

AD-A241 694



2

ANNUAL REPORT

VOLUME 2

TASK 2: SEEKER SCENE EMULATOR DEVELOPMENT

REPORT NO. AR-0142-91-002

September 25, 1991

DIGITAL EMULATION TECHNOLOGY LABORATORY

Contract No. DASG60-89-C-0142

Sponsored By

The United States Army Strategic Defense Command

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

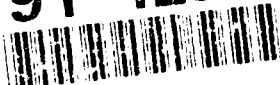
Atlanta, Georgia 30332 - 0540

Contract Data Requirements List Item A005

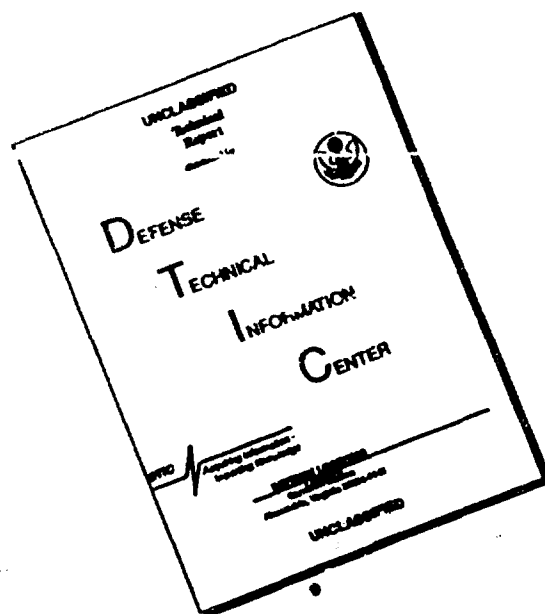
Period Covered: FY 91

Type Report: Annual

91-12577



DISCLAIMER NOTICE



THIS DOCUMENT IS BEST
QUALITY AVAILABLE. THE COPY
FURNISHED TO DTIC CONTAINED
A SIGNIFICANT NUMBER OF
PAGES WHICH DO NOT
REPRODUCE LEGIBLY.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No 0704-018A	
1a REPORT SECURITY CLASSIFICATION Unclassified			1b RESTRICTIVE MARKINGS		
2a SECURITY CLASSIFICATION AUTHORITY			3 DISTRIBUTION/AVAILABILITY OF REPORT 1) Approved for public release; distribution is unlimited 2) continued on reverse side		
2b DECLASSIFICATION/DOWNGRADING SCHEDULE			5 MONITORING ORGANIZATION REPORT NUMBER(S)		
4 PERFORMING ORGANIZATION REPORT NUMBER(S) AR-0142-91-002					
6a NAME OF PERFORMING ORGANIZATION School of Electrical Eng. Georgia Tech		6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION U.S. Army Strategic Defense Command		
6c ADDRESS (City, State, and ZIP Code) Atlanta, Georgia 30332			7b ADDRESS (City, State, and ZIP Code) P.O. Box 1500 Huntsville, AL 35807-3801		
8a NAME OF FUNDING/SPONSORING ORGANIZATION		8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DASG60-89-C-0142		
8c ADDRESS (City, State, and ZIP Code)			10 SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
					WORK UNIT ACCESSION NO.
11 TITLE (Include Security Classification) Guidance, Navigation and Control Digital Emulation Technology Laboratory Volume 2 (Unclassified)					
12 PERSONAL AUTHOR(S) C. O. Alford, Andrew Henshaw, Stephen Giesecking, Roy Melton					
13a TYPE OF REPORT Annual		13b TIME COVERED FROM 9/28/90 TO 9/27/91		14 DATE OF REPORT (Year, Month, Day) 9/27/91	
15 PAGE COUNT 209					
16 SUPPLEMENTARY NOTATION					
17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP			
19 ABSTRACT (Continue on reverse if necessary and identify by block number)					
1. Introduction 1.1 Objectives 1.2 Schedules & Milestones			4.2 Include Files 4.3 Makefiles		
2. Seeker Scene Emulator 2.1 Description 2.2 Hardware Components 2.3 Software components			5. Appendix B: Program Listing 5.1 PC Source Code 5.2 Transputer Implementation Source Code		
3. Related Development Efforts 3.1 Neural Net investigation 3.2 Advanced Signal Processing Testbed 3.3 LATS interface					
4. Appendix A. Seeker Scene Emulator Source Code 4.1 Occam Source Code					
20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RP1. ☐ DTIC USERS			21 ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a NAME OF RESPONSIBLE INDIVIDUAL			22b TELEPHONE (Include Area Code)		22c OFFICE SYMBOL

~~UNCLASSIFIED~~
~~Security Classification of this page~~

Distribution statement continued

- 2) This material may be reproduced by or for the U.S. Government pursuant to the copy license under the clause at DFARS252.227-7013, October 1988.

~~UNCLASSIFIED~~
~~Security Classification of this page~~

DISCLAIMER

DISCLAIMER STATEMENT - The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation.

DISTRIBUTION CONTROL

- (1) **DISTRIBUTION STATEMENT** - Approved for public release; distribution is unlimited.
- (2) This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227 - 7013, October 1988.



Accession For	
NTIS	DTIC
Unannounced	Justification
By	
Distribution/	
Availability Codes	
Dist	Avail and/or special
A-1	

ANNUAL REPORT

VOLUME 2

TASK 2: SEEKER SCENE EMULATOR DEVELOPMENT

September 26, 1991

Authors

**Andrew M. Henshaw,
Steven R. Giesecking, Roy W. Melton**

COMPUTER ENGINEERING RESEARCH LABORATORY

**Georgia Institute of Technology
Atlanta, Georgia 30332 - 0540**

Eugene L. Sanders

USASDC

Contract Monitor

Cecil O. Alford

Georgia Tech

Project Director

Copyright © 1991

Georgia Tech Research Corporation

Centennial Research Building

Atlanta, Georgia 30332

Table of Contents

1. Introduction	1
1.1. Objectives	1
1.2. Schedules and Milestones.....	2
2. Seeker Scene Emulator	5
2.1. Description.....	5
2.2. Hardware components	5
2.2.1. Seeker Scene Emulator	5
2.2.2. SSE/SP Interface	9
2.2.2.1. Description.....	9
2.2.2.2. Hardware.....	13
2.2.2.2.1. Processor	13
2.2.2.2.2. Memory Access Controller	13
2.2.2.2.3. Master Clock Generator.....	16
2.2.2.2.4. Horizontal and Vertical Clock Generators.....	18
2.2.2.2.5. Address Generator	21
2.2.2.3. Programming.....	23
2.2.3. Gamma-injection circuitry	27
2.3. Software components	27
2.3.1. Seeker Scene Emulator operation	27
2.3.2. Seeker Scene Emulator Data Generation.....	28
2.3.2.1. ESSING	28
2.3.2.2. Strategic Scene Generation Model	31
2.3.3. Scene generation for Signal Processing testing	31
3. Related Development Efforts	33
3.1. Neural net investigation.....	33
3.2. Advanced Signal Processing Testbed.....	33
3.3. LATS interface.....	33
4. Appendix A: Seeker Scene Emulator Source Code	A-1
4.1. Occam Source Code	A-1
4.1.1. Seeker.pgm	A-1
4.1.2. B409.occ	A-9
4.1.3. B409Stub.occ.....	A-16

4.1.4. Background.occ.....	A-17
4.1.5. Controller.occ.....	A-21
4.1.6. FirstBuffer.occ.....	A-29
4.1.7. Formatter.occ.....	A-31
4.1.8. G_Line.occ.....	A-33
4.1.9. G_System.occ.....	A-39
4.1.10. G_Text.occ.....	A-41
4.1.11. GIF.occ.....	A-46
4.1.12. GIF02.occ.....	A-52
4.1.13. GTSEI.occ.....	A-58
4.1.14. GTSPI.occ.....	A-60
4.1.15. GTSPI2.occ.....	A-64
4.1.16. Guidance.occ.....	A-67
4.1.17. HostSeeker.occ.....	A-70
4.1.18. HostStub.occ.....	A-73
4.1.19. ImageDisplay.occ.....	A-74
4.1.20. Loader.occ.....	A-78
4.1.21. RunSeekr.occ.....	A-92
4.1.22. SecondBuffer.occ.....	A-97
4.1.23. SP.occ.....	A-100
4.1.24. SPControl.occ.....	A-104
4.1.25. Target.occ.....	A-106
4.1.26. TargetLead.occ.....	A-109
4.1.27. TrackDisplay.occ.....	A-113
4.1.28. XBar.occ.....	A-120
4.2. Include Files.....	A-122
4.2.1. CRTC.inc.....	A-122
4.2.2. G_Header.inc.....	A-123
4.2.3. S_Header.inc.....	A-125
4.2.4. Sys6.inc.....	A-127
4.3. Makefiles.....	A-137
4.3.1. Seeker.....	A-137
4.3.2. HostSeeker.....	A-139
4.3.3. Graphics.....	A-140
4.3.4. B409stub.l2h.....	A-141
4.3.5. GIF02.l8h.....	A-141
4.3.6. HostSeeker.l8h.....	A-141
4.3.7. ImageDisplay.l8h.....	A-142
4.3.8. Loader.l8h.....	A-142
4.3.9. RunSeekr.l8h.....	A-142
4.3.10. TrackDisplay.l8h.....	A-142

5. Appendix B: Program Listings (Scene generation for SP testing)	B-1
5.1. PC Source Code (Modula-2)	B-1
5.1.1. SPTTest.mod	B-1
5.1.2. Image.def	B-2
5.1.3. Image.mod	B-2
5.1.4. Objects.def	B-4
5.1.5. Objects.mod	B-4
5.1.6. Scenario.def	B-6
5.1.7. Scenario.mod	B-6
5.2. Transputer Implementation Source Code (Occam)	B-9
5.2.1. SPTTest.pgm	B-9
5.2.2. SPTTest.occ	B-10
5.2.3. Buffer.occ	B-20
5.2.4. G_Line.occ	B-21
5.2.5. G_System.occ	B-27
5.2.6. G_Text.occ	B-29
5.2.7. Video.occ	B-34

Table of Figures

Figure 1: Schedules and Milestones	3,4
Figure 2a-b:	7,8
Figure 3a-c:	10-12
Figure 4: EXOSEEK 2 Data Sources	34
Figure 5: EXOSEEK 2 Dithering	35
Figure 6: LATS Interface	39

1. Introduction

The Seeker Scene Emulator (SSE) is an important element of Georgia Tech's Digital Emulation Technology Laboratory (DETL). The Seeker Scene Emulator provides for real-time emulation of advanced Infrared Focal Plane Arrays, and thus, for real-time testing of closed-loop missile simulations (EXOSIM, for example) and real-time testing of Signal and Object Processing components.

Several programs described in this volume are not strictly SSE derived (Neural Network Signal Processing, for example), however, these projects are dependent upon the SSE for testing. For this reason, LATS interfacing and Neural Network Signal Processing are included in this volume.

1.1. Objectives

A first generation Seeker Scene Emulator has been developed and documented in previous annual reports. Current development centers around maximizing the usefulness of the SSE and improving operation. A major addition to the SSE is the GT-SPI, or SSE/Signal Processing Interface. This component interfaces the Seeker Scene Emulator to the Custom VLSI Signal and Object Processing Chipset developed by the Computer Engineering Laboratory at Georgia Tech. Several changes were made to the interconnection and programming of the SSE to accommodate this device. Because of these changes, an updated description of the operation of the SSE is presented here.

A planned enhancement to the Signal Processing Interface is gamma-injection circuitry. This addition will allow for the testing of gamma-circumvention techniques and/or hardware. The current design calls for this circuitry to operate as a modular signal conditioner to the existing interface.

The Computer Engineering Research Laboratory has been investigating the use of neural networks in the development of signal and object processing of focal plane array signals. One effort is being conducted by the Seeker Scene Emulator design group for hardware reasons to be discussed in a later section.

Portions of the Seeker Scene Emulator have, in the past, been used as a testbed for Signal Processing development. In a related effort, an Advanced Signal Processing Testbed is being constructed using processing elements similar to those in the Seeker Scene Emulator. This unit will enhance Georgia Tech's abilities in the development of VLSI Signal Processing devices by serving as a real-time or near real-time emulation of candidate algorithms and implementations.

An interface between a LATS signal processor and a CERL object processing sub-system is under development. This interface will provide LMSC with capabilities for conducting closed-loop testing.

1.2. Schedules and Milestones

The schedule for Seeker Scene Emulator development is shown in Figure 1. Major sub-headings in the schedule match sections of discussion in this document.

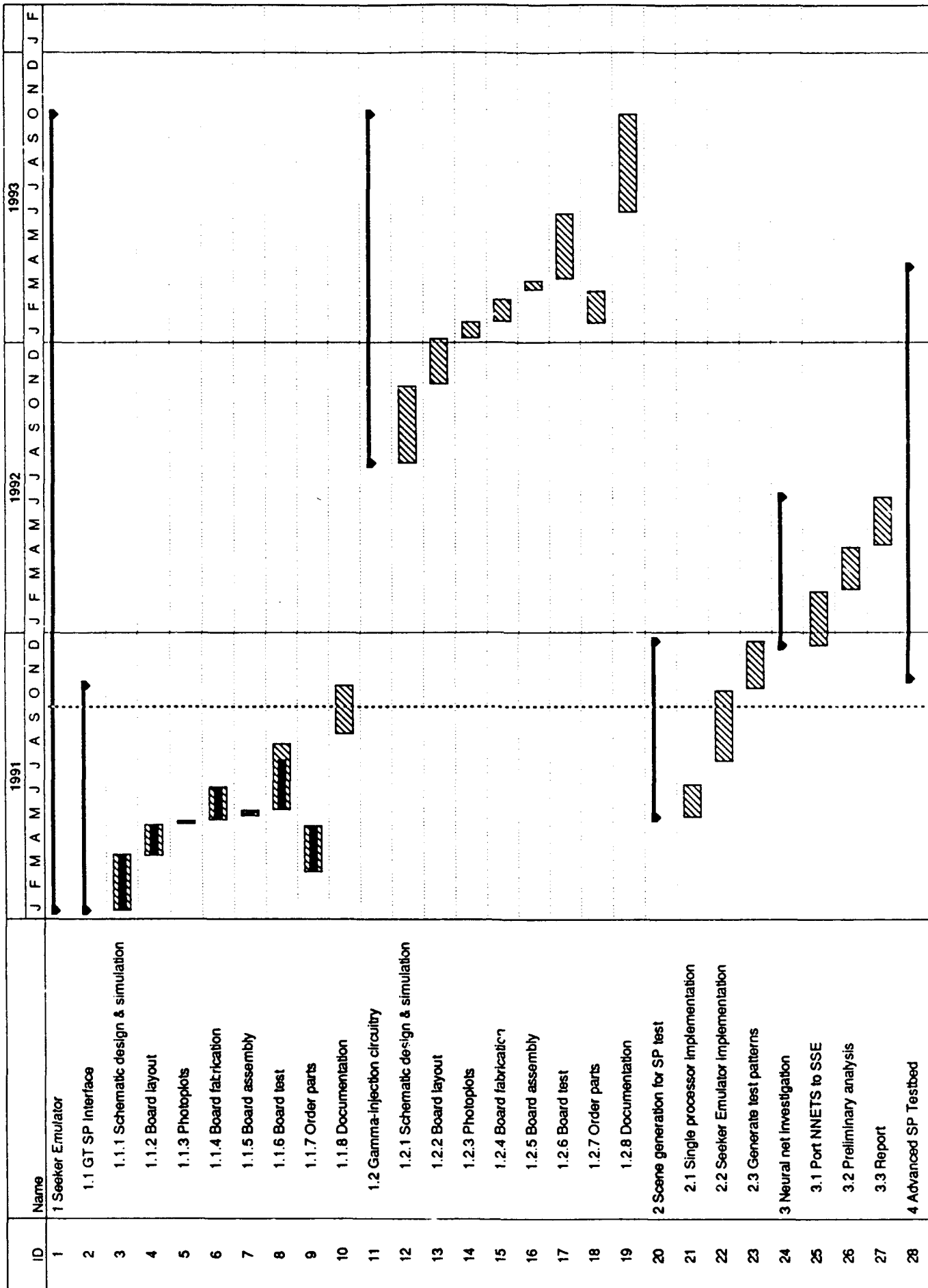
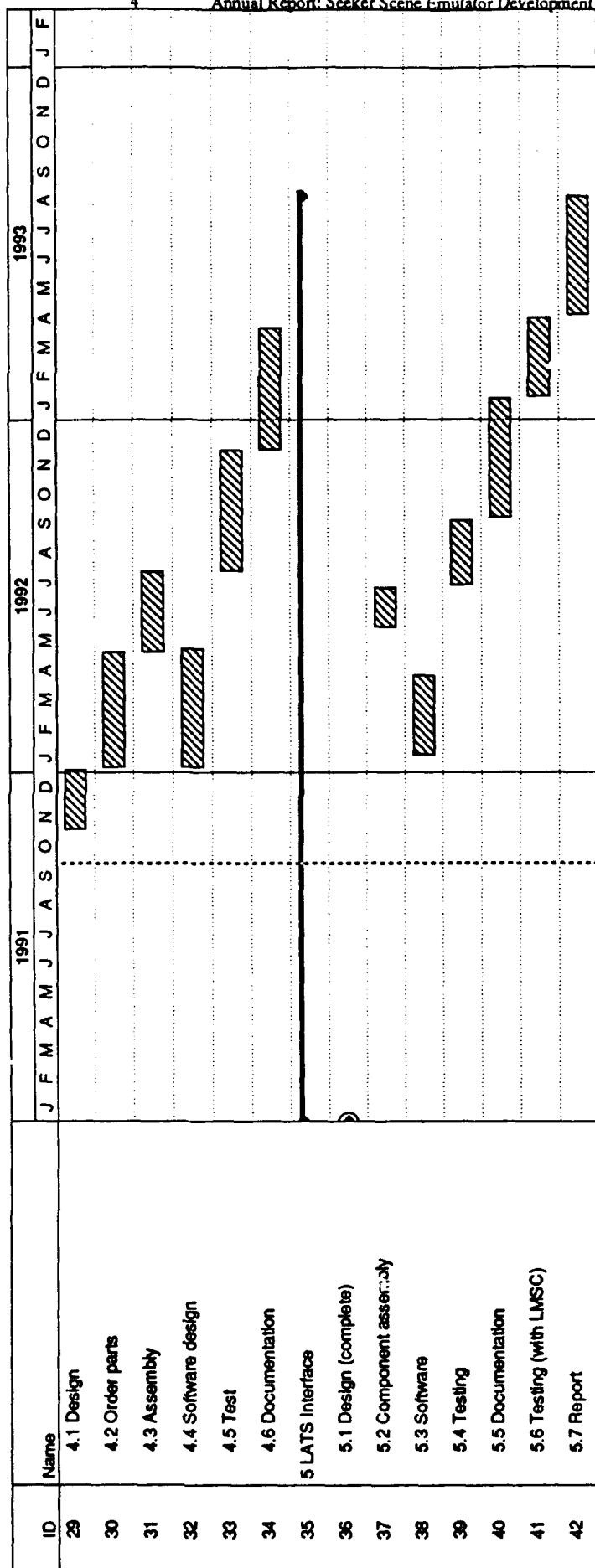


Figure 1. Schedules and Milestones



Figures 1: Schedules and Milestones

2. Seeker Scene Emulator

The Seeker Scene Emulator provides the Georgia Tech Digital Emulation Technology Laboratory with the capability for real-time testing of signal and object processing components. The SSE is capable of emulating infrared focal plane arrays with 128x128 detectors and, up to, 100 frame per second operation.

2.1. Description

The Seeker Scene Emulator has been thoroughly discussed in previous annual reports, so only a brief description will be given here. Updated processor interconnection and software listings are provided. A new component of the Seeker Scene Emulator, the GT-SPI SSE/Signal Processing Interface is presented in detail.

2.2. Hardware components

The whole of the Seeker Scene Emulator encompasses the processors and memory used in storing and manipulating the Seeker Scene data, the custom input/output interfaces, and the off-the-shelf host computers and displays. This can be summarized in the following list:

Main SSE

- 256 T800 Transputers, 2 Mbytes RAM each
- 3 T800 Transputer-based Video Frame Store, 2 MBytes RAM each
- 1 Analog Graphics Display board

Input/Output Interfaces

- GT-XBI Transputer to Georgia Tech PFP Crossbar Interface
- GT-SPI SSE / Signal Processing Interface
- IMS B418 SCSI Transputer Module, Exabyte 4Gbyte, 8mm tape drive

Host Computer, Displays

- 80286-based Industry Standard Architecture Personal Computer
- Enhanced Graphics Adapter and Monitor for PC,
- High-resolution Multi-sync monitor for display of SSE graphics

As a great deal of the Seeker Scene Emulator hardware and software components have previously been published, the following sections will only deal with updating the status of the primary SSE hardware (the processors and their interconnection) and updated software. Additionally, a full discussion of the GT-SPI hardware and software is given.

2.2.1. Seeker Scene Emulator

The Seeker Scene Emulator is primarily composed of 256 Inmos T800 Transputers. The T800 Transputer is a fast 32-bit microprocessor with on-chip floating-point (1.5 MFLOPS) and optimized for parallel processing operation. Each T800 has 4 kbytes of internal high-speed static

RAM and four bi-directional, 20 Mbits/second communication links. In the Seeker Scene Emulator, the majority of the processors are each configured with 2 MBytes of external RAM.

The communication links largely define the operation of a network of Transputers and the interconnection of the processors making up the Seeker Scene Emulator are shown in Figures 2a and 2b.

The bulk of the processors in the SSE handle the storage of the Target data frames. As many processors as desired can be connected into the Target data processor array, and this directly constrains the number of frames allowed per simulation. Frequently, the SSE is operated with 128 processors generating the target image (approximately 230 frames) with the spare processors allocated to Signal Processing emulation, communication routing, etc.

One new component in the SSE is a SCSI Interface module. This module is a commercially available Transputer-based SCSI controller. This module will be used to read scene data from an Exabyte 8mm tape drive. The storage requirements for the SSE data are extremely large with each frame consuming 1 MByte. We have used the Exabyte units, which can hold 4 GBytes per cartridge, as a repository for the data, but an intermediate computer has always been used for control. The new interface should provide direct, high-speed access to the tape data.

Seeker Emulator Interconnections

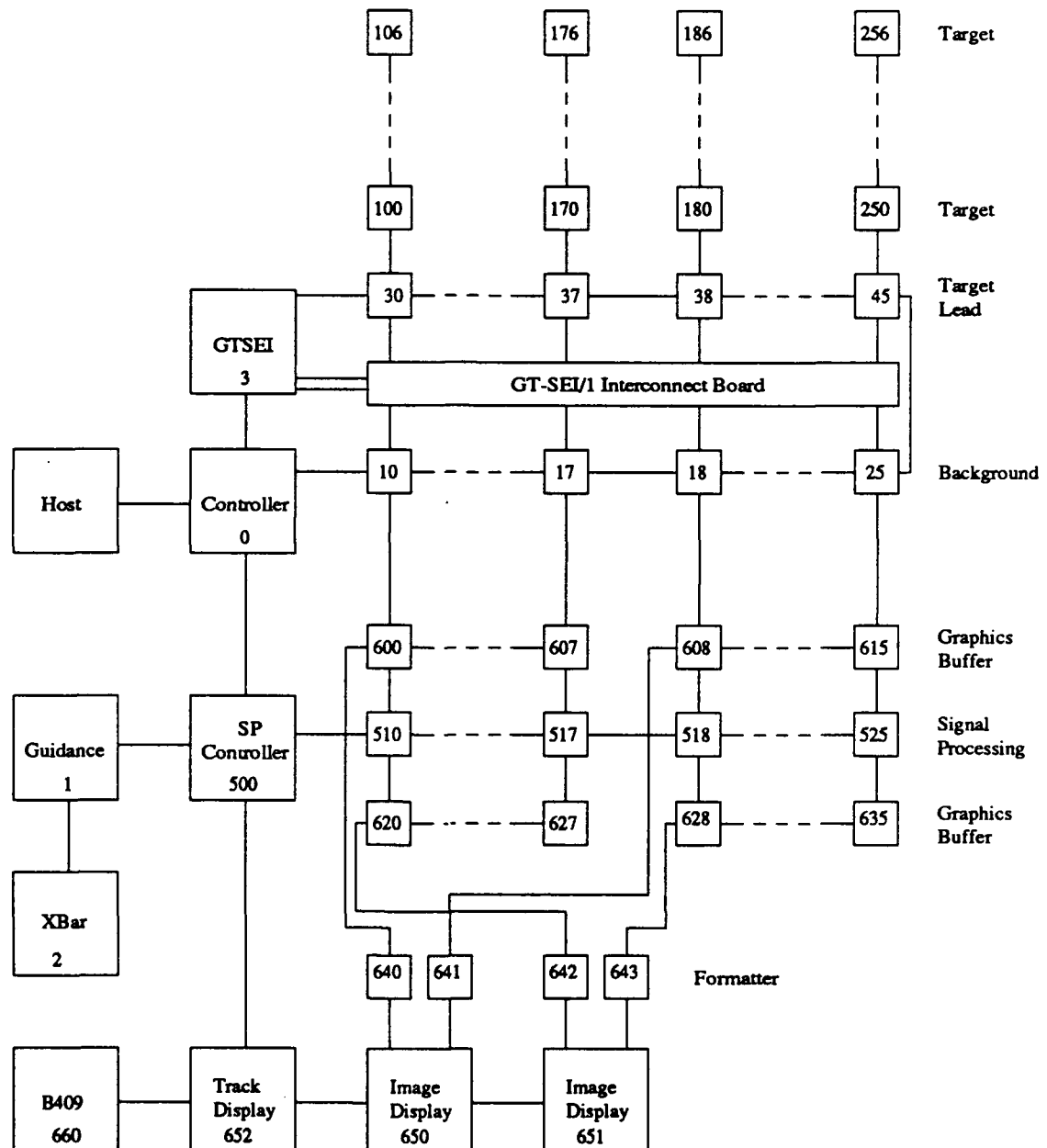


Figure 2a: Processor Identification

Seeker Emulator Interconnections

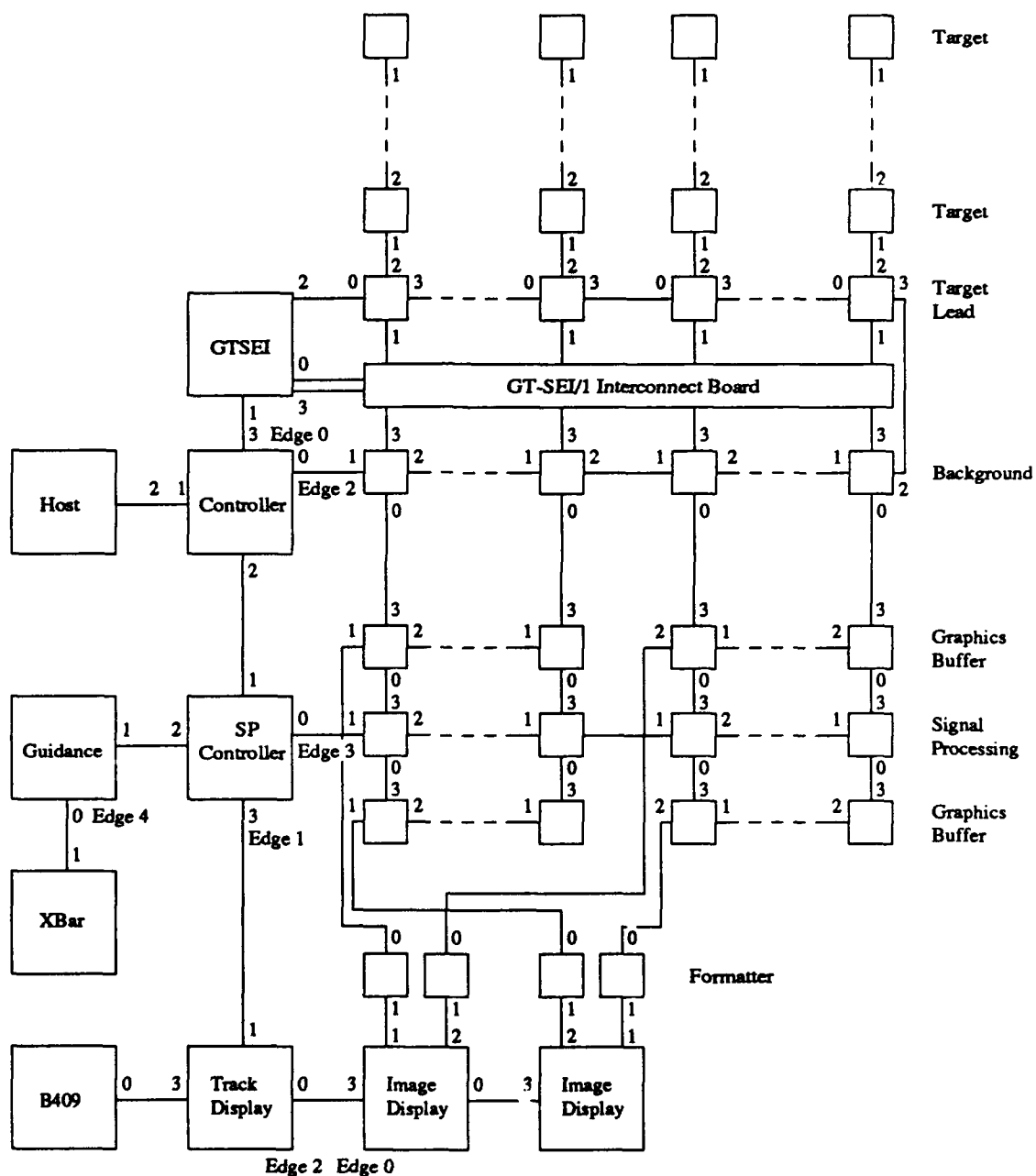


Figure 2b: Link Identification

2.2.2. SSE/SP Interface

The Seeker Scene Emulator is designed for real-time component testing of signal and object processing. Therefore, one requirement for the SSE would be the inclusion of some interface to an external signal processing sub-system. The GT-SPI SSE/SP Interface was built for that purpose.

2.2.2.1. Description

The GT-SPI takes as its basic input a stream of data representing an emulated focal plane array. The Seeker Scene Emulator, as mentioned above, uses the Inmos Transputer as the basis for its architecture and naturally performs communications through any number of Inmos standard links. Therefore, the GT-SPI was built with a Transputer as the controlling processor and it can communicate through up to four 20 Mbit/second links. The output from the GT-SPI consists of a set of clocks, strobes, and 16-bit pixel data.

The schematics for the interface are shown in Figures 3a-3c.

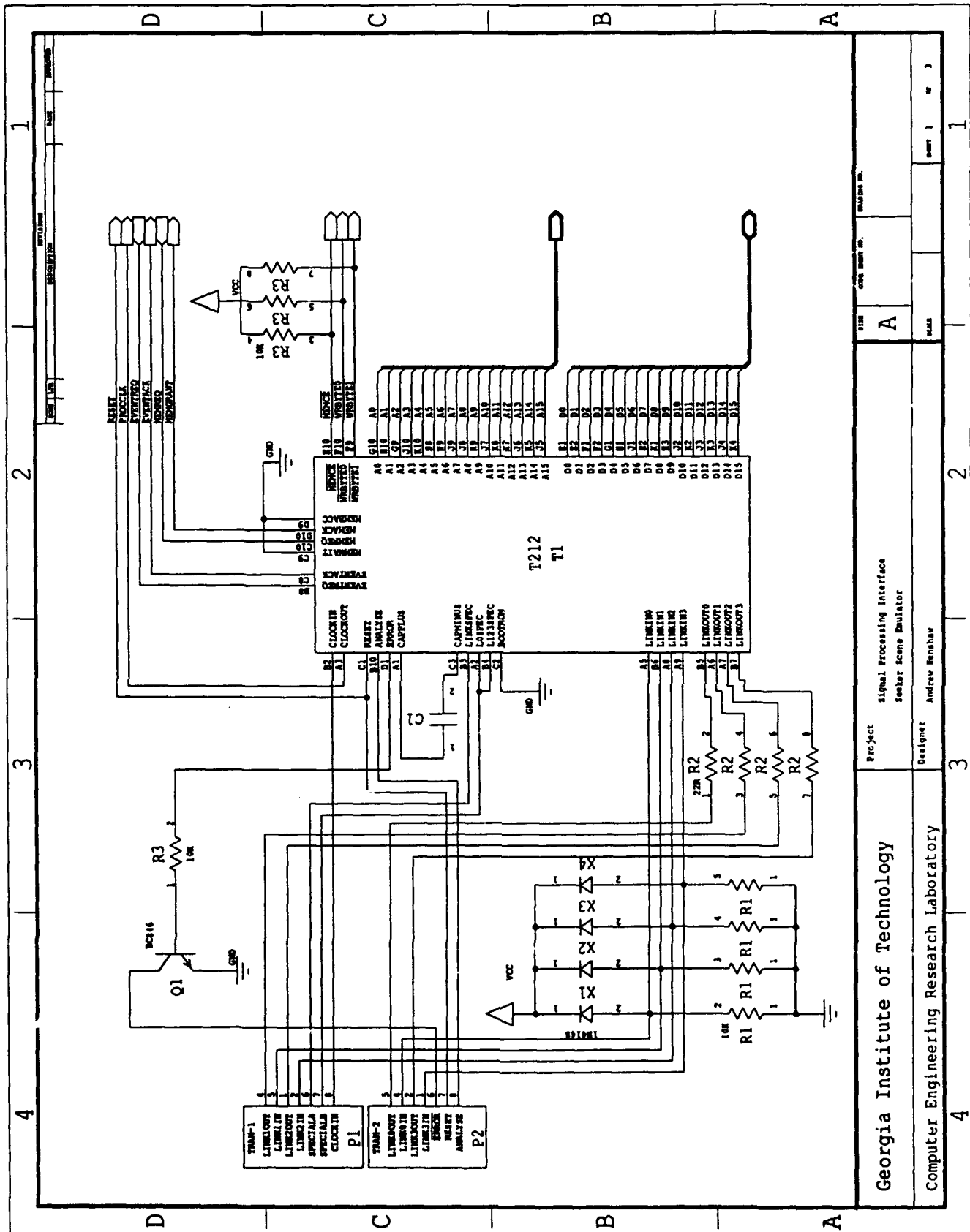


Figure 3a: GT-SPI Schematic

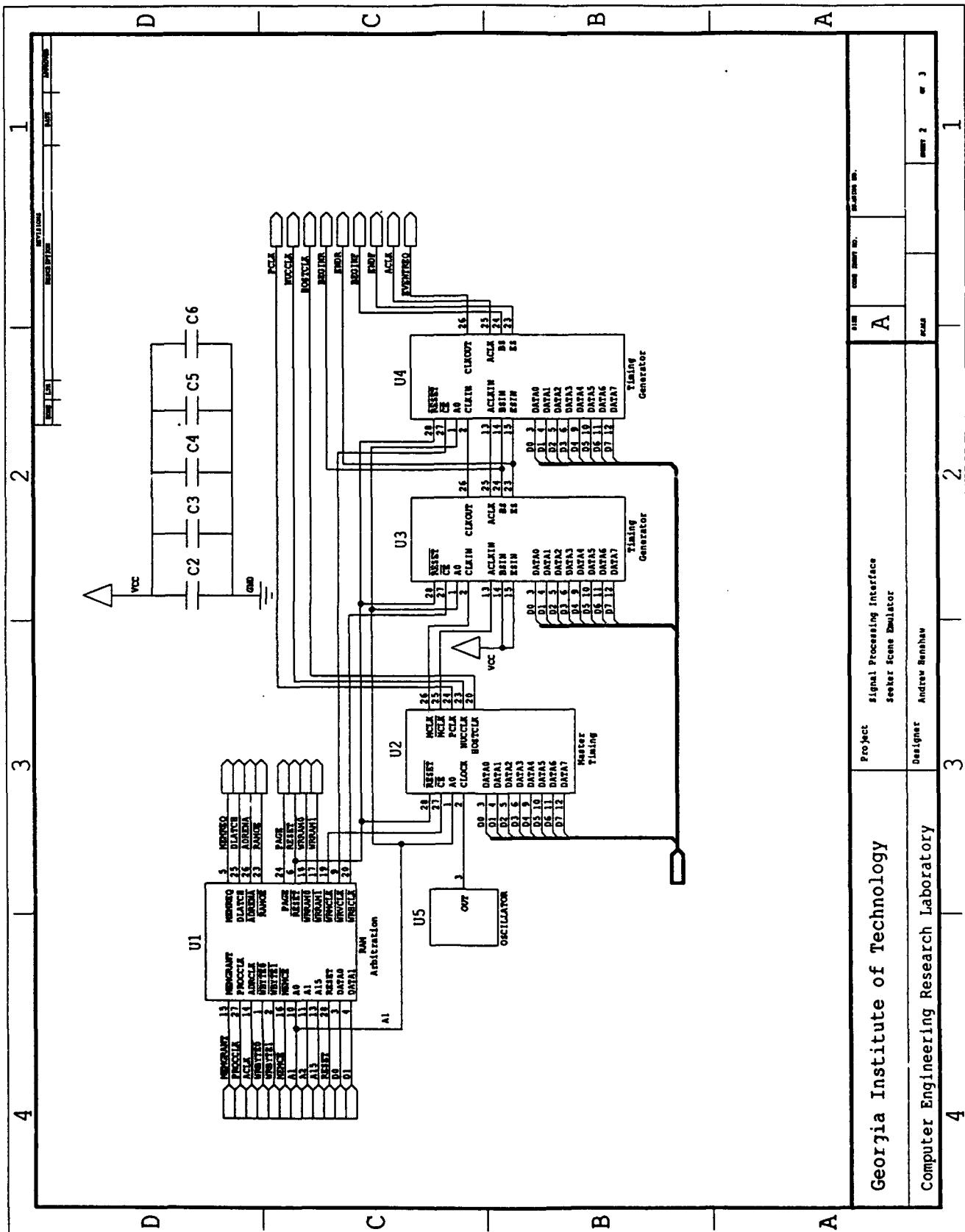


Figure 3b: GT-SPI Schematic
(continued)

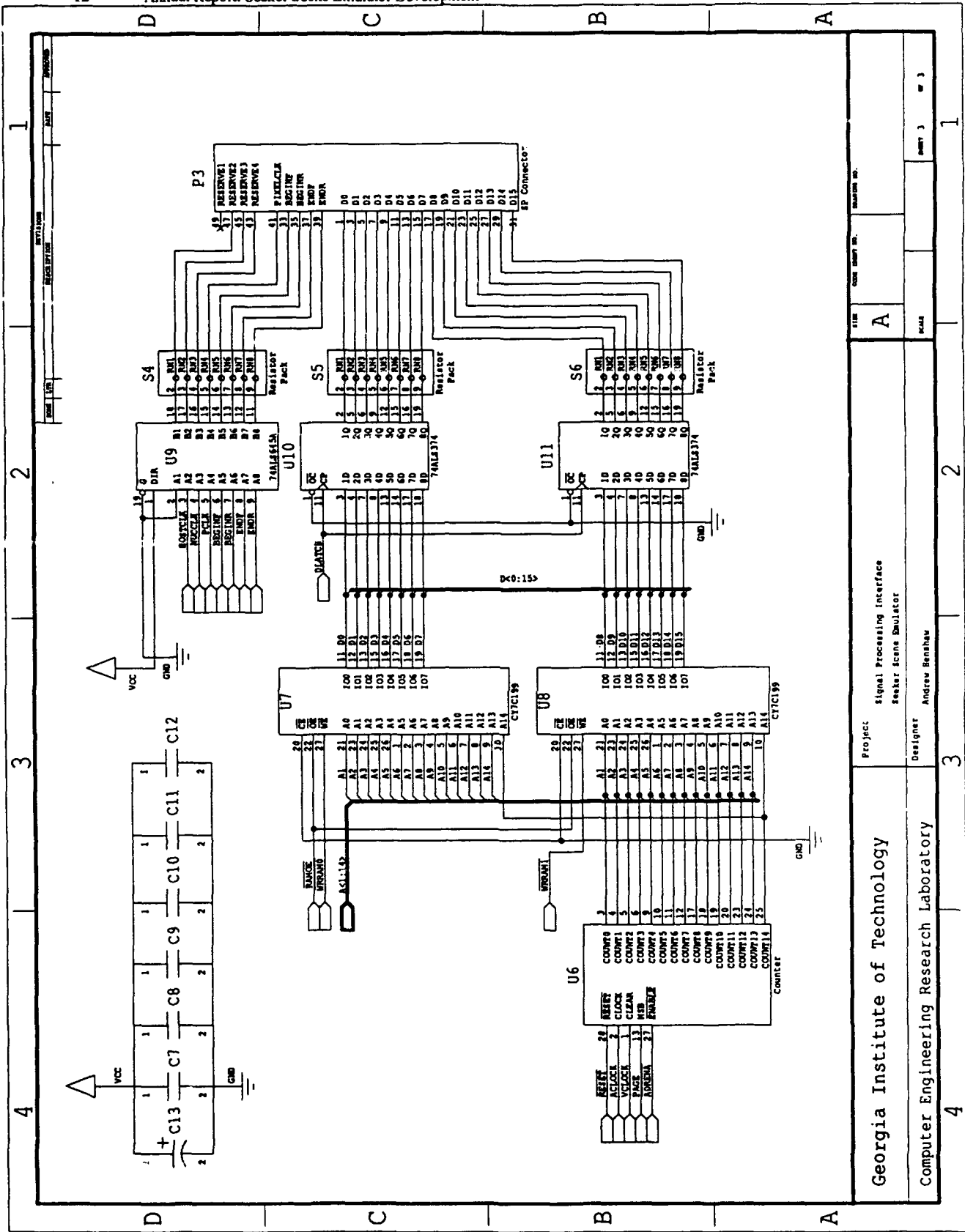


Figure 3c: GT-SPI Schematic
(continued)

2.2.2.2. Hardware

The GT-SPI is primarily controlled by a single processor with an attached frame store. This store is shared with direct memory access readout electronics that presents the pixel data to an output bus at regular intervals. The timing for these intervals is generated by three custom-programmed Erasable Programmable Logic Devices (EPLDs). These EPLDs are the Master Clock Generator, the Vertical Clock Generator, and the Horizontal Clock Generator. Arbitration between frame store accesses is controlled by another custom EPLD, the Memory Access Controller. Finally, a fifth EPLD, the Address Generator, is used to generate 15-bit memory addresses.

2.2.2.2.1. Processor

The controlling processor for the GT-SPI is an Inmos T225 16-bit Transputer. This processor was chosen for its compatibility with the Transputer-based Seeker Scene Emulator, the 16-bit word access, and its fast memory interface. The T225 has 4 Inmos-standard communication links operating at 20 Mbits/second. External memory access cycle time is 100 ns for a bandwidth of 20 Mbytes/second. The T225 has 4 Kbytes of fast (50 ns) internal RAM and can address 60 or 64Kbytes of external memory, depending upon configuration.

2.2.2.2.2. Memory Access Controller

The Memory Access Controller is an Altera MAXPlus EPM5032 EPLD programmed as shown in Listing 2. The primary responsibility of the Memory Access Controller is to arbitrate access to the memory used for storing the FPA image data. The image data RAM is a 35 ns, 32Kx16 memory consisting of two Cypress CY7C192 chips. The 32Kx16 address space permits the storing of two individual frames of 128x128x16 bit data.

As the address space of the T225 does not directly support 64Kbytes of external memory along with 4 Kbytes of internal memory, an addressable register controls access to either of the two banks of 16Kx16 RAM for both the processor and the readout electronics. These bits can be set individually, so that the processor could be writing into one bank while the readout was accessing the other bank.

Although the GT-SPI is intended for use with emulations of 128x128x16 bit Focal Plane Arrays, smaller sizes can be tested (as will be discussed in the Timing chips section). Additionally, provision has been made to allow for readout of the entire 32Kx16 data space for arrays up to 181x181 or 256x128.

In operation, the T225 will write to or read from external memory until a *Memory Request* signal is sent from the Memory Access Controller. This signal is generated once per pixel output and will indicate that the T225 must relinquish control of the external memory bus as soon as possible. The T225 will tri-state its data and address signal lines and issue a *Memory Grant* signal. A T225 running at 20MHz is guaranteed to grant direct memory access with 100 ns. Once the *Memory Grant* has been received, the Memory Access Controller enables the Address Generator (discussed below) and latches the output data into the 74ALS245 buffers. At this

point, the *Memory Request* is terminated and control passes back to the processor. Even while running at the maximum specified output rate (600 ns per pixel), the processor will still be able to access external memory twice per pixel cycle.

The Memory Access Controller chip contains two processor-accessible control registers. These registers and their address offsets are given below in Table 1. Additionally, this EPLD decodes address enable lines for all of the other addressable chips in the system. These addresses are also provided.

Table 1: GT-SPI Memory Map		
Name	Address Offset	Bit designation
Ram Control	6	0 - Page of RAM for processor access 1 - Page of RAM for DMA access
System Control	7	0 - Reset
Master Clock Chip Enable	0	---
Horizontal Clock Chip Enable	2	---
Vertical Clock Chip Enable	4	---
RAM Chip Enable	32768	---

Listing 1: Memory Access Controller

```

*****
*
*           Ram Control Logic
*
*****
TITLE "Ram Logic";
DESIGN IS "LOGIC"
BEGIN
    DEVICE IS "EPM5032D"
    BEGIN
        Reset      @ 28 : INPUT ;
        ProcClock   @ 27 : INPUT ;
        /MemCE      @ 16 : INPUT ;
        /WByte0     @ 1 : INPUT ;
        /WByte1     @ 2 : INPUT ;
        MemGrant    @ 15 : INPUT ;
        A0          @ 10 : INPUT ;
        A1          @ 11 : INPUT ;
        A2          @ 12 : INPUT ;
        A15         @ 13 : INPUT ;
        Data0       @ 3 : INPUT ;
        Data1       @ 4 : INPUT ;
        aclk        @ 14 : INPUT ;
        /AdrEnable  @ 26 : OUTPUT ;
        DataLatch   @ 25 : OUTPUT ;
        MemReq      @ 5 : OUTPUT ;
        Page        @ 24 : OUTPUT ;
        /RamOE      @ 23 : OUTPUT ;
        /reset      @ 6 : OUTPUT ;
        /WrHclock   @ 20 : OUTPUT ;
        /WrMclock   @ 19 : OUTPUT ;
        /WrRam0     @ 18 : OUTPUT ;
        /WrRam1     @ 17 : OUTPUT ;
        /WrVclock   @ 9 : OUTPUT ;

    END;
END;

SUBDESIGN Logic
(
    MemGrant      : INPUT ;
    ProcClock     : INPUT ;
    aclk          : INPUT ;
    /WByte0       : INPUT ;
    /WByte1       : INPUT ;
    /MemCE        : INPUT ;
    A2,A1,A0,
    A15           : INPUT ;
    Reset         : INPUT ;
    Data[1..0]    : INPUT ;

    MemReq        : OUTPUT ;
    DataLatch     : OUTPUT ;
    /AdrEnable    : OUTPUT ;
    /RamOE        : OUTPUT ;
    /WrMclock     : OUTPUT ;
    /WrHclock     : OUTPUT ;
    /WrVclock     : OUTPUT ;
    /WrRam0       : OUTPUT ;
    /WrRam1       : OUTPUT ;
    Page          : OUTPUT ;
    /reset        : OUTPUT ;
)
VARIABLE
    enable,

```

```

activate : TFF ;
mrff      : DFF ;
/ram,
/mclock,
/hclock,
/vclock,
/regs     : NODE ;
/WrRegs   : NODE ;
AdrEnable : NODE ;
/control  : NODE ;
/WrControl: NODE ;
RamA14ff  : DFF ;
CntA14ff  : DFF ;
resetff   : DFF ;
/WByte0int : TFF ;
/WByte1int : TFF ;
BEGIN
/mclock    = NOT (A15 AND !A2 AND !A1) ;
/hclock    = NOT (A15 AND !A2 AND A1) ;
/vclock    = NOT (A15 AND A2 AND !A1) ;

/regs      = NOT (A15 AND A2 AND A1 AND !A0) ;
/control   = NOT (A15 AND A2 AND A1 AND A0) ;
/ram       = A15 ;
/WrMclock  = /mclock OR /WByte0int OR /MemCE ;
/WrHclock  = /hclock OR /WByte0int OR /MemCE ;
/WrVclock  = /vclock OR /WByte0int OR /MemCE ;
/WrRegs    = /regs OR /WByte0int OR /MemCE ;
/WrControl = /control OR /WByte0int OR /MemCE ;
/WrRam0    = /ram OR /WByte0int OR /MemCE ;
/WrRam1    = /ram OR /WByte1int OR /MemCE ;

% RamA14ff holds the high address bit used by the processor %
% when selecting RAM -- enables the T212 to bank switch %
% 64K into 32K %
% CntA14ff holds the corresponding bit for the read-out %
% circuitry. This allows for frame buffering %

RamA14ff.clk = /WrRegs ;
RamA14ff     = Data0 ;
RamA14ff.clrn = !Reset ;

CntA14ff.clk = /WrRegs ;
CntA14ff     = Data1 ;
CntA14ff.clrn = !Reset ;

resetff.clk = /WrControl ;
resetff.clrn = !Reset ;
resetff     = Data0 ;
/reset      = resetff ;

activate.prn = MemGrant ;
activate.t   = enable.q ;
activate.clk = ProcClock ;

enable.t     = VCC ;
enable.prn   = MemGrant ;
enable.clk   = DataLatch ;

mrff.d       = VCC ;
mrff.clk     = aclk ;
mrff.clrn    = DataLatch ;

DataLatch    = activate.q ;
MemReq       = mrff.q ;
AdrEnable    = NOT MCELL(DataLatch) ;
/AdrEnable   = NOT AdrEnable ;

```

```
/WByte0int.t = VCC ;  
/WByte0int.prn = !/WByte0 ;  
/WByte0int.clk = !ProcClock ;  
  
/WBytelint.t = VCC ;  
/WBytelint.prn = !/WByte1 ;  
/WBytelint.clk = !ProcClock ;  
  
/RamCE = NOT ((/WByte0 AND /WByte1) OR /MemCE) ;  
Page = (RamA14ff AND /AdrEnable) OR (CntA14ff AND AdrEnable) ;  
  
END ;
```

2.2.2.2.3. Master Clock Generator

The Master Clock generator is a programmable clock divider for generating frequencies of 2 to 255 times slower than a reference clock (typically a 10 MHz oscillator). This clock is provided to the Signal processing boards as the NUC Clock and is divided by 4 to yield the Pixel Clock.

The Master Clock Generator is contained in one EPM5032 programming logic shown in Listing 2. It has one programmable 8-bit register to hold the divider parameter and a one-bit register to hold a clock enable gate. When this gate is programmed LOW then the master clock is disabled, a HIGH value enables the clock.

The Divider Register is accessed at the base address of the Master Clock Chip Enable (see above) while the Clock Gate Register is at the next higher address.

Listing 2: Master Clock Generator

```

*****
*                               (Brief description of the circuit)                               *
*                               Master Timing Generator                               *
*****
TITLE "Master Timing Generator";

DESIGN IS "master"
BEGIN
    DEVICE IS "EPM5032"
    BEGIN
        /reset    @ 28 : INPUT  ;
        /ce       @ 27 : INPUT  ;
        a0        @ 1  : INPUT  ;
        clock     @ 2  : INPUT  ;
        data0     @ 3  : INPUT  ;
        data1     @ 4  : INPUT  ;
        data2     @ 5  : INPUT  ;
        data3     @ 6  : INPUT  ;
        data4     @ 9  : INPUT  ;
        data5     @ 10 : INPUT  ;
        data6     @ 11 : INPUT  ;
        data7     @ 12 : INPUT  ;
        mclk      @ 26 : OUTPUT  ;
        /mclk     @ 25 : OUTPUT  ;
        pclk      @ 24 : OUTPUT  ;
        nucclk    @ 23 : OUTPUT  ;
        hostclk   @ 20 : OUTPUT  ;

    END;
END;

SUBDESIGN master
(
    /reset,
    clock,
    /ce,
    a0,
    data[7..0] : INPUT;

    mclk, /mclk, pclk,
    nucclk, hostclk : OUTPUT ;
)
VARIABLE
divide[7..0] : DFF ;
count[7..0]  : DFF ;
mclk_ff      : DFF ;
gate         : DFF ;
gateSync     : DFF ;
/select[1..0] : NODE ;
tempclk1     : TFF ;
tempclk2     : TFF ;
BEGIN
    /select[0] = a0 ;
    /select[1] = !a0 ;
    divide[]  = data[] ;
    gate      = data[0] ;

    divide[].clk = /ce OR /select[0] ;
    gate.clk    = /ce OR /select[1] ;

    count[].clrn = %gate AND % /reset ;
    mclk_ff.clrn = %gate AND % /reset ;
    count[].clk = clock;

```

```
mclk_ff.clk = clock ;
IF (count[] == 0) THEN
    count[] = divide[] ;
    mclk_ff = NOT mclk_ff ;
ELSE
    count[] = count[] - 1 ;
    mclk_ff = mclk_ff ;
END IF ;
hostclk = clock ;
nucclk = mclk_ff ;
tempclk1.clk = mclk_ff ;
tempclk1.t = VCC ;
tempclk2.clk = tempclk1 ;
tempclk2.t = VCC ;
% mclk = tempclk2 ; %
% /mclk = !tempclk2 ; %
pclk = mcell (mcell (mcell ( mcell (!tempclk2) ) ) ) );
gateSync.clk = pclk ;
gateSync = gate ;
mclk = tempClk2 AND gateSync ;
/mclk = !(tempClk2 AND gateSync) ;
END ;
```

2.2.2.2.4. Horizontal and Vertical Clock Generators

The Horizontal and Vertical Clock Generators are, conceptually, very similar and will be discussed together. Except for some minor modifications, the programming of the two chips is nearly identical (see Listings 3 and 4) and each chip is a EPM5032.

To provide as much flexibility as possible, the design of these two chips reflects the design of conventional video circuitry. The horizontal timing chip controls the number of pixels per row of the emulated Focal Plane Array, while the vertical timing chip controls the number of rows per frame. Additionally, both chips can be programmed to provide a dead time between active rows or frames. This dead time would correspond to the horizontal and vertical retrace and blanking intervals of a normal video interface. For example, a 128x128 pixel FPA could be emulated by programming the *active registers* of the horizontal and vertical chips 128 each. Then, a small blanking interval of two pixel clocks could be inserted between each readout row by programming the *blanking register* to two.

Both the *active register* and the *blanking register* are 8 bits wide and, thus, timing for FPAs up to 255 active and blanked pixels per row, and 255 active and blanked rows per frame can be emulated (although memory constraints would limit the total number of active pixels). One use of the blanking periods is to reduce the total frame rate. For example, if the base frame rate with no blanking intervals is 100 frames per second, an addition of horizontal or blanking periods equal to the number of pixels per row or rows per frame would reduce the frame rate to 50 frames per second.

In the operation of the GT-SPI, the vertical clock chip takes its input clock from the output of the horizontal clock chip which cycles once per row. The horizontal clock chip uses the pixel clock as its own clock. Both outputs are used to derive the signal which is fed to the Address Generator chip (see below).

The programming address for the *active register* is the base address of each respective chip, while the address for each *blanking register* is at base address plus one. One point to note is that both registers implicitly add one to the value stored. Therefore, the smallest active or blanking interval would be one clock cycle.

Listing 3: Horizontal Clock Generator

```

*****
*
*           Timing Generator
*
*****
TITLE "Timing Generator";
DESIGN IS "HCLOCK"
BEGIN
    DEVICE IS "EPM5032D"
    BEGIN
        /reset    @ 28 : INPUT ;
        /ce       @ 27 : INPUT ;
        a0        @ 1  : INPUT ;
        clkIn     @ 2  : INPUT ;
        aClkIn    @ 13 : INPUT ;
        bsIn      @ 14 : INPUT ;
        esIn      @ 15 : INPUT ;
        blankIn   @ 17 : INPUT ;
        data0     @ 3  : INPUT ;
        data1     @ 4  : INPUT ;
        data2     @ 5  : INPUT ;
        data3     @ 6  : INPUT ;
        data4     @ 9  : INPUT ;
        data5     @ 10 : INPUT ;
        data6     @ 11 : INPUT ;
        data7     @ 12 : INPUT ;
        aClk      @ 25 : OUTPUT ;
        bs        @ 24 : OUTPUT ;
        clkOut    @ 26 : OUTPUT ;
        es        @ 23 : OUTPUT ;

    END;
END;

SUBDESIGN hclock
(
    /reset,
    /ce,
    clkIn,
    blankIn,
    aClkIn,
    a0,
    bsIn, esIn,
    data[7..0] : INPUT;

    clkOut,
    aClk,
    bs,
    es          : OUTPUT ;
)

VARIABLE
    active[7..0] : DFF ;
    blanking[7..0] : DFF ;
    count[7..0] : DFF ;
    clkOut_ff : DFF ;
    /select[1..0] : NODE ;
    /reset_sync : DFF ;

BEGIN
    /select[0]    = a0 ;
    /select[1]    = !a0 ;
    active[]      = data[] ;
    blanking[]    = data[] ;

```

```

active[].clk  = /ce OR /select[0] ;
blanking[].clk = /ce OR /select[1] ;

count[].clk = clkIn;
clkOut_ff.clk = clkIn ;
/reset_sync.clk = clkIn;
/reset_sync.clrn = /reset ;
/reset_sync = /reset ;

% IF NOT (/reset) THEN %
%   count[] = blanking[] ; %
%   clkOut_ff = VCC ; %
%   bs = GND; %
%   es = GND; %
clkOut_ff.prn = /reset_sync ;
count[].clrn = /reset_sync ;

IF (count[] == 0) & (clkOut_ff == GND) THEN
    count[] = blanking[] ;
    clkOut_ff = VCC ;
%   es = esIn AND !blankIn ; %
    es = GND ;
    bs = GND ;
ELSIF (count[] == 1) & (clkOut_ff == GND) THEN
    count[] = count[] - 1 ;
    clkOut_ff = clkOut_ff ;
    es = esIn AND !blankIn ;
    bs = GND ;
ELSIF (count[] == 0) & (clkOut_ff == VCC) THEN
    count[] = active[] ;
    clkOut_ff = GND ;
    bs = bsIn & !blankIn & /reset_sync;
    es = GND ;
ELSE
    count[] = count[] - 1 ;
    clkOut_ff = clkOut_ff ;
    bs = GND;
    es = GND ;
END IF ;
clkOut = clkOut_ff ;
aClk  = aClkIn AND !clkOut;
END ;

```

Listing 4: Vertical Clock Generator

```

*****
*
*           Timing Generator
*
*****
TITLE "Timing Generator";
DESIGN IS "VCLOCK"
BEGIN
    DEVICE IS "EPM5032D"
    BEGIN
        /reset    @ 28 : INPUT ;
        /ce       @ 27 : INPUT ;
        a0        @  1 : INPUT ;
        clkIn     @  2 : INPUT ;
        aClkIn    @ 13 : INPUT ;
        bsIn      @ 14 : INPUT ;
        esIn      @ 15 : INPUT ;
        data0     @  3 : INPUT ;
        data1     @  4 : INPUT ;
        data2     @  5 : INPUT ;
        data3     @  6 : INPUT ;
        data4     @  9 : INPUT ;
        data5     @ 10 : INPUT ;
        data6     @ 11 : INPUT ;
        data7     @ 12 : INPUT ;
        aClk      @ 25 : OUTPUT ;
        bs        @ 24 : OUTPUT ;
        clkOut    @ 26 : OUTPUT ;
        es        @ 23 : OUTPUT ;
        blankOut  @ 17 : OUTPUT ;
    END;
END;

SUBDESIGN vclock
(
    /reset,
    /ce,
    clkIn,
    aClkIn,
    a0,
    bsIn, esIn,
    data[7..0] : INPUT;

    clkOut,
    blankOut,
    aClk,
    bs,
    es          : OUTPUT ;
)

VARIABLE
    active[7..0] : DFF ;
    blanking[7..0] : DFF ;
    count[7..0] : DFF ;
    clkOut_ff : DFF ;
    /select[1..0] : NODE ;
    start : DFF ;

BEGIN
    /select[0] = a0 ;
    /select[1] = !a0 ;
    active[] = data[] ;
    blanking[] = data[] ;

```

```

active[].clk  = /ce OR /select[0] ;
blanking[].clk = /ce OR /select[1] ;

count[].clk   = clkIn;
clkOut_ff.clk = clkIn ;
start.clk     = clkIn ;

% IF NOT (/reset) THEN %
%   count[] = blanking[] ; %
%   clkOut_ff = VCC ; %
%   bs = GND; %
%   es = GND; %
%   blankOut = VCC ;%
%   start = GND ; %

count[].clrn = /reset ;
clkOut_ff.prn = /reset ;
start.clrn = /reset ;

IF (count[] == 0) & (clkOut_ff == GND) THEN
    count[] = blanking[] ;
    clkOut_ff = VCC ;
    es = esIn AND /reset;
    bs = GND ;
    blankOut = start OR !/reset;
ELSIF (count[] == 0) & (clkOut_ff == VCC) THEN
    count[] = active[] ;
    clkOut_ff = GND ;
    bs = bsIn AND /reset;
    es = GND ;
    start = VCC ;
    blankOut = %GND OR % !/reset;
ELSE
    count[] = count[] - 1 ;
    clkOut_ff = clkOut_ff ;
    bs = GND;
    es = GND ;
    blankOut = clkOut_ff OR !/reset;
END IF ;
clkOut = clkOut_ff ;

aClk   = aClkIn AND !clkOut ;
END ;

```

2.2.2.2.5. Address Generator

The Address Generator chip is also programmed (Listing 5) into an EPM5032, demonstrating the flexibility of this EPLD. This EPLD was specified for all of these parts for the principal reason of requiring a small inventory of separate devices for maintenance and development. However, the EPM5032 is so flexible, that almost all of the functions shown here could not have been done with lesser EPLDs.

The Address Generator uses the input clock to drive a simple counter circuit that continually increments until the counter is reset by the clear or /reset inputs or the counter wraps around. The counter is a 15-bit register and can, therefore, generate addresses for the entire 32 kwords of memory. The most significant bit of the output address is combinatorially derived from the most significant bit of the counter register and the *msb* signal controlled by the MemoryControl Chip described above. This signal reflects the status of the page access control bits for the processor and the readout circuitry.

Listing 5: Address Generator

```

*****
*
*           Address Generator
*
*****
TITLE "Address Generator";

DESIGN IS "COUNTER"
BEGIN
    DEVICE IS "EPM5032D"
    BEGIN
        /reset    @ 28 : INPUT ;
        /enable   @ 27 : INPUT ;
        clear     @ 1  : INPUT ;
        clock     @ 2  : INPUT ;
        msb       @ 13 : INPUT ;
        q0        @ 3  : OUTPUT ;
        q1        @ 4  : OUTPUT ;
        q2        @ 5  : OUTPUT ;
        q3        @ 6  : OUTPUT ;
        q4        @ 9  : OUTPUT ;
        q5        @ 10 : OUTPUT ;
        q6        @ 11 : OUTPUT ;
        q7        @ 12 : OUTPUT ;
        q8        @ 17 : OUTPUT ;
        q9        @ 18 : OUTPUT ;
        q10       @ 19 : OUTPUT ;
        q11       @ 20 : OUTPUT ;
        q12       @ 23 : OUTPUT ;
        q13       @ 24 : OUTPUT ;
        q14       @ 25 : OUTPUT ;
    END;
END;

SUBDESIGN Counter
(
    /reset      : INPUT ;
    clock       : INPUT ;
    clear       : INPUT ;
    msb         : INPUT ;
    /enable     : INPUT ;
    q[14..0]    : OUTPUT ;
)
VARIABLE
    count[14..0] : DFF ;
    buffer[13..0]: TRI ;

BEGIN
    q[13..0]      = buffer[13..0] ;
    buffer[13..0] = count[13..0] ;
    buffer[13..0].oe = NOT /enable ;
    q[14]          = (NOT(/enable) AND count[14]) OR msb ;
    count[].prn = (NOT clear) AND /reset ;
    count[].clk  = clock;
    count[] = count[] + 1 ;
END ;

```

2.2.2.3. Programming

A simple program to test the functionality of the GT-SPI interface is shown in Listing 6. In operation, the GT-SPI processor initializes the control registers and clears the frame store. Then it enters a loop waiting for commands from an external processor, which, in this case, is the host processor accepting input from a user.

This program allows a user to specify clock rates, reset the direct memory access circuitry, suspend the generation of the frame and row strobes, and to set individual or groups of pixel elements to specified values.

Appendix A lists the code for the operation of the entire Seeker Scene Emulator. The code supporting the GT-SPI is given in file "gtspi.occ" and shows the programming of the interface as an extension of the SSE. Initialization is done as discussed above and then the program goes into a tight loop where it simply receives and stores blocks of pixel data. In this version, the data arrives in parallel from two separate channels, but up to four could be used, if greater speed was required.

Listing 6: Example program for GT-SPI

```

PROC testerr (CHAN OF ANY test, from.host)
--{{{ external memory and registers
{16384}INT frame.buffer :
PORT OF INT mclock.data :
PORT OF INT mclock.control :
PORT OF INT hclock.active :
PORT OF INT hclock.blanking :
PORT OF INT vclock.active :
PORT OF INT vclock.blanking :
PORT OF INT master.register :
PORT OF INT control.register :
CHAN OF ANY sync :
--}}}
--{{{ placements
VAL base.address      IS #0800 :
PLACE mclock.data     AT base.address :
PLACE mclock.control  AT base.address + 1:
PLACE hclock.active   AT base.address + 2 :
PLACE hclock.blanking AT base.address + 3 :
PLACE vclock.active   AT base.address + 4 :
PLACE vclock.blanking AT base.address + 5 :
PLACE master.register AT base.address + 6 :
PLACE control.register AT base.address + 7 :
PLACE frame.buffer    AT #4000 :

PLACE sync            AT 8 :
--}}}

--{{{ PROC testMem
PROC testMem (VAL INT check)
  INT temp :
  SEQ
    master.register ! 0
    SEQ i = 0 FOR 16384
      frame.buffer[i] := i
    master.register ! 1
    SEQ i = 0 FOR 16384
      frame.buffer[i] := 16383 - i
    master.register ! 0
    --{{{
    SEQ i = 0 FOR 16384
    SEQ
      temp := frame.buffer[i]
      IF
        temp = i
        SKIP
      TRUE
      --{{{
      SEQ
        test ! (INT32 i)
        test ! (INT32 check)
        test ! (INT32 temp)
      --}}}
    --}}}
    master.register ! 1
    --{{{
    SEQ i = 0 FOR 16384
    SEQ
      temp := frame.buffer[i]
      IF
        temp = (16383 - i)
        SKIP
      TRUE
      --{{{

```

```

        SEQ
        test ! (INT32 i)
        test ! (INT32 check)
        test ! (INT32 temp)
        --}}}
    --}}}
:
--}}}
TIMER clock :
INT selection, scale, now :
INT numRows, numCols :
SEQ
    mclock.control ! 0      -- disable clock
    control.register ! 0    -- reset
    control.register ! 1    -- clear the reset condition
    --{{{ initialize
    SEQ
        numRows := 128
        numCols := 128
        hclock.active ! numCols - 1
        hclock.blanking ! 1
        vclock.active ! numRows - 1
        vclock.blanking ! 1
        mclock.data ! 0
    --}}}

    SEQ i = 0 FOR 16384
        frame.buffer[i] := 0
    WHILE TRUE
        SEQ
            from.host ? selection ; scale
            --{{{ CASE selection
            CASE selection
            1
                mclock.data ! scale
            2
                SEQ
                    hclock.active ! scale
                    numCols := scale+1
            3
                hclock.blanking ! scale
            4
                SEQ
                    vclock.active ! scale
                    numRows := scale+1
            5
                vclock.blanking ! scale
            6
                SEQ i = 0 FOR 16384
                    frame.buffer[i] := scale
            7
                INT row, col :
                SEQ
                    from.host ? row; col
                    frame.buffer[col + (row * numCols)] := scale
            8
                control.register ! scale  -- reset control is bit 0
            9
                SEQ
                    mclock.control ! scale  -- clock enable is bit 0
                IF
                    scale = 0
                    SEQ
                        control.register ! 0  -- reset
                        control.register ! 1  -- unreset
                TRUE
                SKIP

```

```
20
  --{{{ monitor event request pin
  INT any :
  BOOL continue :
  SEQ
    continue := TRUE
    WHILE continue
      PRI ALT
        from.host ? any
        continue := FALSE
        sync ? any
        test ! 0
      --}}}
  ELSE
    SKIP
  --}}}
:
```

2.2.3. Gamma-injection circuitry

A requirement for the real-time testing of gamma-circumvention hardware and algorithms has been identified. Unfortunately, the high data rates (greater than 500 frames per second) have in the past made this impracticable. Georgia Tech's Computer Engineering Research Laboratory has proposed an implementation that could provide testing at rates approaching 500 frames per second for FPA windows of size 32x32.

Our initial approach is to use the current output of the Seeker Scene Emulator at rates up to 100 frames per second. A new module will then resample this output at a 5 times rate, and add noise with a distribution equivalent to that of the modeled gamma noise.

Operation of the circuit would rely upon several banks or RAM to store the current output frame from the SSE. These banks would then be accessed in parallel at a total rate of 500 frames per second. Parallel bank access is critical to keeping the chip access time to a reasonable level. For a 500 fps, 32x32 window, the average access rate would be approximately 2 ns per pixel. Therefore, 16 banks of chips could be used to provide attainable access rates of 32 ns per pixel.

As the data is read out of the SSE, a bank of custom circuits will generate gamma-noise and add it to, or replace, the pixel data stream. The new data stream will then be demultiplexed to the required number of output channels.

2.3. Software components

There are several software components making up the operation of the Seeker Scene Emulator. First, there is the run-time code used for loading pre-calculated data, communicating with the PFP, and generating the precisely timed output data. This code is listed in Appendix A, with the code for each processor listed as a separate text file. The master configuration file is "seeker.pgm".

Additionally, an off-line data generation program is currently required by the SSE. This program takes trajectory data, scene descriptions, and focal plane array characteristics, and generates the Target and Fixed-pattern noise data files.

Finally, we are developing another program for the testing of Signal and Object Processing algorithms. This will be a non-real-time implementation that allows the user to generate a wide variety of patterns and structures. This program will provide a developer with the capability of testing specific capabilities or weaknesses of an algorithm without trying to configure an intercept scenario for the same purpose.

2.3.1. Seeker Scene Emulator operation

The Seeker Scene Emulator is designed to provide emulation of focal plane arrays of 128x128 elements at rates up to 100 frames per second. As such, the requirements for data processing are very tight, with data spread over hundreds of processors. Additionally, there are nearly 20

different programs running at one time on the entire SSE. Although, the coding for the SSE has been presented in previous annual reports, Appendix A lists the updated source code. Extensive modifications have been made for loading data and for incorporation of the GT-SPI.

2.3.2. Seeker Scene Emulator Data Generation

The Seeker Scene Emulator uses target and fixed-pattern noise data files that have been generated assuming a perfect line-of-sight. This data is obtained from by providing an intercept trajectory, scene description and focal plane characteristics to some off-line focal plane array simulator. Currently, BDM's ESSING program is used for this, however, we are investigating the use of the Strategic Scene Generation Model (SSGM) as a front-end for this operation.

2.3.2.1. ESSING

BDM Corporation has delivered to Georgia Tech a focal plane array simulation program suitable for data generation for the Georgia Tech Seeker Scene Emulator. This program uses knowledge of RV description, interceptor focal plane characteristics, time, and object position and orientation, to develop a rendering of the image as seen by a focal plane array.

In some respects, a more advanced program is BDM's EXOSEEK Version 2.0. An illustration of the data sources used in the generation of its image is shown in Figure 4. We are requesting that the ESSING program be updated to include all of the capabilities of EXOSEEK Version 2.0, as this would provide us with more accurate emulations of LATS hardware, for example focal plane array dithering (Figure 5).

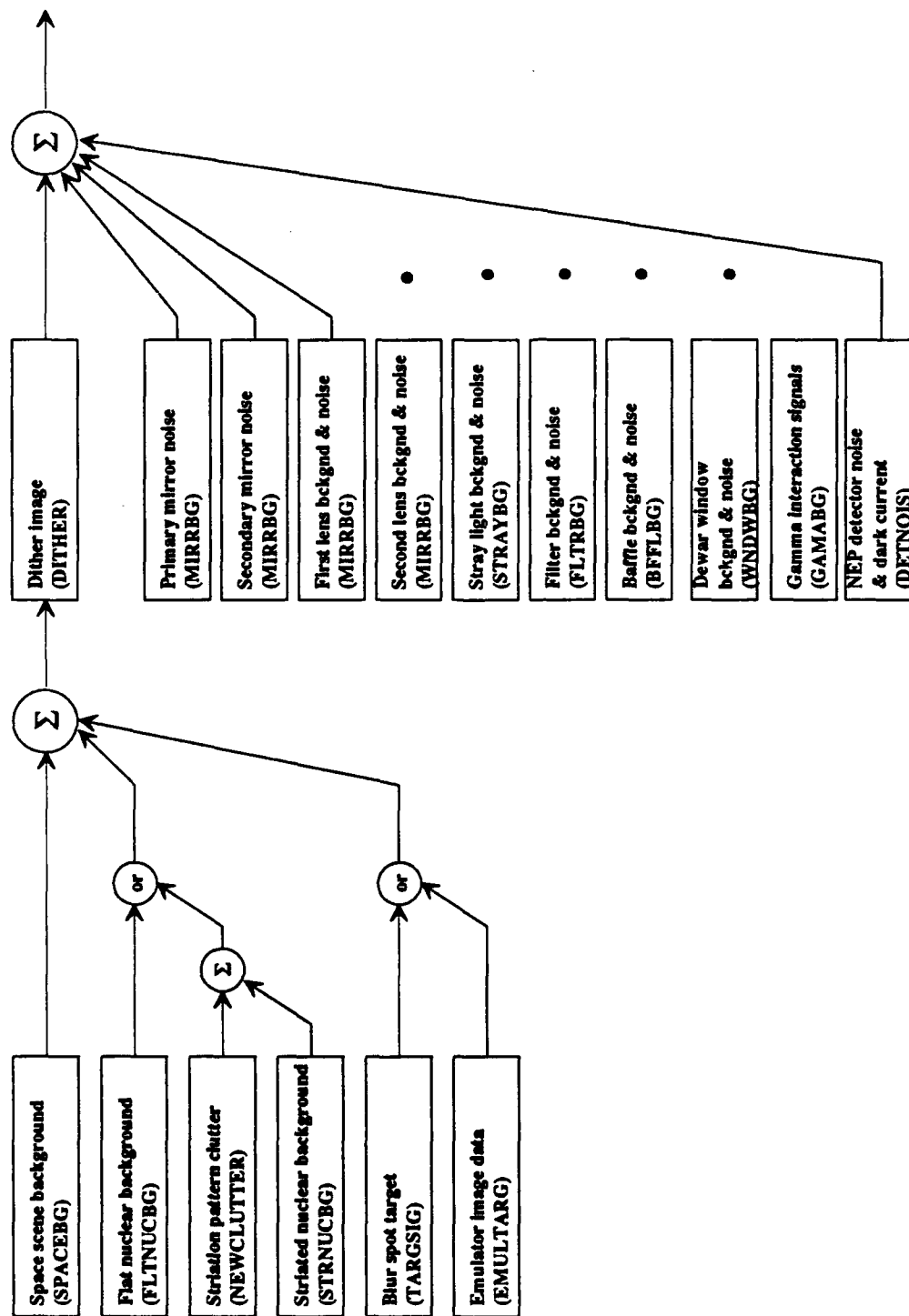


Figure 4: EXOSEEK 2 Data Sources

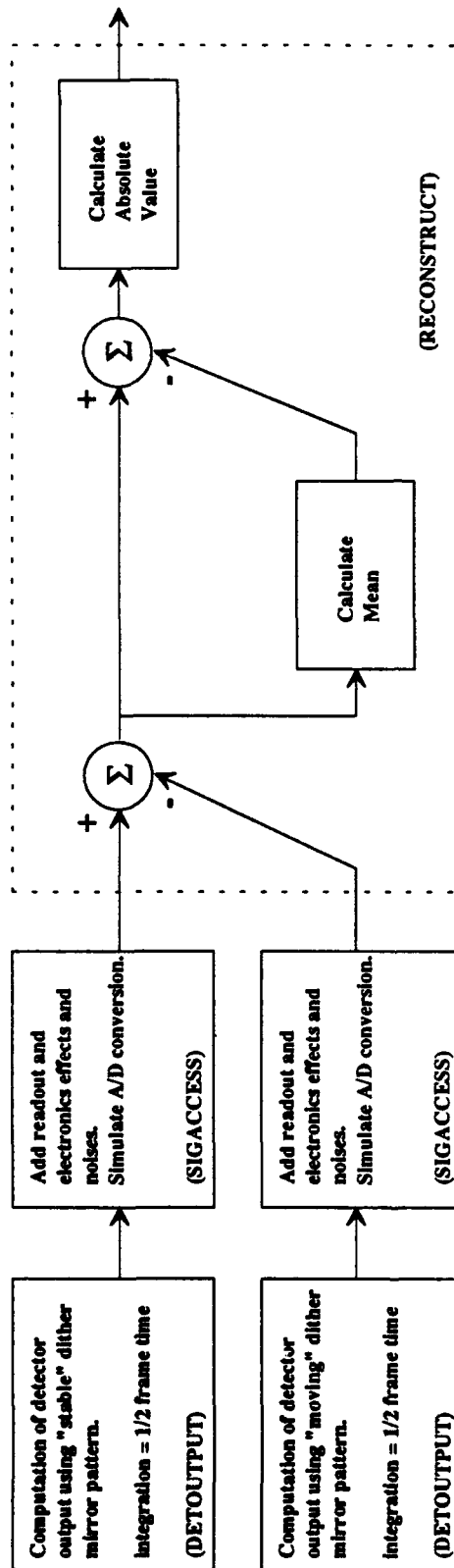


Figure 5: EXOSEEK 2 Dithering

2.3.2.2. Strategic Scene Generation Model

A validated source of input data to the Seeker Scene Emulator is very important. By using the Strategic Scene Generation Model (SSGM) and an appropriate seeker transfer function, the SSE could provide real-time, high-fidelity, validated emulations of LWIR focal plane arrays seekers.

Teledyne Brown Engineering has delivered a set of SSGM-generated data files for testing with the SSE. The SSGM raw image output has been processed by a TBE created LATS-approximate seeker transfer function. We are proposing that a completely LATS compatible seeker transfer function be developed either from TBE's work or BDM's EXOSEEK work.

2.3.3. Scene generation for Signal Processing testing

To provide the Signal Processing algorithm and hardware developer with a convenient method of testing devices and programming, we are developing tools for generating simple and complex scenes. These tools allow a user to build sequences of images using a small set of graphic primitives. The program has been implemented on a Personal Computer with VGA quality graphics, and is currently being ported to the Seeker Scene Emulator with the GT-SPI SSE/Signal Processing Interface. This interface will permit the direct testing of hardware through the program.

The program accepts two data files: an object description file and a scenario construction file. The object file describes bitmaps and polygons that can be placed upon the simulated focal plane array. The scenario construction data file specifies placement, color, and scaling of objects, placement methods, and noise and special effects sources. The scenario includes information describing the beginning and ending of frames and, in the future, timing of frame display. Table 2 describes the currently defined keywords and their meanings.

Table 2: SP Test Keywords	
Keyword	Description
Object	Specifies that the following tokens describe a placeable object
Polygon	Scalable object type made up of n connected line segments
Bitmap	Non-scalable object. Array of points (up to 8x8) with relative intensities
NewFrame	Clear the frame buffer and, depending upon implementation, write old frame to output device

Place	Write object into frame buffer at the specified location,size, and intensity
Array	Replicate the last Place command in a two-dimensional manner
Repeat	Replicate the last Place command in a linear-manner
Noise	Place a rectangular patch of noise into the frame buffer using the specified location and size. If no Noise Model has been selected, a uniform distribution is assumed.
Lorentzian	Lorentzian distribution noise model
Gaussian	Gaussian distribution noise model
Uniform	Uniform distribution noise model
Add	Mode of placing information into the frame buffer where new data is added to old data
Overlay	Mode of placing information into the frame buffer where new data replaces old data
Mask	Limit placement operation to non-zero pixels
NoMask	Clear mask conditions
UnMask	Limit placement operation to zeroed pixels

3. Related Development Efforts

The following sections describe programs that are peripherally related to the Seeker Scene Emulator development.

3.1. Neural net investigation

We have received a copy of the NNETS neural network test software developed by NASA. This software was written for the Transputer using the language C. We will move this onto a subset of the Seeker Scene Emulator and attempt to perform some signal and object processing with it.

3.2. Advanced Signal Processing Testbed

The Advanced Signal Processing Testbed is being developed to facilitate the testing and design of Signal Processing algorithms and implementations. The testbed will consist of a number of next-generation T9000 Transputers with 200 MIPS and 25 MFLOPS capability. We expect to achieve real-time or near real-time emulation of many of the currently developed algorithms.

3.3. LATS interface

We have developed a multi-processor clustering implementation which, for small arrays, duplicates the functionality of the Georgia Tech Clustering and Centroiding chips. this implementation can cluster and centroid a 32x32 pixel data stream at 100 frames per second. In order to provide LMSC with the capability of performing real-time closed-loop testing of the LATS FPA and signal processing hardware, we are developing an interface between the LATS testbed and our multi-processor clustering and centroiding unit. This interface, shown in Figure 6 will consist of the Inmos Transputers needed for the object processing along with a commercially-available Transputer-to-SCSI interface.

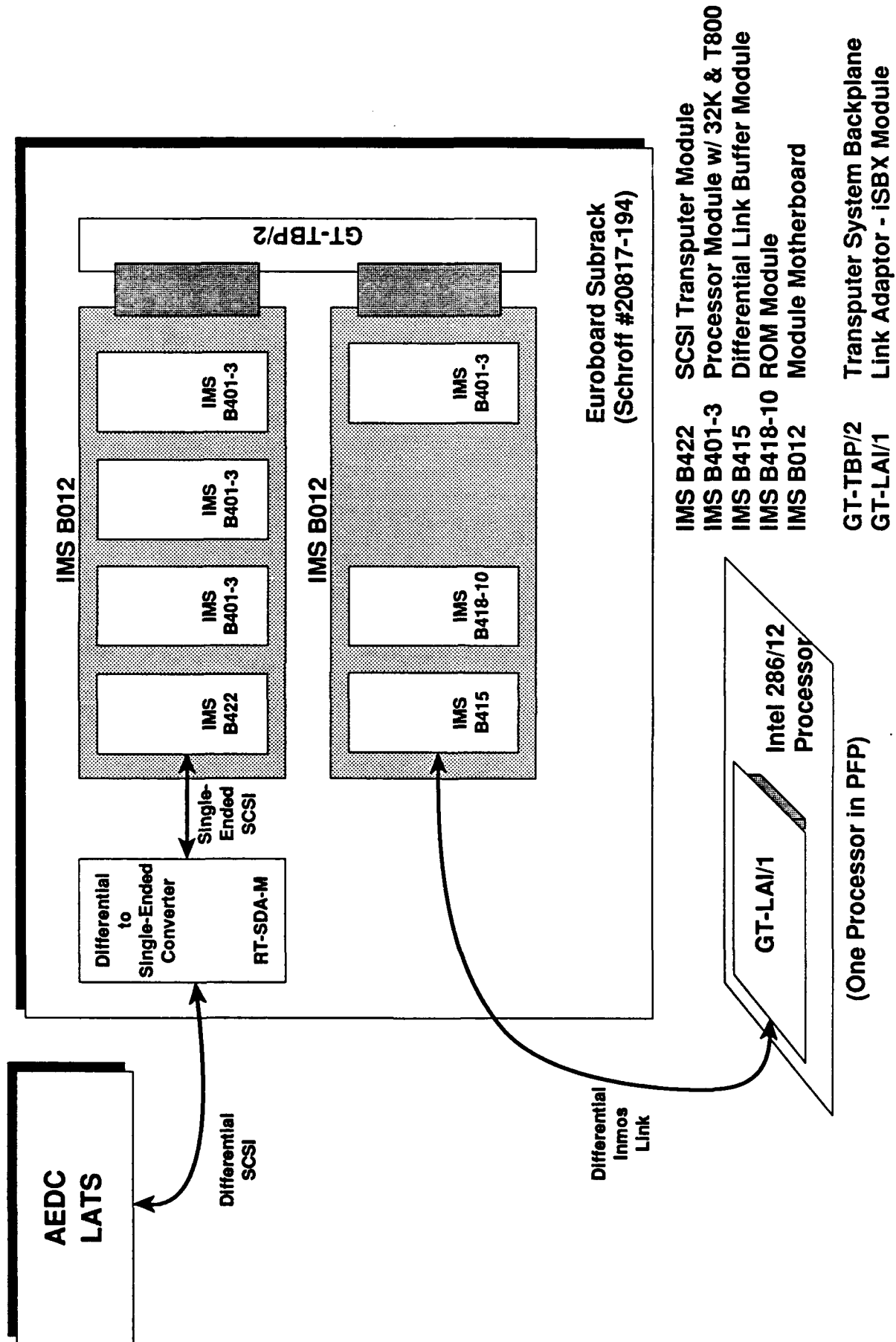


Figure 6: LATs Interface

4. Appendix A: Seeker Scene Emulator Source Code

4.1. Occam Source Code

4.1.1. Seeker.pgm

```
--{{{ SC HostSeek
--:::A 4 10
#USE "HostSeek.c8h"
--{{{F HostSeek
--:::F hostseek.OCC
--}}}
--}}}

--{{{ SC Controller
--:::A 4 10
#USE "controll.t8h"
--{{{F Controller
--:::F CONTROLL.OCC
--}}}
--}}}

--{{{ SC Guidance
--:::A 4 10
#USE "guidance.t8h"
--{{{F Guidance
--:::F GUIDANCE.OCC
--}}}
--}}}

--{{{ SC XBar
--:::A 4 10
#USE "xbar.t2h"
--{{{F XBar
--:::F XBAR.OCC
--}}}
--}}}

--{{{ SC GTSPI
#USE "gtspi.t2h"
--{{{F gtspi.occ
--:::F GTSPI.OCC
--}}}
--}}}

--{{{ SeekerEmulator
#USE "gtsei.t2h"
--{{{ SC GTSEI
--:::A 4 10
--{{{F gtsei.occ
--:::F GTSEI.OCC
--}}}
--}}}

#USE "backgrou.t8h"
--{{{ SC Background
--:::A 4 10
--{{{F Background
--:::F BACKGROU.OCC
--}}}
--}}}

#USE "targetle.t8h"
--{{{ SC TargetLead
--:::A 4 10
--{{{F TargetLead
--:::F TARGETLE.OCC
--}}}
--}}}
```

```
#USE "target.t8h"
--{{{ SC Target
--:::A 4 10
--{{{F Target
--:::F TARGET.OCC
--}}}
--}}}
--}}}
--{{{ SignalProcessing
#USE "spcontro.t8h"
--{{{ SC SPController
--:::A 4 10
--{{{F SPController
--:::F SPCONTRO.OCC
--}}}
--}}}
#USE "sp.t4h"
--{{{ SC SP
--:::A 4 10
--{{{F SP
--:::F SP.OCC
--}}}
--}}}
--}}}
--{{{ Graphics
#USE "firstbuf.t8h"
--{{{ SC FirstBuffer
--:::A 4 10
--{{{F FirstBuffer
--:::F FIRSTBUF.OCC
--}}}
--}}}
--}}}

#USE "secondbu.t8h"
--{{{ SC SecondBuffer
--:::A 4 10
--{{{F SecondBuffer
--:::F SECONDBU.OCC
--}}}
--}}}
--}}}

#USE "formatte.t8h"
--{{{ SC Formatter
--:::A 4 10
--{{{F Formatter
--:::F FORMATTE.OCC
--}}}
--}}}
--}}}

#USE "imagedis.c8h"
--{{{ SC ImageDisplay
--:::A 4 10
--{{{F ImageDisplay
--:::F IMAGEDIS.OCC
--}}}
--}}}
--}}}

#USE "trackdis.c8h"
--{{{ SC TrackDisplay
--:::A 4 10
--{{{F TrackDisplay
--:::F TRACKDIS.OCC
--}}}
--}}}
--}}}

#USE "b409stub.c2h"
--{{{ SC B409.stub
```

```
--:::A 4 10
--(((F B409stub
--:::F B409stub.OCC
--)))
--)))
--)))
--((( configuration
--((( constants
VAL image.shift IS 9 :
--)))
--((( link definitions
VAL link0out IS 0 :
VAL link1out IS 1 :
VAL link2out IS 2 :
VAL link3out IS 3 :
VAL link0in IS 4 :
VAL link1in IS 5 :
VAL link2in IS 6 :
VAL link3in IS 7 :
--)))
--((( channels
CHAN OF ANY Controller.Host, Host.Controller :
CHAN OF ANY Controller.GTSEI, GTSEI.Controller :
CHAN OF ANY Controller.SPC, SPC.Controller :
CHAN OF ANY Crossbar0, Crossbar1 :
CHAN OF ANY Guidance.XBar, XBar.Guidance :
CHAN OF ANY Guidance.SPController, SPController.Guidance :
CHAN OF ANY SPController.Graphics, Graphics.SPController :

[16][8] CHAN OF ANY Target.up, Target.down :
[16] CHAN OF ANY Target.forward, Target.back :
[16] CHAN OF ANY Target.GTSEI, GTSEI.BG :
[17] CHAN OF ANY BG.forward, BG.back :
[16] CHAN OF ANY BG.SP :
[16] CHAN OF ANY Graphics.SP, SP.Graphics :
[17] CHAN OF ANY SP.forward, SP.back :

[9] CHAN OF ANY Extract0, Extract1, Extract2, Extract3, Extract4,
Extract5 :
[4] CHAN OF ANY Format.Graphics :
[4] CHAN OF ANY Graphics.forward, Graphics.back :
--)))
PLACED PAR
--((( HostSeeker
PROCESSOR 1000 T8
PLACE Host.Controller AT link2out :
PLACE Controller.Host AT link2in :
HostSeeker (Controller.Host, Host.Controller)
--)))
--((( Controller
PROCESSOR 0 T8

PLACE Host.Controller AT link1in :
PLACE GTSEI.Controller AT link3in :
PLACE BG.back[0] AT link0in :
PLACE SPC.Controller AT link2in :
PLACE Controller.Host AT link1out :
PLACE Controller.GTSEI AT link3out :
PLACE BG.forward[0] AT link0out :
PLACE Controller.SPC AT link2out :

Controller( Host.Controller, Controller.Host,
GTSEI.Controller, Controller.GTSEI,
BG.back[0], BG.forward[0],
SPC.Controller, Controller.SPC )
--)))
--((( Guidance
```

PROCESSOR 1 T8

```

PLACE SPController.Guidance AT linklin :
PLACE XBar.Guidance AT link0in :
PLACE Guidance.SPController AT linklout :
PLACE Guidance.XBar AT link0out :

```

```

Guidance( SPController.Guidance, Guidance.SPController,
          XBar.Guidance,           Guidance.XBar )

```

--}}}

--{{{ XBar

PROCESSOR 2 T2

```

PLACE Guidance.XBar AT linklin :
PLACE XBar.Guidance AT linklout :

```

```

XBar( Guidance.XBar, XBar.Guidance )

```

--}}}

--{{{ Signal Processing Interface

--}}}

--{{{ SeekerEmulator

--{{{ GTSEI

PROCESSOR 3 T2

```

PLACE Controller.GTSEI AT linklin :
PLACE Target.back[0] AT link2in :
PLACE GTSEI.Controller AT linklout :
PLACE Target.forward[0] AT link2out :
PLACE Crossbar0 AT link0out :
PLACE Crossbar1 AT link3out :

```

```

GTSEI( Controller.GTSEI, GTSEI.Controller,
       Target.back[0], Target.forward[0],
       Crossbar0, Crossbar1 )

```

--}}}

--{{{ GTSPI

PROCESSOR 4 T2

```

PLACE Extract4[0] AT link0in :
PLACE Extract5[0] AT linklin :

```

```

GTSPI( Extract4[0], Extract5[0] )

```

--}}}

--{{{ Background

PLACED PAR i = 0 FOR 16

PROCESSOR 10+i T8

```

PLACE GTSEI.BG[i] AT link3in :
PLACE BG.forward[i] AT linklin :
PLACE BG.back[i+1] AT link2in :
PLACE BG.SP[i] AT link0out :
PLACE BG.back[i] AT linklout :
PLACE BG.forward[i+1] AT link2out :

```

```

Background( GTSEI.BG[i], BG.SP[i],
            BG.forward[i], BG.back[i],
            BG.back[i+1], BG.forward[i+1], i )

```

--}}}

--{{{ TargetLead

PLACED PAR i = 0 FOR 15

PROCESSOR 30+i T8

```

PLACE Target.down[i][0] AT link2in :
PLACE Target.forward[i] AT link0in :
PLACE Target.back[i+1] AT link3in :

```

```

PLACE Target.GTSEI[i]      AT link1out :
PLACE Target.up[i][0]      AT link2out :
PLACE Target.back[i]       AT link0out :
PLACE Target.forward[i+1]  AT link3out :

TargetLead( Target.down[i][0], Target.up[i][0], Target.GTSEI[i],
            Target.forward[i], Target.back[i],
            Target.back[i+1], Target.forward[i+1], i )

PROCESSOR 45 T8
VAL i IS 15 :
PLACE Target.down[i][0]    AT link2in :
PLACE Target.forward[i]    AT link0in :
PLACE BG.forward[16]       AT link3in :
PLACE Target.GTSEI[i]      AT link1out :
PLACE Target.up[i][0]      AT link2out :
PLACE Target.back[i]       AT link0out :
PLACE BG.back[16]          AT link3out :

TargetLead( Target.down[i][0], Target.up[i][0], Target.GTSEI[i],
            Target.forward[i], Target.back[i],
            BG.forward[16], BG.back[16], i )
--}}}
--{{{ Target
PLACED PAR i= 0 FOR 16
    PLACED PAR j = 1 FOR 7

        PROCESSOR 100+((i*10)+j) T8

            PLACE Target.up[i][j-1] AT linklin :
            PLACE Target.down[i][j]  AT link2in :
            PLACE Target.down[i][j-1] AT link1out :
            PLACE Target.up[i][j]     AT link2out :

            Target( Target.up[i][j-1], Target.down[i][j-1],
                    Target.down[i][j], Target.up[i][j], i, j )
--}}}
--}}}
--{{{ SignalProcessing
--{{{ SPController
PROCESSOR 500 T8.

PLACE Controller.SPC      AT linklin :
PLACE Guidance.SPController AT link2in :
PLACE Graphics.SPController AT link3in :
PLACE SP.back[0]          AT link0in :
PLACE SPC.Controller      AT link1out :
PLACE SPController.Guidance AT link2out :
PLACE SPController.Graphics AT link3out :
PLACE SP.forward[0]       AT link0out :

SPController( Controller.SPC, SPC.Controller,
              Guidance.SPController, SPController.Guidance,
              Graphics.SPController, SPController.Graphics,
              SP.back[0], SP.forward[0] )
--}}}
--{{{ SP
PLACED PAR i = 0 FOR 16

    PROCESSOR 510+i T4

        PLACE Graphics.SP[i] AT link3in :
        PLACE SP.forward[i]  AT linklin :
        PLACE SP.back[i+1]   AT link2in :
        PLACE SP.Graphics[i] AT link0out :
        PLACE SP.back[i]     AT link1out :
        PLACE SP.forward[i+1] AT link2out :

```

```

        SP( Graphics.SP[i],      SP.Graphics[i],
            SP.forward[i],      SP.back[i],
            SP.back[i+1],      SP.forward[i+1], i )
--}}}
--}}}
--{{{ Graphics
--{{{ FirstBuffer
PLACED PAR i = 0 FOR 8

    PROCESSOR 600+i T8

        PLACE BG.SP[i]      AT link3in :
        PLACE Extract0[i+1]  AT link2in :
        PLACE Graphics.SP[i] AT link0out :
        PLACE Extract0[i]    AT linklout :

        FirstBuffer( BG.SP[i],      Graphics.SP[i],
                     Extract0[i+1],  Extract0[i],    i, image.shift )

PLACED PAR i = 0 FOR 8

    PROCESSOR 608+i T8

        PLACE BG.SP[i+8]    AT link3in :
        PLACE Extract1[i+1]  AT linklin :
        PLACE Graphics.SP[i+8] AT link0out :
        PLACE Extract1[i]    AT link2out :

        FirstBuffer( BG.SP[i+8],    Graphics.SP[i+8],
                     Extract1[i+1],  Extract1[i],    i, image.shift )
--}}}
--{{{ SecondBuffer
PLACED PAR i = 0 FOR 8

    PROCESSOR 620+i T8

        PLACE SP.Graphics[i] AT link3in :
        PLACE Extract2[i+1]  AT link2in :
        PLACE Extract2[i]    AT linklout :
        PLACE Extract4[i]    AT link0out :

        SecondBuffer( SP.Graphics[i],
                     Extract2[i+1],  Extract2[i], Extract4[i],
                     i, image.shift )

PLACED PAR i = 0 FOR 8

    PROCESSOR 628+i T8

        PLACE SP.Graphics[i+8] AT link3in :
        PLACE Extract3[i+1]    AT linklin :
        PLACE Extract3[i]      AT link2out :
        PLACE Extract5[i]      AT link0out :

        SecondBuffer( SP.Graphics[i+8],
                     Extract3[i+1],  Extract3[i], Extract5[i], i, image.shift )
--}}}
--{{{ Formatters
PROCESSOR 640 T8

        PLACE Extract0[0]      AT link0in :
        PLACE Format.Graphics[0] AT linklout :

        Formatter( Extract0[0],  Format.Graphics[0] )

```

PROCESSOR 641 T8

```
PLACE Extract1[0]      AT link0in :
PLACE Format.Graphics[1] AT linklout :

Formatter( Extract1[0], Format.Graphics[1] )
```

PROCESSOR 642 T8

```
PLACE Extract2[0]      AT link0in :
PLACE Format.Graphics[2] AT linklout :

Formatter( Extract2[0], Format.Graphics[2] )
```

PROCESSOR 643 T8

```
PLACE Extract3[0]      AT link0in :
PLACE Format.Graphics[3] AT linklout :

Formatter( Extract3[0], Format.Graphics[3] )
--}}}
```

--{{{ ImageDisplays

PROCESSOR 650 T8

```
CHAN OF ANY temp1, temp2 :
PLACE Format.Graphics[0]      AT linklin  :
PLACE Format.Graphics[1]      AT link2in  :
PLACE Graphics.back[1]       AT link3in  :
PLACE Graphics.forward[2]     AT link0in  :
PLACE Graphics.forward[1]     AT link3out :
PLACE Graphics.back[2]       AT link0out :

ImageDisplay( Format.Graphics[0], Format.Graphics[1],
              Graphics.back[1], Graphics.forward[1],
              Graphics.forward[2], Graphics.back[2], 0, 1000 )
```

PROCESSOR 651 T8

```
PLACE Format.Graphics[2]      AT link2in  :
PLACE Format.Graphics[3]      AT linklin  :
PLACE Graphics.back[2]       AT link3in  :
PLACE Graphics.forward[3]     AT link0in  :
PLACE Graphics.forward[2]     AT link3out :
PLACE Graphics.back[3]       AT link0out :

ImageDisplay( Format.Graphics[2], Format.Graphics[3],
              Graphics.back[2], Graphics.forward[2],
              Graphics.forward[3], Graphics.back[3], 1, 1000 )
--}}}
```

--{{{ TrackDisplay

PROCESSOR 652 T8

```
PLACE SPController.Graphics AT linklin  :
PLACE Graphics.back[0]      AT link3in  :
PLACE Graphics.forward[1]   AT link0in  :
PLACE Graphics.SPController AT linklout :
PLACE Graphics.forward[0]   AT link3out :
PLACE Graphics.back[1]      AT link0out :

TrackDisplay( SPController.Graphics, Graphics.SPController,
              Graphics.back[0], Graphics.forward[0],
              Graphics.forward[1], Graphics.back[1] )
--}}}
```

--{{{ B409

PROCESSOR 660 T2

```
PLACE Graphics.forward[0] AT link0in :  
PLACE Graphics.back[0] AT link0out :  
  
B409.stub( Graphics.forward[0], Graphics.back[0] )  
--}}}  
--}}}  
--}}}
```

4.1.2. B409.occ

```
--{{{ SC B409
--:::A 3 10
--{{{ B409
#include "crtc.inc"
PROC B409 ( CHAN OF CRTC command )

  --{{{ defs
  VAL bpw.shift IS 1:
  VAL mint IS #8000:
  --}}}
  --{{{ pointers to hardware registers
  VAL ChannelModeSelect.p IS #B000:
  --{{{ ChannelA defs
  VAL ChannelAPixelAddressW.p IS #0000:
  VAL ChannelAColorValue.p IS #0400:
  VAL ChannelAPixelMask.p IS #0800:
  VAL ChannelAPixelAddressR.p IS #0C00:
  --}}}
  --{{{ ChannelB defs
  VAL ChannelBPixelAddressW.p IS #1000:
  VAL ChannelBColorValue.p IS #1400:
  VAL ChannelBPixelMask.p IS #1800:
  VAL ChannelBPixelAddressR.p IS #1C00:
  --}}}
  --{{{ ChannelC defs
  VAL ChannelCPixelAddressW.p IS #2000:
  VAL ChannelCColorValue.p IS #2400:
  VAL ChannelCPixelMask.p IS #2800:
  VAL ChannelCPixelAddressR.p IS #2C00:
  --}}}
  VAL ParameterFIFO.p IS #A000:
  VAL StatusRegister.p IS #A000:
  VAL CommandFIFO.p IS #A002:
  VAL FIFORead.p IS #A002:
  --}}}
  --{{{ hardware placements
  INT ChannelModeSelect :
  --{{{ ChannelA declarations
  INT ChannelAPixelAddressW :
  INT ChannelAColorValue :
  INT ChannelAPixelMask :
  INT ChannelAPixelAddressR :
  --}}}
  --{{{ ChannelB declarations
  INT ChannelBPixelAddressW :
  INT ChannelBColorValue :
  INT ChannelBPixelMask :
  INT ChannelBPixelAddressR :
  --}}}
  --{{{ ChannelC declarations
  INT ChannelCPixelAddressW :
  INT ChannelCColorValue :
  INT ChannelCPixelMask :
  INT ChannelCPixelAddressR :
  --}}}
  INT ParameterFIFO :
  INT StatusRegister :
  INT CommandFIFO :
  INT FIFORead :
  --{{{ ChannelA placements
  PLACE ChannelAPixelAddressW AT (ChannelAPixelAddressW.p >< mint) >> bpw.shift :
  PLACE ChannelAColorValue AT (ChannelAColorValue.p >< mint) >> bpw.shift :
  PLACE ChannelAPixelMask AT (ChannelAPixelMask.p >< mint) >> bpw.shift :
  PLACE ChannelAPixelAddressR AT (ChannelAPixelAddressR.p >< mint) >> bpw.shift :
```

```

--)))
--{{{ ChannelB placements
PLACE ChannelBPixelAddressW AT (ChannelBPixelAddressW.p >< mint) >> bpw.shift :
PLACE ChannelBColorValue AT (ChannelBColorValue.p >< mint) >> bpw.shift :
PLACE ChannelBPixelMask AT (ChannelBPixelMask.p >< mint) >> bpw.shift :
PLACE ChannelBPixelAddressR AT (ChannelBPixelAddressR.p >< mint) >> bpw.shift :
--)))
--{{{ ChannelC placements
PLACE ChannelCPixelAddressW AT (ChannelCPixelAddressW.p >< mint) >> bpw.shift :
PLACE ChannelCColorValue AT (ChannelCColorValue.p >< mint) >> bpw.shift :
PLACE ChannelCPixelMask AT (ChannelCPixelMask.p >< mint) >> bpw.shift :
PLACE ChannelCPixelAddressR AT (ChannelCPixelAddressR.p >< mint) >> bpw.shift :
--)))
PLACE ChannelModeSelect AT (ChannelModeSelect.p >< mint) >> bpw.shift :
PLACE ParameterFIFO AT (ParameterFIFO.p >< mint) >> bpw.shift :
PLACE StatusRegister AT (StatusRegister.p >< mint) >> bpw.shift :
PLACE CommandFIFO AT (CommandFIFO.p >< mint) >> bpw.shift :
PLACE FIFORead AT (FIFORead.p >< mint) >> bpw.shift :
--)))
--{{{ CRTC commands
VAL CTCReset IS #00:
VAL CTCBctrl IS #0D:
--}}}
--{{{ set.colour
PROC set.colour ( VAL INT channel, colour, red, green, blue )
-- set up a colour in the G170 colour look up table
CASE channel
  INT channel.A
  --{{{
  SEQ
    ChannelAPixelAddressW := 255 - (colour /\ #FF)
    ChannelAColorValue := red
    ChannelAColorValue:= green
    ChannelAColorValue := blue
  --}}}
  INT channel.B
  --{{{
  SEQ
    ChannelBPixelAddressW := 255 - (colour /\ #FF)
    ChannelBColorValue := red
    ChannelBColorValue:= green
    ChannelBColorValue := blue
  --}}}
  INT channel.C
  --{{{
  SEQ
    ChannelCPixelAddressW := 255 - (colour /\ #FF)
    ChannelCColorValue := red
    ChannelCColorValue:= green
    ChannelCColorValue := blue
  --}}}
:
--}}}
--{{{ writeCRTC
PROC writeCRTC(INT address, VAL INT data)
  TIMER time:
  INT now:
  SEQ
    address := data
    time ? now
    time ? AFTER (now PLUS 2)
:
--}}}
--{{{ init.G170
PROC init.G170 (VAL INT channel, table)
  INT red, green, blue:
  SEQ

```

```

ChannelAPixelMask := #FF
IF
  --{{{ table 0
  table = 0
  VAL scale IS [0, 18, 36, 54] :
  VAL bias IS 9 :
  SEQ
    set.colour (channel, 0, 0, 0, 0)      -- black
    SEQ i = 1 FOR 255
      INT ix :
      SEQ
        blue := scale[(i>>4)/\3]
        green := scale[(i>>2)/\3]
        red := scale[i/\3]
        IF
          i >= #C0
            blue := blue + bias
          i >= #80
            green := green + bias
          i >= #40
            red := red + bias
        TRUE
        SKIP
      CASE channel
        --{{{ channel.A
        INT channel.A
        SEQ
          ChannelAColorValue := red
          ChannelAColorValue := green
          ChannelAColorValue := blue
        --}}}
        --{{{ channel.B
        INT channel.B
        SEQ
          ChannelBColorValue := red
          ChannelBColorValue := green
          ChannelBColorValue := blue
        --}}}
        --{{{ channel.C
        INT channel.C
        SEQ
          ChannelCColorValue := red
          ChannelCColorValue := green
          ChannelCColorValue := blue
        --}}}
      --}}}
  --{{{ COMMENT table 1
  --:::A 0 0
  --{{{ table 1
  table = 1
  SEQ
    --{{{ 0 - 15 grey scale
    red := -1
    green := -1
    blue := -1
    SEQ i = 0 FOR 16
      SEQ
        red := red + 4
        green := green + 4
        blue := blue + 4
        set.colour (channel, i, red, green, blue)
      --}}}
    --{{{ 16 - 32 red scale
    red := -1
    green := 0
    blue := 0
    SEQ i = 16 FOR 16

```

```

      SEQ
      red := red + 4
      set.colour (channel, i, red, green, blue)
--}}}
--{{{ 32 - 47 green scale
red := 0
green := -1
blue := 0
SEQ i = 32 FOR 16
      SEQ
      green := green + 4
      set.colour (channel, i, red, green, blue)
--}}}
--{{{ 48 - 63 blue scale
red := 0
green := 0
blue := -1
SEQ i = 48 FOR 16
      SEQ
      blue := blue + 4
      set.colour (channel, i, red, green, blue)
--}}}
--{{{ 64 - 79 yellow scale
red := -1
green := -1
blue := 0
SEQ i = 64 FOR 16
      SEQ
      red := red + 4
      green := green + 4
      set.colour (channel, i, red, green, blue)
--}}}
--{{{ 80 - 95 cyan scale
red := 0
green := -1
blue := -1
SEQ i = 80 FOR 16
      SEQ
      green := green + 4
      blue := blue + 4
      set.colour (channel, i, red, green, blue)
--}}}
--{{{ 96 - 111 magenta scale
red := -1
green := 0
blue := -1
SEQ i = 96 FOR 16
      SEQ
      red := red + 4
      blue := blue + 4
      set.colour (channel, i, red, green, blue)
--}}}
--{{{ 112 - 127 red & green scale with third blue
red := 63
green := -1
blue := 21
SEQ i = 112 FOR 16
      SEQ
      green := green + 4
      set.colour (channel, i, red, green, blue)
      red := red - 4
--}}}
--{{{ 128 - 143 green & blue scale with third red
red := 21
green := 63
blue := -1
SEQ i = 128 FOR 16

```

```

SEQ
  blue := blue + 4
  set.colour (channel, i, red, green, blue)
  green := green - 4
--}}}
--{{{ 144 - 159 blue & redscale with third green
red := -1
green := 21
blue := 63
SEQ i = 144 FOR 16
  SEQ
    red := red + 4
    set.colour (channel, i, red, green, blue)
    blue := blue - 4
  --}}}
--{{{ 160 - 175 red & green scale with two-thirds blue
red := 63
green := -1
blue := 42
SEQ i = 160 FOR 16
  SEQ
    green := green + 4
    set.colour (channel, i, red, green, blue)
    red := red - 4
  --}}}
--{{{ 176 - 191 green & blue scale with two-thirds red
red := 42
green := 63
blue := -1
SEQ i = 176 FOR 16
  SEQ
    blue := blue + 4
    set.colour (channel, i, red, green, blue)
    green := green - 4
  --}}}
--{{{ 192 - 207 blue & red scale with two-thirds green
red := -1
green := 42
blue := 63
SEQ i = 192 FOR 16
  SEQ
    red := red + 4
    set.colour (channel, i, red, green, blue)
    blue := blue - 4
  --}}}
--{{{ 208 - 223 red & green scale with full blue
red := 63
green := -1
blue := 63
SEQ i = 208 FOR 16
  SEQ
    green := green + 4
    set.colour (channel, i, red, green, blue)
    red := red - 4
  --}}}
--{{{ 224 - 239 green & blue scale with full red
red := 63
green := 63
blue := -1
SEQ i = 224 FOR 16
  SEQ
    blue := blue + 4
    set.colour (channel, i, red, green, blue)
    green := green - 4
  --}}}
--{{{ 240 - 255 blue & red scale with full green
red := -1

```

```

        green := 63
        blue := 63
        SEQ i = 240 FOR 16
            SEQ
                red := red + 4
                set.colour (channel, i, red, green, blue)
                blue := blue - 4
            --}}}
        --}}}
    --}}}
    TRUE
    SKIP
:
--}}}
--{{{ set.timing
PROC set.timing( VAL INT16 width, height, frame.frequency,
                 VAL INT32 line.frequency, pixel.clock,
                 VAL BOOL  interlace )

--{{{ variables
INT AW, HBP, HFP, HS:
INT horizontal.cells, flyback:
INT AL, SL, VFP, VS, VBP:
--}}}
SEQ
    --{{{ calculate horizontal timing
    horizontal.cells := INT((pixel.clock / line.frequency)>> 5)
    AW := ((INT width) >> 5)
    --{{{ COMMENT
    --:::A 0 0
    --{{{
    IF
        AW > (horizontal.cells - (horizontal.cells/5)) -- 80%
        STOP -- Display set too wide
        TRUE
        SKIP
    --}}}
    --}}}
    flyback := horizontal.cells - AW
    HBP := flyback >> 1
    HFP := (flyback - HBP) >> 1
    HS := HBP - HFP
    --}}}
    --{{{ calculate vertical timing
    SL := INT (line.frequency / (INT32 frame.frequency))
    IF
        (INT height) > 1024
        AL := 1024
        TRUE
        AL := (INT height)
    --{{{ COMMENT test numbers
    --:::A 0 0
    --{{{ test numbers
    IF
        (INT AL) > (SL - (SL/5)) -- 80%
        STOP -- Image is set too tall for frame rate
        TRUE
        SKIP
    --}}}
    --}}}
    flyback := SL - (INT height)
    VFP := 3
    VS := 3
    VBP := flyback - 6
    --}}}
    --{{{ send video timing
    writeCRTC(CommandFIFO, CTCReset)

```

```
IF
  interlace
    writeCRTC(ParameterFIFO, #1B)
  TRUE
    writeCRTC(ParameterFIFO, #12)
  writeCRTC(ParameterFIFO, ((AW - 2) /\ #FE))
  writeCRTC(ParameterFIFO, (HS - 1) \/ ((VS /\ 7) << 5))
  writeCRTC(ParameterFIFO, ((VS /\ #18) >> 3) \/ ((HFP -1) << 2))
  writeCRTC(ParameterFIFO, (HBP - 1) /\ #3F)
  writeCRTC(ParameterFIFO, VFP /\ #3F)
  writeCRTC(ParameterFIFO, AL /\ #FF)
  writeCRTC(ParameterFIFO, ((AL /\ #0300) >> 8) \/ (VBP << 2))
  --}}}
  --{{{ unblank display & select mode
  writeCRTC(CommandFIFO, CTCRCtrl)
  ChannelModeSelect := 1
  --}}}
:
--}}}
--{{{ locals
INT16 width, height, frame.frequency:
INT32 line.frequency, pixel.clock:
BOOL interlace, running:
INT16 channel, pixel, red, green, blue, table:
--}}}
SEQ
  --{{{ command interpreter
  running := TRUE
  init.G170( INT channel.A, 0 )
  init.G170( INT channel.B, 0 )
  init.G170( INT channel.C, 0 )
  WHILE running
    command ? CASE
      crt.c.init: width; height; line.frequency;
        frame.frequency; pixel.clock; interlace
        set.timing( width, height, frame.frequency,
          line.frequency, pixel.clock, interlace )
      crt.c.color; channel; pixel; red; green; blue
        set.colour((INT channel), (INT pixel),
          (INT red), (INT green), (INT blue))
      crt.c.initLUT; channel; table
        init.G170((INT channel), (INT table))
      crt.c.stop
        running := FALSE
    --}}}
:
--}}}
--}}}
```

4.1.3. B409Stub.occ

```
PROC B409.stub ( CHAN OF ANY in, out )
```

```
    #INCLUDE "crtc.inc"
```

```
    #USE "graphics.lib"
```

```
    B409( in )
```

```
:
```

4.1.4. Background.occ

```

PROC Background ( CHAN OF ANY fromTarget, toSP,
                  fromPrev, toPrev, fromNext, toNext,
                  VAL INT position )

    #INCLUDE "s_header.inc"
    REAL32 g.scale :
    --{{{ constants
    VAL packet.length IS 16 :
    VAL num.packets IS (128 * 8) / packet.length :

    VAL min.signal IS 0 :
    VAL max.signal IS 65535 :
    --}}}
    --{{{ ProcessRow
    PROC ProcessRow ( [packet.length] INT data,
                     [packet.length] REAL32 back.row, gain.row, offset.row )

        [] REAL32 target RETYPES data :
        SEQ
            SEQ i = 0 FOR packet.length
            INT digital :
            SEQ
                digital := INT TRUNC( (((target[i] + back.row[i]) *
                                         gain.row[i]) + offset.row[i]) * g.scale )
            IF
                digital < min.signal
                    data[i] := min.signal
                digital > max.signal
                    data[i] := max.signal
            TRUE
                data[i] := digital
        :
    --}}}
    --{{{ ProcessFrame
    PROC ProcessFrame ( CHAN OF ANY in, out,
                       [128][8] REAL32 Background, Gain, Offset )

        --{{{ retype array to packet.length
        [num.packets][packet.length] REAL32 p.background RETYPES Background :
        [num.packets][packet.length] REAL32 p.gain RETYPES Gain :
        [num.packets][packet.length] REAL32 p.offset RETYPES Offset :
        --}}}
        --{{{ variables
        INT in.ptr, out.ptr, process.ptr, temp :
        [3][packet.length] INT buffer :
        PLACE buffer IN WORKSPACE :
        --}}}
        SEQ
            in.ptr := 2
            process.ptr := 1
            out.ptr := 0
            --{{{ get first row
            in ? buffer[0]
            --}}}
            --{{{ get second row and process first row
            PRI PAR
                in ? buffer[1]
                ProcessRow( buffer[0], p.background[1], p.gain[1], p.offset[1] )
            --}}}
            --{{{ do middle rows
            SEQ row = 1 FOR (num.packets-2)
            SEQ
                PRI PAR
                    PAR

```

```

        in ? buffer[in.ptr]
        out ! buffer[out.ptr]
        ProcessRow( buffer[process.ptr], p.background[row],
                    p.gain[row], p.offset[row] )
        temp := out.ptr
        out.ptr := process.ptr
        process.ptr := in.ptr
        in.ptr := temp
    --}}}
    --{{{ process last row
    VAL i IS num.packets - 1 :
    PRI PAR
        out ! buffer[out.ptr]
        ProcessRow( buffer[process.ptr], p.background[i],
                    p.gain[i], p.offset[i] )
    --}}}
    --{{{ output last row
    out ! buffer[ process.ptr ]
    --}}}
:
--}}}
--{{{ CalibrationFrame
PROC CalibrationFrame ( CHAN OF ANY out, VAL REAL32 level,
                        [128][8] REAL32 Gain, Offset )

--{{{ retype array to packet.length
[num.packets][packet.length] REAL32 p.gain      RETYPES Gain      :
[num.packets][packet.length] REAL32 p.offset     RETYPES Offset   :
--}}}
--{{{ variables
INT out.ptr :
[2][packet.length] INT buffer :
PLACE buffer IN WORKSPACE :
[packet.length] REAL32 b.row :
[packet.length] INT t.row :
--}}}
SEQ
    out.ptr := 0
    --{{{ initialize
    VAL INT i.level RETYPES level :
    SEQ i = 0 FOR packet.length
        SEQ
            b.row[i] := 0.0 (REAL32)
            t.row[i] := i.level
        --}}}
    --{{{ process first row
    SEQ
        buffer[0] := t.row
        ProcessRow( buffer[0], b.row, p.gain[1], p.offset[1] )
    --}}}
    --{{{ do middle rows
    SEQ row = 1 FOR (num.packets-1)
        SEQ
            PRI PAR
                out ! buffer[out.ptr]
            SEQ
                buffer[1-out.ptr] := t.row
                ProcessRow( buffer[1-out.ptr], b.row,
                            p.gain[row], p.offset[row] )
            out.ptr := 1 - out.ptr
        --}}}
    --{{{ output last row
    out ! buffer[ out.ptr ]
    --}}}
:
--}}}
--{{{ SelectRow

```

```

PROC SelectRow ( [8] REAL32 dest, [128] REAL32 source )

    SEQ i = 0 FOR 8
        dest[i] := source[ (i*16) + position ]
    :
    --}}}
    --{{{ variables
    [max.frames][128][8] REAL32 Background :
    [128][8] REAL32 Gain, Offset :

    BYTE length :
    [max.message] INT message :
    command IS message[0] :
    params IS [message FROM 1 FOR (max.message-1)] :
    {} REAL32 r.params RETYPES params :
    --}}}
    SEQ
        --{{{ initialize last background frame
        SEQ i = 0 FOR 128
            SEQ j = 0 FOR 8
                Background[max.frames-1][i][j] := 0.0008234782 (REAL32)
            --}}}
        WHILE TRUE
            SEQ
                --{{{ get command and pass on
                fromPrev ? length::message
                IF
                    position < 15
                        toNext ! length::message
                TRUE
                    SKIP
                --}}}
            CASE command
                --{{{ c.set.background
                c.set.background
                ProcessFrame( fromTarget, toSP, Background[ params[0] ],
                    Gain, Offset )
                --}}}
                --{{{ c.background.row
                c.background.row
                SelectRow( Background[ params[0] ][ params[1] ], [r.params FROM 2 FOR
128] )
                --}}}
                --{{{ c.gain.row
                c.gain.row
                SelectRow( Gain[ params[0] ], [r.params FROM 1 FOR 128] )
                --}}}
                --{{{ c.offset.row
                c.offset.row
                SelectRow( Offset[ params[0] ], [r.params FROM 1 FOR 128] )
                --}}}
                --{{{ c.global.scale
                c.global.scale
                g.scale := r.params[0]
                --}}}
                --{{{ c.test.background
                c.test.background
                SEQ
                    SEQ i = 0 FOR 128
                        SEQ j = 0 FOR 8
                            VAL col IS (j << 4) + position :
                            SEQ
                                Gain[i][j] := 0.80008787 (REAL32) -
                                    ((REAL32 ROUND col) / 254.3456 (REAL32))
                                Offset[i][j] := 0.0008723984 (REAL32) +
                                    ((REAL32 ROUND i) * 19.789 (REAL32))
                                Background[max.frames-1][i][j] := 0.0008234782 (REAL32)
                            :
                        :
                    :
                :
            :
        :
    :

```

```
--}}}  
--((( c.calibration.frame  
c.calibration.frame  
    CalibrationFrame( toSP, r.params[0], Gain, Offset )  
--}})
```

:

4.1.5. Controller.occ

```
PROC Controller ( CHAN OF ANY fromHost, toHost, fromGTSEI, toGTSEI,
                  fromBG, toBG, fromSP, toSP )
```

```
PRI PAR
  --{{{ make processing a high priority process
  #INCLUDE "s_header.inc"
  --{{{ variables
  --{{{ command variables
  BYTE length :
  [max.message] INT message :
  command IS message[0] :
  params IS {message FROM 1 FOR (max.message-1)} :
  [] REAL32 r.params RETYPES params :
  --}}}

  [max.sim.frames] REAL32 frame.rate, frame.time, frame.range :
  [max.sim.frames] INT ticks :
  [max.sim.frames][p.length] REAL32 position :

  INT frames.loaded, start, value : -- temporary variables
  INT current.frame, increment :
  INT offset, col, row :
  BOOL test.mode :
  INT calibration.frame, num.cal.frames :
  INT shift.row, shift.col :
  [10] REAL32 calibration.level :
  [10] INT sp.cal.level :

  VAL seconds.per.tick IS 1.0E-6(REAL32) :
  TIMER clock :

  --{{{ force some scalars in vector space
  [3] INT frame.array :
  sim.frame IS frame.array[0] :
  first.frame IS frame.array[1] :
  last.frame IS frame.array[2]:
  --}}}
  --}}}
  SEQ
    --{{{ initialize
    current.frame := max.frames - 1
    increment := 1
    test.mode := TRUE
    calibration.frame := 10
    shift.row := 0
    shift.col := 0
    --}}}
    WHILE TRUE
      SEQ
        --{{{ get command
        fromHost ? length::message
        --}}}
        --{{{ process command
        IF
          --{{{ GTSEI and Target commands
          (command >= 256) AND (command < 768)
          toGTSEI ! length::message
          --}}}
          --{{{ Background commands
          (command >= 768) AND (command < 1024)
          toBG ! length::message
          --}}}
          --{{{ Guidance commands
          (command >= 1280) AND (command < 1536)
```

```

        toSP ! length::message
    --}}}
    --{{{ c.read.graphics
    command = c.read.graphics
    INT bufLength :
    INT number.of.transfers :
    [maxGraphicBuffer]BYTE graphicsBuffer :
    SEQ
        toSP ! length::message
        fromSP ? number.of.transfers
        toHost ! number.of.transfers
        SEQ i = 0 FOR number.of.transfers
        SEQ
            fromSP ? bufLength::graphicsBuffer
            toHost ! bufLength::graphicsBuffer
    --}}}
    --{{{ c.frame.start
    command = c.start.frame
    SEQ
        IF
            params[0] < 0
            SKIP
            params[0] = 0
            sim.frame := params[1]
            TRUE
            sim.frame := first.frame + params[1]
        --{{{ sim.frame := MAX( 0, MIN( last.frame, sim.frame ) )
        IF
            sim.frame < 0
            sim.frame := 0
            sim.frame > last.frame
            sim.frame := last.frame
            TRUE
            SKIP
        --}}}
        increment := params[2]
        --calibration.frame := num.cal.frames
        test.mode := FALSE
    --}}}
    --{{{ c.run.single
    command = c.run.single
    SEQ
        IF
            calibration.frame < num.cal.frames
            --{{{ send calibration frame
            SEQ
                toBG ! BYTE 2; c.calibration.frame; calibration.level[
calibration.frame ]
                toSP ! BYTE 9; c.sp.frame; calibration.frame;
                    sp.cal.level[ calibration.frame ]; 0; 65535;
                    0.0(REAL32); 0.0(REAL32); -1; 64.0(REAL32)
                calibration.frame := calibration.frame + 1
            --}}}
            sim.frame < first.frame
            --{{{ send next non-FPA frame
            SEQ
                toSP ! 12(BYTE); c.guidance.run; 0; frame.range[ sim.frame
];
                    frame.time[ sim.frame ]; 0; 0; position[ sim.frame ]
                toSP ! 6(BYTE); c.display.info; 0; frame.range[ sim.frame ];
                    frame.time[ sim.frame ]; 0; 0

            IF
                test.mode
                frame.time[ sim.frame ] := frame.time[ sim.frame ] +
(1.0(REAL32) /
                    frame.rate[ sim.frame ])

```

```

sim.frame < last.frame
sim.frame := sim.frame + increment
TRUE
SKIP
--}}}
TRUE
--{{{ send next FPA frame
SEQ
current.frame := sim.frame - first.frame

offset := ((row /\ 3) << 2) + (col /\ 3)
toGTSEI ! BYTE 2; c.set.crossbar; (col >> 2) /\ 15
toGTSEI ! BYTE 5; c.set.target; current.frame; offset;

(row>>2); (col>>2)

toBG ! BYTE 2; c.set.background; current.frame
toSP ! BYTE 15; c.sp.frame; -1; 0; 2500; 65535;
frame.range[ sim.frame ]; frame.time[
sim.frame ];
current.frame+1; frame.rate[ sim.frame ];
position[ sim.frame]
IF
test.mode
--{{{ update statistics
SEQ
frame.time[ sim.frame ] := frame.time[ sim.frame ] +
(1.0(REAL32) /
frame.rate[ sim.frame ])
frame.range[ sim.frame ] := frame.range[ sim.frame ] -
(10000.0(REAL32) /
frame.rate[ sim.frame ])
--}}}
sim.frame < last.frame
sim.frame := sim.frame + increment
TRUE
SKIP
--}}}
--{{{ process any guidance commands
VAL delay.ticks IS INT ROUND (0.05 (REAL32) / seconds.per.tick)
:
INT time.now, interrupt.time :
INT command :
BOOL exit :
SEQ
clock ? time.now
interrupt.time := time.now + delay.ticks

exit := FALSE
WHILE NOT exit
PRI ALT
clock ? AFTER interrupt.time
exit := TRUE
fromSP ? command
--{{{ process command
CASE command
cc.shift.image
INT shift.col, shift.row :
SEQ
fromSP ? shift.col; shift.row
col := (col + shift.col) /\ 511
row := (row + shift.row) /\ 511
ELSE
SKIP
--}}}
--}}}
--{{{ c.run.continuous
command = c.run.continuous

```

```

VAL delay.ticks IS INT ROUND( 5.0E-4(REAL32) / seconds.per.tick ) :
INT last.start.time, next.start.time, interrupt.time :
INT command :
BOOL running, exit :
SEQ
  --{{{ check sending calibration frames
  WHILE calibration.frame < num.cal.frames
    --{{{ send calibration frame
    SEQ
      toBG ! BYTE 2; c.calibration.frame; calibration.level[
calibration.frame ]
      toSP ! BYTE 9; c.sp.frame; calibration.frame;
        sp.cal.level[ calibration.frame ]; 0; 65535;
        0.0(REAL32); 0.0(REAL32); -1; 64.0(REAL32)
      calibration.frame := calibration.frame + 1
    --}}}
  --}}}
  --{{{ initialize
  clock ? last.start.time
  interrupt.time := last.start.time
  next.start.time := interrupt.time + delay.ticks
  --}}}
  running := TRUE
  WHILE running
    SEQ
      --{{{ send current frame
      IF
        sim.frame < first.frame
        --{{{ send next non-FPA frame ** this code should be
obsolete **
          SEQ
            --{{{ wait for correct time
            INT current.time :
            VAL wait.ticks IS INT ROUND( 1.0E-4(REAL32) /
seconds.per.tick ) :
            SEQ
              clock ? current.time
              IF
                (next.start.time MINUS current.time) > wait.ticks
                clock ? AFTER next.start.time
              TRUE
              SKIP
            --}}}
            toSP ! 12(BYTE); c.guidance.run; 0; frame.range[
sim.frame ];
              frame.time[ sim.frame ]; 0; 0; position[
sim.frame ]
              toSP ! 6(BYTE); c.display.info; 0; frame.range[
sim.frame ];
              frame.time[ sim.frame ]; 0; 0
            --}}}
          TRUE
          --{{{ send out FPA frame
          SEQ
            current.frame := sim.frame - first.frame
            --{{{ send to SP and wait for response
            -- this code has been added to handle the guidance
commands that
has
work as before,
            -- will occur when running with the PFP. If no guidance
            -- been selected, the shift values should return 0 and
            -- otherwise, the PFP will drive the simulation
          SEQ

```

```

toSP ! BYTE 9; c.sp.frame; -1; 0; 2500; 65535;
    frame.range[ sim.frame ]; frame.time[
sim.frame ];
        current.frame+1; frame.rate[ sim.frame
]
        --position[ sim.frame ]

--}}}
--{{{ wait for response from SP -- syncs to correct
time
fromSP ? command ; col; row
col := (col + shift.col) /\ 511
row := (row + shift.row) /\ 511
--}}}
previous values
offset := ((row /\ 3) << 2) + (col /\ 3) -- uses

toGTSEI ! BYTE 2; c.set.crossbar; (col >> 2) /\ 15
toGTSEI ! BYTE 5; c.set.target; current.frame; offset;
(row>>2); (col>>2)

--{{{ wait for correct time
INT current.time :
VAL wait.ticks IS INT ROUND( 1.0E-4(REAL32) /
seconds.per.tick ) :
SEQ
    clock ? current.time
    IF
        (next.start.time MINUS current.time) > wait.ticks
        SKIP
        --clock ? AFTER next.start.time
        TRUE
        SKIP
    --}}}

--{{{ old code for sending to the SP
-- toSP ! BYTE 15; c.sp.frame; -1; 0; 2500; 65535;
--     frame.range[ sim.frame ]; frame.time[
sim.frame ];
--         current.frame+1; frame.rate[
sim.frame ];
--             position[ sim.frame ]
--}}}
toBG ! BYTE 2; c.set.background; current.frame

--}}}
--{{{ move to next frame
IF
    test.mode
    --{{{ update statistics
    SEQ
        frame.time[ sim.frame ] := frame.time[ sim.frame ] +
(1.0(REAL32) /
                                frame.rate[ sim.frame ])
        frame.range[ sim.frame ] := frame.range[ sim.frame ] -
(10000.0(REAL32) /
frame.rate[ sim.frame ])
    --}}}
    sim.frame < last.frame
    sim.frame := sim.frame + increment
    TRUE
    SKIP
--}}}
--{{{ update for next frame
last.start.time := next.start.time
--clock ? last.start.time

```

```

next.start.time := last.start.time PLUS ticks[ sim.frame ]
interrupt.time := next.start.time MINUS delay.ticks
--}}}
--}}}
exit := FALSE
WHILE NOT exit
  PRI ALT
    --clock ? AFTER interrupt.time
    fromHost ? command
    --{{{ process command
    CASE command
      cc.shift.image
      INT temp.shift.col, temp.shift.row :
      SEQ
        fromHost ? temp.shift.col; temp.shift.row
        --col := (col + shift.col) /\ 511
        --row := (row + shift.row) /\ 511
        shift.row := shift.row + temp.shift.row
        shift.col := shift.col + temp.shift.col
      cc.exit
      SEQ
        exit := TRUE
        running := FALSE
      ELSE
        SKIP
    --}}}
  TRUE & SKIP
  exit := TRUE
  --fromSP ? command
  --{{{ process command ** this code should be obsolete
  **
  --CASE command
  -- cc.shift.image
  -- INT shift.col, shift.row :
  -- SEQ
  --   fromSP ? shift.col; shift.row
  --   col := (col + shift.col) /\ 511
  --   row := (row + shift.row) /\ 511
  -- ELSE
  --   SKIP
  --}}}
--}}}
--{{{ c.frame.rate
command = c.frame.rate
SEQ
  start := params[0]
  frames.loaded := params[1]
  [frame.rate FROM start FOR frames.loaded] :=
    [r.params FROM 2 FOR frames.loaded]
--}}}
--{{{ c.frame.time
command = c.frame.time
SEQ
  start := params[0]
  frames.loaded := params[1]
  [frame.time FROM start FOR frames.loaded] :=
    [r.params FROM 2 FOR
frames.loaded]
--}}}
--{{{ c.frame.range
command = c.frame.range
SEQ
  start := params[0]
  frames.loaded := params[1]
  [frame.range FROM start FOR frames.loaded] :=
    [r.params FROM 2 FOR
frames.loaded]

```

```
--}}
--{{{ c.sim.position
command = c.sim.position
  INT ptr :
  SEQ
    value := params[0]
    start := params[1]
    frames.loaded := params[2]

    ptr := 3
    SEQ i = start FOR frames.loaded
      SEQ
        position[i][value] := r.params[ptr]
        ptr := ptr + 1
--}}}
--{{{ c.sim.start.frames
command = c.sim.start.frames
  SEQ
    first.frame := params[0]
    last.frame := params[1]

    IF
      (first.frame + 1) < last.frame
      SEQ
        SEQ i = 0 FOR last.frame
          ticks[i] := INT ROUND( (frame.time[i+1] - frame.time[i]) /
                                seconds.per.tick )
          ticks[ last.frame ] := ticks[ last.frame-1 ]
      TRUE
      SKIP
--}}}
--{{{ c.test.controller
command = c.test.controller
  SEQ
    sim.frame := first.frame + (max.frames - 1)
    test.mode := TRUE
    row := 0
    col := 0
    ticks[ sim.frame ] := INT ROUND( ( 1.0(REAL32) / 64.0(REAL32) ) /
                                    seconds.per.tick )
    frame.range[ sim.frame ] := 100000.0 (REAL32)
    frame.time[ sim.frame ] := 0.0 (REAL32)
    frame.rate[ sim.frame ] := 64.0 (REAL32)
--}}}
--{{{ c.restart
command = c.restart
  SEQ
    calibration.frame := 0
    test.mode := FALSE
    sim.frame := 0
    row := 0
    col := 0
    shift.row := 0
    shift.col := 0
--}}}
--{{{ c.set.calibration
command = c.set.calibration
  SEQ
    num.cal.frames := params[0]
    [calibration.level FROM 0 FOR num.cal.frames] :=
      [r.params FROM 1 FOR num.cal.frames]
    [sp.cal.level FROM 0 FOR num.cal.frames] :=
      [params FROM 1+num.cal.frames FOR num.cal.frames]
--}}}
--{{{ else SKIP
TRUE
SKIP
```

```
        --}}}  
    --}}}  
--}}}  
SKIP  
:
```

4.1.6. FirstBuffer.occ

```

PROC FirstBuffer ( CHAN OF ANY in, out, fromNext, toPrev,
                  VAL INT position, shift )

  --{{{ variables
  [2][64] INT input.buffer :
  INT count :
  --}}}
  --{{{ channels
  CHAN OF ANY synch0, synch1, internal :
  --}}}
  --{{{ Receiver
  PROC Receiver ( CHAN OF ANY in, out0, out1, [2][64] INT buffer )

    INT i :
    SEQ
      i := 0
      WHILE TRUE
        SEQ
          in ? buffer[i]
          out0 ! i
          out1 ! i
          i := 1 - i
        :
      --}}}
  --{{{ Sender
  PROC Sender ( CHAN OF ANY in, out, [2][64] INT buffer )

    INT i :
    SEQ
      WHILE TRUE
        SEQ
          in ? i
          out ! buffer[i]
        :
      --}}}
  --{{{ Extractor
  PROC Extractor ( CHAN OF ANY internal, in, out,
                  VAL INT count )

    --{{{ variables
    [2][64][2] BYTE buffer :
    INT output :
    --}}}
    SEQ
      internal ? buffer[0]
      output := 0
      WHILE TRUE
        SEQ
          SEQ i = 0 FOR count
          SEQ
            PAR
              out ! buffer[output]
              in ? buffer[ 1-output ]
            output := 1 - output
          PAR
              out ! buffer[output]
              internal ? buffer[1-output]
            output := 1 - output
          :
        --}}}
  --{{{ Formatter
  PROC Formatter ( CHAN OF ANY synch, out,
                  [2][64] INT input.buffer )

```

```

--{{{ variables
[2][64][4] BYTE b.in RETYPES input.buffer :
[64][2] BYTE buffer :
[64*2] BYTE buffer1 RETYPES buffer :
INT in.ptr :
--}}}
SEQ
  WHILE TRUE
    SEQ
      --{{{ form message in buffer
      SEQ
        synch ? in.ptr

        source IS input.buffer[in.ptr] :
        INT p :
        SEQ
          p := 0
          SEQ i = 0 FOR 64
            INT store :
            SEQ
              store := source[i] >> shift
              --{{{ check for zeroing store
              IF
                store = 0
                IF
                  source[i] <> 0
                  store := 1
                  TRUE
                  SKIP
                TRUE
                SKIP
              --}}}
              buffer1[p] := BYTE store
              buffer1[p+1] := BYTE store
              p := p + 2
            --}}}
          out ! buffer
        :
      --}}}
    SEQ
      IF
        position < 8
        count := 7 - position
        TRUE
        count := 15 - position
      PRI PAR
        PAR
          Receiver( in, synch0, synch1, input.buffer )
          Sender( synch0, out, input.buffer )
          Extractor( internal, fromNext, toPrev, count )
          Formatter( synch1, internal, input.buffer )
        :

```

4.1.7. Formatter.occ

```

PROC Formatter ( CHAN OF ANY in, out )

  --{{{ constants
  VAL buffer.size IS 64*16 :
  --}}}
  --{{{ variables
  [2][buffer.size] BYTE input.buffer, output.buffer :
  [8] INT store.offset :
  --}}}
  --{{{ channels
  CHAN OF ANY synch0, synch1 :
  --}}}
  --{{{ Receiver
  PROC Receiver ( CHAN OF ANY in, out, [2][buffer.size] BYTE buffer )

    INT i :
    SEQ
      i := 0
      WHILE TRUE
        SEQ
          in ? buffer[i]
          out ! i
          i := 1 - i
        :
      --}}}
  --}}}
  PROC Formatter ( CHAN OF ANY in, out,
    [2][buffer.size] BYTE in.buffer, out.buffer )

    INT out.ptr, in.ptr :
    SEQ
      out.ptr := 0
      WHILE TRUE
        SEQ
          in ? in.ptr
          --{{{ format
          [8][64][2] BYTE inb RETYPES in.buffer[in.ptr] :
          [64][16] BYTE outb RETYPES out.buffer[out.ptr] :
          SEQ
            SEQ i = 0 FOR 8
              source IS inb[i] :
              VAL start IS i << 1 :
              MOVE2D( source, 0, 0, outb, start, 0, 2, 64 )
            --}}}
          out ! out.ptr
          out.ptr := 1 - out.ptr
        :
      --}}}
  --}}}
  --{{{ Sender
  PROC Sender ( CHAN OF ANY in, out, [2][buffer.size] BYTE buffer )

    INT i :
    SEQ
      WHILE TRUE
        SEQ
          in ? i
          out ! buffer[i]
        :
      --}}}
  --}}}
  SEQ
    WHILE TRUE
      PRI PAR
        PAR
          Receiver( in, synch0, input.buffer )

```

```
        Sender( synch1, out, output.buffer )  
    Formatter( synch0, synch1, input.buffer, output.buffer )  
    :
```

4.1.8. G_Line.occ

```

--{{{ SC line
--:::A 3 10
--{{{ line
--{{{ libraries
#include "g_header.inc"
--}}}
--{{{ plot
PROC plot ( VAL [] INT window, [] BYTE screen,
            VAL INT x, y, VAL BYTE color )

    -- plots a single point on the screen
    -- makes sure the pixels are actually in the window
    VAL pixels.line IS window[ w.pixels.line ] :
    VAL size.x      IS window[ w.size.x ] :
    VAL size.y      IS window[ w.size.y ] :
    SEQ
    IF
        (x < 0) OR (y < 0) OR (x >= size.x) OR (y >= size.y)
        SKIP
    TRUE
        screen[ (y * pixels.line) + x ] := color
:
--}}}
--{{{ draw.line
PROC draw.line ( VAL [] INT window, [] BYTE screen,
                VAL INT x1, y1, x2, y2,
                VAL BYTE color )

--{{{ clip.line
PROC clip.line (VAL INT x1, y1, x2, y2, INT result, VAL []INT window)
    -- decides whether a line is totally on or off screen/window
    --{{{ codes
    VAL code.centre IS #00 :      -- 0000
    VAL code.left   IS #01 :      -- 0001
    VAL code.right  IS #02 :      -- 0010
    VAL code.bottom IS #04 :      -- 0100
    VAL code.top    IS #08 :      -- 1000
    --}}}
    INT code1, code2 :
    --{{{ PROC check
    PROC check (VAL INT x, y, INT code)
        VAL x.max IS window[ w.size.x ] :
        VAL y.max IS window[ w.size.y ] :
        SEQ
        IF
            --{{{ x.min <= x < x.max
            (x >= 0) AND (x < x.max)
            code := code.centre
            --}}}
            --{{{ x < x.min
            x < 0
            code := code.left
            --}}}
            --{{{ x > x.max
            TRUE --x >= x.max
            code := code.right
            --}}}
        IF
            --{{{ y.min <= y < y.max
            (y >= 0) AND (y < y.max)
            SKIP
            --}}}
            --{{{ y < y.min
            y < 0

```

```

        code := code \/ code.top
    --}}}
    --{{{ y >= y.max
    TRUE --y > y.max
        code := code \/ code.bottom
    --}}}
:
--}}}
SEQ
    check (x1, y1, code1)
    check (x2, y2, code2)
    IF
        --{{{ line lies entirely within window
        (code1 \/ code2) = 0
            result := in.range
        --}}}
        --{{{ line lies entirely outside window
        (code1 /\ code2) <> 0
            result := not.inrange
        --}}}
        --{{{ partially in window perhaps
        TRUE
            result := part.inrange
        --}}}
:
--}}}
--{{{ slow.draw.line
PROC slow.draw.line ( VAL [] INT window, [] BYTE screen,
                     VAL INT x1, y1, x2, y2,
                     VAL BYTE color )
    -- uses Bresenham's integer algorithm to calculate plotting points
    -- calls plot to draw actual pixels on the screen
    VAL pixels.line IS window[w.pixels.line] :
    INT dx, dy :
    INT two.dx, two.dy :
    INT error :
    SEQ
        dx := x2 - x1
        dy := y2 - y1
        IF
            (dx <> 0) OR (dy <> 0)
            SEQ
                --{{{ a line to draw
                IF
                    --{{{ dy = 0 -- horizontal line
                    dy = 0
                        SEQ i = x1 FOR dx + 1
                            plot (window, screen, i, y1, color)
                        --}}}
                    --{{{ dx = 0 -- vertical line
                    dx = 0
                        SEQ
                            IF
                                dy > 0
                                    SEQ i = y1 FOR dy + 1
                                        plot (window, screen, x1, i, color)
                                    TRUE
                                        SEQ i = y2 FOR (-dy) + 1
                                            plot (window, screen, x1, i, color)
                                --}}}
                    --{{{ dx <> 0 dy <> 0 -- diagonal line
                    TRUE
                        INT x, y :
                        INT delta.y :
                        SEQ
                            x := x1
                            y := y1

```

```

two.dx := dx + dx
IF
  --{{{ dy > 0
  dy > 0
  SEQ
    two.dy := dy + dy
    IF
      --{{{ dy > dx
      dy > dx
      SEQ
        error := two.dx - dy
        --{{{ plot line
        SEQ i = 0 FOR dy + 1
        SEQ
          plot (window, screen, x, y, color )
          IF
            error >= 0
            SEQ
              x := x + 1
              error := error - two.dy
            TRUE
            SKIP
            y := y + 1
            error := error + two.dx
          --}}}
        --}}}
      --}}}
    --{{{ dy <= dx
    TRUE
    SEQ
      error := two.dy - dx
      --{{{ plot line
      SEQ i = 0 FOR dx + 1
      SEQ
        plot (window, screen, x, y, color )
        IF
          error >= 0
          SEQ
            y := y + 1
            error := error - two.dx
          TRUE
          SKIP
            x := x + 1
            error := error + two.dy
          --}}}
        --}}}
      --}}}
    --}}}
  --{{{ dy < 0
  TRUE
  SEQ
    dy := -dy
    two.dy := dy + dy
    IF
      --{{{ dy > dx
      dy > dx
      SEQ
        error := two.dx - dy
        --{{{ plot line
        SEQ i = 0 FOR dy + 1
        SEQ
          plot (window, screen, x, y, color)
          IF
            error >= 0
            SEQ
              x := x + 1
              error := error - two.dy
            TRUE
            SKIP

```

```

        y := y - 1
        error := error + two.dx
    --}}}
--}}}
--{{{ dy <= dx
TRUE
SEQ
    error := two.dy - dx
    SEQ i = 0 FOR dx + 1
    SEQ
        plot (window, screen, x, y, color)
        IF
            error >= 0
            SEQ
                y := y - 1
                error := error - two.dx
            TRUE
            SKIP
            x := x + 1
            error := error + two.dy
        --}}}
    --}}}
--}}}
--}}}
TRUE
    plot (window, screen, x1, y1, color)
:
--}}}
--{{{ fast.draw.line
PROC fast.draw.line (VAL [] INT window, [] BYTE screen,
    VAL INT x1, y1, x2, y2,
    VAL BYTE color)

-- uses Bresenham's integer algorithm to calculate plotting points
-- points are in increasing values of x
-- only called when the line is known to be on screen / in window and
-- the current pixel size is one
INT dx, dy, two.dx, two.dy, delta.x, delta.y :
INT error, pixel :
VAL pixels.line IS window[ w.pixels.line ] :
SEQ
    dx := x2 - x1                -- always zero or positive
    dy := y2 - y1
    pixel := (y1 * pixels.line) + x1
    IF
        (dx <> 0) OR (dy <> 0)
        SEQ
            --{{{ a line to draw
            IF
                --{{{ dy = 0                -- horizontal line
                dy = 0
                SEQ i = pixel FOR dx + 1
                screen[i] := color
            --}}}
            --{{{ dx = 0                -- vertical line
            dx = 0
            SEQ
                IF
                    dy > 0
                    SEQ i = 0 FOR dy + 1
                    SEQ
                        screen[pixel] := color
                        pixel := pixel + pixels.line
                TRUE
                SEQ i = 0 FOR (-dy) + 1
                SEQ
                    screen[pixel] := color

```

```
        pixel := pixel - pixels.line
--}}}
--{{{ dx <> 0 AND dy <> 0
TRUE
  INT delta.y :
  SEQ
    two.dx := dx + dx
    IF
      dy > 0
        delta.y := pixels.line
      TRUE
        SEQ
          dy := -dy
          delta.y := -pixels.line
    two.dy := dy + dy
    IF
      --{{{ dy > dx
      dy > dx
      SEQ
        error := two.dx - dy
        --{{{ plot line
        SEQ i = 0 FOR dy + 1
        SEQ
          screen[pixel] := color
          IF
            error >= 0
            SEQ
              pixel := pixel + 1
              error := error - two.dy
          TRUE
            SKIP
          pixel := pixel + delta.y
          error := error + two.dx
        --}}}
      --}}}
    --}}}
    --{{{ dy <= dx
    TRUE
      SEQ
        error := two.dy - dx
        --{{{ plot line
        SEQ i = 0 FOR dx + 1
        SEQ
          screen[pixel] := color
          IF
            error >= 0
            SEQ
              pixel := pixel + delta.y
              error := error - two.dx
          TRUE
            SKIP
          pixel := pixel + 1
          error := error + two.dy
        --}}}
      --}}}
    --}}}
  TRUE
    screen[pixel] := color
:
--}}}
INT x3, y3, x4, y4 :
INT result :
SEQ
  --{{{ swap x1,y1 with x2,y2 if x1 > x2
  IF
    x1 > x2
    SEQ
```

```

        x3 := x2
        y3 := y2
        x4 := x1
        y4 := y1
    TRUE
    SEQ
        x3 := x1
        y3 := y1
        x4 := x2
        y4 := y2
    --}}}
clip.line (x3, y3, x4, y4, result, window)
IF
    (result = in.range)
        fast.draw.line (window, screen, x3, y3, x4, y4, color)
    (result = part.inrange) OR (result = in.range)
        slow.draw.line (window, screen, x3, y3, x4, y4, color)
    TRUE
    SKIP
:
--}}}
--{{{ draw.polyline
PROC draw.polyline ( VAL [] INT window, [] BYTE screen,
                    VAL [] [2]INT points, VAL BYTE color)

    -- calls draw line to draw the lines
    INT x, y :
    SEQ
        x := points[0][0]
        y := points[0][1]
        SEQ i = 1 FOR (SIZE points) - 1
            VAL point IS points[i] :
            SEQ
                draw.line( window, screen, x, y, point[0], point[1], color )
                x := point[0]
                y := point[1]
        :
    --}}}
--{{{ draw.rectangle
PROC draw.rectangle ( VAL [] INT window, [] BYTE screen,
                    VAL [2][2]INT p, VAL BYTE color)

    -- calls draw line to draw the lines
    INT x, y :
    SEQ
        draw.line( window, screen, p[0][0], p[0][1], p[1][0], p[0][1], color )
        draw.line( window, screen, p[1][0], p[0][1], p[1][0], p[1][1], color )
        draw.line( window, screen, p[1][0], p[1][1], p[0][0], p[1][1], color )
        draw.line( window, screen, p[0][0], p[1][1], p[0][0], p[0][1], color )
    :
    --}}}
--}}}
--}}}

```

4.1.9. G_System.occ

```
--{{{ SC system
--:::A 3 10
--{{{ system
--{{{ libraries
#include "crtc.inc"
#include "g_header.inc"
--}}}
--{{{ set.colour
PROC set.colour ( CHAN OF CRTC message,
                  VAL INT channel, pixel, red, green, blue )
  -- set up a colour in the G170 colour look up table
  SEQ
    message ! crtc.color; INT16 channel; INT16 pixel;
              INT16 red; INT16 green; INT16 blue
  :
--}}}
--{{{ set.timing
PROC set.timing( CHAN OF CRTC message,
                 VAL INT width,height, line.frequency, frame.rate, pixel.clock,
                 VAL BOOL interlace )

  SEQ
    message ! crtc.init; INT16 width; INT16 height; INT32 line.frequency;
              INT16 frame.rate; INT32 pixel.clock; interlace
  :
--}}}
--{{{ set.B408
PROC set.B408( VAL INT DS, IE, EM, OE, R )

  --{{{ system constants
  VAL bpw.shift IS 2 :
  VAL mint      IS MOSTNEG INT :

  VAL DisplayStart.address IS (#00000000 >< mint) >> bpw.shift :
  VAL InterlaceEnable.address IS (#000C0000 >< mint) >> bpw.shift :
  VAL EventMode.address IS (#00100000 >< mint) >> bpw.shift :
  VAL OutputEnable.address IS (#00140000 >< mint) >> bpw.shift :
  VAL Ready.address IS (#00040000 >< mint) >> bpw.shift :

  INT DisplayStart, InterlaceEnable, EventMode, OutputEnable, Ready :

  PLACE DisplayStart AT DisplayStart.address :
  PLACE InterlaceEnable AT InterlaceEnable.address :
  PLACE EventMode AT EventMode.address :
  PLACE OutputEnable AT OutputEnable.address :
  PLACE Ready AT Ready.address :
  --}}}
  SEQ
    DisplayStart := DS
    InterlaceEnable := IE
    EventMode := EM
    OutputEnable := OE
    Ready := R
  :
--}}}
--{{{ init.G170
PROC init.G170 (CHAN OF CRTC message, VAL INT channel, table)

  SEQ
    message ! crtc.initLUT; INT16 channel; INT16 table
  :
--}}}
--{{{ clear.window
PROC clear.window (VAL [] INT window, [] BYTE screen)
```

```
VAL size.x      IS window[ w.size.x ] :
VAL size.y      IS window[ w.size.y ] :
VAL pixels.line IS window[ w.pixels.line ] :
VAL b.color     IS BYTE window[ w.background.color ] :
INT ptr :
SEQ
  SEQ i = 0 FOR size.x
    screen[i] := b.color
  ptr := pixels.line
  SEQ i = 0 FOR size.y - 1
    SEQ
      [screen FROM ptr FOR size.x] := [screen FROM 0 FOR size.x]
      ptr := ptr + pixels.line
:
--}}}
--}}}
--}}}
```

4.1.10. G_Text.occ

```

--{{{ SC text
--:::A 3 10
--{{{ text
#include "g_header.inc"
--{{{ FUNCTION GetINT
INT FUNCTION GetINT (VAL INT pointer, VAL [] INT table)
  INT return :
  VAL [] BYTE b.table RETYPES table :
  VALOF
    [4]BYTE return.b RETYPES return :
  SEQ
    return.b[0] := b.table[pointer]
    return.b[1] := b.table[pointer + 1]
    return.b[2] := b.table[pointer + 2]
    return.b[3] := b.table[pointer + 3]
  RESULT return
:
--}}})
--{{{ FUNCTION GetINT16
INT16 FUNCTION GetINT16 (VAL INT pointer, VAL [] INT table)
  INT16 return :
  VAL [] BYTE b.table RETYPES table :
  VALOF
    [2]BYTE return.b RETYPES return :
  SEQ
    return.b[0] := b.table[pointer]
    return.b[1] := b.table[pointer + 1]
  RESULT return
:
--}}})
--{{{ FUNCTION GetBYTE
BYTE FUNCTION GetBYTE (VAL INT pointer, VAL [] INT table)
  BYTE return :
  VAL [] BYTE b.table RETYPES table :
  VALOF
    SEQ
      return := b.table[pointer]
  RESULT return
:
--}}})
--{{{ get.font.spec
PROC get.font.spec ( VAL [] INT font, [fs.size] INT spec)

  SEQ
    spec[ fs.PixWidth  ] := INT (GetINT16 (dfPixWidth.p, font))
    spec[ fs.PixHeight ] := INT (GetINT16 (dfPixHeight.p, font))
    spec[ fs.FirstChar ] := INT (GetBYTE  (dfFirstChar.p, font))
    spec[ fs.LastChar  ] := INT (GetBYTE  (dfLastChar.p, font))
    spec[ fs.BitsOffset ] :=      GetINT  (dfBitsOffset.p, font)
  :
--}}})
--{{{ scroll
PROC scroll ( VAL [] INT window, []BYTE screen,
             VAL INT jump.size )
  -- scrolls a screen or window by the required number of lines (jump.size)
  VAL size.x IS window[ w.size.x ] :
  VAL size.y IS window[ w.size.y ] :
  VAL pixels.line IS window[ w.pixels.line ] :
  VAL b.color IS BYTE window[ w.background.color ] :
  INT p1, p2 :
  IF
    (jump.size > 0) AND (jump.size < size.y)
    --{{{ scroll screen
    SEQ

```

```

        p1 := 0
        p2 := pixels.line * jump.size
        SEQ i = 0 FOR (size.y - jump.size)
            SEQ
                [screen FROM p1 FOR size.x] := [screen FROM p2 FOR size.x]
                p1 := p1 + pixels.line
                p2 := p2 + pixels.line
            SEQ i = 0 FOR size.x
                screen[ p1 + i ] := b.color
            p2 := p1 + pixels.line
            SEQ i = 0 FOR (jump.size - 1)
                SEQ
                    [screen FROM p2 FOR size.x] := [screen FROM p1 FOR size.x]
                    p2 := p2 + pixels.line
            --}}}
        jump.size > 0
        --{{{ clear screen
        SEQ

            SEQ i = 0 FOR size.x
                screen[ i ] := b.color
            p2 := pixels.line
            SEQ i = 0 FOR (jump.size - 1)
                SEQ
                    [screen FROM p2 FOR size.x] := [screen FROM 0 FOR size.x]
                    p2 := p2 + pixels.line
            --}}}
        TRUE
        SKIP
    :
    --}}}
    --{{{ draw.char v2.0
PROC draw.char ( [ ] INT window, [ ] BYTE screen,
                VAL BYTE char,
                VAL [ ] INT font, VAL [fs.size] INT spec )

    --{{{ constants
    VAL mask IS 1 << 7 :
    pixels.line IS window[ w.pixels.line ] :
    size.x      IS window[ w.size.x ] :
    size.y      IS window[ w.size.y ] :
    cursor.x    IS window[ w.cursor.x ] :
    cursor.y    IS window[ w.cursor.y ] :

    VAL [ ] BYTE b.font RETYPES font :
    --}}}
    --{{{ variables
    INT bit, pixel :
    INT bitmask :
    INT char.width, offset, PixWidthBytes :
    INT char.spacing :
    INT character :
    --}}}
    --{{{ line feed
PROC line.feed ( )

    SEQ
        cursor.y := cursor.y + spec[ fs.PixHeight ]
        IF
            (cursor.y + spec[ fs.PixHeight ]) < size.y
                SKIP
        TRUE
            INT scroll.lines :
            SEQ
                scroll.lines := (spec[ fs.PixHeight ] - (size.y - cursor.y)) + 1
                cursor.y := (size.y - spec[ fs.PixHeight ]) - 1
                scroll( window, screen, scroll.lines )

```

```

:
--}}}
SEQ
  character := INT char
  IF
    char = '*n'
      line.feed ()
    char = '*c'
      --{{{ carriage return
      cursor.x := 0
      --}}}
  (character >= spec[ fs.FirstChar ]) AND (character <= spec[ fs.LastChar ])
  SEQ
    character := character - spec[ fs.FirstChar ]
    --{{{ set font data
    IF
      spec[ fs.PixWidth ] <> 0
      SEQ
        PixWidthBytes := (spec[ fs.PixWidth ] + 7) >> 3
        char.width := spec[ fs.PixWidth ]
        offset := (character * (PixWidthBytes * spec[ fs.PixHeight ]
          ))) +
          spec[ fs.BitsOffset ]
        char.spacing := char.width
      TRUE
      -- Variable Width
      INT char.width.p, char.pointer.p :
      SEQ
        char.width.p := CharTable.p + (character << 2)
        char.pointer.p := char.width.p + 2
        char.width := INT(GetINT16(char.width.p, font))
        offset := INT(GetINT16(char.pointer.p, font))
        PixWidthBytes := (char.width + 7) >> 3
        char.spacing := char.width + 1
      --}}}
    IF
      --{{{ char too big
      (char.width > size.x) OR (spec[ fs.PixHeight ] > size.y)
      SKIP
      --}}}
      --{{{ room to draw char
      BOOL delayed.crlf :
      TRUE
      SEQ
        delayed.crlf := FALSE
      IF
        --{{{ room to draw whole char
        ((cursor.x + char.spacing) < size.x) AND
          ((cursor.y + spec[ fs.PixHeight ]) < size.y)
        SKIP
        --}}}
        --{{{ room to draw but at end of line
        ((cursor.x + char.spacing) = size.x) AND
          ((cursor.y + spec[ fs.PixHeight ]) < size.y)
        delayed.crlf := TRUE
        --}}}
        --{{{ we need carriage return - line feed
        TRUE
        SEQ
          cursor.x := 0
          line.feed ()
          --}}}
        pixel := (cursor.y TIMES pixels.line) + cursor.x
        --{{{ plot foreground only
        VAL f.color IS BYTE window[ w.foreground.color ] :
        SEQ
          SEQ i = 0 FOR spec[ fs.PixHeight ]
          SEQ

```

```

--{{{ draw row
SEQ j = 0 FOR PixWidthBytes
  SEQ
    bitmask := mask
    VAL this.byte IS INT b.font[ offset+((spec[fs.PixHeight]
TIMES j) + i) ] :
    SEQ k = 0 FOR 8
      SEQ
        bit := this.byte /\ bitmask
        IF
          --{{{ leave background as it is
          bit = 0
          SKIP
          --}}}
          --{{{ plot foreground bit
          TRUE
          screen[ pixel ] := f.color
          --}}}
          pixel := pixel + 1
          bitmask := bitmask >> 1
        --}}}
      pixel := (pixel - (PixWidthBytes<<3)) + pixels.line

    cursor.x := cursor.x + char.spacing
  --}}}
  IF
    delayed.crlf
    --{{{ we need carriage return - line feed
    SEQ
      cursor.x := 0
      line.feed ()
    --}}}
    TRUE
    SKIP
  --}}}
  TRUE
  SKIP
:
--}}}
--{{{ write.string v2.0
PROC write.string ( [] INT window, [] BYTE screen,
  VAL [] BYTE string, VAL [] INT font )

  [fs.size] INT spec :
  SEQ
    get.font.spec( font, spec )
    SEQ i = 0 FOR (SIZE string)
      draw.char( window, screen, string[i], font, spec )
  :
  --}}}
  --{{{ string.width
PROC string.width ( VAL [] INT font, VAL [] BYTE string, INT width )

  [fs.size] INT spec :
  SEQ
    get.font.spec( font, spec )
    width := 0
    SEQ i = 0 FOR SIZE string
      --{{{ add width for character[i]
      INT character :
      SEQ
        character := INT string[i]
        IF
          (character >= spec[ fs.FirstChar ]) AND (character <= spec[ fs.LastChar]
)
          SEQ
            character := character - spec[ fs.FirstChar ]

```

```

--{{{ determine width from font
IF
    spec[ fs.PixWidth ] <> 0                                -- Fixed Width
    width := width + spec[ fs.PixWidth ]
    TRUE                                                    -- Variable
Width
    INT char.width.p, char.pointer.p :
    SEQ
        char.width.p := CharTable.p + (character << 2)
        width := (width + 1) + (INT(GetINT16(char.width.p, font)))
    --}}}
    TRUE
    SKIP
--}}}
:
--}}}
--}}}
--}}}

```

4.1.11. GIF.occ

```

#include "hostio.inc"
#USE "hostio.lib"

PROC Encode (CHAN OF SP fs, ts, VAL INT32 GIFFile,
             VAL INT MaxColumn, MaxRow, BitsPerPixel,
             VAL []INT Palette, VAL [][]INT Pixels)
--{{{ GIF Encoder
PROC Encoder (VAL [][]INT pixels, CHAN OF INT out,
             CHAN OF INT size, VAL INT bitsPerPixel)

--{{{ putCode
PROC putCode (CHAN OF INT out, VAL INT value)
    out ! value
    :
--}}}

VAL max.table.size IS (1 << 13) :

--{{{ Clear
PROC Clear (VAL INT bits.per.pixel, INT CodeSize, NextValidCode,
           MaxCode, [max.table.size] INT Child, Sibling)

SEQ
    CodeSize := bits.per.pixel + 1
    NextValidCode := (1 << bits.per.pixel) + 2
    MaxCode := 1 << (bits.per.pixel + 1)
    SEQ I = 0 FOR max.table.size
        SEQ
            Child [I] := 0
            Sibling [I] := 0
    :
--}}}

VAL pixel.rows IS (SIZE pixels) :
[max.table.size] INT child, sibling, shade :
INT codeSize, clearCode, endCode,
    minCode, maxCode, nextValidCode,
    color, son, parent, maxColor, pixCol, pixRow, pixelColumnsM1 :

SEQ
    maxColor := (1 << bitsPerPixel) - 1
    color := 0

--{{{ Initialize
SEQ
    SEQ i = 0 FOR max.table.size
        SEQ
            child [i] := 0
            sibling [i] := 0
            codeSize := bitsPerPixel + 1
            clearCode := 1 << bitsPerPixel
            endCode := clearCode + 1
            nextValidCode := endCode + 1
            maxCode := clearCode << 1
    --}}}

SEQ
    putCode (out, clearCode)
    size ! codeSize
    IF
        (0 < pixel.rows)
            SEQ
                pixelColumnsM1 := SIZE pixels [0]
            IF

```

```

(0 < pixelColumnsM1)
SEQ
  color := pixels {0}{0}
  pixelColumnsM1 := pixelColumnsM1 -1
  IF
    (1 < pixelColumnsM1)
    SEQ
      pixRow := 0
      pixCol := 1
    TRUE
    SEQ
      pixRow := 1
      pixCol := 0
      pixelColumnsM1 := (SIZE pixels {1}) - 1
    TRUE
    color := maxColor + 2
  TRUE
  color := maxColor + 2
parent := color
WHILE (color <= maxColor)
--{{{ Compress
SEQ
  IF
    (pixRow < pixel.rows)
    SEQ
      color := pixels [pixRow][pixCol]
      IF
        (pixCol < pixelColumnsM1)
        pixCol := pixCol + 1
      TRUE
      SEQ
        pixRow := pixRow + 1
        pixCol := 0
        pixelColumnsM1 := (SIZE pixels [pixRow]) - 1
      TRUE
      color := maxColor + 2
    son := child [parent]
    IF
      son <= 0
      --{{{ Parent has no son
      SEQ
        child [parent] := nextValidCode
        shade [nextValidCode] := color
        putCode (out, parent)
        size ! codeSize
        parent := color
        nextValidCode := nextValidCode + 1
      --}}}
    TRUE
    --{{{ Otherwise
    SEQ
      IF
        shade [son] = color
        parent := son -- make new parent
      TRUE
      --{{{ son not right color
      BOOL looping :
      INT brother :
      SEQ
        brother := son
        looping := TRUE
      WHILE looping
      SEQ
        IF
          sibling [brother] > 0
          --{{{ Brother has brother
          SEQ

```

```

        brother := sibling [brother]
        IF
            shade [brother] = color
            SEQ
                looping := FALSE
                parent := brother
            TRUE
            SKIP
        --}}}
    TRUE
    --{{{ No brother, so create one
    SEQ
        looping := FALSE
        sibling [brother] := nextValidCode
        shade [nextValidCode] := color
        putCode (out, parent)
        size ! codeSize
        parent := color
        nextValidCode := nextValidCode + 1
    --}}}

    --}}}
    --}}}
    --{{{ Change code size if required
    IF
        nextValidCode > maxCode
        IF
            codeSize < 12
            SEQ
                codeSize := codeSize + 1
                maxCode := maxCode << 1
            TRUE
            SEQ
                putCode (out, clearCode)
                size ! codeSize
                Clear (bitsPerPixel, codeSize, nextValidCode,
                    maxCode, child, sibling)
            TRUE
            SKIP
        --}}}
    --}}}
    putCode (out, endCode)
    size ! codeSize
:
--}}}

--{{{ so.fwrite.INT16
--Writes 2-byte integer, LSB first
PROC so.fwrite.INT16 (CHAN OF SP fs, ts, VAL INT32 StreamID,
    VAL INT16 Value, BYTE Result)

    VAL msb IS #FF00 (INT16) :
    VAL lsb IS #00FF (INT16) :

    BYTE Result2 :

    SEQ
        so.fwrite.char (fs, ts, StreamID, BYTE (Value /\ lsb), Result)
        so.fwrite.char (fs, ts, StreamID, BYTE ((Value /\ msb) >> 8), Result2)
        Result := BYTE ((INT Result) \/ (INT Result2))
    :
    --}}}

    --{{{ WriteBlock
    PROC WriteBlock (CHAN OF SP fs, ts, VAL INT32 StreamID,
        [255] BYTE Block, INT Length, BYTE Result)
        BYTE Result2 :

```

```

SEQ
  so.fwrite.char (fs, ts, StreamID, BYTE Length, Result)
  SEQ I = 0 FOR Length
    SEQ
      so.fwrite.char (fs, ts, StreamID, Block [I], Result2)
      Result := BYTE ((INT Result) \/ (INT Result2))
    Length := 0
  :
  --}}

--{{{ BlockByte
PROC BlockByte (CHAN OF SP fs, ts, VAL INT32 StreamID,
  [255] BYTE Block, INT Index, VAL BYTE Data, BYTE Result)

SEQ
  Block [Index] := Data
  Index := Index + 1
  IF
    Index = 255
      WriteBlock (fs, ts, StreamID, Block, Index, Result)
    TRUE
      SKIP
  :
  --}}}

--{{{ Output
--Output process from Encoder
PROC Output (CHAN OF INT FromEncoder, Size, VAL INT32 GIFFile,
  VAL INT MaxColumn, MaxRow, BitsPerPixel,
  VAL []INT Palette
  )
  --{{{ Constants
  VAL byte.mask IS #FF (INT32) :
  VAL size.of.int IS 4 :
  VAL colors IS (1 << BitsPerPixel) :
  VAL max.byte IS 40 :
  VAL depth IS 2 :
  VAL max.gray IS (1 << BitsPerPixel) :
  VAL ppw IS ((size.of.int * 8) / BitsPerPixel) :
  VAL wpsl IS (MaxColumn / ppw) :
  VAL red IS 0 :
  VAL green IS 1 :
  VAL blue IS 2 :

  VAL gif.signature IS "GIF87a" :
  VAL global.color.map IS #80 :
  VAL color.res IS ((depth - 1) << 4) :
  VAL bits IS (BitsPerPixel + 1) :
  VAL screen.height IS (INT16 MaxRow) :
  VAL screen.left IS 0 (INT16) :
  VAL screen.top IS 0 (INT16) :
  VAL screen.width IS (INT16 MaxColumn) :
  VAL screen.descriptor IS
    (BYTE ((global.color.map \/ color.res) \/
      (BitsPerPixel - 1))) :

  VAL background IS 0 (BYTE) :
  VAL endCode IS ((1 << BitsPerPixel) + 1) :
  --}}}

  BYTE Result :
  INT CodeSize, OutByte, Shift, Value :
  INT32 Out :
  [3] BYTE ColorValue :
  [255] BYTE OutBlock :

  SEQ
    OutByte := 0
    --{{{ GIF Signature

```

```

so.fwrite.string (fs, ts, GIFFFile, gif.signature, Result)
--}}}

--{{{ Screen Descriptor
so.fwrite.INT16 (fs, ts, GIFFFile, screen.width, Result)
so.fwrite.INT16 (fs, ts, GIFFFile, screen.height, Result)
so.fwrite.char (fs, ts, GIFFFile, screen.descriptor, Result)
so.fwrite.char (fs, ts, GIFFFile, background, Result)
so.fwrite.char (fs, ts, GIFFFile, 0 (BYTE), Result)
--}}}

--{{{ Global Color Map
VAL PaletteColors IS (SIZE Palette) :
SEQ I = 0 FOR colors
  SEQ
    IF
      (I < PaletteColors)
        SEQ
          ColorValue [blue] :=
            (BYTE ((INT32 Palette [I]) /\ byte.mask))
          ColorValue [green] :=
            (BYTE ((INT32 Palette [I]) >> 8) /\ byte.mask))
          ColorValue [red] :=
            (BYTE ((INT32 Palette [I]) >> 16) /\ byte.mask))
        TRUE
        SEQ
          ColorValue [blue] := 0 (BYTE)
          ColorValue [green] := 0 (BYTE)
          ColorValue [red] := 0 (BYTE)
        SEQ J = 0 FOR 3
          so.fwrite.char (fs, ts, GIFFFile, ColorValue [J], Result)
        --}}}
      --}}}

--{{{ Image Descriptor
so.fwrite.char (fs, ts, GIFFFile, ',', Result)
so.fwrite.INT16 (fs, ts, GIFFFile, screen.left, Result)
so.fwrite.INT16 (fs, ts, GIFFFile, screen.top, Result)
so.fwrite.INT16 (fs, ts, GIFFFile, screen.width, Result)
so.fwrite.INT16 (fs, ts, GIFFFile, screen.height, Result)
so.fwrite.char (fs, ts, GIFFFile, 0 (BYTE), Result)
--}}}

--{{{ Raster Data
--First byte is bits per image pixel
so.fwrite.char (fs, ts, GIFFFile, BYTE BitsPerPixel, Result)

--{{{ Get first code
FromEncoder ? Value
Size ? CodeSize
Shift := CodeSize
Out := (INT32 Value)
--}}}

--{{{ Accept and package codes until end of image
WHILE (Value <> endCode)
  SEQ
    --{{{ Write any finished bytes
    WHILE (Shift > 8)
      SEQ
        BlockByte (fs, ts, GIFFFile, OutBlock, OutByte,
          BYTE (Out /\ byte.mask), Result)
        Out := Out >> 8
        Shift := Shift - 8
      --}}}
    --{{{ Add next code

```

```

    FromEncoder ? Value
    Size ? CodeSize
    Out := Out \/ ((INT32 Value) << Shift)
    Shift := Shift + CodeSize
    --}}}
--}}}

--{{{ Output remaining codes
WHILE (Shift > 0)
    SEQ
        BlockByte (fs, ts, GIFFile, OutBlock, OutByte,
                    BYTE (Out /\ byte.mask), Result)
        Out := Out >> 8
        Shift := Shift - 8
    IF
        OutByte <> 0
        WriteBlock (fs, ts, GIFFile, OutBlock, OutByte, Result)
    TRUE
    SKIP

--Raster data terminates with 0-byte block
WriteBlock (fs, ts, GIFFile, OutBlock, OutByte, Result)
--}}}
--}}}

--{{{ GIF Terminator
so.fwrite.char (fs, ts, GIFFile, ';', Result)
--}}}

:
--}}}

CHAN OF INT FromEncoder :
CHAN OF INT Size :

PAR
    Encoder (Pixels, FromEncoder, Size, BitsPerPixel)
    Output (FromEncoder, Size, GIFFile,
            MaxColumn, MaxRow, BitsPerPixel, Palette)
:

```

4.1.12. GIF02.occ

```

#include "hostio.inc"
#USE "hostio.lib"

PROC Encode (CHAN OF SP fs, ts, VAL INT32 GIFFile,
             VAL INT MaxColumn, MaxRow, BitsPerPixel,
             VAL []INT Palette, VAL [][]INT Pixels)
--{{{ GIF Encoder
PROC Encoder (CHAN OF INT in, out, CHAN OF INT size, VAL INT bitsPerPixel)

--{{{ putCode
PROC putCode (CHAN OF INT out, VAL INT value)
    out ! value
:
--}}}

VAL max.table.size IS (1 << 13) :

--{{{ Clear
PROC Clear (VAL INT bits.per.pixel, INT CodeSize, NextValidCode,
            MaxCode, [max.table.size] INT Child, Sibling)

SEQ
    CodeSize := bits.per.pixel + 1
    NextValidCode := (1 << bits.per.pixel) + 2
    MaxCode := 1 << (bits.per.pixel + 1)
    SEQ I = 0 FOR max.table.size
        SEQ
            Child [I] := 0
            Sibling [I] := 0
:
--}}}

[max.table.size] INT child, sibling, shade :
INT codeSize, clearCode, endCode,
    minCode, maxCode, nextValidCode,
    color, son, parent, maxColor :

SEQ
    maxColor := (1 << bitsPerPixel) - 1
    color := 0

--{{{ Initialize
SEQ
    SEQ i = 0 FOR max.table.size
        SEQ
            child [i] := 0
            sibling [i] := 0
            codeSize := bitsPerPixel + 1
            clearCode := 1 << bitsPerPixel
            endCode := clearCode + 1
            nextValidCode := endCode + 1
            maxCode := clearCode << 1
    --}}}

SEQ
    putCode (out, clearCode)
    size ! codeSize
    in ? color
    parent := color
    WHILE (color <= maxColor)
        --{{{ Compress
        SEQ
            in ? color
            son := child [parent]

```

```
IF
  son <= 0
  --{{{ Parent has no son
  SEQ
    child [parent] := nextValidCode
    shade [nextValidCode] := color
    putCode (out, parent)
    size ! codeSize
    parent := color
    nextValidCode := nextValidCode + 1
  --}}})
TRUE
  --{{{ Otherwise
  SEQ
    IF
      shade [son] = color
      parent := son -- make new parent
    TRUE
      --{{{ son not right color
      BOOL looping :
      INT brother :
      SEQ
        brother := son
        looping := TRUE
        WHILE looping
          SEQ
            IF
              sibling [brother] > 0
              --{{{ Brother has brother
              SEQ
                brother := sibling [brother]
                IF
                  shade [brother] = color
                  SEQ
                    looping := FALSE
                    parent := brother
                  TRUE
                    SKIP
              --}}})
            TRUE
              --{{{ No brother, so create one
              SEQ
                looping := FALSE
                sibling [brother] := nextValidCode
                shade [nextValidCode] := color
                putCode (out, parent)
                size ! codeSize
                parent := color
                nextValidCode := nextValidCode + 1
              --}}})
          --}}})
      --}}})
  --{{{ Change code size if required
  IF
    nextValidCode > maxCode
    IF
      codeSize < 12
      SEQ
        codeSize := codeSize + 1
        maxCode := maxCode << 1
      TRUE
      SEQ
        putCode (out, clearCode)
        size ! codeSize
        Clear (bitsPerPixel, codeSize, nextValidCode,
              maxCode, child, sibling)
    TRUE
```

```

        SKIP
        --}}}
        --}}}
        putCode (out, endCode)
        size ! codeSize
:
--}}}

--{{{ so.fwrite.INT16
--Writes 2-byte integer, LSB first
PROC so.fwrite.INT16 (CHAN OF SP fs, ts, VAL INT32 StreamID,
                    VAL INT16 Value, BYTE Result)

    VAL msb IS #FF00 (INT16) :
    VAL lsb IS #00FF (INT16) :

    BYTE Result2 :

    SEQ
        so.fwrite.char (fs, ts, StreamID, BYTE (Value /\ lsb), Result)
        so.fwrite.char (fs, ts, StreamID, BYTE ((Value /\ msb) >> 8), Result2)
        Result := BYTE ((INT Result) \/ (INT Result2))
:
--}}}

--{{{ WriteBlock
PROC WriteBlock (CHAN OF SP fs, ts, VAL INT32 StreamID,
                [255] BYTE Block, INT Length, BYTE Result)

    BYTE Result2 :

    SEQ
        so.fwrite.char (fs, ts, StreamID, BYTE Length, Result)
        SEQ I = 0 FOR Length
            SEQ
                so.fwrite.char (fs, ts, StreamID, Block [I], Result2)
                Result := BYTE ((INT Result) \/ (INT Result2))
            Length := 0
:
--}}}

--{{{ BlockByte
PROC BlockByte (CHAN OF SP fs, ts, VAL INT32 StreamID,
                [255] BYTE Block, INT Index, VAL BYTE Data, BYTE Result)

    SEQ
        Block [Index] := Data
        Index := Index + 1
        IF
            Index = 255
                WriteBlock (fs, ts, StreamID, Block, Index, Result)
        TRUE
            SKIP
:
--}}}

--{{{ Input
--Input process for Encoder
PROC Input (CHAN OF INT ToEncoder, VAL [][] IN Pixels, VAL INT End)
    SEQ
        SEQ I = 0 FOR (SIZE Pixels)
            SEQ J = 0 FOR (SIZE Pixels [I])
                ToEncoder ! Pixels [I][J]
            ToEncoder ! End
:
--}}}

--{{{ Output

```

```
--Output process from Encoder
PROC Output (CHAN OF INT FromEncoder, Size, VAL INT32 GIFFile,
            VAL INT MaxColumn, MaxRow, BitsPerPixel,
            VAL []INT Palette
            )
--{{{ Constants
VAL byte.mask IS #FF (INT32) :
VAL size.of.int IS 4 :
VAL colors IS (1 << BitsPerPixel) :
VAL max.byte IS 40 :
VAL depth IS 2 :
VAL max.gray IS (1 << BitsPerPixel) :
VAL ppw IS ((size.of.int * 8) / BitsPerPixel) :
VAL wpsl IS (MaxColumn / ppw) :
VAL red IS 0 :
VAL green IS 1 :
VAL blue IS 2 :

VAL gif.signature IS "GIF87a" :
VAL global.color.map IS #80 :
VAL color.res IS ((depth - 1) << 4) :
VAL bits IS (BitsPerPixel + 1) :
VAL screen.height IS (INT16 MaxRow) :
VAL screen.left IS 0 (INT16) :
VAL screen.top IS 0 (INT16) :
VAL screen.width IS (INT16 MaxColumn) :
VAL screen.descriptor IS
    (BYTE ((global.color.map \/ color.res) \/
           (BitsPerPixel - 1))) :
VAL background IS 0 (BYTE) :
VAL endCode IS ((1 << BitsPerPixel) + 1) :
--}}}

BYTE Result :
INT CodeSize, OutByte, Shift, Value :
INT32 Out :
[3] BYTE ColorValue :
[255] BYTE OutBlock :

SEQ
    OutByte := 0
    --{{{ GIF Signature
    so.fwrite.string (fs, ts, GIFFile, gif.signature, Result)
    --}}}

    --{{{ Screen Descriptor
    so.fwrite.INT16 (fs, ts, GIFFile, screen.width, Result)
    so.fwrite.INT16 (fs, ts, GIFFile, screen.height, Result)
    so.fwrite.char (fs, ts, GIFFile, screen.descriptor, Result)
    so.fwrite.char (fs, ts, GIFFile, background, Result)
    so.fwrite.char (fs, ts, GIFFile, 0 (BYTE), Result)
    --}}}

    --{{{ Global Color Map
    VAL PaletteColors IS (SIZE Palette) :
    SEQ I = 0 FOR colors
        SEQ
            IF
                (I < PaletteColors)
                SEQ
                    ColorValue [blue] :=
                        (BYTE ((INT32 Palette [I]) /\ byte.mask))
                    ColorValue [green] :=
                        (BYTE (((INT32 Palette [I]) >> 8) /\ byte.mask))
                    ColorValue [red] :=
                        (BYTE (((INT32 Palette [I]) >> 16) /\ byte.mask))
                TRUE
            SEQ
```

```

        ColorValue [blue] := 0 (BYTE)
        ColorValue [green] := 0 (BYTE)
        ColorValue [red] := 0 (BYTE)
    SEQ J = 0 FOR 3
        so.fwrite.char (fs, ts, GIFFFile, ColorValue [J], Result)
    --}}}

--{{{ Image Descriptor
so.fwrite.char (fs, ts, GIFFFile, ',', Result)
so.fwrite.INT16 (fs, ts, GIFFFile, screen.left, Result)
so.fwrite.INT16 (fs, ts, GIFFFile, screen.top, Result)
so.fwrite.INT16 (fs, ts, GIFFFile, screen.width, Result)
so.fwrite.INT16 (fs, ts, GIFFFile, screen.height, Result)
so.fwrite.char (fs, ts, GIFFFile, 0 (BYTE), Result)
--}}}

--{{{ Raster Data
--First byte is bits per image pixel
so.fwrite.char (fs, ts, GIFFFile, BYTE BitsPerPixel, Result)

--{{{ Get first code
FromEncoder ? Value
Size ? CodeSize
Shift := CodeSize
Out := (INT32 Value)
--}}}

--{{{ Accept and package codes until end of image
WHILE (Value <> endCode)
    SEQ
        --{{{ Write any finished bytes
        WHILE (Shift > 8)
            SEQ
                BlockByte (fs, ts, GIFFFile, OutBlock, OutByte,
                    BYTE (Out /\ byte.mask), Result)
                Out := Out >> 8
                Shift := Shift - 8
            --}}}

        --{{{ Add next code
        FromEncoder ? Value
        Size ? CodeSize
        Out := Out \/ ((INT32 Value) << Shift)
        Shift := Shift + CodeSize
        --}}}
    --}}}

--{{{ Output remaining codes
WHILE (Shift > 0)
    SEQ
        BlockByte (fs, ts, GIFFFile, OutBlock, OutByte,
            BYTE (Out /\ byte.mask), Result)
        Out := Out >> 8
        Shift := Shift - 8
    IF
        OutByte <> 0
            WriteBlock (fs, ts, GIFFFile, OutBlock, Out yte, Result)
        TRUE
        SKIP

--Raster data terminates with 0-byte block
WriteBlock (fs, ts, GIFFFile, OutBlock, OutByte, Result)
--}}}
--}}}

--{{{ GIF Terminator

```

```
        so.fwrite.char (fs, ts, GIFFile, ';', Result)
        --}}
:
--}}

VAL end IS ((1 << BitsPerPixel) + 1) :

CHAN OF INT ToEncoder, FromEncoder :
CHAN OF INT Size :

PAR
    Input (ToEncoder, Pixels, end)
    Encoder (ToEncoder, FromEncoder, Size, BitsPerPixel)
    Output (FromEncoder, Size, GIFFile,
            MaxColumn, MaxRow, BitsPerPixel, Palette)
:
```

4.1.13. GTSEI.occ

```
PROC GTSEI ( CHAN OF ANY fromController, toController, fromTarget, toTarget,
             toCrossbar0, toCrossbar1 )
```

```
  #INCLUDE "s_header.inc"
```

```
  --{{{ Table
```

```
  VAL Table IS [ [ 0, 4 ],
                 [ 2, 5 ],
                 [ 1, 6 ],
                 [ 7, 3 ],
                 [ 29, 31 ],
                 [ 30, 28 ],
                 [ 24, 25 ],
                 [ 27, 26 ],
                 [ 17, 19 ],
                 [ 23, 22 ],
                 [ 21, 16 ],
                 [ 20, 18 ],
                 [ 10, 8 ],
                 [ 13, 9 ],
                 [ 14, 12 ],
                 [ 11, 15 ] ] :
```

```
  --}}}
```

```
  --{{{ variables
```

```
  BYTE synch :
```

```
  BOOL target.wait :
```

```
  [16][3] BYTE Set0, Set1 :
```

```
  INT Current.Selection :
```

```
  INT Selection :
```

```
  PLACE Selection AT #800 :
```

```
  BYTE length :
```

```
  [max.message] INT32 message :
```

```
  command IS message[0] :
```

```
  params IS [message FROM 1 FOR (max.message-1)] :
```

```
  --}}}
```

```
  --{{{ DetermineSetting
```

```
  PROC DetermineSetting ( [16][3] BYTE Set0, Set1, VAL INT offset )
```

```
  SEQ
```

```
  SEQ i = 0 FOR 16
```

```
  VAL connection IS BYTE Table[ ((i + offset) /\ 15) ][ 0 ] :
```

```
  SEQ
```

```
  Set0[i][1] := connection
```

```
  Set1[i][2] := connection
```

```
  :
```

```
  --}}}
```

```
  SEQ
```

```
  --{{{ initialize
```

```
  SEQ i = 0 FOR 16
```

```
  SEQ
```

```
  Set0[i][0] := 0 (BYTE)
```

```
  Set0[i][1] := BYTE Table[i][0]
```

```
  Set0[i][2] := BYTE Table[i][1]
```

```
  Set1[i][0] := 0 (BYTE)
```

```
  Set1[i][1] := BYTE Table[i][1]
```

```
  Set1[i][2] := BYTE Table[i][0]
```

```
  toCrossbar0 ! 4 (BYTE); Set0; 3 (BYTE)
```

```
  toCrossbar1 ! 4 (BYTE); Set1; 3 (BYTE)
```

```
  --}}}
```

```
  target.wait := FALSE
```

```
  WHILE TRUE
```

```
  SEQ
```

```
  fromController ? length::message
```

```

CASE INT command
--({{ c.set.crossbar
c.set.crossbar
  DetermineSetting( Set0, Set1, INT params[0] )
--}})
--({{ c.set.target
c.set.target
  SEQ
    IF
      target.wait
      fromTarget ? synch
      TRUE
      SKIP
    PAR
      toCrossbar0 ! 4(BYTE); Set0; 3(BYTE)
      toCrossbar1 ! 4(BYTE); Set1; 3(BYTE)
      toTarget ! length::message
      target.wait := TRUE
--}})
--({{ c.target.row
c.target.row
  SEQ
    IF
      target.wait
      fromTarget ? synch
      TRUE
      SKIP
      target.wait := FALSE
      toTarget ! length::message
--}})

```

:

4.1.14. GTSPI.occ

```

PROC GTSPI (CHAN OF ANY in1, in2)
  --{{{ external memory and registers
  [16384]INT frame.buffer :
  PORT OF INT mclock.data :
  PORT OF INT mclock.control :
  PORT OF INT hclock.active :
  PORT OF INT hclock.blanking :
  PORT OF INT vclock.active :
  PORT OF INT vclock.blanking :
  PORT OF INT master.register :
  PORT OF INT control.register :
  --}}}
  --{{{ placements
  VAL base.address        IS #0800 :
  PLACE mclock.data       AT base.address :
  PLACE mclock.control    AT base.address + 1:
  PLACE hclock.active     AT base.address + 2 :
  PLACE hclock.blanking   AT base.address + 3 :
  PLACE vclock.active     AT base.address + 4 :
  PLACE vclock.blanking   AT base.address + 5 :
  PLACE master.register   AT base.address + 6 :
  PLACE control.register   AT base.address + 7 :
  PLACE frame.buffer      AT #4000 :
  --}}}
  VAL numRows IS 128 :
  VAL numCols IS 128 :

  --{{{ PROC talk.to.host
  PROC talk.to.host (CHAN OF ANY from.host)
    INT selection, scale:
    INT numCols :
    INT numRows :
    SEQ
      from.host ? selection ; scale
      --{{{ CASE selection
      CASE selection
        1
          mclock.data ! scale
        2
          SEQ
            hclock.active ! scale
            numCols := scale+1
        3
          hclock.blanking ! scale
        4
          SEQ
            vclock.active ! scale
            numRows := scale+1
        5
          vclock.blanking ! scale
        6
          SEQ i = 0 FOR 16384
            frame.buffer[i] := scale
        7
          INT row, col :
          SEQ
            from.host ? row; col
            frame.buffer[col + (row * numCols)] := scale
        8
          control.register ! scale    -- reset control is bit 0
        9
          mclock.control ! scale      -- clock enable is bit 0
      ELSE
        SKIP

```

```

--}}
:
--}}
--{{{ PROC reset ()
PROC reset ()
  TIMER clock :
  INT now :
  SEQ
    control.register ! 0 --reset
    clock ? now
    clock ? AFTER now PLUS 5
    control.register ! 1 -- clear the reset condition
:
--}}

--{{{ variables
[2][64]INT buffer :
TIMER clock :
INT sync, out.frame, in.frame :
BOOL first.frame :
CHAN OF INT EndOfFrame :
PLACE EndOfFrame AT #08 :
INT last.frame.time :
CHAN OF ANY internal :
--}}
--{{{ constants
VAL offset0 IS 0 :
VAL offset1 IS 16 :
VAL offset2 IS 32 :
VAL offset3 IS 48 :
VAL offset4 IS 64 :
VAL offset5 IS 80 :
VAL offset6 IS 96 :
VAL offset7 IS 112 :
--}}
PRI PAR
  --{{{ handle frames
  SEQ
    --{{{ initialize
    SEQ
      reset ()
      hclock.active ! numCols - 1
      hclock.blanking ! 1
      vclock.active ! numRows - 1
      vclock.blanking ! 1
      mclock.data ! 0 -- main clock divider
      mclock.control ! 0 -- disable clock
      master.register ! 1 -- upper frame buffer
      SEQ i = 0 FOR 16384
        frame.buffer[i] := 0
      master.register ! 0 -- lower frame buffer
      SEQ i = 0 FOR 16384
        frame.buffer[i] := 0
      reset ()

      first.frame := TRUE
      in.frame := 0
      out.frame := 0
    --}}}
    --{{{ read calibration frames
    SEQ i = 0 FOR 2
      SEQ j = 0 FOR 16
        --SEQ k = 0 FOR 9
        SEQ k = 0 FOR 16
          --{{{ input
          PAR
            in1 ? buffer[0]

```

```

        in2 ? buffer[1]
        --}}
    --}}
WHILE TRUE
    INT base1, base2 :
    SEQ
        base1 := 0
        base2 := 8
        SEQ i = 0 FOR 127
            SEQ
                PAR
                    SEQ
                        --{{{ in1
                        in1 ? [frame.buffer FROM base1 + offset0 FOR 8]
                        in1 ? [frame.buffer FROM base1 + offset1 FOR 8]
                        in1 ? [frame.buffer FROM base1 + offset2 FOR 8]
                        in1 ? [frame.buffer FROM base1 + offset3 FOR 8]
                        in1 ? [frame.buffer FROM base1 + offset4 FOR 8]
                        in1 ? [frame.buffer FROM base1 + offset5 FOR 8]
                        in1 ? [frame.buffer FROM base1 + offset6 FOR 8]
                        in1 ? [frame.buffer FROM base1 + offset7 FOR 8]
                        --}}}
                    SEQ
                        --{{{ in2
                        in2 ? [frame.buffer FROM base2 + offset0 FOR 8]
                        in2 ? [frame.buffer FROM base2 + offset1 FOR 8]
                        in2 ? [frame.buffer FROM base2 + offset2 FOR 8]
                        in2 ? [frame.buffer FROM base2 + offset3 FOR 8]
                        in2 ? [frame.buffer FROM base2 + offset4 FOR 8]
                        in2 ? [frame.buffer FROM base2 + offset5 FOR 8]
                        in2 ? [frame.buffer FROM base2 + offset6 FOR 8]
                        [8]INT buff :
                        SEQ
                            --in2 ? [frame.buffer FROM base2 + offset7 FOR 8]
                            in2 ? buff
                            [frame.buffer FROM base2 + offset7 FOR 7] := [buff FROM 0 FOR
7]
                        --}}}
                        mclock.control ! 1 -- enable clock
                        base1 := base1 + 128
                        base2 := base2 + 128
                        --{{{ throwaway last line
                        PAR
                            in1 ? buffer[0]
                            in2 ? buffer[1]
                        --}}}
                        --IF
                        -- first.frame
                        -- first.frame := FALSE
                        -- TRUE
                        -- --EndOfFrame ? sync
                        -- --clock ? AFTER (last.frame.time PLUS 400)
                        --internal ! 1
                        --out.frame := in.frame
                        --in.frame := 1 - in.frame
                        --master.register ! in.frame /\ (out.frame << 1)
                        --}}}
                        --{{{ handle syncs
                        INT start :
                        INT last.frame.time :
                        VAL frame.time IS 390 :
                        SEQ
                            internal ? start
                            clock ? last.frame.time
                            WHILE TRUE
                                SEQ
                                    last.frame.time := last.frame.time PLUS frame.time

```

```
clock ? AFTER last.frame.time  
PRI ALT  
  internal ? start  
    SKIP  
  TRUE & SKIP  
    SKIP  
--}}}
```

:

4.1.15. GTSPI2.ocu

```

PROC GTSPI (CHAN OF ANY in1, in2)
  --{{{ external memory and registers
  [16384]INT frame.buffer :
  PORT OF INT mclock.data :
  PORT OF INT mclock.control :
  PORT OF INT hclock.active :
  PORT OF INT hclock.blanking :
  PORT OF INT vclock.active :
  PORT OF INT vclock.blanking :
  PORT OF INT master.register :
  PORT OF INT control.register :
  --}}}
  --{{{ placements
  VAL base.address      IS #0800 :
  PLACE mclock.data     AT base.address :
  PLACE mclock.control  AT base.address + 1:
  PLACE hclock.active   AT base.address + 2 :
  PLACE hclock.blanking AT base.address + 3 :
  PLACE vclock.active   AT base.address + 4 :
  PLACE vclock.blanking AT base.address + 5 :
  PLACE master.register AT base.address + 6 :
  PLACE control.register AT base.address + 7 :
  PLACE frame.buffer    AT #4000 :
  --}}}
  VAL numRows IS 128 :
  VAL numCols IS 128 :

  --{{{ PROC talk.to.host
  PROC talk.to.host (CHAN OF ANY from.host)
    INT selection, scale:
    INT numCols :
    INT numRows :
    SEQ
      from.host ? selection ; scale
      --{{{ CASE selection
      CASE selection
        1
          mclock.data ! scale
        2
          SEQ
            hclock.active ! scale
            numCols := scale+1
        3
          hclock.blanking ! scale
        4
          SEQ
            vclock.active ! scale
            numRows := scale+1
        5
          vclock.blanking ! scale
        6
          SEQ i = 0 FOR 16384
            frame.buffer[i] := scale
        7
          INT row, col :
          SEQ
            from.host ? row; col
            frame.buffer[col + (row * numCols)] := scale
        8
          control.register ! scale    -- reset control is bit 0
        9
          mclock.control ! scale      -- clock enable is bit 0
      ELSE
        SKIP

```

```

--}}}
:
--}}}
--{{{ PROC reset ()
PROC reset ()
  TIMER clock :
  INT now :
  SEQ
    control.register ! 0  --reset
    clock ? now
    clock ? AFTER now PLUS 1
    control.register ! 1  -- clear the reset condition
:
--}}}

TIMER clock :
[8][8]INT buffer1, buffer2 :
SEQ
  --{{{ initialize
  SEQ
    reset ()
    hclock.active ! numCols - 1
    hclock.blanking ! 1
    vclock.active ! numRows - 1
    vclock.blanking ! 1
    mclock.data ! 0  -- main clock divider
    mclock.control ! 0  -- disable clock
    SEQ i = 0 FOR 16384
      frame.buffer[i] := 0
    reset ()
  --}}}
  --{{{ read calibration frames
  SEQ i = 0 FOR 2
    SEQ j = 0 FOR 16
      SEQ k = 0 FOR 8
        --{{{ input
        PAR
          in1 ? buffer1
          in2 ? buffer2
        --}}}
      --}}}
  --}}}

WHILE TRUE
  INT base1, base2 :
  SEQ
    base1 := 0
    base2 := 8
    SEQ i = 0 FOR 16
      SEQ
        --{{{ handle sixteen sets of blocks
        SEQ j = 0 FOR 8
          SEQ
            --{{{ handle 8 blocks per set
            VAL offset1 IS j :
            SEQ
              --{{{ input
              PAR
                in1 ? buffer1
                in2 ? buffer2
              --}}}
            --{{{ write into memory
            SEQ k = 0 FOR 8
              VAL offset2 IS k TIMES 128 :
              SEQ l = 0 FOR 8
                VAL offset3 IS l TIMES 16 :
                IF
                  (i = 15) AND (k = 7)  -- last row

```

```

                                SKIP
                                TRUE
                                INT store1, store2 :
                                SEQ
                                store1 := buffer1[k][l] /\ #007F
                                store2 := buffer2[k][l] /\ #007F
                                frame.buffer[(base1 + offset3) + (offset1 + offset2)]
:= store1

                                IF
                                (j = 7) AND (l = 7)  -- last column
                                SKIP
                                TRUE
                                frame.buffer[(base2 + offset3) +
                                                (offset1 + offset2)] := store2
                                --}}}
                                --}}}
                                --}}}
                                --{{{ increment pointers
                                base1 := base1 + (8 * 128)
                                base2 := base2 + (8 * 128)
                                mclock.control ! 1 -- enable clock
                                --}}}
:

```

4.1.16. Guidance.occ

```

PROC Guidance ( CHAN OF ANY fromSP, toSP,
                fromXBar, toXBar )

--{{{ libraries
#include "s_header.inc"
--}}}
--{{{ constants
VAL min.time IS 0.009 (REAL32) :
--}}}
--{{{ PROC test.setup
PROC test.setup (CHAN OF ANY from.master, to.master)
--{{{ constants
--}}}
INT command :
REAL32 time, shift.x, shift.y:
INT last.transfer.time :
BOOL first.transfer :
TIMER clock :
REAL32 delta.x , delta.y :
INT count :
WHILE TRUE
  SEQ
    time := 115.0 (REAL32)
    shift.x := (-50.0 (REAL32))
    shift.y := (50.0 (REAL32))
    delta.x := (0.05 (REAL32))
    delta.y := (-0.05 (REAL32))
    count := 0
    first.transfer := TRUE
    from.master ? command
    WHILE command = c.guidance.run
      --{{{ run simulation
      SEQ
        IF
          first.transfer
          first.transfer := FALSE
          TRUE
            clock ? AFTER (last.transfer.time PLUS 16)
            to.master ! time ; shift.x ; shift.y
            clock ? last.transfer.time
            from.master ? command
            time := time + 0.001 (REAL32)
            shift.x := shift.x + delta.x
            shift.y := shift.y + delta.y
            count := count + 1
            IF
              count = 2000
              SEQ
                count := 0
                delta.x := (-delta.x)
                delta.y := (-delta.y)
              TRUE
              SKIP
            --}}}
      :
    --}}}
  CHAN OF ANY from.master, to.master :
  PAR
    test.setup (from.master, to.master)
    --{{{ master process
    BYTE length :
    [255]INT command.line :
    INT guidance.mode :
    SEQ

```

```

guidance.mode := gm.none
WHILE TRUE
  SEQ
    fromSP ? length::command.line
    CASE command.line[0]
      --{{{ set mode
      c.guidance.set.mode
      guidance.mode := command.line[1]
      --}}}
      --{{{ initialize
      c.guidance.initialize
      SEQ
        CASE guidance.mode
          gm.internal
            from.master ! c.guidance.initialize
          gm.external
            PAR
              toXBar ! c.guidance.initialize
              from.master ! c.guidance.initialize
            ELSE
              SKIP
          --}}}
      --}}}
      run
    c.guidance.run
      --{{{ abbreviations and variables
      --FPA      IS command.line[1] :
      range      IS command.line[1] :
      time       IS command.line[2] :
      --centroid.x IS command.line[4] :
      --centroid.y IS command.line[5] :
      --seeker.int IS [command.line FROM 6 FOR 3] :
      --target.int IS [command.line FROM 9 FOR 3] :

      --[3]REAL32 seeker.xyz RETYPES seeker.int :
      --[3]REAL32 target.xyz RETYPES target.int :
      REAL32 r.time      RETYPES time :

      REAL32 pfp.time :
      REAL32 shift.x.r, shift.y.r :
      INT   shift.x,   shift.y :
      --}}}
      SEQ
        CASE guidance.mode
          gm.internal
            --{{{ from test proc
            SEQ
              from.master ! c.guidance.run
              to.master ? pfp.time; shift.x.r; shift.y.r
              WHILE (pfp.time < r.time)
                SEQ
                  from.master ! c.guidance.run
                  to.master ? pfp.time ; shift.x.r ; shift.y.r
                --}}}
          gm.external
            --{{{ from PFP
            SEQ
              toXBar ! c.guidance.run
              fromXBar ? pfp.time ; shift.x.r ; shift.y.r
              WHILE (pfp.time < r.time)
                SEQ
                  toXBar ! c.guidance.run
                  fromXBar ? pfp.time ; shift.x.r ; shift.y.r
                --}}}
            ELSE
              SEQ
                shift.x.r := 0.0 (REAL32)
                shift.y.r := 0.0 (REAL32)

```

```
--{{{ time is okay now, send back the shift commands
--{{{ convert pixel and fraction to sub-pixel
shift.x := INT TRUNC (shift.x.r * (-4.0 (REAL32)))
shift.y := INT TRUNC (shift.y.r * 4.0 (REAL32))
--{{{ limit shift.x
IF
  shift.x > 240
    shift.x := 240
  shift.x < (-240)
    shift.x := (-240)
  TRUE
  SKIP
--}}}
--{{{ limit shift.y
IF
  shift.y > 255
    shift.y := 255
  shift.y < (-255)
    shift.y := (-255)
  TRUE
  SKIP
--}}}
--}}}
toSP ! cc.shift.image; INT shift.x; INT shift.y
--}}}
--}}}
:

```

4.1.17. HostSeeker.occ

```

PROC HostSeeker (CHAN OF ANY fromSeeker, toSeeker )
  --{{{ libraries
  #INCLUDE "s_header.inc"
  #INCLUDE "hostio.inc"
  #USE "hostio.lib"
  --}}}
  --{{{ link definitions
  VAL link0out IS 0 :
  VAL link1out IS 1 :
  VAL link2out IS 2 :
  VAL link3out IS 3 :
  VAL link0in IS 4 :
  VAL link1in IS 5 :
  VAL link2in IS 6 :
  VAL link3in IS 7 :
  --}}}
  --{{{ channels
  CHAN OF SP fs, ts :
  PLACE fs AT link0in :
  PLACE ts AT link0out :
  --}}}
  --{{{ constants
  VAL esc IS 27 :
  VAL max.screen.width IS 640 :
  VAL max.screen.height IS 480 :
  --}}}
  --{{{ utility procs
  --{{{ goto.xy
  PROC goto.xy (CHAN OF SP fs, ts, VAL INT x, y)
    VAL esc IS 27 (BYTE) :
    SEQ
      so.write.string (fs, ts, [esc, '['])
      so.write.int (fs, ts, y+1, 0)
      so.write.char (fs, ts, ';')
      so.write.int (fs, ts, x+1, 0)
      so.write.char (fs, ts, 'H')
    :
  --}}}
  --{{{ clear.eos
  PROC clear.eos (CHAN OF SP fs, ts)
    VAL esc IS 27 (BYTE) :
    SEQ
      so.write.string (fs, ts, [esc, '[', 'J'])
    :
  --}}}
  --}}}
  --{{{ SC Encode
  #USE "gif02.c8h"
  --{{{F gif02.occ
  --:::F GIF02.OCC
  --}}}
  --}}}
  --{{{ SC loader
  #USE "loader.c8h"
  --{{{F loader.occ
  --:::F LOADER.OCC
  --}}}
  --}}}
  --{{{ SC runSeeker
  #USE "runseekr.c8h"
  --{{{F runSeekr.occ
  --:::F RUNSEEKR.OCC
  --}}}
  --}}}

```

```

--{{{ spectrum
PROC spectrum ( [256]INT palette )

  PROC set.colour (VAL INT index, r, g, b)
    SEQ
      palette[index] := (r << 16) \/ ((g << 8) \/ b)
  :
  SEQ
    SEQ i = 0 FOR 64
      set.colour( i, i, 0, 31-(i>>1) )
    SEQ i = 0 FOR 64
      set.colour( 64+i, 63, i, i )
    SEQ i = 128 FOR 128
      set.colour( i, 0, 63, 0 )
    set.colour( 0, 0, 0, 0 )
    set.colour( 128, 30, 30, 30 )

  :
--}}}
--{{{ main menu variables
BOOL dont.exit :
BYTE key, result :
--}}}
SEQ
  dont.exit := TRUE
  WHILE dont.exit
    SEQ
      --{{{ Main Menu
      goto.xy( fs, ts, 0, 0 )
      clear.eos( fs, ts )
      so.write.string( fs, ts, "Main Seeker Menu*c*n" )
      so.write.string( fs, ts, "-----*c*n*n" )
      so.write.string( fs, ts, "'Esc' key to exit program.*c*n*n" )
      so.write.string( fs, ts, " (c) capture screen image*c*n" )
      so.write.string( fs, ts, " (l) load new data*c*n" )
      so.write.string( fs, ts, " (r) run seeker*c*n" )
      so.write.nl( fs, ts )

      so.getkey( fs, ts, key, result )
      --}}}
    CASE key
      --{{{ c - capture screen
      'C', 'c'
      [max.screen.width]BYTE screen.buffer :
      [max.screen.height][max.screen.width]INT screen :
      [256]INT palette :
      INT screen.height, screen.width :
      SEQ
        --{{{ header
        goto.xy( fs, ts, 0, 0 )
        clear.eos( fs, ts )
        so.write.string( fs, ts, "Capture Screen Image*c*n" )
        so.write.string( fs, ts, "-----*c*n*n" )
        --}}}
        --{{{ get image
        spectrum(palette)

        --{{{ track display
        so.write.string( fs, ts, "Sending command to track display...*c*n" )
        toSeeker ! 2 (BYTE); c.read.graphics ; track.display
        fromSeeker ? screen.height
        SEQ i = 0 FOR screen.height
          SEQ
            so.write.int (fs, ts, i, 0)
            so.write.char (fs, ts, 'c')
            fromSeeker ? screen.width::screen.buffer
            SEQ j = 0 FOR screen.width

```

```

        screen[i][j] := (INT screen.buffer[j])
so.write.nl (fs, ts)
--}}}
--{{{ image display 0
so.write.string( fs, ts, "Sending command to image display 0..."c*n"
)

toSeeker ! 2 (BYTE); c.read.graphics ; image.display.0
fromSeeker ? screen.height
SEQ i = 0 FOR screen.height
    SEQ
        so.write.int (fs, ts, i, 0)
        so.write.char (fs,ts, '*c')
        fromSeeker ? screen.width::screen.buffer
        SEQ j = 0 FOR screen.width
            screen[i][j] := screen[i][j] \ / (INT screen.buffer[j])
so.write.nl (fs, ts)
--}}}
--{{{ image display 1
so.write.string( fs, ts, "Sending command to image display 1..."c*n"
)

toSeeker ! 2 (BYTE); c.read.graphics ; image.display.1
fromSeeker ? screen.height
SEQ i = 0 FOR screen.height
    SEQ
        so.write.int (fs, ts, i, 0)
        so.write.char (fs,ts, '*c')
        fromSeeker ? screen.width::screen.buffer
        SEQ j = 0 FOR screen.width
            screen[i][j] := screen[i][j] \ / (INT screen.buffer[j])
so.write.nl (fs, ts)
--}}}
--}}}
--{{{ run gif encoder
INT32 streamid :
BYTE result :
SEQ
    so.open (fs, ts, "seeker.gif", spt.binary, spm.output,
        streamid, result)
    Encode (fs, ts, streamid, screen.width, screen.height,
        8, palette, screen)
--}}}
--}}}
--{{{ l - load new data
'L', 'l'
    loader (fs, ts, fromSeeker, toSeeker)
--}}}
--{{{ r - run seeker
'R', 'r'
    runSeeker (fs, ts, fromSeeker, toSeeker)
--}}}
--{{{ escape
(BYTE esc)
    dont.exit := FALSE
--}}}
--{{{ else skip
ELSE
    SKIP
--}}}
so.exit( fs, ts, 0 (INT32) )
:

```

4.1.18. HostStub.occ

```
PROC HostStub (CHAN OF ANY from.emulator, to.emulator)
  SKIP
:
```

4.1.19. ImageDisplay.occ

```

PROC ImageDisplay ( CHAN OF ANY Image0, Image1,
                    fromPrev, toPrev, fromNext, toNext,
                    VAL INT position, FRAMES )

--{{{ libraries
#include "crtc.inc"
#include "g_header.inc"
#include "s_header.inc"

#USE "convert.lib"
#USE "graphics.lib"
--}}}

--{{{ place system variables
[(20*65536)+1280] BYTE screen.map :
PLACE screen.map AT screen.int.address :

INT DisplayStart :
PLACE DisplayStart AT DisplayStart.address :

INT EventMode :
PLACE EventMode AT EventMode.address :

INT SysReady :
PLACE SysReady AT (#00080000 >< (MOSTNEG INT)) >> 2 :

INT Ready :
PLACE Ready AT Ready.address :
--}}}

--{{{ FrameReceiver
PROC FrameReceiver ( CHAN OF ANY in0, in1,
                    [128][1280] BYTE frame,
                    VAL INT count0, count1 )

--{{{ MOVE
PROC MOVE( [][ ] BYTE source, VAL INT s.x, s.y,
           [][ ] BYTE dest,   VAL INT d.x, d.y, l.x, l.y )

    SEQ i = 0 FOR l.y
        [dest[d.y+i] FROM d.x FOR l.x] := [source[s.y+i] FROM s.x FOR l.x]
    :
--}}}

[128][40][32] BYTE s RETYPES frame :
[2][8][16] BYTE buffer0 :
PLACE buffer0 IN WORKSPACE :
[2][8][16] BYTE buffer1 :
PLACE buffer1 IN WORKSPACE :
SEQ
    --{{{ get first lines
    PAR
        in0 ? buffer0[0]
        in1 ? buffer1[0]
    --}}}
    --{{{ get lines and place on display
    SEQ i = 0 FOR 127
        VAL input IS (i+1)/\1 :
        VAL display IS i/\1 :
        si IS s[i] :
        b0 IS buffer0[ display ] :
        b1 IS buffer1[ display ] :
        PAR
            in0 ? buffer0[input]
            --{{{ place on display
            SEQ
                MOVE2D( b0, 0, 0, si, 0, 0, 16, 8 )

```

```

        MOVE2D( b0, 0, 0, si, 0, 20, 16, 8 )
        MOVE2D( b1, 0, 0, si, 16, 0, 16, 8 )
        MOVE2D( b1, 0, 0, si, 16, 20, 16, 8 )
    --}}}
    in1 ? buffer1[input]
--}}}
--{{{ place last lines on display
si IS s[127] :
b0 IS buffer0[ 1 ] :
b1 IS buffer1[ 1 ] :
SEQ
    MOVE2D( b0, 0, 0, si, 0, 0, 16, 8 )
    MOVE2D( b0, 0, 0, si, 0, 20, 16, 8 )
    MOVE2D( b1, 0, 0, si, 16, 0, 16, 8 )
    MOVE2D( b1, 0, 0, si, 16, 20, 16, 8 )
--}}}
:
--}}}
--{{{ place Event channel
CHAN OF ANY Event :
PLACE Event AT 8 :
--}}}
--{{{ set up multiple screens
VAL screen.offset IS [ #00000, #50000, #A0000, #F0000 ] :
VAL screen.address IS [ #00000, #14000, #28000, #3C000 ] :
--}}}
--{{{ EventProc
PROC EventProc ( CHAN OF ANY Event, in )

    INT synch, address :
    WHILE TRUE
        SEQ
            in ? address
            Ready := 1
            Event ? synch
            DisplayStart := address
            Ready := 0
:
--}}}
--{{{ Buffer
PROC Buffer ( CHAN OF ANY in, out )

    INT temp :
    SEQ
        WHILE TRUE
            SEQ
                in ? temp
                out ! temp
:
--}}}
--{{{ SetNewScreen
PROC SetNewScreen ( CHAN OF ANY in, VAL INT screen address )

    INT synch :
    SEQ
        Ready := 1
        in ? synch
        DisplayStart := screen.address
        SEQ i = 0 FOR 100
            SKIP
        Ready := 0
:
--}}}
--{{{ constants
VAL screen.width IS 640 :
VAL screen.height IS 480 :
```

```

VAL screen.size      IS screen.width * screen.height :
--}}}
--{{{ variables
CHAN OF ANY synch, synchl :
INT temp, load :
[w.length] INT window :

INT start.time, end.time :
TIMER clock :

BYTE length :
[256]INT message :
--{{{ x.offset
VAL offset IS [ 32, 352 ] :
VAL x.offset IS offset[ position ] :
--}}}
--}}}
SEQ
  --{{{ initialize
  set.B408( 0, 0, 0, 0, 0 )
  window := [ 0, 640*480, 640, 0, 0, 640, 480, 255, 0, 0, 0 ]

  {} INT int.screen RETYPES screen.map :
  SEQ i = 0 FOR (20*65536)/4
    int.screen[i] := 0

  set.B408( 0, 0, 0, 1, 0 )

  load := 0
  --}}}
PRI PAR
  PAR
    EventProc( Event, synchl )
    Buffer( synch, synchl )
    --{{{ read display info
    command IS message[0] :
    who      IS message[1] :
    WHILE TRUE
      SEQ
        fromPrev ? length::message
      IF
        command = c.read.graphics
        CASE who
          image.display.0 -- this processor
          --{{{
          screen IS [screen.map FROM screen.offset[load] FOR screen.size
        ] :
          [screen.height][screen.width] BYTE s2 RETYPES screen :
          SEQ
            toPrev ! screen.height
            SEQ i = 0 FOR screen.height
              toPrev ! screen.width::s2[i]
            --}}}
          ELSE
            --{{{ image.display 1
            INT number.of.transfers :
            INT width :
            [maxGraphicBuffer]BYTE graphicsBuffer :
            SEQ
              message[1] := message[1] - 1 -- decrement and send on
              toNext ! length::message
              fromNext ? number.of.transfers
              toPrev ! number.of.transfers
              SEQ i = 0 FOR number.of.transfers
                SEQ
                  fromNext ? width::graphicsBuffer
                  toPrev ! width::graphicsBuffer

```

```

--}}
    TRUE
    SKIP
--}}
--((( get display
SEQ
  WHILE TRUE
    VAL start IS screen.offset[load] + ((112 * screen.width) + x.offset) :
    VAL size IS 256 * screen.width :
    screen1 IS [screen.map FROM start FOR size] :
    [128][screen.width*2] BYTE screen RETYPES screen1 :
    SEQ
      FrameReceiver( Image0, Image1, screen, 16, 4 )
      synch ! screen.address[load]
      load := (load + 1) /\ 3
  --}}
:

```

4.1.20. Loader.occ

```

--{{{ libraries
#include "s_header.inc"
#include "hostio.inc"
#USE "hostio.lib"
#USE "convert.lib"
--}}}

PROC loader (CHAN OF SP fs, ts, CHAN OF ANY from.seeker, to.seeker)

    --{{{ constants
    VAL esc IS 27 :

    VAL c.cal.frames IS 2 :
    VAL c.background IS 0 :
    VAL c.foreground IS 1 :
    VAL c.row.data IS 44 :

    VAL max.rows IS 128 :
    VAL max.cols IS 128 :
    VAL max.t.cols IS 512 :
    VAL max.fpa.frames IS max.frames :
    VAL max.buff.size IS 128 :
    VAL sub.pixel.x IS 4 :
    VAL sub.pixel.y IS 4 :

    --}}}

    --{{{ global variables
    INT num.frames, num.sim.frames :
    INT nncols, nnrows :
    BOOL fpa.values, sim.values :

    [max.fpa.frames]REAL32 frame.rate, fpa.time, fpa.range :
    [6][max.fpa.frames]REAL32 fpa.position :

    [max.sim.frames]REAL32 sim.time, sim.range :
    [6][max.sim.frames]REAL32 sim.position :

    --}}}

    --{{{ channels
    VAL link.in.3 IS 7 :
    VAL link.out.3 IS 3 :

    CHAN OF ANY from.vax, to.vax :
    PLACE from.vax AT link.in.3 :
    PLACE to.vax AT link.out.3 :
    --}}}

    --{{{ utility procs
    --{{{ goto.xy
    PROC goto.xy (CHAN OF SP fs, ts, VAL INT x, y)
        VAL esc IS 27 (BYTE) :
        SEQ
            so.write.string (fs, ts, [esc, '['])
            so.write.int (fs, ts, y+1, 0)
            so.write.char (fs, ts, ';')
            so.write.int (fs, ts, x+1, 0)
            so.write.char (fs, ts, 'H')
        :
    --}}}

    --{{{ clear.eos
    PROC clear.eos (CHAN OF SP fs, ts)
        VAL esc IS 27 (BYTE) :
        SEQ
            so.write.string (fs, ts, [esc, '[', 'J'])
        :

```

```
--}}
--{{{ PROC indexed.file.name
PROC indexed.file.name ([]BYTE new.name, INT length,
                        VAL []BYTE base, VAL INT index,
                        VAL INT field.width, VAL []BYTE extension)

  INT temp :
  [8]BYTE index.string :
  SEQ
    length := SIZE (base)
    [new.name FROM 0 FOR length] := base
    INTTOSTRING (temp, index.string, index)
    SF2 i = 0 FOR field.width - temp
    SEQ
      new.name[length] := '0'
      length := length + 1
    [new.name FROM length FOR temp] := [index.string FROM 0 FOR temp]
    length := length + temp
    temp := SIZE (extension)
    [new.name FROM length FOR temp] := extension
    length := length + temp

:
--}}}
--}}}
--{{{ procs
--{{{ proc reverse.bytes ([4]BYTE)
PROC reverse.bytes ([4]BYTE block)
  [4]BYTE n.block :
  SEQ
    n.block := block
    block[0] := n.block[3]
    block[1] := n.block[2]
    block[2] := n.block[1]
    block[3] := n.block[0]

:
--}}}

--{{{ proc get.INT
PROC get.INT (CHAN OF SP fs, ts, VAL INT32 streamid, INT value)
  [4]BYTE block RETYPES value :
  INT length :
  SEQ
    so.read (fs, ts, streamid, length, block)
    reverse.bytes (block)

:
--}}}
--{{{ proc get.BYTE
PROC get.BYTE (CHAN OF SF fs, ts, VAL INT32 streamid, BYTE value)
  [1]BYTE block RETYPES value :
  INT length :
  SEQ
    so.read (fs, ts, streamid, length, block)

:
--}}}
--{{{ proc get.REAL32
PROC get.REAL32 (CHAN OF SP fs, ts, VAL INT32 streamid, REAL32 value)
  [4]BYTE block RETYPES value :
  INT length :
  SEQ
    so.read (fs, ts, streamid, length, block)
    reverse.bytes (block)

:
--}}}
--{{{ proc get.REAL32.vector
PROC get.REAL32.vector (CHAN OF SP fs, ts, VAL INT32 streamid, [ ]REAL32 value)
  [ ]BYTE block RETYPES value :
  INT length :
```

```

BYTE temp :
SEQ
  so.read (fs, ts, streamid, length, block)
  SEQ i = 0 FOR length / 4
    --{{{ reverse ordering
    section IS (block FROM (i TIMES 4) FOR 4) :
    SEQ
      reverse.bytes (section)
    --}}}
  :
  --}}}
  --{{{ proc vax.filter
PROC vax.filter (CHAN OF ANY from.filter, to.filter, from.vax,
  VAL BOOL dataFromTape)
  VAL buffer.size IS 16384 :
  VAL max.record IS 4097 :
  BOOL continue, waiting.flag, full :
  INT need, count, start, end :
  BYTE count.byte :
  [buffer.size]BYTE buffer :
  SEQ
    continue := TRUE
    waiting.flag := FALSE
    start := 0
    end := 0
    full := FALSE
    IF
      dataFromTape
        --{{{ then use sophisticated buffer
        WHILE continue
          PRI ALT
            to.filter ? need
            --{{{
            IF
              need = 0
              continue := FALSE
            TRUE
            --{{{
            IF
              (end - start) >= need
              SEQ
                from.filter ! (buffer FROM start FOR need)
                start := start + need
              IF
                start = end
                SEQ
                  start := 0
                  end := 0
                TRUE
                SKIP
              TRUE
              waiting.flag := TRUE
            --}}}
          --}}}
        (NOT full) & from.vax ? count.byte
        --{{{
        SEQ
          IF
            (count byte <> (BYTE #5E))
            SEQ
              --{{{ get record
              count := (INT count.byte) /\ #0F
              SEQ i = 0 FOR 3
              SEQ
                from.vax ? count.byte
                count := (count * 10) + ((INT count.byte) /\ #0F)

```

```

count := count - 4
from.vax ? [buffer FROM end FOR count]
end := end + count
IF
  waiting.flag AND ((end - start) >= need)
  --{{{
  SEQ
    from.filter ! [buffer FROM start FOR need]
    start := start + need
  IF
    start = end
    SEQ
      start := 0
      end := 0
    TRUE
    SKIP
    waiting.flag := FALSE
  --}}}
  TRUE
  SKIP
  --}}}
  TRUE
  SKIP
  full := end > (buffer.size - max.record)
  --}}}
  (waiting.flag AND full) & SKIP
  --{{{
  [max.record]BYTE temp :
  INT length :
  SEQ
    length := end - start
    [temp FROM 0 FOR length] := [buffer FROM start FOR length]
    [buffer FROM 0 FOR length] := [temp FROM 0 FOR length]
    start := 0
    end := length
    full := FALSE
  --}}}
  --}}}
  TRUE
  --{{{
  WHILE continue
  SEQ
    to.filter ? need
  IF
    need = 0
    continue := FALSE
  TRUE
  SEQ
    from.vax ? [buffer FROM 0 FOR need]
    from.filter ! [buffer FROM 0 FOR need]
  --}}}
:
--}}}
--{{{ PROC fixVaxFloat
PROC fixVaxFloat (VAL REAL32 bad, REAL32 good)
SEQ
  VAL [4]BYTE twiddle.dee RETYPES bad :
  [4]BYTE twiddle.dum RETYPES good :
  SEQ
    twiddle.dum[0] := twiddle.dee[2]
    twiddle.dum[1] := twiddle.dee[3]
    twiddle.dum[2] := twiddle.dee[0]
    twiddle.dum[3] := twiddle.dee[1]
    --twiddle.dum[3] := BYTE ((INT twiddle.dum[3]) + 127)
    INT16 dec.exp RETYPES [twiddle.dum FROM 2 FOR 2] :
    INT16 ieee.exp :
```

```

        VAL zero IS 0 (INT16) :
        SEQ
            ieee.exp := dec.exp - 256 (INT16)
            IF
                (ieee.exp < zero) AND (dec.exp > zero)
                --{{{ fix very small number
                SEQ
                    dec.exp := zero
                --}}}
            TRUE
                dec.exp := ieee.exp
:
--}}}
--{{{ PROC clear.screen
PROC clear.screen(CHAN OF SP fs, ts)
    SEQ
        goto.xy(fs, ts, 0,0)
        clear.eos(fs, ts)
:
--}}}
--{{{ PROC checkFloat
PROC checkFloat(VAL REAL32 in, REAL32 out)
    SEQ
        IF
            ISNAN(in)
                out := 0.0 (REAL32)
                (ABS(in)) > 1.0E+20 (REAL32)
                out := 1.0E+20 (REAL32)
            TRUE
                out := in
:
--}}}
--{{{ PROC message.buffer (CHAN in, out, stop)
PROC message.buffer (CHAN OF ANY in, out, stop)
    BYTE length :
    INT any :
    BOOL continue :
    [max.message]INT buffer :
    SEQ
        continue := TRUE
        WHILE continue
            ALT
                stop ? any
                    continue := FALSE
                in ? length::buffer
                    out ! length::buffer
:
--}}}

--{{{ merge.sim.and fpa (CHAN)
PROC merge.sim.and.fpa (CHAN OF SP fs, ts, CHAN OF ANY to.seeker)
    --{{{ INT FUNCTION MIN (INT,INT)
    INT FUNCTION MIN (VAL INT a, b)
        INT return :
        VALOF
            IF
                a <= b
                    return := a
            TRUE
                return := b
        RESULT return
:
--}}}
SEQ
    --{{{ display some text
    so.write.string(fs, ts,
        "FPA and Sim values have all been loaded.*c*n")

```

```

so.write.string(fs, ts,
  "Will now merge data sets and send to Seeker Emulator*c*n")
so.write.string(fs, ts,
  "range, time, frame rate, and interceptor and target coordinates.*c*n")
--}}}
--{{{ compare sim frames to fpa frames
INT frames.sent, frames.to.send, sim.count :
SEQ
--{{{ send sim.frames
sim.count := 0
frames.sent := 0
WHILE sim.time[sim.count] < fpa.time[0]
  SEQ
  --{{{ compute range values
  range IS sim.range[sim.count] :
  SEQ
  range := 0.0 (REAL32)
  SEQ i = 0 FOR 3
  VAL temp IS (sim.position[i][sim.count] -
    sim.position[i + 3][sim.count]) :
  range := range + (temp * temp)
  range := Sqrt (range)
  --}}}
  sim.count := sim.count + 1
WHILE frames.sent < sim.count
  SEQ
  frames.to.send := MIN(sim.count - frames.sent, max.buff.size)
  SEQ i = 0 FOR 6
  to.seeker ! BYTE (4 + frames.to.send) ;
  c.sim.position; i; frames.sent ; frames.to.send ;
  [sim.position[i] FROM frames.sent FOR frames.to.send]
  to.seeker ! BYTE (3 + frames.to.send) ;
  c.frame.time ; frames.sent ; frames.to.send ;
  [sim.time FROM frames.sent FOR frames.to.send]
  to.seeker ! BYTE (3 + frames.to.send) ;
  c.frame.range ; frames.sent ; frames.to.send ;
  [sim.range FROM frames.sent FOR frames.to.send]
  frames.sent := frames.sent + frames.to.send
--}}}
--{{{ get position values for fpa frames
INT current.sim :
SEQ
current.sim := sim.count - 1
SEQ i = 0 FOR num.frames
  SEQ
  WHILE ABS(fpa.time[i] - sim.time[current.sim+1]) <
    ABS(fpa.time[i] - sim.time[current.sim])
    current.sim := current.sim + 1
  SEQ j = 0 FOR 6
  fpa.position[j][i] := sim.position[j][current.sim]
  SEQ i = 0 FOR 6
  to.seeker ! BYTE (4 + num.frames) ;
  c.sim.position; i; sim.count ; num.frames ;
  [fpa.position[i] FROM 0 FOR num.frames]
  to.seeker ! BYTE (3 + num.frames) ;
  c.frame.time ; sim.count ; num.frames ;
  [fpa.time FROM 0 FOR num.frames]
  to.seeker ! BYTE (3 + num.frames) ;
  c.frame.rate ; sim.count ; num.frames ;
  [frame.rate FROM 0 FOR num.frames]
  to.seeker ! BYTE (3 + num.frames) ;
  c.frame.range ; sim.count ; num.frames ;
  [fpa.range FROM 0 FOR num.frames]
  to.seeker ! BYTE (3) ;
  c.sim.start.frames ; sim.count ; (sim.count + num.frames) -
1
--}}}

```

```

--}}
:
--}}
--{{{ load.background (CHAN, CHAN)
PROC load.background(CHAN OF SP fs, ts, CHAN OF ANY to.seeker)

--{{{ local variables
BYTE key, result :
BOOL error :
INT32 streamid :
--}}}
SEQ
--{{{ welcome
clear.screen (fs, ts)
so.write.string (fs, ts, "Loading background data...*c*n")
--}}}
--{{{ get background frames
REAL32 diam, fnum, pxspcx,
      pxspcy, filfax, filfay :
--[max.frames]REAL32 framrt, range, time :
BYTE cr :
[128]BYTE file.name, base.name :
INT fn.length, base.name.length :
INT dummy :
BYTE result :
BOOL error :
SEQ
so.write.string(fs, ts, "*c*nNoise data...*c*n")
--{{{ Get file name for FPA characteristics
SEQ
so.write.string (fs, ts, "File name for FPA characteristics: ")
so.read.echo.line (fs, ts, fn.length, file.name, result)
so.write.nl (fs, ts)
--}}}
--{{{ open file
SEQ
so.open (fs, ts, [file.name FROM 0 FOR fn.length], spt.binary,
        spm.input, streamid, result)

IF
result <> spr.ok
so.write.string (fs, ts, "Unable to open ")
TRUE
SKIP
so.write.string.nl (fs, ts, [file.name FROM 0 FOR fn.length])
--}}}
--{{{ file header
get.INT (fs, ts, streamid, dummy)
get.INT (fs, ts, streamid, nnrows)
get.INT (fs, ts, streamid, nncols)
get.REAL32 (fs, ts, streamid, diam)
get.REAL32 (fs, ts, streamid, fnum)
get.REAL32(fs, ts, streamid, pxspcx)
get.REAL32(fs, ts, streamid, pxspcy)
get.REAL32(fs, ts, streamid, filfax)
get.REAL32(fs, ts, streamid, filfay)
get.INT (fs, ts, streamid, dummy)
--}}}
--{{{ qeff and dark current
[max.cols][max.cols]REAL32 qeff.full, dkcurr.full :
SEQ
--{{{ quantum efficiency
so.write.string(fs, ts, "Loading and sending quantum efficiency
data.*c*n")

```

```

SEQ i = 0 FOR nnrows
  qeff IS qeff.full[i] :
  SEQ
    get.INT (fs, ts, streamid, dummy)
    get.REAL32.vector(fs, ts, streamid,
                      [qeff FROM 0 FOR nncols])
    get.INT (fs, ts, streamid, dummy)
    so.write.real32( fs, ts, qeff[0], 10, 3 )
    so.write.char( fs, ts, '*c' )
    -- to.seeker ! BYTE (2 + nncols );
    -- c.gain.row ; (nnrows - i) - 1 ; [qeff FROM 0 FOR
nncols]
    --{{{ replicate for smaller arrays
    INT count :
    SEQ
      count := nncols
      WHILE count < max.cols
      SEQ
        [qeff FROM count FOR nncols] := [qeff FROM 0 FOR nncols]
        count := count + nncols
      SEQ j = 0 FOR (max.rows / nnrows)
        to.seeker ! BYTE (2 + max.cols );
        c.gain.row ; ((nnrows - i) - 1) + (j * nnrows);
        [qeff FROM 0 FOR max.cols]
      --}}}
    --}}}
  --{{{ dark current
  so.write.string(fs, ts, "Loading and sending dark current data.*c*n")
  SEQ i = 0 FOR nnrows
    dkcurr IS dkcurr.full[i] :
    SEQ
      get.INT (fs, ts, streamid, dummy)
      get.REAL32.vector(fs, ts, streamid,
                        [dkcurr FROM 0 FOR nncols])
      get.INT (fs, ts, streamid, dummy)
      so.write.real32( fs, ts, dkcurr[0], 10, 3 )
      so.write.char( fs, ts, '*c' )
      --to.seeker ! BYTE (2 + nncols );
      -- c.offset.row ; (nnrows - i) - 1 ; [dkcurr FROM 0 FOR
nncols]
      --{{{ replicate for smaller arrays
      INT count :
      SEQ
        count := nncols
        WHILE count < max.cols
        SEQ
          [dkcurr FROM count FOR nncols] := [dkcurr FROM 0 FOR nncols]
          count := count + nncols
        SEQ j = 0 FOR (max.rows / nnrows)
          to.seeker ! BYTE (2 + max.cols );
          c.offset.row ; ((nnrows - i) - 1) + (j * nnrows);
          [dkcurr FROM 0 FOR max.cols]
        --}}}
      --}}}
    --}}}
  --{{{ close file
  so.close (fs, ts, streamid, result)
  --}}}
  so.write.string(fs, ts, "Loading noise data now.*c*n")
  --{{{ determine Noise data file name(s)
  BYTE result :
  BOOL error :
  SEQ
    so.write.string (fs, ts, "Base file name for Noise data: ")
    so.read.echo.line (fs, ts, base.name.length, base.name, result)
    so.write.nl (fs, ts)

```

```

IF
    num.frames = 0
    SEQ
        so.write.string (fs, ts, "Number of frames to read: ")
        so.read.echo.int(fs, ts, num.frames, error)
    TRUE
    SKIP

--}}}
SEQ i = 0 FOR num.frames
--{{{  read and send data
INT framid :
SEQ
    --{{{  open file
    BYTE result :
    SEQ
        indexed.file.name (file.name, fn.length,
                            [base.name FROM 0 FOR base.name.length],
                            i+1, 3, ".out")
        so.open (fs, ts, [file.name FROM 0 FOR fn.length], spt.binary,
                 spm.input, streamid, result)

    IF
        result <> spr.ok
        SEQ
            so.write.string (fs, ts, "Unable to open ")
            so.write.string.nl (fs, ts, [file.name FROM 0 FOR fn.length])
        TRUE
        SKIP
    --}}}
--{{{  frame header
REAL32 temp :
INT dummy :
SEQ
    get.INT(fs, ts, streamid, dummy)
    get.INT(fs, ts, streamid, framid)
    get.INT(fs, ts, streamid, dummy)

    get.INT(fs, ts, streamid, dummy)
    get.REAL32(fs, ts, streamid, fpa.range[i])

    get.REAL32(fs, ts, streamid, fpa.time[i])

    get.REAL32(fs, ts, streamid, frame.rate[i])
    get.INT(fs, ts, streamid, dummy)

    get.INT(fs, ts, streamid, dummy)
    get.INT(fs, ts, streamid, nncols)

    get.INT(fs, ts, streamid, nnrows)
    get.INT(fs, ts, streamid, dummy)
--}}}
--{{{  show some data values
so.write.int(fs, ts, framid, 10)
so.write.int(fs, ts, nncols, 10)
so.write.int(fs, ts, nnrows, 10)
so.write.real32(fs, ts, fpa.range[i] , 10, 3)
so.write.real32(fs, ts, fpa.time[i] , 10, 3)
so.write.real32(fs, ts, frame.rate[i] , 10, 3)
so.write.nl(fs, ts)
--}}}
SEQ j = 0 FOR nnrows
    [max.cols]REAL32 noise.data :
    INT dummy :
    SEQ
        --get.INT(fs, ts, streamid, dummy)
        --get.REAL32.vector(fs, ts, streamid,

```

```

--                                [noise.data FROM 0 FOR nncols])
--get.INT(fs, ts, streamid, dummy)
SEQ temp = 0 FOR nncols
  noise.data[temp] := 0.0 (REAL32)
--{{{ replicate for smaller arrays
INT count :
SEQ
  count := nncols
  WHILE count < max.cols
    SEQ
      [noise.data FROM count FOR nncols] := [noise.data FROM 0 FOR
nncols]
      count := count + nncols
  SEQ k = 0 FOR (max.rows / nnrows)
    to.seeker ! BYTE (3 + max.cols );
    c.background.row ; i; ((nnrows - j) - 1) + (k *
nnrows);
    [noise.data FROM 0 FOR nncols] := [noise.data FROM 0 FOR nncols]
    --}}}
    --{{{ close file
    so.close (fs, ts, streamid, result)
    --}}}
    --}}}
  --}}}
--{{{ send rate, time, and range
to.seeker ! BYTE (3 + num.frames) ;
  c.frame.time ; 0 ; num.frames ;
  [fpa.time FROM 0 FOR num.frames]
to.seeker ! BYTE (3 + num.frames) ;
  c.frame.rate ; 0 ; num.frames ;
  [frame.rate FROM 0 FOR num.frames]
to.seeker ! BYTE (3 + num.frames) ;
  c.frame.range ; 0 ; num.frames ;
  [fpa.range FROM 0 FOR num.frames]
to.seeker ! BYTE (3) ;
  c.sim.start.frames ; 0 ; num.frames - 1
--}}}
--{{{ so long
so.write.nl (fs, ts)
so.write.string(fs, ts, "Finished loading background data.*c*n ")
--}}}
:
--}}}
--{{{ load.target (CHAN, CHAN)
PROC load.target (CHAN OF SP fs, ts, CHAN OF ANY to.seeker)

  --{{{ local variables
  BYTE key, result :
  BOOL error :
  INT nsize, start.frame :
  BYTE cr :
  INT32 streamid :
  [128]BYTE base.name, file.name :
  INT base.name.length, fn.length :
  --}}}
  SEQ
    --{{{ initialize
    nsize := nncols * 4
    --}}}
    --{{{ display title
    clear.screen(fs, ts)
    so.write.string(fs, ts, "Loading Target data now...*c*n")
    --}}}
    --{{{ clear old data
    --{{{ check number of frames
    IF
      num.frames = 0

```

```

        SEQ
        so.write.string (fs, ts, "Number of frames to read: ")
        so.read.echo.int (fs, ts, num.frames, error)
        so.write.nl (fs, ts)
    TRUE
    SKIP
--}}}
--{{{ zero old data
VAL num.full.rows IS nsize / sub.pixel.y :
VAL num.full.cols IS nsize / sub.pixel.x :
[max.t.cols]REAL32 target.data :
[sub.pixel.x][max.cols]REAL32 arranged.data :
SEQ
--{{{ init
SEQ i = 0 FOR max.t.cols
    target.data[i] := 0.0 (REAL32)
--}}}
--{{{ correct and format it
INT count :
SEQ
    count := 0
    SEQ j = 0 FOR num.full.cols
        SEQ k = 0 FOR sub.pixel.x
            SEQ
                arranged.data[k][j] := target.data[count]
                count := count + 1
--}}}
SEQ frame = 0 FOR 0
SEQ
    so.write.int (fs, ts, frame, 5)
    so.write.char (fs, ts, '*c')
    SEQ i = 0 FOR nsize
        SEQ
            --{{{ send it
            VAL subpixel IS (i \ sub.pixel.y) * sub.pixel.x :
            VAL current.row IS (i / sub.pixel.y) :
            SEQ j = 0 FOR sub.pixel.x
                SEQ
                    to.seeker ! BYTE (4 + num.full.cols) ;
                    c.target.row ; frame ;
                    subpixel + j;
                    current.row ;
                    [arranged.data[j] FROM 0 FOR num.full.cols]

            --}}}
--}}}
--}}}
--{{{ determine file name(s)
BYTE result :
BOOL error :
SEQ
    so.write.string (fs, ts, "Base file name for Target data: ")
    so.read.echo.line (fs, ts, base.name.length, base.name, result)
    so.write.nl (fs, ts)

--}}}
--{{{ get target frames
VAL num.full.rows IS nsize / sub.pixel.y :
VAL num.full.cols IS nsize / sub.pixel.x :
SEQ frame = 0 FOR num.frames
    SEQ
        --{{{ open file
        BYTE result :
        SEQ
            indexed.file.name (file.name, fn.length,
                                [base.name FROM 0 FOR base.name.length],

```

```

                                frame+1, 3, ".out")
so.open (fs, ts, [file.name FROM 0 FOR fn.length], spt.binary,
        spm.input, streamid, result)

IF
    result <> spr.ok
    so.write.string (fs, ts, "Unable to open ")
    TRUE
    SKIP
    so.write.string (fs, ts, [file.name FROM 0 FOR fn.length])
    so.write.char (fs, ts, '*c')
--}}}
--{{{ read data and send
REAL32 temp :
INT dummy :
[2][max.t.cols]REAL32 target.data :
[sub.pixel.x][max.cols]REAL32 arranged.data :
SEQ
    --{{{ get first target row
    target IS target.data[0] :
    SEQ
        get.INT (fs, ts, streamid, dummy)
        get.REAL32.vector(fs, ts, streamid,
            [target FROM 0 FOR nsize])
        get.INT (fs, ts, streamid, dummy)
    --}}}
    SEQ i = 0 FOR nsize
    PAR
        --{{{ get other target rows
        IF
            i = (nsize - 1)    -- done all rows
            SKIP
            TRUE
            target IS target.data[(i+1) /\ 1] :
            SEQ
                get.INT (fs, ts, streamid, dummy)
                get.REAL32.vector(fs, ts, streamid,
                    [target FROM 0 FOR nsize])
                get.INT (fs, ts, streamid, dummy)
            --}}}
        SEQ
            --{{{ correct and format it
            target IS target.data[i /\ 1] :
            INT count :
            SEQ
                count := 0
                SEQ j = 0 FOR num.full.cols
                SEQ k = 0 FOR sub.pixel.x
                SEQ
                    arranged.data[k][j] := target[count]
                    count := count + 1
                --}}}
            --{{{ send it to seeker
            --VAL subpixel IS ((sub.pixel.y - 1) - (i \ sub.pixel.y)) *
sub.pixel.x :
            --VAL current.row IS (num.full.rows - (i / sub.pixel.y)) - 1 :
            VAL subpixel IS (i \ sub.pixel.y) * sub.pixel.x :
            VAL current.row IS (i / sub.pixel.y) :
            SEQ j = 0 FOR sub.pixel.x
            SEQ
                to.seeker ! BYTE (4 + num.full.cols) ;
                c.target.row ; frame ;
                subpixel + j ;
                current.row ;
                [arranged.data[j] FROM 0 FOR num.full.cols]
            --}}}
        --}}}
    --}}}

```

```

        --{{{ close file
        so.close (fs, ts, streamid, result)
        --}}}
    --}}}
    --{{{ finish up
    INT dummy :
    SEQ
        so.write.nl(fs, ts)
        so.write.string(fs, ts, "Finished loading target data.*c*n ")
    --}}}
:
--}}}
--{{{ load.positions (CHAN, CHAN)
PROC load.positions (CHAN OF SP fs, ts, CHAN OF ANY from.vax, to.seeker)
    SEQ
        --{{{ load sim values
        BOOL continue :
        SEQ
            clear.screen(fs, ts)
            continue := TRUE
            num.sim.frames := 0
            WHILE continue
                SEQ
                    --from.vax ? sim.time[num.sim.frames]
                    SEQ i = 0 FOR 6
                        --from.vax ? sim.position[i][num.sim.frames]
                        SKIP
                    --{{{ display values
                    so.write.real32 (fs, ts, sim.time[num.sim.frames], 10, 3)
                    so.write.char(fs, ts, '*c')
                    --}}}
                    IF
                        sim.time[num.sim.frames] < 0.0 (REAL32)
                            continue := FALSE
                        TRUE
                            num.sim.frames := num.sim.frames + 1
                    --}}}
                --{{{ check if fpa values have been loaded
                IF
                    fpa.values
                        merge.sim.and.fpa (fs, ts, to.seeker)
                    TRUE
                        SKIP
                    sim.values := TRUE
                --}}}
            :
            --}}}
        --}}}
    SEQ
        --{{{ initialize
        fpa.values := FALSE
        sim.values := FALSE
        num.frames := 0
        nncols := 128 -- default
        nnrows := 128 -- default
        --}}}
        --{{{ menu
        BYTE key, result :
        BOOL dont.exit :
        SEQ
            dont.exit := TRUE
            WHILE dont.exit
                SEQ
                    --{{{ Loader Menu
                    goto.xy( fs, ts, 0, 0 )
                    clear.eos( fs, ts )

```

```

so.write.string( fs, ts, "Load New Seeker Data Menu*c*n" )
so.write.string( fs, ts, "-----*c*n*n" )
so.write.string( fs, ts, "'Esc*' key to exit program.*c*n*n" )
so.write.string( fs, ts, " (b) background data*c*n" )
so.write.string( fs, ts, " (t) target data*c*n" )
so.write.string( fs, ts, " (p) position information*c*n" )

so.getKey( fs, ts, key, result )
--}}}
CASE key
  --{{{ b - background data
  'B', 'b'
    load.background(fs, ts, to.seeker)
  --}}}
  --{{{ t - target
  'T', 't'
    load.target(fs, ts, to.seeker)
  --}}}
  --{{{ p - position info
  'P', 'p'
    load.positions(fs, ts, from.vax, to.seeker)
  --}}}
  --{{{ escape
  (BYTE esc)
    dont.exit := FALSE
  --}}}
  --{{{ else
  ELSE
    SKIP
  --}}}
--}}}
:

```

4.1.21. RunSeekr.occ

```

--{{{ libraries
#include "s_header.inc"
#include "hostio.inc"
#use "hostio.lib"
--}}}
PROC runSeeker (CHAN OF SP fs, ts, CHAN OF ANY fromSeeker, toSeeker)
--{{{ utility procs
--{{{ goto.xy
PROC goto.xy (CHAN OF SP fs, ts, VAL INT x, y)
  VAL esc IS 27 (BYTE) :
  SEQ
    so.write.string (fs, ts, [esc, '{}'])
    so.write.int (fs, ts, y+1, 0)
    so.write.char (fs, ts, ';')
    so.write.int (fs, ts, x+1, 0)
    so.write.char (fs, ts, 'H')
  :
--}}}
--{{{ clear.eos
PROC clear.eos (CHAN OF SP fs, ts)
  VAL esc IS 27 (BYTE) :
  SEQ
    so.write.string (fs, ts, [esc, '[', 'J'])
  :
--}}}
--{{{ so.get.extended.key
PROC so.get.extended.key (CHAN OF SP fs, ts, INT extended.key)
  BYTE key, result :
  SEQ
    so.getkey (fs, ts, key, result)
  IF
    key = 0 (BYTE)
  SEQ
    so.getkey (fs, ts, key, result)
    extended.key := 256 + (INT key)
  TRUE
    extended.key := (INT key)
  :
--}}}
--{{{ get.real32
PROC get.real32 (CHAN OF SP fs, ts, REAL32 value )

  BOOL error :
  SEQ
    so.read.echo.real32( fs, ts, value, error )
  WHILE error
  SEQ
    so.write.string( fs, ts, "**c*Illegal Real Number : " )
    so.read.echo.real32( fs, ts, value, error )
  :
--}}}
--{{{ get.int
PROC get.int (CHAN OF SP fs, ts, INT value )

  BOOL error :
  SEQ
    so.read.echo.int( fs, ts, value, error )
  WHILE error
  SEQ
    so.write.string( fs, ts, "**c*Illegal Integer : " )
    so.read.echo.int( fs, ts, value, error )
  :
--}}}
--}}}

```

```

--{{{ constants
VAL esc IS 27 :
--}}}
--{{{ move.table
VAL move.table IS [ [ 328, 0, 1 ],      -- up
                    [ 336, 0, -1 ],     -- down
                    [ 331, 1, 0 ],      -- left
                    [ 333, -1, 0 ],     -- right
                    [ 329, -1, 1 ],     -- up and right
                    [ 337, -1, -1 ],    -- down and right
                    [ 335, 1, -1 ],     -- down and left
                    [ 327, 1, 1 ],     -- up and left
                    [ 56, 0, 4 ],      -- up
                    [ 50, 0, -4 ],     -- down
                    [ 52, 4, 0 ],      -- left
                    [ 54, -4, 0 ],     -- right
                    [ 57, -4, 4 ],     -- up and right
                    [ 51, -4, -4 ],    -- down and right
                    [ 49, 4, -4 ],     -- down and left
                    [ 55, 4, 4 ] ] :   -- up and left
--}}}
--{{{ variables
BYTE key, result :
BOOL running :
--}}}
SEQ
  --{{{ initialize
  so.write.string( fs, ts, "Initialize Start Frame (y/n) " )
  so.getkey( fs, ts, key, result )
  CASE key
    'Y', 'y'
      toSeeker ! 3(BYTE); c.sim.start.frames; 0; 0
    ELSE
      SKIP

  toSeeker ! 1(BYTE); c.test.controller
  toSeeker ! 2(BYTE); c.global.scale; 0.5E-3(REAL32)
  toSeeker ! 6(BYTE); c.set.calibration; 2;
                    2000000.0(REAL32); 80000000.0(REAL32); 750; 30000

  running := TRUE
  --}}}
  WHILE running
    SEQ
      --{{{ Run Seeker Menu
      goto.xy( fs, ts, 0, 0 )
      clear.eos( fs, ts )
      so.write.string( fs, ts, "Run Seeker Menu*c*n" )
      so.write.string( fs, ts, "-----c*n*n" )
      so.write.string( fs, ts, "'Esc' key to return to previous menu.*c*n*n" )
      so.write.string( fs, ts, " ( ) single frame step*c*n" )
      so.write.string( fs, ts, " (a) set to first fpa frame*c*n" )
      so.write.string( fs, ts, " (c) continous mode*c*n" )
      so.write.string( fs, ts, " (f) set frame number*c*n" )
      so.write.string( fs, ts, " (g) initialize guidance*c*n" )
      so.write.string( fs, ts, " (r) restart from calibration*c*n" )
      so.write.string( fs, ts, " (s) set A/D gain*c*n" )
      so.write.string( fs, ts, " (t) set test mode*c*n" )
      so.write.string( fs, ts, " (x) set first and last frame*c*n" )
      so.write.string( fs, ts, " (z) set calibration*c*n" )
      so.write.nl( fs, ts )

      so.getkey( fs, ts, key, result )
      --}}}
      CASE key
        --{{{ single frame
        ' '

```

```

        toSeeker ! 1(BYTE); c.run.single
    --}}}
    --{{{ continuous
    'C', 'c'
    INT key :
    SEQ
    --{{{ display continuous menu
    goto.xy (fs, ts, 0, 0)
    clear.eos (fs, ts)
    so.write.string( fs, ts, "Continuous Mode*c*n" )
    so.write.string( fs, ts, "-----*c*n*n" )
    so.write.string( fs, ts, "'Esc*' key to return to previous
menu.*c*n" )
    so.write.string( fs, ts, " Press cursor keys to shift image*c*n*n" )
    so.write.string( fs, ts, " Home      | PgUp*c*n*n" )
    so.write.string( fs, ts, " <---      --->*c*n*n" )
    so.write.string( fs, ts, " End      | PgDn*c*n*n" )
    so.write.string( fs, ts, " Use *'Shift*' key for faster movement." )
    --}}}
    toSeeker ! 1(BYTE); c.run.continuous
    so.get.extended.key (fs, ts, key)
    WHILE (key <> esc)
        SEQ
        IF
            IF i = 0 FOR SIZE move.table
                key = move.table[i][0]
                toSeeker ! cc.shift.image; move.table[i][1];
move.table[i][2]
            TRUE
            SKIP
            so.get.extended.key (fs, ts, key)
            toSeeker ! cc.exit
        --}}}
    --{{{ restart
    'R', 'r'
    toSeeker ! 1(BYTE); c.restart
    --}}}
    --{{{ start from first fpa image
    'A', 'a'
    toSeeker ! 4(BYTE); c.start.frame; 1; 0; 1
    --}}}
    --{{{ scale
    'S', 's'
    REAL32 scale :
    SEQ
    so.write.string( fs, ts, "*c*nEnter new scale: " )
    get.real32( fs, ts, scale )
    toSeeker ! 2(BYTE); c.global.scale; scale
    --}}}
    --{{{ test mode
    'T', 't'
    INT key :
    SEQ
    toSeeker ! 1(BYTE); c.test.background
    toSeeker ! 1(BYTE); c.test.controller
    --}}}
    --{{{ frame
    'F', 'f'
    INT relative, frame :
    SEQ
    so.write.string (fs, ts, "*c*nChange frame (y/n)? ")
    so.getkey (fs, ts, key, result)
    CASE key
        'Y', 'y'
        --{{{ get frame
        SEQ
        so.write.string (fs, ts, "*c*nStart relative to first FPA

```

```

frame (y/n)? ")
    so.getkey (fs, ts, key, result)
    CASE key
        'Y', 'y'
            relative := 1
        ELSE
            relative := 0
    so.write.string( fs, ts, "**c*nStart Frame: " )
    get.int( fs, ts, frame )
    --}}}
    ELSE
        relative := -1
    so.write.string( fs, ts, "**c*nDisplay fixed frame (y/n)? " )
    so.getkey (fs, ts, key, result)
    CASE key
        'Y', 'y'
            toSeeker ! 4(BYTE); c.start.frame; relative; frame; 0
        ELSE
            toSeeker ! 4(BYTE); c.start.frame; relative; frame; 1
    --}}}
    --{{{ set calibration
    'Z', 'z'
    REAL32 c0, c1 :
    INT sp0, spl :
    SEQ
        so.write.string( fs, ts, "**c*nLevel for First Calibration on Seeker:
" )
        get.real32( fs, ts, c0 )
        so.write.string( fs, ts, "**c*nLevel for Second Calibration on
Seeker: " )
        get.real32( fs, ts, c1 )
        so.write.string( fs, ts, "**c*nLevel for First Calibration on SP: " )
        get.int( fs, ts, sp0 )
        so.write.string( fs, ts, "**c*nLevel for Second Calibration on SP: "
)
        get.int( fs, ts, spl )
        toSeeker ! 6(BYTE); c.set.calibration; 2; c0; c1; sp0; spl
    --}}}
    --{{{ set first frame
    'X', 'x'
    INT first, last :
    SEQ
        so.write.string( fs, ts, "**c*nEnter value for first frame: " )
        get.int( fs, ts, first )
        so.write.string( fs, ts, "**c*nEnter value for last frame: " )
        get.int( fs, ts, last )
        toSeeker ! 3(BYTE); c.sim.start.frames; first; last
    --}}}
    --{{{ guidance initialize
    'G', 'g'
    BOOL not.valid.key, abort :
    SEQ
        --{{{ display guidance menu
        goto.xy (fs, ts, 0, 0)
        clear.eos (fs, ts)
        so.write.string( fs, ts, "Guidance Selection*c*n" )
        so.write.string( fs, ts, "-----c*n" )
        so.write.string( fs, ts, "'Esc' key to return to previous
menu.*c*n*n" )
        so.write.string( fs, ts, " (x) crossbar test*c*n" )
        so.write.string( fs, ts, " (i) internal test*c*n" )
        so.write.string( fs, ts, " (n) no guidance*c*n" )
        --}}}
        not.valid.key := TRUE
        WHILE not.valid.key
            SEQ

```

```

not.valid.key := FALSE
abort := FALSE
so.getkey (fs, ts, key, result)
CASE key
  --{{{ x
  'X', 'x'
    toSeeker ! 2(BYTE); c.guidance.set.mode; gm.external
  --}}}
  --{{{ i
  'I', 'i'
    toSeeker ! 2(BYTE); c.guidance.set.mode; gm.internal
  --}}}
  --{{{ n
  'N', 'n'
    toSeeker ! 2(BYTE); c.guidance.set.mode; gm.none
  --}}}
  --{{{ escape
  (BYTE esc)
    abort := TRUE
  --}}}
  --{{{ else
  ELSE
    not.valid.key := TRUE
  --}}}
IF
  abort
  SKIP
  TRUE
    toSeeker ! 1(BYTE); c.guidance.initialize
  --}}}
  --{{{ escape
  (BYTE esc)
    running := FALSE
  --}}}
  --{{{ other
  ELSE
    SKIP
  --}}}

```

4.1.22. SecondBuffer.occ

```

PROC SecondBuffer ( CHAN OF ANY in, fromNext, toPrev, to.out,
                    VAL INT position, shift )

--{{{ constants
--}}}
--{{{ variables
[2][64] INT input.buffer :
INT count :
--}}}
--{{{ channels
CHAN OF ANY synch, internal, internal2 :
--}}}
--{{{ Receiver
PROC Receiver ( CHAN OF ANY in, out, [2][64] INT buffer )

    INT i :
    SEQ
        i := 0
        WHILE TRUE
            SEQ
                in ? buffer[i]
                out ! i
                i := 1 - i
            :
        --}}}
--{{{ Extractor
PROC Extractor ( CHAN OF ANY internal, in, out, out2,
                 VAL INT count, position )

--{{{ variables
[2][64][2] BYTE buffer :
INT output :
--}}}
SEQ
    internal ? buffer[0]
    output := 0
    WHILE TRUE
        SEQ
            SEQ i = 0 FOR count
            SEQ
                PAR
                    out ! buffer[output]
                    IF
                        (position = 0) OR (position = 8)
                        out2 ! buffer[output]
                    TRUE
                    SKIP
                    in ? buffer[ 1-output ]
                    output := 1 - output
                PAR
                    IF
                        (position = 0) OR (position = 8)
                        out2 ! buffer[output]
                    TRUE
                    SKIP
                    out ! buffer[output]
                    internal ? buffer[1-output]
                    output := 1 - output
            :
        --}}}
--{{{ Formatter
PROC Formatter ( CHAN OF ANY synch, out,
                 [2][64] INT input.buffer )

```

```

--{{{ variables
[2][64][4] BYTE b.in RETYPES input.buffer :
[64][2] BYTE buffer :
[64*2] BYTE buffer1 RETYPES buffer :
INT in.ptr :
--}}}
SEQ
  WHILE TRUE
    SEQ
      --{{{ form message in buffer
      SEQ
        synch ? in.ptr

        source IS input.buffer[in.ptr] :
        INT p :
        SEQ
          p := 0
          SEQ i = 0 FOR 64
            INT store :
            SEQ
              store := source[i] >> shift
              --{{{ check for zeroing store
              IF
                store = 0
                IF
                  source[i] <> 0
                  store := 1
                  TRUE
                  store := 0
                TRUE
                SKIP
              --}}}
              buffer1[p] := BYTE store
              buffer1[p+1] := BYTE store
              p := p + 2
            --}}}
          out ! buffer
        :
      --}}}
    --{{{ GTSPI.Formatter
    PROC GTSPI.Formatter (CHAN OF ANY in, out)
      --{{{ variables
      [2][64][2] BYTE buffer :
      [2][8][64][2] BYTE master.format.buffer :
      INT output, input :
      --}}}
      CHAN OF ANY sync :
      SEQ
        input := 0
        output := 0
        PAR
          --{{{ input and format
          WHILE TRUE
            SEQ
              in ? buffer[0]
              SEQ i = 0 FOR 8
                SEQ
                  PAR
                    --{{{ input
                    IF
                      i = 7
                      SKIP
                    TRUE
                      in ? buffer[(i+1) /\ 1]
                    --}}}
                  --{{{ process
                  [8][64]INT16 format.buffer RETYPES master.format.buffer[input] :

```

```

[8][8]INT16 work.buffer RETYPES buffer[i /\ 1] :
--format.buffer IS master.format.buffer[input] :
--work.buffer IS buffer[i /\ 1] :
--[64][16]BYTE format.buffer.f RETYPES format.buffer :
--SEQ
-- MOVE2D ( work.buffer, 0, 0,
--          format.buffer.f, i * 2, 0, 2, 64)
SEQ 1 = 0 FOR 8
  VAL base IS (1 TIMES 8) + i:
  SEQ
    format.buffer[0][base] := work.buffer[0][1]
    format.buffer[1][base] := work.buffer[1][1]
    format.buffer[2][base] := work.buffer[2][1]
    format.buffer[3][base] := work.buffer[3][1]
    format.buffer[4][base] := work.buffer[4][1]
    format.buffer[5][base] := work.buffer[5][1]
    format.buffer[6][base] := work.buffer[6][1]
    format.buffer[7][base] := work.buffer[7][1]
  --}}}
  input := 1 - input
  sync ! 1
--}}}
--{{{ output
WHILE TRUE
  INT ok :
  SEQ
    sync ? ok
    out ! master.format.buffer[output]
    output := 1 - output
--}}}
:
--}}}
SEQ
  IF
    position < 8
    count := 7 - position
  TRUE
    count := 15 - position
PRI PAR
  PAR
    Receiver( in, synch, input.buffer )
    Extractor( internal, fromNext, toPrev, internal2, count, position )
  --{{{ GTSPI.Formatter
  IF
    (position = 0) OR (position = 8)
    GTSPI.Formatter (internal2, to.out)
  TRUE
    SKIP
  --}}}
  Formatter( synch, internal, input.buffer )
:

```

4.1.23. SP.occ

```

PROC SP ( CHAN OF ANY in, out, fromPrev, toPrev, fromNext, toNext,
          VAL INT position )

  #INCLUDE "s_header.inc"
  --{{{ constants
  VAL packet.length IS 8 :
  VAL num.packets IS (128 * 8) / packet.length :

  VAL fraction.bits IS 8 :
  VAL tolerance IS 4 :
  --}}}
  --{{{ ProcessFrame
  PROC ProcessFrame ( CHAN OF ANY in, out,
                      VAL INT lower.threshold, upper.threshold,
                      [128][8] INT Gain, Offset,
                      INT max.data, max.row, max.col )

    --{{{ constants
    VAL l.thr IS lower.threshold << fraction.bits :
    VAL u.thr IS upper.threshold << fraction.bits :
    --}}}
    --{{{ ProcessRow
    PROC ProcessRow ( [packet.length] INT data, gain, offset, VAL INT row )

      SEQ i = 0 FOR packet.length
      INT value :
      SEQ
        value := (data[i] TIMES gain[i]) + offset[i]
      IF
        value < l.thr
          data[i] := 0
        value > u.thr
          data[i] := upper.threshold
      TRUE
        data[i] := value >> fraction.bits
      --{{{ find hot spot
      IF
        data[i] > max.data
          SEQ
            max.data := data[i]
            max.row := row
            max.col := i
          TRUE
            SKIP
      --}}}
    :
    --}}}
    --{{{ retype array to packet.length
    [num.packets][packet.length] INT p.gain RETYPES Gain :
    [num.packets][packet.length] INT p.offset RETYPES Offset :
    --}}}
    --{{{ variables
    INT in.ptr, out.ptr, process.ptr, temp :
    [3][packet.length] INT buffer :
    PLACE buffer IN WORKSPACE :
    --}}}
    SEQ
      --{{{ initialize
      in.ptr := 2
      process.ptr := 1
      out.ptr := 0

      max.data := -1
      max.row := 0

```

```

max.col := 0
--}}}
--{{{ get first row
in ? buffer[0]
--}}}
--{{{ get second row and process first row
PRI PAR
  in ? buffer[1]
  ProcessRow( buffer[0], p.gain[1], p.offset[1], 0 )
--}}}
--{{{ do middle rows
SEQ row = 1 FOR (num.packets-2)
  PRI PAR
    PAR
      in ? buffer[in.ptr]
      out ! buffer[out.ptr]
      ProcessRow( buffer[process.ptr], p.gain[row], p.offset[row], row )
      temp := out.ptr
      out.ptr := process.ptr
      process.ptr := in.ptr
      in.ptr := temp
    --}}}
  --{{{ process last row
  VAL i IS num.packets - 1 :
  PRI PAR
    out ! buffer[out.ptr]
    ProcessRow( buffer[process.ptr], p.gain[i], p.offset[i], i )
  --}}}
  --{{{ output last row
  out ! buffer[ process.ptr ]
  --}}}
  --{{{ determine actual max.col position
  max.col := (max.col << 4) + position
  --}}}
:
--}}}
--{{{ CalibrateFrame
PROC CalibrateFrame ( CHAN OF ANY in, out,
  VAL INT c.frame, c.level,
  [128][8] INT Gain, Offset )

--{{{ global variables
INT D0, D1, d0, delta.D, half.range :
--}}}
--{{{ ProcessRow
PROC ProcessRow ( [packet.length] INT data, gain, offset )

IF
  c.frame = 0
  gain := data
  TRUE
  SEQ i = 0 FOR packet.length
    INT previous.pixel, delta.p :
    SEQ
      previous.pixel := gain[i]
      delta.p := data[i] - previous.pixel
      IF
        delta.p < tolerance
        --{{{ dead pixel
        SEQ
          gain[i] := 0
          offset[i] := (previous.pixel - tolerance) << fraction.bits
        --}}}
      TRUE
      --{{{ normal pixel
      SEQ

```

```

        gain[i] := delta.D / delta.p
        offset[i] := (d0 - (gain[i] TIMES previous.pixel)) +
half.range
        --}}}
:
--}}}
--{{{ retype array to packet.length
[num.packets][packet.length] INT  p.gain      RETYPES Gain      :
[num.packets][packet.length] INT  p.offset    RETYPES Offset    :
--}}}
--{{{ variables
INT in.ptr, out.ptr, process.ptr, temp :
[3][packet.length] INT buffer :
PLACE buffer IN WORKSPACE :
--}}}
SEQ
--{{{ set up calibration
IF
    c.frame = 0
    D0 := c.level
    TRUE
    SEQ
        D1 := c.level
        delta.D := (D1 - D0) << fraction.bits
        d0 := D0 << fraction.bits
        half.range := 1 << (fraction.bits-1)
--}}}
in.ptr := 2
process.ptr := 1
out.ptr := 0
--{{{ get first row
in ? buffer[0]
--}}}
--{{{ get second row and process first row
PRI PAR
    in ? buffer[1]
    ProcessRow( buffer[0], p.gain[1], p.offset[1] )
--}}}
--{{{ do middle rows
SEQ row = 1 FOR (num.packets-2)
    SEQ
        PRI PAR
            PAR
                in ? buffer[in.ptr]
                out ! buffer[out.ptr]
                ProcessRow( buffer[process.ptr], p.gain[row], p.offset[row] )
                temp := out.ptr
                out.ptr := process.ptr
                process.ptr := in.ptr
                in.ptr := temp
            --}}}
        --{{{ process last row
        VAL i IS num.packets - 1 :
        PRI PAR
            out ! buffer[out.ptr]
            ProcessRow( buffer[process.ptr], p.gain[i], p.offset[i] )
        --}}}
        --{{{ output last row
        out ! buffer[ process.ptr ]
        --}}}
:
--}.
--{{{ variables
--{{{ command variables
BYTE length :
[max.message] INT message :
command IS message[0] :

```

```

params IS [message FROM 1 FOR (max.message-1)] :
--}}}

[128][8] INT Gain, Offset :
INT max.data, max.row, max.col :
--}}}
SEQ
  --{{{ initialize
  --}}}
  WHILE TRUE
    SEQ
      --{{{ get command and pass on
      fromPrev ? length::message
      IF
        position < 15
          toNext ! length::message
          TRUE
          SKIP
        --}}}
      --{{{ process command
      CASE command
        --{{{ c.sp.frame
        c.sp.frame
        IF
          params[0] < 0
            SEQ
              ProcessFrame( in, out, params[2], params[3], Gain, Offset,
                            max.data, max.row, max.col )
              --{{{ return hot spot
              --IF
              -- position < 15
              -- INT next.data, next.row, next.col :
              -- SEQ
              -- fromNext ? next.data; next.row; next.col
              -- IF
              -- next.data > max.data
              -- toPrev ! next.data; next.row; next.col
              -- TRUE
              -- toPrev ! max.data; max.row; max.col
              -- TRUE
              -- toPrev ! max.data; max.row; max.col
              --}}}
            TRUE
          CalibrateFrame( in, out, params[0], params[1], Gain, Offset )
          --}}}
        --}}}
  :

```

4.1.24. SPControl.occ

```

PROC SPController ( CHAN OF ANY fromController, toController,
                    fromGuidance, toGuidance,
                    fromGraphics, toGraphics, fromSP, toSP )

--{{{ control SP and give guidance commands
#include "s_header.inc"
--{{{ variables
BYTE length :
[ max.message ] INT message :
command IS message[0] :
params IS [ message FROM 1 FOR ( max.message-1 ) ] :
INT shift.x, shift.y :
INT shift.command :
--}}}
SEQ
  WHILE TRUE
    ALT
      --{{{ transfer shift command from guidance to controller
      fromGuidance ? shift.command; shift.x; shift.y
      SEQ
        toController ! shift.command; shift.x; shift.y
      --}}}
      fromController ? length::message
      IF
        --{{{ Guidance commands
        ( command >= 1280 ) AND ( command < 1536 )
        toGuidance ! length::message
        --}}}
        --{{{ Track Display commands
        ( command >= 1536 ) AND ( command < 1792 )
        toGraphics ! length::message
        --}}}
        --{{{ c.sp.frame
        command = c.sp.frame
        PAR
          toSP ! length::message
          IF
            params[0] < 0
            --{{{ regular frame
            [ 4 ] REAL32 r.param RETYPES [ params FROM 4 FOR 4 ] :
            [ 4 ] REAL32 d.param :
            PAR
              toGuidance ! 3(BYTE); c.guidance.run; [ params FROM 4 FOR 2 ]
              toGraphics ! 6(BYTE); c.display.info; 1; [ params FROM 4 FOR
4]
              --SEQ
              -- d.param[0] := r.param[0] * 1.0 (REAL32) --scaling
factors
              -- d.param[1] := r.param[1] * 1.0 (REAL32) -- (removed)
              -- d.param[2] := r.param[2]
              -- d.param[3] := r.param[3] * 1.0 (REAL32)
              -- toGraphics ! 6(BYTE); c.display.info; 1; [ d.param FROM 0
FOR 4]
              --}}}
            TRUE
            --{{{ calibration frame
            SEQ
              toGraphics ! 6(BYTE); c.display.info; 1;
              [ params FROM 4 FOR 4 ]
            --}}}
          --}}}
        --{{{ c.read.graphics
        command = c.read.graphics
        INT bufLength :

```

```
INT number.of.transfers :  
[maxGraphicBuffer]BYTE graphicsBuffer :  
SEQ  
  toGraphics    ! length::message  
  fromGraphics ? number.of.transfers  
  toController ! number.of.transfers  
  SEQ i = 0 FOR number.of.transfers  
    SEQ  
      fromGraphics ? bufLength::graphicsBuffer  
      toController ! bufLength::graphicsBuffer  
  --}}}  
--}}}  
:
```

4.1.25. Target.OCC

```

PROC Target ( CHAN OF ANY fromDown, toDown, fromUp, toUp,
              VAL INT column.position, row.position )

  #INCLUDE "s_header.inc"
  --{{{ constants
  VAL first.row IS row.position TIMES 16 :
  VAL next.first.row IS first.row + 16 :
  --}}}
  --{{{ SendFrame
  PROC SendFrame ( CHAN OF ANY in, out, [16][8] REAL32 target,
                  VAL INT shift.y, shift.x )

    --{{{ variables
    INT start.col, crossbar.offset :
    INT total.in, diff :
    INT output :

    [16][8] INT buffer :
    [2][8] INT b :
    --}}}
    SEQ
    --{{{ calculate values
    crossbar.offset := shift.x /\ 15
    start.col := shift.x >> 4
    IF
      column.position < crossbar.offset
      start.col := start.col + 1
    TRUE
    SKIP

    total.in := (7 - row.position) << 4
    --}}}
    --{{{ shift target data
    [16][8*4] BYTE b.buffer RETYPES buffer :
    [16][8*4] BYTE b.target RETYPES target :

    VAL start IS start.col << 2 :
    VAL length IS (8 - start.col) << 2 :
    SEQ
    IF
      start <> 32
      MOVE2D( b.target, start, 0, b.buffer, 0, 0, length, 16 )
      TRUE
      SKIP
    IF
      start.col <> 0
      MOVE2D( b.target, 0, 0, b.buffer, length, 0, start, 16 )
      TRUE
      SKIP
    --}}}
    --{{{ send target data
    diff := shift.y - (row.position << 4)
    IF
      row.position = 7
      --{{{ no one is above you
      SEQ
      IF
        diff < 1
        diff := 0
      TRUE
      SKIP
      SEQ i = diff FOR (16-diff)
        out ! buffer[i]
      SEQ i = 0 FOR diff

```

```

        out ! buffer[i]
    --}}}
diff < 0
--{{{ start row is below
SEQ
    PAR
        in ? b[0]
        SEQ i = 0 FOR 16
        out ! buffer[i]
    output := 0
    SEQ i = 0 FOR (total.in - 1)
    SEQ
        PAR
            in ? b[1-output]
            out ! b[output]
            output := 1-output
        out ! b[output]
    --}}}
(diff - 16) <= 0
--{{{ start row is in the middle
SEQ
    PAR
        in ? b[0]
        SEQ i = diff FOR 16-diff
        out ! buffer[i]
    output := 0
    SEQ i = 0 FOR (total.in - 1)
    SEQ
        PAR
            in ? b[1-output]
            out ! b[output]
            output := 1-output
        out ! b[output]
    SEQ i = 0 FOR diff
    out ! buffer[i]
    --}}}
TRUE
--{{{ start row is in middle of processor further down
SEQ
    in ? b[0]
    output := 0
    SEQ i = 0 FOR (127 - shift.y)
    SEQ
        PAR
            in ? b[1-output]
            out ! b[output]
            output := 1-output
        out ! b[output]
    PAR
        in ? b[output]
        SEQ i = 0 FOR 16
        out ! buffer[i]
    SEQ i = 0 FOR ((total.in + shift.y) - 129)
    SEQ
        PAR
            in ? b[1-output]
            out ! b[output]
            output := 1-output
        out ! b[output]
    --}}}
--}}}
:
--}}}
--{{{ GetTargetRow
PROC GetTargetRow ( [16][8] REAL32 Target, VAL INT row, [128] INT data )

    [] REAL32 r.data RETYPES data :
```

```

IF
  (row >= first.row) AND (row < next.first.row)
  SEQ i = 0 FOR 8
    Target[ row-first.row ][ i ] := r.data[ (i*16) + column.position ]
  TRUE
  SKIP
:
--}}}
--{{{ variables
BYTE length :
[max.message] INT message :
command IS message[0] :
params IS [message FROM 1 FOR (max.message-1)] :

[max.frames][16][16][8] REAL32 Frame :
--}}}
SEQ
  --{{{ initialize last target frame
  SEQ r = 0 FOR 64
    SEQ c = 0 FOR 8
      SEQ o = 0 FOR 4
        INT column, row :
        REAL32 value :
        SEQ
          --{{{ determine col value
          column := ((c << 6) + o) + (column.position << 2) + 1
          IF
            column < 256
              SKIP
            TRUE
              column := 513 - column
          --}}}
          --{{{ determine row value
          row := ((first.row << 2) + r) + 1
          IF
            row < 256
              SKIP
            TRUE
              row := 513 - row
          --}}}
          value := REAL32 ROUND( (row * column) )
          Frame[max.frames-1][((r/\3)<<2)+o][r>>2][c] := value
        --}}}
      WHILE TRUE
        SEQ
          --{{{ get command and send up
          fromDown ? length::message
          IF
            row.position < 7
              toUp ! length::message
            TRUE
              SKIP
          --}}}
          CASE command
            --{{{ c.set.target
            c.set.target
              SendFrame( fromUp, toDown, Frame[ params[0] ][ params[1] ],
                params[2], params[3] )
            --}}}
            --{{{ c.target.row
            c.target.row
              GetTargetRow( Frame[ params[0] ][ params[1] ], params[2],
                [params FROM 3 FOR 128] )
            --}}}
          :

```

4.1.26. TargetLead.occ

```

PROC TargetLead ( CHAN OF ANY fromUp, toUp, toDown,
                  fromPrev, toPrev, fromNext, toNext,
                  VAL INT column.position )

    #INCLUDE "s_header.inc"
    --{{{ constants
    VAL row.position      IS 0 :
    VAL first.row        IS 0 :
    VAL next.first.row   IS 16 :
    --}}}
    --{{{ SendFrame
    PROC SendFrame ( CHAN OF ANY in, out, [16][8] REAL32 target,
                    VAL INT shift.y, shift.x )

        --{{{ variables
        INT start.col, crossbar.offset :
        INT total.in, diff :
        INT output :

        [16][8] INT buffer :
        [2][8] INT b :
        --}}}
        SEQ
        --{{{ calculate values
        crossbar.offset := shift.x /\ 15
        start.col := shift.x >> 4
        IF
            column.position < crossbar.offset
                start.col := start.col + 1
            TRUE
            SKIP

        total.in := (7 - row.position) << 4
        --}}}
        --{{{ shift target data
        [16][8*4] BYTE b.buffer RETYPES buffer :
        [16][8*4] BYTE b.target RETYPES target :

        VAL start IS start.col << 2 :
        VAL length IS (8 - start.col) << 2 :
        SEQ
        IF
            start <> 32
                MOVE2D( b.target, start, 0, b.buffer, 0, 0, length, 16 )
            TRUE
            SKIP
        IF
            start.col <> 0
                MOVE2D( b.target, 0, 0, b.buffer, length, 0, start, 16 )
            TRUE
            SKIP
        --}}}
        --{{{ send target data
        diff := shift.y - (row.position << 4)
        IF
            row.position = 7
                --{{{ no one is above you
                SEQ
                IF
                    diff < 1
                        diff := 0
                    TRUE
                    SKIP
                SEQ i = diff FOR (16-diff)
            
```

```

        out ! buffer[i]
    SEQ i = 0 FOR diff
        out ! buffer[i]
    --}}}
diff < 0
--{{{ start row is below
SEQ
    PAR
        in ? b[0]
        SEQ i = 0 FOR 16
            out ! buffer[i]
        output := 0
        SEQ i = 0 FOR (total.in - 1)
            SEQ
                PAR
                    in ? b[1-output]
                    out ! b[output]
                    output := 1-output
                out ! b[output]
            --}}}
        (diff - 16) <= 0
        --{{{ start row is in the middle
        SEQ
            PAR
                in ? b[0]
                SEQ i = diff FOR 16-diff
                    out ! buffer[i]
                output := 0
                SEQ i = 0 FOR (total.in - 1)
                    SEQ
                        PAR
                            in ? b[1-output]
                            out ! b[output]
                            output := 1-output
                        out ! b[output]
                    SEQ i = 0 FOR diff
                        out ! buffer[i]
                --}}}
        TRUE
        --{{{ start row is in middle of processor further down
        SEQ
            in ? b[0]
            output := 0
            SEQ i = 0 FOR (127 - shift.y)
                SEQ
                    PAR
                        in ? b[1-output]
                        out ! b[output]
                        output := 1-output
                    out ! b[output]
                PAR
                    in ? b[output]
                    SEQ i = 0 FOR 16
                        out ! buffer[i]
                    SEQ i = 0 FOR ((total.in + shift.y) - 129)
                        SEQ
                            PAR
                                in ? b[1-output]
                                out ! b[output]
                                output := 1-output
                            out ! b[output]
                        --}}}
                --}}}
        :
        --}}}
        --{{{ GetTargetRow
        PROC GetTargetRow ( [16][8] REAL32 Target, VAL INT row, [128] INT data )

```

```

[ ] REAL32 r.data RETYPES data :
IF
  (row >= first.row) AND (row < next.first.row)
    SEQ i = 0 FOR 8
      Target[ row-first.row ][ i ] := r.data[ (i*16) + column.position ]
    TRUE
    SKIP
:
--}}}
--{{{ variables
BYTE length :
[ max.message ] INT message :
command IS message[0] :
params IS [ message FROM 1 FOR (max.message-1) ] :

[ max.frames ][ 16 ][ 16 ][ 8 ] REAL32 Frame :
--}}}
SEQ
  --{{{ initialize last target frame
  SEQ r = 0 FOR 64
    SEQ c = 0 FOR 8
      SEQ o = 0 FOR 4
        INT column, row :
        REAL32 value :
        SEQ
          --{{{ determine col value
          column := ((c << 6) + o) + (column.position << 2) + 1
          IF
            column < 256
              SKIP
            TRUE
              column := 513 - column
          --}}}
          --{{{ determine row value
          row := ((first.row << 2) + r) + 1
          IF
            row < 256
              SKIP
            TRUE
              row := 513 - row
          --}}}
          value := REAL32 ROUND( (row * column) )
          Frame[ max.frames-1 ][ ((r/\3) << 2) + o ][ r >> 2 ][ c ] := value
        --}}}
      WHILE TRUE
        SEQ
          --{{{ get command and send up and accross
          fromPrev ? length::message
          PAR
            toUp ! length::message
            IF
              column.position < 15
                toNext ! length::message
              TRUE
                SKIP
            --}}}
          CASE command
            --{{{ c.set.target
            c.set.target
            SEQ
              SendFrame( fromUp, toDown, Frame[ params[0] ][ params[1] ],
                params[2], params[3] )
            IF
              column.position = 15
                toPrev ! 0 (BYTE)
              TRUE

```

```
        BYTE synch :
        SEQ
            fromNext ? synch
            toPrev ! synch
--}}}
--{{{ c.target.row
c.target.row
    GetTargetRow( Frame[ params[0] ][ params[1] ], params[2],
                  [params FROM 3 FOR 128] )
--}}}
```

:

4.1.27. TrackDisplay.occ

```

PROC TrackDisplay ( CHAN OF ANY fromSP, toSP,
                    fromPrev, toPrev, fromNext, toNext )

--{{{ libraries
#INCLUDE "s_header.inc"
#INCLUDE "g_header.inc"
#INCLUDE "crtc.inc"

#USE "convert.lib"
#USE "graphics.lib"
--#USE "extrio.lib"

--}}}
--{{{ display constants
VAL width      IS      640 :
VAL height     IS      480 :
VAL line.frequency IS    60000 :
VAL frame.rate  IS      90 :
VAL pixel.clock IS 64000000 :
VAL interlace   IS     FALSE :
--}}}
--{{{ fonts
#INCLUDE "sys6.inc"
--}}}
--{{{ place system variables
[(20*65536)+1280] BYTE screen.map :
PLACE screen.map AT screen.int.address :

INT DisplayStart :
PLACE DisplayStart AT DisplayStart.address :

INT EventMode :
PLACE EventMode AT EventMode.address :

INT SysReady :
PLACE SysReady AT (#00080000 >< (MOSTNEG INT)) >> 2 :

INT Ready :
PLACE Ready AT Ready.address :
--}}}
--{{{ set up multiple screens
VAL screen.offset IS [ #00000, #50000, #A0000, #F0000 ] :

VAL screen.address IS [ #00000, #14000, #28000, #3C000 ] :
--}}}
--{{{ place Event channel
CHAN OF ANY Event :
PLACE Event AT 8 :
--}}}
--{{{ constants
VAL screen.width  IS  640 :
VAL screen.height IS  480 :
VAL screen.size   IS  screen.width * screen.height :

VAL char.width    IS   18 :
VAL char.height   IS   33 :
--}}}
--{{{ procs
--{{{ spectrum
PROC spectrum ( CHAN OF ANY out )

SEQ
    SEQ i = 0 FOR 64
        set.colour( out, 0, i, i, 0, 31-(i>>1) )
        -- blue to red scale for 1 to 63
    
```

```

SEQ i = 0 FOR 64                                -- adding green and blue
  set.colour( out, 0, 64+i, 63, i, i )          --   for 64 to 127

SEQ i = 128 FOR 128                             -- green for 128-255
  set.colour( out, 0, i, 0, 63, 0 )
  set.colour( out, 0, 0, 0, 0, 0 )              -- black for 0
  set.colour( out, 0, 128, 30, 30, 30 )         -- grey for 128
:
--}}}
--{{{ center.string
PROC center.string ( [] INT window, [] BYTE screen,
  VAL INT cx, sy, VAL [] BYTE string,
  VAL [] INT font )

  INT width :
  SEQ
    string.width( font, string, width )
    window[ w.cursor.y ] := sy
    window[ w.cursor.x ] := cx - (width >> 1)
    write.string( window, screen, string, font )
  :
  --}}}
--{{{ place.string
PROC place.string ( [] INT window, [] BYTE screen,
  VAL INT sx, sy, VAL [] BYTE string,
  VAL [] INT font )

  SEQ
    window[ w.cursor.y ] := sy
    window[ w.cursor.x ] := sx
    write.string( window, screen, string, font )
  :
  --}}}
--{{{ EventProc
PROC EventProc ( CHAN OF ANY Event, in )

  INT synch, address :
  WHILE TRUE
    SEQ
      in ? address
      Ready := 1
      Event ? synch
      DisplayStart := address
      Ready := 0
    :
  --}}}
--{{{ Buffer
PROC Buffer ( CHAN OF ANY in, out )

  INT temp :
  SEQ
    WHILE TRUE
      SEQ
        in ? temp
        out ! temp
      :
    --}}}
--{{{ set.text.window
PROC set.text.window ( [] INT window,
  VAL [] BYTE string1, string2, VAL [] INT font,
  VAL INT start.x, start.y, size.y )

  SEQ
    string.width( font, string1, window[ w.start.x ] )
    string.width( font, string2, window[ w.size.x ] )
    window[ w.start.x ] := window[ w.start.x ] + start.x
    window[ w.size.x ] := window[ w.size.x ] + 1

```

```

window[ w.start.y ] := start.y
window[ w.size.y ] := size.y + 1
window[ w.start ] := (screen.width * start.y) + window[ w.start.x ]
window[ w.size ] := screen.width * window[ w.size.y ]
window[ w.pixels.line ] := screen.width
window[ w.foreground.color ] := 255
window[ w.background.color ] := 0
window[ w.cursor.x ] := 0
window[ w.cursor.y ] := 0
:
--}}}
--{{{ display.text
PROC display.text ( [ ] INT window, [ ] BYTE screen,
                    VAL [ ] BYTE text, VAL [ ] INT font )

    s IS [screen FROM window[ w.start ] FOR window[ w.size ] ] :
    SEQ
        window[ w.cursor.x ] := 0
        window[ w.cursor.y ] := 0
        write.string( window, s, text, font )
    :
--}}}
--{{{ place.numbers
PROC place.numbers ( [ ] [ ] BYTE screen, [ ] [ ] [ ] BYTE char.array,
                    VAL [ ] BYTE string,
                    VAL INT start.x, start.y, size.x, size.y )

    INT x :
    SEQ
        x := start.x
        SEQ i = 0 FOR SIZE string
            VAL char IS INT string[i] :
            SEQ
                IF
                    --{{{ display space character
                    char = (INT ' ')
                    VAL source IS char.array[11] :
                    MOVE2D( source, 0, 0, screen, x, start.y, size.x, size.y )
                    --}}}
                    --{{{ display decimal character
                    char = (INT '.')
                    VAL source IS char.array[10] :
                    MOVE2D( source, 0, 0, screen, x, start.y, size.x, size.y )
                    --}}}
                    --{{{ display number character
                    TRUE
                    VAL source IS char.array[char - (INT '0')] :
                    MOVE2D( source, 0, 0, screen, x, start.y, size.x, size.y )
                    --}}}
                    x := x + size.x
                :
            --}}}
        --{{{ MAX
        REAL32 FUNCTION MAX ( VAL REAL32 a, b )

        REAL32 r :
        VALOF
            IF
                a > b
                r := a
            TRUE
                r := b
        RESULT r
        :
        --}}}
        --{{{ MIN
        REAL32 FUNCTION MIN ( VAL REAL32 a, b )

```

```

REAL32 r :
VALOF
  IF
    a < b
    r := a
  TRUE
    r := b
  RESULT r
:
--}}}
--{{{ IMAX
INT FUNCTION IMAX ( VAL INT a, b )

  INT r :
  VALOF
    IF
      a > b
      r := a
    TRUE
      r := b
    RESULT r
:
--}}}
--{{{ IMIN
INT FUNCTION IMIN ( VAL INT a, b )

  INT r :
  VALOF
    IF
      a < b
      r := a
    TRUE
      r := b
    RESULT r
:
--}}}
--}}}
--{{{ variables
[w.length] INT window :
[4][w.length] INT text.window :
[12][char.height][char.width] BYTE number :

CHAN OF ANY synch, synchl :
--}}}
VAL font IS SYS6 :
SEQ
  --{{{ initialize
  set.B408( 0, 0, 0, 0, 0 )
  --{{{ set up B409
  set.timing( toPrev, width, height, line.frequency,
              frame.rate, pixel.clock, interlace )
  spectrum( toPrev )
  --}}}
  --{{{ generate display table for numbers
  [w.length] INT window :
  [12][char.width*char.height] BYTE n RETYPES number :
  SEQ
    SEQ i = 0 FOR 12
      SEQ j = 0 FOR char.width*char.height
        n[i][j] := 0 (BYTE)
      window := [ 0, char.width*char.height, char.width,
                  0, 0, char.width, char.height, 255, 0, 0, 0 ]
      SEQ i = 0 FOR 10
        display.text( window, n[i], [ BYTE( i+(INT '0') ) ], font )
        display.text( window, n[10], ".", font )
        display.text( window, n[11], " ", font )

```

```

--}}
--{{{ initial display
[] INT int.screen RETYPES screen.map :
SEQ i = 0 FOR (20*65536)/4
  int.screen[i] := 0

window := [ 0, screen.size, screen.width, 0, 0,
            screen.width, screen.height, 255, 0, 0, 0 ]

screen IS [screen.map FROM screen.offset[0] FOR screen.size] :
SEQ
  --{{{ draw first window
  draw.line( window, screen, 31, 111, 288, 111, 255(BYTE) )
  draw.line( window, screen, 288, 111, 288, 368, 255(BYTE) )
  draw.line( window, screen, 31, 368, 288, 368, 255(BYTE) )
  draw.line( window, screen, 31, 111, 31, 368, 255(BYTE) )
  --}}}
  --{{{ draw second window
  draw.line( window, screen, 351, 111, 608, 111, 255(BYTE) )
  draw.line( window, screen, 608, 111, 608, 368, 255(BYTE) )
  draw.line( window, screen, 351, 368, 608, 368, 255(BYTE) )
  draw.line( window, screen, 351, 111, 351, 368, 255(BYTE) )
  --}}}
  --{{{ draw text
  SEQ
    center.string( window, screen, 320, 1, "FPA Seeker Emulator", font )
    center.string( window, screen, 160, 70, "Raw FPA Image", font )
    center.string( window, screen, 480, 70, "Processed Image", font )

    place.string( window, screen, 0, 400, "Range: ", font )
    place.string( window, screen, 0, 440, "Sim Time: ", font )
    place.string( window, screen, 350, 400, "Frame Rate:", font )
    place.string( window, screen, 350, 440, "Frame Number:", font )

    set.text.window( text.window[0], "Range:", "0123456.789", font, 16,400,
32 )
    set.text.window( text.window[1], "Sim Time: ", "123.456", font, 16,440,
32 )
    set.text.window( text.window[2], "Frame Rate:  ", "123", font, 350,400,
32 )
    set.text.window( text.window[3], "Frame Number: ", "123", font, 350,440,
32 )

    --{{{ COMMENT place initial numbers
    --:::A 0 0
    --{{{ place initial numbers
    --[screen.height][screen.width] BYTE s2 RETYPES screen :
    --SEQ
    --place.numbers( s2, number, " 0.000", text.window[0][w.start.x],
    --               text.window[0][w.start.y], char.width, char.height )
    --place.numbers( s2, number, " 0.000", text.window[1][w.start.x],
    --               text.window[1][w.start.y], char.width, char.height )
    --place.numbers( s2, number, " 0", text.window[2][w.start.x],
    --               text.window[2][w.start.y], char.width, char.height )
    --place.numbers( s2, number, " 0", text.window[3][w.start.x],
    --               text.window[3][w.start.y], char.width, char.height )
    --}}}
    --}}}
  --}}}
  [screen.map FROM screen.offset[1] FOR screen.size] := screen
  [screen.map FROM screen.offset[2] FOR screen.size] := screen
  [screen.map FROM screen.offset[3] FOR screen.size] := screen
--}}}
set.B408( 0, 0, 0, 1, 0 )
--}}}
PRI PAR
  --{{{ synchronize display to event

```

```

PAR
    EventProc( Event, synchl )
    Buffer( synch, synchl )
--}}}
--{{{ run system
--{{{ variables
INT load :
INT frame :
REAL32 range, time, rate :
INT len :
[12] BYTE string string1 :

--{{{ command variables
BYTE length :
[max.message] INT message :
command IS message[0] :
params IS [message FROM 1 FOR max.message-1] :
[] REAL32 r.params RETYPES params :
--}}}
--}}}
SEQ
    load := 0
    WHILE TRUE
        SEQ
            fromSP ? length::message
            IF
                --{{{ read graphics display
                command = c.read.graphics
                who IS params[0] :
                CASE who
                    track.display
                    --{{{
                    screen IS [screen.map FROM screen.offset[load] FOR screen.size
] :
                    [screen.height][screen.width] BYTE s2 RETYPES screen :
                    SEQ
                        toSP ! screen.height
                        SEQ i = 0 FOR screen.height
                        toSP ! screen.width::s2[i]
                    --}}}
                    display.palette
                    --{{{
                    SKIP
                    --}}}
                ELSE
                    --{{{ image.display 0 or 1
                    INT number.of.transfers :
                    INT width :
                    [maxGraphicBuffer] BYTE graphicsBuffer :
                    SEQ
                        toNext ! length::message
                        fromNext ? number.of.transfers
                        toSP ! number.of.transfers
                        SEQ i = 0 FOR number.of.transfers
                        SEQ
                            fromNext ? width::graphicsBuffer
                            toSP ! width::graphicsBuffer
                    --}}}
                --}}}
                --{{{ otherwise, normal processing
                TRUE
                SEQ
                    --{{{ place range, time, frame number, and frame rate on screen
                    screen IS [screen.map FROM screen.offset[load] FOR screen.size ]
:
                    range IS r.params[1] :
                    time IS r.params[2] :

```

```

frame IS params[3] :
rate IS r.params[4] :
SEQ
  IF
    frame >= 0
    --{{{ display numbers
    [screen.height][screen.width] BYTE s2 RETYPES screen :
    SEQ
      --{{{ range
      range := MAX( 0.001(REAL32), MIN( 9999999.999(REAL32),
range ) )
      REAL32TOSTRING( len, string, range, 7, 3 )
      place.numbers( s2, number, [string FROM 1 FOR 11],
text.window[0][w.start.x],
      text.window[0][w.start.y], 16, 32 )
      --}}}
      --{{{ time
      time := MAX( 0.001(REAL32), MIN( 999.999(REAL32), time )
)
      REAL32TOSTRING( len, string, time, 3, 3 )
      place.numbers( s2, number, [string FROM 1 FOR 7],
text.window[1][w.start.x],
      text.window[1][w.start.y], 16, 32 )
      --}}}
      --{{{ frame rate
      rate := MAX( 0.0(REAL32), MIN( 999.0(REAL32), rate ) )
      INTTOSTRING( len, string1, INT ROUND rate )
      [string FROM 0 FOR 3] := " "
      [string FROM 3-len FOR len] := [string1 FROM 0 FOR len]
      place.numbers( s2, number, [string FROM 0 FOR 3],
text.window[2][w.start.x],
      text.window[2][w.start.y], 16, 32 )
      --}}}
      --{{{ frame number
      frame := IMAX( 0, IMIN( 999, frame ) )
      INTTOSTRING( len, string1, frame )
      [string FROM 0 FOR 3] := " "
      [string FROM 3-len FOR len] := [string1 FROM 0 FOR len]
      place.numbers( s2, number, [string FROM 0 FOR 3],
text.window[3][w.start.x],
      text.window[3][w.start.y], 16, 32 )
      --}}}
    --}}}
    TRUE
    --{{{ clear the text windows
    SEQ i = 0 FOR 4
      win IS text.window[i] :
      clear.window( win, [screen FROM win[ w.start ] FOR win[
w.size ]] )
    --}}}
  --}}}
  --{{{ update screen
  IF
    params[0] = 1
    synch ! screen.address[load]
    TRUE
    DisplayStart := screen.address[load]
  --}}}
  load := (load + 1) /\ 3
  --}}}
  --}}}
  :

```

4.1.28. XBar.occ

```

PROC XBar (CHAN OF ANY fromGuidance, toGuidance)
  #INCLUDE "s_header.inc"
  --{{{ hardware values
  VAL XbarBase IS #0800 :

  PORT OF INT xbar.data :
  PLACE xbar.data AT XbarBase :

  PORT OF INT xbar.status :
  PLACE xbar.status AT XbarBase + 1 :

  CHAN OF INT Event :
  PLACE Event AT 8 :
  --}}}
  --{{{ constants
  VAL local.c.guidance.initialize IS 0 :
  VAL local.c.guidance.run IS 1 :
  VAL one.millisecond IS 1000 : -- ticks of high-priority clock
  --}}}
  --{{{ procs
  --{{{ INT.to.xbar (VAL INT value)
  PROC INT.to.xbar (VAL INT value)
    INT test.status :
    SEQ
      xbar.status ? test.status
      WHILE ((test.status /\ #0001) = 1)
        xbar.status ? test.status
        xbar.data ! value
      :
    --}}}
  --{{{ INT.from.xbar (INT value)
  PROC INT.from.xbar (INT value)
    INT signal :
    SEQ
      Event ? signal
      xbar.data ? value
    :
  --}}}

  --{{{ vector.to.xbar (VAL []INT value)
  PROC vector.to.xbar (VAL []INT value)
    SEQ i = 0 FOR SIZE(value)
      INT.to.xbar (value[i])
    :
  --}}}
  --{{{ vector.from.xbar ([]INT value)
  PROC vector.from.xbar ([]INT value)
    SEQ i = 0 FOR SIZE(value)
      INT.from.xbar (value[i])
    :
  --}}}
  --}}}
  --{{{ variables
  INT last.xbar.transfer :
  BOOL received.first.value :
  TIMER clock :
  --}}}
  PRI PAR
    SEQ
      --{{{ clear input buffer at start up
      INT dummy :
      xbar.data ? dummy
      --}}}
      WHILE TRUE

```

```

INT32 command :
SEQ
  fromGuidance ? command
CASE (INT command)
  --{{{ initialize
  c.guidance.initialize
  INT dummy :
  SEQ
    received.first.value := FALSE
  --}}}
  --{{{ run
  c.guidance.run
  [6]INT data.vector : --REAL32 time, shift.x, shift.y,
  SEQ
    --IF
    -- received.first.value
    -- SEQ -- force crossbar transfers to execute at real-time
rates
    -- clock ? AFTER last.xbar.transfer PLUS one.millisecond
    -- TRUE
    -- received.first.value := TRUE
    vector.from.xbar (data.vector)
    clock ? last.xbar.transfer
    toGuidance ! data.vector
  --}}}
  SKIP
:
```

4.2. Include Files

4.2.1. CRTC.inc

```
--{{{  crt
PROTOCOL CRTC
CASE
    crt.init      ;  INT16; INT16; INT32; INT16; INT32; BOOL
    crt.color     ;  INT16; INT16; INT16; INT16; INT16
    crt.initLUT   ;  INT16; INT16
    crt.stop
:

VAL channel.A IS 0 (INT16):
VAL channel.B IS 1 (INT16):
VAL channel.C IS 2 (INT16):
--}}}
```

4.2.2. G_Header.inc

```
--{{{ system constants
VAL bpw          IS 4 :
VAL bpw.shift    IS 2 :
VAL mint         IS MOSTNEG INT :

VAL screen.int.address    IS (#80100000 >< mint) >> bpw.shift :
VAL DisplayStart.address IS (#00000000 >< mint) >> bpw.shift :
VAL InterlaceEnable.address IS (#000C0000 >< mint) >> bpw.shift :
VAL EventMode.address     IS (#00100000 >< mint) >> bpw.shift :
VAL OutputEnable.address  IS (#00140000 >< mint) >> bpw.shift :
VAL Ready.address        IS (#00040000 >< mint) >> bpw.shift :
--}}}

--{{{ window constants
VAL w.start      IS 0 :
VAL w.size       IS 1 :
VAL w.pixels.line IS 2 :
VAL w.start.x    IS 3 :
VAL w.start.y    IS 4 :
VAL w.size.x     IS 5 :
VAL w.size.y     IS 6 :
VAL w.foreground.color IS 7 :
VAL w.background.color IS 8 :
VAL w.cursor.x   IS 9 :
VAL w.cursor.y   IS 10 :
VAL w.length     IS 11 :
--}}}

--{{{ text drawing modes
VAL normal.mode    IS 0 :
VAL foreground.mode IS 4 :
VAL and.mode       IS 1 :
VAL or.mode        IS 2 :
VAL xor.mode       IS 3 :
--}}}

--{{{ window decisions
VAL in.range       IS 0 :
VAL part.inrange   IS 1 :
VAL not.inrange    IS 2 :
--}}}

--{{{ font file format
VAL dfVersion.p    IS 0 : --2
VAL dfSize.p       IS 2 : --4
VAL dfCopyright.p  IS 6 : --60
VAL dfType.p       IS 66 : --2
VAL dfPoints.p     IS 68 : --2
VAL dfVertRes.p    IS 70 : --2
VAL dfHorizRes.p   IS 72 : --2
VAL dfAscent.p     IS 74 : --2
VAL dfInternalLeading.p IS 76 : --2
VAL dfExternalLeading.p IS 78 : --2
VAL dfItalic.p     IS 80 : --1
VAL dfUnderline.p  IS 81 : --1
VAL dfStrikeOut.p  IS 82 : --1
VAL dfWeight.p     IS 83 : --2
VAL dfCharSet.p    IS 85 : --1
VAL dfPixWidth.p   IS 86 : --2
VAL dfPixHeight.p  IS 88 : --2
VAL dfPitchAndFamily.p IS 90 : --1
VAL dfAvgWidth.p   IS 91 : --2
VAL dfMaxWidth.p   IS 93 : --2
VAL dfFirstChar.p  IS 95 : --1
VAL dfLastChar.p   IS 96 : --1
VAL dfDefaultChar.p IS 97 : --1
VAL dfBreakChar.p  IS 98 : --1
VAL dfWidthBytes.p IS 99 : --2
```

```
VAL dfDevice.p          IS 101 : --4
VAL dfFace.p            IS 105 : --4
VAL dfBitsPointer.p     IS 109 : --4
VAL dfBitsOffset.p      IS 113 : --4
VAL CharTable.p         IS 118 : --
--)}}
--{{{ font specification offsets
VAL fs.PixWidth         IS 0 :
VAL fs.PixHeight        IS 1 :
VAL fs.FirstChar        IS 2 :
VAL fs.LastChar         IS 3 :
VAL fs.BitsOffset       IS 4 :
VAL fs.size             IS 5 :
--}}}
```

4.2.3. S_Header.inc

```
--{{{ misc. constants
VAL max.message      IS 240 :
VAL max.frames       IS 232 :
VAL max.sim.frames   IS 2000 :
--}}}

--{{{ commands
-- Controller commands
VAL c.start.frame      IS 0 : -- frame
VAL c.run.single       IS 1 :
VAL c.run.continuous   IS 2 :
VAL c.frame.rate       IS 3 : -- start; frames; [frames] data
VAL c.frame.time       IS 4 : -- start; frames; [frames] data
VAL c.frame.range      IS 5 : -- start; frames; [frames] data
VAL c.sim.position     IS 6 : -- variable; start; frames; [] data
VAL c.sim.start.frames IS 7 : -- first.frame; last.frame
VAL c.test.controller  IS 8 :
VAL c.restart          IS 9 :
VAL c.set.calibration  IS 10 : -- back.level0; back.level1;
                                -- sp.level0; sp.level1

-- GTSEI commands
VAL c.set.crossbar      IS 256 : -- shift

-- Target commands
VAL c.set.target        IS 512 : -- frame; subpixel; row.shift; col.shift
VAL c.target.row        IS 513 : -- frame; subpixel; row; [128] data
VAL c.test.target       IS 514 :

-- Background commands
VAL c.set.background    IS 768 : -- frame
VAL c.background.row    IS 769 : -- frame; row; [128] data
VAL c.gain.row          IS 770 : -- row; [128] data
VAL c.offset.row        IS 771 : -- row; [128] data
VAL c.global.scale      IS 772 : -- scale
VAL c.test.background   IS 773 :
VAL c.calibration.frame IS 774 : -- calibration.level

-- Signal Processing commands
VAL c.sp.frame          IS 1024 : -- calibration.frame; calibration.level;
                                -- lower.threshold; upper.threshold;
                                -- frame.range; frame.time;
                                -- frame.number; frame.rate;
                                -- [6]seeker.target.position

-- Guidance commands
VAL c.guidance.initialize IS 1280 :
VAL c.guidance.set.mode   IS 1281 : -- mode
VAL c.guidance.run        IS 1282 : -- FPA; frame.range; frame.time;
                                -- column.hot.spot; row.hot.spot;
                                -- [3]seeker.position;
                                -- [3]target.position

-- Track Display commands
VAL c.display.info      IS 1536 : -- FPA; frame.range; frame.time;
                                -- frame.number; frame.rate

-- command for reading graphics buffers or palette
VAL c.read.graphics     IS 2000 : -- display
--}}}

--{{{ commands for continuous frame operation
VAL cc.exit             IS 0 :
VAL cc.shift.image      IS 1 : -- column.shift; row.shift
--}}}

--{{{ variables in position
VAL p.seeker.x          IS 0 :
```

```
VAL p.seeker.y      IS 1 :
VAL p.seeker.z      IS 2 :
VAL p.target.x      IS 3 :
VAL p.target.y      IS 4 :
VAL p.target.z      IS 5 :
VAL p.length        IS 6 :
--}}
--{{{ guidance mode constants
VAL gm.none         IS 0 :
VAL gm.external     IS 1 :
VAL gm.internal     IS 2 :
--}}
--{{{ graphics read constants
VAL image.display.0 IS 0 :
VAL image.display.1 IS 1 :
VAL track.display   IS 2 :
VAL display.palette IS 3 :

VAL maxGraphicBuffer IS 640 :
--}}}
```

4.2.4. Sys6.inc

```
VAL SYS6 IS [#44DA0200, #6F430000, #69727970, #20746867,
#20296328, #37383931, #6F535A20, #43207466, #6F70726F, #69746172, #6E6F ,
#0 , #0 , #0 , #0 , #0 , #0 , #12C00008 ,
#18012C , #2 , #90000000, #10FF01 , #10300020, #1001000 , #1F2DFF ,
#47602 , #4A800 , #0 , #4DA00 , #100000 , #1004DA , #10051A ,
#10055A , #10059A , #1005DA , #10061A , #10065A , #10069A , #1006DA ,
#10071A , #10075A , #10079\ , #1007DA , #10081A , #10085A , #10089A ,
#1008DA , #10091A , #10095A , #10099A , #1009DA , #100A1A , #100A5A ,
#100A9A , #100ADA , #100B1A , #100B5A , #100B9A , #100BDA , #100C1A ,
#100C5A , #100C9A , #100CDA , #100D1A , #100D5A , #100D9A , #100DDA ,
#100E1A , #100E5A , #100E9A , #100EDA , #100F1A , #100F5A , #100F9A ,
#100FDA , #10101A , #10105A , #10109A , #1010DA , #10111A , #10115A ,
#10119A , #1011DA , #10121A , #10125A , #10129A , #1012DA , #10131A ,
#10135A , #10139A , #1013DA , #10141A , #10145A , #10149A , #1014DA ,
#10151A , #10155A , #10159A , #1015DA , #10161A , #10165A , #10169A ,
#1016DA , #10171A , #10175A , #10179A , #1017DA , #10181A , #10185A ,
#10189A , #1018DA , #10191A , #10195A , #10199A , #1019DA , #101A1A ,
#101A5A , #101A9A , #101ADA , #101B1A , #101B5A , #101B9A , #101BDA ,
#101C1A , #101C5A , #101C9A , #101CDA , #101D1A , #101D5A , #101D9A ,
#101DDA , #101E1A , #101E5A , #101E9A , #101EDA , #101F1A , #101F5A ,
#101F9A , #101FDA , #10201A , #10205A , #10209A , #1020DA , #10211A ,
#10215A , #10219A , #1021DA , #10221A , #10225A , #10229A , #1022DA ,
#10231A , #10235A , #10239A , #1023DA , #10241A , #10245A , #10249A ,
#1024DA , #10251A , #10255A , #10259A , #1025DA , #10261A , #10265A ,
#10269A , #1026DA , #10271A , #10275A , #10279A , #1027DA , #10281A ,
#10285A , #10289A , #1028DA , #10291A , #10295A , #10299A , #1029DA ,
#102A1A , #102A5A , #102A9A , #102ADA , #102B1A , #102B5A , #102B9A ,
#102BDA , #102C1A , #102C5A , #102C9A , #102CDA , #102D1A , #102D5A ,
#102D9A , #102DDA , #102E1A , #102E5A , #102E9A , #102EDA , #102F1A ,
#102F5A , #102F9A , #102FDA , #10301A , #10305A , #10309A , #1030DA ,
#10311A , #10315A , #10319A , #1031DA , #10321A , #10325A , #10329A ,
#1032DA , #10331A , #10335A , #10339A , #1033DA , #10341A , #10345A ,
#10349A , #1034DA , #10351A , #10355A , #10359A , #1035DA , #10361A ,
#10365A , #10369A , #1036DA , #10371A , #10375A , #10379A , #1037DA ,
#10381A , #10385A , #10389A , #1038DA , #10391A , #10395A , #10399A ,
#1039DA , #103A1A , #103A5A , #103A9A , #103ADA , #103B1A , #103B5A ,
#103B9A , #103BDA , #103C1A , #103C5A , #103C9A , #103CDA , #103D1A ,
#103D5A , #103D9A , #103DDA , #103E1A , #103E5A , #103E9A , #103EDA ,
#103F1A , #103F5A , #103F9A , #103FDA , #10401A , #10405A , #10409A ,
#1040DA , #10411A , #10415A , #10419A , #1041DA , #10421A , #10425A ,
#10429A , #1042DA , #10431A , #10435A , #10439A , #1043DA , #10441A ,
#10445A , #449A , #0 , #0 , #0 , #0 , #0 ,
#0 , #0 , #0 , #0 , #0 , #0 , #0 ,
#74737953, #36206D65, #34206400, #0 , #0 , #0 , #0 ,
#0 , #0 , #0 , #0 , #0 , #0 , #1C1E0F07,
#76763838, #70707676, #77777070, #1E1C3B3B, #70F , #0 , #0 ,
#3878F0E0, #6E6E1C1C, #E0E6E6E , #EEEE0E0E, #7838DCDC, #E0F0 , #0 ,
#0 , #1F1F0F07, #79793F3F, #7F7F7979, #78787F7F, #1F1F3C3C, #70F ,
#0 , #0 , #F8F8F0E0, #9E9EFCFC, #FEFE9E9E, #1E1EFEFE, #F8F83C3C,
#E0F0 , #0 , #0 , #7F7F3E1C, #7F7F7F7F, #1F1F3F3F, #7070F0F ,
#1010303 , #0 , #0 , #0 , #7F7F3E1C, #FFFFFFF, #FCFCFEFE,
#F0F0F8F8, #C0C0E0E0, #8080 , #0 , #0 , #1010000 , #7070303 ,
#1F1F0F0F, #7070F0F , #1010303 , #0 , #0 , #0 , #C0C08080,
#F0F0E0E0, #FCFCF8F8, #F0F0F8F8, #C0C0E0E0, #8080 , #0 , #0 ,
#7070301 , #3030707 , #7F7F7D39, #397D7F7F, #1010101 , #303 , #0 ,
#0 , #F0F0E0C0, #E0E0F0F0, #FFFDFFCE, #CEDFFFFFF, #C0C0C0C0, #E0E0 ,
#0 , #0 , #1010000 , #7070303 , #1F1F0F0F, #1D3D3F3F, #1010101 ,
#303 , #0 , #0 , #C0C08080, #F0F0E0E0, #FCFCF8F8, #DCDEFEFE,
#C0C0C0C0, #E0E0 , #0 , #0 , #0 , #0 , #7070703 ,
#3070707 , #0 , #0 , #0 , #0 , #0 , #0 ,
#E0E0E0C0, #C0E0E0E0, #0 , #0 , #0 , #0 , #FFFFFFF,
#FFFFFFF, #F8F8F8FC, #FCF8F8F8, #FFFFFFF, #FFFFFFF, #FFFFFFF, #FFFF ,
#FFFFFFF, #FFFFFFF, #1F1F1F3F, #3F1F1F1F, #FFFFFFF, #FFFFFFF, #FFFFFFF,
#FFFF , #0 , #0 , #1C1E0F07, #38383838, #70F1E1C , #0 ,
```

```

#0      ,#0      ,#0      ,#0      ,#3878F0E0,#1C1C1C1C,#E0F07838,
#0      ,#0      ,#0      ,#FFFFFFF,#FFFFFFF,#E3E1F0F8,#C7C7C7C7,
#F8F0E1E3,#FFFFFFF,#FFFFFFF,#FFFFF,#FFFFFFF,#C7870F1F,
#E3E3E3E3,#1F0F87C7,#FFFFFFF,#FFFFFFF,#FFFFF,#0      ,#303      ,
#1F0F0000,#70703838,#38387070,#F1F      ,#0      ,#0      ,#0      ,
#3E1EFEFE,#E6E67676,#7070E0E0,#E0E07070,#80C0      ,#0      ,#0      ,
#E0E0703      ,#1C1C1C1C,#3070E0E      ,#1010101      ,#1010707      ,#101      ,#0      ,
#0      ,#3838F0E0,#1C1C1C1C,#E0F03838,#C0C0C0C0,#C0C0F0F0,#C0C0      ,
#0      ,#0      ,#3030303      ,#3030303      ,#3030303      ,#1F0F0303,#F1F3F3F      ,
#0      ,#0      ,#0      ,#9C9CF8F0,#80808E8E,#80808080,#80808080,
#808080      ,#0      ,#0      ,#0      ,#7070707      ,#7070707      ,#7070707      ,
#3F1F0707,`E3F7F7F,#0      ,#0      ,#07FFFF      ,#707FFF      ,
#7070707      ,#7070707      ,#7F7F3F1F,#1E3F      ,#0      ,#0      ,#39390101,
#70F1D1D      ,#7C7C1C0E,#F070E1C      ,#39391D1D,#101      ,#0      ,#0      ,
#CECEC0C0,#F0F8DCDC,#1F1F1C38,#F8F0381C,#CECEDCDC,#C0C0      ,#0      ,
#0      ,#3E3C3830,#3F3F3F3F,#3F3F3F3F,#3F3F3F3F,#3C3E3F3F,#3038      ,
#0      ,#0      ,#0      ,#E0C08000,#FCFCF8F0,#C0E0F0F8,#80      ,
#0      ,#0      ,#0      ,#0      ,#7030100      ,#3F3F1F0F,#3070F1F      ,
#1      ,#0      ,#0      ,#0      ,#7C3C1C0C,#FCFCFCFC,#FCFCFCFC,
#FCFCFCFC,#3C7CFCFC,#C1C      ,#0      ,#0      ,#1010000      ,#7070303      ,
#1010F0F      ,#F0F0101      ,#3030707      ,#101      ,#0      ,#0      ,#C0C08080,
#F0F0E0E0,#C0C0F8F8,#F8F8C0C0,#E0E0F0F0,#8080C0C0,#0      ,#0      ,
#3C3C3C18,#3C3C3C3C,#183C3C3C,#18181818,#3C180000,#183C      ,#0      ,
#0      ,#3C3C3C18,#3C3C3C3C,#183C3C3C,#18181818,#3C180000,#183C      ,
#0      ,#0      ,#737B3F1F,#73737373,#7B737373,#3031F3F      ,#3030303      ,
#303      ,#0      ,#0      ,#9C9CFFFF      ,#9C9C9C9C,#9C9C9C9C,#9C9C9C9C,
#9C9C9C9C,#9C9C      ,#0      ,#0      ,#383C1F0F,#E1C3838      ,#1C1C0F07,
#70F      ,#1E1C0000,#70F      ,#0      ,#0      ,#3878F0E0,#0      ,
#3838F0E0,#3870E0F0,#3C1C1C1C,#F0F8      ,#0      ,#0      ,#0      ,
#0      ,#0      ,#F0F0F0F      ,#F0F      ,#0      ,#0      ,#0      ,
#0      ,#0      ,#0      ,#F0F0F0F0,#F0F0      ,#0      ,#0      ,
#0      ,#1010000      ,#7070303      ,#1010F0F      ,#F0F0101      ,#3030707      ,#1F000101,
#0      ,#0      ,#C0C08080,#F0F0E0E0,#C0C0F8F8,#F8F8C0C0,#E0E0F0F0,
#FC80C0C0,#0      ,#0      ,#1010000      ,#7070303      ,#1010F0F      ,#1010101      ,
#1010101      ,#101      ,#0      ,#0      ,#C0C08080,#F0F0E0E0,#C0C0F8F8,
#C0C0C0C0,#C0C0C0C0,#C0C0      ,#0      ,#0      ,#1010000      ,#1010101      ,
#1010101      ,#F0F0101      ,#3030707      ,#101      ,#0      ,#0      ,#C0C00000,
#C0C0C0C0,#C0C0C0C0,#F8F8C0C0,#E0E0F0F0,#8080C0C0,#0      ,#0      ,
#0      ,#1010101      ,#7F7F0101,#1010101      ,#101      ,#0      ,#0      ,
#0      ,#0      ,#F0E0C080,#FEFEFCF8,#E0F0F8FC,#80C0      ,#0      ,
#0      ,#0      ,#0      ,#F070301      ,#7F7F3F1F,#70F1F3F      ,#103      ,
#0      ,#0      ,#0      ,#0      ,#80808080,#FEFE8080,#80808080,
#8080      ,#0      ,#0      ,#0      ,#0      ,#38380000,#38383838,
#3F3F      ,#0      ,#0      ,#0      ,#0      ,#0      ,#0      ,
#0      ,#FCFC      ,#0      ,#0      ,#0      ,#0      ,#0      ,
#E060000      ,#7F7F3E1E,#60E1E3E      ,#0      ,#0      ,#0      ,#0      ,
#0      ,#70600000,#FEFE7C78,#6070787C,#0      ,#0      ,#0      ,
#0      ,#0      ,#1010000      ,#7070303      ,#1F1F0F0F,#3F3F      ,#0      ,
#0      ,#0      ,#0      ,#80800000,#E0E0C0C0,#F8F8F0F0,#FCFC      ,
#0      ,#0      ,#0      ,#0      ,#3F3F0000,#F0F1F1F      ,#3030707      ,
#101      ,#0      ,#0      ,#0      ,#0      ,#0      ,#FCFC0000,#F0F0F8F8,
#C0C0E0E0,#8080      ,#0      ,#0      ,#0      ,#0      ,#0      ,
#0      ,#0      ,#0      ,#0      ,#0      ,#0      ,#0      ,
#0      ,#0      ,#0      ,#0      ,#0      ,#0      ,#0      ,
#F0F0F07      ,#F0F0F0F      ,#70F0F0F      ,#7070707      ,#F070000      ,#70F      ,#0      ,
#0      ,#80808000,#80808080,#808080      ,#0      ,#80000000,#80      ,
#0      ,#0      ,#E0E0E0E      ,#60606      ,#0      ,#0      ,#0      ,
#0      ,#0      ,#0      ,#70707070,#606060      ,#0      ,#0      ,
#0      ,#0      ,#0      ,#0      ,#E0E0000      ,#3F3F0E0E,#E0E0E0E      ,
#3F3F0E0E,#E0E0E0E      ,#0      ,#0      ,#0      ,#70700000,#FCFC7070,
#70707070,#FCFC7070,#70707070,#0      ,#0      ,#0      ,#1010101      ,
#393D1F0F,#F1F3D39      ,#1010101      ,#1011F1F      ,#101      ,#0      ,#0      ,
#C0C0C0C0,#C0C0FCFC,#FCF8C0C0,#DECECEDE,#C0C0F8FC,#C0C0      ,#0      ,
#0      ,#33333F1E,#1E3F      ,#1010000      ,#7070303      ,#1C1C0E0E,#3838      ,
#0      ,#0      ,#1C1C0E0E,#70703838,#C0C0E0E0,#8080      ,#66667E3C,
#3C7E      ,#0      ,#0      ,#1C1C0F07,#1C1C1C1C,#70F1D1D      ,#713B1F0F,
#38707070,#F1F      ,#0      ,#0      ,#E0E0C080,#E0E0E0E0,#8080C0C0,

```

#F89C0E00	#F870F0F0	#8EDE	#0	#0	#1030301	#303010F
#0	#0	#0	#0	#0	#0	#E0E0E0C0
#80C0E0	#0	#0	#0	#0	#0	#0
#7070301	#7070707	#7070707	#7070707	#7070707	#103	#0
#0	#80F0F0	#0	#0	#0	#80000000	#F0F0
#0	#0	#707	#0	#0	#0	#0
#707	#0	#0	#70F0E0C0	#70707070	#70707070	#70707070
#F0707070	#C0E0	#0	#0	#0	#E0E1C1C	#7F7F0707
#E0E0707	#1C1C	#0	#0	#0	#0	#70703838
#FEFEE0E0	#7070E0E0	#3838	#0	#0	#0	#0
#3030000	#3F3F0303	#3030303	#0	#0	#0	#0
#0	#80800000	#F8F88080	#80808080	#0	#0	#0
#0	#0	#0	#0	#0	#F070000	#703070F
#C0E	#0	#0	#0	#0	#0	#80000000
#8080E0	#0	#0	#0	#0	#3F3F0000	#0
#0	#0	#0	#0	#0	#0	#F8F80000
#0	#0	#0	#0	#0	#0	#0
#0	#0	#F070000	#70F	#0	#0	#0
#0	#0	#0	#80000000	#80	#0	#0
#0	#0	#1010000	#7070303	#1C1C0E0E	#3838	#0
#0	#1C1C0E0E	#70703838	#C0C0E0E0	#8080	#0	#0
#0	#0	#383C1F0F	#38383838	#39393838	#3E3E3B3B	#3C383C3C
#F1F	#0	#0	#1C3CF8F0	#7C7C3C3C	#9C9CDCDC	#1C1C1C1C
#3C1C1C1C	#F0F8	#0	#0	#F0F0703	#1010101	#1010101
#1010101	#1010101	#1F1F	#0	#0	#C0C0C0C0	#C0C0C0C0
#C0C0C0C0	#C0C0C0C0	#C0C0C0C0	#FCFC	#0	#0	#1C1E0F07
#0	#0	#7030100	#38381C0E	#3F3F	#0	#0
#1C3CF8F0	#1C1C1C1C	#70381C1C	#80C0E0	#0	#FCFC	#0
#0	#1C1E0F07	#0	#3030000	#0	#1E1C0000	#70F
#0	#0	#1C3CF8F0	#1C1C1C1C	#F0F0381C	#1C1C1C38	#3C1C1C1C
#F0F8	#0	#0	#3030100	#E0E0707	#38381C1C	#7F7F0707
#0	#0	#0	#0	#F0F0F0F0	#70707070	#70707070
#FEFE7070	#70707070	#7070	#0	#0	#38383F3F	#38383838
#3F3F	#0	#3C380000	#F1F	#0	#0	#F8F8
#0	#0	#1C3CF8F0	#1C1C1C1C	#3C1C1C1C	#F0F8	#0
#383C1F0F	#38383838	#3F3F3838	#38383838	#3C383838	#F1F	#0
#0	#3878F0E0	#0	#F8F00000	#1C1C1C3C	#3C1C1C1C	#F0F8
#0	#0	#383F3F	#0	#0	#0	#3030101
#303	#0	#0	#1C1CFCFC	#1C1C1C1C	#38381C1C	#E0E07070
#8080C0C0	#8080	#0	#0	#383C1F0F	#38383838	#F0F1C38
#3838381C	#3C383838	#F1F	#0	#0	#1C3CF8F0	#1C1C1C1C
#F0F0381C	#1C1C1C38	#3C1C1C1C	#F0F8	#0	#0	#383C1F0F
#38383838	#F1F3C38	#0	#1C1C0000	#70F	#0	#0
#1C3CF8F0	#1C1C1C1C	#FCFC1C1C	#1C1C1C1C	#3C1C1C1C	#F0F8	#0
#0	#0	#0	#70F0F07	#0	#F070000	#70F
#0	#0	#0	#0	#808000	#0	#80000000
#80	#0	#0	#0	#0	#70F0F07	#0
#F070000	#703070F	#C0E	#0	#0	#0	#808000
#0	#80000000	#808080	#0	#0	#0	#1010000
#7070303	#7070E0E	#1010303	#0	#0	#0	#0
#C0C0E0E0	#8080	#0	#C0C08080	#E0E0	#0	#0
#0	#0	#3F3F	#3F3F0000	#0	#0	#0
#0	#0	#0	#FCFC	#FCFC0000	#0	#0
#0	#0	#0	#3030707	#101	#0	#3030101
#707	#0	#0	#0	#80800000	#E0E0C0C0	#E0E07070
#8080C0C0	#0	#0	#0	#1C1E0F07	#0	#1000000
#3030303	#7030000	#307	#0	#0	#1C3CF8F0	#1C1C1C1C
#C0E07038	#80808080	#C0800000	#80C0	#0	#0	#1C1E0F07
#39383838	#3B3B3B3B	#38393B3B	#1E1C3838	#70F	#0	#0
#3878F0E0	#FCFC1C1C	#9C9C9C9C	#FCFC9C9C	#0	#F0F0	#0
#0	#E0E0703	#1C1C1C1C	#38383838</			


```

#0 ,#0 ,#0 ,#0 ,#0 ,#0 ,#3030101 ,
#E0E0707 ,#38381C1C ,#7F7F ,#0 ,#0 ,#0 ,#0 ,
#80800000 ,#E0E0C0C0 ,#38387070 ,#FCFC ,#0 ,#0 ,#0 ,
#0 ,#383C1F0F ,#38383838 ,#38383838 ,#38383838 ,#3C383838 ,#1010F1F ,
#707 ,#707 ,#1C3CF8F0 ,#0 ,#0 ,#0 ,#3C1C0000 ,
#C0C0F0F8 ,#E0E0C080 ,#80C0 ,#1C000000 ,#1C1C ,#38383838 ,#38383838 ,
#3C383838 ,#F1F ,#0 ,#0 ,#38000000 ,#3838 ,#1C1C1C1C ,
#1C1C1C1C ,#1C1C1C1C ,#FCFC ,#0 ,#0 ,#1000000 ,#703 ,
#383C1F0F ,#3F3F3838 ,#3C383838 ,#F1F ,#0 ,#0 ,#C0E07038 ,
#80 ,#1C3CF8F0 ,#FCFC1C1C ,#0 ,#F8F8 ,#0 ,#0 ,
#7030100 ,#1C0E ,#1F1F ,#1F0F0000 ,#3C38383C ,#F1F ,#0 ,
#0 ,#70E0C080 ,#1C38 ,#1C3CF8F0 ,#FCFC1C1C ,#1C1C1C1C ,#FCFC ,
#0 ,#0 ,#1C000000 ,#1C1C ,#1F1F ,#1F0F0000 ,#3C38383C ,
#F1F ,#0 ,#0 ,#38000000 ,#3838 ,#1C3CF8F0 ,#FCFC1C1C ,
#1C1C1C1C ,#FCFC ,#0 ,#0 ,#3070E1C ,#1 ,#1F1F ,
#1F0F0000 ,#3C38383C ,#F1F ,#0 ,#0 ,#80000000 ,#E0C0 ,
#1C3CF8F0 ,#FCFC1C1C ,#1C1C1C1C ,#FCFC ,#0 ,#0 ,#6060703 ,
#307 ,#1F1F ,#1F0F0000 ,#3C38383C ,#F1F ,#0 ,#0 ,
#6060E0C0 ,#C0E0 ,#1C3CF8F0 ,#FCFC1C1C ,#1C1C1C1C ,#FCFC ,#0 ,
#0 ,#0 ,#0 ,#383C1F0F ,#38383838 ,#3C383838 ,#1010F1F ,
#1010000 ,#0 ,#0 ,#0 ,#1C3CF8F0 ,#0 ,#3C1C0000 ,
#E0C0F0F8 ,#C0E07070 ,#0 ,#7030100 ,#1C0E ,#383C1F0F ,#3F3F3838 ,
#3C383838 ,#F1F ,#0 ,#0 ,#70E0C080 ,#1C38 ,#1C3CF8F0 ,
#FCFC1C1C ,#0 ,#F8F8 ,#0 ,#0 ,#1C000000 ,#1C1C ,
#383C1F0F ,#3F3F3838 ,#3C383838 ,#F1F ,#0 ,#0 ,#380C0000 ,
#3838 ,#1C3CF8F0 ,#FCFC1C1C ,#0 ,#F8F8 ,#0 ,#0 ,
#103070E ,#0 ,#383C1F0F ,#3F3F3838 ,#3C383838 ,#F1F ,#0 ,
#0 ,#C0800000 ,#70E0 ,#1C3CF8F0 ,#FCFC1C1C ,#0 ,#F8F8 ,
#0 ,#0 ,#1C000000 ,#1C1C ,#1010F0F ,#1010101 ,#1010101 ,
#1F1F ,#0 ,#0 ,#38000000 ,#3838 ,#C0C0C0C0 ,#C0C0C0C0 ,
#C0C0C0C0 ,#FCFC ,#0 ,#0 ,#7030100 ,#1C0E ,#1010F0F ,
#1010101 ,#1010101 ,#1F1F ,#0 ,#0 ,#70E0C080 ,#1C38 ,
#C0C0C0C0 ,#C0C0C0C0 ,#C0C0C0C0 ,#FCFC ,#0 ,#0 ,#103070E ,
#0 ,#1010F0F ,#1010101 ,#1010101 ,#1F1F ,#0 ,#0 ,
#C0800000 ,#70E0 ,#C0C0C0C0 ,#C0C0C0C0 ,#C0C0C0C0 ,#FCFC ,#0 ,
#0 ,#1C1C1C00 ,#7030000 ,#38381C0E ,#38383F3F ,#38383838 ,#3838 ,
#0 ,#0 ,#38383800 ,#E0C00000 ,#1C1C3870 ,#1C1CF0F0 ,#1C1C1C1C ,
#1C1C ,#0 ,#30000000 ,#70E0E07 ,#7030003 ,#38381C0E ,#38383F3F ,
#38383838 ,#3838 ,#0 ,#E0000000 ,#F03838F0 ,#E0C000E0 ,#1C1C3870 ,
#1C1CF0F0 ,#1C1C1C1C ,#1C1C ,#0 ,#0 ,#E070301 ,#3F3F001C ,
#38383838 ,#38383F3F ,#38383838 ,#3F3F ,#0 ,#E0000000 ,#80C0 ,
#FCFC0000 ,#0 ,#E0E0 ,#0 ,#FCFC ,#0 ,#0 ,
#0 ,#0 ,#3073E3C ,#3F1F0303 ,#73737373 ,#1C3E ,#0 ,
#0 ,#0 ,#9C9CF870 ,#FCFC9C9C ,#CC8C8080 ,#78FC ,
#0 ,#0 ,#7070301 ,#E0E0E0E ,#1C1C1C1C ,#38383F3F ,#70707070 ,
#7070 ,#0 ,#0 ,#E0E0FEFE ,#E0E0E0E0 ,#FCFCE0E0 ,#E0E0E0E0 ,
#E0E0E0E0 ,#FEFE ,#0 ,#0 ,#7030100 ,#1C0E ,#383C1F0F ,
#38383838 ,#3C383838 ,#F1F ,#0 ,#0 ,#70E0C080 ,#1C38 ,
#1C3CF8F0 ,#1C1C1C1C ,#3C1C1C1C ,#F0F8 ,#0 ,#0 ,#1C000000 ,
#1C1C ,#383C1F0F ,#38383838 ,#3C383838 ,#F1F ,#0 ,#0 ,
#38000000 ,#3838 ,#1C3CF8F0 ,#1C1C1C1C ,#3C1C1C1C ,#F0F8 ,#0 ,
#0 ,#103070E ,#0 ,#383C1F0F ,#38383838 ,#3C383838 ,#F1F ,
#0 ,#0 ,#C0800000 ,#70E0 ,#1C3CF8F0 ,#1C1C1C1C ,#3C1C1C1C ,
#F0F8 ,#0 ,#0 ,#7030100 ,#1C0E ,#38383838 ,#38383838 ,
#3C383838 ,#F1F ,#0 ,#0 ,#70E0C080 ,#1C38 ,#1C1C1C1C ,
#1C1C1C1C ,#1C1C1C1C ,#FCFC ,#0 ,#0 ,#103070E ,#0 ,
#38383838 ,#38383838 ,#3C383838 ,#F1F ,#0 ,#0 ,#C0800000 ,
#70E0 ,#1C1C1C1C ,#1C1C1C1C ,#1C1C1C1C ,#FCFC ,#0 ,#0 ,
#1C000000 ,#1C1C ,#1C383838 ,#E0E1C1C ,#3030707 ,#101 ,#3030101 ,
#707 ,#1C000000 ,#1C1C ,#E0E0E0E ,#1C1C1C1C ,#B8B83838 ,#E0E0F0F0 ,
#8080C0C0 ,#0 ,#1C1C1C ,#383C1F0F ,#38383838 ,#38383838 ,#3C383838 ,
#F1F ,#0 ,#0 ,#383838 ,#1C3CF8F0 ,#1C1C1C1C ,#1C1C1C1C ,
#3C1C1C1C ,#F0F8 ,#0 ,#0 ,#1C1C1C ,#38383838 ,#38383838 ,
#38383838 ,#3C383838 ,#F1F ,#0 ,#0 ,#383838 ,#1C1C1C1C ,
#1C1C1C1C ,#1C1C1C1C ,#3C1C1C1C ,#F0F8 ,#0 ,#0 ,#1010000 ,
#3F1F0101 ,#71717179 ,#1F3F7971 ,#1010101 ,#0 ,#0 ,#0 ,
#C3C00000 ,#FEFCC0C0 ,#C0C0C0CE ,#FCFECECO ,#C0C0C0C0 ,#0 ,#0 ,

```



```

#7070707 ,#7070707 ,#7070707 ,#38380707 ,#38383838 ,#38383838 ,#38383838 ,
#38383838 ,#38383838 ,#38383838 ,#38383838 ,#7073838 ,#7070707 ,#7070707 ,
#7070707 ,#7070707 ,#7070707 ,#7070707 ,#7070707 ,#38380707 ,#38383838 ,
#38383838 ,#38383838 ,#38383838 ,#38383838 ,#38383838 ,#38383838 ,#38383838 ,
#0 ,#0 ,#FFF0000 ,#FFF0000 ,#7070707 ,#7070707 ,#7070707 ,
#707 ,#0 ,#0 ,#F8F80000 ,#38383838 ,#38383838 ,#38383838 ,
#38383838 ,#7073838 ,#7070707 ,#7070707 ,#FFF0707 ,#FFF0000 ,#0 ,
#0 ,#0 ,#38380000 ,#38383838 ,#38383838 ,#38383838 ,#F8F83838 ,
#0 ,#0 ,#0 ,#7070000 ,#7070707 ,#7070707 ,#7070707 ,
#FFFF ,#0 ,#0 ,#0 ,#38380000 ,#38383838 ,#38383838 ,
#38383838 ,#F8F8 ,#0 ,#0 ,#0 ,#1010000 ,#1010101 ,
#1010101 ,#FFF0101 ,#FFF0101 ,#0 ,#0 ,#0 ,#C0C00000 ,
#C0C0C0C0 ,#C0C0C0C0 ,#C0C0C0C0 ,#C0C0C0C0 ,#0 ,#0 ,#0 ,
#0 ,#0 ,#0 ,#0 ,#101FFFF ,#1010101 ,#1010101 ,
#1010101 ,#101 ,#0 ,#0 ,#0 ,#C0C0C0C0 ,#C0C0C0C0 ,
#C0C0C0C0 ,#C0C0C0C0 ,#101C0C0 ,#1010101 ,#1010101 ,#1010101 ,#101 ,
#0 ,#0 ,#0 ,#C0C00000 ,#C0C0C0C0 ,#C0C0C0C0 ,#C0C0C0C0 ,
#FFFF ,#0 ,#0 ,#0 ,#1010000 ,#1010101 ,#1010101 ,
#1010101 ,#FFFF ,#0 ,#0 ,#0 ,#0 ,#C0C00000 ,#C0C0C0C0 ,
#C0C0C0C0 ,#C0C0C0C0 ,#FFFF ,#0 ,#0 ,#0 ,#0 ,#0 ,
#0 ,#0 ,#0 ,#101FFFF ,#1010101 ,#1010101 ,#1010101 ,
#101 ,#0 ,#0 ,#0 ,#C0C0FFFF ,#C0C0C0C0 ,#C0C0C0C0 ,
#C0C0C0C0 ,#101C0C0 ,#1010101 ,#1010101 ,#1010101 ,#1010101 ,#1010101 ,
#1010101 ,#1010101 ,#C0C00101 ,#C0C0C0C0 ,#C0C0C0C0 ,#C0C0C0C0 ,#C0C0FFFF ,
#C0C0C0C0 ,#C0C0C0C0 ,#C0C0C0C0 ,#C0C0 ,#0 ,#0 ,#0 ,
#FFFF ,#0 ,#0 ,#0 ,#0 ,#0 ,#0 ,#0 ,
#0 ,#0 ,#FFFF ,#0 ,#0 ,#0 ,#0 ,#1010000 ,#1010101 ,
#1010101 ,#1010101 ,#101FFFF ,#1010101 ,#1010101 ,#1010101 ,#C0C00101 ,
#C0C0C0C0 ,#C0C0C0C0 ,#C0C0C0C0 ,#C0C0FFFF ,#C0C0C0C0 ,#C0C0C0C0 ,#C0C0C0C0 ,
#101C0C0 ,#1010101 ,#1010101 ,#1010101 ,#1010101 ,#1010101 ,#1010101 ,
#1010101 ,#C0C00101 ,#C0C0C0C0 ,#C0C0C0C0 ,#FFFFC0C0 ,#FFFFC0C0 ,#C0C0C0C0 ,
#C0C0C0C0 ,#C0C0C0C0 ,#707C0C0 ,#7070707 ,#7070707 ,#7070707 ,#7070707 ,
#7070707 ,#7070707 ,#7070707 ,#38380707 ,#38383838 ,#38383838 ,#38383838 ,
#38383838 ,#38383838 ,#38383838 ,#38383838 ,#7073838 ,#7070707 ,#7070707 ,
#7070707 ,#7070707 ,#0 ,#0 ,#0 ,#38380000 ,#38383838 ,
#38383838 ,#3F3F3838 ,#FFF0000 ,#0 ,#0 ,#0 ,#0 ,
#0 ,#0 ,#7070000 ,#7070707 ,#7070707 ,#7070707 ,#7070707 ,#7070707 ,
#707 ,#0 ,#0 ,#FFFF0000 ,#3F3F0000 ,#38383838 ,#38383838 ,
#38383838 ,#7073838 ,#7070707 ,#7070707 ,#FFF0707 ,#FFF0000 ,#0 ,
#0 ,#0 ,#38380000 ,#38383838 ,#38383838 ,#3F3F3838 ,#FFF0000 ,
#0 ,#0 ,#0 ,#0 ,#0 ,#0 ,#FFFF0000 ,
#FFFF0000 ,#7070707 ,#7070707 ,#7070707 ,#707 ,#0 ,#0 ,
#FFFF0000 ,#3F3F0000 ,#38383838 ,#38383838 ,#38383838 ,#7073838 ,#7070707 ,
#7070707 ,#7070707 ,#707C07 ,#7070707 ,#7070707 ,#7070707 ,#38380707 ,
#38383838 ,#38383838 ,#3F3F3838 ,#3F3F0000 ,#38383838 ,#38383838 ,#38383838 ,
#3838 ,#0 ,#0 ,#FFFF0000 ,#FFF0000 ,#0 ,#0 ,
#0 ,#0 ,#0 ,#0 ,#FFFF0000 ,#FFF0000 ,#FFF0000 ,#0 ,
#0 ,#0 ,#7070000 ,#7070707 ,#7070707 ,#FFF0707 ,#FFF0000 ,
#7070707 ,#7070707 ,#7070707 ,#38380707 ,#38383838 ,#38383838 ,#3F3F3838 ,
#3F3F0000 ,#38383838 ,#38383838 ,#38383838 ,#1013838 ,#1010101 ,#1010101 ,
#FFFF0101 ,#FFF0000 ,#0 ,#0 ,#0 ,#C0C00000 ,#C0C0C0C0 ,
#C0C0C0C0 ,#FFFFC0C0 ,#FFF0000 ,#0 ,#0 ,#0 ,#7070000 ,
#7070707 ,#7070707 ,#7070707 ,#FFFF ,#0 ,#0 ,#0 ,
#38380000 ,#38383838 ,#38383838 ,#38383838 ,#FFFF ,#0 ,#0 ,
#0 ,#0 ,#0 ,#FFFF0000 ,#FFF0000 ,#FFF0000 ,#1010101 ,
#1010101 ,#1010101 ,#101 ,#0 ,#0 ,#FFFF0000 ,#FFF0000 ,
#C0C0C0C0 ,#C0C0C0C0 ,#C0C0C0C0 ,#C0C0 ,#0 ,#0 ,#0 ,
#707FFFF ,#7070707 ,#7070707 ,#70707 . ,#707 ,#0 ,#0 ,
#0 ,#3838FFFF ,#38383838 ,#38383838 ,#38383838 ,#7073838 ,#7070707 ,
#7070707 ,#7070707 ,#707 ,#0 ,#0 ,#0 ,#38380000 ,
#38383838 ,#38383838 ,#38383838 ,#FFFF ,#0 ,#0 ,#0 ,
#1010000 ,#1010101 ,#1010101 ,#1010101 ,#1010101 ,#0 ,#0 ,
#0 ,#C0C00000 ,#C0C0C0C0 ,#C0C0C0C0 ,#FFFFC0C0 ,#FFFFC0C0 ,#0 ,
#0 ,#0 ,#0 ,#0 ,#0 ,#1010000 ,#1010101 ,
#1010101 ,#1010101 ,#1010101 ,#101 ,#0 ,#0 ,#FFFF0000 ,
#FFFFC0C0 ,#C0C0C0C0 ,#C0C0C0C0 ,#C0C0C0C0 ,#C0C0 ,#0 ,#0 ,
#0 ,#7C70707 ,#7070707 ,#7070707 ,#7070707 ,#707 ,#0 ,

```

```
#0 ,#0 ,#3838FFFF,#38383838,#38383838,#38383838,#7073838 ,
#7070707 ,#7070707 ,#7070707 ,#707FFFF ,#7070707 ,#7070707 ,#7070707 ,
#38380707 ,#38383838 ,#38383838 ,#38383838 ,#3838FFFF ,#38383838 ,#38383838 ,
#38383838 ,#1013838 ,#1010101 ,#1010101 ,#FFFF0101 ,#FFFF0101 ,#1010101 ,
#1010101 ,#10.0101 ,#C0C00101 ,#C0C0C0C0 ,#C0C0C0C0 ,#FFFFC0C0 ,#FFFFC0C0 ,
#C0C0C0C0 ,#C0C0C0C0 ,#C0C0C0C0 ,#101C0C0 ,#1010101 ,#1010101 ,#1010101 ,
#FFFF ,#0 ,#0 ,#0 ,#0 ,#C0C00000 ,#C0C0C0C0 ,#C0C0C0C0 ,
#C0C0C0C0 ,#C0C0 ,#0 ,#0 ,#0 ,#0 ,#0 ,#0 ,
#0 ,#0 ,#1010101 ,#1010101 ,#1010101 ,#1010101 ,#101 ,
#0 ,#0 ,#0 ,#0 ,#C0C0FFFF ,#C0C0C0C0 ,#C0C0C0C0 ,#C0C0C0C0 ,
#FFFFC0C0 ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,
#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,
#FFFFFF ,#FFFFFF ,#FFF ,#0 ,#0 ,#0 ,#0 ,#FFFF0000 ,
#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFF ,#0 ,#0 ,#0 ,#0 ,
#FFFF0000 ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,
#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFF ,#0 ,
#0 ,#0 ,#0 ,#0 ,#0 ,#0 ,#0 ,#0 ,
#0 ,#0 ,#0 ,#0 ,#0 ,#0 ,#0 ,#0 ,
#FFFF0000 ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,
#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFF ,#0 ,
#0 ,#0 ,#0 ,#FFFF0000 ,#FFFFFF ,#FFFFFF ,#FFFFFF ,#FFF ,
#0 ,#0 ,#0 ,#0 ,#0 ,#0 ,#0 ,#0 ,#70783F1F ,
#70707070 ,#78707070 ,#1F3F ,#0 ,#0 ,#0 ,#0 ,
#78FCCE87 ,#38383838 ,#FC783838 ,#C7EE ,#0 ,#0 ,#0 ,#7070707 ,
#E0E0707 ,#E0E0E0E ,#1C1C1C1C ,#3F3F1C1C ,#38383838 ,#3838 ,#0 ,
#1C1CF8F0 ,#70381C1C ,#383870E0 ,#1C1C1C1C ,#E0F0783C ,#0 ,#0 ,
#0 ,#38383F3F ,#38383838 ,#38383838 ,#38383838 ,#38383838 ,#3838 ,
#0 ,#0 ,#C0CF3C ,#0 ,#0 ,#0 ,#0 ,
#0 ,#0 ,#0 ,#0 ,#0 ,#0 ,#0 ,#6E6E3F1F ,#1C1C0E0E ,
#38381C1C ,#3838 ,#0 ,#0 ,#0 ,#0 ,#0 ,#3838FEFE ,
#70703838 ,#E0E07070 ,#E0E0 ,#0 ,#0 ,#0 ,#1C1C3F3F ,#7070E0E ,
#1010303 ,#7070303 ,#1C1C0E0E ,#3F3F ,#0 ,#0 ,#C0CF3C ,
#0 ,#C0C08080 ,#8080 ,#C0C0000 ,#FCFC ,#0 ,#0 ,
#0 ,#0 ,#70783F1F ,#70707070 ,#78707070 ,#1F3F ,#0 ,
#0 ,#0 ,#0 ,#3870FFFF ,#38383838 ,#78383838 ,#E0F0 ,
#0 ,#0 ,#0 ,#0 ,#E0E0E0E ,#E0E0E0E ,#F0E0E0E ,
#E0E0E0F ,#6E0E0E0E ,#3C7E ,#0 ,#0 ,#7070707 ,#7070707 ,
#F070707 ,#FCFE ,#0 ,#0 ,#0 ,#0 ,#31311F0F ,
#3030101 ,#7070303 ,#707 ,#0 ,#0 ,#0 ,#0 ,
#C0C0FCFC ,#8080C0C0 ,#8080 ,#0 ,#0 ,#0 ,#0 ,#1010707 ,
#F0F0101 ,#1C1C1C1C ,#F0F1C1C ,#1010101 ,#707 ,#0 ,#0 ,
#C0C0F0F0 ,#F8F8C0C0 ,#1C1C1C1C ,#F8F81C1C ,#C0C0C0C0 ,#F0F0 ,#0 ,
#0 ,#383C1F0F ,#38383838 ,#3F3F3838 ,#38383838 ,#3C383838 ,#F1F ,
#0 ,#0 ,#1C3CF8F0 ,#1C1C1C1C ,#FCFC1C1C ,#1C1C1C1C ,#3C1C1C1C ,
#F0F8 ,#0 ,#0 ,#383C1F0F ,#38383838 ,#38383838 ,#38383838 ,
#E0E1C1C ,#3E3E ,#0 ,#0 ,#1C3CF8F0 ,#1C1C1C1C ,#1C1C1C1C ,
#1C1C1C1C ,#70703838 ,#7C7C ,#0 ,#0 ,#3030100 ,#101 ,
#383C1F0F ,#38383838 ,#3C383838 ,#F1F ,#0 ,#0 ,#8C8CF8F0 ,
#E0E0C0C0 ,#1C3CF8F0 ,#1C1C1C1C ,#3C1C1C1C ,#F0F8 ,#0 ,#0 ,
#0 ,#0 ,#71733F1E ,#1E3F7371 ,#0 ,#0 ,#0 ,#0 ,
#0 ,#0 ,#0 ,#C7E77E3C ,#3C7EE7C7 ,#0 ,#0 ,#0 ,
#0 ,#0 ,#0 ,#F070000 ,#1C1C1C1C ,#1D1D1D1D ,#303070F ,
#303 ,#0 ,#0 ,#60600000 ,#F8F06060 ,#DCDCDCDC ,#9C9C9C9C ,
#F0F8 ,#0 ,#0 ,#0 ,#E0F0703 ,#38381C1C ,#3F3F3838 ,
#38383838 ,#F0E1C1C ,#307 ,#0 ,#0 ,#FCFC ,#0 ,
#FCFC0000 ,#0 ,#0 ,#FCFC ,#0 ,#0 ,#383C1F0F ,
#38383838 ,#38383838 ,#38383838 ,#38383838 ,#3838 ,#0 ,#0 ,
#1C3CF8F0 ,#1C1C1C1C ,#1C1C1C1C ,#1C1C1C1C ,#1C1C1C1C ,#1C1C ,#0 ,
#0 ,#0 ,#0 ,#3F3F ,#3F3F ,#3F3F ,#0 ,#0 ,
#0 ,#0 ,#0 ,#0 ,#FCFC ,#FCFC ,#FCFC ,
#0 ,#0 ,#0 ,#0 ,#3030000 ,#3F3F0303 ,#3030303 ,
#3F3F0000 ,#0 ,#0 ,#0 ,#0 ,#80800000 ,#F8F88080 ,
#80808080 ,#F8F80000 ,#0 ,#0 ,#0 ,#0 ,#3030707 ,
#101 ,#0 ,#0 ,#3030101 ,#707 ,#1F1F ,#0 ,#0 ,
#80800000 ,#E0E0C0C0 ,#E0E07070 ,#8080C0C0 ,#0 ,#F0F0 ,#0 ,
#0 ,#1010000 ,#7070303 ,#7070E0E ,#1010303 ,#0 ,#F0F ,
#0 ,#0 ,#C0C0E0E0 ,#8080 ,#0 ,#C0C08080 ,#E0E0 ,
```

```

#F0F0      ,#0      ,#0      ,#0      ,#0      ,#0      ,#0      ,
#1010000 ,#1010101 ,#101      ,#0      ,#E0E6FE7C,#E0E0E0E0,#E0E0E0E0,
#E0E0E0E0,#C0C0E0E0,#C0C0C0C0,#101C0C0 ,#1010101 ,#3030101 ,#3030303 ,
#3030303 ,#3030303 ,#1F3F3303,#0      ,#C0C00000,#C0C0C0C0,#8080C0C0,
#80808080,#80808080,#80808080,#808080 ,#0      ,#0      ,#0      ,
#3070703 ,#3F3F0000,#7030000 ,#307 ,#0      ,#0      ,#0      ,
#0      ,#80C0C080,#F8F80000,#C0800000,#80C0 ,#0      ,#0      ,
#0      ,#0      ,#1F0F0000,#3839 ,#38391F0F,#0      ,#0      ,
#0      ,#0      ,#0      ,#9C1C0000,#F0F8 ,#F0F89C1C,#0      ,
#0      ,#0      ,#0      ,#E0E0703 ,#3070E0E ,#0      ,#0      ,
#0      ,#0      ,#0      ,#0      ,#3838F0E0,#E0F03838,#0      ,
#0      ,#0      ,#0      ,#0      ,#0      ,#7070300 ,#30707 ,
#0      ,#0      ,#0      ,#0      ,#0      ,#0      ,#E0E0C000,
#C0E0E0 ,#0      ,#0      ,#0      ,#0      ,#0      ,#0      ,
#0      ,#3000000 ,#7070707 ,#3      ,#0      ,#0      ,#0      ,
#0      ,#0      ,#C0000000,#E0E0E0E0,#C0 ,#0      ,#0      ,
#0      ,#0      ,#0      ,#0      ,#0      ,#1C1C7878,#7070E0E ,
#1010303 ,#0      ,#7F7F0000,#70707070,#70707070,#70707070,#70707070,
#70707070,#F0F0F0F0,#7070F0F0,#0      ,#6060707 ,#6060606 ,#606 ,
#0      ,#0      ,#0      ,#0      ,#0      ,#6060C080,#60606060,
#6060 ,#0      ,#0      ,#0      ,#0      ,#0      ,#707 ,
#6060301 ,#707 ,#0      ,#0      ,#0      ,#0      ,#0      ,
#6060E0C0,#80C0 ,#E0E0 ,#0      ,#0      ,#0      ,#0      ,
#0      ,#0      ,#7070000 ,#7070707 ,#7070707 ,#0      ,#0      ,
#0      ,#0      ,#0      ,#E0E00000,#E0E0E0E0,#E0E0E0E0,#0      ,
#0      ,#0      ,#0      ,#383C1F0F,#77777370,#77777777,#77777777,
#3C387073,#F1F ,#0      ,#0      ,#E1EFCF8 ,#37F7E707,#7070737 ,
#F7373707,#1E0E07E7,#F8FC ,#0      ,#0      ,#0      ,#0      ,
#0      ,#0      ,#0      ,#0      ,#0      ,#0      ,#0      ,
#0      ,#0      ,#0      ,#0      ,#0      ,#0      ,#1A1A0000]:

```

4.3. Makefiles

4.3.1. Seeker

```
LIBRARIAN=iibr
OCCAM=occam
LINK=ilink
CONFIG=iconf
ADDBOOT=iboot
LIBOPT=
OCCOPT=/a
LINKOPT=
CONFOPT=
BOOTOPT=

seeker.btl:   seeker.pgm controll.t8h guidance.t8h xbar.t2h gtsei.t2h \
             backgrou.t8h targetle.t8h target.t8h spcontro.t8h sp.t4h firstbuf.t8h \
             secondbu.t8h formatte.t8h imagedis.c8h trackdis.c8h b409stub.c2h \
             gtspi.t2h hostseek.c8h
             $(CONFIG) seeker /o seeker.btl $(CONFOPT)

HostSeek.c8h: HostSeek.l8h HostSeek.t8h
             $(LINK) /f HostSeek.l8h /o HostSeek.c8h $(LINKOPT)

HostSeek.t8h: HostSeek.occ s_header.inc c:\itools\libs\hostio.inc \
             c:\itools\libs\hostio.lib c:\itools\libs\hostio.liu \
             c:\itools\libs\convert.lib gif02.c8h loader.c8h runseekr.c8h
             $(OCCAM) HostSeek /t8 /h /o HostSeek.t8h $(OCCOPT)

gif02.c8h:    gif02.l8h gif02.t8h
             $(LINK) /f gif02.l8h /o gif02.c8h $(LINKOPT)

gif02.t8h:    gif02.occ c:\itools\libs\hostio.inc c:\itools\libs\hostio.lib \
             c:\itools\libs\hostio.liu c:\itools\libs\convert.lib
             $(OCCAM) gif02 /t8 /h /o gif02.t8h $(OCCOPT)

loader.c8h:   loader.l8h loader.t8h
             $(LINK) /f loader.l8h /o loader.c8h $(LINKOPT)

loader.t8h:   loader.occ s_header.inc c:\itools\libs\hostio.inc \
             c:\itools\libs\hostio.lib c:\itools\libs\hostio.liu \
             c:\itools\libs\convert.lib
             $(OCCAM) loader /t8 /h /o loader.t8h /a

runseekr.c8h: runseekr.l8h runseekr.t8h
             $(LINK) /f runseekr.l8h /o runseekr.c8h $(LINKOPT)

runseekr.t8h: runseekr.occ s_header.inc c:\itools\libs\hostio.inc \
             c:\itools\libs\hostio.lib c:\itools\libs\hostio.liu \
             c:\itools\libs\convert.lib
             $(OCCAM) runseekr /t8 /h /o runseekr.t8h $(OCCOPT)

controll.t8h: controll.occ s_header.inc
             $(OCCAM) controll /t8 /h /o controll.t8h $(OCCOPT)

guidance.t8h: guidance.occ s_header.inc
             $(OCCAM) guidance /t8 /h /o guidance.t8h $(OCCOPT)

xbar.t2h:     xbar.occ s_header.inc
             $(OCCAM) xbar /t2 /h /o xbar.t2h $(OCCOPT)

gtspi.t2h:    gtspi.occ s_header.inc
             $(OCCAM) gtspi /t2 /h /o gtspi.t2h $(OCCOPT)

gtsei.t2h:    gtsei.occ s_header.inc
             $(OCCAM) gtsei /t2 /h /o gtsei.t2h $(OCCOPT)
```

```

backgrou.t8h:  backgrou.occ s_header.inc
               $(OCCAM) backgrou /t8 /h /o backgrou.t8h $(OCCOPT)

targetle.t8h:  targetle.occ s_header.inc
               $(OCCAM) targetle /t8 /h /o targetle.t8h $(OCCOPT)

target.t8h:    target.occ s_header.inc
               $(OCCAM) target /t8 /h /o target.t8h $(OCCOPT)

spcontro.t8h:  spcontro.occ s_header.inc
               $(OCCAM) spcontro /t8 /h /o spcontro.t8h $(OCCOPT)

sp.t4h: sp.occ s_header.inc
        $(OCCAM) sp /t4 /h /o sp.t4h $(OCCOPT)

firstbuf.t8h:  firstbuf.occ
               $(OCCAM) firstbuf /t8 /h /o firstbuf.t8h $(OCCOPT)

secondbu.t8h:  secondbu.occ
               $(OCCAM) secondbu /t8 /h /o secondbu.t8h $(OCCOPT)

formatte.t8h:  formatte.occ
               $(OCCAM) formatte /t8 /h /o formatte.t8h $(OCCOPT)

imagedis.c8h:  imagedis.l8h imagedis.t8h
               $(LINK) /f imagedis.l8h /o imagedis.c8h $(LINKOPT)

imagedis.t8h:  imagedis.occ crtc.inc g_header.inc c:\itools\libs\convert.lib \
               graphics.lib s_header.inc
               $(OCCAM) imagedis /t8 /h /o imagedis.t8h $(OCCOPT)

graphics.lib:  graphics.lbb b409.t2h g_line.t8h g_system.t8h g_text.t8h
               $(LIBRARIAN) /f graphics.lbb /o graphics.lib $(LIBOPT)

b409.t2h:      b409.occ crtc.inc
               $(OCCAM) b409 /t2 /h /o b409.t2h $(OCCOPT)

g_line.t8h:    g_line.occ g_header.inc
               $(OCCAM) g_line /t8 /h /o g_line.t8h $(OCCOPT)

g_system.t8h:  g_system.occ crtc.inc g_header.inc
               $(OCCAM) g_system /t8 /h /o g_system.t8h $(OCCOPT)

g_text.t8h:    g_text.occ g_header.inc
               $(OCCAM) g_text /t8 /h /o g_text.t8h $(OCCOPT)

trackdis.c8h:  trackdis.l8h trackdis.t8h
               $(LINK) /f trackdis.l8h /o trackdis.c8h $(LINKOPT)

trackdis.t8h:  trackdis.occ s_header.inc g_header.inc crtc.inc \
               c:\itools\libs\convert.lib graphics.lib sys6.inc
               $(OCCAM) trackdis /t8 /h /o trackdis.t8h $(OCCOPT)

b409stub.c2h:  b409stub.l2h b409stub.t2h
               $(LINK) /f b409stub.l2h /o b409stub.c2h $(LINKOPT)

b409stub.t2h:  b409stub.occ crtc.inc graphics.lib
               $(OCCAM) b409stub /t2 /h /o b409stub.t2h $(OCCOPT)

```

4.3.2. HostSeeker

```
LIBRARIAN=ilibr  
OCCAM=occam  
LINK=ilink  
CONFIG=iconf  
ADDBOOT=iboot  
LIBOPT=  
OCCOPT=  
LINKOPT=  
CONFOPT=  
BOOTOPT=
```

```
hostseek.c8h: hostseek.l8h hostseek.t8h  
          $(LINK) /f hostseek.l8h /o hostseek.c8h $(LINKOPT)
```

```
hostseek.t8h: hostseek.occ s_header.inc c:\itools\libs\hostio.inc \  
          c:\itools\libs\hostio.lib c:\itools\libs\hostio.liu \  
          c:\itools\libs\convert.lib  
          $(OCCAM) hostseek /t8 /h /o hostseek.t8h $(OCCOPT)
```

4.3.3. Graphics

```
LIBRARIAN=ilibr
OCCAM=occam
LINK=ilink
CONFIG=iconf
ADDBOOT=iboot
LIBOPT=
OCCOPT=
LINKOPT=
CONFOPT=
BOOTOPT=
```

```
graphics.lib: graphics.lbb b409.t2h g_line.t8h g_system.t8h g_text.t8h
               $(LIBRARIAN) /f graphics.lbb /o graphics.lib $(LIBOPT)
```

```
b409.t2h:      b409.occ crtc.inc
               $(OCCAM) b409 /t2 /h /o b409.t2h $(OCCOPT)
```

```
g_line.t8h:    g_line.occ g_header.inc
               $(OCCAM) g_line /t8 /h /o g_line.t8h $(OCCOPT)
```

```
g_system.t8h:  g_system.occ crtc.inc g_header.inc
               $(OCCAM) g_system /t8 /h /o g_system.t8h $(OCCOPT)
```

```
g_text.t8h:    g_text.occ g_header.inc
               $(OCCAM) g_text /t8 /h /o g_text.t8h $(OCCOPT)
```

4.3.4. B409stub.l2h

```
--  
-- Version 2.80, 28th April 1989  
--  
-- linker command file "c:\seeker\b409stub.l2h"  
--  
b409stub.t2h  
graphics.lib  
occam2h.lib
```

4.3.5. GIF02.l8h

```
--  
-- Version 2.80, 28th April 1989  
--  
-- linker command file "gif02.l8h"  
--  
gif02.t8h  
hostio.lib  
convert.lib  
occam8h.lib
```

4.3.6. HostSeeker.l8h

```
--  
-- Version 2.80, 28th April 1989  
--  
-- linker command file "HostSeek.l8h"  
--  
HostSeek.t8h  
=gif02.t8h  
=loader.t8h  
=runseekr.t8h  
hostio.lib  
convert.lib  
gif02.c8h  
loader.c8h  
runseekr.c8h  
occam8h.lib
```

4.3.7. ImageDisplay.l8h

```
--  
-- Version 2.80, 28th April 1989  
--  
-- linker command file "imagedis.l8h"  
--  
imagedis.t8h  
convert.lib  
graphics.lib  
occam8h.lib
```

4.3.8. Loader.l8h

```
--  
-- Version 2.80, 28th April 1989  
--  
-- linker command file "loader.l8h"  
--  
loader.t8h  
hostio.lib  
convert.lib  
occam8h.lib
```

4.3.9. RunSeekr.l8h

```
--  
-- Version 2.80, 28th April 1989  
--  
-- linker command file "runseekr.l8h"  
--  
runseekr.t8h  
hostio.lib  
convert.lib  
occam8h.lib
```

4.3.10. TrackDisplay.l8h

```
--  
-- Version 2.80, 28th April 1989  
--  
-- linker command file "trackdis.l8h"  
--  
trackdis.t8h  
convert.lib  
graphics.lib  
occam8h.lib
```

5. Appendix B: Program Listings (Scene generation for SP testing)

5.1. PC Source Code (Modula-2)

5.1.1. SPTTest.mod

```
MODULE SPTest ;
IMPORT Graph, IO, Objects, FIO, Image, Scenario;
VAR
  c      : CHAR ;
  f, scene : FIO.File ;
  i, j : CARDINAL ;
  dummy : BOOLEAN ;
BEGIN

  (*f:= FIO.OpenRead ("objects.dat") ;*)
  scene := FIO.OpenRead ("scene.dat") ;
  (* Objects.LoadObjects(f) ;*)
  Image.Init () ;
  Scenario.ParseScenario (scene) ;
  c:=IO.RdKey () ;
END SPTest.
```

5.1.2. Image.def

```

DEFINITION MODULE Image;

VAR
  MaskMode : (NOMASK, MASK, UNMASK) ;
  PixelMode : (OVERLAY, ADD) ;

PROCEDURE NewFrame () ;

PROCEDURE Plot (x, y, Color : CARDINAL) ;

PROCEDURE Polygon (n : CARDINAL; px, py: ARRAY OF CARDINAL;
                  FillColor : CARDINAL) ;

PROCEDURE Init () ;

END Image.

```

5.1.3. Image.mod

```

IMPLEMENTATION MODULE Image ;
IMPORT Graph ;
PROCEDURE Min (a,b :CARDINAL) : CARDINAL ;
BEGIN
  IF a > b THEN RETURN b
  ELSE RETURN a END ;
END Min;

PROCEDURE NewFrame () ;
BEGIN
  Graph.Rectangle (0, 0, 127, 127, 0, TRUE) ;
END NewFrame ;

PROCEDURE Plot (x, y, Color : CARDINAL) ;
VAR temp : CARDINAL ;
BEGIN
  temp :=Graph.Point (x, y) ;
  IF (MaskMode = MASK) AND (temp = 0) THEN RETURN END ;
  IF (MaskMode = UNMASK) AND (temp # 0) THEN RETURN END ;
  CASE PixelMode OF
    | OVERLAY :
      Graph.Plot (x, y, Min(189, Color)) ;
    | ADD :
      Graph.Plot (x, y, Min(189, Color+ temp)) ;
  END ; (* case *)
END Plot;

PROCEDURE Polygon (n : CARDINAL; px, py: ARRAY OF CARDINAL;
                  FillColor : CARDINAL) ;
BEGIN
  Graph.Polygon (n, px, py, FillColor) ;
END Polygon;

PROCEDURE Init () ;

VAR dummy : BOOLEAN ;
    palette : ARRAY [0..255] OF LONGCARD ;
    longI, temp : LONGCARD ;
    i : CARDINAL ;
BEGIN
  MaskMode := NOMASK ;
  PixelMode := OVERLAY ;
  FOR i := 0 TO 63 DO
    longI := LONGCARD (i) ;
    temp := (longI) + (longI << 8) + (longI << 16) ;

```

Image.mod

```
    palette[i*3] := temp ;
    palette[i*3 + 1] := palette[i*3] + 1 ;
    palette[i*3 + 2] := palette[i*3+1] + 256 ;
END ; (* for *)
dummy := Graph.SetVideoMode (Graph._MRES256COLOR) ;
IF Graph.RemapAllPalette(palette) <> 0 THEN END ;
Graph.Line (0, 0, 0, 128, 200) ;
Graph.Line (128, 128, 128, 0, 200) ;
Graph.Line (0, 128, 128, 128, 200) ;
Graph.Line (128, 0, 0, 0, 200) ;
Graph.SetClipRgn (1, 1, 127, 127) ;
END Init;

BEGIN
END Image.
```

5.1.4. Objects.def

```

DEFINITION MODULE Objects ;

IMPORT FIO ;

PROCEDURE LoadObjects (f: FIO.File) ;

PROCEDURE DisplayObject (id, x, y, color: CARDINAL;
                        scale : REAL) : BOOLEAN;

END Objects.

```

5.1.5. Objects.mod

```

IMPLEMENTATION MODULE Objects ;

IMPORT Str, FIO, IO, Image ;

CONST
  maxPoints = 32 ;
  maxDimension = 8 ;
  maxObjects = 120 ;
TYPE
  ObjectType = (Polygon, Bitmap) ;
  Object = RECORD
    id : CARDINAL ;
    CASE type : ObjectType OF
      | Polygon :
        numPoints : CARDINAL ;
        x : ARRAY [1..maxPoints] OF REAL;
        y : ARRAY [1..maxPoints] OF REAL;
      | Bitmap : xExtent, yExtent : CARDINAL ;
        intensity : ARRAY [1..maxDimension] OF
          ARRAY [1..maxDimension] OF REAL ;
    END ;
  END ;
  ObjectArray = ARRAY [0..maxObjects] OF Object ;

VAR
  object : ObjectArray ;

PROCEDURE LoadObjects (f: FIO.File) ;
VAR
  id, i, j : CARDINAL ;
  temp : ARRAY [0..32] OF CHAR ;
BEGIN
  (* REPEAT*)
    id := FIO.RdCard (f) ;
    (*IO.WrCard (id, 6) ;*)
    FIO.RdStr (f, temp) ;
    Str.Caps (temp) ;
    WITH object[id] DO
      IF Str.Match (temp, "**POLYGON**") THEN
        type := Polygon ;
        (*IO.WrStr (" Polygon") ;*)
        (*IO.WrLn() ;*)
        numPoints := FIO.RdCard (f) ;
        FOR i := 1 TO numPoints DO
          x[i] := FIO.RdReal (f) ;
          y[i] := FIO.RdReal (f) ;
        END ;
      ELSIF Str.Match (temp, "**BITMAP**") THEN
        type := Bitmap ;
        (*IO.WrStr (" Bitmap") ;*)

```

```
(*IO.WrLn() ;*)
    xExtent := FIO.RdCard (f) ;
    yExtent := FIO.RdCard (f) ;
    FOR i := 1 TO yExtent DO
        FOR j := 1 TO xExtent DO
            intensity[i][j] := FIO.RdReal (f) ;
        END ;
    END ;
END ; (* if *)

END; (* with*)
(* UNTIL FIO.EOF ;*)
END LoadObjects ;

PROCEDURE DisplayObject (id, sx, sy, color : CARDINAL;
                        scale : REAL ) :BOOLEAN;
VAR
    px , py : ARRAY [0..maxPoints-1] OF CARDINAL ;
    i, j : CARDINAL ;
BEGIN
    WITH object[id] DO
        CASE type OF
            | Polygon :
                FOR i := 1 TO numPoints DO
                    px[i-1] := CARDINAL (INTEGER(sx)
                        + VAL(INTEGER, (x[i] * scale + 0.5)) );
                    py[i-1] := CARDINAL (INTEGER(sy)
                        + VAL (INTEGER, (y[i] * scale + 0.5)));
                END ;
                Image.Polygon (numPoints, px, py, color) ;
            | Bitmap :
                FOR i := 1 TO yExtent DO
                    FOR j := 1 TO xExtent DO
                        Image.Plot (sx + j, sy + i,
                            TRUNC (REAL(color) * intensity[i][j] + 0.5)) ;
                    END ;
                END ;
            ELSE
                RETURN FALSE ;
            END ;
        END ;
    RETURN TRUE ;
END DisplayObject;

BEGIN
END Objects.
```

5.1.6. Scenario.def

```

DEFINITION MODULE Scenario;
IMPORT FIO ;

PROCEDURE ParseScenario (f : FIO.File) ;

END Scenario.

```

5.1.7. Scenario.mod

```

IMPLEMENTATION MODULE Scenario;

IMPORT Objects, Image, Str, Lib, Sim, IO ;

TYPE
  NoiseType = (UNIFORM, POISSON, NORMAL, LORENTZIAN) ;

PROCEDURE Equal (a, b : ARRAY OF CHAR ) : BOOLEAN ;
BEGIN
  RETURN (Str.Compare (a, b) = 0) ;
END Equal;

PROCEDURE NoiseSource (noise :NoiseType ; power : CARDINAL;
                      x, y : CARDINAL) : CARDINAL ;
BEGIN
  CASE noise OF
    | LORENTZIAN : RETURN 0;
    | NORMAL : RETURN TRUNC (ABS (Sim.GaussianDeviation ())
                          * REAL(power) + 0.5);
    | POISSON : RETURN 0;
    | UNIFORM : RETURN Lib.RANDOM (power);
  END ; (* case *)
END NoiseSource ;

PROCEDURE ParseScenario (f : FIO.File) ;
VAR
  dummy : CHAR ;
  noise : NoiseType ;
  token : ARRAY [0..31] OF CHAR ;
  id, x, y, color : CARDINAL ;
  scale, xOffsetReal, yOffsetReal : REAL ;
  endScenario : BOOLEAN ;
  i, j, rep, rep2, x_off, y_off : CARDINAL ;
BEGIN
  endScenario := FALSE ;
  noise := UNIFORM ;
  REPEAT
    FIO.RdItem (f, token) ;
    Str.Caps (token);
    IF Equal (token, "OBJECT") THEN
      Objects.LoadObjects(f) ;
    ELSIF Equal (token, "PLACE") THEN
      id := FIO.RdCard (f) ;
      x := FIO.RdCard (f) ;
      y := FIO.RdCard (f) ;
      color := FIO.RdCard (f) ;
      scale := FIO.RdReal (f) ;
      IF Objects.DisplayObject (id, x, y, color, scale) THEN END ;
    ELSIF Equal (token, "REPEAT") THEN
      rep := FIO.RdCard (f) ;
      x_off := FIO.RdCard (f) ;
      y_off := FIO.RdCard (f) ;
      FOR i := 1 TO rep - 1 DO
        IF Objects.DisplayObject (id, x+ (x_off*i),
                                y+(y_off*i), color, scale) THEN END ;
      END FOR ;
    ELSE
      dummy := token[0] ;
      IF dummy = 0 THEN
        endScenario := TRUE ;
      END IF ;
    END IF ;
  UNTIL endScenario
END ParseScenario ;

```

```

END ;
ELSIF Equal (token, "ARRAY") THEN
  rep := FIO.RdCard (f) ;
  rep2 := FIO.RdCard (f) ;
  x_off := FIO.RdCard (f) ;
  y_off := FIO.RdCard (f) ;
  FOR i := 0 TO rep - 1 DO
    FOR j := 0 TO rep2 - 1 DO
      IF Objects.DisplayObject (id, x+ (j*x_off), y+(i*y_off),
                               color, scale) THEN END ;
    END ;
  END ;
ELSIF Equal (token, "NOISE") THEN
  x := FIO.RdCard (f) ;
  y := FIO.RdCard (f) ;
  color := FIO.RdCard (f) ;
  rep := FIO.RdCard (f) ;
  rep2 := FIO.RdCard (f) ;
  FOR i := 0 TO rep - 1 DO
    FOR j := 0 TO rep2 - 1 DO
      Image.Plot(x+j, y+i, NoiseSource (noise, color, x+j, y+i)) ;
    END ;
  END ;
ELSIF Equal (token, "GRADIENT") THEN
  x := FIO.RdCard (f) ;
  y := FIO.RdCard (f) ;
  color := FIO.RdCard (f) ;
  rep := FIO.RdCard (f) ;
  rep2 := FIO.RdCard (f) ;
  xOffsetReal := FIO.RdReal (f) ;
  yOffsetReal := FIO.RdReal (f) ;
  FOR i := 0 TO rep - 1 DO
    FOR j := 0 TO rep2 - 1 DO
      Image.Plot ((x+ j), (y+i), color + TRUNC (REAL(j)*xOffsetReal + 0.5)
                  + TRUNC (REAL(i)*yOffsetReal + 0.5)) ;
    END ;
  END ;
ELSIF Equal (token, "LORENTZIAN") THEN
  noise := LORENTZIAN ;
ELSIF Equal (token, "NORMAL") THEN
  noise := NORMAL ;
ELSIF Equal (token, "POISSON") THEN
  noise := POISSON ;
ELSIF Equal (token, "UNIFORM") THEN
  noise := UNIFORM ;
ELSIF Equal (token, "NEWFRAME") THEN
  Image.NewFrame () ;
ELSIF Equal (token, "MASK") THEN
  Image.MaskMode := Image.MASK ;
ELSIF Equal (token, "UNMASK") THEN
  Image.MaskMode := Image.UNMASK ;
ELSIF Equal (token, "NOMASK") THEN
  Image.MaskMode := Image.NOMASK ;
ELSIF Equal (token, "ADD") THEN
  Image.PixelMode := Image.ADD ;
ELSIF Equal (token, "OVERLAY") THEN
  Image.PixelMode := Image.OVERLAY ;
ELSIF Equal (token, "PAUSE") THEN
  dummy := IO.RdKey () ;
ELSIF Equal (token, "END") THEN
  endScenario := TRUE ;
END ;

UNTIL endScenario
END ParseScenario ;
BEGIN

```

END Scenario.

5.2. Transputer Implementation Source Code (Occam)

SPTTest.pgm

```
#INCLUDE "hostio.inc"
#USE "sptest.c8h"
#USE "video.c8h"
#USE "b409stub.c2h"
#USE "buffer.c8h"

--{{{ channels
VAL link0out IS 0 :
VAL link1out IS 1 :
VAL link2out IS 2 :
VAL link3out IS 3 :
VAL link0in  IS 4 :
VAL link1in  IS 5 :
VAL link2in  IS 6 :
VAL link3in  IS 7 :
--}}

CHAN OF ANY fromAnalog, toAnalog, bufferLink :
CHAN OF ANY fromHost, toGraphics :
CHAN OF SP  fs, ts :

PLACED PAR
  PROCESSOR 0 T8
    PLACE fs AT link0in :
    PLACE ts AT link0out :
    PLACE fromHost AT link2out :
    SPTtest (fs, ts, fromHost)

  PROCESSOR 1 T8
    PLACE fromHost AT link1in :
    PLACE bufferLink AT link2out :
    buffer (fromHost, bufferLink)

  PROCESSOR 2 T8
    PLACE bufferLink AT link1in :
    PLACE toGraphics AT link3out :
    buffer (bufferLink, toGraphics)

  PROCESSOR 3 T8
    PLACE toGraphics AT link1in :
    PLACE toAnalog AT link3out :
    PLACE fromAnalog AT link3in :
    Video (toGraphics, fromAnalog, toAnalog)

  PROCESSOR 4 T2
    PLACE toAnalog AT link0in :
    PLACE fromAnalog AT link0out :
    B409.stub ( toAnalog, fromAnalog)
```

SPTTest.occ

```

--{{{ #includes and #uses
#include "hostio.inc"
#include "common.inc"
#USE      "hostio.lib"
#USE      "string.lib"
#USE      "convert.lib"
#USE      "snglmath.lib"
--}}}}
PROC SPTest (CHAN OF SP fs, ts, CHAN OF ANY toGraphics )

    --{{{ constants
    VAL debug IS FALSE :

    -- Object types
    VAL UNINITIALIZED IS 0 :
    VAL POLYGON IS 1 :
    VAL BITMAP IS 2 :

    -- Image masking
    VAL NOMASK IS 0 :
    VAL MASK IS 1 :
    VAL UNMASK IS 2 :

    -- Image Pixel Mode
    VAL OVERLAY IS 0 :
    VAL ADD IS 1 :

    -- noise types
    VAL UNIFORM IS 0 :
    VAL NORMAL IS 0 :
    VAL LORENTZIAN IS 0 :
    VAL POISSON IS 0 :

    VAL MAXPOINTS IS 32 :
    --}}}}
    --{{{ global variables
    VAL maxObj IS 100 :

    [maxObj]INT      object.type :
    [maxObj][32]REAL32 object.x :
    [maxObj][32]REAL32 object.y :
    [maxObj]INT      object.numPoints :
    [maxObj]INT      object.xExtent :
    [maxObj]INT      object.yExtent :
    [maxObj][8][8]REAL32 object.intensity :

    INT noiseType, maskMode, pixelMode :
    --}}}}
    --{{{ PROCS
    --{{{ utility
    --{{{ FioReadItem
    PROC FioReadItem (CHAN OF SP fs, ts,
        VAL INT32 streamid,
        [ ]BYTE data, INT index,
        BOOL result)
        --{{{ PROC white.space
        BOOL FUNCTION white.space (VAL BYTE c)
        BOOL return :
        VALOF
        SEQ
        CASE c
        ' ', '*c', '*n', '*t'
        return := TRUE
        ELSE

```

```

        return := FALSE
    RESULT return
:
--}}}
[1]BYTE character :
INT length :
BOOL continue :
SEQ
    index := 0
    continue := TRUE
    WHILE continue
        SEQ
            so.read (fs, ts, streamid, length, character)
            IF
                --{{{ no more data
                length = 0
                SEQ
                    continue := FALSE
                    result := FALSE
                --}}}
                --{{{ leading white space
                white.space (character[0]) AND (index = 0)
                SKIP
                --}}}
                --{{{ termination of string
                white.space (character[0])
                SEQ
                    continue := FALSE
                    result := TRUE
                --}}}
                --{{{ significant character
                TRUE
                SEQ
                    data[index] := character[0]
                    index := index + 1
                --}}}
:
--}}}
--{{{ FioReadInt
PROC FioReadInt (CHAN OF SP fs, ts, VAL INT32 streamid,
                INT value, BOOL result)
    INT length :
    [128]BYTE item :
    BOOL error :
    SEQ
        FioReadItem (fs, ts, streamid, item, length, result)
        IF
            result
            SEQ
                STRINGTOINT (error, value, [item FROM 0 FOR length])
                result := NOT error
            TRUE
            SKIP
:
--}}}
--{{{ FioReadReal32
PROC FioReadReal32 (CHAN OF SP fs, ts, VAL INT32 streamid,
                   REAL32 value, BOOL result)
    INT length :
    [128]BYTE item :
    BOOL error :
    SEQ
        FioReadItem (fs, ts, streamid, item, length, result)
        IF
            result
            SEQ
                STRINGTOREAL32 (error, value, [item FROM 0 FOR length],

```

```

        result := NOT error
    TRUE
    SKIP
:
--}}}
--}}}
--{{{ image
--{{{ PROC UpdateModes
PROC UpdateModes ()
    SEQ
        toGraphics ! 2 ; c.pixel.mode; pixelMode
        toGraphics ! 2 ; c.mask.mode; maskMode
:
--}}}
--{{{ PROC Plot
PROC Plot (VAL INT x, y, color)
    SEQ
        toGraphics ! 4; c.plot; x ; y; color
:
--}}}
--{{{ PROC NoiseSource
PROC NoiseSource (INT noise, VAL INT noiseType, magnitude, x, y,
    INT32 seed)
    REAL32 random :
    SEQ
        random, seed := RAN (seed)
        noise := INT ROUND ((REAL32 TRUNC magnitude) * random)
:
--}}}
--{{{ PROC HLine
PROC HLine (VAL INT x0, y, x1, color)
    SEQ
        WHILE debug
            SEQ
                so.write.int (fs, ts, y, 6)
                so.write.int (fs, ts, x0, 6)
                so.write.string (fs, ts, "--")
                so.write.int (fs, ts, x1, 6)
                so.write.int (fs, ts, color, 6)
                so.write.nl (fs, ts)
            toGraphics ! 6; c.line; x0; y; x1; y; color
:
--}}}
--{{{ PROC Sort
PROC Sort ([]INT x, xord, VAL INT first, last)
    IF
        first < last
        --{{{ okay to sort
        INT check, next :
        INT v1, v2 :
        SEQ
            SEQ check = first FOR (last - first)
            SEQ
                next := check + 1
                v1 := x[xord[check]]
                WHILE next <= last
                    SEQ
                        v2 := x[xord[next]]
                        IF
                            v2 < v1
                            --{{{ switch indexes
                            INT temp :
                            SEQ
                                v1 := v2
                                temp := xord[next]
                                xord[next] := xord[check]
                                xord[check] := temp

```

```

--}}}
TRUE
SKIP
next := next + 1
--}}}
TRUE
SKIP
:
--}}}
--{{{ PROC Polygon
PROC Polygon (VAL {}INT px, py, VAL INT n, color)

--{{{ variables
INT miny, maxy, nextEdge, active :
{32}INT xord, x, e :
--}}}
SEQ
--{{{ find external y points
miny := py[0]
maxy := miny
SEQ i = 0 FOR n
  VAL test IS py[i] :
  IF
    test < miny
      miny := test
    test > maxy
      maxy := test
  TRUE
  SKIP
--}}}
SEQ y = miny FOR ((maxy - miny) + 1)
SEQ
--{{{ debug
WHILE debug
  SEQ
    so.write.string(fs, ts, " y=")
    so.write.int (fs, ts, y, -5)
    so.write.nl (fs, ts)
--}}}
active := (-1)
INT x0, x1, y0, y1 :
SEQ edge = 0 FOR n
  SEQ
    --{{{ debug
    WHILE debug
      SEQ
        so.write.string(fs, ts, " edge=")
        so.write.int (fs, ts, edge, -5)
        so.write.nl (fs, ts)
      --}}}
    --{{{ calculate nextEdge
    IF
      edge = (n - 1)
        nextEdge := 0
      TRUE
        nextEdge := edge + 1
    --}}}
    --{{{ assign start and end points
    x0 := px[edge]
    y0 := py[edge]
    x1 := px[nextEdge]
    y1 := py[nextEdge]
    --}}}
    --{{{ reverse if necessary
    IF
      y0 > y1
        INT temp:

```

[illegible]

```

        x[edge] := x[edge] - 1
    --{{{ debug
    WHILE debug
    SEQ
        so.write.string(fs, ts, "x1 < x0      ")
        so.write.string(fs, ts, " e[edge]=")
        so.write.int    (fs, ts, e[edge], -5)
        so.write.string(fs, ts, " x[edge]=")
        so.write.int    (fs, ts, x[edge], -5)
        so.write.nl (fs, ts)
    --}}}
    --}}}
    --{{{ increment active
    active := active + 1
    xord[active] := edge
    --}}}
    --}}}
    TRUE
    SKIP
    --}}}
    --{{{ sort and plot
    Sort (x, xord, 0, active)
    INT i :
    SEQ
        i := 0
        WHILE i < active
        SEQ
            HLine (x[xord[i]], y, x[xord[i+1]], color)
            i := i + 2
        --}}}
    :
    --}}}
    --}}}

--{{{ Objects module
--{{{ LoadObjects
PROC LoadObjects (CHAN OF SP fs, ts, VAL INT32 streamid)
    --{{{ check result
    PROC check.result (CHAN OF SP fs, ts, VAL BOOL value, VAL []BYTE message)
    IF
        value
        SKIP
    TRUE
        so.write.string.nl (fs, ts, message)
    :
    --}}}
    BOOL ok :
    INT id, length :
    [128]BYTE id.name :
    SEQ
        ok := TRUE
        WHILE ok
        SEQ
            FioReadInt (fs, ts, streamid, id, ok)
            FioReadItem (fs, ts, streamid, id.name, length, ok)
            to.upper.case (id.name)
            IF
                --{{{ POLYGON
                eqstr ([id.name FROM 0 FOR length], "POLYGON")
                SEQ
                    object.type[id] := POLYGON
                    so.write.string (fs, ts, "polygon          *c")
                    FioReadInt (fs, ts, streamid, object.numPoints[id], ok)
                    check.result (fs, ts, ok, "--Error reading numPoints")
                    SEQ i = 0 FOR object.numPoints[id]
                    SEQ

```

```

        FioReadReal32 (fs, ts, streamid, object.x[id][i], ok)
        check.result (fs, ts, ok, "--Error reading x")
        FioReadReal32 (fs, ts, streamid, object.y[id][i], ok)
        check.result (fs, ts, ok, "--Error reading y")
        --so.write.char (fs, ts, '.')
    --}}}
    --{{{ BITMAP
    eqstr ([id.name FROM 0 FOR length], "BITMAP")
    SEQ
        object.type[id] := BITMAP
        so.write.string (fs, ts, "bitmap" *c")
        FioReadInt (fs, ts, streamid, object.xExtent[id], ok)
        check.result (fs, ts, ok, "--Error reading xExtent")
        FioReadInt (fs, ts, streamid, object.yExtent[id], ok)
        check.result (fs, ts, ok, "--Error reading yExtent")
        SEQ i = 0 FOR object.yExtent[id]
        SEQ
            SEQ j = 0 FOR object.xExtent[id]
            so.write.char
            --so.write.char (fs, ts, '.')
            FioReadReal32 (fs, ts, streamid,
object.intensity[id][i][j], ok)
                check.result (fs, ts, ok, "--Error reading intensity")
        --}}}
        TRUE
        ok := FALSE
    :

--}}}
--{{{ DisplayObject
PROC DisplayObject (VAL INT id, sx, sy, color, VAL REAL32 scale)
    [MAXPOINTS]INT x, y :
    SEQ
        UpdateModes ()
        CASE object.type[id]
            POLYGON
                SEQ
                    SEQ i = 0 FOR object.numPoints[id]
                    SEQ
                        x[i] := (INT TRUNC (scale * object.x[id][i])) + sx
                        y[i] := (INT TRUNC (scale * object.y[id][i])) + sy
                    Polygon (x, y, object.numPoints[id], color)
            BITMAP
                SEQ
                    SEQ i = 0 FOR object.yExtent[id]
                    SEQ j = 0 FOR object.xExtent[id]
                    Plot (sx + j, sy + i, INT ROUND ((REAL32 TRUNC color) *
                        object.intensity[id][i][j]))
            ELSE
                SKIP
    :
--}}}
--}}}
--{{{ Scenario module
--{{{ ParseScenario
PROC ParseScenario (VAL INT32 streamid)
    --{{{ variables
    BOOL endScenario :
    INT id, x, y, color, repeat, repeat2, x.offset, y.offset :
    REAL32 scale :
    BOOL result :
    [128]BYTE long.token :
    INT32 seed :
    --}}}
    SEQ
        seed := 3245 (INT32)
        endScenario := FALSE

```

```

noiseType := UNIFORM
maskMode := NOMASK
pixelMode := OVERLAY
WHILE NOT endScenario
  INT length :
  BOOL ok :
  SEQ
    FioReadItem (fs, ts, streamid, long.token, length, result)
    token IS [long.token FROM 0 FOR length] :
    IF
      --{{{ placements
      --{{{ PLACE
      eqstr (token, "PLACE" )
      SEQ
        so.write.string (fs, ts, "place" *c")
        FioReadInt (fs, ts, streamid, id, ok)
        FioReadInt (fs, ts, streamid, x, ok)
        FioReadInt (fs, ts, streamid, y, ok)
        FioReadInt (fs, ts, streamid, color, ok)
        FioReadReal32 (fs, ts, streamid, scale, ok)
        DisplayObject (id, x, y, color, scale)
      --}}}}
      --{{{ REPEAT
      eqstr (token, "REPEAT")
      SEQ
        so.write.string (fs, ts, "repeat" *c")
        FioReadInt (fs, ts, streamid, repeat, ok)
        FioReadInt (fs, ts, streamid, x.offset, ok)
        FioReadInt (fs, ts, streamid, y.offset, ok)
        SEQ i = 1 FOR (repeat - 1)
          DisplayObject (id, x + (x.offset * i),
            y + (y.offset * i), color, scale)
        --}}}}
      --{{{ ARRAY
      eqstr (token, "ARRAY")
      SEQ
        so.write.string (fs, ts, "array" *c")
        FioReadInt (fs, ts, streamid, repeat, ok)
        FioReadInt (fs, ts, streamid, repeat2, ok)
        FioReadInt (fs, ts, streamid, x.offset, ok)
        FioReadInt (fs, ts, streamid, y.offset, ok)
        SEQ i = 0 FOR repeat
          SEQ j = 0 FOR repeat2
            IF
              (i = 0) AND (j = 0)
              SKIP
            TRUE
              DisplayObject (id, x + (x.offset * j),
                y + (y.offset * i), color, scale)
          --}}}}
      --}}}
      --{{{ backgrounds
      --{{{ NOISE
      eqstr (token, "NOISE")
      SEQ
        so.write.string (fs, ts, "noise" *c")
        FioReadInt (fs, ts, streamid, x, ok)
        FioReadInt (fs, ts, streamid, y, ok)
        FioReadInt (fs, ts, streamid, color, ok)
        FioReadInt (fs, ts, streamid, repeat, ok)
        FioReadInt (fs, ts, streamid, repeat2, ok)
        UpdateModes ()
        SEQ i = 0 FOR repeat
          SEQ j = 0 FOR repeat2
            INT n :
            SEQ
              NoiseSource (n, noiseType, color, x + j, y + i, seed)

```

```

        Plot (x + j, y + i, n)
    --}}}
    --{{{ GRADIENT
    eqstr (token, "GRADIENT")
    REAL32 x.offset.real :
    REAL32 y.offset.real :
    SEQ
    so.write.string (fs, ts, "gradient" *c")
    FioReadInt (fs, ts, streamid, x, ok)
    FioReadInt (fs, ts, streamid, y, ok)
    FioReadInt (fs, ts, streamid, color, ok)
    FioReadInt (fs, ts, streamid, repeat, ok)
    FioReadInt (fs, ts, streamid, repeat2, ok)
    FioReadReal32 (fs, ts, streamid, x.offset.real, ok)
    FioReadReal32 (fs, ts, streamid, y.offset.real, ok)
    UpdateModes ()
    SEQ i = 0 FOR repeat
    SEQ j = 0 FOR repeat2
    INT newColor :
    SEQ
    newColor := color + ((INT TRUNC ( (REAL32 TRUNC j) *
    (REAL32) ))) +
    (INT TRUNC ( (REAL32 TRUNC i) *
    (REAL32) )))
    Plot (x + j, y + i, newColor )
    --}}}
    --}}}
    --{{{ noise models
    eqstr (token, "LORENTZIAN")
    noiseType := LORENTZIAN
    eqstr (token, "NORMAL")
    noiseType := NORMAL
    eqstr (token, "UNIFORM")
    noiseType := UNIFORM
    eqstr (token, "POISSON")
    noiseType := POISSON
    --}}}
    --{{{ masks
    eqstr (token, "MASK")
    maskMode := MASK
    eqstr (token, "UNMASK")
    maskMode := UNMASK
    eqstr (token, "NOMASK")
    maskMode := NOMASK
    --}}}
    --{{{ pixel construction
    eqstr (token, "ADD")
    pixelMode := ADD
    eqstr (token, "OVERLAY")
    pixelMode := OVERLAY
    --}}}
    --{{{ time and frame manipulation
    eqstr (token, "NEWFRAME")
    --NewFrame ()
    SKIP
    eqstr (token, "PAUSE")
    SEQ
    so.write.string (fs, ts, "Paused -- hit any key to continue.")
    BYTE key, result :
    so.getkey (fs, ts, key, result)
    so.write.string (fs, ts, "*c")
    eqstr (token, "END")
    endScenario := TRUE
    --}}}

```

```

--{{{ unrecognized token
TRUE
  SEQ
    so.write.string (fs, ts, "Unrecognized token *'")
    so.write.string (fs, ts, token)
    so.write.string.nl (fs, ts, "'")
    endScenario := TRUE
  --}}}

:
--}}}
--}}}

--}}}

--{{{ variables
BYTE result :
INT32 streamid :
[128]BYTE buffer :
INT length :
BOOL ok :
[32]INT temp, tempord :
--}}}
SEQ
  so.open (fs, ts, "objects.dat", spt.text, spm.input, streamid, result)
  --{{{ load object data file
  IF
    result = spr.ok
    SEQ
      LoadObjects (fs, ts, streamid)
      so.close (fs, ts, streamid, result)
    TRUE
    SKIP
  --}}}
  --{{{ parse scenario
  IF
    result = spr.ok
    SEQ
      so.open (fs, ts, "scene.dat",
        spt.text, spm.input, streamid, result)
      ParseScenario (streamid)
      so.close (fs, ts, streamid, result)
    TRUE
    SKIP
  --}}}
  so.exit (fs, ts, 0 (INT32))
:

```

Buffer.occ

```
PROC buffer (CHAN OF ANY in, out)
  [2]INT length :
  [2][256]INT command :
  INT send, receive :
  SEQ
    receive := 0
    in ? length[receive]::command[receive]
  WHILE TRUE
    SEQ
      send := receive
      receive := 1 - receive
    PAR
      in ? length[receive]::command[receive]
      out ! length[send]::command[send]
  :
```

G_Line.occ

```

--{{{ SC line
--:::A 3 10
--{{{ line
--{{{ libraries
#include "g_header.inc"
--}}}
--{{{ plot
PROC plot ( VAL [] INT window, [] BYTE screen,
            VAL INT x, y, VAL BYTE color )

    -- plots a single point on the screen
    -- makes sure the pixels are actually in the window
    VAL pixels.line IS window[ w.pixels.line ] :
    VAL size.x      IS window[ w.size.x ] :
    VAL size.y      IS window[ w.size.y ] :
    SEQ
    IF
        (x < 0) OR (y < 0) OR (x >= size.x) OR (y >= size.y)
        SKIP
    TRUE
        screen[ (y * pixels.line) + x ] := color
:
--}}}
--{{{ draw.line
PROC draw.line ( VAL [] INT window, [] BYTE screen,
                VAL INT x1, y1, x2, y2,
                VAL BYTE color )

--{{{ clip.line
PROC clip.line (VAL INT x1, y1, x2, y2, INT result, VAL []INT window)
    -- decides whether a line is totally on or off screen/window
    --{{{ codes
    VAL code.centre IS #00 :      -- 0000
    VAL code.left   IS #01 :      -- 0001
    VAL code.right  IS #02 :      -- 0010
    VAL code.bottom IS #04 :      -- 0100
    VAL code.top    IS #08 :      -- 1000
    --}}}
    INT code1, code2 :
    --{{{ PROC check
    PROC check (VAL INT x, y, INT code)
        VAL x.max IS window[ w.size.x ] :
        VAL y.max IS window[ w.size.y ] :
        SEQ
        IF
            --{{{ x.min <= x < x.max
            (x >= 0) AND (x < x.max)
            code := code.centre
            --}}}
            --{{{ x < x.min
            x < 0
            code := code.left
            --}}}
            --{{{ x > x.max
            TRUE --x >= x.max
            code := code.right
            --}}}
        IF
            --{{{ y.min <= y < y.max
            (y >= 0) AND (y < y.max)
            SKIP
            --}}}
            --{{{ y < y.min
            y < 0

```

```

        code := code \/ code.top
    --}}}
    --{{{ y >= y.max
    TRUE --y > y.max
        code := code \/ code.bottom
    --}}}
:
--}}}
SEQ
    check (x1, y1, code1)
    check (x2, y2, code2)
    IF
        --{{{ line lies entirely within window
        (code1 \/ code2) = 0
            result := in.range
        --}}}
        --{{{ line lies entirely outside window
        (code1 /\ code2) <> 0
            result := not.inrange
        --}}}
        --{{{ partially in window perhaps
        TRUE
            result := part.inrange
        --}}}
:
--}}}
--{{{ slow.draw.line
PROC slow.draw.line ( VAL [] INT window, [] BYTE screen,
                     VAL INT x1, y1, x2, y2,
                     VAL BYTE color )
    -- uses Bresenham's integer algorithm to calculate plotting points
    -- calls plot to draw actual pixels on the screen
    VAL pixels.line IS window[w.pixels.line] :
    INT dx, dy :
    INT two.dx, two.dy :
    INT error :
    SEQ
        dx := x2 - x1
        dy := y2 - y1
        IF
            (dx <> 0) OR (dy <> 0)
            SEQ
                --{{{ a line to draw
                IF
                    --{{{ dy = 0                      -- horizontal line
                    dy = 0
                        SEQ i = x1 FOR dx + 1
                            plot (window, screen, i, y1, color)
                        --}}}
                    --{{{ dx = 0                      -- vertical line
                    dx = 0
                        SEQ
                            IF
                                dy > 0
                                    SEQ i = y1 FOR dy + 1
                                        plot (window, screen, x1, i, color)
                                    TRUE
                                    SEQ i = y2 FOR (-dy) + 1
                                        plot (window, screen, x1, i, color)
                                --}}}
                    --{{{ dx <> 0 dy <> 0            -- diagonal line
                    TRUE
                        INT x, y :
                        INT delta.y :
                        SEQ
                            x := x1
                            y := y1

```

```

two.dx := dx + dx
IF
  --{{{ dy > 0
  dy > 0
  SEQ
    two.dy := dy + dy
    IF
      --{{{ dy > dx
      dy > dx
      SEQ
        error := two.dx - dy
        --{{{ plot line
        SEQ i = 0 FOR dy + 1
        SEQ
          plot (window, screen, x, y, color )
          IF
            error >= 0
            SEQ
              x := x + 1
              error := error - two.dy
            TRUE
            SKIP
            y := y + 1
            error := error + two.dx
          --}}}
        --}}}
      --}}}
    --{{{ dy <= dx
    TRUE
    SEQ
      error := two.dy - dx
      --{{{ plot line
      SEQ i = 0 FOR dx + 1
      SEQ
        plot (window, screen, x, y, color )
        IF
          error >= 0
          SEQ
            y := y + 1
            error := error - two.dx
          TRUE
          SKIP
            x := x + 1
            error := error + two.dy
          --}}}
        --}}}
      --}}}
    --}}}
  --{{{ dy < 0
  TRUE
  SEQ
    dy := -dy
    two.dy := dy + dy
    IF
      --{{{ dy > dx
      dy > dx
      SEQ
        error := two.dx - dy
        --{{{ plot line
        SEQ i = 0 FOR dy + 1
        SEQ
          plot (window, screen, x, y, color)
          IF
            error >= 0
            SEQ
              x := x + 1
              error := error - two.dy
            TRUE
            SKIP

```

```

        y := y - 1
        error := error + two.dx
    --}}}
--}}}
--{{{ dy <= dx
TRUE
SEQ
    error := two.dy - dx
    SEQ i = 0 FOR dx + 1
    SEQ
        plot (window, screen, x, y, color)
        IF
            error >= 0
            SEQ
                y := y - 1
                error := error - two.dx
            TRUE
            SKIP
            x := x + 1
            error := error + two.dy
        --}}}
    --}}}
--}}}
--}}}
TRUE
    plot (window, screen, x1, y1, color)
:
--}}}
--{{{ fast.draw.line
PROC fast.draw.line (VAL [] INT window, [] BYTE screen,
    VAL INT x1, y1, x2, y2,
    VAL BYTE color)

    -- uses Bresenham's integer algorithm to calculate plotting points
    -- points are in increasing values of x
    -- only called when the line is known to be on screen / in window and
    -- the current pixel size is one
    INT dx, dy, two.dx, two.dy, delta.x, delta.y :
    INT error, pixel :
    VAL pixels.line IS window[ w.pixels.line ] :
    SEQ
        dx := x2 - x1                -- always zero or positive
        dy := y2 - y1
        pixel := (y1 * pixels.line) + x1
        IF
            (dx <> 0) OR (dy <> 0)
            SEQ
                --{{{ a line to draw
                IF
                    --{{{ dy = 0                -- horizontal line
                    dy = 0
                    SEQ i = pixel FOR dx + 1
                    screen[i] := color
                --}}}
                --{{{ dx = 0                -- vertical line
                dx = 0
                SEQ
                    IF
                        dy > 0
                        SEQ i = 0 FOR dy + 1
                        SEQ
                            screen[pixel] := color
                            pixel := pixel + pixels.line
                    TRUE
                    SEQ i = 0 FOR (-dy) + 1
                    SEQ
                        screen[pixel] := color

```

```

        pixel := pixel - pixels.line
--}}}
--{{{ dx <> 0 AND dy <> 0
TRUE
  INT delta.y :
  SEQ
    two.dx := dx + dx
    IF
      dy > 0
        delta.y := pixels.line
      TRUE
        SEQ
          dy := -dy
          delta.y := -pixels.line
    two.dy := dy + dy
    IF
      --{{{ dy > dx
      dy > dx
      SEQ
        error := two.dx - dy
        --{{{ plot line
        SEQ i = 0 FOR dy + 1
        SEQ
          screen[pixel] := color
          IF
            error >= 0
            SEQ
              pixel := pixel + 1
              error := error - two.dy
          TRUE
            SKIP
          pixel := pixel + delta.y
          error := error + two.dx
        --}}}
      --}}}
    --{{{ dy <= dx
    TRUE
      SEQ
        error := two.dy - dx
        --{{{ plot line
        SEQ i = 0 FOR dx + 1
        SEQ
          screen[pixel] := color
          IF
            error >= 0
            SEQ
              pixel := pixel + delta.y
              error := error - two.dx
          TRUE
            SKIP
          pixel := pixel + 1
          error := error + two.dy
        --}}}
      --}}}
    --}}}
  TRUE
    screen[pixel] := color
:
--}}}
INT x3, y3, x4, y4 :
INT result :
SEQ
  --{{{ swap x1,y1 with x2,y2 if x1 > x2
  IF
    x1 > x2
    SEQ

```

```

        x3 := x2
        y3 := y2
        x4 := x1
        y4 := y1
    TRUE
    SEQ
        x3 := x1
        y3 := y1
        x4 := x2
        y4 := y2
    --}}}
clip.line (x3, y3, x4, y4, result, window)
IF
    (result = in.range)
        fast.draw.line (window, screen, x3, y3, x4, y4, color)
    (result = part.inrange) OR (result = in.range)
        slow.draw.line (window, screen, x3, y3, x4, y4, color)
    TRUE
    SKIP
:
--}}}
--{{{ draw.polyline
PROC draw.polyline ( VAL [] INT window, [] BYTE screen,
                    VAL [] [2] INT points, VAL BYTE color)

    -- calls draw line to draw the lines
    INT x, y :
    SEQ
        x := points[0][0]
        y := points[0][1]
        SEQ i = 1 FOR (SIZE points) - 1
            VAL point IS points[i] :
            SEQ
                draw.line( window, screen, x, y, point[0], point[1], color )
                x := point[0]
                y := point[1]
        :
    --}}}
--{{{ draw.rectangle
PROC draw.rectangle ( VAL [] INT window, [] BYTE screen,
                    VAL [2][2] INT p, VAL BYTE color)

    -- calls draw line to draw the lines
    INT x, y :
    SEQ
        draw.line( window, screen, p[0][0], p[0][1], p[1][0], p[0][1], color )
        draw.line( window, screen, p[1][0], p[0][1], p[1][0], p[1][1], color )
        draw.line( window, screen, p[1][0], p[1][1], p[0][0], p[1][1], color )
        draw.line( window, screen, p[0][0], p[1][1], p[0][0], p[0][1], color )
    :
    --}}}
--}}}
--}}}

```


G_System.occ

```

--{{{ SC system
--:::A 3 10
--{{{ system
--{{{ libraries
#include "crtc.inc"
#include "g_header.inc"
--}}}
--{{{ set.colour
PROC set.colour ( CHAN OF CRTC message,
                  VAL INT channel, pixel, red, green, blue )
  -- set up a colour in the G170 colour look up table
  SEQ
    message ! crtc.color; INT16 channel; INT16 pixel;
              INT16 red; INT16 green; INT16 blue
  :
--}}}
--{{{ set.timing
PROC set.timing( CHAN OF CRTC message,
                 VAL INT width,height, line.frequency, frame.rate, pixel.clock,
                 VAL BOOL interlace )

  SEQ
    message ! crtc.init; INT16 width; INT16 height; INT32 line.frequency;
              INT16 frame.rate; INT32 pixel.clock; interlace
  :
--}}}
--{{{ set.B408
PROC set.B408( VAL INT DS, IE, EM, OE, R )

  --{{{ system constants
  VAL bpw.shift IS 2 :
  VAL mint      IS MOSTNEG INT :

  VAL DisplayStart.address IS (#00000000 >< mint) >> bpw.shift :
  VAL InterlaceEnable.address IS (#000C0000 >< mint) >> bpw.shift :
  VAL EventMode.address IS (#00100000 >< mint) >> bpw.shift :
  VAL OutputEnable.address IS (#00140000 >< mint) >> bpw.shift :
  VAL Ready.address IS (#00040000 >< mint) >> bpw.shift :

  INT DisplayStart, InterlaceEnable, EventMode, OutputEnable, Ready :

  PLACE DisplayStart AT DisplayStart.address :
  PLACE InterlaceEnable AT InterlaceEnable.address :
  PLACE EventMode AT EventMode.address :
  PLACE OutputEnable AT OutputEnable.address :
  PLACE Ready AT Ready.address :
  --}}}
  SEQ
    DisplayStart := DS
    InterlaceEnable := IE
    EventMode := EM
    OutputEnable := OE
    Ready := R
  :
--}}}
--{{{ init.G170
PROC init.G170 (CHAN OF CRTC message, VAL INT channel, table)

  SEQ
    message ! crtc.initLUT; INT16 channel; INT16 table
  :
--}}}
--{{{ clear.window
PROC clear.window (VAL [] INT window, [] BYTE screen)

  VAL size.x IS window[ w.size.x ] :

```

```
VAL size.y      IS  window[ w.size.y ] :
VAL pixels.line IS  window[ w.pixels.line ] :
VAL b.color     IS  BYTE window[ w.background.color ] :
INT ptr :
SEQ
  SEQ i = 0 FOR size.x
    screen[i] := b.color
  ptr := pixels.line
  SEQ i = 0 FOR size.y - 1
    SEQ
      [screen FROM ptr FOR size.x] := [screen FROM 0 FOR size.x]
      ptr := ptr + pixels.line
:
--}}
--}}
--}}
```

G_Text.occ

```

--{{{  SC text
--:::A  3 10
--{{{  text
#include "g_header.inc"
--{{{  FUNCTION GetINT
INT FUNCTION GetINT (VAL INT pointer, VAL [] INT table)
  INT return :
  VAL [] BYTE b.table RETYPES table :
  VALOF
    [4]BYTE return.b RETYPES return :
  SEQ
    return.b[0] := b.table[pointer]
    return.b[1] := b.table[pointer + 1]
    return.b[2] := b.table[pointer + 2]
    return.b[3] := b.table[pointer + 3]
  RESULT return
:
--}}}}
--{{{  FUNCTION GetINT16
INT16 FUNCTION GetINT16 (VAL INT pointer, VAL [] INT table)
  INT16 return :
  VAL [] BYTE b.table RETYPES table :
  VALOF
    [2]BYTE return.b RETYPES return :
  SEQ
    return.b[0] := b.table[pointer]
    return.b[1] := b.table[pointer + 1]
  RESULT return
:
--}}}}
--{{{  FUNCTION GetBYTE
BYTE FUNCTION GetBYTE (VAL INT pointer, VAL [] INT table)
  BYTE return :
  VAL [] BYTE b.table RETYPES table :
  VALOF
    SEQ
      return := b.table[pointer]
  RESULT return
:
--}}}}
--{{{  get.font.spec
PROC get.font.spec ( VAL [] INT font, [fs.size] INT spec)

  SEQ
    spec[ fs.PixWidth   ] := INT (GetINT16 (dfPixWidth.p, font))
    spec[ fs.PixHeight  ] := INT (GetINT16 (dfPixHeight.p, font))
    spec[ fs.FirstChar  ] := INT (GetBYTE  (dfFirstChar.p, font))
    spec[ fs.LastChar   ] := INT (GetBYTE  (dfLastChar.p, font))
    spec[ fs.BitsOffset ] :=      GetINT  (dfBitsOffset.p, font)
  :
--}}}}
--{{{  scroll
PROC scroll ( VAL [] INT window, []BYTE screen,
             VAL INT jump.size )
  -- scrolls a screen or window by the required number of lines (jump.size)
  VAL size.x IS window[ w.size.x ] :
  VAL size.y IS window[ w.size.y ] :
  VAL pixels.line IS window[ w.pixels.line ] :
  VAL b.color IS BYTE window[ w.background.color ] :
  INT p1, p2 :
  IF
    (jump.size > 0) AND (jump.size < size.y)
    --{{{  scroll screen
    SEQ

```

```

p1 := 0
p2 := pixels.line * jump.size
SEQ i = 0 FOR (size.y - jump.size)
  SEQ
    [screen FROM p1 FOR size.x] := [screen FROM p2 FOR size.x]
    p1 := p1 + pixels.line
    p2 := p2 + pixels.line
  SEQ i = 0 FOR size.x
    screen[ p1 + i ] := b.color
  p2 := p1 + pixels.line
  SEQ i = 0 FOR (jump.size - 1)
    SEQ
      [screen FROM p2 FOR size.x] := [screen FROM p1 FOR size.x]
      p2 := p2 + pixels.line
    --}}}
  jump.size > 0
  --{{{ clear screen
  SEQ

    SEQ i = 0 FOR size.x
      screen[ i ] := b.color
    p2 := pixels.line
    SEQ i = 0 FOR (jump.size - 1)
      SEQ
        [screen FROM p2 FOR size.x] := [screen FROM 0 FOR size.x]
        p2 := p2 + pixels.line
      --}}}
  TRUE
  SKIP
:
--}}}
--{{{ draw.char v2.0
PROC draw.char ( [ ] INT window, [ ] BYTE screen,
                 VAL BYTE char,
                 VAL [ ] INT font, VAL [fs.size] INT spec )

  --{{{ constants
  VAL mask IS 1 << 7 :
  pixels.line IS window[ w.pixels.line ] :
  size.x      IS window[ w.size.x      ] :
  size.y      IS window[ w.size.y      ] :
  cursor.x    IS window[ w.cursor.x    ] :
  cursor.y    IS window[ w.cursor.y    ] :

  VAL [ ] BYTE b.font RETYPES font :
  --}}}
  --{{{ variables
  INT bit, pixel :
  INT bitmask :
  INT char.width, offset, PixWidthBytes :
  INT char.spacing :
  INT character :
  --}}}
  --{{{ line feed
  PROC line.feed ( )

    SEQ
      cursor.y := cursor.y + spec[ fs.PixHeight ]
      IF
        (cursor.y + spec[ fs.PixHeight ]) < size.y
          SKIP
      TRUE
      INT scroll.lines :
      SEQ
        scroll.lines := (spec[ fs.PixHeight ] - (size.y - cursor.y)) + 1
        cursor.y := (size.y - spec[ fs.PixHeight ]) - 1
        scroll( window, screen, scroll.lines )

```

```

:
--}}}
SEQ
  character := INT char
  IF
    char = '*n'
      line.feed ()
    char = '*c'
      --{{{ carriage return
      cursor.x := 0
      --}}}
    (character >= spec[ fs.FirstChar ]) AND (character <= spec[ fs.LastChar ])
  SEQ
    character := character - spec[ fs.FirstChar ]
    --{{{ set font data
    IF
      spec[ fs.PixWidth ] <> 0
      SEQ
        PixWidthBytes := (spec[ fs.PixWidth ] + 7) >> 3
        char.width := spec[ fs.PixWidth ]
        offset := (character * (PixWidthBytes * spec[ fs.PixHeight
          spec[ fs.BitsOffset ]
        char.spacing := char.width
      TRUE
      -- Variable Width
      INT char.width.p, char.pointer.p :
      SEQ
        char.width.p := CharTable.p + (character << 2)
        char.pointer.p := char.width.p + 2
        char.width := INT(GetINT16(char.width.p, font))
        offset := INT(GetINT16(char.pointer.p, font))
        PixWidthBytes := (char.width + 7) >> 3
        char.spacing := char.width + 1
    --}}}
  IF
    --{{{ char too big
    (char.width > size.x) OR (spec[ fs.PixHeight ] > size.y)
    SKIP
    --}}}
    --{{{ room to draw char
    BOOL delayed.crlf :
    TRUE
    SEQ
      delayed.crlf := FALSE
      IF
        --{{{ room to draw whole char
        ((cursor.x + char.spacing) < size.x) AND
        ((cursor.y + spec[ fs.PixHeight ]) < size.y)
        SKIP
        --}}}
        --{{{ room to draw but at end of line
        ((cursor.x + char.spacing) = size.x) AND
        ((cursor.y + spec[ fs.PixHeight ]) < size.y)
        delayed.crlf := TRUE
        --}}}
        --{{{ we need carriage return - line feed
        TRUE
        SEQ
          cursor.x := 0
          line.feed ()
          --}}}
        pixel := (cursor.y TIMES pixels.line) + cursor.x
        --{{{ plot foreground only
        VAL f.color IS BYTE window[ w.foreground.color ] :
        SEQ
          SEQ i = 0 FOR spec[ fs.PixHeight ]
          SEQ

```

```

--{{{ draw row
SEQ j = 0 FOR PixWidthBytes
  SEQ
    bitmask := mask
    VAL this.byte IS INT b.font[ offset+((spec[fs.PixHeight]
TIMES j) + i) ] :
      SEQ k = 0 FOR 8
        SEQ
          bit := this.byte /\ bitmask
          IF
            --{{{ leave background as it is
            bit = 0
              SKIP
            --}}}
            --{{{ plot foreground bit
            TRUE
              screen[ pixel ] := f.color
            --}}}
          pixel := pixel + 1
          bitmask := bitmask >> 1
        --}}}
      pixel := (pixel - (PixWidthBytes<<3)) + pixels.line

    cursor.x := cursor.x + char.spacing
  --}}}
  IF
    delayed.crlf
    --{{{ we need carriage return - line feed
    SEQ
      cursor.x := 0
      line.feed ()
    --}}}
    TRUE
      SKIP
    --}}}
  TRUE
    SKIP
:
--}}}
--{{{ write.string v2.0
PROC write.string ( [ ] INT window, [ ] BYTE screen,
  VAL [ ] BYTE string, VAL [ ] INT font )

  [fs.size] INT spec :
  SEQ
    get.font.spec( font, spec )
    SEQ i = 0 FOR (SIZE string)
      draw.char( window, screen, string[i], font, spec )
  :
  --}}}
  --{{{ string.width
PROC string.width ( VAL [ ] INT font, VAL [ ] BYTE string, INT width )

  [fs.size] INT spec :
  SEQ
    get.font.spec( font, spec )
    width := 0
    SEQ i = 0 FOR SIZE string
      --{{{ add width for character[i]
      INT character :
      SEQ
        character := INT string[i]
        IF
          (character >= spec[ fs.FirstChar ]) AND (character <= spec[ fs.LastChar]
)
          SEQ
            character := character - spec[ fs.FirstChar ]

```

```

--{{{ determine width from font
IF
  spec[ fs.PixWidth ] <> 0
    width := width + spec[ fs.PixWidth ]
  TRUE
Width
  INT char.width.p, char.pointer.p :
  SEQ
    char.width.p := CharTable.p + (character << 2)
    width := (width + 1) + (INT(GetINT16(char.width.p, font)))
  --}}}
  TRUE
  SKIP
  --}}}
:
--}}}
--}}}
--}}}

```

Video.occ

```

PROC Video ( CHAN OF ANY fromHost, fromAnalog, toAnalog )

--{{{ libraries
--#INCLUDE "s_header.inc"
#INCLUDE "\seeker\g_header.inc"
#INCLUDE "\seeker\crtc.inc"
#INCLUDE "common.inc"

#USE "convert.lib"
#USE "\seeker\graphics.lib"
--#USE "extrio.lib"
--}}}}

--{{{ display constants
VAL width      IS      640 :
VAL height     IS      480 :
VAL line.frequency IS    60000 :
VAL frame.rate  IS      90 :
VAL pixel.clock IS  64000000 :
VAL interlace   IS     FALSE :
--}}}}

--{{{ fonts
#INCLUDE "\seeker\sys6.inc"
--}}}}

--{{{ place system variables
[(20*65536)+1280] BYTE screen.map :
PLACE screen.map AT screen.int.address :

INT DisplayStart :
PLACE DisplayStart AT DisplayStart.address :

INT EventMode :
PLACE EventMode AT EventMode.address :

INT SysReady :
PLACE SysReady AT (#00080000 >< (MOSTNEG INT)) >> 2 :

INT Ready :
PLACE Ready AT Ready.address :
--}}}}

--{{{ set up multiple screens
VAL screen.offset IS [ #00000, #50000, #A0000, #F0000 ] :

VAL screen.address IS [ #00000, #14000, #28000, #3C000 ] :
--}}}}

--{{{ place Event channel
CHAN OF ANY Event :
PLACE Event AT 8 :
--}}}}

--{{{ constants
VAL screen.width  IS  640 :
VAL screen.height IS  480 :
VAL screen.size   IS  screen.width * screen.height :

VAL char.width    IS   18 :
VAL char.height   IS   33 :
--}}}}

--{{{ procs
--{{{ libraries
#INCLUDE "g_header.inc"
--}}}}

--{{{ IMAX
INT FUNCTION IMAX ( VAL INT a, b )

INT r :

```

```

VALOF
  IF
    a > b
    r := a
  TRUE
    r := b
  RESULT r
:
--}}}
--{{{ IMIN
INT FUNCTION IMIN ( VAL INT a, b )

  INT r :
  VALOF
    IF
      a < b
      r := a
    TRUE
      r := b
    RESULT r
:
--}}}
--{{{ plot
PROC plot ( VAL [] INT window, [] BYTE screen,
            VAL INT x, y, VAL BYTE color )

  -- plots a single point on the screen
  -- makes sure the pixels are actually in the window
  VAL pixels.line IS window[ w.pixels.line ] :
  VAL size.x      IS window[ w.size.x ] :
  VAL size.y      IS window[ w.size.y ] :
  VAL start.x     IS window[ w.start.x ] :
  VAL start.y     IS window[ w.start.y ] :
  VAL fat.pixels.x IS window[ w.cursor.x ] :
  VAL fat.pixels.y IS window[ w.cursor.y ] :
  VAL mask.mode    IS (window[ w.foreground.color] /\ #0F) :
  VAL pixel.mode   IS ((window[ w.foreground.color] /\ #F0) >> 4) :
  VAL [4]BYTE fg   RETYPES (window[ w.foreground.color]) :
  VAL max.color    IS (INT fg[1]) :
  INT current.color :
  INT screen.position :
  SEQ
    IF
      (x < 0) OR (y < 0) OR (x >= size.x) OR (y >= size.y)
      SKIP
    TRUE
      SEQ
        --{{{ get address and current color
        VAL fat.y IS y TIMES fat.pixels.y :
        VAL fat.x IS x TIMES fat.pixels.x :
        screen.position := (start.x + (start.y TIMES pixels.line)) +
                           ((fat.y TIMES pixels.line) + fat.x)
        current.color := INT screen[screen.position]
        --}}}
        IF
          --{{{ skip under certain conditions
          (mask.mode = MASK) AND (current.color = 0 )
          SKIP
          (mask.mode = UNMASK) AND (current.color <> 0 )
          SKIP
          --}}}
          TRUE
            BYTE temp.color :
            SEQ
              IF
                pixel.mode = ADD
                INT i.color :

```

```

        SEQ
            i.color := (INT color) + current.color
            temp.color := BYTE (IMIN (IMAX(i.color, 0), max.color))
        TRUE
            temp.color := color
        --{{{ write pixel(s)
        SEQ i = 0 FOR fat.pixels.y
            SEQ j = 0 FOR fat.pixels.x
                screen[ (screen.position + j) +
                    (i TIMES pixels.line)] := temp.color
            --}}}
        :
        --}}}
        --{{{ draw.line
PROC draw.line ( VAL {} INT window, [] BYTE screen,
                VAL INT x1, y1, x2, y2,
                VAL BYTE color )

--{{{ clip.line
PROC clip.line (VAL INT x1, y1, x2, y2, INT result, VAL {}INT window)
    -- decides whether a line is totally on or off screen/window
    --{{{ codes
    VAL code.centre IS #00 :      -- 0000
    VAL code.left  IS #01 :      -- 0001
    VAL code.right IS #02 :      -- 0010
    VAL code.bottom IS #04 :     -- 0100
    VAL code.top   IS #08 :      -- 1000
    --}}}
    INT code1, code2 :
    --{{{ PROC check
    PROC check (VAL INT x, y, INT code)
        VAL x.max IS window[ w.size.x ] :
        VAL y.max IS window[ w.size.y ] :
        SEQ
            IF
                --{{{ x.min <= x < x.max
                (x >= 0) AND (x < x.max)
                    code := code.centre
                --}}}
                --{{{ x < x.min
                x < 0
                    code := code.left
                --}}}
                --{{{ x > x.max
                TRUE --x >= x.max
                    code := code.right
                --}}}
            IF
                --{{{ y.min <= y < y.max
                (y >= 0) AND (y < y.max)
                    SKIP
                --}}}
                --{{{ y < y.min
                y < 0
                    code := code \/ code.top
                --}}}
                --{{{ y >= y.max
                TRUE --y > y.max
                    code := code \/ code.bottom
                --}}}
            :
        --}}}
    SEQ
        check (x1, y1, code1)
        check (x2, y2, code2)
        IF
            --{{{ line lies entirely within window

```

```

      (code1 \ / code2) = 0
      result := in.range
    --}}}
    --{{{ line lies entirely outside window
    (code1 \ / code2) <> 0
      result := not.inrange
    --}}}
    --{{{ partially in window perhaps
    TRUE
      result := part.inrange
    --}}}
:
--}}}
--{{{ slow.draw.line
PROC slow.draw.line ( VAL [] INT window, [] BYTE screen,
                     VAL INT x1, y1, x2, y2,
                     VAL BYTE color )
-- uses Bresenham's integer algorithm to calculate plotting points
-- calls plot to draw actual pixels on the screen
VAL pixels.line IS window[w.pixels.line] :
INT dx, dy :
INT two.dx, two.dy :
INT error :
SEQ
  dx := x2 - x1
  dy := y2 - y1
  IF
    (dx <> 0) OR (dy <> 0)
    SEQ
      --{{{ a line to draw
      IF
        --{{{ dy = 0 -- horizontal line
        dy = 0
          SEQ i = x1 FOR dx + 1
            plot (window, screen, i, y1, color)
          --}}}
        --{{{ dx = 0 -- vertical line
        dx = 0
          SEQ
            IF
              dy > 0
                SEQ i = y1 FOR dy + 1
                  plot (window, screen, x1, i, color)
              TRUE
                SEQ i = y2 FOR (-dy) + 1
                  plot (window, screen, x1, i, color)
            --}}}
        --{{{ dx <> 0 dy <> 0 -- diagonal line
        TRUE
          INT x, y :
          INT delta.y :
          SEQ
            x := x1
            y := y1
            two.dx := dx + dx
            IF
              --{{{ dy > 0
              dy > 0
                SEQ
                  two.dy := dy + dy
                  IF
                    --{{{ dy > dx
                    dy > dx
                      SEQ
                        error := two.dx - dy
                        --{{{ plot line
                        SEQ i = 0 FOR dy + 1

```

```

SEQ
  plot (window, screen, x, y, color )
  IF
    error >= 0
    SEQ
      x := x + 1
      error := error - two.dy
    TRUE
    SKIP
    y := y + 1
    error := error + two.dx
  --}}}
--}}}
--{{{ dy <= dx
TRUE
  SEQ
    error := two.dy - dx
    --{{{ plot line
    SEQ i = 0 FOR dx + 1
    SEQ
      plot (window, screen, x, y, color )
      IF
        error >= 0
        SEQ
          y := y + 1
          error := error - two.dx
        TRUE
        SKIP
        x := x + 1
        error := error + two.dy
      --}}}
    --}}}
--}}}
--{{{ dy < 0
TRUE
  SEQ
    dy := -dy
    two.dy := dy + dy
    IF
      --{{{ dy > dx
      dy > dx
      SEQ
        error := two.dx - dy
        --{{{ plot line
        SEQ i = 0 FOR dy + 1
        SEQ
          plot (window, screen, x, y, color)
          IF
            error >= 0
            SEQ
              x := x + 1
              error := error - two.dy
            TRUE
            SKIP
            y := y - 1
            error := error + two.dx
          --}}}
        --}}}
      --}}}
    --{{{ dy <= dx
    TRUE
    SEQ
      error := two.dy - dx
      SEQ i = 0 FOR dx + 1
      SEQ
        plot (window, screen, x, y, color)
        IF
          error >= 0

```

```

                                SEQ
                                y := y - 1
                                error := error - two.dx
                                TRUE
                                SKIP
                                x := x + 1
                                error := error + two.dy
                                --}}}
                                --}}}
                                --}}}
                                --}}}
                                TRUE
                                plot (window, screen, x1, y1, color)
:
--}}}
--{{{ fast.draw.line
PROC fast.draw.line (VAL [] INT window, [] BYTE screen,
                    VAL INT x1, y1, x2, y2,
                    VAL BYTE color)

-- uses Bresenham's integer algorithm to calculate plotting points
-- points are in increasing values of x
-- only called when the line is known to be on screen / in window and
-- the current pixel size is one
INT dx, dy, two.dx, two.dy, delta.x, delta.y :
INT error, pixel :
VAL pixels.line IS window[ w.pixels.line ] :
SEQ
  dx := x2 - x1                      -- always zero or positive
  dy := y2 - y1
  pixel := (y1 * pixels.line) + x1
  IF
    (dx <> 0) OR (dy <> 0)
    SEQ
      --{{{ a line to draw
      IF
        --{{{ dy = 0                      -- horizontal line
        dy = 0
        SEQ i = pixel FOR dx + 1
        screen[i] := color
        --}}}
        --{{{ dx = 0                      -- vertical line
        dx = 0
        SEQ
          IF
            dy > 0
            SEQ i = 0 FOR dy + 1
            SEQ
              screen[pixel] := color
              pixel := pixel + pixels.line
            TRUE
            SEQ i = 0 FOR (-dy) + 1
            SEQ
              screen[pixel] := color
              pixel := pixel - pixels.line
          --}}}
        --{{{ dx <> 0 AND dy <> 0
        TRUE
        INT delta.y :
        SEQ
          two.dx := dx + dx
          IF
            dy > 0
            delta.y := pixels.line
          TRUE
          SEQ
            dy := -dy

```

```

        delta.y := -pixels.line
two.dy := dy + dy
IF
  --{{{ dy > dx
  dy > dx
  SEQ
    error := two.dx - dy
    --{{{ plot line
    SEQ i = 0 FOR dy + 1
    SEQ
      screen[pixel] := color
      IF
        error >= 0
        SEQ
          pixel := pixel + 1
          error := error - two.dy
        TRUE
        SKIP
        pixel := pixel + delta.y
        error := error + two.dx
      --}}}
    --}}}
  --}}}
  --{{{ dy <= dx
  TRUE
  SEQ
    error := two.dy - dx
    --{{{ plot line
    SEQ i = 0 FOR dx + 1
    SEQ
      screen[pixel] := color
      IF
        error >= 0
        SEQ
          pixel := pixel + delta.y
          error := error - two.dx
        TRUE
        SKIP
        pixel := pixel + 1
        error := error + two.dy
      --}}}
    --}}}
  --}}}
  --}}}
  TRUE
  screen[pixel] := color
:
--}}}
INT x3, y3, x4, y4 :
INT result :
SEQ
  --{{{ swap x1,y1 with x2,y2 if x1 > x2
  IF
    x1 > x2
    SEQ
      x3 := x2
      y3 := y2
      x4 := x1
      y4 := y1
    TRUE
    SEQ
      x3 := x1
      y3 := y1
      x4 := x2
      y4 := y2
  --}}}
clip.line (x3, y3, x4, y4, result, window)
IF

```

```

        (result = in.range)
        --fast.draw.line (window, screen, x3, y3, x4, y4, color)
        slow.draw.line (window, screen, x3, y3, x4, y4, color)
        (result = part.inrange) OR (result = in.range)
        slow.draw.line (window, screen, x3, y3, x4, y4, color)
    TRUE
    SKIP
:
--}}}
--{{{ draw.polyline
PROC draw.polyline ( VAL {} INT window, {} BYTE screen,
                    VAL {}[2]INT points, VAL BYTE color)

    -- calls draw line to draw the lines
    INT x, y :
    SEQ
        x := points[0][0]
        y := points[0][1]
        SEQ i = 1 FOR (SIZE points) - 1
            VAL point IS points[i] :
            SEQ
                draw.line( window, screen, x, y, point[0], point[1], color )
                x := point[0]
                y := point[1]
:
--}}}
--{{{ draw.rectangle
PROC draw.rectangle ( VAL {} INT window, {} BYTE screen,
                    VAL [2][2]INT p, VAL BYTE color)

    -- calls draw line to draw the lines
    INT x, y :
    SEQ
        draw.line( window, screen, p[0][0], p[0][1], p[1][0], p[0][1], color )
        draw.line( window, screen, p[1][0], p[0][1], p[1][0], p[1][1], color )
        draw.line( window, screen, p[1][0], p[1][1], p[0][0], p[1][1], color )
        draw.line( window, screen, p[0][0], p[1][1], p[0][0], p[0][1], color )
:
--}}}
--{{{ spectrum
PROC spectrum ( CHAN OF ANY out )

    SEQ
        SEQ i = 0 FOR 64
            set.colour( out, 0, i, i, 0, 31-(i>>1) )
        SEQ i = 0 FOR 64
            set.colour( out, 0, 64+i, 63, i, i )
        SEQ i = 128 FOR 128
            set.colour( out, 0, i, 0, 63, 0 )
            set.colour( out, 0, 0, 0, 0, 0 )
            set.colour( out, 0, 128, 30, 30, 30 )
:
--}}}
--{{{ center.string
PROC center.string ( {} INT window, {} BYTE screen,
                    VAL INT cx, sy, VAL {} BYTE string,
                    VAL {} INT font )

    INT width :
    SEQ
        string.width( font, string, width )
        window[ w.cursor.y ] := sy
        window[ w.cursor.x ] := cx - (width >> 1)
        write.string( window, screen, string, font )
:
--}}}

```

```
--{{{ place.string
PROC place.string ( [ ] INT window, [ ] BYTE screen,
                    VAL INT sx, sy, VAL [ ] BYTE string,
                    VAL [ ] INT font )

    SEQ
        window[ w.cursor.y ] := sy
        window[ w.cursor.x ] := sx
        write.string( window, screen, string, font )
    :
--}}}}
--{{{ EventProc
PROC EventProc ( CHAN OF ANY Event, in )

    INT synch, address :
    WHILE TRUE
        SEQ
            in ? address
            Ready := 1
            Event ? synch
            DisplayStart := address
            Ready := 0
        :
    --}}}}
--{{{ Buffer
PROC Buffer ( CHAN OF ANY in, out )

    INT temp :
    SEQ
        WHILE TRUE
            SEQ
                in ? temp
                out ! temp
            :
        --}}}}
--{{{ set.text.window
PROC set.text.window ( [ ] INT window,
                       VAL [ ] BYTE string1, string2, VAL [ ] INT font,
                       VAL INT start.x, start.y, size.y )

    SEQ
        string.width( font, string1, window[ w.start.x ] )
        string.width( font, string2, window[ w.size.x ] )
        window[ w.start.x ] := window[ w.start.x ] + start.x
        window[ w.size.x ] := window[ w.size.x ] + 1
        window[ w.start.y ] := start.y
        window[ w.size.y ] := size.y + 1
        window[ w.start ] := (screen.width * start.y) + window[ w.start.x ]
        window[ w.size ] := screen.width * window[ w.size.y ]
        window[ w.pixels.line ] := screen.width
        window[ w.foreground.color ] := 255
        window[ w.background.color ] := 0
        window[ w.cursor.x ] := 0
        window[ w.cursor.y ] := 0
    :
    --}}}}
--{{{ display.text
PROC display.text ( [ ] INT window, [ ] BYTE screen,
                   VAL [ ] BYTE text, VAL [ ] INT font )

    s IS [screen FROM window[ w.start ] FOR window[ w.size ] ] :
    SEQ
        window[ w.cursor.x ] := 0
        window[ w.cursor.y ] := 0
        write.string( window, s, text, font )
    :
    --}}}}
```

```

--{{{ place.numbers
PROC place.numbers ( [[]] BYTE screen, [[]] BYTE char.array,
                    VAL [] BYTE string,
                    VAL INT start.x, start.y, size.x, size.y )

INT x :
SEQ
  x := start.x
  SEQ i = 0 FOR SIZE string
    VAL char IS INT string[i] :
    SEQ
      IF
        --{{{ display space character
        char = (INT ' ')
          VAL source IS char.array[11] :
          MOVE2D( source, 0, 0, screen, x, start.y, size.x, size.y )
        --}}}
        --{{{ display decimal character
        char = (INT '.')
          VAL source IS char.array[10] :
          MOVE2D( source, 0, 0, screen, x, start.y, size.x, size.y )
        --}}}
        --{{{ display number character
        TRUE
          VAL source IS char.array[char - (INT '0')] :
          MOVE2D( source, 0, 0, screen, x, start.y, size.x, size.y )
        --}}}
      x := x + size.x
    :
  --}}}
--{{{ MAX
REAL32 FUNCTION MAX ( VAL REAL32 a, b )

REAL32 r :
VALOF
  IF
    a > b
      r := a
    TRUE
      r := b
  RESULT r
:
--}}}
--{{{ MIN
REAL32 FUNCTION MIN ( VAL REAL32 a, b )

REAL32 r :
VALOF
  IF
    a < b
      r := a
    TRUE
      r := b
  RESULT r
:
--}}}
--}}}
--{{{ variables
[w.length] INT window :
[4][w.length] INT text.window :
[12][char.height][char.width] BYTE number :

INT pixel.mode :
INT mask.mode :

CHAN OF ANY synch, synch1 :
--}}}

```

```

VAL font IS SYS6 :
SEQ
  --{{{ initialize
  set.B408( 0, 0, 0, 0, 0 )
  --{{{ set up B409
  set.timing( toAnalog, width, height, line.frequency,
              frame.rate, pixel.clock, interlace )
  spectrum( toAnalog )
  --}}}
  --{{{ generate display table for n characters
  [w.length] INT window :
  [12][char.width*char.height] BYTE n RETYPES number :
  SEQ
    SEQ i = 0 FOR 12
      SEQ j = 0 FOR char.width*char.height
        n[i][j] := 0 (BYTE)
      window := [ 0, char.width*char.height, char.width,
                  0, 0, char.width, char.height, 255, 0, 0, 0 ]
      SEQ i = 0 FOR 10
        display.text( window, n[i], [ BYTE( i+(INT '0') ) ], font )
        display.text( window, n[10], ".", font )
        display.text( window, n[11], " ", font )
      --}}}
  --{{{ initial display
  [ ] INT int.screen RETYPES screen.map :
  SEQ i = 0 FOR (20*65536)/4
    int.screen[i] := 0

  window := [ 0, screen.size, screen.width, 0, 0,
              screen.width, screen.height, 255, 0, 1, 1 ]
  screen IS [screen.map FROM screen.offset[0] FOR screen.size] :
  SEQ
    --{{{ draw first box
    draw.line( window, screen, 31, 111, 288, 111, 255(BYTE) )
    draw.line( window, screen, 288, 111, 288, 368, 255(BYTE) )
    draw.line( window, screen, 31, 368, 288, 368, 255(BYTE) )
    draw.line( window, screen, 31, 111, 31, 368, 255(BYTE) )
    --}}}
    --{{{ draw second box
    draw.line( window, screen, 351, 111, 608, 111, 255(BYTE) )
    draw.line( window, screen, 608, 111, 608, 368, 255(BYTE) )
    draw.line( window, screen, 351, 368, 608, 368, 255(BYTE) )
    draw.line( window, screen, 351, 111, 351, 368, 255(BYTE) )
    --}}}
    --{{{ draw text
    SEQ
      center.string( window, screen, 320, 1, "FPA Seeker Emulator", font )
      center.string( window, screen, 160, 70, "Raw FPA Image", font )
      center.string( window, screen, 480, 70, "Processed Image", font )

      --place.string( window, screen, 0, 400, "Range: ", font )
      --place.string( window, screen, 0, 440, "Sim Time: ", font )
      --place.string( window, screen, 350, 400, "Frame Rate:", font )
      --place.string( window, screen, 350, 440, "Frame Number:", font )

      --set.text.window( text.window[0], "Range:", "0123456.789", font, 16,400,
32 )
      --set.text.window( text.window[1], "Sim Time: ", "123.456", font, 16,440,
32 )
      --set.text.window( text.window[2], "Frame Rate:  ", "123", font, 350,400,
32 )
      --set.text.window( text.window[3], "Frame Number: ", "123", font, 350,440,
32 )
    --}}}

    window := [0, screen.size, screen.width, 32, 112, 128, 128,
               0, 0, 2, 2 ]

```

```

--[screen.map FROM screen.offset[1] FOR screen.size] := screen
--[screen.map FROM screen.offset[2] FOR screen.size] := screen
--[screen.map FROM screen.offset[3] FOR screen.size] := screen
--}}}
set.B408( 0, 0, 0, 1, 0 )
--}}}
--{{{ run
screen IS [screen.map FROM screen.offset[0] FOR screen.size ] :
[max.command.line]INT command.line :
INT length :
INT color.shift :
WHILE TRUE
  SEQ
    --{{{ initialize
    color.shift := 1
    --}}}
    fromHost ? length::command.line

CASE command.line[0]
--{{{ plot point
c.plot
  x      IS command.line[1] :
  y      IS command.line[2] :
  VAL color IS (command.line[3] >> color.shift):
  plot (window, screen, x, y, (BYTE color))
--}}}
--{{{ line
c.line
  x1      IS command.line[1] :
  y1      IS command.line[2] :
  x2      IS command.line[3] :
  y2      IS command.line[4] :
  VAL color IS (command.line[5] >> color.shift):
  draw.line (window, screen, x1, y1, x2, y2, (BYTE color))
--}}}
--{{{ clear
--}}}
--{{{ mask.mode
c.mask.mode
  INT temp :
  SEQ
    mask.mode := command.line[1]
    temp := window[w.foreground.color]
    temp := temp /\ #F0 -- mask off 4 bits
    temp := temp \/ mask.mode
    window[w.foreground.color] := temp
--}}}
--{{{ pixel.mode
c.pixel.mode
  INT temp :
  SEQ
    pixel.mode := command.line[1]
    temp := window[w.foreground.color]
    temp := temp /\ #0F -- mask off 4 bits
    temp := temp \/ (pixel.mode << 4)
    window[w.foreground.color] := temp
--}}}
--}}}
:

```