X−Ray Multi Mirror Mission Optical Monitor

Detailed Design Document for Digital Processing Unit Flight Software: C−code

Document: XMM−OM/UCSB/ML/0013 Version 1.3

Timothy P. Sasseen
University of California, Santa Barbara

Cheng Ho
Los Alamos National Laboratory

Jamie A. Kennea
University of California, Santa Barbara

22 May, 2000

This page intentionally left blank.

X−Ray Multi Mirror Mission Optical Monitor Detailed Design Document for Digital Processing Unit Flight Software: C−code

Document: XMM−OM/UCSB/ML/0013 Version 1.3

Summary:

This document provides a functional description of the for Release 11 of the C−code routines in the Optical Monitor Digital Processing Unit flight software. For detailed explanations of recent changes to the software detailed in this document please refer to "Summary of DPU Flight Software Changes 5/00−8/00" (XMM−OM/UCSB/TC/0052) and "DPU Flight Software test reports for Release 11" (XMM−OM/UCSB/RP/0007).

Record of Changes

Version	Date	Released by
1.0	Oct. 26, 1998	T. Sasseen
1.1	Sept. 22, 1999	T. Sasseen
1.2	May 24, 2000	J. Kennea
1.3	October 5, 2000	J. Kennea

Table of Contents

Documents referenced:

XMM−OM/UCSB/ML/0012, User Manual Part 1B, Experiment on Board Software − DPU

XMM−OM/UCSB/ML/0014, Detailed Design Document for Digital Processing Unit Flight Software: Operating System and Assembly Code

How To Use This Document:

This document contains primarily a listing and functional description of each software routine for the DPU C−code software. An overview of how the DPU software works is given in XMM−OM/UCSB/ML/0012, User Manual Part 1B, Experiment on Board Software − DPU, which is the first document that should be consulted for general questions. That document provides a general description of the DPU software.

For someone wanting to understand the DPU code, start with XMM−OM/UCSB/ML/0012, then refer to this document, then refer to the code itself for increasing levels of detail.

The specific comments on each routine listed here are generally taken from the header files contained in each code routine. These headers are updated in principle with each functional change. In case of conflicts, the headers contained in new code releases are likely to be more current than this document.

This document details release 10, and is current as of that release.

Overview:

The DPU flight software is divided into two main parts, the operating system and the higher level c−language code. The operating system was written by James Klarkowski of Sandia National Laboratory, while most of the high level C−code was written by Dr. Cheng Ho of Los Alamos National Laboratory. In general, the operating system is written in assembly language, but includes two main C−code routines, cwhite.c and cred.c.

This document details the higher level c−language code. It is organized by processor, white and red. Blue processors are entirely assembly language operating system code.

In addition to the division between operating system and higher−level C codes, codes are divided into boot codes that operate only during DPU startup, and full codes that would be uploaded following a successful boot process. Also in the flight software tree are a library of "include" routines in directory called "include."

How to compile the DPU software

In the directory tree dpu/util are the make files to compile the DPU codes. The primary makefile used at this time is “makedsp −c”. This calls a number of other make files within the util directory. In the course of compilation several intermediate compilation products are created by the DSP gcc compiler and stored in directories with names

cln/
lod/
lst/
cld/
map/

Generally these can be ignored as they are more useful for development and debugging. Their exact contents can be found in documentation on the gcc DSP compiler. The executable, uploadable code is stored in the dpu/eeprom directory, in files ending in cmd.0. The exact loading of these files into the DPU depends on the GSE configuration. The boot codes are stored within the rest of the C−codes and are placed in a directory called dpu/*/boot.

Here is an example of how we configure and compile the DPU codes. We use CVS to store the latest official version, so our first step is to check out the codes. We strongly recommend ESA use the same directory tree structure that is used by the OM/DPU team, to facilitate future communication and deliveries. This information is also available from ftp://xmmom.physics.ucsb.edu/pub/howto_release.html.

We set environment variables:

```
setenv MYXMMOM /tmp/xmmom
setenv XMMOMHOME /tmp/xmmom
setenv XMMOMSIMHOME /tmp/xmmom/sim
setenv XMMOMSDTHOME /tmp/xmmom/sdt
setenv XMMOMDPUHOME /tmp/xmmom/dpu
setenv XMMOMWDSPHOME /tmp/xmmom/dpu/development/white
setenv XMMOMRDSPHOME /tmp/xmmom/dpu/development/red
setenv XMMOMBDSPHOME /tmp/xmmom/dpu/development/blue
set path = ( /home/pjs1/bin /usr/local/bin /usr/local/X11R6/bin /usr/teTeX/bin/sparc−
solaris2.5 /opt/sparc/bin /usr/bin /bin /usr/ucb /usr/local/sbin /sbin /usr/sbin
/usr/X11R6/bin /usr/openwin/bin /usr/local/X11/bin /usr/local/dsp/bin /usr/local/gnu/bin
/tmp/xmmom/dpu/util/bin )
```

To checkout and compile the DPU software:

```
cd $XMMOMHOME
cvs checkout dpu/util
cd dpu
```

```
mkdir development
cd development
cvs checkout red white blue include
mkdir eepromcd ../util/src
rehash
makedsp −c
```

Then wait for a long time if you have a slow Sun.

Summary of Individual routines

Include:

The following routines are in the “include” directory tree:

```
NAME:

     dpu_util.h

PURPOSE:

CALLING SEQUENCE:
     #include "dpu_util.h"

CALLED FROM:
     setup.h

INCLUDE FILES:

RESTRICTIONS:
     THIS IS NOT THE REAL DPU_util.h FILE       jmk 081192

MODIFICATION HISTORY:
     11Sep95   ADW   add standard comment header

RCS $Id: dpu_util.h,v 1.8 1996/06/13 17:11:31 welty Exp $
```

NAME:

dpucfg_map.h

PURPOSE:
This file contains an address map of the "proc" area of global memory.

CALLING SEQUENCE:
#include "dpucfg_map.h"

CALLED FROM:
cwhite.c
setup.h

INCLUDE FILES:

RESTRICTIONS:
asm_headers script requires lines starting with #define<tab><tab>

MODIFICATION HISTORY:
```
08Dec93   SH    Added standard comment header
11Sep95   ADW   add standard comment header
112397      jrmk  Added arrays for the output of FEC errors from
            bootStrap FEC checking.
112697      jrmk  Added flags to switch FEC code output with
```

RCS $Id: dpucfg_map.h,v 1.10 1998/09/23 22:21:06 ho Exp $

Flag to enable(1) disable(0) FEC checking of IC_LOAD_CODE commands
Map of the PROC area
The following definitions are setup for B1 and B2 fast mode. CH 18Mar94
red_error & red_semaphore are not used in reddsp.c, but are in red.asm.
words_per_bus_access: use when White & Red share WORDS_PER_BUS_ACCESS
see white_default.h
Remove when RED_WORDS_PER_BUS_ACCESS is used by Red (red_default.h)
locations for White log stuff. 27Oct95: only keep_a_log still used ADW
locations for variables passed to Red by White for shift-and-add
not used (commented passing between White & Red)
allocation of storage space for heartbeat returns from the Red and Blue processors and error returns from the Blue processors

Proposed PROC variables for Red
THESE ARE ONLY PLACE HOLDERS
added by SDH
Starting address of Yin memory area
Starting address of Red accumulation memory area
Number of Red pixels in a Red frame

```
NAME:

      global_access.c

 PURPOSE:
      Read and write to and from various integer memory areas.

 FUNCTIONS IN THIS FILE:
      read_proc_int
      read_proc_int_array
      write_proc_int
      write_proc_int_array
      read_bw_int
      read_bw_int_array
      write_bw_int
      write_bw_int_array
      read_sw_int
      read_sw_int_array
      write_sw_int
      write_sw_int_array

 INCLUDE FILES:

 MODIFICATION HISTORY:
      12Jul95   ADW   add standard comment header

 RCS $Id: global_access.c,v 1.11 1998/09/03 06:46:42 ho Exp $

*

Define CHECK_BOUNDS if desired.
```

```
NAME:

      read_proc_int

 PURPOSE:
      Read an integer from proc memory.

 CALLING SEQUENCE:
      n = read_proc_int(base, destination);

 INPUTS:
      base
      destination

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns 0 if address is out of allowed range [if tested...]
                     1 otherwise.)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      access_alert     [cwhite.c]
      read_int

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jul95   ADW   add standard comment header


   If the CHECK_BOUNDS flag is defined, make sure the input proc address
   is within the allowed limits.  Return 0 if it is not.
   If the address is acceptable, read an integer, and return 1.
```

```
NAME:

      read_proc_int_array

 PURPOSE:
      Read an array of integers from proc memory.

 CALLING SEQUENCE:
      n = read_proc_int_array(base, ysize, yskip, xsize, destination);

 INPUTS:
      base
      ysize
      yskip
      xsize
      destination

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns 0 if address is out of  allowed range [if
tested...]
                    n, the number of integers read, otherwise.)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      access_alert          [cwhite.c]
      read_int_array

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jul95   ADW   add standard comment header


   If the CHECK_BOUNDS flag is defined, make sure the input proc address
   is within the allowed limits.  Return 0 if it is not.
   If the address is acceptable, read the specified integers
   and return the number of integers read.
```

```
NAME:

      write_proc_int

 PURPOSE:
      Write an integer to proc memory.

a999* CALLING SEQUENCE:
      n = write_proc_int(to, from);

 INPUTS:
      to
      from

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns 0 if address is out of allowed range [if tested...]
                     1 otherwise.)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      access_alert     [cwhite.c]
      write_int

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jul95   ADW   add standard comment header



   If the CHECK_BOUNDS flag is defined, make sure the input proc address
   is within the allowed limits.  Return 0 if it is not.
   If the address is acceptable, write an integer and return 1.
```

```
NAME:

      write_proc_int_array

 PURPOSE:
      Write an array of integers to proc memory.

 CALLING SEQUENCE:
      n = write_proc_int_array(to, ysize, yskip, xsize, from);

 INPUTS:
      to
      ysize
      yskip
      xsize
      from

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns 0 if address is out of  allowed range [if
tested...]
                     n, the number of integers written, otherwise.)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      access_alert           [cwhite.c]
      write_int_array

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jul95   ADW   add standard comment header



   If the CHECK_BOUNDS flag is defined, make sure the input proc address
   is within the allowed limits.  Return 0 if it is not.
   If the address is acceptable, write the specified integers
   and return the number of integers written.
```

```
NAME:

       read_bw_int

 PURPOSE:
      Read an integer from bw memory.

 CALLING SEQUENCE:
      n = read_bw_int(base, destination);

 INPUTS:
      base
      destination

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns 0 if address is out of  allowed range [if
tested...]
                     1 otherwise.)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      access_alert
      read_int

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jul95   ADW   add standard comment header



   If the CHECK_BOUNDS flag is defined, make sure the input bw address
   is within the allowed limits.  Return 0 if it is not.
   If the address is acceptable, read an integer, and return 1.
```

```
NAME:

      read_bw_int_array

 PURPOSE:
      Read an array of integers from bw memory.

 CALLING SEQUENCE:
      n = read_bw_int_array(base, ysize, yskip, xsize, destination);

 INPUTS:
      base
      ysize
      yskip
      xsize
      destination

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns 0 if address is out of  allowed range [if
tested...]
                     n, the number of integers read, otherwise.)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      access_alert          [cwhite.c]
      read_int_array

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jul95   ADW   add standard comment header


   If the CHECK_BOUNDS flag is defined, make sure the input bw address
   is within the allowed limits.  Return 0 if it is not.
   If the address is acceptable, read the specified integers
   and return the number of integers read.
```

```
NAME:

       write_bw_int

 PURPOSE:
      Write an integer to bw memory.

 CALLING SEQUENCE:
      n = write_bw_int(to, from);

 INPUTS:
      to
      from

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns 0 if address is out of  allowed range [if
tested...]
                     1 otherwise.)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      access_alert     [cwhite.c]
      write_int

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jul95   ADW   add standard comment header



   If the CHECK_BOUNDS flag is defined, make sure the input bw address
   is within the allowed limits.  Return 0 if it is not.
   If the address is acceptable, write an integer and return 1.
```

```
NAME:

      write_bw_int_array

 PURPOSE:
      Write an array of integers to bw memory.

 CALLING SEQUENCE:
      n = write_bw_int_array(to, ysize, yskip, xsize, from);

 INPUTS:
      to
      ysize
      yskip
      xsize
      from

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns 0 if address is out of  allowed range [if
tested...]
                     n, the number of integers written, otherwise.)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      access_alert           [cwhite.c]
      write_int_array

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jul95   ADW   add standard comment header


   If the CHECK_BOUNDS flag is defined, make sure the input bw address
   is within the allowed limits.  Return 0 if it is not.
   If the address is acceptable, write the specified integers
   and return the number of integers written.
```

```
NAME:

      read_sw_int

 PURPOSE:
      Read an integer from sw memory.

 CALLING SEQUENCE:
      n = read_sw_int(base, destination);

 INPUTS:
      base
      destination

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns 0 if address is out of  allowed range [if
tested...]
                     1 otherwise.)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      access_alert     [cwhite.c]
      read_int

 CALLED FROM:
      compress_data_task   [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jul95   ADW   add standard comment header


   If the CHECK_BOUNDS flag is defined, make sure the input sw address
   is within the allowed limits.  Return 0 if it is not.
   If the address is acceptable, read an integer, and return 1.
```

```
NAME:

      read_sw_int_array

 PURPOSE:
      Read an array of integers from sw memory.

 CALLING SEQUENCE:
      n = read_sw_int_array(base, ysize, yskip, xsize, destination);

 INPUTS:
      base
      ysize
      yskip
      xsize
      destination

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns 0 if address is out of  allowed range [if
tested...]
                     n, the number of integers read, otherwise.)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      access_alert          [cwhite.c]
      read_int_array

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jul95   ADW   add standard comment header


   If the CHECK_BOUNDS flag is defined, make sure the input sw address
   is within the allowed limits.  Return 0 if it is not.
   If the address is acceptable, read the specified integers
   and return the number of integers read.
```

```
NAME:

      write_sw_int

 PURPOSE:
      Write an integer to sw memory.

 CALLING SEQUENCE:
      n = write_sw_int(to, from);

 INPUTS:
      to
      from

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns 0 if address is out of allowed range [if tested...]
                     1 otherwise.)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      access_alert     [cwhite.c]
      write_int

 CALLED FROM:
      as of 17May96, this routine is not used.

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jul95   ADW   add standard comment header



   If the CHECK_BOUNDS flag is defined, make sure the input sw address
   is within the allowed limits.  Return 0 if it is not.
   If the address is acceptable, write an integer and return 1.
```

```
NAME:

      write_sw_int_array

 PURPOSE:
      Write an array of integers to sw memory.

 CALLING SEQUENCE:
      n = write_sw_int_array(to, ysize, yskip, xsize, from);

 INPUTS:
      to
      ysize
      yskip
      xsize
      from

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns 0 if address is out of  allowed range [if
tested...]
                     n, the number of integers written, otherwise.)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      access_alert           [cwhite.c]
      write_int_array

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jul95   ADW   add standard comment header


   If the CHECK_BOUNDS flag is defined, make sure the input sw address
   is within the allowed limits.  Return 0 if it is not.
   If the address is acceptable, write the specified integers
   and return the number of integers written.
```

```
NAME:

      global_zero          function

 PURPOSE:

      generic C function checking ranges for single word access

 CALLING SEQUENCE:
      integer_variable = global_zero(base,length);

 INPUTS:
      base   = base address in global memory
      length = length to be zeroed

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns 0)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      zero_frame    [sun_util.c]

 CALLED FROM:
      initialize    [su_initialize.c]
      scanbs        [su_scanbs.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      18May95   ADW   add standard comment header, structured English
comments
```

```
NAME:

      idiot.h

 PURPOSE:
      Definitions of ICU<->DPU protocol datatypes

      From:  Document        XMM-OM/MSSL/SP/0014
             Latest version                   .03  24Mar95

      After a serious discussion, JMK and I (CH) arrived at a name for
      official data type definition header file. In the spirit of
      XOMBI, we propose this file to be named: ICU - DPU Interface
      Official Translation. In lower case, this is idiot.h

      I suggest this header file to include all definitions
      which is used for communication of between different software
      domains, i.e. between ICU and DPU, between DPUOS and whitedsp,
      between DPU codes and SDT codes.

 CALLING SEQUENCE:
      #include <idiot.h>

 CALLED FROM:
      setup.h    [included by many files]
      cwhite.c

 INCLUDE FILES:

 RESTRICTIONS:
      asm_headers script requires lines starting with #define<tab><tab>
      csh_headers script requires variable name to be no longer than 18
chs
      vsw_headers script does not allow in-line comments

 NOTES:

   Each symbol is headed by a two letter prefix, e.g. IC. The first
letter
   indicates the origin of the data, I=ICU, D=DPU, the second letter
   indicates the nature of the data or the mechanism of data
transmission:
   C=COMMAND, D=DATA, A=ALERT, P=PRIORITY, R=RAM.

   ABRV                       FULL SYNTAX
   ----                       -----------
   BEGOF                      BEGINNING_OF
   BFAST                      BLUE_FAST_MODE_AREA
   CID                        COMMAND_ID
   CLK                        CLOCK
   CMD                        COMMAND
   CMPRS                      COMPRESS
   CONF                       CONFIGURE or CONFIGURATION
   DIAGS                      DIAGNOSTICS
   DET                        DETECTOR
   DLVR                       DELIVER
   DPU                        DIGITAL_PROCESSING_UNIT
   DPUOS                      DIGITAL_PROCESSING_UNIT_OPERATING_SYSTEM
   DQ                         DATA_QUEUE
   DSP                        DIGITAL_SIGNAL_PROCESSOR
   ENBL                       ENABLE
   ENG                        ENGINEERING
   EOS                        END_OF_SWAP_UNIT
   EOT                        END_OF_TASK
   EVENTS                     EVENTS_PROCESSING
```

```
   EXP                           EXPOSURE
   EXTRCT                        EXTRACT
   E_ALERT                       ERROR_ALERT
   FAQ                           FIELD_ACQUISITION
   FILT                          FILTER
   FLD                           FIELD (absolute pointing)
   FST                           FAST
   GADE                          GLOBAL_ADDRESS_ERROR
   GEN                           GENERATE
   GS                            GUIDE_STARS
   HBEAT                         HEART_BEAT
   ICU                           INSTRUMENT_CONTROL_UNIT
   IMG                           IMAGE
   INIT                          INITIALIZE
   LNK                           LINK
   LRM                           LOCAL_RAM
   MEM                           MEMORY (as in windows)
   NORM                          NORMALIZATION
   N                             AND
   PROG                          PROGRAM
   Q                             QUEUE
   RCVR                          RECEIVER
   REF                           REFERENCE
   REQ                           REQUEST
   SCI                           SERIAL_COMMUNICATION_INTERFACE
   SCI_WDW                       SCIENCE_WINDOW
   SEL                           SELECT
   S_ALERT                       STATUS_ALERT
   SYNCH                         SYNCHRONIZE/SYNCHRONIZATION
   TRH                           TRACKING_HISTORY
   TRK                           TRACKING
   TST                           TEST
   WDW                           WINDOW

 MODIFICATION HISTORY:
      07Oct93   CH    created
      08Dec93   SH    Added standard comment header
      11Sep95   ADW   add standard comment header
      04May96   CH    reorganized to have consistent abbreviations
      07May96   CH    rearrange DA's

 RCS $Id: idiot.h,v 1.19 1998/03/21 02:02:47 xmmom Exp $

DPU priority data type DP/0XA2yy
These three are place holders, CH050796, reserved
DPU regular data type DD/0XA3yy
ICU to DPU command IC/0XA4yy
Basic DPUOS manipulation and IO commands
commented out by CH 01/12/98
#define           IC_SEL_SCI_LNK_TST        0XA40C

Memory managements
commented out by CH 01/12/98
#define           IC_ENBL_RAM_TST           0XA412
#define           IC_ENBL_ROM_TST           0XA413

Exposure configuration commands
place holder   CH07/07/97
Major task invocation commands
DPU Data Alert data type DA/0XA5yy
Synchronous Alert
Asynchronous health status alert
Must sent alerts
major milestone alerts
minor milestone alerts, may be turned off
Asynchronous exception alerts
```

```
exception alerts from colored DSPs
global memory access exception alerts
DPU RAM DUMP data type DR/0XA6yy
```

```
NAME:

      macro.h

 PURPOSE:
      Define function macros

 CALLING SEQUENCE:
      #include <macro.h>

 CALLED FROM:
      setup.h

 INCLUDE FILES:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      08Dec93   SH    Added standard comment header
      29Nov94   jmk   added TRUE and FALSE
      11Sep95   ADW   add standard comment header

 RCS $Id: macro.h,v 1.4 1996/01/11 23:42:08 welty Exp $
```

```
NAME:

      mic_restore_params.h

 PURPOSE:
      Parameter definitions for parameters used in su_restore_stars.c

 CALLING SEQUENCE:
      #include "mic_restore_params.h"

 CALLED FROM:
      setup.h

 INCLUDE FILES:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      08Dec93   SH    Added standard comment header
      11Sep95   ADW   add standard comment header
      21Aug99   CH     added difference for prime/redundant DPU

 RCS $Id: mic_restore_params.h,v 1.5 1999/08/21 06:49:46 Ho Exp $
```

```
NAME:

      our_own_types.h

 PURPOSE:
      typedefs used in DPU WHITE DSP code

 CALLING SEQUENCE:
      #include "our_own_types.h"

 CALLED FROM:
      setup.h

 INCLUDE FILES:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      09Feb93   DTL   Created
      08Dec93   SH    Added standard comment header
      11Sep95   ADW   add standard comment header

 RCS $Id: our_own_types.h,v 1.9 1996/08/03 13:55:08 welty Exp $

Enable/disable usage of floats
   Record positions to reciprocal of given values (pixels):
```

```
NAME:

      red_control.h

 PURPOSE:

 CALLING SEQUENCE:
      #include "red_control.h"

 CALLED FROM:
      setup.h

 INCLUDE FILES:

 RESTRICTIONS:
      asm_headers script requires lines starting with #define<tab><tab>

 MODIFICATION HISTORY:
      13Nov94   SDH   file header added
      29Sep95   ADW   change for shift-and-add on Red DSP
      14Nov95   CH    renumber red task_ids

 RCS $Id: red_control.h,v 1.6 1997/09/09 15:23:04 ho Exp $
```

```
NAME:

      red_default.h

 PURPOSE:
      Default values of parameters used in Red DSP code

 CALLING SEQUENCE:
      #include "red_default.h"

 INCLUDE FILES:

 CALLED FROM:
      setup.h

 RESTRICTIONS:

 MODIFICATION HISTORY:
      28Nov94   CH    restructured header files
      03Oct95   ADW   change for shift-and-add on Red DSP

 $Id: red_default.h,v 1.10 1998/09/12 15:22:42 ho Exp $


Defines related to Red Operation (used on Red and White DSPs)
Symbolic constants used for RED_SEMAPHORE values
General Red symbolic constants
Red raw data mode starting address
Symbolic constants related to cosmic ray filtering
                                            ADW 25Oct95
--------------------------------------------------------------------
Parameters used exclusively in su_saa and advice on their adjustment:
CH  11Oct93: These two numbers were 16*16.  Changed to 4*64
ADW 05Oct95: moved to red_default.h
--------------------------------------------------------------------
```

```
NAME:

      red_shared_variables.h

 PURPOSE:
      Default values of variables used in Red DSP task code

 CALLING SEQUENCE:
      #include "red_shared_variables.h"

 CALLED FROM:
      setup.h

 INCLUDE FILES:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      11Sep95   ADW   change for shift-and-add on Red DSP

 RCS $Id: red_shared_variables.h,v 1.7 1998/09/19 10:01:06 ho Exp $


Declare the global variables used in the shift-and-add process.
```

```
NAME:

      vbtwl.def

 PURPOSE:
      This file contains the Variable Block Tiered Word Length
      scheme global variable definitions.

      Cheng Ho,
      Original scheme 1990, developed for the XMMOM project.
      Version 0.x, 1991, 1992, 1995
      Ver. 1.0  April, 19, 1996

      Copyright, 1996

 CALLING SEQUENCE:
      #include "vbtwl.def"

 CALLED FROM:
      su_compress.c

 INCLUDE FILES:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      26Jun96   ADW   add standard comment header

 RCS $Id: red_shared_variables.h,v 1.7 1998/09/19 10:01:06 ho Exp $
```

```
NAME:

      sci_com.h

 PURPOSE:
      SCI communication parameters between DSPs

 CALLING SEQUENCE:
      #include "sci_com.h"

 CALLED FROM:
      setup.h

 INCLUDE FILES:

 RESTRICTIONS:
      asm_headers script requires lines starting with #define<tab><tab>

 MODIFICATION HISTORY:
      28nov94   CH    restructured header files
      07Feb95   JMK   addition of expected responses (passed in global
ram)
                      to requests for heartbeat message from blues and
red
      11Sep95   ADW   add standard comment header
      02Oct95   ADW   add Red DSP task commands

 RCS $Id: sci_com.h,v 1.9 1999/08/21 06:54:20 ho Exp $
```

```
NAME:

      setup.h

 PURPOSE:
      Setup to run swap code on DPU or in simulation mode

 CALLING SEQUENCE:
      #include "setup.h"

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:
      idiot.h
      our_own_types.h
      stdio.h
      sun_util.h
      dpu_util.h
      dpucfg_map.h
      sci_com.h
      white_control.h
      white_default.h
      red_control.h
      red_default.h
      macro.h
      white_shared_variables.h
      red_shared_variables.h
      mic_restore_params.h

 SUBROUTINES CALLED:

 CALLED FROM:
      cwhite.c
      whitedsp.c
      All swap units

 RESTRICTIONS:

 MODIFICATION HISTORY:
      11Feb93   DTL
      06Dec93   SDH   Added comment header; moved include of macro.h
                      from shared_data.h to setup.h
      31May95   ADW   Fix comments.

 RCS $Id: setup.h,v 1.5 1999/08/21 07:05:37 ho Exp $
```

```
NAME:

      sim_blue_local_ram.h

 PURPOSE:
      Definition of the Blue DSP local RAM variables

 CALLING SEQUENCE:
      #include <sim_blue_local_ram.h>

 CALLED FROM:
      shared_io_functions.c
      sun_util.c

 INCLUDE FILES:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      06Apr94   CH
      11Sep95   ADW   add standard comment header

 RCS $Id: sim_blue_local_ram.h,v 1.5 1996/03/20 18:55:02 welty Exp $


This section is related to the MIC/BlueDSP simulation.
Declare the variables in blue_local_ram.
```

```
NAME:

      sun_util.h

 PURPOSE:
      Provides C prototypes for the basic Software/Hardware
      Handles required for the operation of the SNL DPU.

 CALLING SEQUENCE:
      #include <sun_util.h>

 CALLED FROM:
      setup.h

 INCLUDE FILES:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      11Aug92   jmk   created
      08Dec93   SH    Added standard comment header
      11Sep95   ADW   add standard comment header

 RCS $Id: sun_util.h,v 1.5 1996/06/13 17:11:34 welty Exp $
```

```
NAME:

      vbtwl.h

 PURPOSE:
      This file contains the Variable Block Tiered Word Length
      scheme structural definitions.


 CALLING SEQUENCE:
      #include "vbtwl.h"

 CALLED FROM:
      su_compress.c

 INCLUDE FILES:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      11Sep95   ADW   add standard comment header
      Cheng Ho,
      Original scheme 1990, developed for the XMMOM project.
      Version 0.x, 1991, 1992, 1995
      Ver. 1.0  April, 19, 1996

      Copyright, 1996

 RCS $Id: vbtwl.h,v 1.3 1996/07/15 17:58:38 welty Exp $
```

```
NAME:

      white_control.h

 PURPOSE:
      Control values/parameters between DPUOS and whitedsp

 CALLING SEQUENCE:
      #include "white_control.h"

 CALLED FROM:
      setup.h

 INCLUDE FILES:

 RESTRICTIONS:
      asm_headers script requires lines starting with #define<tab><tab>

 MODIFICATION HISTORY:
      06Oct93   CH    replace the 0x00001 bit to PRIORITY_QUEUE_STATUS
      28nov94   CH    restructured header files
      11Sep95   ADW   add standard comment header
      03Nov95   ADW   add COMPRESS_DATA task & SWAP_COMPRESS su indices
      14Nov95   CH    renumber task_id for faster checking in whitedsp()
      12Mar96   ADW   add engineering mode support

 RCS $Id: white_control.h,v 1.14 1999/08/21 07:02:14 ho Exp $

Interpretation of bits in control_word
                                          reddsp/send_dp_data too - ADW
Task indices
Swap unit indices (used by the load_and_go function)
Red DSP shift-and-add status values
Engineering mode functions
Define SCRUB_KEY0 (!=0) and startup condition SCRUB_KEY1
Define compilation flags for tarcking algorithm for magnifier and grism
Define compilation flag for ENG4 output chunks
Define compilation flag for restoration coeff for prime/redundant
channels
Set a flag for the special single frame mode
```

```
NAME:

      white_default.h

 PURPOSE:
      Default values of parameters used in White DSP (swap unit) code

 CALLING SEQUENCE:
      #include "white_default.h"

 CALLED FROM:
      setup.h

 INCLUDE FILES:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      28nov94   CH    restructured header files
      11Sep95   ADW   add standard comment header
      05Oct95   ADW   remove stuff now in red_default.h & proc_default.h

 RCS $Id: white_default.h,v 1.20 1999/09/20 05:22:57 ho Exp $

Look here for:
DPU memory layout
      base of compress data buffer
            base of fast data buffer
SAVE_GS_COORDS is used by the preprocessor to load code that would rely
  on I/O from global RAM rather than shared_externals variables for
access
  to guide star coordinates
Frame and exposure times
Detector dimension in units of binned (2 x 2) pixels
Location and size of log
Address offset constants for housekeeping data blocks
WORDS_PER_BUS_ACCESS is supposed to be a conservative estimate of the
number of words that can flow across the global bus per DPU cycle
-----------------------------------------------------------------------
Parameters used exclusively in su_calbgd and advice on their adjustment:
-----------------------------------------------------------------------
-----------------------------------------
Parameters used exclusively in su_caldrft:
-----------------------------------------
------------------------------------------------------------------------
-
Parameters used exclusively in su_dowindow and advice on their
adjustment:
------------------------------------------------------------------------
-
----------------------------------------
Parameters used exclusively in su_getref:
----------------------------------------
------------------------------------------
Parameters used exclusively in su_locategs:
------------------------------------------
-------------------------------------------------------
Parameters used exclusively in su_makeags_id parameters:
-------------------------------------------------------
-----------------------------------------------
Parameters used exclusively in su_restore_stars:
-----------------------------------------------
--------------------------------------------------------------------
Parameters used exclusively in su_saa and advice on their adjustment:
CH  11Oct93: These two numbers were 16*16.  Changed to 4*64
```

```
ADW 05Oct95: moved to red_default.h
--------------------------------------------------------------------
-----------------------------------------------------------------------
Parameters used exclusively in su_scanbs and advice on their adjustment:
A significant reduction in global bus access time might
-----------------------------------------------------------------------
-----------------------------------------------------------------------
Parameters used exclusively in su_scangs and advice on their adjustment:
-----------------------------------------------------------------------
-----------------------------------------------
Parameters used exclusively in su_compress:
-----------------------------------------------
------------------------------------------------------------------------
-
Parameters used in more than one swap unit and advice on their
adjustment:
------------------------------------------------------------------------
-
Swap unit log labels
```

```
NAME:

      white_shared_variables.h

 PURPOSE:
      Default values of variables used in White DSP swap unit code

 CALLING SEQUENCE:
      #include "white_shared_variables.h"

 CALLED FROM:
      setup.h

 INCLUDE FILES:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      09Feb93   DTL
      28Sep93   CH    move task_id to volatile; add data_delivery
      06Oct93   CH    add priority_q[512]
      06Dec93   SH    Added comment header
      08Feb94   SH    moved variable declarations from
                      shared_defines to shared_variables
      11Sep95   ADW   add standard comment header

 RCS $Id: white_shared_variables.h,v 1.17 1999/08/21 06:59:08 ho Exp $


*

Global memory addresses shared by White and Red DSPs
Status information
Calculated parameters
Detector window parameters for MIC full format, low-resolution
Data needed for housekeeping
White DSP swap unit parameters

 * The values of some of the symbolic constants listed above are stored
 * in variables with the same name, but in all lower case instead of
 * upper.  This is done so that after initialization these values
 * can be changed by "poking" the memory.  The assignment statements for
 * these variables are in su_initialize.c.


 * The following are variables assigned to symbolic constants defined
 * in shared_defines.h.

Integer parameters
Floating point parameters
             temporary q_alert buffer for extended output, 022196 ch
```

White:

```
NAME:

      shared_io_functions.c

 PURPOSE:

 FUNCTIONS IN THIS FILE:
      read_shared_data
      write_shared_data
      global_ram_io
      proc_ram_io
      local_ram_io
      blue_local_ram_io
      red_local_ram_io

 INCLUDE FILES:
      stdio.h
      sim_blue_local_ram.h
      sim_red_local_ram.h

 MODIFICATION HISTORY:
      19May95   ADW   add standard comment header
      21Sep95   ADW   add shell for Red local RAM stuff

 RCS $Id: shared_io_functions.c,v 1.23 1998/09/12 16:35:26 ho Exp $
```

```
NAME:

      read_shared_data       function

 PURPOSE:
      Read the White DSP local RAM file.

 CALLING SEQUENCE:
      integer_variable = read_shared_data();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns -1 if the local RAM file does not exist, 0
otherwise)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      local_ram_io      [shared_io_functions.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19May95   ADW   add standard comment header

   Open the local RAM file for reading.
   If the file does not exist, return -1.
   Scan the local RAM file to get the shared parameters.
   Scan the local RAM file to get the shared data.
```

```
NAME:

      write_shared_data       function

 PURPOSE:
      Write the White DSP local RAM file.

 CALLING SEQUENCE:
      integer_variable = write_shared_data();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns 0)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      local_ram_io      [shared_io_functions.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19May95   ADW   add standard comment header



   Open the local RAM file for writing.
   Write the shared parameters into the local RAM file.
   Write the shared data into the local RAM file.
```

```
NAME:

      global_ram_io          function

PURPOSE:
      This function takes care of the IO of the SUN-simulated RAM area
in DPU.

 CALLING SEQUENCE:
      global_ram_io(id);

 INPUTS:
      id   = integer to specify read (-1) or write (1) global RAM file

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      main         [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      25Sep93   CH    Start from scratch
      19May95   ADW   add standard comment header



   If id equals -1, then read from the global RAM file.
   Otherwise write out the global RAM file (normally id = 1 for that).
```

```
NAME:

      proc_ram_io          function

 PURPOSE:
      This function takes care of the IO
      of the SUN-simulated PROC RAM area in DPU.

      THIS program should be used in mutual exclusion with
global_ram_io.

      Unless you are absolutely certain you only need to READ from
      PROC and not write it back, always use global_ram_io.

 CALLING SEQUENCE:
      proc_ram_io(id);

 INPUTS:
      id   = integer to specify read (-1) or write (1) global RAM file

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      mic   [mic1.c]

      proc_ram_io is not currently used in white DSP code

 RESTRICTIONS:

 MODIFICATION HISTORY:
      25Sep93   CH    Start from scratch
      19May95   ADW   add standard comment header
```

```
NAME:

      local_ram_io          function

 PURPOSE:
      This function takes care of the IO of the SUN-simulated
      White DSP local RAM area in DPU.

 CALLING SEQUENCE:
      integer_variable = local_ram_io(id);

 INPUTS:
      id   = integer to specify read (-1) or write (1) local RAM file

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns the return value from read/write_shared_data)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      read_shared_data   [shared_io_functions.c (this file)]
      write_shared_data  [shared_io_functions.c (this file)]

 CALLED FROM:
      main               [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      25Sep93   CH    Start from scratch
      19May95   ADW   add standard comment header


   If id equals -1, then
      call read_shared_data to read the local RAM file.
   Otherwise (normally id = 1 here)
      call write_shared_data to write the local RAM file.
```

```
NAME:

      blue_local_ram_io       function

 PURPOSE:
      This function takes care of the IO of the SUN−simulated
      LOCAL RAM area in DPU.

 CALLING SEQUENCE:
      integer_variable = blue_local_ram_io(id);

 INPUTS:
      id   = integer to specify read (−1) or write (1) local RAM file

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns −1 on read if the blue local RAM file does not
exist,
                        on write if there is a file open error.)
             [There is no other return value in this routine yet...]

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      main             [whitedsp.c]
      sci_command      [sun_util.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      25Sep93   CH    Start from scratch
      19May95   ADW   add standard comment header



      Return the value −1 to the calling routine
      if there is a file open error.
      Read the blue local RAM file.
      Close the blue local RAM file.
      Return the value −1 to the calling routine
      if there is a file open error.
      Write out the blue local RAM file.
      Close the blue local RAM file.
```

```
NAME:

      red_local_ram_io

 PURPOSE:

 CALLING SEQUENCE:
      iv = red_local_ram_io(id);

 INPUTS:
      id

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      main   [reddsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      21Sep95   ADW   add standard comment header
```

```
NAME:

      shared_variables.c

 PURPOSE:
      Declaration of White DSP local RAM variables.

 CALLING SEQUENCE:
      #include "shared_variables.c"

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      setup.h

 RESTRICTIONS:

 MODIFICATION HISTORY:
      09Feb93   DTL
      28Sep93   CH   move task_id to volatile; add data_delivery
      06Oct93   CH   add priority_q[512]
      06Dec93   SH   Added comment header
      08Feb94   SH   moved variable declarations from
                     shared_defines to shared_variables

 RCS $Id: shared_variables.c,v 1.16 1998/03/21 02:03:19 xmmom Exp $


 * The values of some of the symbolic constants listed above are stored
 * in variables with the same name, but in all lower case instead of
 * upper.  This is done so that after initialization these values
 * can be changed by "poking" the memory.  The assignment statements for
 * these variables are in su_initialize.c.

The following are White DSP swap unit parameters.
The following are variables assigned to symbolic constants
that are defined in white_shared_variables.h.
These were originally global inside su_saa.c.
```

```
NAME:

      su_calbgd.c         (Swap unit)

 PURPOSE:
      This swap unit calculates the threshold level used in the stellar
image
  construction function.  It samples three blocks of pixels spaced at
equal
  intervals in the previous (ping or pong) frame without regard to the
window
  dimensions (optimized to make efficient use of the global bus).
  The threshold level is NTHLD times the average level in this sampling.
  Integer arithmetic is used.

      Used in tasks:  ACQUIRE_FIELD
                      CHOOSE_GUIDE_STARS
                      TRACK_GUIDE_STARS

      Prerequisites:  prev_frame - from ping/pong
                      task_id - what are we doing?
                      keep_a_log - yes or no?
                      log_indx - address of current place in log

      Provides:       ebgx
                      ebgy
                      thld - star recognition threshold
                      threshold_iter - initialized to 0

      Global RAM access:  (N_PIX_BLKS + 1) DPU cycles
                          (+ 1 more cycle if log kept)

      DSP RAM required:   ~300 words

 FUNCTIONS IN THIS FILE:
      calbgd

 INCLUDE FILES:
      setup.h

 MODIFICATION HISTORY:
      18May95   ADW   add standard comment header

 RCS $Id: su_calbgd.c,v 1.23 1998/05/28 04:41:22 xmmom Exp $
```

```
NAME:

      calbgd

 PURPOSE:
      This swap unit calculates the threshold level used in the stellar
image
  construction function.  It samples three blocks of pixels spaced at
equal
  intervals in the previous (ping or pong) frame without regard to the
window
  dimensions (optimized to make efficient use of the global bus).
  The threshold level is NTHLD times the average level in this sampling.
  Integer arithmetic is used.

 CALLING SEQUENCE:
      swap_to = calbgd();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none (returns 1 if sum_cts = 0, 0 otherwise)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      q_alert              [sun_util.c, cwhite.c]
      read_proc_int        [sun_util.c, global_access.c]
      read_proc_int_array  [sun_util.c, global_access.c]
      read_sw_int_array    [sun_util.c, global_access.c]
      write_proc_int       [sun_util.c, global_access.c]

      return_bus           [sun_util.c] *** currently commented ***

 CALLED FROM:
      sunload              [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      ancient   CH    original code
      04Sep92   DTL   major revision, optimized to DPU
      26Oct92   DTL   thld shared; ebgx, ebgy local
      22Jan93   DTL   full_frame defined here, not in shared_externals
      22Jan93   DTL   added file open/close calls for Sun
      04Apr93   DTL   change "swap_id" to shared variable "swap_to"
      05Jul93   JMK   added test for sun_cnts == 0
                      if true clear window data flag and return
      02May95   ADW   Added standard comment header
      09Jun95   ADW   change "printf(" to "fprintf(stderr," for
simulation
                      (timing of log messages was screwy with printf),
```

NAME:

su_caldrft.c (Swap unit)

PURPOSE:
This swap unit calculates the maximum likelihood values DX, DY, and sin(THETA) needed to make the current positions of reference stars agree (as well as possible) with their nominal positions. Of the available reference stars, a subset consisting of those identified and of good quality are used in the calculation. Weights initially assigned to the stars are adjusted downward in inverse proportion to their distances from expected locations given the current best solution, then the solution is iterated using the adjusted weights.

Notes:

The current coordinates are assumed to be related to the nominal coordinates by a rotation/translation transformation as if, relative to the detector reference frame at the time of observation, the detector needs to undergo a translation by DX pixels in x and DY pixels in y, then a rotation by angle THETA (assumed to be small) about a roll center at (Xo, Yo) in the shifted coordinates, for consistency with the nominal coordinates. Thus,

```
(x_obs - Xo) = (x_nominal - Xo) * cos(THETA) -
               (y_nominal - Yo) * sin(THETA) + DX
```

and

```
(y_obs - Yo) = (x_nominal - Xo) * sin(THETA) +
               (y_nominal - Yo) * cos(THETA) + DY.
```

Note that Xo and Yo are not fitted parameters; they are constants which give the location of the rotation axis in the detector field of view.

The maximum likelihood parameter values are the values that minimize the sum of the squared differences between the observed star positions and the positions estimated using these transformation equations.

In the ACQUIRE_FIELD task, the nominal coordinates are those of the (identified) absolute guide stars in the reference frame corresponding to the commanded satellite orientation, and the current coordinates are those of the corresponding stars as observed in a field acquisition reference frame (see su_makeags_id.c). The resulting parameters DX, DY, and sin(THETA) give the "absolute" transformation needed to center the detector windows on various targets (the coordinates of which are given in the reference frame corresponding to the commanded satellite orientation).

In the TRACK_GUIDE_STARS task, the nominal coordinates are those of a set of selected guide stars as observed in the initial reference frame (not necessarily the one obtained for field acquisition because a new filter may be required), and the current coordinates are those of

```
  the same stars as located in a subsequent tracking frame.  The
  resulting parameters DX, DY, and sin(THETA) give the "shift" required
  to compensate for satellite drift (the roll is not used for this
purpose)
  to accumulate an image for a time longer than that over which
significant
  drift occurs (the so-called "shift-and-add" function).

  DX and DY are determined (in this swap unit) on the basis of the
  assumption that the roll is small but possibly non-zero.
Specifically,
  it is assumed in the derivation of DX and DY that cos(THETA) ~ 1, in
  which case

      x_obs ~ x_nominal - (y_nominal - Yo) * sin(THETA) + DX

  and

      y_obs ~ y_nominal + (x_nominal - Xo) * sin(THETA) + DY.

  In the shift-and-add function (su_saa.c) the roll is neglected
  altogether; there it is simply assumed that

      x_obs ~ x_nominal + DX and y_obs ~ y_nominal + DY.


/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\

  NOTE: It would not be necessary to check the guide star quality in
this
      swap unit if, in su_dowindow, the guide stars to which windows
were
      not assigned were culled from the list of good guide stars (this
      was the purpose of CH's reshuffle() function, which is not
      implemented at present).  Making this change would improve the
      efficiency of su_caldrft; because it runs frequently, the overall
      code efficiency would improve somewhat.


\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/


      Used in tasks:      ACQUIRE_FIELD and TRACK_GUIDE_STARS

      Prerequisites:
               nominal_coords[] - nominal pos'ns of ref stars
               current_coords[] - observed pos'ns of ref stars
               ref_star_quality[] - how "good" are the stars?
               ref_star_weight[] - weights assigned to stars
               stars_found - number of guide stars identified
               roll_ctr_x - x coord of roll center
               roll_ctr_y - y coord of roll center
               task_id - what are we doing?
               keep_a_log - yes or no?
               log_indx - address of current place in log

      Provides:
               calc_deltax - DX in above transformation eq
               calc_deltay - DY in above transformation eq
               calc_sintheta - sin(THETA) in transform eq
               new estimate of current_coords if a star is lost

      Global RAM access:   write (# outliers) * SIZEOF_POINT words
                              (+ a few if log is kept)
```

```
               => ~1 DPU cycle

     DSP RAM required:   ~70 variables (plus code)

FUNCTIONS IN THIS FILE:
     caldrft
     calculate_parameters

INCLUDE FILES:
     setup.h

MODIFICATION HISTORY:
     18May95   ADW   add standard comment header
     28Oct98   CH    Major update on in-code comments

RCS $Id: su_caldrft.c,v 1.22 1998/09/12 15:57:02 ho Exp $
```

```
NAME:

      caldrft

PURPOSE:
      This swap unit calculates the maximum likelihood values DX, DY,
and
      sin(THETA) needed to make the current positions of reference stars
      agree (as well as possible) with their nominal positions.  Of the
      available reference stars, a subset consisting of those identified
      and of good quality are used in the calculation.  Weights
initially
      assigned to the stars are adjusted downward in inverse proportion
      to their distances from expected locations given the current best
      solution, then the solution is iterated using the adjusted
weights.

 CALLING SEQUENCE:
      swap_to = caldrft();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns 0)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      calculate_parameters   [su_caldrft.c (this file)]
      write_bw_int           [sun_util.c; global_access.c]
      write_bw_int_array     [sun_util.c; global_access.c]
      write_proc_int         [sun_util.c; global_access.c]
      write_proc_int_array   [sun_util.c; global_access.c]

 CALLED FROM:
      sunload                [whitedsp.c]
      execute                [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      00Jul90   CH    original code
      13Oct92   DTL   documented and converted to proto-flight code
      08Dec92   DTL   estimated coords of "lost" guide stars
                      written to current_coords and global RAM
      04Jan93   DTL   guide star quality checked when total
                      weight calculated (as elsewhere)
      28Jan93   DTL   shifted_gstar structure elements changed
                      from floats to ints
      04Feb93   DTL   added file open/close calls for Sun
      11Mar93   DTL   added capability to queue tracking data
      23Mar93   DTL   leave room for tracking data header
                      (3 words) in global memory
      04Apr93   DTL   change "swap_id" to shared variable "swap_to"
      10May95   ADW   add standard comment header
      09Jun95   ADW   change "printf(" to "fprintf(stderr," for
simulation
                      (timing of log messages was screwy with printf),


   Initialize weight and quality pointers and total_weight.
   If the keep_a_log flag is set, report stars_found.
   Loop over all stars found.
```

```
    Accumulate the sum of weights only for stars
    whose quality is less than quality_tolerance
    (i.e., good stars only).
    Move pointers.
 Solve for maximum likelihood values and iterate weights.
 Initialize previous_resid to 10 billion (a large number).
 Start a loop of max_caldrft_iterations passes.
    Initialize accumulators and pointers
    Loop over all stars found.
       Test the star's quality rating.
          If the star is of good quality
          determine its distance from its expected position.
          The method is different for different data representations.
          NOTE: The roll center coordinates, Xo and Yo, had been
          subtracted previously to obtain the nominal coordinates.
          Adjust the weights for the next iteration.
          Accumulate the total adjusted weight
          and weighted sum of squared residuals.
       Record guide star data in (moved) verbose_ptr.
       Move the pointers for the next star.
    Establish the inverse of the total weight
    Calculate the change in the mean squared residuals.
    Stop iterating if the change is sufficiently small.
       Break out if the change is small.
    Set previous_resid to mean_sq_resid for next iteration.
 If we're tracking and the star is lost (see su_locategs),
 assign a realistic (estimated) position to the star in the
 hope that it will be relocated in the next frame.
 Note: only guide stars of good quality are repositioned
    Initialize nominal and observed counters.
    Loop over the stars in the current batch.
          Save the estimated current_coords for the next frame.
       Move the coordinate pointers.
    Save the housekeeping data.
```

```
NAME:

      calculate_parameters

PURPOSE:
      This function solves for the maximum likelihood values of
parameters
      DX (called calc_deltax), DY (called calc_deltay), and sin(THETA)
      (called calc_sintheta) assuming that:

      (a) the nominal_coordinates (reference positions) are
          known exactly (i.e., essentially error-free); and
      (b) cos(THETA) = 1 (i.e., the roll angle, THETA, is small).

 CALLING SEQUENCE:
      calculate_parameters();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_proc_int   [sun_util.c; global_access.c]

 CALLED FROM:
      caldrft          [su_caldrft.c (this file)]

 RESTRICTIONS:

  Note: the maximum likelihood value of THETA satisfies

         c_s * sin(THETA) + c_c * cos(THETA) = 0

  This implies that ctn(THETA) = -c_s/c_c, and hence

         sin(THETA) = + or - c_c / sqrt(c_s ^ 2 + c_c ^ 2)

      where:
         + should be chosen if c_s < 0 and
         - should be chosen if c_s > 0.

  In the limit that the observed coordinates are equal to their
  nominal values,

      c_s = sum{w_i} * sum{w_i * [(x_i - <x>) ^ 2 + (y_i - <y>) ^ 2]},

  which is positive-definite.  Hence, it is further assumed that
  c_s > 0, and therefore that

      sin(THETA) = -c_c / sqrt(c_s ^ 2 + c_c ^ 2)

      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

      It would not be very costly computationally to make the
      sign of sin(THETA) conditional on the sign of c_s.  I
      recommend that we take this precaution (DTL 9 Oct 92).

      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

 MODIFICATION HISTORY:
      10May95   ADW   add standard comment header, structured English
```

```
comments
      09Jun95   ADW   change "printf(" to "fprintf(stderr," for
simulation
                      (timing of log messages was screwy with printf)


   Initialize the pointers and variables used for summation.
   Loop over the stars in the current batch.
      If running in simulation (SUN) and DEBUGging is enabled,
      print a message about the star's quality.
      Use only stars of acceptable quality.
         Accumulate the sin and cos coefficients.
         Increment the number of good stars found.
      Move pointers for the next star.
   Floating point data representation
   Add products of summed terms to sin and cos coefficients.
   Calculate the quadrature sum of c_s and c_c.
   qsum_coeffs = sqrt(c_s * c_s + c_c * c_c);
   Use approximation good to 2nd order in (c_c/c_s) to save time
   Calculate parameter values.
   Integer representation of floats
   Add products of summed terms to sin and cos coefficients.
   Calculate the quadrature sum of c_s and c_c.
   qsum_coeffs = sqrt(c_s * c_s + c_c * c_c);
   Use approximation good to 2nd order in (c_c/c_s) to save time
   Calculate parameter values.
```

```
NAME:

      su_compress.c      (Swap unit)

 PURPOSE:
      This swap unit does the VBTWL compression

 FUNCTIONS IN THIS FILE:
      main
      compress
      read_sw_cmp_block
      read_bw_cmp_block
      write_cmp_block
      vbtwl
      optvbtwl
      resetlx
      realignlx
      spectrum0
      spectrum1
      spectrum2
      predict
      msbb
      tiering
      msb
      com_para
      encode
      reset_cstr
      do_header
      encode0
      encode1
      encode2
      do_plusminus
      enfilter
      insert0
      insert

 INCLUDE FILES:
      setup.h
      vbtwl.h
      vbtwl.def

 MODIFICATION HISTORY:
      01may96   CH    Original Codes

 RCS $Id: su_compress.c,v 1.11 1998/07/09 17:30:28 xmmom Exp $


Include setup.h for Sun and DPU.
If running in simulation, declare the external integer *global_ram.
```

```
NAME:

      compress function

 PURPOSE:
      This function drives the vbtwl compression

 CALLING SEQUENCE:
      swap_to = compress();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      return status    compress_status

 INCLUDE FILES:

 SUBROUTINES CALLED:
      read_proc_int_array
      q_alert
      read_sw_cmp_block
      read_bw_cmp_block
      vbtwl
      write_cmp_block
      hdr_manager
      data_queue

 CALLED FROM:
      sunload   [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      01May96   CH    start from scratch



   Declare the DPU integer variables.
   Set the compression status word to idle
   if there are no entries on the compression queue.

      Notice stripping off the highest bit CH *
```

```
NAME:

      read_cmp_block

 PURPOSE:

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

NAME:

write_cmp_block

PURPOSE:

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:

```
     12Jun96   ADW   add standard comment header
```

```
NAME:

      vbtwl

 PURPOSE:

 CALLING SEQUENCE:

 INPUTS:
      rawint *block     array of raw data to be compressed.
      int    blockid    id of block designated by the DPU.
      int    blocksize  number of elements in the block
      int    cmpmode    binary-coded instruction from the DPU to ICU on
                        which algorithm to use to compress the data

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      cmpshort *cstr    pointer to the compressed data string. The size
                        of the template string MUST be defined and
                        allocated by the calling function.
      int      *imm     the selected predictor algorithm

      int vbtwl() returns the number of bits in the compressed string.

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header

This file contains the Variable Block Tiered Word Length compression
engine.

  Requires:

  vbtwl.h:      definition for structural information
  vbtwl.def:    definition of global variables

  Cheng Ho
  Original scheme 1990, developed for the XMMOM project.
  Version 0.x, 1991, 1992, 1995
  Ver. 1.0  April, 19, 1996

  This function is the overall image compression routine to be performed
by the VBTWL.

  This file is self-contained. But it can be broken into two major
parts, in addition to the vbtwlcmprss calling routine: optvbtwl.c and
compress.c.



**** Need to explicitly initialize these arrays for DSP implementation
!! **
                  Find the optimal compression parameters
                  Do the compression


  Section 1: Determination of the optimal vbtwl prediction and
```

```
            encoding scheme
The following modules determine the optimal compression parameters.
```

```
NAME:

      optvbtwl
 PURPOSE:
      This function determines the ``optimal'' tiering structure for a
      block of image of size (isize*jsize) whose lower left corner is
      located at (i*isize, j*jsize) of the image.

      Three decorrelation schemes using three different predictors are
      included in the consideration. The predictors are 0. the minimum,
      1. the mean and 2. the previous twixel count. The word length are
      divided into three tiers. See Doc3 for more disciusison. At the
      end, this function yields:

      im:         decorrelation/predictor mode (0, 1, 2)
      it1:        number of bits in tier 1.
      it2:        number of bits in tier 2.
      it3:        number of bits in tier 3.
      nlx3:       number of twixels in tier 3.
      nlx2:       number of twixels in tier 2.
The numbers above needs to be included in the header.

      Other relevant quantities:

      nlx1:       number of twixels in tier 1.
      n_bits:         total number of bits require in this block.
      nlx1+nlx2+nlx3=blocksize=isize*jsize.
      n_bits=blocksize*it1+nlx2*it2+nlx3*it3+nheader.
      nheader is the number of bits in the header. 64 bits appears
      to be a good number with enough redundancy for error checking.

 CALLING SEQUENCE:

 INPUTS:
      n_tier
      *block
      blocksize
      cmpmode

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

```
NAME:

      resetlx

 PURPOSE:
      This function resets the array lx[] to zero. The array lx[] will
      hold the ``spectral'' information of the block.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

```
NAME:

      realignlx

 PURPOSE:
      This function turns the spectrum lx[] from a density distribution
      to an accumulated function.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

```
NAME:

       spectrum0

 PURPOSE:
      Analyze the current block using the minimum within the block as
the
      predictor. Yields lx[].

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

NAME:

spectrum1

PURPOSE:
Analyze the current block using the mean within the block as the predictor. Yields lx[].

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
12Jun96   ADW   add standard comment header

NAME:

spectrum2

PURPOSE:
Analyze the current block using the count in the previous twixel as
as the predictor. Yields lx[].

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
12Jun96 ADW add standard comment header

```
NAME:
      predict

 PURPOSE:
      This function initializes the predictor for the decorrelation
      scheme specified by im.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      18Oct95   ADW   add standard comment header



      Minimum as predictor
      Mean as predictor
      Previous (cyclic) values as predictor
```

NAME:

msbb

PURPOSE:
Number of bits required to express a SIGNED integer k. Leave one addition
slot for the two flags needed. Here, the two flags are −0(it1) and
−(2^(it1−1)−1). Calls the msb().

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
12Jun96 ADW add standard comment header

```
NAME:

      tiering

 PURPOSE:
      Tiering analysis based on the accumulated spectral distribution
lx[].
      Yields it1, it2, it3, nlx3, and n_bits. Returns n_bits.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

NAME:

msb

PURPOSE:
The most significant bit for an UNSIGNED integer. The minimum number
of bits is one (1). This is done by properly initializing i and k.

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
12Jun96 ADW add standard comment header

Section 2. Encoding

The following modules do the actual compression.

```
NAME:

       com_para

 PURPOSE:
      This function calculates the parametric quantities to be used in
      the vbtwl compression. They are the levels separating the tiers,
the
      filters to get the tiered number and the flags which indicates
      the higher tiers.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

```
NAME:

       encode

 PURPOSE:
      This fucntion carries out the compression and produces a string of
      bits. which is (n_bits/8) * 8 int. This will then be used to
either
      be passed to the telemetry requester or a FIFO stack for
telemetry.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

```
NAME:

      reset_cstr

 PURPOSE:
      Reset the template telemetry string.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

```
NAME:

       do_header

 PURPOSE:
      This function establishes the header. Current thinking is that
      the header will have 64 bits (8 bytes) containing all sorts of
      information.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

NAME:

encode0

PURPOSE:
Encoder for the VBTWL scheme with decorrelation scheme 0
(minimum as predictor).

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
12Jun96 ADW add standard comment header

NAME:

encode1

PURPOSE:
Encoder for the VBTWL scheme with decorrelation scheme 0
(average as predictor).

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
12Jun96 ADW add standard comment header

```
NAME:

      encode2

 PURPOSE:
      Encoder for the VBTWL scheme with decorrelation scheme 2
      (previous value as predictor).

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

```
NAME:
      do_plusminus

 PURPOSE:
      Processing SIGNED remnant and calling enfilter() to encode the
remnant
      number n into bit stream s.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

NAME:

enfilter

PURPOSE:
This function checks which tier the int number n is in.
The tiering is given by the levels l1 and l2. The number n is
then passed throguh the appropriate filter (Do we need that?)
and returned. Returns number of bits in the tirer.

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
12Jun96 ADW add standard comment header

```
NAME:

      insert0

 PURPOSE:
      Flag inserter into the compressed data steram. Once the flag is
      inserted, calls insert to insert the bit stream n of length k
      at the current bitstream location.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

```
NAME:

      insert

 PURPOSE:
      This function inserts a numerial string n of length nbits into the
      template telemetry string. This function will append a bit string
n of
      length nbits to the template bit stream. The beginning pointer of
the
      template stream is cstring. The insertion starts at the cptr
pointer
      at the rema−th bit.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      21Jan92   CH
      12Jun96   ADW   add standard comment header
```

```
NAME:

      su_deliverdata.c      (Swap unit)

 PURPOSE:
      This swap unit manages the timing of DPU operations, selects the
      task requested by the ICU, and queues data for delivery in the
      telemetry stream.

 FUNCTIONS IN THIS FILE:
      deliverdata
      deliver_faq_data
      deliver_bluefast_data
      deliver_image_data
      deliver_wdw_data
      deliver_trk_data
      deliver_bfast
      prepare_priority_queue
      deliver_ref_data
      deliver_trh_data
      deliver_eng_data

 INCLUDE FILES:
      setup.h

 MODIFICATION HISTORY:
      18May95   ADW   add standard comment header
      12Mar96   ADW   add engineering mode

 RCS $Id: su_deliverdata.c,v 1.41 1999/08/21 05:36:33 ho Exp $
```

```
NAME:

      deliverdata          function

 PURPOSE:
      This swap unit manages the timing of DPU operations, selects the
      task requested by the ICU, and queues data for delivery in the
      telemetry stream.

 CALLING SEQUENCE:
      swap_to = deliverdata();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none    (returns 0 if ok, 1 if bad data_delivery specified)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      deliver_bfast           [su_deliverdata.c (this file)]
      deliver_bluefast_data   [su_deliverdata.c (this file)]
      deliver_faq_data        [su_deliverdata.c (this file)]
      deliver_image_data      [su_deliverdata.c (this file)]
      deliver_ref_data        [su_deliverdata.c (this file)]
      deliver_trh_data        [su_deliverdata.c (this file)]
      deliver_trk_data        [su_deliverdata.c (this file)]
      deliver_wdw_data        [su_deliverdata.c (this file)]
      deliver_eng_data        [su_deliverdata.c (this file)]

 CALLED FROM:
      sunload                 [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      30Sep93   CH    start from scratch
      08May95   ADW   add standard comment header
      12Mar96   ADW   add engineering mode support

   Initialize array access parameters xsize and yskip.
   Call appropriate function based on data type to be delivered.
   Return with value 0 for every valid data type, else 1.
If we get here, there is a bogus data type specified.
In this case, print a message to that effect if we're
running in simulation (SUN).
      Return to calling routine with value 1.

      Auxiliary functions
```

```
NAME:

      deliver_ee_dump_data

 PURPOSE:
      This function queues ee_dump data for priority to
      in the telemetry stream.

 CALLING SEQUENCE:
      deliver_ee_dump_data();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      prepare_priority_queue  [su_deliverdata.c (this file)]
      priority_queue          [sun_util.c, cwhite.c]
      read_proc_int           [sun_util.c, global_access.c]
      read_proc_int_array     [sun_util.c, global_access.c]

 CALLED FROM:
      deliverdata             [su_deliverdata.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      10May95   ADW   add standard comment header
```

```
NAME:

      deliver_faq_data

 PURPOSE:
      This function queues field acquisition housekeeping data for
delivery
      in the telemetry stream.

 CALLING SEQUENCE:
      deliver_faq_data();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:


 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      prepare_priority_queue  [su_deliverdata.c (this file)]
      priority_queue          [sun_util.c, cwhite.c]
      read_proc_int           [sun_util.c, global_access.c]
      read_proc_int_array     [sun_util.c, global_access.c]

 CALLED FROM:
      deliverdata             [su_deliverdata.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      10May95   ADW   add standard comment header


   Prepare the priority queue.
   Enter the data into onboard priority buffer.
   Activate the priority queue (at which point we're done here).
```

```
NAME:

      deliver_bluefast_data

 PURPOSE:
      This function queues science data for delivery in the telemetry
stream.

 CALLING SEQUENCE:
      deliver_bluefast_data();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      data_queue     [sun_util.c, cwhite.c]
      hdr_manager    [whitedsp.c]
      read_proc_int  [sun_util.c, global_access.c]
      q_alert        [sun_util.c, cwhite.c]

 CALLED FROM:
      deliverdata    [su_deliverdata.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      10May95   ADW   add standard comment header
```

```
NAME:
      deliver_image_data

 PURPOSE:
      This function queues image data for delivery in the telemetry
stream.

 CALLING SEQUENCE:
      deliver_image_data();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      data_queue           [sun_util.c, cwhite.c]
      hdr_manager          [whitedsp.c]
      read_proc_int        [sun_util.c, global_access.c]
      read_proc_int_array  [sun_util.c, global_access.c]

 CALLED FROM:
      deliverdata          [su_deliverdata.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      10May95   ADW   add standard comment header


   Read the science window configuration data from global RAM.
   Initialize scw_id counter to 0.
   For values of scw from science_windows to
science_windows+n_valid_scw-1
   loop (through the current crop of science windows).
         Set up the header data.
            Send the data out on the data queue.
      Increment scw_id.
```

```
NAME:

      deliver_wdw_data

 PURPOSE:
      This program delivers the window configuration data
      via the priority queue.

 CALLING SEQUENCE:
      deliver_wdw_data();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      prepare_priority_queue  [su_deliverdata.c (this file)]
      priority_queue          [sun_util.c, cwhite.c]
      read_proc_int           [sun_util.c, global_access.c]
      read_proc_int_array     [sun_util.c, global_access.c]

 CALLED FROM:
      deliverdata             [su_deliverdata.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      10May95   ADW   add standard comment header



   Prepare the priority queue.
   Enter the data into onboard priority buffer.
   Activate the priority queue (at which point we're done here).
```

NAME:

deliver_trk_data

```
 PURPOSE:
      This program delivers the frame-by-frame tracking information
      via the priority queue.

 CALLING SEQUENCE:
      deliver_trk_data();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      prepare_priority_queue  [su_deliverdata.c (this file)]
      priority_queue          [sun_util.c, cwhite.c]

 CALLED FROM:
      deliverdata             [su_deliverdata.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      10May95   ADW   add standard comment header



   Deliver tracking history data if requested
      Prepare data for alert
      Send out alert
```

```
NAME:

      deliver_bfast

 PURPOSE:
      This program delivers the bfast information via the priority
queue.

 CALLING SEQUENCE:
      deliver_bfast();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      prepare_priority_queue  [su_deliverdata.c (this file)]
      priority_queue          [sun_util.c, cwhite.c]
      read_proc_int           [sun_util.c, global_access.c]
      read_proc_int_array     [sun_util.c, global_access.c]

 CALLED FROM:
      deliverdata             [su_deliverdata.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      10May95   ADW   add standard comment header



      Prepare the priority queue.
      Enter the data into onboard priority buffer.
      Activate the priority queue.
         Prepare the priority queue.
         Enter the data into onboard priority buffer.
         Activate the priority queue.
```

```
NAME:

      prepare_priority_queue

 PURPOSE:
      This program establishes the availability of the priority queue.

 CALLING SEQUENCE:
      prepare_priority_queue();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      q_alert            [sun_util.c; cwhite.c]

 CALLED FROM:
      deliver_bfast      [su_deliverdata.c (this file)]
      deliver_faq_data   [su_deliverdata.c (this file)]
      deliver_trk_data   [su_deliverdata.c (this file)]
      deliver_wdw_data   [su_deliverdata.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      10May95   ADW   add standard comment header

   If keep_a_log is enabled, make a log entry.
   While the priority queue is busy, do nothing.
   (i.e., wait until the queue has been cleared)
   This lets previous queue activity complete.
   Block out the priority queue.
   DPUOS should clear the queue when it is done.
   If keep_a_log is enabled, make a log entry.
```

```
NAME:

      deliver_ref_data

 PURPOSE:
      This function queues reference frame for delivery
      in the telemetry stream.

 CALLING SEQUENCE:
      deliver_ref_data();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      data_queue           [sun_util.c, cwhite.c]
      hdr_manager          [whitedsp.c]
      read_proc_int        [sun_util.c, global_access.c]
      read_proc_int_array  [sun_util.c, global_access.c]
      write_bw_int         [sun_util.c, global_access.c]
      write_bw_int_array   [sun_util.c, global_access.c]

 CALLED FROM:
      deliverdata          [su_deliverdata.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      10May95   ADW   add standard comment header

Set base and initialize offset for reference frame data
   Construct a header for reference frame data.
   Gather data on good guide stars
      Gather data on rejected "stars"
   If keep_a_log is enabled, make a log entry.
   Queue the reference frame data for delivery
   and zero-out accum memory.
```

NAME:

submit_cq

PURPOSE:

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
```
     15Jul96   ADW   add standard comment header
```

```
NAME:

      deliver_trh_data

 PURPOSE:
      This function queues tracking history data for
      delivery in the telemetry stream.

 CALLING SEQUENCE:
      deliver_trh_data();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      data_queue      [sun_util.c; cwhite.c]
      hdr_manager     [whitedsp.c]
      q_alert         [sun_util.c; cwhite.c]

 CALLED FROM:
      deliverdata     [su_deliverdata.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      10May95   ADW   add standard comment header



   Deliver tracking history data if requested
   This is now the default - CH 08May96
      If keep_a_log is enabled, make a log entry.
         Construct header for tracking data
         Write header to global memory
         Queue tracking data for delivery (and zero-out accum memory)
```

```
NAME:

      deliver_eng_data

 PURPOSE:
      This function queues image data for delivery in the telemetry
stream.

 CALLING SEQUENCE:
      deliver_eng_data();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      data_queue           [sun_util.c, cwhite.c]
      hdr_manager          [whitedsp.c]
      read_proc_int        [sun_util.c, global_access.c]

 CALLED FROM:
      deliverdata          [su_deliverdata.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      15Dec95   ADW   new



   The data_length assignment could be generalized to something like
   data_length = dtw.xsize * dtw.ysize;
         Send the data out on the data queue.
```

```
NAME:

      su_dowindow.c          (Swap unit)

 PURPOSE:
      This swap unit establishes (a) memory, science, and detector
window
      parameters corresponding to user-specified science and memory
windows;
      and (b) up to MAX_TRACKING_WINDOWS science windows and, if
necessary,
      additional memory and detector windows corresponding to a set of
      guide stars for tracking.

  Notes:

  First the user-specified windows are shifted to take the estimated
  error in absolute pointing into account (the window coordinates
  specified by the observer are relative to the commanded spacecraft
  pointing direction).

  The validity of user-specified windows is checked (e.g., windows must
  be non-overlapping and must fit in the memory available), science
  windows are matched to the appropriate memory windows, and detector
  windows are established to cover the memory windows.  (Note: some of
  these functions should be performed instead using a ground-based
  program which could be made available to observers.)

  The quality of an otherwise good guide star is set to
CROSS_TKW_QUALITY
  (bad) if the square memory window, of size TRACKING_MMW_DIM, required
to
  enclose it crosses a valid user-specified memory window.  Any other
  potentially good guide stars to which science, memory and detector
  windows are not assigned are given a quality value 2 *
CROSS_TKW_QUALITY,
  which is also interpreted as "bad."

  Windows are subject to the following constraints:

   1. Frame memory (ping or pong) may not exceed FRAME_MEMORY
      short words;

   2. Total (ping plus pong) data accumulation memory may not exceed
      ACCUM_MEMORY long words; and

   3. The maximum numbers of science, memory and detector windows are
      given, respectively, by MAX_SCIENCE_WINDOWS, MAX_MEMORY_WINDOWS,
      and MAX_DETECTOR_WINDOWS.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~
  CAVEAT: The detector window parameters calculated herein are
incompatible
         with the UCL/MIC Y addressing scheme as described on 08/08/92.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~

      Used in tasks:      CHOOSE_GUIDE_STARS

      Prerequisites:
               abs_deltax - field acquisition x error
               abs_deltay - field acquisition y error
               keep_a_log - yes or no?
```

```
                log_indx - address of current place in log

      Provides:
                memory window parameters
                science window parameters
                detector window parameters
                windows of each type for tracking guide stars
                number of (valid) windows of each type

      Global RAM access:
                max 258 words to read user-specified window data
                max 258 words to write mmw and scw data
                max approx 65 (TBD) words to write dtw data
                (plus a few words to write if log kept)

                => total about 2 or 3 DPU cycles

      DSP RAM required:
                approx 450 words data plus code
                (Note: if space is tight, consider eliminating
                 mmw_corners[] as per in-code advice)

 FUNCTIONS IN THIS FILE:
      dowindow
      shift_windows
      bound
      check_mmw_overlap
      in_box
      remember_mmw_corners
      make_detector_windows
      create_dtw
      make_bfast
      adjust_dtw_kompy
      shift_dtw
      reshuffle

 INCLUDE FILES:
      setup.h

 MODIFICATION HISTORY:
      18May95   ADW   add standard comment header
      09Jun95   ADW   change "printf(" to "fprintf(stderr," for
simulation
                      (timing of log messages was screwy with printf)
      28Oct98   CH    Major update on in-code comments

 RCS $Id: su_dowindow.c,v 1.33 1999/08/21 05:40:15 ho Exp $
```

```
NAME:

      dowindow          function

PURPOSE:
      This swap unit establishes (a) memory, science, and detector
window
      parameters corresponding to user-specified science and memory
windows;
      and (b) up to MAX_TRACKING_WINDOWS science windows and, if
necessary,
      additional memory and detector windows corresponding to a set of
      guide stars for tracking.

 CALLING SEQUENCE:
      swap_to = dowindow();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns 0)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      adjust_dtw_kompy        [su_dowindow.c (this file)]
      check_mmw_overlap       [su_dowindow.c (this file)]
      make_bfast              [su_dowindow.c (this file)]
      make_detector_windows   [su_dowindow.c (this file)]
      remember_mmw_corners    [su_dowindow.c (this file)]
      shift_windows           [su_dowindow.c (this file)]
      q_alert                 [sun_util.c, cwhite.c]
      read_proc_int           [sun_util.c, global_access.c]
      read_proc_int_array     [sun_util.c, global_access.c]
      write_proc_int          [sun_util.c, global_access.c]
      write_proc_int_array    [sun_util.c, global_access.c]

      bound                   [su_dowindow.c (this file)] *** commented
***
      reshuffle               [su_dowindow.c (this file)] *** commented
***

 CALLED FROM:
      sunload                 [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      08Aug92   CH    original code
      27Nov92   DTL   flight code; orig. code adapted to DPU
      10Dec92   DTL   save guide star id as parameter of tracking window
      14Dec92   DTL   tracking windows set up around current_coords,
                      not nominal_coords
      28Jan92   DTL   *guide_star points to a FLT_Point and
                      coords cast to int
      02Feb93   DTL   added file open/close calls for Sun
      11Mar93   DTL   blocked out section of bw memory for
                      housekeeping data queue
      23Mar93   DTL   make room for data queue header in
                      global memory (3 words)
      04Apr93   DTL   change "swap_id" to shared variable "swap_to"
      19May93   CH    Force the detector window initialization.
                      Prepare garbage collection from blue detector.
```

```
                      Force the detector to always load 16 dtws.
      12May95   ADW   add standard comment header
      23Jun95   ADW   Commented the scw shift to match specified mmw
BLC.
      14Sep95   ADW   Changed goto constructions in memory window
checking to
                      else{ stuff } to allow loop to finish.  Got base
right.
      15Sep95   ADW   Same as yesterday, but for science windows.
      27Sep95   ADW   Fix goto condition for tracking windows;
                      (re)set *good_qual_ptr to "bad" for all failed tkw
      28Oct98   CH    Major update on in-code comments



   Structure variables
   Functions
   Establish the parameters according to the bpe_binning factor
   Fetch memory window configuration data from global memory.
   Note: the memory window bases have not yet been assigned,
   Reset all detector windows to the garbage collector configuration.
   Fetch science window configuration data from global memory.
   Shift user-defined windows to compensate for absolute pointing error.
   Validate user-defined memory windows.
   Initialize MMW and DTW counters and base address in frame memory.
   Loop through user-specified memory windows.
      Determine whether the current memory window
      intersects a previously validated memory window.
      If there is no overlap
         record the base location of window.
         Calculate the size of the current memory window.
         If this window will not fit in available memory,
            and reset the base address.
         If the current memory window will fit in available memory,
            attempt to establish detector window(s) for it.
            If detector window creation fails
               and reset the base address.
            If detector window creation succeeds, then
               accept the current memory window,
               record the memory window boundaries,
               and increment the number of allocated windows.
      Report diagnostics if current memory window is invalid.
      If the current MMW intersects one already established
      If the current MMW is embedded in another already established,
   Determine which memory window contains each science window.
   At most one of the memory windows will satisfy this
   condition because memory windows are non-overlapping.
   Initialize the science window counter
   and base address in accumulation memory.
   Loop through user-specified science windows.
      We insist that a science window must be fully embedded
      in a valid memory window.
      Determine whether the current SCW is fully, partially,
      or not at all embedded in a valid memory window.
         Make mode-dependent assignments for
         image mode,
         fast mode,
            For now, set to Blue Image Mode when spec is bad.
            Continue in the loop...  ADW  15Sep95
         Determine whether or not there is enough room
         in memory to store data from this window.
         If there is not enough room,
            and reset the accum_base for the next user scw.
         If there is enough space for the current science window, then
            set the base location of the science window,
```

```
            accept the current science window,
            and increment the number of valid allocated science windows.
      Report diagnostics if unable to assign
      science window to a memory window.
      If the SCW partially overlaps a valid memory window
      If the current SCW does not overlap a valid memory window
            bus_cycle_temp = bus_cycle_count;
            (void) write_proc_int(log_indx, &bus_cycle_temp);
            log_indx += SIZEOF_BIG_WORD;
            (void) write_proc_int(log_indx, &su_dowindow_log9);
            log_indx += SIZEOF_BIG_WORD;
            if (log_indx > end_of_log) log_indx = log_base;
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
What should we do here if no user science windows were validated?
Should we simply send an alert and wait for some intervention, or
should we attempt to set up a default, e.g., center in valid mmw?
Without user scw, there's no point in doing anything more...  ADW
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
If the tracking is turned off, bypass the allocation of the tracking
windows
   Allocate windows to guide stars
   (up to n_guide_stars) for tracking.
   Initialize array pointers.
   Loop over the list of "good" guide stars.
      We can not allocate more tracking windows if the maximum number
      of science or tracking windows has already been allocated.
      We can allocate tracking windows even if the maximum number of
      memory windows has been reached because guide stars may be found
      in previously validated memory windows.
      Define the minimum size memory window specification for
      a tracking science window for the current guide star.
      This must follow the same BLC and size rules as for mmw.
      NOTE: There will be a problem if the guide_star x and y
         "Making tracking window %d with bottom left corner (%d, %d)\n",
          n_tracking_windows, tkw_x0, tkw_y0);
      Determine which previously validated memory window, if any,
      contains the current tracking window.  No more than one mmw
      will satisfy this because memory windows are non-overlapping.
      If the tracking window crosses a valid memory window
      set the window quality to cross_tkw_quality (bad).
            "Tracking window %d crosses mmw %d => cannot use.\n",
      If the tracking window specification does not intersect a valid
      memory window, attempt to create a corresponding memory window.
         If we have not already allocated the
         maximum number of memory windows,
            set the memory window location and size,
            and record the base location of the window.
            Then check to see whether or not there is
            enough memory for the new memory window.
            If there is not enough memory,
               reset the base address,
               set the star quality to a "bad" value,
               and skip to the next guide star.
            If there is enough memory, attempt to create
            a detector window for this memory window.
            If a detector window could not be made,
               reset the base address
               set the star quality to a "bad" value,
               and skip to the next guide star.
            If a detector window was created, accept it:
            Copy it into the array of valid memory windows,
            record the memory window boundaries,
            increment the number of allocated memory windows,
            and increment the memory window pointer.
            Reset in_window to indicate that the tracking window
            is now located in a (valid) memory window.
```

```
          If the maximum number of memory windows have been allocated
          we can not use the current guide star.  In that case
       If the guide star is found within a valid memory window
       create the corresponding tracking science window.
       Note: This is an "if" block, not and "else if" block
       because a new memory window may have been created for it.
          Set the memory window ID and index for the science window.
          Center the science window on the guide star
          (as opposed to centering in the memory window).
          Increment the number of validated tracking windows.
          Accept the current science window.
          Increment number of allocated science windows and scw pointer.
       This is the goto point for cases of otherwise good guide stars
       that can't be used because of various window problems (see the
       code and comments above).  This should come before the next
       two lines because we want to increment things every time
       through the loop over guide stars.   20Sep95  ADW
       Increment the tracking window quality pointer and guide star ID.
    Set quality to 2 * cross_tkw_quality (bad) for all
    guide stars to which no tracking window was assigned.
    Save memory window parameters in global memory.
    Write the ping/pong frame size into the PROC area for Bluedsp to
fetch.
    Save science window parameters in global memory.
    Save detector window parameters in global memory.
    Write out bfast into global memory.
    Allocate the reference frame and tracking array space.   CH
```

```
NAME:

      shift_windows         function

PURPOSE:
      This function shifts the user-defined science and memory windows
to
      compensate for absolute pointing error.

 CALLING SEQUENCE:
      shift_windows();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      bound         [su_dowindow.c (this file)] *** all calls commented
***
      round         [macro.h]

 CALLED FROM:
      dowindow      [su_dowindow.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12May95   ADW   add standard comment header
      13Sep95   ADW   Preserve shift amounts; resize windows shifted off
det.
      14Sep95   ADW   Memory windows must be at least
bpe_lut_scale*2xbpe_lut_scale*2 detector pixels.
      28Oct98   CH    Major update on in-code comments


   Round x and y shifts to nearest pixel.
   Note that abs_deltax and abs_deltay could be read from global RAM.
   Shift the memory windows.    ADW VERSION
   Note that we need to make sure that mmw borders
   on MIC boundaries of multiples of bpe_lut_scale !!
      Calculate the desired mmw shift amounts,
      taking into account possible zero-crossing
      interaction with integer arithmetic below.
      Shift memory window BLC coordinates to multiple of bpe_lut_scale
boundaries.
      If the shifted mmw BLC is off the detector BLC, adjust the mmw.
      If a resultant mmw dimension is less than bpe_lut_scale*2 pixels,
discard
      the mmw by setting the dimension to 0 and start point to
max_pixel.
      If the shifted mmw TRC is off the detector TRC, adjust the mmw.
      If a resultant mmw dimension is less than bpe_lut_scale*2 pixels,
discard
      the mmw by setting the dimension to 0 and start point to
max_pixel.
   Shift the science windows.  ADW VERSION
   Everything is similar to the memory window shifting except
   that science windows do not have to follow the bpe_lut_pixel pixel
rules.
```

```
NAME:

      bound          function

 PURPOSE:
       This function restricts the value x to be between xl and xu

 CALLING SEQUENCE:
      integer_variable = bound (x, xl, xu);

 INPUTS:
      x         value to restrict
      xl         lower limit for x on output
      xu         upper limit for x on output

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns restricted x such that xl<=x<=xu)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      dowindow           [su_dowindow.c (this file)]  *** calls
commented ***
      shift_windows      [su_dowindow.c (this file)]  *** calls
commented ***

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12May95   ADW   add standard comment header, structured English
comments
      28Oct98   CH    Major update on in-code comments
```

NAME:

check_mmw_overlap function

```
 PURPOSE:
      This function determines the relationship of a window of
dimensions
      xsize by ysize with bottom-left corner at (xmin, ymin), the "test
      window," to the set of validated memory windows.

      Three outcomes are possible:

      a) the test window does not intersect any of the established
         memory windows, in which case the value returned is 0;

      b) the test window is embedded in an established memory window,
         in which case the value returned is the memory window
         index plus 1 (1 added to make all indices > 0); or

      c) the test window intersects but is not embedded in (at least)
         one of the established memory windows, in which case the
         returned value is -(memory index + 1) (to make the
         result negative) (Note: if more than one window is
         intersected, the value returned refers to the first such
         window in the list of valid memory windows).

      If no memory windows have yet been established, the value returned
      is 0, consistent with the usage of 0 in case (a).

 CALLING SEQUENCE:
      integer_variable = check_mmw_overlap(xmin, ymin, xsize, ysize);

 INPUTS:
      xmin         blc x coord of window
      ymin         blc y coord of window
      xsize        x size of window (pixels)
      ysize        y size of window (pixels)

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns +(window_index+1): test window within memory
window
                      -(window_index+1): test window overlaps memory
window
                      0: for no valid memory windows or no overlap)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      in_box         [su_dowindow.c (this file)]

 CALLED FROM:
      dowindow       [su_dowindow.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments

   Return if there are no valid memory windows yet (no overlap...).
   Calculate the test window TRC coordinates.
   Test whether test window corners are within a valid memory window.
   Loop through the valid memory windows.
```

```
    There is overlap if at least one corner of the test window
    is within the current valid memory window.
       If so, determine whether or not all of the
       corners are within the current memory window.
       Return and report this to the calling routine.
 Otherwise there is no overlap of the test window
 and the current valid memory window. Return zero.
```

```
NAME:
```

in_box function

```
 PURPOSE:
      This function determines if the pixel coordinates (i,j) lie within
      the box to which the pointer passed in the argument list points.

 CALLING SEQUENCE:
      integer_variable = in_box(i, j, box);

 INPUTS:
      i,j         pixel location
      box         box location structure

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns 0 if (i,j) is not in the box,
                      1 if (i,j) is in the box.)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      check_mmw_overlap   [su_dowindow.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments
```

```
NAME:
```

remember_mmw_corners function

```
 PURPOSE:
      This function stores the corner coordinates of a newly validated
      memory window in the array mmw_corners[].

 CALLING SEQUENCE:
      remember_mmw_corners(memory_window);

 INPUTS:
      memory_window      pointer to a memory window

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      dowindow       [su_dowindow.c (this file)]

 RESTRICTIONS:

  Note: at only marginal cost in CPU cycles, this function and the
        mmw_corners[] array could be deleted.  Instead, in the
        check_mmw_overlap() function, the corner coordinates would
        be calculated as is done here.  This would result in more
        readily understandable code.

 MODIFICATION HISTORY:
      12May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments



   Point to the first empty box.
   Add the new corners.
```

NAME:

make_detector_windows function

```
 PURPOSE:
      This function establishes the set of detector windows needed to
      cover the memory window to which the passed pointer refers.

 CALLING SEQUENCE:
      integer_variable = make_detector_windows(mmw);

 INPUTS:
      mmw         pointer to a memory window

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns 0 if window creation fails,
                      1 if window creation succeeds.)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      create_dtw              [su_dowindow.c (this file)]

 CALLED FROM:
      dowindow      [su_dowindow.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12May95   ADW   add standard comment header, structured English
comments
      28Oct98   CH    Major update on in-code comments



   Return if window has no pixels.
   Initialize x_remainder and dtw_x0.
   Continue until memory window is completely covered by detector
windows.
      Size of a detector window cannot be greater than max_dtw_size.
      (Re-)initialize y_remainder and dtw_y0.
         Size of a detector window cannot be greater than max_dtw_size.
         Make the detector window.
         Return 0 if detector window not created.
         Increment blc y coordinate for next detector window.
         Decrement number of mmw pixels in y to be assigned to a dtw.
      Increment blc x coordinate for next detector window.
      Decrement number of mmw pixels in x to be assigned to a dtw.
   Return 1 to indicate successful creation of required detector
windows.
```

```
NAME:
```

create_dtw function

```
 PURPOSE:
      This function establishes a single detector window and calculates
      the coefficients (aa, bb, kompx, and kompy) needed to decipher the
      blue detector data as a frame memory address, a task performed by
      the BLUE DSPs.

 CALLING SEQUENCE:
      integer_variable = create_dtw(x0, y0, xsize, ysize, mmw);

 INPUTS:
      x0          x coord at blc of det wndw
      y0          y coord at blc of det wndw
      xsize       det window x dimension
      ysize       det window y dimension
      mmw         pointer to a memory window

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns 0 for failed window creation,
                      1 for successful window creation.)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      make_detector_windows   [su_dowindow.c (this file)]

 RESTRICTIONS:

  Detector data from MIC are formatted as follows:

       bit(s)         contain(s)
  _______________________________________

  (msb)    23      double event flag
      14 - 22      Y address
       5 - 13      X address
       1 -  4      detector window ID
  (lsb)     0      parity


  The frame address is calculated according to:

      address = aa + ((X address + kompx) & 01FF) * bb +
                      (Y address + kompy) & 01FF

      (Note: 01FF, = 511, when used as a bit mask, selects the 9 LSB)

 MODIFICATION HISTORY:
      12May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments



   If there is enough memory, set the detector window parameters.
      Set detector window pointer and size and location information.
      Calculate the MIC window parameters for the detector window.
      Increment the number of windows allocated.
```

Return 1 to indicate successful DTW creation.

```
NAME:

      make_bfast          function

PURPOSE:
      This function create/modify the bfast structure in the PROC
      area to be fetched by the bluedsp.

 CALLING SEQUENCE:
      integer_variable = make_bfast(scw, bfast0);

 INPUTS:
      scw
      bfast0

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns -3 if (scw->param1 != bfast0->sample_time),
                      -2 if
(scw->xsize*scw->ysize+bfast0->n_fast_pixels>512)
                      -1 if (scw->mode != SCW_MODE_BFA)
                       0 otherwise [everything is okay].)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      dowindow      [su_dowindow.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments
```

```
NAME:
```

adjust_dtw_kompy function

```
PURPOSE:
     This function adjusts the detector window kompy parameters to
     compensate for the original UCL y-addressing scheme according
     to which detector x addresses and y addresses are calculated
     differently.  The purpose is to allow for CCD rows that are
     not read out.

CALLING SEQUENCE:
     adjust_dtw_kompy();

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:
     dowindow      [su_dowindow.c (this file)]

RESTRICTIONS:

MODIFICATION HISTORY:
     12May95   ADW   add standard comment header
     28Oct98   CH    Major update on in-code comments
```

```
  Initialize flags as if no CCD rows are read out.
  Turn off the "not-read" flags for CCD rows that are read out.
  Adjust the kompy values to compensate for rows not read.
     How many rows below the row containing dtw->y0 are not read?
     Shift kompy by the number of rows of MIC pixels not read.
     Make sure that kompy is in the range 0 to 511 (retain 9 lsb).
```

```
NAME:
```

shift_dtw function

```
 PURPOSE:
      This routine or its derivative will be used for real−time tracking

 CALLING SEQUENCE:
      integer_variable = shift_dtw(idxp, idyp);

 INPUTS:
      idxp
      idyp

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      dpul_read_wndwcfg
      dpul_write_wndwcfg
      dpul_check_wndwcfg

 CALLED FROM:
      This function is not currently in use.

 RESTRICTIONS:

      Not used in EOB−1.       CH08/08/92

 MODIFICATION HISTORY:
      12May95   ADW   add standard comment header, structured English
comments
      28Oct98   CH    Major update on in−code comments
```

```
NAME:
```

reshuffle function

```
 PURPOSE:
      This routine reorders the ggs list and put those ggs with badness
      >=4096 (i.e. no scw, mmw and dtw assignemen due to whatever
reason)
      to the bottom of the ggs list. The eventual number of ggs is
determined
      in the third loop of the dowindow().

 CALLING SEQUENCE:
      reshuffle();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      dowindow      [su_dowindow.c (this file)]
      *** The reference to reshuffle in dowindow is currently commented
***

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments


      find a ggs without home, put it back
         push up the stack
         place the problem one at back
```

```
NAME:

      su_dpuabort.c          (Swap unit)

 PURPOSE:
      This swap unit does nothing,
      then returns to the calling routine with status = 0.

 FUNCTIONS IN THIS FILE:
      dpuabort

 INCLUDE FILES:
      setup.h

 MODIFICATION HISTORY:
      18May95   ADW   add standard comment header

 RCS $Id: su_dpuabort.c,v 1.10 1996/08/03 13:47:46 welty Exp $
```

```
NAME:

      dpuabort

 PURPOSE:
      This swap unit does nothing,
      then returns to the calling routine with status = 0.

 CALLING SEQUENCE:
      swap_to = dpuabort();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none (returns 0)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      sunload         [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:

      03May95   ADW   Added standard comment header and other comments.


   Set the variable zero equal to 0.
   Return to calling routine with value 0.
```

NAME:

su_engineering.c (Swap unit)

PURPOSE:
This swap unit handles setup for and acquisition of engineering data.

FUNCTIONS IN THIS FILE:
engineering

INCLUDE FILES:
setup.h

MODIFICATION HISTORY:
```
10Nov95   ADW   first cut
22May96   ADW   remove references to science and memory windows
07Apr98   CH    major surgery
```

RCS $Id: su_engineering.c,v 1.10 1998/05/28 04:41:34 xmmom Exp $

```
NAME:

      engineering

 PURPOSE:
      This swap unit sets up detector window parameters and writes them
      into PROC memory so that engineering data are written into known
      memory locations.  For modes ENG_RAW_DATA, CENTROID_LUT_CALC,
      and ENG_PULSE_HEIGHT these window parameters do not correspond to
      detector pixels.  In the engineering modes, the ICU dictates the
      portion of the detector to use; the output engineering data will
      be compatible with the window parameters defined here.

 CALLING SEQUENCE:
      swap_to = engineering();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none (returns 0)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      sunload         [whitedsp.c]

 RESTRICTIONS:
      Assumes that pix_per_bus_access is a multiple of 8.

 MODIFICATION HISTORY:
      09Dec95   ADW   original code

   In sci_command the detector specification for engineering mode
   is not needed.  We use it here              to identify the
   section of the engineering swap unit to execute when running
   simulations.
   A value of "1" executes the detector window setup section, and
   a value of "0" executes the data processing section.

     but because the output of the data doesn't look at big word memory,
      pjs1 changed this back temporarily to SMALL_MIN to get it working
   Initialize the window parameters.
   Set up the window parameters for the
   specified engineering mode function.
      The centroid look up table calculation requires 2 256x256 windows
      to hold the M vs. N data for x and y.
      The centroid look up table confirmation requires a full frame
      at high resolution, which requires 16 512x512 windows.
      This requires all of small word memory.
      The centroid look up table confirmation requires a full frame
      at high resolution, which requires 16 512x512 detector windows.
      This requires all of small word memory.
       The pulse height & event energy mode requires
       2 1x256 windows to hold the respective arrays.
   Write the window parameters to PROC memory.
   This frame memory calculation depends on each mode having
   a window #1, and all i windows the same size and shape.
   We have to fix the number of allocated detector windows to include
   window 0.  Full frame actually uses it as one of the maximum of 16.

   q_alert( 0xA5E0, 1, &ndtw);
```

```
   Tell the Blue DSPs to fetch the detector
   window parameters from proc memory.
   In simulations, exit here for generation of data.
   Enable Blue DSP event collection.

      Send a beginning of exposure alert (fulfillment of ECR 90).
      The following in-line asm codes direct data to bluedsp1 only

      temp_q_alert_buf[0]=IC_DIRECT_DATA;
      temp_q_alert_buf[0]=0;
      do_00( temp_q_alert_buf );


      The following in-line asm codes direct data to bluedsp2 only

      temp_q_alert_buf[0]=IC_DIRECT_DATA;
      temp_q_alert_buf[0]=1;
      do_00( temp_q_alert_buf );


      If running in track-by-hand mode,
      idle until commanded to continue.
      If auto-piloting, wait for the exposure time to elapse.
      For the time being, let the exposure time be equal to
      the frame time (currently set at 10 seconds), as each
      engineering mode "observation" will be consist of a
      single "frame."  Eventually, engineering mode exposure
      times may be commandable.
      Add the exposure time to the bus_cycle_count to get
      the bus cycle count at the end of the exposure.
   Disable Blue DSP event collection.
   Direct data back to both bluedsps

   temp_q_alert_buf[0]=IC_DIRECT_DATA;
   temp_q_alert_buf[0]=5;
   do_00( temp_q_alert_buf );


   Read the detector window parameters from proc memory
   if running in simulation.  (Lost in 2 pass setup)
   Enter detector window parameters into the simulation log file.
   Read the engineering data from global memory
   and perform any required data processing.



  zero some memory ready for summing events later
      loop over 512 pixels in this window
          lut_array[grid_x][pixel_x % 8][grid_y][pixel_y % 8] +=
pix_counts[n++];
```

```
NAME:

      su_getref.c         (Swap unit)

 PURPOSE:
      This swap unit calculates (or obtains from global RAM) the roll
      center coordinates.  The nominal locations of guide stars are then
      expressed in a coordinate system with origin at the roll center
based
      on the rescaled current guide star coordinates from su_scangs and
      su_restore_stars.

      Used in tasks:      CHOOSE_GUIDE_STARS

      Prerequisites:
               current_coords - from su_scangs and
               su_restore_stars
               user-specified roll ctr coords (optional)
               keep_a_log - yes or no?
               log_indx - address of current place in log

      Provides:
               nominal_coords - translated to roll
               center frame of reference
               roll_ctr_x - x coord of roll ctr (0 -> 2047)
               roll_ctr_y - y coord of roll ctr (0 -> 2047)

      Global RAM access:   only a few words to read or write
                           => ~ 1 DPU cycle

      DSP RAM required:   very little

 FUNCTIONS IN THIS FILE:
      getref

 INCLUDE FILES:
      setup.h

 MODIFICATION HISTORY:
      19May95   ADW   add standard comment header
      09Jun95   ADW   change "printf(" to "fprintf(stderr," for
simulation
                      (timing of log messages was screwy with printf)

 RCS $Id: su_getref.c,v 1.20 1998/05/28 04:41:36 xmmom Exp $
```

```
NAME:

      getref

 PURPOSE:
      This swap unit calculates (or obtains from global RAM) the roll
      center coordinates.  The nominal locations of guide stars are then
      expressed in a coordinate system with origin at the roll center
based
      on the rescaled current guide star coordinates from su_scangs and
      su_restore_stars.

 CALLING SEQUENCE:
      swap_to = getref();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns 0)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      q_alert               [sun_util.c, cwhite.c]
      read_proc_int_array   [sun_util.c, global_access.c]
      write_proc_int        [sun_util.c, global_access.c]
      write_proc_int_array  [sun_util.c, global_access.c]

      return_bus            [sun_util.c] *** currently commented ***

 CALLED FROM:
      sunload               [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      ??????   CH    original code
      08Dec92  DTL   adapted for use in DPU
      14Dec92  DTL   nominal guide star coords based on
current_coords[]
      29Jan93  DTL   integer_coords[] and ctr_coords introduced
                     as integer equiv to coordinates for global RAM I/O
      03Feb93  DTL   added file open/close calls for Sun
      04Apr93  DTL   change "swap_id" to shared variable "swap_to";
                     send alert message to indicate completion of task
      05May95  ADW   add standard comment header


   If keep_a_log is enabled, make a log entry.

   Set array access parameters.
   Option 1: Center of detector field of view.  This is the Default!
   Set the pixel binning factors.
   Set ctr_coords.  Method depends on data representation.
   Option 2: User-specified roll center (on 2048 pixel scale)
      Set ctr_coords.
      Convert from integer representation to floats
   Option 0: Default: Roll center at center of mass of guide stars
      If stars_found is greater than 0, then
         Initialize ctr_coords x and y to 0.
If SAVE_GS_COORDS is defined as 1 (yes), then
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !!  NOTE: if the switch SAVE_GS_COORDS is not activated   !!
```

```
!!      advantage will be taken of the coordinates in     !!
!!      the shared_variables array current_coords[].      !!
!!      SAVE_GS_COORDS should be set to 1 only if it      !!
!!      turns out that there is insufficient room for     !!
!!      the coordinates in shared_variables.              !!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      Obtain integer_coords from PROC memory.
      Set pointer int_coord_ptr equal to pointer integer_coords.
      Set pointer last_ref_star equal to the memory location
      of the last star found (current_coords plus stars_found).
      For values of current from current_coords to last_ref_star
      (i.e., loop over the list of stars found).
         Convert coordinates from integer to floating point.
         Increment the int_coord_ptr pointer.
      Set pointer last_ref_star equal to the memory location
      of the last star found (current_coords plus stars_found).
      Loop over the list of stars found.
         Accumulate the sums of the current
         x and y coordinates in ctr_coords.
      Divide ctr_coords x and y by the number of stars foind.
      This gives the "center of mass" of the stars found.
   Convert coordinate data representation from floating point to
integer.
   Write the integer coordinates into PROC RAM.
   Set the roll center coordinates equal to ctr_coords.
   ( roll_ctr_x and _y are shared_variables)
   Save current (reference) frame coordinates of guide stars
   (relative to the roll center) as nominal_coords.
   Set the current and last_ref_star pointers to the memory locations
   of the first and last stars in the current stars_found list.
   If SAVE_GS_COORDS is defined as 1 (yes), then
   set the int_coord_ptr pointer to integer_coords.
   Loop over the stars in the current list.
      Set nominal x and y coordinates to the values of
      current coordinates minus roll_ctr coordinates.
      If SAVE_GS_COORDS is defined as 1 (yes), then
      convert from floating point to integer representation.
   If SAVE_GS_COORDS is defined as 1 (yes), then
   save the guide star coordinates in global RAM
   Initialize frame counter to 0
   What next?
   Return to the calling routine with status 0.
```

```
NAME:

      su_init_exp.c          (Swap unit)

 PURPOSE:
      This swap unit initializes the Blue detector window parameters to
the
      configuration required for field acquisition and reference frames
      (full frame, low resolution), and writes them into PROC RAM.

 FUNCTIONS IN THIS FILE:
      init_exp

 INCLUDE FILES:
      setup.h

 MODIFICATION HISTORY:
      19May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments

 RCS $Id: su_init_exp.c,v 1.9 1998/09/19 10:07:21 ho Exp $


Include setup.h.  This is for Sun and DPU.
Declare srtuctures.  (see our_own_types.h)
```

```
NAME:
```

init_exp function

```
 PURPOSE:

 CALLING SEQUENCE:
      swap_to = init_exp();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns 0)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      q_alert                [sun_util.c, cwhite.c]
      write_proc_int         [sun_util.c, global_access.c]
      write_proc_int_array   [sun_util.c, global_access.c]

 CALLED FROM:
      sunload                [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      11Feb93   DTL   original code
      11Mar93   DTL   new variable exposure initialized and declaration
                      of full_low_res_dtw[] moved to shared_variables.h
      17Mar93   DTL   frame_memory stored in proc memory
      04Apr93   DTL   change "swap_id" to shared variable "swap_to";
                      added alert to indicate initialization complete;
                      control_word initialized to all-zeros
      17Jan95   CH    Separate out the tasks for exposure
initialization.
                      Split from su_initialize.
      02May95   ADW   add standard comment header
      16May95   ADW   change to "int init_exp()" (was not declared int)
      28Oct98   CH    Major update on in-code comments

   Declare the integer variables.
   Initialize blue window parameters
   for full field of view & low resolution.
   Write detector window parameters to proc memory.
   Initialize portions of White local RAM for each exposure.
   Tell Blue DSPs to fetch the default detector window parameters from
the
   proc memory (the default parameters are full field low resolution).
   Tell the Blue DSPs to use the ping memory area.
   Return to calling routine with value 0.
```

```
NAME:

      su_initialize.c         (Swap unit)

 PURPOSE:
      This swap unit initializes shared variables with default values.
      It is used at DPU boot-up.

 FUNCTIONS IN THIS FILE:
      initialize

 INCLUDE FILES:
      setup.h

 MODIFICATION HISTORY:
      19May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments

 RCS $Id: su_initialize.c,v 1.39 1999/08/21 05:55:13 ho Exp $


The STRESS_SSI_TEST flag stresses the alert and SSI output mehcanism

The STRESS_DATA_Q flag tests the data q: the 30 1 problem circa Jan/Feb
1998

The SYNCH_CLOCK_TEST flag forces an synch clock error

The TEST_RAM flag sets the test rma configuration

The STRANGE_INIT flag creates the pacman problem, circa March 1996

If WHOLESALEINIT equals 1, initialize() will set all the variables to 0
before setting them to the values in the include files (called by
setup.h).
```

```
NAME:
```

initialize function

```
 PURPOSE:
      This swap unit initializes shared_defines and shared_variables
      variables with default values.  It is used at DPU boot-up.

 CALLING SEQUENCE:
      swap_to = initialize();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns swap_to [set to 0 below])

 INCLUDE FILES:

 SUBROUTINES CALLED:
      global_zero      [whitedsp.c]
      q_alerts         [whitedsp.c]
      write_proc_int   [global_access.c]

 CALLED FROM:
      sunload          [whitedsp.c]
      execute          [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      11Feb93   DTL   original code
      11Mar93   DTL   new variable exposure initialized and
                      declaration of full_low_res_dtw[]
                      moved to shared_variables.h
      17Mar93   DTL   frame_memory stored in proc memory
      04Apr93   DTL   change "swap_id" to shared variable "swap_to";
                      added alert to indicate initialization complete;
                      control_word initialized to all-zeros
      19May93   CH    Change the detector window initialization.
                      Prepare garbage collection from blue detector.
                      Force the detector to always load 16 dtws.
      03May95   ADW   add standard comment header, structured English
comments
      16May95   ADW   change to "int initialize()" (was not declared
int)
      09Jun95   ADW   change "printf(" to "fprintf(stderr," for
simulation
                      (timing of log messages was screwy with printf)
      28Oct98   CH    Major update on in-code comments

   If STRESS_SSI_ALERT, Run switch_ping_pong function
   If WHOLESALEINIT equals 1, then do the "wholesale" initialization.
   Initialize various addresses.
   Initialize status information variables.
   Initialize some parameters that will be *calculated* when needed.
   Initialize variables needed for housekeeping data.
   Initialize the WHITE DSP swap unit parameters.

 * The following are variables assigned to symbolic constants defined
 * in white_shared_variables.h.
 ***********************************************************************
*
   Initialize "several" integer variables.
```

```
Initialize "several" floating point variables.
Zero the small word and big word memories.
Initialize the integer parameters.
Initialize the floating point variables.
* Initialization of shared_variables
Some of the above are read from PROC memory by red_initialize_task
immediately after this initialization.  Therefore write those few
values to PROC memory now.                              ADW - 11Oct95
If running in simulation (SUN), then:
activate the DSP_ACTIVE bits in control_word.
Command the Red DSP to initialize.
Wait for Red DSP to finish its initialization.
If running in simulation (SUN), then:
Initialize bus_cycle_count (the "clock").
```

```
NAME:
```

test_ram function

```
 PURPOSE:
      This function call write_int_array and read_int_array.  These are
      low-level calls without bound checks. Use with extreme care!!

 CALLING SEQUENCE:
      test_ram();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_int_array   [sun_util.c; global_access.c]
      read_int_array    [sun_util.c; global_access.c]

 CALLED FROM:
      scanbs           [su_scanbs.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      28Oct98   CH    Major update on in-code comments


    set up the test pattern
    write the test array
    read it back
    Carefully send out the array in alerts
    set up the complimentary test pattern
    write the test array
    read it back
    Carefully send out the array in alerts
```

NAME:

w1

```
PURPOSE:
     Generic debug function for memory write access

CALLING SEQUENCE:
     w1(int tbase, int ysize, int yskip, int xsize, int *des);

INPUTS:
    tbase: base address of the array
    ysize: ysize
    yskip; skip in y
    xsize: xsize
    des: starting address of source;

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
     write_proc_int_array        [global_access.c]
     write_bw_int_array          [global_access.c]
     write_sw_int_array          [global_access.c]

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
     28Oct98   CH    Major update on in-code comments
```

```
 Write according to the where the target is
```

NAME:

r1

```
PURPOSE:
     Generic debug function for memory read access

CALLING SEQUENCE:
     r1(int tbase, int ysize, int yskip, int xsize, int *des);

INPUTS:
    tbase: base address of the array
    ysize: ysize
    yskip; skip in y
    xsize: xsize
    des: starting address of destination;

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
     read_proc_int_array        [global_access.c]
     read_bw_int_array          [global_access.c]
     read_sw_int_array          [global_access.c]

CALLED FROM:

RESTRICTIONS:


MODIFICATION HISTORY:
     28Oct98   CH    Major update on in-code comments

 Read according to the where the array is
```

NAME:

stress_data_q

```
PURPOSE:
     This function stresses the data queue.

CALLING SEQUENCE:
     stress_data_q();

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
     write_sw_int_array          [global_access.c]
     q_alert                     [cwhite.c]
     data_queue                  [white.asm]

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
     28Oct98   CH    Major update on in-code comments

   set up the numbers
   write those numbers into arrays
   send out an alerts
   submit the data queue
```

```
NAME:
```

submit_cq

```
 PURPOSE:
      This function submit mock compressed data queue.

 CALLING SEQUENCE:
      submit_cq();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_proc_int_array          [global_access.c]

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      28Oct98   CH    Major update on in-code comments


  set up the compress_q pointer
   Make sure the q pointer is within bounds
```

```
NAME:

      su_locategs.c          (Swap unit)

 PURPOSE:
      This swap unit calculates the locations of the guide stars in
      the most recently completed (i.e., previous) frame.  The tracking
      windows are each scanned for a bright star located approximately
at
      the guide star's last recorded location (current_coords) and new
      centroid coordinates are entered as the current_coords.

      Used in tasks:      TRACK_GUIDE_STARS

      Prerequisites:
               n_valid_mmw - from su_dowindow
               memory_windows - from su_dowindow
               n_valid scw - from su_dowindow
               science_windows - from su_dowindow
               ebgx - from su_calbgd
               ebgy - from su_calbgd
               keep_a_log - yes or no?
               log_indx - address of current place in log
               current_coords - last recorded coordinates
               ref_star_weight - last recorded counts in star
                       (to check for brightness chg)
               prev_frame - address of the last completed
                       tracking frame

      Provides:      current_coords - updated guide star coordinates

      Global RAM access:   read 258 words of window configuration data,
               and ~576 words/star_image * 10 images = 5760
               words => total ~ 6000 words => ~ 33 DPU cycles

               write ~96 words (if guide star coords saved)
               => 1 more DPU cycle

      DSP RAM required:   ~6000 words for variables

 FUNCTIONS IN THIS FILE:
      locategs
      find_brightest_pixel
      buildstar1
      summer1

 INCLUDE FILES:
      setup.h

 MODIFICATION HISTORY:
      19May95   ADW   add standard comment header
      09Jun95   ADW   change "printf(" to "fprintf(stderr," for
simulation
                      (timing of log messages was screwy with printf)
      28Oct98   CH    Major update on in-code comments

 RCS $Id: su_locategs.c,v 1.21 1998/05/28 04:41:39 xmmom Exp $

Include setup.h.  This is for Sun and DPU.
Declare global integer variables.
Declare arrays of int2 variables (see our_own_types.h).
Declare global structures.
```

```
NAME:
```

locategs function

```
 PURPOSE:
      This swap unit calculates the locations of the guide stars in
      the most recently completed (i.e., previous) frame.  The tracking
      windows are each scanned for a bright star located approximately
at
      the guide star's last recorded location (current_coords) and new
      centroid coordinates are entered as the current_coords.

 CALLING SEQUENCE:
      swap_to = locategs();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns 0)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      buildstar1             [su_locategs.c (this file)]
      find_brightest_pixel   [su_locategs.c (this file)]
      read_proc_int          [sun_util.c, global_access.c]
      read_proc_int_array    [sun_util.c, global_access.c]
      read_sw_int_array      [sun_util.c, global_access.c]
      write_proc_int         [sun_util.c, global_access.c]
      write_proc_int_array   [sun_util.c, global_access.c]

 CALLED FROM:
      sunload                [whitedsp.c]
      execute                [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      00Jul90   CH    original code
      16Dec92   DTL   adapted to run on DPU
      28Jan93   DTL   current_coords to integer_coords[]
                      if saving in global RAM.
      03Feb93   DTL   added file open/close calls for Sun
      04Apr93   DTL   change "swap_id" to shared variable "swap_to"
      11May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments

   Declare the local integer variables.
   Declare pointers to integers.
   Declare local structures. (see our_own_types.h)
   Declare the functions called.

   Read the threshold data from global memory.
   Read the window configuration data from global RAM.
   Loop through the list of valid science windows.
      If the window is a tracking window, then:
         point to the corresponding memory window,
         point to the corresponding guide star coordinates,
       If BPE binning, scale the star position
       to the binned coordinates
         Calculate the coordinates of the
         bottom left corner of the search box
         If running in simulation and if DEBUGging is enabled,
```

```
        print a status message.
        Extract the image of this search box
        from previous the frame.
        Set the estimated coordinates of star in the search box.
        Determine the region within which
        to search for the brightest pixel.
        Find the brightest pixel in the vicinity of (x_bp, y_bp)
        Call buildstar1 to analyze the bright star image.
        Transform the coordinates to detector reference frame.
        Apply the rescaling due to BPE bining
        Calculate the equivalent floating point values.
        If running in simulation and if DEBUGging is enabled,
        print a status message.
        Check for brightness fluctuation.


        Check for position fluctuation.

  If SAVE_GS_COORDS equals 1, then
  Save the new coordinates and weights in global memory.
  Return value 0 to calling routine.

     Auxiliary functions
```

NAME:

find_brightest_pixel function

```
PURPOSE:
     This function finds the brightest pixel in the rectangle
     bounded by left_edge, right_edge, bottom_edge, top_edge.
     The pixel coordinates are returned in the pointers ixp and iyp.

CALLING SEQUENCE:
     find_brightest_pixel(&x_bp, &y_bp, left_edge, right_edge,
                   bottom_edge, top_edge);

INPUTS:
     ixp            pointer to pixel x coord
     iyp            pointer to pixel y coord
     left_edge      min x search boundary
     right_edge     max x search boundary
     bottom_edge    min y search boundary
     top_edge       max y search boundary

OPTIONAL INPUT PARAMETERS:

OUTPUTS:
     ixp         pointer to pixel x coord
     iyp         pointer to pixel y coord

INCLUDE FILES:

SUBROUTINES CALLED:
     write_proc_int  [sun_util.c; global_access.c]

CALLED FROM:
     locategs        [su_locategs.c (this file)]

RESTRICTIONS:

MODIFICATION HISTORY:
     11May95   ADW   add standard comment header
     28Oct98   CH    Major update on in-code comments

  Declare the local integer variables.

  If keep_a_log is enabled, make a log entry.
  Save the initial coordinates for DEBUG printout.
  Initialize max_counts to counts in the starting pixel
  Scan subset of star_image to find position with maximum counts.
  Loop through columns.
     Loop through rows.
        If the counts in any window pixel
        exceed the current max_counts, then
           set coordinates to that pixel
           and update max_counts.
  If running in simulation and if DEBUGging is enabled,
  then print information about the brightest pixel.
```

NAME:

buildstar1 function

```
PURPOSE:
     This function integrates over a stellar image and extracts total
     counts and centroid.  This is a subset of the functionality of
     buildstar() in su_scanbs.c.  Summation is carried out separately
     over each of four quadrants using function ``summer1.''

CALLING SEQUENCE:
     star = buildstar1(x_bp, y_bp);

INPUTS:
     x_bp         x pos'n of bright pixel
     y_bp         y pos'n of bright pixel

OPTIONAL INPUT PARAMETERS:

OUTPUTS:
     none   (returns the structure "star")

INCLUDE FILES:

SUBROUTINES CALLED:
     summer1         [su_locategs.c (this file)]
     write_proc_int  [sun_util.c; global_access.c]

CALLED FROM:
     locategs        [su_locategs.c (this file)]

RESTRICTIONS:

MODIFICATION HISTORY:
     11May95   ADW   add standard comment header
     28Oct98   CH    Major update on in-code comments
```

```
  Declare the local integer variables.
  Declare the local structure variables.
  Declare the local floating point variables.
  Declare the functions called.

  Initialize the accumulators xmnt and ymnt.
  Set the integration boundaries.
  Set the increment factors.
  Call summer1 to sum over the first quadrant.
  Set the integration boundaries.
  Reset the x increment factor.
  Call summer1 to sum over the second quadrant.
  Set the integration boundaries.
  Reset the y increment factor.
  Call summer1 to sum over the third quadrant.
  Set the integration boundaries.
  Reset the x increment factor.
  Call summer1 to sum over the fourth quadrant.
  If there are counts then
     calculate the star image centroid and
     transform to subimage reference frame.
     (different method for different data types)
  If cts is less than or equal to 0
     assume centroid is at estimated position.
     If running in simulation (SUN) and DEBUGging is enabled,
```

```
      then print a "No guide star found" warning message.
   Set star.count to cts, the integrated counts.
   Return to the calling routine with the star structure.
```

```
NAME:
```

summer1 function

```
 PURPOSE:
      This function sums over pixels in a quadrant bounded by int_xmin,
      int_xmax, int_ymin, and int_ymax in a direction controlled by dx
and dy.
      (The values of dx and dy are, respectively, (1,1) in quadrant I;
      (-1, 1) in quadrant II; (-1, -1) in quadrant III; and (1, -1) in
      quadrant IV.)
      Summation is terminated either when the integration range
encompasses
      3-sigma for the psf (n3sig = 500), or when the local count level
drops
      to the background level (ebgx or ebgy) estimated in su_calbgd().
The
      value returned is the number of counts in the quadrant.
Functionality
      is the same as that in summer() in su_scanbs.c except that second
      moments and summation boundaries are not calculated here.

 CALLING SEQUENCE:
      cts = summer1(x_bp, y_bp);

 INPUTS:
      x_bp
      y_bp

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns counts_in_quadrant)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_proc_int  [sun_util.c; global_access.c]

 CALLED FROM:
      buildstar1      [su_locategs.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      11May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments


   Declare the local integer variables.
   Make entry in log (optional)
   Scan quadrant in raster fashion over the star image.
   Initialize the counts_in_quadrant accumulator to 0.
   If there is more than one pixel in the quadrant, then
   (otherwise there is nothing to accumulate)
      Scan in x from x=int_xmin to int_xmax-1.
      While x is not equal to int_xmax...
      (x should approach from below...)
         Get column distance from peak.
         Initialize counts_in_column.
         Scan in y from y=int_ymin to int_ymax-1.
         Get the counts in the pixel given by the current x and y.
         While y is not equal to int_ymax...
         (y should approach from below...)
            Get pixel location relative to peak.
```

```
            Accumulate sums for moments
            Terminate column sum at "edge of image"
            Increment y to the next position in the column.
         Accumulate the sum of counts in the quadrant.
         Terminate row sum at "edge of image"
         Increment x to the next column.
   If running in simulation (SUN) and if DEBUGging is enabled, then
   if there are no counts in the quadrant, print for diagnosis.
   Return total number of counts_in_quadrant to the calling routine.
```

```
NAME:

      su_makeags_id.c         (Swap unit)

 PURPOSE:
     This swap unit identifies the uplinked guide stars in a list of
bright
  stars generated by su_scanbs (with positions compensated for detector
  distortion by su_restore_stars).  The identifications are made solely
  on the basis of position comparison, however the bright stars are
  checked in order of decreasing brightness.  The total search space is
  set to the nominal spacecraft pointing error but is subdivided into a
  grid of N x N search boxes, where N is given by the
SEARCH_SPACE_DIMENSION
  parameter.  As an initial guess, the true spacecraft orientation is
  assumed to deviate from the commanded position by the search box
  coordinates.  For each initial guess, first the guide stars are sought
  in a wide search range; a second iteration is deemed worthwhile if at
  least half of the guide stars are identified using the wide search
  range.  On the second iteration the true spacecraft position is
  assumed to deviate from the commanded position by an amount given by
  the translation and roll offsets calculated (in su_caldrft) during the
  first iteration, and then the stars are required to lie within a
relatively
  narrow range of their expected positions in order to be "identified."
If
  all but 4 (or more) of the guide stars are identified, then the field
is
  said to have been acquired and the final translation and roll offsets
  give the estimated deviation of the true spacecraft orientation from
the
  commanded orientation.  If no acceptable solution has been found after
  all boxes have been searched, then the acquisition is said to have
  failed and the commanded spacecraft orientation is accepted as the
true
  orientation by default (i.e., the translation and roll offsets are set
  to 0).

  Note that the uplinked guide star coordinates are with respect to the
  commanded satellite orientation and have had the roll center offsets
  subtracted.

      Used in tasks:      ACQUIRE_FIELD

      Prerequisites:
               bright star catalog sorted by decreasing brightness
               and with positions "restored"
               ref_stars - coordinates of guide stars
               roll_ctr_x - x coord of roll center
               roll_ctr_y - y coord of roll center
               keep_a_log - yes or no?
               log_indx - address of current place in log

      Provides:
               nominal_coords[] - locations of the matched guide stars
               ref_star_quality[] - quality of guide stars (all are
"good")
               stars_found - number of guide stars found
               current_coords[] - tentative id of guide star
                        as a cataloged bright star for use by su_caldrft
               ref_star_weight[] -   weights to use in
                           offset/roll calculation (caldrft swap unit)
               box - search box index for future reference
                  by this swap unit
               acquisition_iter - iteration counter for
```

```
                  future reference by this swap unit

      Global RAM access:   read ~ 2 * [(# ref stars) + (# bright stars)]
                  words (max 1090 words)

               write ~ 3 words, but only once per final
                  solution, not every time this swap
                  unit is called (i.e., negligible
                  bus demand)

               => ~7 DPU cycles

      DSP RAM required:   ~1.1 K words (sensitive to MAX_BRIGHT_STARS)

 FUNCTIONS IN THIS FILE:
      makeags_id
      look_for_ref_stars

 INCLUDE FILES:
      setup.h

 MODIFICATION HISTORY:
      19May95   ADW   add standard comment header
      09Jun95   ADW   change "printf(" to "fprintf(stderr," for
simulation
                      (timing of log messages was screwy with printf)

 RCS $Id: su_makeags_id.c,v 1.17 1997/09/09 15:19:41 ho Exp $


Include setup.h.  This is for Sun and DPU.
Declare global floating point variables.
Declare global structure variables.
```

```
NAME:

      makeags_id          function

 PURPOSE:
      see PURPOSE above

 CALLING SEQUENCE:
      swap_to = makeags_id();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns -2 if no reference stars are found,
                      -1 if field acquisition fails,
                       0 upon successful field acquisition, and
                       ITERATE (set to 1023 in white_default.h) to
iterate.)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      look_for_ref_stars    [su_makeags_id.c (this file)]
      read_proc_int         [sun_util.c; global_access.c]
      read_proc_int_array   [sun_util.c; global_access.c]
      write_proc_int        [sun_util.c; global_access.c]
      write_proc_int_array  [sun_util.c; global_access.c]

      q_alert               [sun_util.c, cwhite.c] *** currently
commented ***

 CALLED FROM:
      sunload               [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      04Nov91   CH    original code
      29Sep92   DTL   major revision, optimized to DPU
      13Oct92   DTL   guide star coords and default quality
                      provided for use in su_caldrft
      08Dec92   DTL   option to save or not save guide star
                      coordinates to global RAM
      28Jan93   DTL   integer_coords[] added for saving in global RAM;
                      float <-> int conversion
                      for ref_stars[] and bright_stars[]
      05Feb93   DTL   added file open/close calls for Sun
      11Mar93   DTL   initialization of acquisition_iter and box moved
to
                      su_manager; added capability to queue field acq
data
      04Apr93   DTL   change "swap_id" to shared variable "swap_to";
                      send alert message when field acquisition task
completed
      11May95   ADW   add standard comment header


   Declare the local integer variables.
   Declare the local floating point variables.
   Declare the local structures.
   If SAVE_GS_COORDS equals 1, then
   declare items used only in that case.
   Declare the functions called.
```

```
   Calculate search box parameters.
   Read the number of reference stars from memory.
   If n_ref_stars is not positive, then
      adopt default offsets equal to 0.
      Convert the offsets to integer representation.
      Save results in global memory
      If running in simulation, print a status message.
      If there are no stars, we're done, so
      set ref_stars_found equal to stars_found,
      and swap_to to 0.
      Return to calling routine with alert value.
   Set the array access parameters yskip and xsize.
   Read the reference star coordinates from global memory.
   Read n_bright_stars from global memory.
   If running in simulation (SUN) and if DEBUGging is enabled,
   then print a message with current star information.
   Initialize the array pointers last_bright_star and last_ref_star.
   Read the bright star coordinates from global memory
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !! NOTE: this could be done more efficiently if we !!
   !!    store the bright star coordinates separately !!
   !!    from the bright_stars Star structure array.  !!
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   While the value of box is less than n_search_boxes...
   (i.e., step through all the search boxes)
      Set the box coordinates
      Iterate toward field acquisition.
      If acquisition_iter is less than max_makeags_iterations, then
         Set initial guess and search range based on iteration number.
         If acquisition_iter equals 0, then
            Initial guess based on search box
            Set the search_range to wide_search_range.
         Else (not the first iteration)
            Initial guess based on previous iteration
            Set the search_range to narrow_search_range.
         Set stars_found equal to the result
         from a call to look_for_ref_stars.
         If running in simulation (SUN) and if DEBUGging is enabled,
         print star, range, and deltas information.
         The response is based on number of matches:
         If stars_found is less than half n_ref_stars, then
            there is no need to iterate.
            Set up for the next box.
         Else if (stars_found >= n_ref_stars/2) then
            we have a potential solution,
            and we should iterate.
            If running in simulation (SUN), print a status message.
            Increment acquisition_iter,
            set swap_to to SWAP_CALDRFT,
            and return ITERATE status.
      Else (we're done iterating)
         Set offsets and count matches in narrow search range
         If running in simulation (SUN) and if DEBUGging is enabled,
         print star, range, and deltas information.
         Our response is based on number of matches:
 *         !!  NOTE: the value of allowed_to_miss, which defaults !!
 *         !!      to the #defined value ALLOWED_TO_MISS,         !!
 *         !!      currently 4, may need to be adjusted in flight !!
 *         !!      operations on a pointing by pointing basis.    !!
 *         !!      This option exists in the sense that a ground  !!
 *         !!      command could override the default.            !!
 *         !!                                                     !!
 *         !!      In sim code, allowed_to_miss was hard-wired to !!
 *         !!      4, but simulations assumed n_ref_stars = 16.   !!
 *         !!      (i.e., that 16 reference stars would be        !!
```

```
*          !!      available).                                     !!
*          !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        if stars_found is greater than n_ref_stars minus
        allowed_to_miss, then (allow for missed ref stars)
           Accept current offsets as final solution.
           Convert to integer representation
           Save results in global memory
           IF SAVE_GS_COORDS equals 1, then save the star coordinates.
           Task ACQUIRE_FIELD completed (send alert to ICU).
           Return 0.
        Else if not (stars_found > (n_ref_stars-allowed_to_miss))
           The solution is not good.  Try the next box.
           Reset acquisition_iter and go to next_box.
     Increment box.
  Set default offsets in the case of field acquisition failure.
  Convert to integer representation
  Save results in global memory
  If SAVE_GS_COORDS equals 1, save the coordinates also.
  Task ACQUIRE_FIELD completed (send alert to ICU).
  Return -1.

     Auxiliary functions
```

```
NAME:

      look_for_ref_stars       function

 PURPOSE:
      This function looks for matches between the absolute reference
      stars and the detected bright stars.  A match implies agreement
      in x and y coordinates to within a specified pixel range.  The
      number of matches is returned.

 CALLING SEQUENCE:
      stars_found = look_for_ref_stars();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns matches)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      makeags_id      [su_makeags_id.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      11May95   ADW   add standard comment header, structured English
comments


Declare the local variables.
Initialize array pointers and counter
   Loop through the list of absolute reference stars.
      Convert integer representation to floating point representation
      Location expected based on current estimate of deviation
      of spacecraft orientation from commanded orientation

 *      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
 *      !! NOTE: The roll center coordinates, Xo and Yo, had     !!
 *      !!     been subtracted previously to obtain the ref_star !!
 *      !!      coordinates.                                     !!
 *      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

      Initialize the roll center coordinates.
      Calculate the expected coordinates.
      Check for a matching bright star.
      Initialize an index.
      For values of bright_star_ptr from bright_stars
      to last_bright_star-1, loop (through the star list).
         Convert from integer to floating point representation
         If one is found ... then
            Set coordinates and weights for use in su_caldrft
            Go to match_found (i.e., stop looking)
            (ready to check the next ref star)
         Increment the index.
Come here when a match is found.  Do nothing (currently).  To next star.
   Return to calling routine with the number of matches.
```

```
NAME:
```

su_restore_stars.c (Swap unit)

```
 PURPOSE:
      This swap unit restores the coordinates of stars in a "catalog" to
the
      unbinned detector pixel scale and uses polynomial approximations
to
      the MIC distortion to compensate for spatial non-linearity.

      Used in tasks:
               CHOOSE_GUIDE_STARS
               ACQUIRE_FIELD
               TRACK_GUIDE_STARS

      Prerequisites:
               nbs - number of bright stars
               star catalog - bright stars or guide stars
                     if guide stars, then need
                     either nominal_coords[] or current_coords[]
               keep_a_log - yes or no?
               log_indx - address of current place in log

      Provides:
               scaled and/or distortion-corrected star coords
                  if bright stars then I/O to global RAM
                  if guide stars then I/O to shared array
                    (nominal_coords or current_coords) and
                    copy sent to global RAM

      Global RAM access:
               ~20 * number of bright stars words if processing
BRIGHT_STARS

                  (=> max 1024 words or 4 DPU cycles)
               or

               ~2 * number of guide stars words if processing
                  INITIAL_GUIDE_STAR_COORDS or
                  CURRENT_GUIDE_STAR_COORDS

                  (=> max 64 words or 1 DPU cycle)

      DSP RAM required:   ~ 200 words

 FUNCTIONS IN THIS FILE:
      restore_stars
      restore
      poly_3_model
      poly_4_model
      poly_5_model
      poly_7_model

 INCLUDE FILES:
      setup.h

 MODIFICATION HISTORY:
      19May95   ADW   add standard comment header

 RCS $Id: su_restore_stars.c,v 1.22 1998/05/28 04:41:42 xmmom Exp $


Include setup.h.  This is for Sun and DPU.
```

```
NAME:

      restore_stars          function

 PURPOSE:
      This swap unit restores the coordinates of stars in a "catalog" to
the
      unbinned detector pixel scale and uses polynomial approximations
to
      the MIC distortion to compensate for spatial non-linearity.

 CALLING SEQUENCE:
      swap_to = restore_stars();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns -1 for invalid task_id, previous_swap_unit spec.,
                     0 for successful completion)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      q_alert               [sun_util.c, cwhite.c]
      read_proc_int         [sun_util.c, global_access.c]
      read_proc_int_array   [sun_util.c, global_access.c]
      restore               [su_restore_stars.c (this file)]
      write_proc_int        [sun_util.c, global_access.c]
      write_proc_int_array  [sun_util.c, global_access.c]

      return_bus            [sun_util.c] *** currently commented ***

 CALLED FROM:
      sunload               [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      ?????9?   CH    original code
      22Sep92   DTL   converted to proto-flight code
      21Oct92   DTL   not used to rescale bright stars in
CHOOSE_GUIDE_STARS
      02Dec92   DTL   clarify distinction between scaling and distortion
                      correction; reinstate usage in CHOOSE_GUIDE_STARS
      04Dec92   DTL   separate operations on bright stars and guide
stars;
                      scale bright star psf integ ranges and moments as
well
                      as coordinates.
      08Dec92   DTL   option to save or not save restored
                      guide star coords to global RAM
      29Jan93   DTL   Points and Twomnts converted between
                      ints and floats for global RAM I/O
      01Feb93   DTL   added file open/close calls for Sun
      25Feb93   DTL   floating format used in printfs
      04Apr93   DTL   change "swap_id" to shared variable "swap_to"
      11May95   ADW   add standard comment header
      09Jun95   ADW   change "printf(" to "fprintf(stderr," for
simulation
                      (timing of log messages was screwy with printf),

   Declare the local integer variables.
   Declare the structure variables.
```

```
Declare the functions called.

Set catalog address, binning factors, and coordinate scaling
and distortion correction flags according to context
Possibilities are:
task = ACQUIRE_FIELD,
task = CHOOSE_GUIDE_STARS and previous swap unit = SWAP_SCANBS,
task = CHOOSE_GUIDE_STARS and previous swap unit = SWAP_DOWINDOW, and
task = TRACK_GUIDE_STARS.
Set default values in case there's a problem.
If none of those conditions is met, send an alert and return −1.
Set the array access parameters xsize and yskip.
If the star_catalog is BRIGHT_STARS, then process the bright stars.
   Read the number of stars in the current catalog from memory.
   Initialize a counter for the number of stars processed so far.
   Start a loop over the list of bright stars.
      Get the address of the next star's data.
      Calculate how many stars can be read now.
      Read the bright_star data from global memory.
      Loop over the stars in the current batch.
         Convert coordinates and moments from integer
         to floating point representation.
         If the scale_coords flag is set
            scale the coordinates to 2048x2048 detector format,
            Add 0.5 to adjust for possible round−off errors,
            put the new box values into the star structure,
            convert moment values to integer,
            and put the new moment values in the star structure.
         If the correct_distortion flag is set,
         (Normally correct_distortion = 0 or 1.)
            call the function restore to calculate the
            spatial distortion−corrected coordinates.
         Convert coordinates from floating point to integer
         representation and replace in the star structure.
      Save results in global memory.
      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
         !! NOTE: The current implementation will overwrite the  !!
         !!   original star catalog.  This needn't be the case.  !!
         !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      Increment stars_examined by the number of stars
      processed in the batch just completed.
If star_catalog is not BRIGHT_STARS, then process the guide stars.
   Set n_stars to stars_found, the number of stars in the BSC.
   If star_catalog equals CURRENT_GUIDE_STAR_COORDS,
      point to the current guide star coordinates.
   Otherwise,
      point to the nominal guide star coordinates.
   Initialize global_address if we are
   saving coordinates in global memory.
   Loop over all the stars in the catalog.
      If the scale_coords flag is set,
         scale to 2048x2048 detector coordinates
         and adjust for round−off.
      If the correct_distortion flag is set,
         call the function restore to calculate
         the spatial distortion−corrected coordinates.
      Move the pointer index to star_coordinates.
      IF SAVE_GS_COORDS is set to 1, save results in global memory.
      Convert coordinates from floating point
      to integer representation and replace.
      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
         !! NOTE: The current implementation will overwrite the !!
         !!   original coordinates.  This needn't be the case.  !!
         !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Return to calling routine with value 0.
```

```
Auxiliary functions
```

```
NAME:

      restore          function

 PURPOSE:
      This function delegates responsibility for the spatial distortion
      correction to one of several polynomial model functions based on
the
      value of restoration_mode.

 CALLING SEQUENCE:
      star_coordinates = restore(star_coordinates);

 INPUTS:
      star_coordinates   coordinates to be corrected for distortion

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns FLT_Point structure corrected)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      poly_3_model()      [su_restore_stars.c (this file)]
      poly_4_model()      [su_restore_stars.c (this file)]
      poly_5_model()      [su_restore_stars.c (this file)]
      poly_7_model()      [su_restore_stars.c (this file)]

 CALLED FROM:
      restore_stars       [su_restore_stars.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      11May95   ADW   add standard comment header


   Declare the local structures.
   Declare the functions called.
......................................................................
..
   Set "original" coordinates to star_coordinates.
   Call the appropriate model function;
   restoration_mode has a value equal to
   the desired polynomial order.
   If restoration_mode has an "improper" value,
   simply set corrected equal to original coordinates.
   Return the corrected coordinates to the calling routine.
```

NAME:

poly_3_model ***function***

PURPOSE:
This function is a 3rd order polynomial approximation to the MIC spatial distortion.

CALLING SEQUENCE:
```
corrected =  poly_3_model(original);
```

INPUTS:
original   original coordinates

OPTIONAL INPUT PARAMETERS:

OUTPUTS:
none   (returns FLT_Point structure corrected)

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:
restore   [su_restore_stars.c (this file)]

RESTRICTIONS:

MODIFICATION HISTORY:
11May95   ADW   add standard comment header

NAME:

poly_4_model function

PURPOSE:
This function is a 4th order polynomial approximation to the MIC spatial distortion.

CALLING SEQUENCE:
```
corrected =  poly_4_model(original);
```

INPUTS:
original   original coordinates

OPTIONAL INPUT PARAMETERS:

OUTPUTS:
none   (returns FLT_Point structure corrected)

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:
restore   [su_restore_stars.c (this file)]

RESTRICTIONS:

MODIFICATION HISTORY:
11May95   ADW   add standard comment header

```
NAME:

      poly_5_model       function

 PURPOSE:
      This function is a 5th order polynomial approximation to the MIC
      spatial distortion.

 CALLING SEQUENCE:
      corrected =  poly_5_model(original);

 INPUTS:
      original      original coordinates

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns FLT_Point structure corrected)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      restore         [su_restore_stars.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      11May95   ADW   add standard comment header
```

```
NAME:

      poly_7_model       function

 PURPOSE:
      This function is a 7th order polynomial approximation to the MIC
      spatial distortion.

 CALLING SEQUENCE:
      corrected =  poly_7_model(original);

 INPUTS:
      original      original coordinates

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns FLT_Point structure corrected)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      restore         [su_restore_stars.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      11May95   ADW   add standard comment header
```

```
NAME:
```

su_rtrd.c (Swap unit)

```
 PURPOSE:
      This swap unit controls the data acquisition, processing and
      output for the real-time raw data mode.

 CALLING SEQUENCE:
      swap_to = rtrd();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none (returns 0)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      sunload         [whitedsp.c]
      execute         [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      28Oct98   CH    Major update on in-code comments

RCS $Id: su_rtrd.c,v 1.5 1999/08/21 05:56:39 ho Exp $

   This code is to be implemented for the RTRD mode
```

```
NAME:
```

rtrd

```
 PURPOSE:
      This swap unit controls the data acquisition, processing and
      output for the real-time raw data mode.

 CALLING SEQUENCE:
      swap_to = rtrd();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none (returns 0)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      sunload         [whitedsp.c]
      execute         [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      28Oct98   CH    Major update on in-code comments


   This code is to be implemented for the RTRD mode
--------
Command the Red DSP to do the shift-and-add process.
   Direct data to the single RTRD blue processor.
   Turn on RTRD on rtrd_processor.
   Turn on the blue event processing promptly.
   Send a beginning of exposure alert.
   disable Blue DSP event processing
   and issue an "end of exposure" alert.
   Reset the data direction in the arbitor to BOTH blue DSPs
   disable Blue DSP RTRD mode
   Force Red DSP to flush the RTRD memory
   Command the Red DSP to do the shift-and-add process.
```

NAME:

engineering

```
PURPOSE:
     This swap unit sets up detector window parameters and writes them
     into PROC memory so that engineering data are written into known
     memory locations.  For modes ENG_RAW_DATA, CENTROID_LUT_CALC,
     and ENG_PULSE_HEIGHT these window parameters do not correspond to
     detector pixels.  In the engineering modes, the ICU dictates the
     portion of the detector to use; the output engineering data will
     be compatible with the window parameters defined here.

CALLING SEQUENCE:
     swap_to = engineering();

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:
     none (returns 0)

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:
     sunload         [whitedsp.c]

RESTRICTIONS:
     Assumes that pix_per_bus_access is a multiple of 8.

MODIFICATION HISTORY:
     09Dec95   ADW   original code


  In sci_command the detector specification for engineering mode
  is not needed.  We use it here              to identify the
  section of the engineering swap unit to execute when running
  simulations.
  A value of "1" executes the detector window setup section, and
  a value of "0" executes the data processing section.
  Initialize the window parameters.
  Set up the window parameters for the
  specified engineering mode function.
     The centroid look up table calculation requires 2 256x256 windows
     to hold the M vs. N data for x and y.
     The centroid look up table confirmation requires a full frame
     at high resolution, which requires 16 512x512 windows.
     This requires all of small word memory.
     The centroid look up table confirmation requires a full frame
     at high resolution, which requires 16 512x512 detector windows.
     This requires all of small word memory.
     The pulse height & event energy mode requires
     2 1x256 windows to hold the respective arrays.
  Write the window parameters to PROC memory.
  This frame memory calculation depends on each mode having
  a window #1, and all i windows the same size and shape.
  We have to fix the number of allocated detector windows to include
  window 0.  Full frame actually uses it as one of the maximum of 16.

  q_alert( 0xA5D0, 1, &ndtw);
  Tell the Blue DSPs to fetch the detector
  window parameters from proc memory.
```

```
   In simulations, exit here for generation of data.
   Enable Blue DSP event collection.


      The following in-line asm codes direct data to bluedsp1 only

      temp_q_alert_buf[0]=IC_DIRECT_DATA;
      temp_q_alert_buf[0]=0;
      do_00( temp_q_alert_buf );


      The following in-line asm codes direct data to bluedsp2 only

      temp_q_alert_buf[0]=IC_DIRECT_DATA;
      temp_q_alert_buf[0]=1;
      do_00( temp_q_alert_buf );


      If running in track-by-hand mode,
      idle until commanded to continue.
      If auto-piloting, wait for the exposure time to elapse.
      For the time being, let the exposure time be equal to
      the frame time (currently set at 10 seconds), as each
      engineering mode "observation" will be consist of a
      single "frame."  Eventually, engineering mode exposure
      times may be commandable.
      Add the exposure time to the bus_cycle_count to get
      the bus cycle count at the end of the exposure.
   Disable Blue DSP event collection.
   Direct data back to both bluedsps

   temp_q_alert_buf[0]=IC_DIRECT_DATA;
   temp_q_alert_buf[0]=5;
   do_00( temp_q_alert_buf );


   Read the detector window parameters from proc memory
   if running in simulation.  (Lost in 2 pass setup)
   Enter detector window parameters into the simulation log file.
   Read the engineering data from global memory
   and perform any required data processing.
         Initialize the 64 element LUT check array.
         The time stamps (two per frame time in current mic1.c -
22May96)
         add a "count" each to pixel 0,0.  Correct for that here.
         Note:  the frame_time in mic1.c is 0.01 sec and the exposure
         time here is 10 sec; thus 2000 markers will be present.
However
         the frame_time variable here contains the number of
milliseconds
         in the frame, thus the division by 10.  Obviously, this will
         require some clean up...                         ADW  22May96

         base = (*dtw).aa;
         read_sw_int (base, &izero);
         izero -= 2 * frame_time / 10;
         write_sw_int (base, &izero);

         Loop over x dimension of LUT check array.
         Set the base address for the first column
         at the current x value,
         and read each of the 64 columns to be added.
                  Read a block of pixel values into pix_counts.
                  Loop over y dimension of LUT check array.
                  Add pixels into appropriate LUT array element.
```

```
            Increment the address by the amount just read.
         Increment the address to the start of the next column.
   Write the LUT array to memory.
```

NAME:

su_scanbs.c (Swap unit)

```
 PURPOSE:
      This swap unit makes a catalog of bright stars in an image as
follows:
   (1) a reference frame is searched in raster fashion for pixels
      with more than a threshold number of counts and those pixels
      are entered into a buffer;
   (2) when the buffer becomes full, the neighborhood of each bright
      pixel is searched for the single brightest pixel;
   (3) image properties of the bright spot (e.g., counts, centroid
      position, roundness) are derived and recorded in the "bright
      star catalog," bright_stars[];
   (4) finally, an index array is created which can be used to access
      cataloged stars in order of decreasing brightness.  (The indices
      are used to save the bright star catalog in global memory in
      this order.)

      Notes:

      In the CHOOSE_GUIDE_STARS task, the bright star catalog is
searched for
      star concentrations (i.e., crowded fields) and these are recorded
for
      later use as exclusion zones when guide stars are sought for
tracking
      in su_scangs.c.

      In this swap unit, the detector field of view is mapped into
      nmaxx x nmaxy (nominally 1024 x 1024) "binned pixels."  Hence
      the bright star coordinates and the crowded box locations are
      expressed in units of binned pixels.

      Prerequisites:      thld - from su_calbgd.c
               ebgx - from su_calbgd.c
               ebgy - from su_calbgd.c
               threshold_iter - counter from su_calbgd.c
               keep_a_log
               log_indx
               prev_frame
               nmaxx (expected to be 1024)
               nmaxy (expected to be 1024)

      Provides:
               nbs - number of bright stars
               bright_stars[] - array with star positions,
                      counts, and second moments
                      (saved in global RAM in order
                      of decreasing brightness)

      Global RAM access:
                  read ~ 5 + (# iterations) *
                  [(nmaxx * nmaxy) + (nominal_x_dim * nominal_y_dim) *
nbs]
                  words from global RAM (this is typically
                  1.3 M words per iteration; number of
                  iterations currently limited to 4)

                  and

               write ~ 1 + nbs * SIZEOF_STAR words to
                  global memory (+ few more if log kept)
                  (this is typically 6.7 k words if
```

```
                   nbs = max_bright_stars = 512)

                DPU cycles ~ 5.6 k per iteration to read +
                        26 to write (=> ~6 seconds
                     required for i/o over bus)

      DSP RAM required:
                ~ 615 external + ~8044 in scanbs() +
                handful in other auxiliary functions
                => total ~8700 words (if MAX_BRIGHT_STARS = 512)
                   (if too many words, decreasing
                   MAX_BRIGHT_STARS will make a big difference)

 FUNCTIONS IN THIS FILE:
      scanbs
      exclusion
      bs_find_brightest_pixel
      buildstar
      summer
      hindx
      find_crowded_boxes

 INCLUDE FILES:
      setup.h

 MODIFICATION HISTORY:
      19May95   ADW   add standard comment header
      09Jun95   ADW   change "printf(" to "fprintf(stderr," for
simulation
                      (timing of log messages was screwy with printf)
      28Oct98   CH    Major update on in-code comments

 RCS $Id: su_scanbs.c,v 1.25 1998/05/28 04:41:45 xmmom Exp $


Include setup.h.  This is for Sun and DPU.
Declare the global variables.
Declare the integer variables.
Declare the "int2" variables. (see our_own_types.h)
Declare the floating point variables.
Declare the structure variables. (see our_own_types.h)
```

```
NAME:
```

scanbs function

```
 PURPOSE:
      This swap unit makes a catalog of bright stars in an image as
follows:
   (1) a reference frame is searched in raster fashion for pixels
      with more than a threshold number of counts and those pixels
      are entered into a buffer;
   (2) when the buffer becomes full, the neighborhood of each bright
      pixel is searched for the single brightest pixel;
   (3) image properties of the bright spot (e.g., counts, centroid
      position, roundness) are derived and recorded in the "bright
      star catalog," bright_stars[];
   (4) finally, an index array is created which can be used to access
      cataloged stars in order of decreasing brightness.  (The indices
      are used to save the bright star catalog in global memory in
      this order.)

 CALLING SEQUENCE:
      swap_to = scanbs();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns 0)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      bs_find_brightest_pixel   [su_scanbs.c (this file)]
      buildstar                 [su_scanbs.c (this file)]
      exclusion                 [su_scanbs.c (this file)]
      find_crowded_regions      [su_scanbs.c (this file)]
      get_time                  [sun_util.c]
      global_zero               [whitedsp.c]
      hindx                     [su_scanbs.c (this file)]
      read_proc_int             [sun_util.c, global_access.c]
      read_sw_int_array         [sun_util.c, global_access.c]
      write_proc_int            [sun_util.c, global_access.c]
      write_proc_int_array      [sun_util.c, global_access.c]

 CALLED FROM:
      sunload                   [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      ??????   CH    original sim code
      18Sep92   DTL   optimized and converted to proto-flight code
      20Oct92   DTL   iteration counter shared, initialized in
su_calbgd.c
      26Oct92   DTL   thld, nmaxx, and nmaxy moved to shared externals
      03Nov92   DTL   fix compilation bugs
                      (variables not declared or defined in previous
version)
      15Dec92   DTL   changed rows_to_skip
      26Jan93   DTL   added file open/close calls for Sun
      29Jan93   DTL   all Points and Twomnts converted between integer
                      representation and floats for global memory I/O
      01Mar93   DTL   include zero_frame() call to clear reference
frame.
```

```
    04Apr93   DTL   change "swap_id" to shared variable "swap_to"
    05Jul93   DCH   Modify execption handling for
                    too many or too few bright stars
    04May95   ADW   add standard comment header
    28Oct98   CH    Major update on in-code comments
```

```
  Declare the integer variables.
  Declare the integer array variables.
  Declare pointers to integers.
  Declare the "int2" arrays. (see our_own_types.h)
  Declare the pointers to "int2" arrays. (see our_own_types.h)
  Declare the structure variables. (see our_own_types.h)
  Declare the functions used.
......................................................................
..
  Obtain necessary data from global memory.
  Set values_obtained equal to the sum of the statuses of ebgx and
ebgy.
  This should be equal to 2 (1 each) if everything is okay.
  If values_obtained is not equal to 2, then (we have a problem...):
     If running in simulation (SUN), print a FATAL ERROR message.
  Set ref_frame equal to prev_frame.
  This sets the address of the reference frame.
  Start looking for bright stars
  When more than one iteration is required,
  "beginning" serves as the goto point.
  If running in simulation, get the time and print a status message.
  If thld is less than 2, then:
  (We have a problem when the threshold level is too low...)
     If running in simulation (SUN), print a FATAL ERROR message.
  Initialize the bright star exclusion boxes.
  Initialize the entire star structure - ADW
  Initialize nbs (number of bright stars counter)
  and bs_ptr, a pointer to the bright star arrays.
  Setup to read a block of pixels from reference frame.
  It doesn’t matter if there’s a fraction of a block left
  Scan image for bright pixels and save their locations in a buffer
  We will return to "find_bright_pixels" after checking the full
buffer.
  Initialize n_bright_pix (number of bright pixels counter)
  and bright_ptr, a pointer to the start of the buffer.
  Cycle through the reference frame, one block at a time.
     Set values_obtained equal to the  number of pixels in the block.
     Look for bright pixels in the block.
     Set pix_ptr equal to pix_counts.
     (Point to the start of the block.)
     For all values of pix less than values_obtained, loop:
        If *pix_ptr++ is greater than thld, then:
        (If the pixel value exceeds the threshold)
           Set *bright_ptr to first_in_block plus pix
           (Point to the block address of pix.)
           Increment n_bright_pix.
           If n_bright_pix is greater than or equal to
           the value of nbrightpixmax, then:
           (If the buffer is full, look for BS images)
     If running in simulation (SUN) and if debugging is enabled,
     then print a status message identifying the block processed.
  Look for brightest pixel in the neighborhood of each bright pixel,
  determine characteristics of the small image surrounding that spot,
  and set up a zone of exclusion around the "star" to avoid looking
  at it again.  Finally, add the star to the bright star catalog.
  Note: it would probably be more efficient to implement this section
  When the bright pixel buffer is filled, we "goto look_for_stars".
```

```
   If running in simulation (SUN), print a status message.
   Set end_of_buffer equal to bright_pix_buffer plus n_bright_pix.
   Loop over each bright pixel found.
      Reconstruct the bright pixel location on detector
      Analyze neighborhood only if not in a zone of exclusion
         Locate bottom left corner (blc)
         of "star_image" and prepare to read
         Read star_image from reference frame
         NOTE: the above will tend to require the bus over several
         Transform bright pixel coordinates to star_image frame
         Determine the boundary within which
         to search for the brightest pixel.
         Find brightest pixel in vicinity of (x_bp, y_bp)
         Analyze the bright star image
         If running in simulation (SUN) and if debugging is enabled
         then print a status message with the star's location.
         Transform coordinates to detector reference frame
         If running in simulation (SUN) and if debugging is enabled
         then print a status message with the transformed location.
         Enter data in bright star "catalog"
         If nbs is less than max_bright_stars, then:
         (If the catalog is not yet full...)
         Otherwise (if the catalog is full)
            In this case there are "too many" bright stars.
            To deal with this, increase the threshold.
   Return for next block if not finished processing frame.
   If we are not done processing the frame
      Add to first_in_block the value of block_size.
      Add to block_address the size of block_size.
      Go to find_bright_pixels to find bright pixels
      in the next block of the current frame.
   If running in simulation (SUN),
   print a message stating the number of stars found.
   If the frame processing has been completed, there is no harm
   in proceeding to the "adjust_threshold" section of the code.
   It will be skipped unless nbs is less than the minimum required
   number of bright stars.  In that case we need to decrease the
   value of thld below.  The case of nbs too large is handled by
   a "goto adjust_threshold" above.
   Adjust the threshold and try again if too many bright stars were
found.
   "adjust_threshold" is the goto point for threshold adjustment.
   If too many bright stars were found...
      Set thld to 3/2 its present value. (INCREASE by 50%)
      If running in simulation (SUN), print a warning message
      indicating the increase in threshold value.
      If threshold_iter exceeds max_scanbs_iterations, then:
      (This is a software limit to the maximum number
         If running in simulation (SUN), get the time and
         print a FATAL ERROR message.
         clear bit to indicate availability of window data
      If we have not reached the threshold adjustment limit
         try again on current frame with new threshold.
   If too few bright stars were found
   [It seems the way to get here is to finish processing the
      Set thld to 4/5 of its current value. (DECREASE by 20%)
      If running in simulation (SUN), print a warning message
      indicating the decrease in threshold value.
      If debugging is enabled, print some diagnostic information.
      If the threshold has been set too low
      If threshold_iter exceeds max_scanbs_iterations, then:
      (This is a software limit to the maximum number
         clear bit to indicate availability of window data
      If we have not reached the threshold adjustment limit
         try again on current frame with new threshold.
   We don't need the "goto sort_and_save" statements above because
```

```
   control comes here naturally when the catalog is ready.
   Sort bright star catalog in order of decreasing brightness
   and save in global RAM

   Initialize the bs_count_ptr and bs_prt pointers.
   Loop over the bright stars found.
      Increment the bs pointers.
   Call hindx to sort the bright stars in order of decreasing
brightness.
   Set base address to the BRIGHT_STARS portion of memory.
   Note:  xsize = 1; and yskip = 0; (initialized above)
   For each bright star memory location,
      Write the bright star data into memory.
      Add to current memory address the value of SIZEOF_STAR.
   Write number of bright stars into NBS memory location.
   If task_id equals CHOOSE_GUIDE_STARS, then:
      Set n_crowded to the number of crowded fields
      by calling find_crowded_regions.
      Write the number of crowded fields into global memory.
      If there are crowded fields
         Write the crowded field data into global memory.
   We're done, so reinitialize reference frame to all zeros.
   If running in simulation (SUN), print a status message.
   Set izero to the return status of the global_zero routine,
   which here resets the reference frame to all zeros.
   Return to calling routine with value 0.
   If we have failed to identify an acceptable number of stars
   we must reset the memory area occupied by the reference frame
   and several of the PROC memory locations so that subsequent
   operations will proceed properly.
   There are four failure modes that lead to this point.
```

```
NAME:
```

exclusion function

```
 PURPOSE:
      This function checks to see if the pixel (i,j) is in any of an
array
      of rectangular exclusion zones.  If so, then value returned is
      (boxid + 1); otherwise 0.

 CALLING SEQUENCE:
      exclusion (i, j, n_stars, star_ptr);

 INPUTS:
      i        = x position of pixel to test
      j        = y position of pixel to test
      n_stars  = current number of stars
      star_ptr = pointer to list of stars

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns 0       if i,j is not in an exclusion zone;
              returns boxid+1 if i,j is in an exclusion zone.)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_proc_int   [sun_util.c; global_access.c]

 CALLED FROM:
      scanbs           [su_scanbs.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      05May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments

   Declare the integer variables.

   Set the loop index k equal to 0 (initialize it).
   If keep_a_log is set, make a log entry.
   Look for (i,j) in existing boxes.
   Check every input star position.
      If (i,j) is within the limits of the box
      specified by star_ptr (i.e., box k), then
         Return box id + 1
      Increment the box id pointer.
   If we get here, (i,j) is not in an exclusion box.
   In that case, Return to the calling routine with value 0.
```

```
NAME:
```

bs_find_brightest_pixel function

```
 PURPOSE:
      This function finds the brightest pixel in the rectangle bounded
      by left_edge, right_edge, bottom_edge, top_edge.
      The pixel coordinates are returned in the pointers ixp and iyp.

 CALLING SEQUENCE:
      bs_find_brightest_pixel (ixp, iyp, left_edge, right_edge,
                                          bottom_edge, top_edge);

 INPUTS:
      ixp         = pointer to pixel x coord
      iyp         = pointer to pixel y coord
      left_edge   = min x search boundary
      right_edge  = max x search boundary
      bottom_edge = min y search boundary
      top_edge    = max y search boundary

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      ixp         pointer to pixel x coord
      iyp         pointer to pixel y coord

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_proc_int   [sun_util.c; global_access.c]

 CALLED FROM:
      scanbs           [su_scanbs.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      05May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments


   Declare the integer variables.
   Save the initial coordinates for DEBUG printout
   Initialize max_counts to counts in the starting pixel
   Scan subset of star_image to find position with maximum counts.
   Loop over columns.
      Loop over rows.
         If counts in (i,j) are more than than max_counts, then
            Set (ixp, iyp) to (i, j) and
            set max_counts equal to counts.
   If running in simulation (SUN) and if debugging is enabled,
   print a message with information on the brightest pixel found.
```

```
NAME:

      buildstar          function

 PURPOSE:
      This function integrates over a stellar image and extracts total
      counts, centroid, size, and second moments.  Summation is carried
      out separately over each of four quadrants using function
``summer.''

 CALLING SEQUENCE:
      buildstar

 INPUTS:
      x_bp         x pos'n of bright pixel
      y_bp         y pos'n of bright pixel

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns Star structure star)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      summer          [su_scanbs.c (this file)]
      write_proc_int  [sun_util.c, global_access.c]

      return_bus      [sun_util.c]  *** currently commented ***

 CALLED FROM:
      sunload         [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      05May95  ADW  add standard comment header

 RCS $Id: su_scanbs.c,v 1.25 1998/05/28 04:41:45 xmmom Exp $


   Declare the integer variables.
   Declare the structure variables.  (see our_own_types.h)
   Declare the functions called.
.........................................................................
..
   If keep_a_log is enabled, make a log entry.



   Calculate the sum of counts in the specified region.
   Initialize the accumulators
   Set the integration boundaries
   Set increment factors
   Sum over first quadrant
   Set the integration x boundaries
   Set increment factor
   Sum over second quadrant
   Set integration y boundaries
   Set increment factor
   Sum over third quadrant
   Set integration boundaries
   Set increment factor
   Sum over fourth quadrant
   Calculate image centroid position and second moments.
```

```
   If cts is greater than 0, then
   (We expect star counts > 0...)
      If USE_FLOATS equals 1 (currently activated), then
      Normalize second moments
      Transform centroid to subimage reference frame
      Else use extended integers.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !! NOTE: extended integer version needs to be tested !!
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      Calculate centroid position
      Normalize second moments
      Else (if cts >= 10000)
      If "star" has no counts, put its centroid at a silly place
   Enter integrated counts and integration boundaries in star structure
   Return structure to calling routine
```

```
NAME:

      summer             function

 PURPOSE:
      This function sums over pixels in a quadrant bounded by int_xmin,
      int_xmax, int_ymin, and int_ymax in a direction controlled by dx
and dy.
      (The values of dx and dy are, respectively, (1, 1) in quadrant I;
      (-1, 1) in quadrant II; (-1, -1) in quadrant III; and (1, -1) in
      quadrant IV.)
      Summation is terminated either when the integration range
encompasses
      3-sigma for the psf (n3sig = 500), or when the local count level
drops
      to the background level (ebgx or ebgy) estimated in su_calbgd().
The
      value returned is the number of counts in the quadrant.

 CALLING SEQUENCE:
      summer(x_bp, y_bp)

 INPUTS:
      x_bp
      y_bp

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns counts_in_quadrant)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_proc_int   [sun_util.c; global_access.c]

      return_bus       [sun_util.c]  *** currently commented ***

 CALLED FROM:
      sunload          [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      05May95  ADW  add standard comment header


Scan quadrant in raster fashion over the star image.
   Initialize counts_in_quadrant to 0.
   If there is more than one pixel in either dimension, then
   (Otherwise there is nothing to accumulate...)
      Set the x pixel counter to int_xmin.
      While x is not equal to int_xmax, loop
      (This starts a loop over included columns.)
         Set the x pixel counter to int_xmin.
         Capture the current column into image_column.
         While y is not equal to int_ymax, loop
         (This starts a loop over included rows.)
            Accumulate sums for moments.
            Terminate column sum at "edge of image".
            Increment y position variables.
         Update exclusion zone y range.
         Accumulate sum of counts in image.
         Terminate row sum at "edge of image".
         Increment x.
```

```
      Update exclusion zone x range.
   If running in simulation (SUN), then
   give data for diagnosis if there are no counts.
   Return to calling routine with total number of counts in quadrant
```

```
NAME:

      hindx            function

 PURPOSE:
      This function employs the Heapsort algorithm (see Numerical
Recipes
      in C by Press et al., 1988) to determine the indices of array a[
],
      of size n, according to which the array elements are sorted in
      DESCENDING order.  The resulting index array, indx[ ], is also of
      size n.  a[ ] remains in its original order.

 CALLING SEQUENCE:
      hindx (n, a, indx)

 INPUTS:
      n         size of array to be sorted
      a         array to be sorted

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      indx         index into array a

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_proc_int   [sun_util.c; global_access.c]

 CALLED FROM:
      scanbs           [su_scanbs.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      05May95  ADW  add standard comment header
      28Oct98   CH    Major update on in-code comments


   Declare the integer variables.

   Initialize indices and counters.
   For(ever), then
   (At the appropriate time, "return" will be encountered.)
      Sift through heap, dropping element in appropriate place.
```

```
NAME:
```

find_crowded_regions function

```
 PURPOSE:
      This function seeks square boxes of dimension "crowded_box_dim" in
      which the density of bright stars is high (number of stars in box
>
      threshold value "stars_in_crowd") for future use (in su_scangs) as
zones
      of exclusion when guide stars are selected.  The idea is to avoid
      choosing guide stars in crowded fields such as globular clusters.

 CALLING SEQUENCE:
      find_crowded_regions (beginning_of_catalog, nbs);

 INPUTS:
      beginning_of_catalog   pointer to bright star catalog
      nbs                    number of bright stars

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns n_crowded_boxes)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      scanbs         [su_scanbs.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      06Aug92   CH    original code developed
      05May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments


   Declare the integer variables.
   Declare the structure variables.  (see our_own_types.h)

   Initialize pointers and count of crowded boxes.
   Divide detector field of view within border into a grid
   of boxes and check each box for high star density
         Initialize stars_in_box accumulation variable.
         for each entry in the bright star catalog, loop
            Convert centroid coordinates from integer
            to floatinf point representation.
               If the centroid is within the current box, then
                  increment stars_in_box.
         If stars_in_box is greater than stars_in_crowd, then
            Save crowded box data.
         Reset ymin for next set of y boxes
      Reset xmin for next set of x boxes
   Return to calling routine with number of crowded boxes.
```

NAME:

get_r0

```
PURPOSE:
     This is a debugging function to capture the R0 register

CALLING SEQUENCE:
     get_r0();

INPUTS:
     None.

OPTIONAL INPUT PARAMETERS:

OUTPUTS:
     Value placed in global variable chipmunk. Not used.

INCLUDE FILES:

SUBROUTINES CALLED:
     None.

CALLED FROM:
     Currently not used.

RESTRICTIONS:

MODIFICATION HISTORY:
     15Jul96   ADW   add standard comment header
     28Oct98   CH    Major update on in-code comments


  Disable interrupts, grabs r0 and enable interrupts
```

```
NAME:

      su_scangs.c            (Swap unit)

 PURPOSE:
      This swap unit selects from a catalog of bright "stars" a set of
      guide stars to be used for tracking and a second set of objects
      thought not to be useful for this purpose because they are either
      not star-like, or they are too bright or too faint, or they lie
too
      close to the edge of the field of view, in a crowded field, or too
      close to a star already selected.

      Used in tasks:      CHOOSE_GUIDE_STARS

      Prerequisites:
               bright_stars[] - sorted catalog of bright
               stars provided by su_scanbs
               nbs - number of bright stars
               crowded_boxes - regions with high density of
               bright stars (from su_scanbs)
               nmaxx - x dimension of reference frame
               nmaxy - y dimension of reference frame
               keep_a_log - yes or no?
               log_indx - address of current place in log

      Provides:
               current_coords[] - locations of guide stars
                  in the current (reference) frame
               ref_star_quality[] - quality of guide stars
               ref_star_weight[] - counts in guide stars
                  in the reference frame
               stars_found - number of guide stars
               rejects[] - coords of "bright stars" that
                  do not qualify as guide stars
               reject_quality[] - quality of rejected stars
               reject_weight[] - ref frame counts in rejects

      Global RAM access:
               (+ 1 more cycle if log kept)

      DSP RAM required:

 FUNCTIONS IN THIS FILE:
      scangs
      exclude_border
      exclude_crowded_fields
      is_it_good
      in_exclusion_zone
      add_exclusion_box

 INCLUDE FILES:
      setup.h

 MODIFICATION HISTORY:
      19May95   ADW   add standard comment header
      09Jun95   ADW   change "printf(" to "fprintf(stderr," for
simulation
                      (timing of log messages was screwy with printf)
      28Oct98   CH    Major update on in-code comments

 RCS $Id: su_scangs.c,v 1.24 1998/05/28 04:41:48 xmmom Exp $


Include setup.h.  This is for Sun and DPU.
```

```
Declare the global integer variables.
Declare the global structure variables.  (see our_own_types.h)
```

```
NAME:
      scangs              function

 PURPOSE:
      This swap unit selects from a catalog of bright "stars" a set of
      guide stars to be used for tracking and a second set of objects
      thought not to be useful for this purpose because they are either
      not star-like, or they are too bright or too faint, or they lie
      too close to the edge of the field of view, in a crowded field, or
      too close to a star already selected. Two sets of guide star
      selection parameters are defined, one for normal observing (i.e.
      U,B,V, UVW1, UVW2 and UVM2 filters) and one for the magnifier (due
      to the differences in the PSF). These parameters are defined in
      white_default

 CALLING SEQUENCE:
      swap_to = scangs();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns SWAP_SCANBS   after threshold adjustment,
                      SWAP_DPUABORT after too many adjustments,
                      0             for successful completion)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      add_exclusion_box       [su_scangs.c (this file)]
      exclude_border          [su_scangs.c (this file)]
      exclude_crowded_fields  [su_scangs.c (this file)]
      get_time                [sun_util.c]
      is_it_good              [su_scangs.c (this file)]
      q_alert                 [sun_util.c, cwhite.c]
      read_proc_int           [sun_util.c, global_access.c]
      read_proc_int_array     [sun_util.c, global_access.c]
      write_proc_int          [sun_util.c, global_access.c]
      write_proc_int_array    [sun_util.c, global_access.c]

 CALLED FROM:
      sunload                 [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      ?????9?   CH    original code
      23Oct92   DTL   converted to proto-flight code and optimized to
DPU
      04Dec92   DTL   bright star scale factors set in
                      su_restore_stars do not need to be set here.
      08Dec92   DTL   option to save or not save guide star
                      and reject data to global RAM
      14Dec92   DTL   guide star coords saved as
                      current_coords, not nominal_coords
      07Jan93   DTL   MAX_QUALITY_PARAMETERS introduced to
                      specify dimension of parameters[]
      29Jan93   DTL   integer coord and moment representation
                      introduced for current_coords[],
                      rejects[], star coords in is_it_good()
                      and bright star coords.  Also, quality
                      params changed to floats as needed.
      01Feb93   DTL   added file open/close calls for Sun
      12Feb93   DTL   quality parameters #defined;
                      get_quality_parameters() eliminated.
```

```
   11Mar93   DTL   added capability to queue reference frame data
   04Apr93   DTL   change "swap_id" to shared variable "swap_to"
   08May95   ADW   add standard comment header, structured English
                   comments
   28Oct98   CH    Major update on in-code comments
   02Feb00   JAK   Added in magnifier tracking CGS parameters

Declare the local integer variables.
Declare the local integer arrays.
Declare local pointers to integers.
Declare the local structure variables.  (see our_own_types.h)
Declare the functions called.

Initialize the x and y pixel binning factors to 2.
Call exclude_border to set up the initial guide star
exclusion zone to exclude the border around the detector.
Read the number of excluded fields from memory.
If n_crowded (is greater than 0), then
(If there are any crowded fields...)
   Call exclude_crowded_fields to mask out the crowded fields.
Read the number of stars in the bright star catalog.
Initialize the array access parameters.
Initialize pointers needed.
Select which guide star selection parameters we use
     If this exposure uses the magnifier use relevant parameters
     Otherwise use the normal parameters
Initialize counters needed.
Scan the BSC until enough accepted and rejected stars are identified.
   Set stars_to_read to the smaller of stars_per_bus_access and
   the number of stars left to be examined in the catalog.
   Read bright star information from global memory.
   Note that stars are already sorted in order of
   decreasing brightness.
   Set last_star to the location of the last star in the
   current batch, i.e., to bright_stars + stars_to_read.
   Set bs_index to the number of stars already examined.
   For values of bs_prt from bright_stars to last_star−1, loop
   (Loop through the current batch of bright stars.)
      Convert data representation of coordinates
      from integer to floating point.
      Call is_it_good to evaluate the current
      star’s usefulness as a guide star.
      If the current star would be a good guide star, then
         If we can still accept more guide stars
            Save the current star’s information
            in the next good_* locations.
            Increment the n_accepted counter.
            call add_exclusion_box to add an exclusion
            box around the star.  This keeps the guide
            stars well separated.
      Else (the star would not be good as a guide star)
         If there are not yet too many rejected stars
            Save the current star’s information
            in the next reject_* locations.
            Increment the n_rejected counter.
      Increment the bs_index pointer.
   Add stars_to_read to stars_examined.
   (Keep a tally of all stars checked.)
Set n_recorded equal to n_accepted plus n_rejected.
If running in simulation (SUN) and debugging is enabled,
then print some useful diagnostic information.
If too few guide stars were found, GIVE UP!!!
   If we the bright star threshold has already been
   adjusted the maximum number of times, then
   If running in simulation, get the time and print a status message.
      Set swap_to to SWAP_DPUABORT to specify next swap unit.
```

```
      Return value swap_to to the calling routine.
   Otherwise reduce threshold and look for more bright stars.
      Reduce thld to 80% of its current value.
      Set swap_to to SWAP_SCANBS to specify next swap unit.
      Return value swap_to to the calling routine.
Set stars_found equal to n_accepted, the number of good guide stars.
If SAVE_GS_COORDS equals 1, then
Save guide star data in global memory.
Convert coordinates to integer representation
Set stars_rejected equal to n_rejected, the number of
bright stars that are not good enough to be guide stars.
If save_reject_data (is greater than 0), then
Save rejected star data in global memory (optional)
   Convert coordinates to integer representation
Return to calling routine with status 0.
```

```
NAME:
```

exclude_border function

```
 PURPOSE:
    This function initializes a zone of exclusion from which guide stars
    should not be selected.  The exclusion zone is the union of a set of
    rectangular "boxes."  Four boxes are chosen which mask out a border
    region of width "border_width" around the edge of the detector.

 CALLING SEQUENCE:
      exclude_border();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_proc_int      [sun_util.c; global_access.c]

 CALLED FROM:
      scangs              [su_scangs.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      01Feb91   CH    original code developed
      09May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments



    Declare the local integer variables.
    Declare the local structure variables.  (see our_own_types.h)

    Set x_edge and y_edge to the detector limits.
    Set the first exclusion box as the left edge of the detector.
    Set the second exclusion box as the bottom edge of the detector.
    (Note that exclusion boxes are allowed to overlap.)
    Set the third exclusion box as the right edge of the detector.
    Set the fourth exclusion box as the top edge of the detector.
    Set the number of exclusion boxes to 4, the current value.
    If running in simulation (SUN), print a diagnostic message, and
       If debugging is enabled, print some more diagnostic information.
```

NAME:

exclude_crowded_fields function

```
 PURPOSE:
      This function adds crowded areas to the list of exclusion boxes.
The
      crowded regions were identified in su_scanbs and are now simply
read
      back from global memory.

 CALLING SEQUENCE:
      exclude_crowded_fields();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      read_proc_int_array  [sun_util.c; global_access.c]
      write_proc_int       [sun_util.c; global_access.c]

 CALLED FROM:
      scangs               [su_scangs.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      08May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments

   Declare the local integer variables.
   Declare the local structure variables. (see our_own_types.h)

   If n_crowded is greater than max_excl_boxes minus n_excl_boxes, then
   (We have to take the non-crowded exclusion boxes into account here
   when limiting the number of crowded boxes.)
      Set n_crowded equal to max_excl_boxes minus n_excl_boxes.
      If running in simulation (SUN) and if debugging is enabled,
      then print a WARNING message - that ther are too many crowded
boxes.
   Set the excl_box_ptr to point at the first vacant exclusion box.
   Read the crowded box data from global memory.
   Transform box coordinates to 2048 x 2048 detector pixel scale
   For values of i from 0 to n_crowded-1, loop (over exclusion boxes)
   If running in simulation (SUN), print a diagnostic message and
      IF debugging is enabled, print some more diagnostic information.
   Add n_crowded to the current value of n_excl_boxes.
   (i.e., get the new correct total of exclusion boxes)
```

```
NAME:
```

is_it_good function

```
 PURPOSE:
      This function sets bits in a quality word to indicate which of 9
      criteria are satisfied by a star image.  The criteria collectively
      measure how good a candidate the star is for tracking.
      If a criterion is not satisfied the corresponding bit is set to 1;
      0 means the test was passed.

 CALLING SEQUENCE:
      integer_variable = is_it_good(star);

 INPUTS:
      star         pointer to the star on trial

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns "quality" - see purpose)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_proc_int      [sun_util.c; global_access.c]
      expand              [macro.h]
      shrink              [macro.h]
      round               [macro.h]
      in_exclusion_zone   [su_scangs.c (this file)]

 CALLED FROM:
      scangs              [su_scangs.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      09May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments


   Declare the local integer variables.
   Declare the local structure variables. (see our_own_types.h)
   Declare the functions called.

   Initialize the quality word (star "innocent" until proven guilty).
   Get the substructures from the star structure
   Convert coordinates and moments from integer representation to floats
   *** QUALITY BIT #1 *** centroid off-center; parameter max_off_center
   If running in simulation (SUN) and DEBUGging is enabled,
   then print a status message about center coordinates.
      (centroid.x - middle_x), (centroid.y - middle_y));
   If the coordinates are too far off, then
      Set quality bit 1 to 1.
   *** QUALITY BIT #2 *** non-circular shape; parameter max_noncirc
   If running in simulation (SUN) and DEBUGging is enabled,
   then print a status message about non-circularity.
      "Non-circular? width = %d, height = %d\n", width, height);
   If our non-circularity tolerance is not met, then
      Set quality bit 2 to 1.
   *** QUALITY BIT #3 *** star image too big; parameter max_image_size
   If running in simulation (SUN) and DEBUGging is enabled,
   then print a status message about image size.
      "Image too big? Width and height compared with %d pixels\n",
      max_image_size);
```

```
   If image width or height exceeds (or equals) max_image_size, then
      Set quality bit 3 to 1.
   *** QUALITY BIT #4 *** sq moments too big; parameter max_sq_moment
   If running in simulation (SUN) and DEBUGging is enabled,
   then print a status message about square moments.
      moment.x2mnt, moment.y2mnt);
   If either x2mnt or y2mnt moment exceeds (or equals) max_sq_moment,
then
      Set quality bit 4 to 1.
   *** QUALITY BIT #5 *** unequal sq moments; parameter max_moment_diff
   If running in simulation (SUN) and DEBUGging is enabled,
   then print a status message about square moment inequality.
   If our moment inequality tolerance is not met, then
      Set quality bit 5 to 1.
   *** QUALITY BIT #6 *** xy moment too big; parameter max_xy_moment
   If running in simulation (SUN) and DEBUGging is enabled,
   then print a status message about xy moment magnitude.
   If moment.xymnt is not less than max_xy_moment, then
      Set quality bit 6 to 1.
   *** QUALITY BIT #7 *** NOT USED
   *** QUALITY BIT #8 *** centroid in exclusion zone?
   Call in_exclusion_zone to see whether the star is in an exclusion
zone.
   If the star is in an exclusion zone, then
      If running in simulation (SUN) and DEBUGging is enabled,
      then notify us that star is in an exclusion zone.
      Set quality bit 8 to 1.
   *** QUALITY BIT #9 *** star too bright; parameter
saturation_threshold
   If running in simulation (SUN) and DEBUGging is enabled,
   then print a message with star brightness information.
      star->count, saturation_threshold);
   If star->count exceeds the saturation_threshold, then
      Set quality bit 9 to 1.
   *** QUALITY BIT #10 *** star too faint; parameter min_counts
   If running in simulation (SUN) and DEBUGging is enabled,
   then print a message with star brightness information.
      star->count, min_counts);
   If star->count is less than min_counts, then
      Set quality bit 10 to 1.
   If running in simulation (SUN) and DEBUGging is enabled,
   then print a message stating the star’s quality value.
   Return to the calling routine with the quality value.
```

```
NAME:
```

in_exclusion_zone function

```
 PURPOSE:
      This function checks to see if the pixel (x,y) is in any of an
array
      of rectangular exclusion zones.  If so, then value returned is the
      box number (0th box has box number 1); otherwise 0.

 CALLING SEQUENCE:
      integer_variable = in_exclusion_zone(x,y);

 INPUTS:
      x         x position of pixel to test
      y         y position of pixel to test

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns 0      if (x,y) is not in an exclusion zone,
                      box_id if (x,y) is in an exclusion zone)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_proc_int       [sun_util.c; global_access.c]

 CALLED FROM:
      is_it_good           [su_scangs.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      09May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments


   Declare the local integer variables.
   Declare the local structure variables.

   Look for (x,y) in existing exclusion boxes
   and return box_id if (x,y) in box.
   Loop over the boxes in the current batch.
   Loop over all exclusion boxes.
      Increment box_id.  NOTE: 0th box has box_id=1
      If (x,y) is within current box limits, then
         Return box_id to calling routine.
   If (x,y) is not in an exclusion box,
   return to calling routine with value 0.
```

NAME:

add_exclusion_box function

```
 PURPOSE:
      This function establishes an exclusion zone around a guide star,
      thus preventing selection of a nearby bright star as another guide
      star.  The dimensions of the exclusion zone are twice the
      box_halfwidth plus the psf integration range.  The new zone is
added
      to the list of exclusion boxes. Nothing happens (i.e., no new box
is
      created) if the list is full.

 CALLING SEQUENCE:
      add_exclusion_box(star);

 INPUTS:
      star         newly selected guide star

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_proc_int      [sun_util.c; global_access.c]

 CALLED FROM:
      scangs              [su_scangs.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      10May95   ADW   add standard comment header
      28Oct98   CH    Major update on in-code comments


   Declare the local integer variables.
   Declare the local structure variables.

   If we haven't already specified the maximum
   number of exclusion boxes
      Define the new exclusion box.
      Use the star box plus a buffer of box_halfwidth on each side.
      Put the newbox information into exclusion_boxes[i]
      where i is the new n_excl_boxes counter value.
   If running in simulation and if DEBUGgging is enabled,
   then print a WARNING message (no room for more boxes).
```

```
NAME:

      su_submit_cmprss_q.c      (Swap unit)

 PURPOSE:
      This swap unit manages the timing of DPU operations, selects the
      task requested by the ICU, and queues data for delivery in the
      telemetry stream.

 FUNCTIONS IN THIS FILE:
      submit_cmprss_q
      submit_bluefast_data
      submit_image_data
      submit_wdw_data
      submit_trk_data
      submit_ref_data
      submit_trh_data
      submit_eng_data

 INCLUDE FILES:
      setup.h

 MODIFICATION HISTORY:
      18May95   ADW   add standard comment header
      12Mar96   ADW   add engineering mode

 RCS $Id: su_submit_cmprss_q.c,v 1.4 1998/05/28 04:41:50 xmmom Exp $
```

```
NAME:

      submit_cmprss_q           function

 PURPOSE:
      This swap unit manages the timing of DPU operations, selects the
      task requested by the ICU, and queues data for delivery in the
      telemetry stream.

 CALLING SEQUENCE:
      swap_to = submit_cmprss_q();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none    (returns 0 if ok, 1 if bad data_delivery specified)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      submit_bluefast_data  [su_submitdata.c (this file)]
      submit_image_data     [su_submitdata.c (this file)]
      submit_ref_data       [su_submitdata.c (this file)]
      submit_trh_data       [su_submitdata.c (this file)]
      submit_wdw_data       [su_submitdata.c (this file)]
      submit_eng_data       [su_submitdata.c (this file)]

 CALLED FROM:
      sunload                [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      30Sep93   CH    start from scratch
      08May95   ADW   add standard comment header
      12Mar96   ADW   add engineering mode support


If running in simulation, declare integer variable contin.
   Initialize array access parameters xsize and yskip.
   Call appropriate function based on data type to be submited.
   Return with value 0 for every valid data type, else 1.
If we get here, there is a bogus data type specified.
In this case, print a message to that effect if we're
running in simulation (SUN).
      Return to calling routine with value 1.
```

```
NAME:

      submit_bluefast_data

 PURPOSE:
      This function queues science data for delivery in the telemetry
stream.

 CALLING SEQUENCE:
      submit_bluefast_data();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      data_queue     [sun_util.c, cwhite.c]
      read_proc_int  [sun_util.c, global_access.c]
      q_alert        [sun_util.c, cwhite.c]

 CALLED FROM:
      submit_cmprss_q    [su_submit_cmprss_q.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      10May95   ADW   add standard comment header


   If n_bfast is greater than 0, then
      Send out the first one
      If n_bfast equals 2, then
         Send out the second one
```

```
NAME:

      submit_image_data

 PURPOSE:
      This function queues image data for delivery in the telemetry
stream.

 CALLING SEQUENCE:
      submit_image_data();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      data_queue            [sun_util.c, cwhite.c]
      read_proc_int         [sun_util.c, global_access.c]
      read_proc_int_array   [sun_util.c, global_access.c]

 CALLED FROM:
      submit_cmprss_q           [su_submit_cmprss_q.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      10May95   ADW   add standard comment header



   Declare the local integer variables.
   Declare the local structure variables.
.......................................................................
..
   Read the science window configuration data from global RAM.
   Initialize scw_id counter to 0.
   For values of scw from science_windows to
science_windows+n_valid_scw-1
   loop (through the current crop of science windows).
      Increment scw_id.
```

```
NAME:

      submit_ref_data

 PURPOSE:
      This function queues reference frame for delivery
      in the telemetry stream.

 CALLING SEQUENCE:
      submit_ref_data();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      data_queue           [sun_util.c, cwhite.c]
      read_proc_int        [sun_util.c, global_access.c]
      read_proc_int_array  [sun_util.c, global_access.c]
      write_bw_int         [sun_util.c, global_access.c]
      write_bw_int_array   [sun_util.c, global_access.c]

 CALLED FROM:
      submit_cmprss_q          [su_submit_cmprss_q.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      10May95   ADW   add standard comment header

   Declare pointers to integers.
   Declare structure variables.  (see our_own_types.h)
   Set base and initialize offset for reference frame data
   Construct a header for reference frame data.
   Gather data on good guide stars
   Gather data on rejected "stars"
```

NAME:

submit_trh_data

```
PURPOSE:
     This function submit tracking history data for
     compression queue;

CALLING SEQUENCE:
     submit_trh_data();

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
     write_proc_int_array

CALLED FROM:
     submit_cmprss_q     [su_submit_cmprss_q.c (this file)]

RESTRICTIONS:

MODIFICATION HISTORY:
     10May95   ADW   add standard comment header
```

```
NAME:

      submit_eng_data

 PURPOSE:
      This function queues image data for delivery in the telemetry
stream.

 CALLING SEQUENCE:
      submit_eng_data();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      data_queue           [sun_util.c, cwhite.c]
      read_proc_int        [sun_util.c, global_access.c]

 CALLED FROM:
      submit_cmprss_q          [su_submit_cmprss_q.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      15Dec95   ADW   new


   Declare the local integer variables.
   The data_length assignment could be generalized to something like
   data_length = dtw.xsize * dtw.ysize;
```

```
NAME:

      sun_util.c

 PURPOSE:
      The set of DPU handles intended to run
      on the SUN version of the DPU emulator code.

 FUNCTIONS IN THIS FILE:
      open_global_ram_proc
      open_global_ram_big
      open_global_ram_small
      close_global_ram_proc
      close_global_ram_big
      close_global_ram_small
      return_bus
      get_time
      zero_frame
      data_queue
      priority_queue
      data_queue_stat  *** call commented ***
      q_alert
      sci_command
      debuggingermost
      main

   The following are compiled only when GLOBAL_ACCESS_C = 1
      read_int
      write_int
      read_int_array
      write_int_array

   The following are compiled only when GLOBAL_ACCESS_C != 1
      read_proc_int
      read_proc_int_array
      write_proc_int
      write_proc_int_array
      read_bw_int
      read_bw_int_array
      write_bw_int
      write_bw_int_array
      read_sw_int
      read_sw_int_array
      write_sw_int
      write_sw_int_array

 INCLUDE FILES:
      stdio.h
      sim_blue_local_ram.h
      global_access.c       (when GLOBAL_ACCESS is defined with value 1)

 MODIFICATION HISTORY:
      14Aug92   JMK
      22May95   ADW   add standard comment header
      09Jun95   ADW   printf( -> fprintf(stderr,

 RCS $Id: sun_util.c,v 1.27 1998/09/23 22:29:31 ho Exp $

NOTE: in the DPU, PROC_SIZE and BIG_SIZE are 3-byte words (0x00FFFFFF)
```

```
NAME:

      open_global_ram_proc

 PURPOSE:

 CALLING SEQUENCE:
      open_global_ram_proc(filename);

 INPUTS:
      filename   = name of file to open

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns TRUE  if input filename exists,
                      FALSE if it does not exist.)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      main         [sun_util.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      22May95   ADW   add standard comment header
```

```
NAME:

      open_global_ram_big

 PURPOSE:

 CALLING SEQUENCE:
      open_global_ram_big(filename);

 INPUTS:
      filename   = name of file to open

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns TRUE if input filename exists,
                      FALSE if it does not exist.)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      main         [sun_util.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      22May95   ADW   add standard comment header
```

```
NAME:

      open_global_ram_small

 PURPOSE:

 CALLING SEQUENCE:
      open_global_ram_small(filename);

 INPUTS:
      filename   = name of file to open

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns TRUE if input filename exists,
                      FALSE if it does not exist.)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      main         [sun_util.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      22May95   ADW   add standard comment header
```

```
NAME:

      close_global_ram_proc

 PURPOSE:

 CALLING SEQUENCE:
      close_global_ram_proc();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns TRUE if all is well,
                      FALSE if a problem exists.)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      main         [sun_util.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      22May95   ADW   add standard comment header
```

```
NAME:

      close_global_ram_big

 PURPOSE:

 CALLING SEQUENCE:
      close_global_ram_big();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns TRUE if all is well,
                      FALSE if a problem exists.)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      main         [sun_util.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      22May95   ADW   add standard comment header
```

```
NAME:

      close_global_ram_small

 PURPOSE:

 CALLING SEQUENCE:
      close_global_ram_small();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns TRUE if all is well,
                      FALSE if a problem exists.)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      main         [sun_util.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      22May95   ADW   add standard comment header
```

```
NAME:

      return_bus

 PURPOSE:

 CALLING SEQUENCE:
      integer_variable = return_bus();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns TRUE)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      just about everywhere, except that
      *** all calls to this routine are currently commented out ***

 RESTRICTIONS:

 MODIFICATION HISTORY:
      22May95   ADW   add standard comment header
```

```
NAME:

      get_time

 PURPOSE:

 CALLING SEQUENCE:
      integer_variable = get_time();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none (returns the "time")

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      scanbs         [su_scanbs.c]
      scangs         [su_scangs.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      22May95   ADW   add standard comment header
```

```
NAME:

      zero_frame

 PURPOSE:

 CALLING SEQUENCE:
      n = zero_frame(base, length);

 INPUTS:
      base
      length

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      data_queue   [sun_util.c (this file)]
      global_zero  [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header
```

```
NAME:

      data_queue

 PURPOSE:

 CALLING SEQUENCE:
      data_queue(base, length, type, hdr_base, hdr_length, ping_pong,
padding);

 INPUTS:
      base
      length
      type
      hdr_base
      hdr_length
      ping_pong      *** not used in this function ***
      padding

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      q_alert        [sun_util.c (this file)]
      zero_frame     [sun_util.c (this file)]

 CALLED FROM:
      deliver_bluefast_data   [su_deliverdata.c]
      deliver_image_data      [su_deliverdata.c]
      deliver_ref_data        [su_deliverdata.c]
      deliver_trh_data        [su_deliverdata.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      26Sep93   CH    Rewrite, do away with original file-based
simulation.
                      Use RAM-based global Ram scheme.
                      Enforce only one data queueing storage file.
      22May95   ADW   add standard comment header
```

```
NAME:

      priority_queue

 PURPOSE:
      This is the priority queue for medium size (<= 512 words) urgent
data.

 CALLING SEQUENCE:
      integer_variable = priority_queue(type,length);

 INPUTS:
      type
      length

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none (returns 0)

 INCLUDE FILES:
      setup.h

 SUBROUTINES CALLED:

 CALLED FROM:
      deliver_faq_data   [su_deliverdata.c]
      deliver_wdw_data   [su_deliverdata.c]
      deliver_trk_data   [su_deliverdata.c]
      deliver_bfast      [su_deliverdata.c]
      send_dp_data       [reddsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      06Oct93   CH    Start from scratch.
      22May95   ADW   add standard comment header
```

```
NAME:

      data_queue_stat

 PURPOSE:

 CALLING SEQUENCE:
      n = data_queue_stat(ping_pong);

 INPUTS:
      ping_pong      *** not used in this function ***

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      track_guide_stars_task   [whitedsp.c]  *** call commented ***

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header
```

```
NAME:
```

q_alert

```
 PURPOSE:

 CALLING SEQUENCE:
      n = q_alert(type, length, data)

 INPUTS:
      type
      length
      data

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      just about everywhere

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header



   Turn off output buffering to maintain TM sequence.
```

```
NAME:

      delayed_q_alert

 PURPOSE:

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      q_alert                  [sun_util.c, cwhite.c]

 CALLED FROM:
      whitedsp                 [whitedsp.c (this file)]
      choose_guide_stars_task  [whitedsp.c (this file)]
      engineering_task         [whitedsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      27Mar96   ADW   add standard comment header
```

```
NAME:

      sci_command

 PURPOSE:

 CALLING SEQUENCE:
      sci_command(target, message);

 INPUTS:
      target
      message

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      read_proc_int             [sun_util.c (this file)]
      read_proc_int_array       [sun_util.c (this file)]
      blue_local_ram_io         [shared_io_functions.c]
      red_local_ram_io          [shared_io_functions.c]

 CALLED FROM:
      abort_task                [whitedsp.c]
      acquire_field_task        [whitedsp.c]
      choose_guide_stars_task   [whitedsp.c]
      init_exp_task             [whitedsp.c]
      switch_ping_pong          [whitedsp.c]
      track_guide_stars_task    [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header
      02Oct95   ADW   add Red DSP task commands
```

```
NAME:

     hdr_manager          function

 PURPOSE:
      hdr_manager is used to add the DPU header

 CALLING SEQUENCE:
      integer_variable = hdr_manager (hdr, hdr_length);

 INPUTS:
      hdr
      hdr_length

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none (returns base address for header)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_proc_int_array   [sun_util.c; global_access.c]

 CALLED FROM:
      do_10                  [cwhite.c]
      compress               [su_compress.c]
      deliver_bluefast_data  [su_deliverdata.c]
      deliver_image_data     [su_deliverdata.c]
      deliver_ref_data       [su_deliverdata.c]
      deliver_trh_data       [su_deliverdata.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      17May95   ADW   add standard comment header



   Write to proc memory at address, ysize, yskip, xsize for ...???
   (see white.asm for details on write_proc_int_array)
   @@@ what is the yskip for?
```

```
NAME:

      debuggingermost

 PURPOSE:

 CALLING SEQUENCE:
      debuggingermost();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      read_proc_int             [sun_util.c (this file)]
      write_proc_int            [sun_util.c (this file)]
      read_proc_int_array       [sun_util.c (this file)]
      write_proc_int_array      [sun_util.c (this file)]
      read_proc_float           (declared in sun_util.h, but no function!)
      write_proc_float               "
      read_proc_float_array          "
      write_proc_float_array         "
      read_bw_int               [sun_util.c (this file)]
      write_bw_int              [sun_util.c (this file)]
      read_bw_int_array         [sun_util.c (this file)]
      write_bw_int_array        [sun_util.c (this file)]
      read_sw_int               [sun_util.c (this file)]
      write_sw_int              [sun_util.c (this file)]
      read_sw_int_array         [sun_util.c (this file)]
      write_sw_int_array        [sun_util.c (this file)]

 CALLED FROM:
      main                      [sun_util.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header
```

```
NAME:

      main

 PURPOSE:

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      close_global_ram_big
      close_global_ram_proc
      close_global_ram_small
      debuggingermost
      make_bw_ram
      make_sw_ram
      open_global_ram_big
      open_global_ram_proc
      open_global_ram_small

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header

  The following are routines for the GLOBAL_ACCESS_C = 1 case
```

```
NAME:

      read_int

 PURPOSE:

 CALLING SEQUENCE:
      n = read_int(base, destination);

 INPUTS:
      base
      destination

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      read_proc_int   [global_access.c]
      read_bw_int     [global_access.c]
      read_sw_int     [global_access.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header
```

```
NAME:

      write_int

 PURPOSE:

 CALLING SEQUENCE:
      n = write_int(to, from);

 INPUTS:
      to
      from

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      write_proc_int   [global_access.c]
      write_bw_int     [global_access.c]
      write_sw_int     [global_access.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header
```

```
NAME:

      read_int_array

 PURPOSE:

 CALLING SEQUENCE:
      n = read_int_array(base, ysize, yskip, xsize, destination);

 INPUTS:
      base
      ysize
      yskip
      xsize
      destination

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns the number of integers read)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      read_proc_int_array   [global_access.c]
      read_bw_int_array     [global_access.c]
      read_sw_int_array     [global_access.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header
```

```
NAME:

      write_int_array

 PURPOSE:

 CALLING SEQUENCE:
      n = write_int_array(to, ysize, yskip, xsize, from);

 INPUTS:
      to
      ysize
      yskip
      xsize
      from

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns the number of integers written)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      write_proc_int_array   [global_access.c]
      write_bw_int_array     [global_access.c]
      write_sw_int_array     [global_access.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header
```

```
NAME:

      read_proc_int

 PURPOSE:

 CALLING SEQUENCE:
      n = read_proc_int(base, ysize, yskip, xsize, destination);

 INPUTS:
      base
      destination

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns the number of integers read)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      calbgd                    [su_calbgd.c]
      deliver_faq_data          [su_deliverdata.c]
      deliver_bluefast_data     [su_deliverdata.c]
      deliver_image_data        [su_deliverdata.c]
      deliver_wdw_data          [su_deliverdata.c]
      deliver_bfast             [su_deliverdata.c]
      deliver_ref_data          [su_deliverdata.c]
      dowindow                  [su_dowindow.c]
      locategs                  [su_locategs.c]
      makeags_id                [su_makeags_id.c]
      restore_stars             [su_restore_stars.c]
      saa                       [reddsp.c]
      scanbs                    [su_scanbs.c]
      scangs                    [su_scanbg.c]
      debuggingermost           [sun_util.c (this file)]
      track_guide_stars_task    [whitedsp.c]
      mic                       [mic1.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header
```

NAME:

read_proc_int_array

PURPOSE:

CALLING SEQUENCE:
```
n = read_proc_int_array(base, ysize, yskip, xsize, destination);
```

INPUTS:
```
base
ysize
yskip
xsize
destination
```

OPTIONAL INPUT PARAMETERS:

OUTPUTS:
```
none  (returns the number of integers read)
```

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:
```
calbgd                  [su_calbgd.c]
deliver_faq_data        [su_deliverdata.c]
deliver_image_data      [su_deliverdata.c]
deliver_wdw_data        [su_deliverdata.c]
deliver_bfast           [su_deliverdata.c]
deliver_ref_data        [su_deliverdata.c]
dowindow                [su_dowindow.c]
getref                  [su_getref.c]
locategs                [su_locategs.c]
makeags_id              [su_makeags_id.c]
restore_stars           [su_restore_stars.c]
saa                     [reddsp.c]
scangs                  [su_scangs.c]
exclude_crowded_fields  [su_scangs.c]
debuggingermost         [sun_util.c (this file)]
mic                     [mic1.c]
```

RESTRICTIONS:

MODIFICATION HISTORY:
```
19Jul95   ADW   add standard comment header
```

```
NAME:

      write_proc_int

 PURPOSE:

 CALLING SEQUENCE:
      n = write_proc_int(to, from);

 INPUTS:
      to
      from

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns the number of integers written)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      calbgd                   [su_calbgd.c]
      caldrft                  [su_caldrft.c]
      calculate_parameters     [su_caldrft.c]
      dowindow                 [su_dowindow.c]
      getref                   [su_getref.c]
      init_exp                 [su_init_exp.c]
      locategs                 [su_locategs.c]
      find_brightest_pixel     [su_locategs.c]
      buildstar1               [su_locategs.c]
      summer1                  [su_locategs.c]
      makeags_id               [su_makeags_id.c]
      restore_stars            [su_restore_stars.c]
      saa                      [reddsp.c]
      align                    [reddsp.c]
      accumulate               [reddsp.c]
      add_frame_buffer         [reddsp.c]
      scanbs                   [su_scanbs.c]
      exclusion                [su_scanbs.c]
      bs_find_brightest_pixel  [su_scanbs.c]
      buildstar                [su_scanbs.c]
      summer                   [su_scanbs.c]
      hindx                    [su_scanbs.c]
      scangs                   [su_scangs.c]
      exclude_border           [su_scangs.c]
      exclude_crowded_fields   [su_scangs.c]
      is_it_good               [su_scangs.c]
      in_exclusion_zone        [su_scangs.c]
      add_exclusion_box        [su_scangs.c]
      debuggingermost          [sun_util.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header
```

```
NAME:
```

write_proc_int_array

```
 PURPOSE:

 CALLING SEQUENCE:
      n = write_proc_int_array(to, ysize, yskip, xsize, from);

 INPUTS:
      to
      ysize
      yskip
      xsize
      from

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns the number of integers written)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      caldrft                [su_caldrft.c]
      dowindow               [su_dowindow.c]
      getref                 [su_getref.c]
      init_exp               [su_init_exp.c]
      locategs               [su_locategs.c]
      makeags_id             [su_makeags_id.c]
      restore_stars          [su_restore_stars.c]
      scanbs                 [su_scanbs.c]
      scangs                 [su_scangs.c]
      debuggingermost        [sun_util.c (this file)]
      hdr_manager            [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header
```

```
NAME:

      read_bw_int

 PURPOSE:

 CALLING SEQUENCE:
      n = read_bw_int(base, destination);

 INPUTS:
      base
      destination

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      debuggingermost   [sun_util.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header
```

NAME:

read_bw_int_array

PURPOSE:

CALLING SEQUENCE:
```
     n = read_bw_int_array(base, ysize, yskip, xsize, destination);
```

INPUTS:
```
     base
     ysize
     yskip
     xsize
     destination
```

OPTIONAL INPUT PARAMETERS:

OUTPUTS:
```
     none  (returns the number of integers read)
```

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:
```
     accumulate        [reddsp.c]
     debuggingermost   [sun_util.c (this file)]
```

RESTRICTIONS:

MODIFICATION HISTORY:
```
     19Jul95   ADW   add standard comment header
```

```
NAME:

      write_bw_int

 PURPOSE:

 CALLING SEQUENCE:
      n = write_bw_int(to, from);

 INPUTS:
      to
      from

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns the number of integers written)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      caldrft            [su_caldrft.c]
      deliver_ref_data   [su_deliverdata.c]
      debuggingermost    [sun_util.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header
```

```
NAME:

      write_bw_int_array

 PURPOSE:

 CALLING SEQUENCE:
      n = write_bw_int_array(to, ysize, yskip, xsize, from);

 INPUTS:
      to
      ysize
      yskip
      xsize
      from

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns the number of integers written)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      caldrft                 [su_caldrft.c]
      deliver_ref_data        [su_deliverdata.c]
      accumulate              [reddsp.c]
      debuggingermost         [sun_util.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header
```

```
NAME:

      read_sw_int

 PURPOSE:

 CALLING SEQUENCE:
      n = read_sw_int(base, destination);

 INPUTS:
      base
      destination

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns the number of integers read)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      debuggingermost    [sun_util.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header
```

```
NAME:

      read_sw_int_array

 PURPOSE:

 CALLING SEQUENCE:
      n = read_sw_int_array(base, ysize, yskip, xsize, destination);

 INPUTS:
      base
      ysize
      yskip
      xsize
      destination

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns the number of integers read)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      locategs           [su_locategs.c]
      add_frame_buffer   [reddsp.c]
      scanbs             [su_scanbs.c]
      debuggingermost    [sun_util.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header
```

```
NAME:

      write_sw_int

 PURPOSE:

 CALLING SEQUENCE:
      n = write_sw_int(to, from);

 INPUTS:
      to
      from

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none  (returns the number of integers written)

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:
      debuggingermost   [sun_util.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      19Jul95   ADW   add standard comment header
```

NAME:

write_sw_int_array

PURPOSE:

CALLING SEQUENCE:

```
n = write_sw_int_array(to, ysize, yskip, xsize, from);
```

INPUTS:

```
to
ysize
yskip
xsize
from
```

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

```
none  (returns the number of integers written)
```

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

```
debuggingermost   [sun_util.c (this file)]
```

RESTRICTIONS:

MODIFICATION HISTORY:

```
19Jul95   ADW   add standard comment header
```

NAME:

whitedsp.c

```
 PURPOSE:
      These are the main programs of the White DSP software.
      They control the traffic of the DPU operations.

 INCLUDE FILES:
      setup.h                (SUN and DPU)
      shared_variables.c     (SUN only)
      sun_util.c             (SUN only)
      shared_io_functions.c  (SUN only)
      global_access.c        (DPU only)

 FUNCTIONS IN THIS FILE:
      sunload
      main
      whitedsp
      abort_task
      initialize_task
      init_exp_task
      acquire_field_task
      choose_guide_stars_task
      track_guide_stars_task
      compress_data_task
      engineering_task
      finish_frame_task
      flush_compress_task
      switch_ping_pong
      hdr_manager
      global_zero
      delayed_q_alert

 MODIFICATION HISTORY:
      16May95   ADW   add standard comment header
      29Sep95   ADW   Make changes for moving shift-and-add to the Red
DSP
      03Nov95   ADW   settle saa to Red changes for DPU testing;
                      add skeleton for compression task.
      12Mar96   ADW   add engineering mode support
      28Oct98   CH    Major update on in-code comments

 RCS $Id: whitedsp.c,v 1.73 2000/08/11 13:57:25 ho Exp $

Define parameters and variables by including setup.h
Declare the functions used.

If SUN is defined, we'll be running whitedsp in simulation mode on a Sun
UNIX system.  In this case, we need to do some simulation-specific
setup, and compile a whitedsp main routine and a swap unit loader
function.

Include three simulation-only source files which
define utilities, variables, and I/O functions.
Define sunload as the execute function for simulation.
```

```
NAME:
```

sunload function

```
 PURPOSE:
      Sunload function (cast as integer) used when running in simulation
      on a UNIX system (SUN) to call the swap units as subroutines.
      "swap_to" is a parameter passed to sunload in the call.

 CALLING SEQUENCE:
      return_status = execute(swap_to);
      where execute implies sunload (above)

 INPUTS:
      swap_to         swap unit identifier (see white_control.h)

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns swap_to)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      calbgd                   [su_calbgd.c]
      caldrft                  [su_caldrft.c]
      compress                 [su_compress.c]
      deliverdata              [su_deliverdata.c]
      dowindow                 [su_dowindow.c]
      dpuabort                 [su_dpuabort.c]
      engineering              [su_engineering.c]
      getref                   [su_getref.c]
      initialize               [su_initialize.c]
      init_exp                 [su_init_exp.c]
      locategs                 [su_locategs.c]
      makeags_id               [su_makeags_id.c]
      restore_stars            [su_restore_stars.c]
      scanbs                   [su_scanbs.c]
      scangs                   [su_scangs.c]
      submit_cmprss_q          [su_submit_cmprss_q.c]

 CALLED FROM:
      abort_task               [whitedsp.c (this file)]
      initialize_task          [whitedsp.c (this file)]
      init_exp_task            [whitedsp.c (this file)]
      acquire_field_task       [whitedsp.c (this file)]
      choose_guide_stars_task  [whitedsp.c (this file)]
      track_guide_stars_task   [whitedsp.c (this file)]
      compress_data_task       [whitedsp.c (this file)]
      finish_frame_task        [whitedsp.c (this file)]
      engineering_task         [whitedsp.c (this file)]
      flush_compress_task      [whitedsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      11May95   ADW   add standard comment header
      29Sep95   ADW   comment shift-and-add block (saa now in reddsp)
      12Mar96   ADW   add engineering mode support
      28Oct98   CH    Major update on in-code comments


   Run the requested swap unit code.
   Because sunload will run only in a UNIX/Sun environment,
```

```
use of switch/case (inefficient on DPU) is okay here.
The input swap_id should have the value of one of the
swap unit symbolic constants, e.g., SWAP_INITIALIZE,
defined in the include file(s asm.)white_control.h.
   Run the indicated swap unit and
   Place the return value in swap_id.
   If the swap unit specification is invalid,
   print an error message and exit.
Return the return value from the swap unit code to the calling task.
```

```
NAME:
```

main ***main function***

```
 PURPOSE:
      Main function used when running in simulation on a UNIX system
      (SUN) to emulate the DPU operating system

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      global_ram_io      [shared_io_functions.c]
      local_ram_io       [shared_io_functions.c]
      global_zero        [whitedsp.c (this file)]
      blue_local_ram_io  [shared_io_functions.c]
      red_local_ram_io   [shared_io_functions.c]
      whitedsp           [whitedsp.c (this file)]
      q_alert            [sun_util.c, cwhite.c]
      delayed_q_alert    [whitedsp.c (this file)]

 CALLED FROM:

 RESTRICTIONS:

      Note:   see header comments at the top of this file;
         some information there applies to this main function...

 MODIFICATION HISTORY:
      16May95   ADW   add standard comment header, structured English
comments
      02Oct95   ADW   add command to initialize Red DSP
      28Oct98   CH    Major updates on in−code comments



   Read the global ram file into memory.
   Read the (White) local ram file into memory.
   If the local ram file does not exist
      zero the PROC area,
      mimic the alert sent by the DPU when loading the DPUOS,
      create the Blue and Red local RAM files,
      command the Red DSP to IDLE,
      and set the current White DSP task to idle.
   Read the blue_local_ram file.
   Put the whitedsp entry status into the log file.
   Run the whitedsp function.
   Mock increment of bus_cycle_count
   Put the whitedsp exit status into the log file.
   Write the global_ram file.
   Write the (White) local_ram file.
   Write the blue_local_ram file.

When running whitedsp on the DPU itself,
define load function (operating system function to load
the swap units, see white.asm) as the execute function,
and (optionally) include global_access.c [now at bottom of file].
```

```
NAME:
```

whitedsp

```
 PURPOSE:
      Whitedsp function controls the traffic of the DPU operations by
      loading the proper sequence of swap units given the task_id passed
      to it via local ram from the DPUOS (see ext_psu.h and cwhite.c).

 CALLING SEQUENCE:
      whitedsp();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      abort_task                [whitedsp.c (this file)]
      engineering_task          [whitedsp.c (this file)]
      finish_frame_task         [whitedsp.c (this file)]
      acquire_field_task        [whitedsp.c (this file)]
      choose_guide_stars_task   [whitedsp.c (this file)]
      initialize_task           [whitedsp.c (this file)]
      init_exp_task             [whitedsp.c (this file)]
      track_guide_stars_task    [whitedsp.c (this file)]

 CALLED FROM:
      main                      [whitedsp.c (this file)]
      main                      [cwhite.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      15Sep93   CH    Start from scratch.
      29Nov93   SDH   Added standard header
      05Aug94   SDH   Added structured English comments; removed gotos
      19Aug94   SDH   revised comment headers
      12Mar96   ADW   add engineering mode support
      28Oct98   CH    Major update on in-code comments


   Enter an infinite loop.  A break out occurs
   when task_id becomes IDLE in Sun simulations.
      Test the value of task_id and execute the indicated task.
      If task_id equals IDLE,
         return to main if running in simulation,
         continue looping if running on the DPU.
      If task_id equals ABORT, run abort_task.
      If task_id equals INITIALIZE, run initialize_task.
      If task_id equals INIT_EXP, run init_exp_task.
      If task_id equals ACQUIRE_FIELD, run acquire_field_task.
      If task_id equals CHOOSE_GUIDE_STARS, run choose_guide_stars_task.
      If task_id equals TRACK_GUIDE_STARS, run track_guide_stars_task.
      If task_id equals COMPRESS_DATA, run compress_data_task.
      If task_id equals FINISH_FRAME, run finish_frame_task.
      If task_id equals FLUSH_COMPRESS, run flush_compress_task.
      If task_id equals ENGINEERING, run engineering_task.
      If the task_id is invalid, set it to ABORT.  Then
      abort_task will be run on the next pass through the loop.
```

NAME:

idle_task

```
 PURPOSE:
      idle, check for scrub errors

 CALLING SEQUENCE:
      idle_task();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      q_alert             [sun_util.c, cwhite.c]
      read_proc_int       [sun_util.c, global_access.c]
      write_proc_int      [sun_util.c, global_access.c]
      read_proc_int_array [sun_util.c, global_access.c]

 CALLED FROM:
      whitedsp            [whitedsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      14Sep98   CH     Start from scratch.
      28Oct98   CH     Major updates on in-code comments.


  Read in the FEC error pointer array
  Ready to loop through the error pointer array
  Loop through the three sets, blue1, blue2, and red
    If there is a mismatch in the current error and reported
    error pointer, then there is a new FEC error.
      Make sure the no of new FEC errors is not negative
      Find out which one we have already sent out
      Read in the entire FEC error message array
      Loop through all of the new errors
      Set the index to a new FEC error
      Make sure we don't overrun 16 entries
      Grab the error message
      Getbthe upper 8 bits of the error message
        If the error-nature indiecator is 1, single bit error!
        Set up the alert buffer and send it out
      Otherwise, it is a multiple bit error
        Set up buffer and send out message for multiple bit error
      Update the last FEC sent out
    Done with the FEC error checking for the previous processor
    Move on to the next one
```

```
NAME:
```

abort_task

```
 PURPOSE:
      Gracefully abort from the current task

 CALLING SEQUENCE:
      abort_task();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      q_alert             [sun_util.c, cwhite.c]
      execute (sunload)   [whitedsp.c (this file)]
      sci_command         [sun_util.c, white.asm]

 CALLED FROM:
      whitedsp            [whitedsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      15Sep93   CH    Start from scratch.
      29Nov93   SDH   Added standard header
      05Aug94   SDH   Added structured English comments, removed gotos
      19Aug94   SDH   revised comment headers
      25Nov94   JMK   enable BDSP fast mode event collection if
n_bfast>0
      28Oct98   CH    Major updates on in-code comments


   Issue an alert that abort is in progress (DA_DPU_ABORTING).
   Disable Blue DSP event collection.
   If n_bfast is greater than zero
      disable Blue DSP fast mode event collection.
   Run the su_dpuabort swap unit to gracefully abort the current task.
   Set task_id to IDLE.
```

```
NAME:
```

initialize_task   function

```
 PURPOSE:
      This task initializes all variables, clean up the PROC areas,
      set default values, and zero out small and big word memory.

 CALLING SEQUENCE:
      initialize_task();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      execute (sunload)  [whitedsp.c (this file)]
      sci_command        [sun_util.c, white.asm]
      q_alert            [sun_util.c, cwhite.c]

 CALLED FROM:
      whitedsp           [whitedsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      17May95   ADW   add standard comment header, structured English
comments
      03Oct95   ADW   add Red DSP initialize command
      28Oct98   CH    Major updates on in-code comments


   Run the su_initialize swap unit to initialize
   the shared variables at their default values.
   Issue end of initialize_task alert (DA_EOT_INIT_DPU)
   If the task_id is still INITIALIZE
      set task_id to IDLE.
   Otherwise accept the new value.
```

```
NAME:
```

init_exp_task function

```
 PURPOSE:
      This task initialize the exposure. Specifically, it sets up the
      bluedsp to acquire a BPE-binned 1kx1k image.

 CALLING SEQUENCE:
      init_exp_task();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      execute (sunload)  [whitedsp.c (this file)]
      q_alert            [sun_util.c, cwhite.c]
      sci_command        [sun_util.c, white.asm]

 CALLED FROM:
      whitedsp           [whitedsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      17May95   ADW   add standard comment header, structured English
comments
      28Oct98   CH    Major updates on in-code comments

   Run the init_exp swap code to prepare for reference frame
acquisition.
   Issue an end of init_exp_task alert (DA_EOT_INIT_EXP).
   If the task_id is INIT_EXP
      set task_id to IDLE.
   Otherwise accept the new value.
```

```
NAME:
```

acquire_field_task function

```
 PURPOSE:
      This task acquire the absolute field pointing.

 CALLING SEQUENCE:
      acquire_field_task();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      execute (sunload)  [whitedsp.c (this file)]
      q_alert            [sun_util.c, cwhite.c]
      sci_command        [sun_util.c, white.asm]
      switch_ping_pong   [whitedsp.c (this file)]

 CALLED FROM:
      whitedsp           [whitedsp.c]

 RESTRICTIONS:

      currently this is exclusively for the Blue detector, I assume we
will
      make the detector used for acquisition commandable

 MODIFICATION HISTORY:
      17May95   ADW   add standard comment header, structured English
comments
      28Oct98   CH    Major updates on in-code comments


   Tell the blue DSPs to fetch the detector
   window parameters from proc memory.
   Enable blue DSP event collection.
   Add current bus cycle count to the frame time
   to get bus cycle count at end of frame time.
   In simulation, the acquisition data have already been written
   into the global memory, so we don't need to wait now.
   If running on the DPU, wait until the frame time has elapsed.
   Disable Blue DSP event collection.
   Issue an end of field acquisition frame alert
   so the ICU can move the filer wheels for the next task.
   Run the switch_ping_pong function.
   Run the su_calbgd swap unit to calculate
   the level of the background in a frame.
   Optionally, issue an alert when done.
   If the number of guide stars found is 0 or fewer, issue an alert.
   And turn tracking off
   If CGS has failed, make sure the Red DSP knows it.
   Run the su_scanbs swap unit to locate and
   catalog the brightest stars in a frame.
   Optionally, issue an alert when done.
   Run the su_restore_stars swap unit to convert data to unbinned
detector coordinates and correct for Blue detector spatial distortion.
   Optionally, issue an alert when done.
   Run the su_makeags_id swap unit to identify up-linked guide stars
   from the list of bright stars generated by su_scanbs.
```

```
Loop as long as the makeags_id return_status is ITERATE.
   Run the su_caldrft swap unit to calculate the position
   of the guide stars relative to the reference frame.
   Optionally, issue an alert when done.
   Run the su_makeags_id swap unit to identify up-linked guide stars
   from the list of bright stars generated by su_scanbs.
   Optionally, issue an alert when done.
Issue an end of acquisition task alert (DA_EOT_ACQUIRE_FLD).
Load data type DP_FAQ into data_delivery variable
in global memory for passage to su_deliverdata, and
run the su_deliverdata swap unit to prepare the data
for output and alert the ICU that the data are available.
If the task_id is ACQUIRE_FIELD
   set task_id to IDLE.
Otherwise, accept the new value.
```

NAME:

choose_guide_stars_task function

```
 PURPOSE:
      This task scan the bright stars in the reference frame,
      choose the guide stars and set up the window/memory allocation

 CALLING SEQUENCE:
      choose_guide_stars_task();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      execute (sunload)  [whitedsp.c (this file)]
      q_alert            [sun_util.c, cwhite.c]
      sci_command        [sun_util.c, white.asm]
      switch_ping_pong   [whitedsp.c (this file)]

 CALLED FROM:
      whitedsp           [whitedsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      17May95   ADW   add standard comment header, structured English
comments
      28Oct98   CH    Major updates on in-code comments



     Set up the null condition for no tracking whatsoever
     Allocate the windows and memory
     Still wants to send out the DP_WDW to trigger ICU to set up the BPE
     Issue an end of choose_guide_stars_task alert (DA_EOT_CHOOSE_GS).
     If the task_id is CHOOSE_GUIDE_STARS
        set task_id to idle.
   Initialize number of guide stars found to zero.
   If no tracking, set up the null condition
   If tracking_on == 1, then expose reference frame and choose GS
   Tell the blue DSPs to fetch the detector
   window parameters from proc memory.
   Enable blue DSP event collection.
   Add current bus cycle count to the frame time
   to get bus cycle count at end of frame time.
   In simulation, the reference data have already been written
   into the global memory, so we don't need to wait now.
   If running on the DPU, wait until the frame time has elapsed.
   Disable blue DSP event collection.
   Run the switch_ping_pong function.
   Run the su_calbgd swap unit to calculate
   the level of the background in a frame.
   Optionally, issue an alert when done.
   The following loop has been disabled because the return value
   from scangs was never ITERATE.     23Jul96  ADW
   Set return_status to ITERATE to enter the following while loop.
   Loop while the returned value from su_scangs is ITERATE.
      Run the su_scanbs swap unit to locate
      and catalog the brightest stars in a frame.
```

```
      Optionally, issue an alert when done.
      If drift tracking is enabled:
      (Tracking can be disabled by explicit command or if
      scanbs does not find an acceptable number of bright stars.)
         Run the su_restore_stars swap unit to convert data
         to unbinned detector coordinates and correct
         for blue detector spatial distortion.
         Optionally, issue an alert when done.
         Run the su_scangs swap unit to select appropriate
         guide stars from the catalog of bright stars.
         Optionally, issue an alert when done.
         If the number of guide stars found is 0 or fewer, issue an
alert.
   Run the su_dowindow swap unit to establish the science, memory,
   and tracking windows corresponding to the observer-specified science
   and memory windows, and the science, memory, and detector windows
   required for the guide stars selected.
   Optionally, issue an alert when done.
   If drift tracking is enabled:
      Run the su_restore_stars swap unit to convert data
      to unbinned detector coordinates and correct
      for Blue detector spatial distortion.
      Optionally, issue an alert when done.
      Run the su_getref swap unit to calculate the guide star
      positions relative to the roll center of spacecraft.
      Optionally, issue an alert when done.
   Issue an end of choose_guide_stars_task alert (DA_EOT_CHOOSE_GS).
   Load data type DP_WDW into data_delivery variable
   in global memory for passage to su_deliverdata, and
   run the su_deliverdata swap unit to prepare the data
   for output and alert the ICU that the data is available.
   If there are blue fast mode areas
      load data type DP_BFAST into data_delivery variable
      in global memory for passage to su_deliverdata,
      and run the su_deliverdata swap unit to prepare the data
      for output and alert the ICU that the data are available.
   Optionally, issue an alert when done.
   Tell the blue DSPs to fetch the detector
   window parameters from proc memory.
   If there are blue fast mode areas
      tell the blue DSPs to fetch the fast mode
      detector window parameters from proc memory.
   If running in simulation, enable Blue DSP event collection,
   including fast mode if appropriate.
   Enable Blue DSP event collection, including fast mode events,
   if appropriate, if we are running in "track by hand" mode.
   (When auto-piloting, this will be done in track_guide_stars_task().
   If the task_id is CHOOSE_GUIDE_STARS
      set task_id to idle.
   Otherwise, accept the new value.
```

NAME:

track_guide_stars_task function

```
 PURPOSE:
      This task performs the frame-by-frame tracking, directs
      bluedsp ping/pong data acquisition, direct the red to
      perform shift and add, AND performs the compression in its
      spare time.

 CALLING SEQUENCE:
      track_guide_stars_task();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      data_queue_stat     [sun_util.c, cwhite.c]    commented
      execute (sunload)   [whitedsp.c (this file)]
      q_alert             [sun_util.c, cwhite.c]
      read_proc_int       [sun_util.c, global_access.c]
      sci_command         [sun_util.c, white.asm]
      switch_ping_pong    [whitedsp.c (this file)]
      write_proc_int      [sun_util.c, global_access.c]

 CALLED FROM:
      whitedsp            [whitedsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      17May95   ADW   add standard comment header, structured English
comments
      29Sep95   ADW   Change for shift-and-add move to Red DSP
      28Oct98   CH    Major updates on in-code comments



   Increment frame count by 1.
   If we are just starting the exposure
      Initialize counter of verbose tracking data.
      Initialize the compress queue parameters
      to invoke the compression processing
      Signal the compress queue to start when invoked.
      Add current bus cycle count to the frame time (in milliseconds)
      to get bus cycle count at end of frame time
      If running in simulation, just send a beginning of exposure alert.
      The tracking frame data are already in global memory.
      If running on the DPU, enable Blue DSP event collection,
      including fast mode events if necessary.
      Note that this has already been done for track-by-hand
      mode in choose_guide_stars_task.  There is no harm in
      doing this again here.
      Send a beginning of exposure alert.
      If the DPU is auto-piloting (the normal mode),
      wait for the tracking frame time to complete,
      and update the timer for the next frame.
       *** If single frame exposure is allowed, do compression *
           Call the compression swap unit.
           If there are no more entries in the compression queue,
```

```
         then exit the "work on compression" loop.
         Update the time_left in the current tracking frame.
   If the frame count is equal to or greater than
   the number of frames per exposure, then
      disable Blue DSP event collection,
      if there are active fast mode areas
      disable Blue DSP fast mode event collection,
      and issue an "end of exposure" alert.
   Run switch_ping_pong function
   If the task_id has been set to ABORT,
   return control to whitedsp().
      Run the su_calbgd swap unit to calculate
      the level of the background in a frame.
      Optionally, issue an alert when done.
      Run the su_locategs swap unit to locate the guide stars
      in a frame given the guide stars' previous location.
      Optionally, issue an alert when done.
      Run the su_restore_stars swap unit to convert data to unbinned
      detector coordinates and correct for detector spatial distortion.
      Optionally, issue an alert when done.
      Run the su_caldrft swap unit to calculate the position
      of the guide stars relative to the reference frame.
      Optionally, issue an alert when done.
      Load data type (DA_TRK) into data_delivery variable
      in global memory for passage to su_deliverdata, and
      run the su_deliverdata swap unit to prepare the data
      for output and alert the ICU that the data are available
      Optionally, issue an alert when done.
   If task_id is no longer TRACK_GUIDE_STARS
   set it to ABORT and return to whitedsp().
   If the task_id is still TRACK_GUIDE_STARS we must
   command Red DSP to shift-and-add the latest frame into the
   accumulating exposure.  While Red does that, the White DSP
   can work on compressing the previous exposure.
      Red DSP will need current values of these PROC variables.
      Command the Red DSP to do the shift-and-add process.
      Write the shift-and-add status SAA_START into PROC RAM.
      When running in simulation, we need to exit from whitedsp here
      so that we can run reddsp to do the shift-and-add operation.
      This is accomplished by setting the White DSP task_id to IDLE.
      After reddsp is run, whitedsp will be invoked again with the
      task_id set to COMPRESS_DATA.
      In the flight code, set the White DSP task_id to COMPRESS_DATA.
```

```
NAME:
```

compress_data_task

```
 PURPOSE:
      This task compressed the previous exposure data while the Red DSP
is
      performing the shift-and-add of the previous frame and the current
      frame is still exposing

 CALLING SEQUENCE:
      compress_data_task();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      execute (sunload)   [whitedsp.c (this file)]
      read_sw_int         [sun_util.c, global_access.c]
      q_alert             [sun_util.c, cwhite.c]
      read_proc_int       [sun_util.c, global_access.c]
      write_proc_int      [sun_util.c, global_access.c]

 CALLED FROM:
      whitedsp            [whitedsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      03Nov95   ADW   new; shell only
      28Oct98   CH    Major updates on in-code comments


   Determine the amount of time left in the current tracking frame.
   In simulation mode, we have no way of simulating real-time
   multiprocessor operation.  To mimic what happens in the
   flight code, compress 50 blocks per frame.
   As long as the remaining time is greater than COMPRESS_TIME_BUFFER
   White DSP may work on compressing the previous exposure.
      Call the compression swap unit.
      If there are no more entries in the compression queue,
      then exit the "work on compression" loop.
      Update the time_left in the current tracking frame.
   In waiting for Red DSP to finish the shift-and-add, we can loop here
   or in the main whitedsp() control loop.  WAIT_FOR_SAA_INSIDE == 1
   indicates that the former option is desired.
         If the value of saa_status is (still) SAA_START,
         the Red DSP is still preforming the shift-and-add.
         We have to wait for it to finish before moving on
         to finish_frame_task and begin the next frame.
         Doing nothing here (followed by "return" below)
         creates a wait loop: whitedsp will execute this
         task/function until SAA_DONE or SAA_IDLE is seen.
         Optionally, issue an end of saa alert.
         If the value of saa_status is SAA_DONE,
         the Red DSP has just completed the shift-and-add.
         In this case, it is safe to run finish_frame_task.
         If we are waiting inside compress_data_task for shift-and-add
       to finish, we break out of the wait loop here.
```

```
  There's nothing special to do if saa_status is SAA_IDLE.
  In fact, we shouldn't get here at all in the current code.
  This may change if compression is invoked somewhere other
  than at the end of track_guide_stars_task.
  Set the task ID to IDLE for now.   ADW 03Nov95
  If we are waiting inside compress_data_task for shift-and-add
to finish, we break out of the wait loop here.
```

```
NAME:
```

finish_frame_task function

```
 PURPOSE:
      This task is called to specifically clean up the last frame in an
      exposure

 CALLING SEQUENCE:
      finish_frame_task();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      q_alert             [sun_util.c, cwhite.c]
      execute (sunload)   [whitedsp.c (this file)]
      flush_compress_task [this file]

 CALLED FROM:
      whitedsp            [whitedsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      17May95   ADW   add standard comment header, structured English
comments
      29Sep95   ADW   add conditions to handle shift-and-add move to Red
DSP
      03Nov95   ADW   rename to match current functionality;
                      saa stuff is now at end of TGS and CD tasks.
      28Oct98   CH    Major updates on in-code comments


   If the exposure is complete, we must swap the accumulation memory
   areas and submit the exposure data to the data or compression queue.
       First flush out all data in the existing compression queue
       Then set task_id back
       Swap the addresses of the two exposure accumulation memory areas.
       The exposure data will normally be submitted to the compression
       queue.  The option exists to send the data directly to the data
       queue.  Determine which is in effect, and act accordingly.
       This action involves setting the data_delivery variable and
       invoking the appropriate swap unit.
        Load the DD_TRH (tracking history), DD_REF (reference data),
       DD_IMG (image data), and if necessary DD_FST (fast mode data)
       data types into the data_delivery variable in sequence for
       passage to the submit_compress_q swap unit.
         Load the DD_TRH (tracking history), DD_REF (reference data),
       DD_IMG (image data), and if necessary DD_FST (fast mode data)
       data types into the data_delivery variable in sequence for
       passage to the deliverdata swap unit.
      Send a complete exposure signal.
      Set the task_id to IDLE and return. This is okay because the
      exposure is done and we need to get a new command from the ICU.
   Otherwise the exposure is not done.  In this case
   set the task ID to IDLE for Sun simulation
   runs so that the whitedsp program will exit,
   allowing the next tracking frame data to be
```

```
written into global memory.
carry out traffic control if running on the DPU.

If running in "track by hand" mode, then:
   Set task_id to idle task to wait for command from ICU to continue
Else if not "track by hand"
   Be very careful with this automated framing
   If task_id is no longer FINISH_FRAME, then
      set task_id to ABORT and return to whitedsp.
   Wait until the frame time has elapsed.
   Increment last_frame_time by the frame time.
   If task_id is no longer FINISH_FRAME
      set task_id to ABORT.
   If task_id is still FINISH_FRAME
      reset task_id back to TRACK_GUIDE_STARS to find
      the guide stars in the just-completed frame.
```

NAME:

flush_compress_task

```
PURPOSE:
     This task is called to force the compression of all processed
     and uncompressed data sets stored in memory to be compressed.

CALLING SEQUENCE:
     flush_compress_task();

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
     q_alert             [sun_util.c, cwhite.c]
     execute (sunload)   [whitedsp.c (this file)]

CALLED FROM:
     whitedsp            [this file]

RESTRICTIONS:

MODIFICATION HISTORY:
     07Jun96   CH    new
     28Oct98   CH    Major updates on in-code comments


  Set up the queues
```

NAME:

engineering_task

```
PURPOSE:
     This task is the driver for all engineering mode data acquisition
     and processing.

CALLING SEQUENCE:
     engineering_task();

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
     execute (sunload)   [whitedsp.c (this file)]

CALLED FROM:
     whitedsp            [whitedsp.c (this file)]

RESTRICTIONS:

MODIFICATION HISTORY:
     10Nov95   ADW   new
     28Oct98   CH    Major updates on in-code comments


  Run the su_engineering swap unit.
     Idle if running in track-by-hand mode.
  Issue an (optional) end of su_engineering alert.
  At the end of engineering, set the primary pointers back to default
  Issue an end of engineering_task alert (DA_EOT_ENG).
  If the task_id is still ENGINEERING, set it to IDLE and return.
  Otherwise accept the new value.
```

NAME:

rtrd_task

```
PURPOSE:
     This is a placehoder task for the Real−Time Raw−Data Mode.

CALLING SEQUENCE:
     rtrd_task();

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
     execute (sunload)   [whitedsp.c (this file)]

CALLED FROM:
     whitedsp            [whitedsp.c (this file)]

RESTRICTIONS:

MODIFICATION HISTORY:
     28Oct98   CH    Major updates on in−code comments


  Run the su_engineering swap unit.
```

```
NAME:
```

switch_ping_pong function

```
 PURPOSE:
      Switches the current Blue frame memory
      from ping to pong or from pong to ping.

 CALLING SEQUENCE:
      switch_ping_pong();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      q_alert                   [sun_util.c, cwhite.c]
      sci_command               [sun_util.c, white.asm]

 CALLED FROM:
      acquire_field_task        [whitedsp.c (this file)]
      choose_guide_stars_task   [whitedsp.c (this file)]
      track_guide_stars_task    [whitedsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      17May95   ADW   add standard comment header, structured English
comments
      28Oct98   CH    Major update on in-code comments

   Swap the addresses in crnt_frame
   (the current Blue frame in small word memory area, either ping or
pong)
   and prev_frame (the previous Blue frames's area, either pong or ping)
   Set variable switch_to_frame to Ping or Pong memory area
   depending on the value of crnt_frame (Ping -> Pong; Pong -> Ping)
   Tell Blue DSPs to switch Blue small word memory area
   used to area defined by switch_to frame
..........................................................
If the GLOBAL_ACCESS_C flag is set to 1,
include the file global_access.c in the flight code
```

cwhite.c

```
NAME:
```

cwhite.c

```
PURPOSE:
     Basic White DPU functions that are easiest implemented in C.

FUNCTIONS IN THIS FILE:
     q_alert
     priority_queue
     data_queue
     init_queue
     data_queue_stat          not used ?
     flush_queue
     pop_que                  used only by asm codes
     data_q_full
     check_bus                called by return_bus
     return_bus               all C calls commented
     dflt                     ISR Stub for interrupt B
     do_F0
     do_30
     do_20
     do_10
     do_00
     command_inter            called by white.asm
     main

INCLUDE FILES:
     setup.h
     cwhite.h

MODIFICATION HISTORY:
               JMK
     26Sep95   ADW   add standard comment header
     081297      jrmk  Hamming update
     121297      jrmk  addition of powerdown command
                 fix load local ram
                 implement dump program ram

RCS $Id: cwhite.c,v 1.46 1999/09/20 05:16:26 ho Exp $
```

```
NAME:
```

q_alert

```
PURPOSE:

CALLING SEQUENCE:
     q_alert(type, length, data)

INPUTS:
     type
     length
     data

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
     __asm

CALLED FROM:
     data_queue              [cwhite.c (this file)]
     do_30                   [cwhite.c (this file)]
     do_20                   [cwhite.c (this file)]
     do_10                   [cwhite.c (this file)]
     do_00                   [cwhite.c (this file)]
     command_inter           [cwhite.c (this file)]
     main                    [cwhite.c (this file)]
     read_proc_int           [global_access.c]
     read_proc_int_array     [global_access.c]
     write_proc_int          [global_access.c]
     write_proc_int_array    [global_access.c]
     read_bw_int             [global_access.c]
     read_bw_int_array       [global_access.c]
     write_bw_int            [global_access.c]
     write_bw_int_array      [global_access.c]
     read_sw_int             [global_access.c]   not used
     read_sw_int_array       [global_access.c]
     write_sw_int            [global_access.c]   not used
     write_sw_int_array      [global_access.c]
     calbgd                  [su_calbgd.c]
     deliver_bluefast_data   [su_deliverdata.c]
     prepare_priority_queue  [su_deliverdata.c]
     deliver_trh_data        [su_deliverdata.c]
     dowindow                [su_dowindow.c]
     getref                  [su_getref.c]
     init_exp                [su_init_exp.c]
     locategs                [su_locategs.c]   calls commented
     makeags_id              [su_makeags_id.c]   calls commented
     restore_stars           [su_restore_stars.c]
     scangs                  [su_scangs.c]

RESTRICTIONS:

MODIFICATION HISTORY:
     10Jun93   JMK   first release
     09Dec93   JMK   updates to support new SSI output routines
     26Sep95   ADW   add standard comment header
```

```
NAME:
```

priority_queue

```
PURPOSE:

CALLING SEQUENCE:
     priority_queue(type, length);

INPUTS:
     type
     length

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:
     deliver_faq_data   [su_deliverdata.c]
     deliver_wdw_data   [su_deliverdata.c]
     deliver_trk_data   [su_deliverdata.c]
     deliver_bfast      [su_deliverdata.c]

RESTRICTIONS:
     1) length must be less than or equal to 512

     2) The user of Priority Queue must take responsibility for not
over-
        writing the data in the priority_q buffer.  An example of the
prefered
        way to do this is in su_deliverdata.c,
        functions: prepare_priority_queue and deliver_faq_data.

        This methode of handling the priority_q buffer assumes that the
user
        will set the PRIORITY_QUEUE_STATUS flag in control_word,
indicating
        ownership of the queue by that particular function. The
        PRIORITY_QUEUE_STATUS flag is cleared (indicating that the data
has
        been transmitted to the ICU) by the DPUOS.

MODIFICATION HISTORY:
     10Dec93   JMK   first release
     26Sep95   ADW   add standard comment header
```

```
NAME:
```

data_queue

```
PURPOSE:

CALLING SEQUENCE:
     data_queue(address, length, type, pipo);

INPUTS:
     DataAddress   = starting address of data block to be enqueued for
output
     DataLength    = number of data words to output
     DataType      = data type of the block to be output
     HeaderAddress =
     HeaderLength  =
     pipo          = unique identifier supplied by the calling routine;
                     used as a key for searches of output queue
performed
                     by data_queue_stat().

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
     queue_alert              [cwhite.c (this file)]

CALLED FROM:
     do_F0                    [cwhite.c (this file)]
     do_20                    [cwhite.c (this file)]
     do_10                    [cwhite.c (this file)]
     deliver_bluefast_data    [su_deliverdata.c]
     deliver_image_data       [su_deliverdata.c]
     deliver_ref_data         [su_deliverdata.c]   not used

RESTRICTIONS:

MODIFICATION HISTORY:
     26Sep95   ADW   add standard comment header
```

NAME:

init_queue

PURPOSE:

CALLING SEQUENCE:
```
     iv = init_queue();
```

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:
```
     main   [cwhite.c (this file)]
```

RESTRICTIONS:

MODIFICATION HISTORY:
```
     26Sep95   ADW   add standard comment header
```

```
NAME:
```

flush_queue

```
PURPOSE:

CALLING SEQUENCE:
     iv = flush_queue();

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:
     do_00   [cwhite.c (this file)]

RESTRICTIONS:

MODIFICATION HISTORY:
     26Sep95   ADW   add standard comment header
```

NAME:

pop_que

```
PURPOSE:

CALLING SEQUENCE:
     iv = pop_que();

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:
     Not used in any of the C functions. Used in white.asm

RESTRICTIONS:

MODIFICATION HISTORY:
     26Sep95   ADW   add standard comment header
```

```
NAME:
```

data_q_full

```
PURPOSE:

CALLING SEQUENCE:
     iv = data_q_full();

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:
     do_00   [cwhite.c (this file)]

RESTRICTIONS:

MODIFICATION HISTORY:
     26Sep95   ADW   add standard comment header
```

NAME:

dflt

```
PURPOSE:

CALLING SEQUENCE:
     dflt();

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:
     THIS FUNCTION IS NOT CURRENTLY IN USE - ADW 18Oct95
     No, white.asm calls it. CH18Dec96

RESTRICTIONS:

MODIFICATION HISTORY:
     26Sep95   ADW   add standard comment header
```

NAME:

do_F0

```
PURPOSE:

CALLING SEQUENCE:
     do_F0(ep);

INPUTS:
     ep

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
     write_command_int_array
     data_queue                [cwhite.c (this file)]

CALLED FROM:
     command_inter             [cwhite.c (this file)]

RESTRICTIONS:

MODIFICATION HISTORY:
     26Sep95   ADW   add standard comment header


     dump white onboard ram to global memory

     Dump Program storage to ICU
```

NAME:

check_fec_error

```
PURPOSE:

CALLING SEQUENCE:
     check_fec_error();

INPUTS:
     nothing

OPTIONAL INPUT PARAMETERS:

OUTPUTS:
     null

INCLUDE FILES:

SUBROUTINES CALLED:
     read_proc_int_array
     write_proc_int
     q_alert

CALLED FROM:
     idle_task             [whitedsp.c]
     track_guide_star_task             [whitedsp.c]
     flush_compress_task             [whitedsp.c]

RESTRICTIONS:

MODIFICATION HISTORY:
     10APr00     CH    First implementation


 i = do_red_scrub;
 write_proc_int( (dpucfg+DO_SCRUB), &i);
 Read in the FEC error pointer array
 Ready to loop through the error pointer array
 Loop through the three sets, blue1, blue2, and red
   If there is a mismatch in the current error and reported
   error pointer, then there is a new FEC error.
     Make sure the no of new FEC errors is not negative
     Find out which one we have already sent out
     Loop through all of the new errors
       Set the index to a new FEC error
       Make sure we don't overrun 16 entries
       Grab the error message
       Get the upper 8 bits of the error message
       If the error-nature indicator is 1, single bit error!
           Set up the alert buffer and send it out
           Set up buffer and send out message for multiple bit error
```

```
NAME:
```

do_30

```
PURPOSE:

CALLING SEQUENCE:

INPUTS:
     ep

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:
     command_inter   [cwhite.c (this file)]

RESTRICTIONS:

MODIFICATION HISTORY:
     09May96  CH  created after reorganization

     Initialize DPU
     Initialize exposure, set bluedsp into 1kx1k binned by MIC by 2
     Acquire Field Command
     Choose Guide Stars Command
     Track Guide Stars Command
     Change Task ID to reflect request for Abort
     Change Task ID to reflect request for Flush_Compress
```

```
NAME:
```

do_20

```
 PURPOSE:

 CALLING SEQUENCE:

 INPUTS:
      ep

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      q_alert         [cwhite.c (this file)]
      data_queue      [cwhite.c (this file)]
      write_command_int_array
      __asm

 CALLED FROM:
      command_inter   [cwhite.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      17Oct95   ADW   add standard comment header

      Set Frame Time Command
      Set Exposure Time
      assumes a 24 bit value in 2 16 bit words
      configure guide star selection criteria
      IC_LOAD_GS

   if (ep == IC_LOAD_EXP_CONF)

      load filter configuration
      load memory window
      load science window
      IC_ENBL_VERBOSE

   if (ep == IC_ENBL_MIC_NORM)
      Enable Mic Normalization

      load detector windows
```

```
NAME:
```

do_10

```
 PURPOSE:

 CALLING SEQUENCE:

 INPUTS:
      ep

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      q_alert         [cwhite.c (this file)]
      data_queue      [cwhite.c (this file)]
      hdr_manager     [whitedsp.c]
      __asm

 CALLED FROM:
      command_inter   [cwhite.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      17Oct95   ADW   add standard comment header


      load alert with hi and low data from "address"

      3 word header
            1) index
            2) address
            3) Length

      Disable GlobAL BUS interrupts
      Enable GlobAL BUS interrupts

        sci_command((int)BLUE1_SCI_ADDRESS, (int)DUMP_LOCAL_RAM);

        sci_command((int)BLUE2_SCI_ADDRESS, (int)DUMP_LOCAL_RAM);

        sci_command((int)RED_SCI_ADDRESS, (int)DUMP_LOCAL_RAM);

      enable value in local memory

      Dump RAM to ICU and zero in the process
      load table in global memory
      Dump RAM to ICU
      load Global RAM (BigWord) with opcodes from "address"
```

NAME:

do_00

PURPOSE:

CALLING SEQUENCE:

INPUTS:
```
     ep
```

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
```
     __asm
     flush_queue     [cwhite.c (this file)]
     q_alert         [cwhite.c (this file)]
     sci_command
```

CALLED FROM:
```
     command_inter   [cwhite.c (this file)]
```

RESTRICTIONS:

MODIFICATION HISTORY:
```
     17Oct95   ADW   add standard comment header

     Reset DSP DSP
     there is no return because we will not be coming home


     call assembly code to load - go flight code
     increase wait states (slow ROMs)
     there is no return because we will not be coming home
     enable load code
     Change variable romis to point to the requested eeprom
     load code

     Select Klingon DPU
     turns on DSP

     enable/disable events on dsp "DSP"
     Direct data command (to Blue1 or Blue2 or both)
     Generate A Bad Command ID

     send an sci command
     Request Data Command
     Synch DPU time to S/C time
     Flush Data Queue
```

NAME:

command_inter

```
PURPOSE:

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
     q_alert   [cwhite.c (this file)]
     do_F0     [cwhite.c (this file)]
     do_20     [cwhite.c (this file)]
     do_10     [cwhite.c (this file)]
     do_00     [cwhite.c (this file)]

CALLED FROM:
     white.asm global bus interrupt handler

RESTRICTIONS:

MODIFICATION HISTORY:
     17Oct95   ADW   add standard comment header
```

```
NAME:
```

delayed_q_alert

```
PURPOSE:

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
     q_alert                   [sun_util.c, cwhite.c]

CALLED FROM:
     whitedsp                  [whitedsp.c (this file)]
     choose_guide_stars_task   [whitedsp.c (this file)]
     engineering_task          [whitedsp.c (this file)]

RESTRICTIONS:

MODIFICATION HISTORY:
     27Mar96   ADW   add standard comment header


NAME:
     hdr_manager         function

PURPOSE:
     hdr_manager is used to add the DPU header

CALLING SEQUENCE:
     integer_variable = hdr_manager (hdr, hdr_length);

INPUTS:
     hdr
     hdr_length

OPTIONAL INPUT PARAMETERS:

OUTPUTS:
     none (returns base address for header)

INCLUDE FILES:

SUBROUTINES CALLED:
     write_proc_int_array   [sun_util.c; global_access.c]

CALLED FROM:
     do_10                  [cwhite.c]
     compress               [su_compress.c]
     deliver_bluefast_data  [su_deliverdata.c]
     deliver_image_data     [su_deliverdata.c]
     deliver_ref_data       [su_deliverdata.c]
     deliver_trh_data       [su_deliverdata.c]

RESTRICTIONS:

MODIFICATION HISTORY:
     17May95   ADW   add standard comment header
```

```
NAME:
```

main

```
PURPOSE:

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
     init_queue   [cwhite.c (this file)]
     q_alert      [cwhite.c (this file)]
     whitedsp     [whitedsp.c]

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
     17Oct95   ADW   add standard comment header


  Send Jim From The Rim (JFTR).
  Call the Flight software manager.
  Send Fred From The Dead (FFTD).


  Send Jim From The Rim (JFTR).
  Call the Flight software manager.
  Sit in an infinite loop while we wait for commands to start.
  If we ever get to this point in the code we are in trouble.
  Send a flight code dead message.
  Sit in an infinite loop while we wait for commands to start.
```

Red

```
NAME:

     reddsp.c

PURPOSE:

FUNCTIONS IN THIS FILE:
     main
     reddsp
     red_abort_task
     red_initialize_task
     red_accumulate_image_task
     red_rtrd_task
     saa
     align
     accumulate
     add_frame_buffer
     saa_y
     send_dp_data
     signal_white

INCLUDE FILES:
     setup.h
     shared_variables.c      (simulation only)
     sun_util.c              (simulation only)
     shared_io_functions.c   (simulation only)
     global_access.c         (flight only)

MODIFICATION HISTORY:
     21Sep95   ADW   effectively new...

RCS $Id: reddsp.c,v 1.26 1999/08/21 06:23:45 ho Exp $
```

```
NAME:

      main

 PURPOSE:
      Main function used when running in simulation on a UNIX system
(SUN)
      to emulate the operating system

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      global_ram_io      [shared_io_functions.c]
      red_local_ram_io   [shared_io_functions.c]
      reddsp             [reddsp.c (this file)]

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      21Sep95   ADW   effectively new...

   Read the global and local RAM files into memory.
   Print out diagnostic information.
   Run the reddsp function.
   Print out diagnostic information.
   Write the global and local RAM files.
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::
When running on the DPU itself,
include global_access.c if its flag is set.
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::
```

```
NAME:

      reddsp

 PURPOSE:

 CALLING SEQUENCE:
      reddsp();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_proc_int              [sun_util.c, global_access.c]
      red_abort_task              [reddsp.c (this file)]
      red_initialize_task         [reddsp.c (this file)]
      red_accumulate_image_task   [reddsp.c (this file)]

 CALLED FROM:
      main                        [reddsp.c (this file)]
      main                        [cred.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      21Sep95   ADW   Begin changes for Red DSP handling shift-and-add



      Check the Red DSP task ID and execute the indicated task.
         When reddsp is idling in simulation mode
         just return (to exit from the program).
         When reddsp is idling on the DPU, continue looping.
```

```
NAME:
```

red_idle_task

```
PURPOSE:

CALLING SEQUENCE:
     red_idle_task();

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
     scrub()

CALLED FROM:
     reddsp          [reddsp.c (this file)]

RESTRICTIONS:

MODIFICATION HISTORY:
     12Sep98  CH     started


Now call the JMK scrubbing routine
          If multiple bit error, imcrement red_error for prompt
reporting
      Move to the next block for scrubbing
      move ahead
      cycle back if >= 256
      reset the scrub address from the loader_lut
```

NAME:

red_abort_task

```
 PURPOSE:

 CALLING SEQUENCE:
      red_abort_task();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_proc_int  [sun_util.c, global_access.c]

 CALLED FROM:
      reddsp          [reddsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      24Aug94   SH    created
      21Sep95   ADW



   There is nothing "special" to do when Red aborts. (not yet?)
   For now just set the Red task ID to IDLE and write it to PROC.
```

```
NAME:

      red_initialize_task

 PURPOSE:

 CALLING SEQUENCE:
      red_initialize_task();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_proc_int  [sun_util.c, global_access.c]
      read_proc_int   [sun_util.c, global_access.c]

 CALLED FROM:
      reddsp          [reddsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      03Oct95   ADW   new



   Assign values to Red DSP local RAM variables here.
   Initialize dpucfg, the base address for the PROC section
   of global RAM.  Keep this the same in su_initialize.c
   Get needed values from PROC memory.
   Get memory map for scrubbing.

   Set the Red DSP task ID to idle and return.
```

```
NAME:

      red_accumulate_image_task      function

 PURPOSE:
  This task updates the images accumulating in all science windows
  designated as image mode windows (mode = 0).  Images extracted from
  the tracking frame most recently completed (prev_frame) are aligned
  with the reference frame, then binned if necessary (according to
  param1 and param2 of the science window configuration), and added
  to the corresponding accumulating images.

  If, as a result of drift by an amount greater than that anticipated
when
  the windows were configured, the science window drifts beyond the edge
of
  a memory window, some degradation will occur at the edge of the
accumulating
  image.  For this reason, the science windows should be embedded in
memory
  windows surrounded by a border at least as wide as the estimated
maximum
  drift in the exposure time (nominally 1000 s); in this case the image
quality
  will be uniform in all science window pixels
  (i.e., same cumulative integration time per pixel).

      Prerequisites:
               calc_deltax - from su_caldrft
               calc_deltay - from su_caldrft
               crnt_accum - base address of current accum area
               prev_frame - base address of previous frame area
               keep_a_log - yes or no?
               red_log_indx - address of current place in log

      Provides:      crnt_accum - updated

      Global RAM access:
          read 258 words of window configuration data (~2 DPU cycles)
          read Npix words of tracking frame data where
            Npix is the total number of detector pixels in all image
mode
            science windows (not the bigger memory windows)
          read and write a number of words equal to Npix/B, where
            the pixel binning factor
            B = (1 << bin_factor_x) * (1 << bin_factor_y).
          => total words = 258 + Npix * (1 + 2/B) and
             total DPU cycles ~ total words/256
             (assuming 256 words_per_bus_access).

          NOTE: an upper limit on Npix is implied by the fact that the
             bus access time cannot be greater than the tracking frame
             interval.  Assuming a 10 s frame time,
Npix<2,560,000/(1+2/B)
             => Npix < 850,000 if pixels not binned
             => Npix < 1,700,000 if (2 x 2) binning
             (these estimates do not take into account computational
overhead,
             nor do they allow for the fact that the White DSP has to
run the
             TRACK_GUIDE_STARS task too during each tracking frame).

      DSP RAM required:   ~830 variables

 CALLING SEQUENCE:
```

```
      red_accumulate_image_task();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      read_proc_int    [sun_util.c, global_access.c]
      saa              [reddsp.c (this file)]
      write_proc_int   [sun_util.c, global_access.c]

 CALLED FROM:
      reddsp           [reddsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      21Sep95   ADW   taken from whitedsp.c, now that Red DSP does this
task.



      red_task_id = RED_IDLE_TASK;
   Get updated values for several variables from PROC memory.
   Those read here are used in more than one function below.
   Those needed in only one function are read when needed.
   Run the saa function to perform the shift-and-add process
   of shifting the Blue image mode science windows by the amount
   determined by the tracking process and accumulate them into the
   Blue image mode exposure frame.

   goto skip_saa;
```

```
NAME:

      saa              function

 PURPOSE:

 CALLING SEQUENCE:
      saa();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      none   (returns 0)

 INCLUDE FILES:

 SUBROUTINES CALLED:
      align                        [reddsp.c (this file)]
      accumulate                   [reddsp.c (this file)]
      read_proc_int                [sun_util.c, global_access.c]
      read_proc_int_array          [sun_util.c, global_access.c]
      round                        [macro.h]
      write_proc_int               [sun_util.c, global_access.c]
      global_zero                  [reddsp.c (this file)]

 CALLED FROM:
      red_accumulate_image_task    [reddsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      24Dec92 DTL   original code for DPU
      04Jan93 DTL   read_saa_int_array() replaces
                    read_sw_int_array() (see note in code)
      04Feb93 DTL   added file open/close calls for Sun description of
the file
      19Feb93 DTL   replaced read_saa_int_array() with
                    read_sw_int_array() and zero_frame()
WHERE?? ADW
      04Apr93 DTL   change "swap_id" to shared variable "swap_to"
      25Aug94 SDH   add standard comment headers & structured English
comments




.....................................................................
..
   Make entry in log (optional)

   Read science window configuration data from global RAM.
   Read memory window configuration data from global RAM.
   Round shift for drift compensation to the nearest pixel.
   Loop through the science windows.
      If the window is an image mode science window then:
         load a pointer to its memory window,
         get the y size of the memory window from the mmw structure,
         get the pixel binning factors from the scw structure,
         calculate the actual binning factors,
         run the align function to calculate base addresses of frame
         and accumulating images, shifting to compensate for drift,
         and run the accumulate function to add
         the window to the accumulating image.
```

Reinitialize previous frame to all zeros.

Zero out the memory area occupied by the frame just added.

```
NAME:

      align

 PURPOSE:
  This function takes into account the (x, y) translation of the most
  recently integrated tracking frame relative to the reference frame
  and calculates orig_frame_base, the base address of the bottom left
  frame pixel (relative to prev_frame), and orig_accum_base, the base
  address (relative to crnt_accum) of the corresponding bottom left
  binned pixel in the accumulating image that will yield alignment of
  the image in a science window with its location in the reference
  frame.  This function also provides values for the global variables
  accum_y_dim, accum_x_max, and accum_y_max which are used in
  add_frame_buffer(), respectively, to locate an accumulation subimage
  in global RAM, and to give the number of columns and rows of binned
  pixels to be updated in the accumulation image.

 CALLING SEQUENCE:
      align(shift_in_x, shift_in_y, scw, mmw);

 INPUTS:
      shift_in_x      calculated x drift rounded to nearest pixel
      shift_in_y      calculated y drift rounded to nearest pixel

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_proc_int   [sun_util.c, global_access.c]

 CALLED FROM:
      saa              [reddsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      25Aug94   SDH   add standard comment headers, structured English
comments
      18May95   ADW   work on comment headers & coding standards



      These offsets are non-zero only if the drift is greater than
      the size difference between the science window and the memory
      window in the y dimension
.................................................................
..
   Make an entry in the log (optional).

      (void) write_proc_int(red_log_indx, &bus_cycle_temp);
      red_log_indx += SIZEOF_BIG_WORD;
      (void) write_proc_int(red_log_indx, &su_saa_log2);
      red_log_indx += SIZEOF_BIG_WORD;
      if (red_log_indx > red_end_of_log) red_log_indx = red_log_base;

   Calculate the x size of the accumulated image.
   If the shift is positive in x, then
      set the x offset to the drift amount,
      calculate the minimum number of pixels
      between the edge of the science window and the
```

```
      edge of the memory window in the x dimension,
      and set the number of (binned) pixels to fill.
         If the drift is greater than the size difference
         between the science window and the memory window
         in the x dimension, account for that fact.
   If the shift is negative in x, then
      calculate the minimum number of pixels
      between the edge of the science window and the
      edge of the memory window in the x dimension,
      and set the offset and number of pixels to fill,
      accounting for the possibility that the window
      may have drifted out of its memory window.
   If there is no drift in x
      set the offsets to zero and the number of pixels to the accum
size.
   Account for the drift in y in like manner.
   Calculate base addresses for alignment.
   Calculate memory address of the pixel of the current,
   frame, science window that is in the lower left hand corner
   of the accumulation image window
   only non-zero if the drift is greater than the size difference
   between the science window and the memory window
```

```
NAME:

      accumulate

 PURPOSE:
  This function adds the aligned tracking frame image to the correspond-
  ing accumulating image.  Because the memory available locally to the
  WHITE DSP cannot accomodate all the image data, the images are divided
  into subimages of dimensions BUFFER_X_DIM x BUFFER_Y_DIM.  Those di-
  mensions are tailored to the DPU in the sense that all of the pixels
  in a subimage can be read from or written to global RAM in 1 DPU
cycle.
  If the accumulating image contains binned pixels, multiple frame sub-
  images are processed for each accumulation subimage until the entire
  accumulation subimage has been updated.  Then the updated accumulation
  subimage is replaced in global RAM.  The frame and accumulation
  images are raster-scanned with subimages, first in y, then in x.
  If the number of rows (columns) to be filled in the  accumulating
image
  is not evenly divisible by BUFFER_Y_DIM (BUFFER_X_DIM), then the last
  subimage in y (x) will have a dimension smaller than the nominal
  BUFFER_Y_DIM (BUFFER_X_DIM); any residual in the (unbinned) frame
  pixels is treated analogously.  The actual summation of a frame sub-
  image (data in frame_buffer[]) into the accumulating subimage
  (data in accum_buffer[]) is performed in the function
  add_frame_buffer().

 CALLING SEQUENCE:
      accumulate();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      add_frame_buffer     [reddsp.c (this file)]
      read_bw_int_array    [sun_util.c, global_access.c]
      send_dp_data         [reddsp.c (this file)]
      write_bw_int_array   [sun_util.c, global_access.c]
      write_proc_int       [sun_util.c, global_access.c]

 CALLED FROM:
      saa                  [reddsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      25Aug94   SDH   add standard comment headers, structured English
comments
      18May95   ADW   fix up comments & impose coding standards


   Do a raster scan of the accumulation image, stepping in x and y.
   Loop over the possible x values in the accumulation image window.
      Calculate the number of columns in the accumulation
      image window remaining to be read.
      Set accum_xsize to the smaller of the x dimension of the local
      memory buffer or the number of columns remaining to be read.
      @@@ There has got to be a better way of doing this than fixing
      buffer_x_dim to 4 in shared_defines.h.  What if you have a window
      that is 64x8?  This method may work fine for the limited selection
```

```
      of window configurations we have tested to date, but will be very
      inefficient for others.
      Determine the size of the unbinned frame subwindow window.
      Loop over the possible y values in the accumulation image window.
         Repeat above calculations for y direction.
            "%d out of %d in x; %d out of %d in y; %d %d %d %d\n",
            accum_blc_x, accum_x_max, accum_blc_y, accum_y_max,
            accum_xsize, accum_ysize, frame_x_max, frame_y_max);
            accum_base);
         Read the accumulation subwindow from the big word memory.
         Cover the accumulation subimage with one or more frame
subimages.
         Loop over x until the number of x pixels processed
         equals the unbinned frame subwindow x size.
            Calculate the x dimension of frame subimage.
            Loop over y until the number of y pixels processed
            equals the unbinned frame subwindow y size.
               Calculate the x dimension of frame subimage.
                  "  %d out of %d in x; %d out of %d in y; %d %d\n",
                  frame_blc_x, frame_x_max, frame_blc_y,
                  frame_y_max, frame_xsize, frame_ysize);
               Add the frame subimage to the accumulation subimage.
         Replace accumulation subwindow in global RAM
```

```
NAME:

      add_frame_buffer

 PURPOSE:
      This function adds one frame subimage (loaded from global RAM into
      the frame_buffer[]) into the accumulation subimage
(accum_buffer[]).
      If the accumulating pixels are not binned, one call to this
function
      will fill the accum_buffer[]; otherwise, this function is called
      several times by accumulate() to fill the accum_buffer[] before
the
      accumulation subimage is replaced in global RAM.

 CALLING SEQUENCE:
      add_frame_buffer(frame_x0, frame_y0, frame_xsize, frame_ysize,
          accum_ysize);

 INPUTS:
       frame_x0    = bottom left x coordinate of frame subimage relative
to
                     the bottom left corner of the accumulation subimage
       frame_y0    = bottom left y coordinate of frame subimage relative
to
                     the bottom left corner of the accumulation subimage
       frame_xsize = x dimension of the frame subimage
       frame_ysize = y dimension of the frame subimage
       accum_ysize = y dimension of accumulation subimage

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      write_proc_int      [sun_util.c; global_access.c]
      read_proc_int       [sun_util.c; global_access.c]
      read_sw_int_array   [sun_util.c; global_access.c]
      saa_y               [reddsp.c (this file)]

 CALLED FROM:
      accumulate

 RESTRICTIONS:
      None

 MODIFICATION HISTORY:
      25Aug94   SDH   add standard comment headers, structured English
comments


   Make an entry in the log (optional).

   Fetch the frame subimage from global RAM.
   Read the current subwindow from the frame into the frame buffer.

   Add the frame subimage to the accumulation subimage.
   Copy the pointer to the current frame subwindow to fbuf_ptr.
   Loop over the current subwindow x pointer
   from minimum to maximum x of the frame subwindow.
      Calculate the accumulation x pointer given the x binning factor.
      Establish the accum pointer at the bottom of current row.
      Run saa_y, the top level workhorse.
```

```
Increment the frame buffer pointer value by the frame size.
```

```
NAME:

      saa_y

 PURPOSE:
      This is the top level workhorse, calls y_hopper_0, y_hopper_1, or
      y_hopper_2 to do the dirty work. The y_hopper_? are coded in
      assembly for efficiency.  The assembly versions of y_hopper_?
      are in white.asm.  The C versions, used for the simulations,
      are in sun_hopper.c.

 CALLING SEQUENCE:
      saa_y( accum_ptr, fbuf_ptr, frame_ysize, frame_y0);

 INPUTS:
      accum_ptr   = pointer to the address at the bottom
                    of the first column in the frame segment
      fbuf_ptr    = pointer to the frame buffer
      frame_ysize = y dimension of the frame
      frame_y0    = bottom left y coordinate of frame subimage relative
to
                    the bottom left corner of the accumulation subimage

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      y_hopper_0         [sun_hopper.c, white.asm]
      y_hopper_1         [sun_hopper.c, white.asm]
      y_hopper_2         [sun_hopper.c, white.asm]

 CALLED FROM:
      add_frame_buffer   [reddsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      25Aug94   SDH   add standard comment headers, structured English
comments



   If binning in y dimension is by 1 (bin_factor_y = 0), then
      Put
      Run y_hopper_0 function to
   Else the binning in y is not 1 (is 2 or 4)
      Line up the accum pointer
      Take care of extra head
         Add up the head
      Check for extra tail.
      Add up the bulk of the data with customized code

      izero = accum_ptr - accum_buffer;
      izero = bin_factor_y;
      izero = abin_factor_y;

      If binning in y dimension is by 4 (bin_factor_y = 2), then
         Run y_hopper_2
         Run y_hopper_1
      do the pointer arithemetics here!
      Add extra tail
```

```
NAME:

      red_rtrd_task       function

 PURPOSE:

      Prerequisites:

      Provides:

      Global RAM access:
          read 258 words of window configuration data (~2 DPU cycles)
          read Npix words of tracking frame data where
            Npix is the total number of detector pixels in all image
mode
            science windows (not the bigger memory windows)
          read and write a number of words equal to Npix/B, where
            the pixel binning factor
            B = (1 << bin_factor_x) * (1 << bin_factor_y).
          => total words = 258 + Npix * (1 + 2/B) and
             total DPU cycles ~ total words/256
             (assuming 256 words_per_bus_access).

          NOTE: an upper limit on Npix is implied by the fact that the
             bus access time cannot be greater than the tracking frame
             interval.  Assuming a 10 s frame time,
Npix<2,560,000/(1+2/B)
             => Npix < 850,000 if pixels not binned
             => Npix < 1,700,000 if (2 x 2) binning
             (these estimates do not take into account computational
overhead,
             nor do they allow for the fact that the White DSP has to
run the
             TRACK_GUIDE_STARS task too during each tracking frame).

      DSP RAM required:   ~830 variables

 CALLING SEQUENCE:
      red_rtrd_task();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:
      read_proc_int    [sun_util.c, global_access.c]
      saa              [reddsp.c (this file)]
      write_proc_int   [sun_util.c, global_access.c]

 CALLED FROM:
      reddsp           [reddsp.c (this file)]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      21Sep95   ADW   taken from whitedsp.c, now that Red DSP does this
task.
```

NAME:

send_dp_data ***function***

PURPOSE:

CALLING SEQUENCE:

```
send_dp_data( data_type, data, length);
```

INPUTS:

```
data_type
data
length
```

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

```
read_proc_int    [sun_util.c, global_access.c]
write_proc_int   [sun_util.c, global_access.c]
priority_queue   [sun_util.c, cred.c]
```

CALLED FROM:

```
accumulate       [reddsp.c (this file)]
```

RESTRICTIONS:

MODIFICATION HISTORY:

```
18May95   ADW   add standard comment header
```

```
Block out the priority queue.
DPUOS should clear the queue when done.
```

NAME:

signal_white

PURPOSE:
Alert the White DSP, via PROC memory, what Red DSP task is in progress.

CALLING SEQUENCE:
```
signal_white(current_red_task);
```

INPUTS:
current_red_task =

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
write_proc_int [sun_util.c, global_access.c]

CALLED FROM:
*** no longer used ***

RESTRICTIONS:

MODIFICATION HISTORY:
21Sep95 ADW

If running in simulation, declare the external integer *global_ram.

```
NAME:
      compress function

 PURPOSE:
      This function drives the vbtwl compression

 CALLING SEQUENCE:
      swap_to = compress();

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      return status    compress_status

 INCLUDE FILES:

 SUBROUTINES CALLED:
      read_proc_int_array
      read_sw_cmp_block
      read_bw_cmp_block
      vbtwl
      write_cmp_block

 CALLED FROM:
      sunload   [whitedsp.c]

 RESTRICTIONS:

 MODIFICATION HISTORY:
      01May96   CH    start from scratch


   Declare the DPU integer variables.
   Set the compression status word to idle
   if there are no entries on the compression queue.
      **** Notice stripping off the highest bit CH *
```

NAME:

read_cmp_block

PURPOSE:

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
```
     12Jun96   ADW   add standard comment header
```

```
NAME:

      write_cmp_block

 PURPOSE:

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header


  This file contains the Variable Block Tiered Word Length compression
engine.

  Requires:

  vbtwl.h:      definition for structural information
  vbtwl.def:    definition of global variables

  Cheng Ho
  Original scheme 1990, developed for the XMMOM project.
  Version 0.x, 1991, 1992, 1995
  Ver. 1.0  April, 19, 1996

  Copyright, 1996



  This function is the overall image compression routine to be
  performed by the VBTWL.

  This file is self-contained. But it can be broken into two major
  parts, in addition to the vbtwlcmprss calling routine: optvbtwl.c
  and compress.c.
```

```
NAME:

      vbtwl

 PURPOSE:

 CALLING SEQUENCE:

 INPUTS:
      rawint *block     array of raw data to be compressed.
      int    blockid    id of block designated by the DPU.
      int    blocksize  number of elements in the block
      int    cmpmode    binary-coded instruction from the DPU to ICU on
                        which algorithm to use to compress the data

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:
      cmpshort *cstr    pointer to the compressed data string. The size
                        of the template string MUST be defined and
                        allocated by the calling function.
      int      *imm     the selected predictor algorithm

      int vbtwl() returns the number of bits in the compressed string.

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header



**** Need to explicitly initialize these arrays for DSP implementation
!! **
                          Find the optimal compression parameters
                           Do the compression


  Section 1: Determination of the optimal vbtwl prediction and
             encoding scheme

  The following modules determine the optimal compression parameters.
```

```
NAME:

      optvbtwl

 PURPOSE:
      This function determines the ``optimal'' tiering structure for a
      block of image of size (isize*jsize) whose lower left corner is
      located at (i*isize, j*jsize) of the image.

      Three decorrelation schemes using three different predictors are
      included in the consideration. The predictors are 0. the minimum,
      1. the mean and 2. the previous twixel count. The word length are
      divided into three tiers. See Doc3 for more disciusison. At the
      end, this function yields:

      im:         decorrelation/predictor mode (0, 1, 2)
      it1:        number of bits in tier 1.
      it2:        number of bits in tier 2.
      it3:        number of bits in tier 3.
      nlx3:       number of twixels in tier 3.
      nlx2:       number of twixels in tier 2.

      The numbers above needs to be included in the header.

      Other relevant quantities:

      nlx1:       number of twixels in tier 1.
      n_bits:         total number of bits require in this block.

      nlx1+nlx2+nlx3=blocksize=isize*jsize.
      n_bits=blocksize*it1+nlx2*it2+nlx3*it3+nheader.

      nheader is the number of bits in the header. 64 bits appears
      to be a good number with enough redundancy for error checking.

 CALLING SEQUENCE:

 INPUTS:
      n_tier
      *block
      blocksize
      cmpmode

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

NAME:

resetlx

PURPOSE:
This function resets the array lx[] to zero. The array lx[] will hold the ``spectral'' information of the block.

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
12Jun96 ADW add standard comment header

```
NAME:

      realignlx

 PURPOSE:
      This function turns the spectrum lx[] from a density distribution
      to an accumulated function.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

NAME:

spectrum0

PURPOSE:
Analyze the current block using the minimum within the block as the
predictor. Yields lx[].

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
12Jun96 ADW add standard comment header

```
NAME:

       spectrum1

 PURPOSE:
      Analyze the current block using the mean within the block as the
      predictor. Yields lx[].

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

NAME:

spectrum2

PURPOSE:
Analyze the current block using the count in the previous twixel as
as the predictor. Yields lx[].

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
12Jun96 ADW add standard comment header

```
NAME:

      predict

 PURPOSE:
      This function initializes the predictor for the decorrelation
scheme specified by im.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      18Oct95   ADW   add standard comment header

      Minimum as predictor
      Mean as predictor
      Previous (cyclic) values as predictor
```

NAME:

msbb

PURPOSE:

```
      Number of bits required to express a SIGNED integer k. Leave one
addition
      slot for the two flags needed. Here, the two flags are −0(it1) and
      −(2^(it1−1)−1). Calls the msb().
```

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:

```
      12Jun96   ADW   add standard comment header
```

```
NAME:

      tiering

 PURPOSE:
      Tiering analysis based on the accumulated spectral distribution
lx[].
      Yields it1, it2, it3, nlx3, and n_bits. Returns n_bits.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

```
NAME:

      msb

 PURPOSE:
      The most significant bit for an UNSIGNED integer. The minimum
number
      of bits is one (1). This is done by properly initializing i and k.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

Section 2. Encoding

The following modules do the actual compression.

```
NAME:

      com_para

 PURPOSE:
      This function calculates the parametric quantities to be used in
      the vbtwl compression. They are the levels separating the tiers,
the
      filters to get the tiered number and the flags which indicates
      the higher tiers.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

```
NAME:

       encode

 PURPOSE:
      This fucntion carries out the compression and produces a string of
      bits. which is (n_bits/8) * 8 int. This will then be used to
either
      be passed to the telemetry requester or a FIFO stack for
telemetry.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

```
NAME:

      reset_cstr

 PURPOSE:
      Reset the template telemetry string.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

NAME:

do_header

PURPOSE:
This function establishes the header. Current thinking is that the header will have 64 bits (8 bytes) containing all sorts of information.

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
12Jun96 ADW add standard comment header

```
NAME:

      encode0

 PURPOSE:
      Encoder for the VBTWL scheme with decorrelation scheme 0
      (minimum as predictor).

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

NAME:

encode1

PURPOSE:
Encoder for the VBTWL scheme with decorrelation scheme 0
(average as predictor).

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
12Jun96 ADW add standard comment header

```
NAME:

       encode2

 PURPOSE:
      Encoder for the VBTWL scheme with decorrelation scheme 2 (previous
value as predictor).

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

NAME:

do_plusminus

PURPOSE:
Processing SIGNED remnant and calling enfilter() to encode the remnant
number n into bit stream s.

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
12Jun96 ADW add standard comment header

```
NAME:

      enfilter

 PURPOSE:
      This function checks which tier the int number n is in.
      The tiering is given by the levels l1 and l2. The number n is
      then passed throguh the appropriate filter (Do we need that?)
      and returned. Returns number of bits in the tirer.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

```
NAME:

      insert0

 PURPOSE:
      Flag inserter into the compressed data steram. Once the flag is
      inserted, calls insert to insert the bit stream n of length k
      at the current bitstream location.

 CALLING SEQUENCE:

 INPUTS:

 OPTIONAL INPUT PARAMETERS:

 OUTPUTS:

 INCLUDE FILES:

 SUBROUTINES CALLED:

 CALLED FROM:

 RESTRICTIONS:

 MODIFICATION HISTORY:
      12Jun96   ADW   add standard comment header
```

NAME:

insert

PURPOSE:
This function inserts a numerial string n of length nbits into the template telemetry string. This function will append a bit string n of length nbits to the template bit stream. The beginning pointer of the template stream is cstring. The insertion starts at the cptr pointer at the rema−th bit.

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:

```
21Jan92   CH
12Jun96   ADW   add standard comment header
```

cred.c

NAME:

cred.c

```
PURPOSE:
     Basic Red DSP functions that are easiest implemented in "c".

FUNCTIONS IN THIS FILE:
     priority_queue    CURRENTLY COMMENTED
            removed by jrmk 120397
     dflt
     main

INCLUDE FILES:
     setup.h
     cred.h

MODIFICATION HISTORY:
     18Oct95   ADW   add standard comment header
     120397      jrmk  unused stuff removed

RCS $Id: cred.c,v 1.11 1999/02/19 06:00:14 jrklark Exp $
```

NAME:

dflt

PURPOSE:
default unused interrupt handler. Will cause system to halt processing.

CALLING SEQUENCE:
```
dflt();
```

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:

CALLED FROM:
red.asm unused interrupt vectors

RESTRICTIONS:

MODIFICATION HISTORY:
18Oct95 ADW add standard comment header

```
NAME:
```

scrub

```
PURPOSE:

CALLING SEQUENCE:
     scrub(int StartingAddress);

INPUTS:
     StartingAddress: Address of first opcode in program storage RAM to
                  scrub. A block of 16 addresses will be read starting
                  at StartingAddress.



OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
                 FfwrdErrCrection, dpu/development/include/FecCode.asm

CALLED FROM:

RESTRICTIONS:
           This routine blocks until execution is complete.

           This routine assumes that caller will not attempt to scrub
           more blocks than can be scrubbed in one (1) global bus time
           slice.

           This routine uses resources that are accessed by
           FfwrdErrCrection. These resources would be modified
           by a call to "load", the swap unit loader.

MODIFICATION HISTORY:
     060998      jrmk  Creation
```

```
NAME:
```

main

```
PURPOSE:

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:

OUTPUTS:

INCLUDE FILES:

SUBROUTINES CALLED:
     reddsp   [reddsp.c]

CALLED FROM:

RESTRICTIONS:

MODIFICATION HISTORY:
     18Oct95   ADW   add standard comment header


  Call the Red DSP flight software manager.
  We should never reach this point
  (reddsp() should never return).
```

Index