16-89	APPROVED BY: P/N 3-1060
3-144 NASA	DRAWING NUMBER 3964

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-185114		

4. Title and Subtitle	5. Report Date
Film Annotation System for a Space Experiment	July 1989
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
W.R. Browne & S.S. Johnson	
	10. Work Unit No.
9. Performing Organization Name and Address	11. Contract or Grant No.
Miletus Associates, Inc. 3876 Hawkins NE Albuquerque, NM 87109	NAS3-25055
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Flight Hardware Report
National Aeronautics and Space Administration Lewis Research Center 21000 Brookpark Road Cleveland, OH 44135	14. Sponsoring Agency Code

15. Supplementary Notes

NASA Project Manager, Mr. G.A. Kraft, NASA Lewis Research Center
Telephone Number (216) 433-3932

16. Abstract

This microprocessor system was designed to control and annotate a Nikon 35mm camera for the purpose of obtaining photographs and data at predefined time intervals. The single STD BUSS interface card has been designed in such a way as to allow it to be used in either a stand alone application with minimum features or installed in a STD BUSS computer allowing for maximum features.

This control system also allows the exposure of twenty eight alpha/numeric characters across the bottom of each photograph. The data contains such information as camera identification, frame count, user defined text, and time to .01 seconds.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Film Annotation System 35mm Camera Intervalometer	Unclassified - Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No of pages	22. Price*
Unclassified	Unclassified	121	

ASA FORM 1626 OCT 86

*For sale by the National Technical Information Service, Springfield, Virginia 22161