# X-Ray Multi Mirror Mission Optical Monitor

# Detailed Design Document for Digital Processing Unit Flight Software: Assembly

Document: XMM-OM/UCSB/ML/0014 Version 1.4

James R. Klarkowski
*Sandia National Labs, Albuquerque*

Cheng Ho
*Los Alamos National Laboratory*

Jamie A. Kennea
*University of California, Santa Barbara*

22 May, 2000

CHANGE RECORD

| ISSUE | DATE | PAGES CHANGED | COMMENTS |
|---|---|---|---|
| 01 | 26-Oct-98 | | Initial Release |
| 02 | 26-Jan-99 | 29 .. 94 | Missing STD.ASM |
| 03 | 23-Sep-99 | All | Reformatted layout |
| 04 | 22-May-00 | | Updated for Rel 10 |

Table of Contents

# Summary

This document provides a functional description of the for Release 10 of the Assembly code routines in the XMM-Newton 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 9/99-5/00" (XMM-OM/UCSB/TC/0050) and "DPU Flight Software test reports for Release 10" (XMM-OM/UCSB/RP/0005).

# Notes

All assembly routines that are C callable are prefixed with a "F", which is not used when the command is called by DPU C code. For example, a assembly routine `Fboot_flight` would be called from C code as `boot_flight()`.

# White.asm Detailed Design Document

```
FILE:
            white.asm
            First cut at white processor assembly code

PURPOSE:
            System initialization
            Interrupt Service Routine Handlers
            Optimized C code
            Interface between C code and Global Resources

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            Fboot_flight
            FfwrdErrCrection
            Fload
            Fsci_command
            Fread_n_zero_sw_int_array
            Fwrite_command_int_array
            F__start
            F__crt0_end

INCLUDE FILES:
            white.h
            asm_sci_com.h
            asm_idiot.h
            asm_white_control.h
            asm_dpucfg_map.h

SUBROUTINES CALLED:
            None.

CALLED FROM:
            ROM based code pointed to by hardware after Reset.

RESTRICTIONS:
            None.
MODIFICATION HISTORY:
      jmk101292   initial creation
       020393     conversion to c based code jmk
       051993     conversion to ssi highest system priority
                  and lots of other little !#@$.
       040194     add mutual exclusion to sci command to prevent
                  command interleaving between science and ISR code
       020795     addition of non-sci heartbeat returns from the
                  red and blue processors jmk
```

```
        070897      removed modulus addressing from ssi receiver
                    interrupt handlers jrmk
        120497      jrmk  added a copy of data initialization before
                    enabling outputs hopefully to avoid any
                    glitches on the control line to the power supply
                    FEC Stuff
        081198      jrmk  fixed race condition in code that loads a SU
                    (two places clearing the bit flag LOAD_A_SU, one in
                    Fload and one in Floader).
        081298      jrmk  added standard wait states reference
                    removed "Prom" wait state changes in FLoader, no longer
                    needed because programs are stored in RAM, not ROM
        102498      addition of standard header   jrmk

        040599      in BUSIO, removed delay between output of SCI Commands
                    was ~4mSec now ~1mSec. See SCIOutRequired.      jmrk

RCS $Id: white.asm,v 1.25 1998/09/23 22:25:47 ho Exp $


******************************************************************
```

```
NAME:
            Boot strap Loader Code

PURPOSE:
            Copy Bootstrap Loader from Global ROM into Local RAM
            for execution.
            Forward Error Correction is not used. ROM storage is immune
            to errors.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            Jumps to address 0000h after code is loaded

INCLUDE FILES:
            None.


SUBROUTINES CALLED:
            None.

CALLED FROM:
            ROM based code pointed to by hardware after Reset.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk


*******************************************************************
      point to starting address of code stored in ROM
      enable access of program storage card
      Load Interrupts
      Load asm code into DSP memory and external memory
      Load C Code into external memory
      disable access of program storage card
```

```
NAME:
            Fboot_flight -- Flight Loader Code

PURPOSE:
            Load code stored in KAL powered Global Program Storage RAM onto Local
            RAM. Provides C callable routine boot_flight().

CALLING SEQUENCE:


INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            FfwrdErrCrection - Forward Error Correction code, uses constants
                               defined in FecTab.asm

INCLUDE FILES:
            FecTab.asm

SUBROUTINES CALLED:
            FfwrdErrCrection

CALLED FROM:
            cwhite.c

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      102398      Standard header added


******************************************************************
      initialize Forward Error Correction table
      enable access of program storage card
      Lookup location of flight BIOS, asm
      starting address in RAM of code to load
      get length of flight bios, asm code
      calculate the number of 16 opcode grabs that need to be done
      load interrupts into DSP memory
      load asm code into DSP memory and on board memory
      Lookup location of flight BIOS, C code
      get starting address in RAM of code to load
      get length of flight bios, C code
      calculate the number of 16 opcode grabs that need to be done
      load C code into board memory
      Lookup location of flight WHITEDSP
      get starting address in RAM of code to load
      get length of flight bios, WHITEDSP
      calculate the number of 16 opcode grabs that need to be done
      load whitedsp code into board memory
      disable access of program storage card
```

```
Start this bad boy!!
```

NAME:

# Definition of interrupt vectors

PURPOSE:

Definition of interrup vectors, used by both Bootstrap and Flight code. Provides interrupt entry point, which consists of a serious of jump functions to the relevant intterupt routine. These entry points are located at the beginning on Local Memory.

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

None.

FUNCTIONS IN THIS FILE:

None

INCLUDE FILES:

None

SUBROUTINES CALLED:

F__start
Stkerr
Trace
Swi
BUSIO
TimeSync
ssirx
essirx
ssitx
essitx
SCIRec
ESCIRec
SCIXmit

CALLED FROM:

None - triggered by interrupt call.

RESTRICTIONS:

None.

MODIFICATION HISTORY:

jmk101292 initial creation

*****************************************************************

Initialize interrupt vectors
Fill remaining unused vectors with dummy call

NAME:

# FfwrdErrCrection -- Forward Error Correction Code

PURPOSE:

The forward error correction code provides checking for possible corruption of the DPU software being loaded into local RAM. If possible errors are fixed.

CALLING SEQUENCE:

jsr FfwrdErrCrection

INPUTS:

NextReadFromAddress

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

0 – No Errors Found
1 – Single Bit Error Found and Fixed
2 – Multi-Bit Error Found, can not fix

FUNCTIONS IN THIS FILE:

INCLUDE FILES:

None.

SUBROUTINES CALLED:

None.

CALLED FROM:

Floader -- this file

RESTRICTIONS:

None.

MODIFICATION HISTORY:

102498 addition of standard header jrmk

*******************************************************************

See FecCode.asm section

```
NAME:
```

# BUSIO -- Global bus interrupt handler

```
PURPOSE:
            Service periodic (~ 1 millisecond) interrupt that indicated opportunity
            to access Global Resources

CALLING SEQUENCE:
            None -- ISR

INPUTS:
            Control Bits in Packed word:
                  Global_bus_flags
                  Global_bus_flags2

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            Fq_alert -- cwhite.c
            Fcommand_inter -- cwhite.c
            SCIHbeatOutput -- this file
            SSIBufferFillRoutine -- this file
            Floader -- this file
            zero_global -- this file
            write_global -- this file
            read_global -- this file

CALLED FROM:
            None - ISR.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk
      040599      removed delay between output of SCI Commands
                  was ~4mSec now ~1mSec. See SCIOutRequired.      jmrk

******************************************************************
      do klingon checks
      if we're not the klingon lets get on with business
      if we're the klingon, is it time to give the bus back?
         White is the Klingon and it's time to give the bus back
         Return Global Bus
         wait a while for the CD4000 parts
      else Time to do some work
       save registers
```

```
 Look for and process incomming Commands
 Check for end of incomming SSI command from the ICU
 Finished command in the queue, HOT Damn
 Ping / Pong incoming command buffers
 End of Command Stuff

increment bus cycle counter
check blue1 communications flags
    Check if BLUE1 is first time. If yes, don't report status
    if there has been a change in the semaphore:
      save a copy,
      tell the world
    if there has been a change in the error:
      save a copy,
      tell the world

    Check if BLUE2 is first time. If yes, don't report status
    if there has been a change in the semaphore:
      save a copy,
      tell the world
    if there has been a change in the error:
      save a copy,
      tell the world

Check red communications flags
    Check if RED is first time. If yes, don't report status
    If there has been a change in the semaphore:
      save a copy,
      tell the world
    If there has been a change in the error:
      save a copy,
      tell the world

If SCI Queue not empty.
   Set flag requesting delay before xmit next SCI message
   turn on transmitter interrupt

Is it time to output a SSI heartbeat?
   call Fque_alert and send out heartbeat

Is it time to output a SCI heartbeat?
   provide mutial exclusion to prevent command
   interleaving between science code and ISR code
   call sci_command() to set up for next SCI Heartbeat

is it time to fill an SSI output buffer?
   yes, call SSI buffer fill routine

if it's time to load a SwapUnit lets do it
   to make sure we finish FECing previously loaded
   OpCodes, disable su_loading until current load
   has been FECed this is done by using COPY_OF_LOAD_A_SU
   to disable calls to Floader in the BUSIO ISR.

Check for requests to zero buffer area in the global RAM
   if true call zero global routine

Check the write global memory queue to see if we need to output some
stuff for those scum end users
   if true call write_global
```

```
Check the read global memory queue to see if we need to input some
stuff for the ungrateful end users.
   if true call read global

If (white is not the Klingon)
   wiggle Return Global Bus Bit

restore registers

End of BUSIO interrupt handler
```

```
NAME:
```

# Fload -- Load a swap unit

```
PURPOSE:
            Loads, forward error corrects, and executes a module of C code

CALLING SEQUENCE:
            Fload(int numberOfSwapUnitToLoad)

INPUTS:
            numberOfSwapUnitToLoad -- indicates which swap unit to load.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            FfwrdErrCrection -- this file
            Floader -- this file

CALLED FROM:
            whitedsp.c

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk


*******************************************************************
      Load a swap unit
         set up the required stuff to preform the load from global memory.

         Then wait for an interrupt service routine to load up a block of
         Program Storage RAM.

         Once it has it will set the FEC_PROG_BLOCK bit in
         Fglobal_bus_flags2 word.

         This indicates the opcodes and Forward error correction bits
         have been loaded into memory and require checking.

         When finished move the code so the FEC words are written over
         by opcodes.

         to make sure we finish, disable su_loading until
         this load has been FECed this is now done by using
         COPY_OF_LOAD_A_SU to disable calls to Floader in the BUSIO ISR.
         move the opcodes into the FEC incomming buffer.
```

call Forward Error Correction

now that we have finished see if we need to enable SU loading

```
NAME:
            Floader --

PURPOSE:
            called from BUSIO ISR, this routine access the KAL
            powered Global Program Storage RAM card and moves
            code from there into local memory.

CALLING SEQUENCE:
            jsr Floader

INPUTS:
            rom_table_vector -- pointer to:
                        address of where to load from
                        number of words to load

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            BUSIO -- this file

RESTRICTIONS:
            Assumes access to the Global Bus.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk


*****************************************************************
      enable access of program storage card
      get starting address in RAM of code to load
      read 4 16 word blocks
      disable access of program storage card


*****************************************************************
```

NAME:

# SSIBufferFillRoutine -- SSI Transmitt ping/pong buffer fill routine

PURPOSE:

Fills SSI ping/pong transmit buffers with either data stored in Global 24 bit or 16 bit (Big word / Small Word) RAM cards, or alert messages stored locally. The SSI interface is the communications link between the ICU and the DPU.

CALLING SEQUENCE:

jsr SSIBufferFillRoutine

INPUTS:

Control bits stored in the packed word Fglobal_bus_flags.

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

None.

FUNCTIONS IN THIS FILE:

None.

INCLUDE FILES:

None.

SUBROUTINES CALLED:

Fdata_q_full -- cwhite.c
Fq_alert -- cwhite.c
Fpop_que -- cwhite.c

CALLED FROM:

BUSIO -- this file

RESTRICTIONS:

Assumes access to the Global Bus.

MODIFICATION HISTORY:

```
      jmk101292   initial creation
      102498      addition of standard header   jrmk


*******************************************************************
      test for waiting until the end of an interblock delay
      Delay is not over, decrement count and return

      OK, we're doing something, what is it?
      test for alert output required
      alert fill pointer == alert empty pointer?
      Yes, alert queue empty, check for priority data

      Else Alert queue non-empty, buffer up next alert.

         into ping or pong
         1=ping/0=pong, state = buffer we are outputting from
         set to pong, so fill ping
         set to ping, so fill pong
```

```
   if can't find start of an alert
      then alert queue empty, check for priority data
   else Have start of alert, queue up alert

      calculate number of items that will be in buffer
      store count in correct output count register into ping or pong
      if length > 0
         do loop the rest into the buffer
      save updated alert empty pointer

      Nominally set message delay for next item to be
      output to 10 milli-seconds

      reset modulo, enable SSI interrupts and return

test for priority data
   if priority queue empty
      then clear TX prority flag and look for data to transmitt
   else Priority queue non-empty

      Fill ping or pong?
      First buffer of priority data?
      initialize output pointer
      Fill the first buffer with, type, length, synch, then data
      calculate length of this buffer

      if 30 or less words to output
         calculate number of items that will be in buffer

      if More than 30 words for output .AND.
      this is any buffer but the First one

      Time to output full buffers
      if 32 or less words to output
         then calculate number of items that will be in buffer
      else More than 32 words for output

      store word count in correct output count register
      do loop the rest into the buffer

      set priority output flag

      if last buffer, set message delay for next item to be
      output to 10 milli-seconds, clear priority output flag,
      and clear priority queue full flag

test for data
      Are we currently in the middle of outputting a block?

      Current block is done, have we output the entire data item?

      if Data item finished
         then cleanup flags, zero number of blocks requested
         send end of data alert, pop completed data item off queue
         output end of data alert

      Block finished, data item not finished
      if we still have pending block requests
         then 0 pending block requests,
      If data available alert has not been sent yet send it
```

```
        else return.

        Block finished, data item not finished,
        if we have pending block requests,
           then decrement number of blocks requested and send some data.
        fill up a buffer with data

        if First buffer of data?
           then Fill the first buffer with, type, length, synch
           if the first buffer of a data product
              then fill with header , then data
              Maximum header size allowable, 56 int16s
           else just data
           calculate number of entries in this buffer
        else this is any buffer but the First one
           then Fill the first buffer with, type, length, synch
           Time to output full buffers, =< 64 words to output
           if greater than 64 words for output
              then do loop the rest into the buffer

  Test for end of block
     if True:
        then clear data to transmit flag
             request 10 milli-second delay after this buffer
     else set data to transmit flag

  restore registers

  End of SSI Transmitt ping/pong buffer fill routine
```

NAME:

# essitx

PURPOSE:
ISR for SSI Transmitter underflow error.If entered, will cause system to reset.

CALLING SEQUENCE:
None.

INPUTS:
None.

OPTIONAL INPUT PARAMETERS:
None.

OUTPUTS:
None.

FUNCTIONS IN THIS FILE:
None.

INCLUDE FILES:
None.

SUBROUTINES CALLED:
None.

CALLED FROM:
None - ISR.

RESTRICTIONS:
None.

MODIFICATION HISTORY:
```
      jmk101292   initial creation
      102498      addition of standard header   jrmk

*****************************************************************
      reset system
```

```
NAME:
```

## ssitx -- SSI Transmitter buffer empty interrupt

```
PURPOSE:
            Service SSI Transmitter:
                  Fill hardware output buffer when empty
                  (generates this interrupt)
            Manage intermediate software ping/pong buffers:
                  Swap buffers when one empties
                  Request filling of empty buffer

            Disable SSI Transmit interrupt when out of data .OR. time for inter-
            block gap.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            Invoked when SSI Transmitter Hardware Buffer is empty.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk


*****************************************************************
      save registers

      if data available for output
         then decrement data available count
              output next word
      else
         if pong full
            then set pointer to pong
                 request ping buffer refill
         else if ping full
            then set pointer to ping
```

```
            request pong buffer refill

    restore registers
```

```
NAME:
```

# ssirx -- SSI Recevier interrupt

```
PURPOSE:
            Accumulate incoming SSI words until a complete
            message has been received

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            Invoked when a 16 bit word is received by the SSI
            Receiver Hardware.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk


******************************************************************
   save registers
      reset rx delay
      input and format received word
   restore registers
```

NAME:

# essirx -- SSI Recevier interrupt with framing Error

PURPOSE:
Flush received words of current incoming message when an error is detected. Sent alert to ground indicating Receiver error.

CALLING SEQUENCE:
None.

INPUTS:
None.

OPTIONAL INPUT PARAMETERS:
None.

OUTPUTS:
None.

FUNCTIONS IN THIS FILE:
None.

INCLUDE FILES:
None.

SUBROUTINES CALLED:
Fq_alert -- cwhite.c

CALLED FROM:
Invoked when the SSI Receiver Hardware detects a framing error on an incoming word.

RESTRICTIONS:
None.

MODIFICATION HISTORY:

```
      jmk101292   initial creation
      102498      addition of standard header   jrmk
```

```
******************************************************************
      save registers
      flush current message
      restore registers
```

NAME:

# TimeSync -- Synchronize Bus Cycle Count with S/C Time

PURPOSE:

Verify Synchronization of DPU time with S/C 1 Second time. If a IC_SYNCH_CLK command has been received, then the next time this routine is invoked the milli-second resolution time stored in Fspace_craft_time will be loaded into Fbus_cycle_count Any difference between expected/commanded time and system time (stored in Fbus_cycle_count) will be reported to the ground via an alert.

CALLING SEQUENCE:

None.

INPUTS:

None.

OPTIONAL INPUT PARAMETERS:

Fspace_craft_time

OUTPUTS:

Fbus_cycle_count can be modified by this routine.

FUNCTIONS IN THIS FILE:

None.

INCLUDE FILES:

None.

SUBROUTINES CALLED:

Fq_alert - cwhite.c

CALLED FROM:

Invoked when the Interrupt B line is exerted

RESTRICTIONS:

None.

MODIFICATION HISTORY:

```
      jmk101292   initial creation
      102498      addition of standard header   jrmk
```

```
*****************************************************************
      save registers
      if re-synching?
         then if DPU time == S/C time
                  then return
              else set bus cycle count to new value
                   output s/c and dpu times in an alert
      else checking milliseconds off
         if millisecond count == 0
            then return
         else
            output difference in an alert
            zero millisecond count
      restore registers
```

NAME:

# ESCIRec -- SCI Recevier interrupt with framing error

PURPOSE:

To handle framing errors in message words from other processors internal to the DPU. *Note: No longer in use.*

CALLING SEQUENCE:

None.

INPUTS:

None.

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

Alert indicating framing error on message from another processor.

FUNCTIONS IN THIS FILE:

None.

INCLUDE FILES:

None.

SUBROUTINES CALLED:

Fq_alert – cwhite.c

CALLED FROM:

Invoked by SCI Receiver hardware when a Framing Error is detected.

RESTRICTIONS:

No Longer In Use. SCI communication is now unidirectional. From the White processor to the others. No reply messages are sent via the SCI interface.

MODIFICATION HISTORY:

```
      jmk101292   initial creation
      102498      addition of standard header   jrmk
```

```
******************************************************************
      save registers
      flush current message
      send receiver back to sleep
      restore registers
```

```
NAME:
```

# SCIRec -- SCI receiver interrupt

```
PURPOSE:
            To input and format bytes onto complete SCI messages. Note: No longer in
            use.

CALLING SEQUENCE:
            None.

INPUTS:
      ASSUME:
            order of xmission: Address, Length, CID, Low, Mid, High, Low ......

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            Invoked by SCI Receiver hardware when a byte is received.

RESTRICTIONS:
            No longer in use. SCI communication is now unidirectional, from the
            white processor to the others. No reply messages are sent via the SCI
            interface.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk


******************************************************************
      save registers

      Put SCI back to Sleep
      restore registers
```

```
NAME:
```

# SCIXmit -- SCI Transmitter interrupt

```
PURPOSE:
            Manage the transmission of messages to the Red and Blue processors via
            the SCI interface. After all messages have been sent, turn off SCI
            Transmit hardware.

CALLING SEQUENCE:
            None.

INPUTS:
            sci_xmit_empty_pointer

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            Invoked buy the SCI Transmit Buffer Empty Hardware interrupt.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk


*****************************************************************
       ASSUMES: 0) That a 24 bit word ($073528C) is a marker for the start of
                   a SCI message
                1) The marker is followed by the SCI address  (in ls byte of
                   next word) of the reciepant of the message.
                2) The SCI address is followed by the message length (in ls
                   byte of next word).
                3) The Command ID (in ls byte of next word).
                4) The message follows in packed words.
                               Address,        8 bits
                               Length,         8 bits
                               CID,            8 bits
                               Packed word,    24 bits
                                       .
                                       .
                               Packed word,    24 bits
                   Low byte, Mid byte, High byte ........

      save registers
```

```
If start of message:
   flush start of message marker
   clear flag for length byte xmit
   clear flag for CID byte xmit
   clear flag for low byte xmit
   clear flag for mid byte xmit
   clear flag for hi byte xmit
   see if there is data in the xmitt software buffer
if software buffer empty, turn off transmitter interrupt
else,
   are we waiting to send MS byte?
      xmit MS byte
      clear flag for hi byte xmit
      set flag for low byte xmit, and
      return
   are we waiting to send Middle byte?
      xmit Middle byte
      clear flag for mid byte xmit
      set flag for hi byte xmit
      return
   are we waiting to send LS byte?
      xmit LS byte
      clear flag for low byte xmit
      set flag for Middle byte xmit
      return
   are we waiting to send CID word?
      xmit CID
      clear flag for cid word xmit
      set flag for Low word xmit
      return
   are we waiting to send length byte?
      xmit length
      clear flag for length byte xmit
      set flag for CID byte xmit
      return
   else send address, and
   set flag for length byte xmit, and
   return

if Queue not empty.
      then Set flag requesting delay before xmit next SCI message
      init delay
      turn off transmitter interrupt

restore registers
```

```
NAME:
```

# Fsci_command --

```
PURPOSE:
            Format and buffer messages to be sent to Red or Blue processors.
            Turn on SCI transmit hardware.
            Enable SCI Transmit Interrupt.

CALLING SEQUENCE:
            sci_command(int target_processor, int message)

INPUTS:
            target_processor -- BLUE1_SCI_ADDRESS, BLUE2_SCI_ADDRESS,
                                RED_SCI_ADDRESS
            message -- See Blue and Red code for a list of valid messages.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            cwhite.c, do_00, IC_SEL_KLINGON
            cwhite.c, do_00, IC_ENBL_EVENTS
            cwhite.c, do_00, IC_SEND_SCI_CMD
            cwhite.c, do_10, IC_DUMP_LOCAL_RAM
            whitedsp.c
            su_engineering.c
            su_init_exp.c
            su_initialize.c
            su_rtrd.c


RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk


******************************************************************

      provide mutial exclusion to prevent command interleaving between
      science code and ISR code

      save registers

      Set flag requesting delay before xmit next SCI message
```

```
init delay

pushed into the SCI output buffer:
      start of sci command (73528C)
      traget processor
      length (1)
      message

restore registers

provide mutial exclusion to prevent command
interleaving between science code and ISR code

turn on SCI transmitter
```

NAME:

# Fread_n_zero_sw_int_array

PURPOSE:

Interface between C code and global 16 bit RAM to allow user to read then zero out sections of Global memory. Assumes user wants to access 2D array. *Note: Not currently used.*

CALLING SEQUENCE:

```
Fread_n_zero_sw_int_array
(int base, int ysize, int yskip, int xsize, int *destination)
```

INPUTS:

```
int *destination (Local RAM address)
int xsize (The number of values in the X dimension
           of the 2D array to read and zero)
int yskip (The number of storage
           locations to skip to get to the first
           Y value to read)
int ysize (The number of values in the Y dimension
           of the 2D array to read and zero)
int base (global RAM address of x=0, y=0)
```

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

Number of words zeroed.

FUNCTIONS IN THIS FILE:

None.

INCLUDE FILES:

None.

SUBROUTINES CALLED:

None.

CALLED FROM:

Nowhere.

RESTRICTIONS:

None.

MODIFICATION HISTORY:

```
jmk101292   initial creation
102498      addition of standard header   jrmk
```

```
******************************************************************
      save registers

      save base
      save ysize
      save yskip
      save xsize
      save destination
      set zero-out flag
```

```
wait for BUSIO ISR to perform read and zero

clear zero-out flag

return transfer count
restore registers
```

```
NAME:
```

# Fwrite_command_int_array

```
PURPOSE:


CALLING SEQUENCE:
            Fwrite_command_int_array
            (int base, int ysize, int yskip, int xsize, int *destination)

INPUTS:
            int *destination (Global RAM address)
            int xsize (The number of values in the X dimension
                       of the 2D array)
            int yskip (not used)
            int ysize (The number of values in the Y dimension
                       of the 2D array)
            int base (Local RAM address of data to be written)

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            Number of words transfered.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            cwhite.c, do_00, IC_LOAD_REF_STARS
            cwhite.c, do_00, IC_LOAD_SCI_WDW
            cwhite.c, do_10, IC_DUMP_LOCAL_RAM

RESTRICTIONS:
            Assumes it is called when processor has Global Bus

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk


*****************************************************************
      save registers
      load ysize
      load yskip
      load xsize
      load source
      load destination
      write data from base to destination
      return number of words written to Global RAM
      restore registers
```

```
NAME:
            read_global

PURPOSE:
            Chooses between subroutines that input data from
            Global RAM

CALLING SEQUENCE:
            None.

INPUTS:
            Packed bit flags stored in Fglobal_bus_flags.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            read_int_array
            read_int

CALLED FROM:
            BUSIO ISR -- This file

RESTRICTIONS:
            Requires Global Bus access

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk


******************************************************************
      if array input
            then call read_int_array
      else call read_int
```

```
NAME:
```

# write_global

```
PURPOSE:
            Chooses between subroutines that output data to the
            Global RAM

CALLING SEQUENCE:
            None.

INPUTS:
            Packed bit flags stored in Fglobal_bus_flags.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            write_int_array
            write_int

CALLED FROM:
            BUSIO ISR -- This file

RESTRICTIONS:
            Requires Global Bus access

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk



******************************************************************
      if array output
            then call write_int_array
      else call write_int
```

```
NAME:
            zero_global

PURPOSE:
            Initialize a range of Global RAM to Zero

CALLING SEQUENCE:
            None.

INPUTS:
            Packed bit flags stored in Fglobal_bus_flags.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            BUSIO -- in this file

RESTRICTIONS:
            Requires Global Bus access

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk


******************************************************************
      save registers
      get starting address
      get length
      write out zeros
      restore register
      return number of locations zeroed
```

```
NAME:
            SCIHbeatOutput

PURPOSE:
            Verify correct operation of enabled DSPs by
            sequentially requesting a standard reply from
            each enabled DSP.

CALLING SEQUENCE:
            SCIHbeatOutput

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            An alert is output if an enabled processor does not
            respond to the command within one second

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            Fq_alert -- cwhite.c

CALLED FROM:
            BUSIO -- this file

RESTRICTIONS:
            ASSUMES called while white has global bus

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk



*****************************************************************
      save registers
      reads the current heartbeat response in global ram
      clears response in global ram
      checks for correct response
      if fail
         test for throw away of first error
         send ssi to tell who died

      point to next enabled dpu
      send sci heartbeat request
      restore registers
```

NAME:

# Stkerr -- Stack error interrupt

PURPOSE:
ISR stub to provide indication when a stack error occurs

CALLING SEQUENCE:
None.

INPUTS:
None.

OPTIONAL INPUT PARAMETERS:
None.

OUTPUTS:
None.

FUNCTIONS IN THIS FILE:
None.

INCLUDE FILES:
None.

SUBROUTINES CALLED:
None.

CALLED FROM:
Invoked by circuitry internal to the DSP56001.

RESTRICTIONS:
None.

MODIFICATION HISTORY:

```
jmk101292   initial creation
102498      addition of standard header   jrmk
```

```
******************************************************************
      indicate error with a task_id of F0XX
      reset the system
```

NAME:

# Trace

PURPOSE:

ISR stub. *Note: Not currently used.*

CALLING SEQUENCE:

None.

INPUTS:

None.

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

None.

FUNCTIONS IN THIS FILE:

None.

INCLUDE FILES:

None.

SUBROUTINES CALLED:

None.

CALLED FROM:

None.

RESTRICTIONS:

None.

MODIFICATION HISTORY:

```
      jmk101292   initial creation
      102498      addition of standard header   jrmk
```

```
******************************************************************
      indicate error with a task_id of E0XX
      reset the system
```

NAME:

# Illi -- Illegal interrupt

PURPOSE:
ISR stub to provide indication that a non-valid op-code was executed. This can occur when programming errors cause the attempted execution of data areas.

CALLING SEQUENCE:
None.

INPUTS:
None.

OPTIONAL INPUT PARAMETERS:
None.

OUTPUTS:
None.

FUNCTIONS IN THIS FILE:
None.

INCLUDE FILES:
None.

SUBROUTINES CALLED:
None.

CALLED FROM:
Invoked by circuitry internal to the DSP56001.

RESTRICTIONS:
None.

MODIFICATION HISTORY:

```
jmk101292   initial creation
102498      addition of standard header   jrmk
```

```
******************************************************************
      indicate error with a task_id of D0XX
      reset the system
```

NAME:

# Swi -- Software Interrupt

PURPOSE:

ISR Stub for possible later implementation of the Software Interrupt Instruction. *Note: Currently not used*

CALLING SEQUENCE:

None.

INPUTS:

None.

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

None.

FUNCTIONS IN THIS FILE:

None.

INCLUDE FILES:

None.

SUBROUTINES CALLED:

None.

CALLED FROM:

None.

RESTRICTIONS:

None.

MODIFICATION HISTORY:

```
      jmk101292   initial creation
      102498      addition of standard header   jrmk
```

```
******************************************************************
      indicate error with a task_id of C0XX
      reset the system
```

NAME:

# F__start -- System Initialization

PURPOSE:
To initialize white processor to a known state after power up or a reset.

CALLING SEQUENCE:
None.

INPUTS:
None.

OPTIONAL INPUT PARAMETERS:
None.

OUTPUTS:
None.

FUNCTIONS IN THIS FILE:
None.

INCLUDE FILES:
None.

SUBROUTINES CALLED:
None.

CALLED FROM:
Invoked when a reset interrupt is triggered.

RESTRICTIONS:
None.

MODIFICATION HISTORY:

```
jmk101292   initial creation
102498      addition of standard header   jrmk
```

```
******************************************************************

      reset ssi and sci interfaces
      inspect the stack
      reset the 'on chip' stack by setting the 56001 SP to 0

      clear red and blue communications flags in Global RAM
      clear track_guide_stars debugging enabled flag
      clear bus cycle counter
      clear bus cycle counter for SSI heartbeat
      clear bus cycle counter for SCI heartbeat
      initialize SCI heartbeat response array
      initialize SCI heartbeat address array
      initialize SCI heartbeat global response area
      initialize SCI hardware
      clear all global bus control flags
      set white as the default Klingon
      initialize code upload and table upload locks
      Initialize SCI port
      initialize SSI software
```

```
initialize Alert queue fill and empty pointers
initialize number of SSI blocks requested
initialize SSI transmit output count
clear ssi rx command delay
clear ssi tx interblock delay
clear ssi tx ping buffer fill count
clear ssi tx pong buffer fill count
initialize SSI hardware
initialize Forward Error Correction table
perform crt0 initializations to make C run
initialize floating point conversion table
To change the base of the stack
Turn on interrupts and let 'er scream
Return Global Bus
```

# Red.asm Detailed Design Document

```
*****************************************************************
FILE:
            Red.asm

PURPOSE:
            System initialization
            Interrupt Service Routine Handlers
            Optimized C code
            Interface between C code and Global Resources

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            red.h
            asm_dpucfg_map.h
            asm_sci_com.h
            asm_idiot.h
            asm_red_control.h
            FecTab.asm
            FecCode.asm

SUBROUTINES CALLED:
            None.

CALLED FROM:
            ROM based code pointed to by hardware after Reset.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      101292      initial creation jmk
      02795       addition of non-sci heartbeat returns from the
                  red and blue processors jmk
      120397      jrmk  removal of unsued stuff & addition of FEC
      050598      jrmk  1) moved red SAA asm calls into "on-chip" memory
                  2) disabled "debug_code" mode (where all code is pushed
                  off on chip RAM)
            NOTE: A backup copy of red.asm is called red.asm.050598
                  it is the version
      "-rw-r--r--  1 xmmom       86908 Dec  3 18:36 red.asm.120397"
```

```
      081198        jrmk  fixed race condition in code that loads a SU
                    (two places clearing the bit flag LOAD_A_SU, one in
                    Fload and one in Floader).
      081298        jrmk  added standard wait states reference
                    removed "Prom" wait state changes in FLoader, no longer
                    needed because programs are stored in RAM, not ROM
      091798        jrmk  added code to read in loader.lut on powerup.
      012299        addition of standard header   jrmk

RCS $Id: red.asm,v 1.18 1999/02/19 20:29:16 jrklark Exp $



******************************************************************
******************************************************************
       Switch for pushing code off chip so we see it with the logic analyzer
               0 = allow code on chip
               1 = push code onto off chip memory
      DEBUG_CODE    EQU   $00


******************************************************************
       Switch to enable the use of spare#1 to time read_n_z_sw_int_array
               0 = Disable
               1 = Enable
      TIME_RNZSWIA     EQU     $00



******************************************************************
       Switch to enable the use of spare#1 to debug/time busio
               0 = Disable
               1 = Enable
      TIME_BUSIO   EQU   $00


******************************************************************
       Switch to enable the use of spare#1 to debug/time IN_DATA
               0 = Disable
               1 = Enable
      TIME_IN_DATA       EQU   $00


******************************************************************
       Switch to indicate to FEC Code which processor we are running on


      FEC_DSP            EQU   RED
```

```
NAME:
            Boot strap Loader Code

PURPOSE:
            Copy executables for the Red processor from KAL
            powered Global Program Storage(GPS) RAM into the On
            Board RAM. These executables stored in GPS RAM
            are protected by Forward Error Correction.

CALLING SEQUENCE:
            none.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            Jumps to address 0000h after code is loaded

INCLUDE FILES:
            FecTab.asm
            FecCode.asm

SUBROUTINES CALLED:
            FfwrdErrCrection -- this file.

CALLED FROM:
            Firmware internal to the DSP56001 after the reset
            signal removed from the processor.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012299      addition of standard header   jrmk


******************************************************************
      Forward Error Correction table included here
      Initialize pointer to Red Global Status
      Initialize pointer to Red Global Errors
      Initialize pointer to Red FEC address error array location
      To change the base of the stack, change the value loaded into the
      stack pointer here. F__y_size is defined in crt0 as the last location
      of the on DSP card global variables. After that starts the stack
      area.
      enable access of program storage card
      Initialize increment variable
      Use Global Lookup Table to get location of flight Asm BIOS
      Read starting address of asm code in GPS RAM
      Read length of code to load
      because of FEC we need to input the code in 16 word chunks
      calculate the number of code chunks that need to be input
```

```
input the first section of the executables and FEC them:
      interrupts vectors
      red.asm code
Use Global Lookup Table to get location of flight C BIOS
Read starting address of C code in GPS RAM
Read length of code to load
because of FEC we need to input the code in 16 word chunks
calculate the number of code chunks that need to be input
input the second section of the executables and FEC them:
      Cred.C code
Use Global Lookup Table to get location of flight REDDSP BIOS
Read starting address of REDDSP code in GPS RAM
Read length of code to load
because of FEC we need to input the code in 16 word chunks
calculate the number of code chunks that need to be input
input the second section of the executables and FEC them:
      REDDSP.C code
disable access of program storage card
Start this bad boy!!
Forward Error Correction code included here.
```

```
NAME:
```

# Definition of interrupt vectors

```
        used by Flight code
           ORG           P:$0000


MODIFICATION HISTORY:
      jmk101292   initial creation



******************************************************************
      Initialize interrupt vectors

      reset/cold boot   @ P:$0000

      Stack error interrupt   @ P:$0002

      Trace interrupt   @ P:$0004

      Software interrupt      @ P:$0006

      int A, Global Bus I/O   @ P:$0008

      interrupt B       @ P:$000A

      ssi receive data  @ P:$000C

      ssi rec dat w/error    @ P:$000E

      ssi transmit data @ P:$0010

      ssi xmit data w/error   @ P:$0012

      sci receive data  @ P:$0014

      sci rec data w/error    @ P:$0016

       fill remaining unused vectors with dummy call


******************************************************************
            !!!!!!!!!!!!!!!START APPLICATION CODE HERE!!!!!!!!!!!!!!!
******************************************************************

      Standard ASM code included here
      Moved up by CH 052698


******************************************************************
      Standard Shift and Add code included here

******************************************************************
      Debugging option of pushing all executables off Chip Memory
      included here
```

```
******************************************************************
      Forward Error Correction Code included here
```

```
NAME:
```

# BUSIO -- Global bus interrupt handler

```
PURPOSE:
            Service periodic (~ 1 millisecond) interrupt that
            indicated opportunity to access Global Resources

CALLING SEQUENCE:
            None -- ISR

INPUTS:
            Control Bits in Packed word:
                  Global_bus_flags
                  Global_bus_flags2

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            Mem_test -- this file
            ScrubLoader -- this file
            FecFix -- this file
            Floader -- this file
            zero_global -- this file
            write_global -- this file
            read_global -- this file

CALLED FROM:
            None - ISR.


RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012299      addition of standard header   jrmk


******************************************************************
      Check if we are the klingon
      if we're not the klingon lets get on with business
      if we're the klingon, is it time to give the bus back?
      Time to give the bus back!
      Return Global Bus
      wiggle Return Global Bus Bit
      wait a while for the CD4000 parts
      Return
      Time to do some work
      set indicator that we have the bus
```

```
save registers
         RAM testing?

increment bus cycle counter
is there a scrub request?
 is there a Fix request?
if it's time to load a SwapUnit lets do it

to make sure we finish FECing previously loaded
OpCodes, disable su_loading until current load
has been FECed this is done by using COPY_OF_LOAD_A_SU
to disable calls to Floader in the BUSIO ISR.

Check for requests to zero buffer area in the global RAM
if request pending, honor it
Check the write global memory queue to see if we need to output some
stuff for those scum end users
if request pending, honor it
Check the read global memory queue to see if we need to input some
stuff for the ungrateful end users.
if request pending, honor it
pop an global read request off the queue
Update red global semaphore and error flag
then clear MS 16 bits of Fred_semaphore and clear Fred_error
If there is a pending Heartbeat request honor it

wiggle Return Global Bus Bit
restore registers
```

```
NAME:
            Fload -- Load a swap unit

PURPOSE:
            Loads, forward error corrects, and executes a module of C code

CALLING SEQUENCE:
            Fload(int numberOfSwapUnitToLoad)

INPUTS:
            numberOfSwapUnitToLoad -- indicates which swap unit to load.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            FfwrdErrCrection -- this file
            Floader -- this file

CALLED FROM:
            reddsp.c

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk


*******************************************************************
      save registers
      pop argument off stack
      set bits in Fglobal_bus_flags to indicate pending request
      for loading of a swap unit
      if FEC is enabled (and it always is)
      set up the required stuff to perform the load from
      global memory. Then wait for a BUSIO interrupt service routine to
      load up a block of Program Storage RAM. Once it has it will set
      the FEC_PROG_BLOCK bit in Fglobal_bus_flags2 word. This indicates
      the opcodes and Forward error correction bits have been loaded
      into memory and require checking. When finished move the code
      so the FEC words are written over by opcodes.
      to make sure we finish, disable su_loading until
      this load has been FECed
      calculate how many times we need to call FEC
      move the opcodes into the FEC incomming buffer
      indicate that we have FECed the current chunck of opcodes
      because we have reduced the size of the code loaded by
      15/16, we must change the "load_to_address" used by
```

```
Floader
now that we have finished see if we need to enable
SU loading
now see what we do next
if FEC is not enabled (this option is no longer used)
set bits in Fglobal_bus_flags and block until BUSIO
ISR finishes loading the executable
executables have been loaded, jump to
jsr to the starting address in the Swap Unit loading area.

after returning from the jsr to the Swap Unit save
an indication of what Swap Unit was called for Cheng's code
restore registers
and return to calling program
```

```
NAME:
```

# Floader --

```
PURPOSE:
            called from BUSIO ISR, this routine access the KAL
            powered Global Program Storage(GPS) RAM card and moves
            executables from there into local memory.

CALLING SEQUENCE:
            jsr Floader

INPUTS:
            rom_table_vector -- pointer to:
                  address of where to load from (in GPS RAM)
                  number of words to load
            rom_table_vector is actually a Look up Table stored in GPS RAM

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            BUSIO -- this file

RESTRICTIONS:
            Assumes access to the Global Bus.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk



******************************************************************
      save registers
      enable access of program storage card
      set bits in Fglobal_bus_flags indicating we are loading
      a Swap Unit (might take multiple bus cycles)
      on first pass through Floader read GPS RAM base lookup table
      for starting address of executables and length of executable
      and save locally

      Read starting address of Swap Unit code in GPS RAM
      Read length of code to load
      move executables from GPS RAM to On Board RAM

      first pass or nth pass read from local RAM:
```

```
      number of opcodes to load
      where to read them from
      where to put them
disable access of program storage card
save
      where to read them from
      where to put them
and request the block just loaded be FECed
restore registers
```

```
NAME:
            FFecABlock

PURPOSE:
            C callable routine that reads in a 16 word chunk of executables
            and performs checks for Single and Multibit errors.

CALLING SEQUENCE:
            FFecABlock(dpuint StartingAddress)

INPUTS:
            StartingAddress
            (address of executables stored in the GPS RAM to scrub)

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            0 == no errors found
            1 == single bit error, fixable
            2 == multibit error, not fixable

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            FfwrdErrCrection

CALLED FROM:
            Reddsp.c

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012299      addition of standard header   jrmk


*****************************************************************
      save registers

      get parameter off stack
      block until the chunk to be scrubbed is input from GPS RAM
      Chunk input, call FEC routine
      NOTE:
            error is returned from FfwrdErrCrection in the A Reg
            to the c-calling routine
      restore registers
      return to calling program
```

```
NAME:
            ScrubLoader
PURPOSE:
            input a Chunk from GPS RAM and store in global FEC buffer
            (located in on board RAM) for FEC Scrubbing

CALLING SEQUENCE:
            called from BUSIO ISR when flag bits in
            Fglobal_bus_flags2 are set.

INPUTS:
            load_from_address, initialized by FFecABlock

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            BUSIO ISR

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012299      addition of standard header   jrmk


******************************************************************
      save registers

      enable access of program storage card
      input the chunk
      disable access of program storage card
      restore registers
      END OF FEC A Block CODE
```

```
NAME:
            FFecFixAnOpcode

PURPOSE:
            Fix a word in the GPS RAM that has been shown to have
            a single bit error (by FFecABlock).

CALLING SEQUENCE:
            FFecFixAnOpcode

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            Reddsp.c

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012299      addition of standard header   jrmk


******************************************************************
      save registers
      set flag bit in Fglobal_bus_flags2 requesting
      repair of corrupted GPS RAM location.
      Then block until fix is completed.
      fix completed
      restore registers
```

```
NAME:
            FecFix

PURPOSE:
            update FEC error count and FEC error location
            fix corrupted location in GSP RAM

CALLING SEQUENCE:
            None, called from BUSIO ISR

INPUTS:
            FGlobalAddressOfError
            FFixGlobalErrors
      both generated by FFecABlock and stored in global RAM

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            BUSIO ISR

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012299      addition of standard header   jrmk


******************************************************************
      save registers
      increment FEC error count
      update FEC list of address where errors have been found
      enable access of program storage card
      fix error
      disable access of program storage card
      restore registers
```

```
NAME:
            essitx

PURPOSE:
            ISR for SSI Transmitter underflow error.
            If entered, will cause system to reset.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            None.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk

******************************************************************
```

```
NAME:
            ssitx

PURPOSE:
            ISR for SSI Transmitter output buffer empty.
            If entered, will cause system to reset.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            None.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk

*****************************************************************
```

```
NAME:
```

## ssirx ------ This interface is NOT USED

```
PURPOSE:
            ISR for SSI Receiver input buffer full.
            If entered, will cause system to reset.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            None.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk

******************************************************************
```

```
NAME:
            essirx

PURPOSE:
            ISR for SSI receiver overflow error.
            If entered, will cause system to reset.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            None.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012299      addition of standard header   jrmk

******************************************************************
```

NAME:

# ESCIRec -- SCI Recevier interrupt with framing error

PURPOSE:

To handle framing errors in message words from other processors internal to the DPU. When error is detected any command fragment currently held is flushed.

CALLING SEQUENCE:

None.

INPUTS:

None.

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

None.

FUNCTIONS IN THIS FILE:

None.

INCLUDE FILES:

None.

SUBROUTINES CALLED:

None.

CALLED FROM:

Invoked by SCI Receiver hardware when a Framing Error is detected.

RESTRICTIONS:

None.

MODIFICATION HISTORY:

```
      jmk101292   initial creation
      012399      addition of standard header   jrmk
```

```
*******************************************************************
      save registers
      flush input command buffer

      clear received byte count

      send receiver back to sleep
      clear start of collection flag
      reset fill and empty pointers
      restore registers
```

```
NAME:
```

# SCIRec -- SCI receiver interrupt

```
PURPOSE:
            To input and format bytes onto complete SCI messages.

CALLING SEQUENCE:
            None.

INPUTS:
      ASSUME:
            order of xmission: Address, Length, CID, Low, Mid, High, Low ......

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            Invoked by SCI Receiver hardware when a byte is received.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk


******************************************************************
      save registers
      are we currently collecting a message?
      if yes jump past initialization and go see
      what part of the message we are waiting for
      not collecting, check address to see if it's for us
      Is the message for this processor?
      Is the message for everyone?
      Not a message for us, put SCI back to Sleep
      is a message for us:
      Start receiving a new message by:
            clearing message length count
            set start of collection flag
      clear flags indicating:
            CID was input
            hi nibble received
            middle nibble received
            low nibble received
      then return until next nibble is received
      What part of the message are we waiting for?
```

```
input command LENGTH
set up for next input, then return
input Command ID
set up for next input, then return
input Low Byte of next data word
set up for next input, then return
input Middle Byte of next data word
set up for next input
test number of nibbles received against command length
if equal jump to command interpreter, else return
input High Byte of next data word
set up for next input
test number of nibbles received against command length
if equal jump to command interpreter, else return
If full command received:
      Put SCI back to Sleep
clear start of collection flag
execute command
restore registers
```

```
NAME:
            Fread_n_zero_sw_int_array

PURPOSE:
            Interface between C code and global 16 bit RAM
            to allow user to read then zero out sections of Global
            memory.
            Assumes user wants to access 2D array.

CALLING SEQUENCE:
            Fread_n_zero_sw_int_array
            (int base, int ysize, int yskip, int xsize, int *destination)

INPUTS:
            int *destination (Local RAM address)
            int xsize (The number of values in the X dimension
                       of the 2D array to read and zero)
            int yskip (The number of storage
                       locations to skip to get to the first
                       Y value to read)
            int ysize (The number of values in the Y dimension
                       of the 2D array to read and zero)
            int base (global RAM address of x=0, y=0)

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            Number of words zeroed.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            callable from C high level code.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk


******************************************************************
      save registers
      save base
      save ysize
      save yskip
      save xsize
      save destination
      set zero-out flag
      set flag bits in Fglobal_bus_flags requesting
```

```
BUSIO ISR to service this request
block until the BUSIO ISR completes request
BUSIO has completed task:
      clear zero-out flag
transfer count is returned in base from bus xfer routine
restore registers
```

```
NAME:
            read_global

PURPOSE:
            Chooses between subroutines that input data from
            Global RAM

CALLING SEQUENCE:
            None.

INPUTS:
            Packed bit flags stored in Fglobal_bus_flags.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            read_int_array
            read_int

CALLED FROM:
            BUSIO ISR -- This file

RESTRICTIONS:
            Requires Global Bus access

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk

*******************************************************************
```

```
NAME:
```

write_global

```
PURPOSE:
            Chooses between subroutines that output data to the
            Global RAM

CALLING SEQUENCE:
            None.

INPUTS:
            Packed bit flags stored in Fglobal_bus_flags.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            write_int_array
            write_int

CALLED FROM:
            BUSIO ISR -- This file

RESTRICTIONS:
            Requires Global Bus access

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk

******************************************************************
```

NAME:

# zero_global

PURPOSE:

Initialize a range of Global RAM to Zero

CALLING SEQUENCE:

None.

INPUTS:

Packed bit flags stored in Fglobal_bus_flags.

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

None.

FUNCTIONS IN THIS FILE:

None.

INCLUDE FILES:

None.

SUBROUTINES CALLED:

None.

CALLED FROM:

BUSIO -- in this file

RESTRICTIONS:

Requires Global Bus access

MODIFICATION HISTORY:

```
      jmk101292   initial creation
      012399      addition of standard header   jrmk
```

```
*****************************************************************
      save registers
      get starting address and length
      zero them out

      number of zeroed location is returned in global_add
      restore registers
```

```
NAME:
```

# Stkerr -- Stack error interrupt

```
PURPOSE:
            ISR stub to provide indication when a stack error occurs
            Used for debugging.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            Invoked by circuitry internal to the DSP56001.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk

******************************************************************
```

NAME:

# Trace

PURPOSE:

ISR stub. Used for debugging. *Note: Not currently used.*

CALLING SEQUENCE:

None.

INPUTS:

None.

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

None.

FUNCTIONS IN THIS FILE:

None.

INCLUDE FILES:

None.

SUBROUTINES CALLED:

None.

CALLED FROM:

None.

RESTRICTIONS:

None.

MODIFICATION HISTORY:

```
jmk101292   initial creation
102498      addition of standard header   jrmk
```

*******************************************************************

```
NAME:
```

# Illi -- Illegal interrupt

```
PURPOSE:
            ISR stub to provide indication that a non-valid op-code
            was executed. This can occur when programming errors
            cause the attempted execution of data areas.
            Used for debugging.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            Invoked by circuitry internal to the DSP56001.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk

******************************************************************
```

NAME:

CURRENTLY NOT USED

# Swi -- Software Interrupt

PURPOSE:

ISR Stub for possible later implementation of the Software Interrupt Instruction.
Used for debugging.

CALLING SEQUENCE:

None.

INPUTS:

None.

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

None.

FUNCTIONS IN THIS FILE:

None.

INCLUDE FILES:

None.

SUBROUTINES CALLED:

None.

CALLED FROM:

None.

RESTRICTIONS:

None.

MODIFICATION HISTORY:

```
jmk101292   initial creation
012399      addition of standard header   jrmk
```

******************************************************************

```
NAME:
```

# F__start -- System Initialization

```
PURPOSE:
            To initialize red processor to a known state
            after power up or a reset.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.
FUNCTIONS IN THIS FILE:

            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            Invoked when a reset interrupt is triggered.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk


*****************************************************************

      reset peripheral interface
      initialize on chip wait state generator
      ext X, ext Y, ext P, ext I/O
      Make port B a parallel I/O port
      output:H0,H1,H3,H4,H6...input:all the rest output
      set red task ID to idle
      clear bus cycle counter
      clear all global bus control flags
      clear red semaphore
      clear red error flag
      clear Fend_of_frame
      but, red is the default Klingon
      initialize red FEC address error array location
      start and end
      initialize SCI software stuff
      initialize SCI hardware
      Initialize SCI port
      initialize pixel buffers fill and empty pointers
```

```
from back when we had a Red Detector
initialize Forward Error Correction table
crt0 initializations to make C run
point to floating point conversion table
initialize floating point conversion table
To change the base of the stack, change the value loaded into the
stack pointer here. F__y_size is defined in crt0 as the last location
of the on DSP card global variables. After that starts the stack
area.
 end of crt0 initializations to make C run
read in loader.lut for background scrubbing task
091798      new   jrmk
Turn on interrupts and let 'er scream
ENABLE SCI@1 A@0 & SSI@2
Turn on interrupts
turn on sci and ssi port
Return Global Bus
wiggle Return Global Bus Bit
wait a while for the CD4000 parts
Now that the operating system is initialized
start executing C code.
```

```
NAME:
            sci_command_inter

PURPOSE:
            to interpret and execute SCI commands sent from the White DSP

CALLING SEQUENCE:
            None. Inputs are passed through globals.

INPUTS:
            Current command is pointed to by the sci_empty_pointer.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            SCIRec -- This file

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk


*****************************************************************
   save registers
      Execute the current command
      Called from the SCI interrupt, expecting A, B, X1, R7 and M7 to already
      be saved by the caller. And to be replaced by the caller.
      Command ID:
            $0A0 == Heart beat request
            $0A1 ==
            $0A2 ==
            $0A3 == Enable Klingon
            $0A4 == Disable Klingon
            $0A5 == Enable Events
            $0A6 == Disable Events
            $0A7 == Enable raw data output
            $0A8 == output local ram contents to global ram
            $0A9 == Enable RAM test
            $0AA == Disable RAM test
            $0AB == change task to abort
            $0AC == change task to cosmic ray filter
            $0AD == change task to accumulate
            $0AE == change task to fast mode
```

```
            $0AF ==

HUGE S!

*****************************************************************



*****************************************************************
            execute the valid command
      Command ID:
            $0A0 == Heart beat request



*****************************************************************
      Command ID:
            spare ==




*****************************************************************
      Command ID:
            spare ==




*****************************************************************
      Command ID:
            ENABLE_KLINGON == Allow this DSP to be the Klingon

      set bit flag in Fglobal_bus_flags indicating Red is
      the Klingon



*****************************************************************
      Command ID:
            DISABLE_KLINGON == Make this DSP not the Klingon

      set bit flag in Fglobal_bus_flags indicating Red is
      no longer the Klingon



*****************************************************************
      Command ID:
            ENABLE_EVENTS == Enable Event Interrupt
      NOT USED

      initialize pixel buffers fill and empty pointers
            reset Red data interrupt




*****************************************************************
      Command ID:
```

```
        DISABLE_EVENTS == Disable Event interrupt
    NOT USED



*****************************************************************
    Command ID:
        ENABLE_ENGINEERING == Enable raw data output
    NOT USED

    setup raw data addresses
    signal start of raw event dump



*****************************************************************
    Command ID:
        DUMP_LOCAL_RAM == Write contents of local RAM into 0x0FFFFF
    NOT USED



*****************************************************************
    Command ID:
        ENABLE_RAM_TEST== Start testing global RAM

    Make sure events are disabled.
    setup test parameters
    set bit flag in Fglobal_bus_flags to start RAM testing



*****************************************************************
    Command ID:
        DISABLE_RAM_TEST== Stop testing global RAM

    set bit flag in Fglobal_bus_flags to stop RAM testing



*****************************************************************
    Command ID:
        DO_RED_TASK_IDLE ==  change task to idle

    set Fred_task_id  to #>RED_IDLE_TASK



*****************************************************************
    Command ID:
        DO_RED_TASK_ABORT ==  change task to abort

    set Fred_task_id  to #>RED_ABORT_TASK



*****************************************************************
    Command ID:
```

```
            DO_RED_TASK_ACCUMULATE ==  change task to accumulate

      set Fred_task_id  to #> RED_ACCUMULATE_TASK


*****************************************************************
      Command ID:
            DO_RED_ABORT_INITIALIZE ==  change task to initialize

      set Fred_task_id  to #>RED_INITIALIZE_TASK


*****************************************************************
      Command ID:
            RED_RAW_DATA_TASK == change task to fast mode
      NOT USED

TESTAF      move  #>RED_RAW_DATA_TASK,A1
            cmp   X1,A
            jne   BADnsci_command_inter
            move  #$600000,X1
            move  X1,Y:Fred_raw_address
            move  #>RED_RAW_DATA_TASK,X1
            move  X1,Y:Fred_task_id
            jmp     nsci_command_inter


*****************************************************************
   Tell the world that we've had a problem
  CHCHCHCHCHCHCHCHCHCHCHCHCHCHCH
  Need to do a more descriptive and graceful trap here!


*****************************************************************
   restore registers
```

```
NAME:
            Mem_test:

PURPOSE:
            Increment the last 265 locations on each
            RAM board three times. During each global bus
            a different RAM board is incremented.

CALLING SEQUENCE:
            SCI command from White

INPUTS:
            MEM_TEST_COUNT initialized by SCI command ENABLE_RAM_TEST
            MEM_TEST_SWITCH initialized by SCI command ENABLE_RAM_TEST

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            The white processor is informed of the end of
            test when the processor status in the heartbeat
            reply indicates memory test is finished.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk


*******************************************************************
      test MEM_TEST_SWITCH to see which Global RAM Board we are testing

      if falls through then error condition, end of test
      else, if branched to, then end of test
       Test board 0x700000 and update pass count
       Test board 0x600000
       Test board 0x500000
       Test board 0x400000
       Test board 0x080000
       Test board 0x000000
       increment the assigned addressed
      update switch pointer
      end of section red
```

# Blue.asm Detailed Design Document

```
NAME:
            blue.asm
            Flight Blue DSP's code
      jmk 100892

PURPOSE:
            to convert 24 bit serial data from MIC detector into
            an address in Global RAM, and to increment that address.
            to provide high temporal resolution data on a maximum
            of 512 pixels.
CALLING SEQUENCE:
            none.
INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            "asm_sci_com.h"
            "asm_dpucfg_map.h"
            "FecTab.asm"
            "FecCode.asm"

SUBROUTINES CALLED:
            None.

CALLED FROM:


RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      100892      initial creation
      last update 091292
      080193      memtest commands
      011993      address calculation changes
      021593      more address calculation changes
      021993      fix around the nonfunctional motorola compiler stuff
      040193      fix for fractional build up during address calculation
                  leading to an error of 1 in the final address. Fixed by
                  clearing A2 before doing the final add.
      040193      addition of 9 bit anding after adding kx to xoff and
                  after adding ky to yoff.
      040193      add fix to prevent stack over-writes when a series of
                  pushes or pops is interrupted by a hardware interrupt.
      051993      Added code to always load 16 memory windows.
      121493      Added engineering mode code to dump raw events,
```

```
                included changes to boot strap
022394          changes to support ram program storage
020795          addition of non-sci heartbeat returns from the
                red and blue processors jmk
091897          as part of debugging effort changed wait states
                from 1117 to 1119 jrmk
112697          attempt to add FEC to boot strap
050598          replaced nops with nop.asm
081198          removed code to use xbit (double event flag) jrmk
                set wait states back to 1117
081298          removed unused code (code that supported old MIC
                output format) added output of milliSecond event
                rate to BW Proc area added standard wait state variable
012399          addition of standard header   jrmk
Operational Notes:
      1) Enable_RAM_test disables events and disable_RAM_test
      doesn't re-enable them.
      Register definitions:

         R0 == Stack
         M0 == mod 65536
         N0 == FM Status Bits
         R1 == Raw photon buffer fill pointer
         M1 == mod 256
         N1 == System status flags
         R2 == Raw photon buffer empty pointer
         M2 == mod 256
         N2 == Output data do loop counter
         R3 == Address output queue fill pointer
         M3 == mod 256
         N3 == #of events to output
         R4 == Address output queue empty pointer
         M4 == mod 256
         N4 == The constant 1
         R5 == Base pointer to ping/pong calculation array
         M5 == mod 65536
         N5 == The constant $01FF
         R6 == Current calculation pointer
         M6 == mod 65536
         N6 == Storage for the Window ID
         R7 == Generic undedicated address pointer
         M7 == change able
         N7 == The 9 bit shift factor

BIGTIME NEXT LARGE PART

     DebugFM:
          0 => output FM data
          1 => output the index of the data
     DebugFM          EQU   0


******************************************************************
      Debug Switch:
            0 => Flight code
            1 => push all code off chip
     DebugProgram  EQU 0
```

```
*****************************************************************
       Debug Switch:
               0 => Flight code
               1 => Spare1 defines the envelope of operation of the
                  Bus interrupt
      DebugGlobal     EQU  0



*****************************************************************
       Debug Switch:
               0 => Flight code
               1 => Spare1 defines the envelope of operation of the
                  FastMode calculation
      DebugFMCalc     EQU  1



*****************************************************************
       Klingon Switch:
               0 => Normal system mode
               1 => Enable This processor to be the klingon for debugging
      KLING       EQU  0



*****************************************************************
       Bad address switch:
               0 => Normal operations mode
               1 => write bad address out to X:$8888
      BADD        EQU  1



*****************************************************************
       Switch to indicate to FEC Code which processor we are running on


      FEC_DSP            EQU   BLUE



****************************************************************
      System status flags and misc. addresses


      Where you get the data
      I/O address of latch for global bus address buss
      I/O address of global bus data buffer
      CID of first implemented command
      CID of last implemented command
      maximum number of raw events to
      output during a global bus cycle
      maximum number of FM pixel address to
      input during a global bus cycle (bfast input)
      maximum number of FM pixels to
      sample during a global bus cycle
      maximum number of FM data points to
      output during a global bus cycle
      maximum number of FM pixels to process in one
```

```
pass, number of pixels to process before you
process one blue event. IF MAX_FM_PROCESS is 10
then you would process 1 blue event, 10 FM pixels,
1 blue event, 10 FM pixels.........
address of area in proc memory where the delta
between ping and pong is stored. It is assumed
that this is the first location of an array of
window constants (see Grab_Address_Constants)
address of area in proc memory where the number of windows is stored
bit to set/clear to toggle between ping and pong address constants
bit mask used to enable EVENT interrupts

bit mask used to disable EVENT interrupts

 N0 status bits definition
1 = First FM pass through this tracking frame, 0 = nth pass
FM_NEW_FRAME            EQU         $0

1 = ping FM addresses being used, 0 = pong FM
FM_PING_PONG_ADDRESS    EQU             $1

1 = ping FM addresses being used, 0 = pong FM
FM_PING_PONG_DATA EQU             $2

1 = collect Fast Mode Data, 0 = Not
FAST_MODE         EQU         $3

1 = time to load Fast Mode window constants
LOAD_FM_CONSTANTS EQU         $4

1 = time to sample FM Pixels, 0 = not
GRAB_FM_SAMPLES         EQU         $5

1 = time to process FM data, 0 = not
PROCESS_FM_DATA         EQU         $6

1 = time to output FM data, 0 = not
OUTPUT_FM_DATA          EQU         $7

1 = In the middle of loading source pixel addresses, 0 = not
CONTINUE_FM_ADD_LOAD    EQU         $8

1 = In the middle of processing FM data, 0 = not
CONTINUE_FM_PROCESSING  EQU         $9

1 = Fresh Fastmode Window Constants have been loaded, 0 = not
BFAST_LOADED            EQU         $A

1 = output FM event count and last address written to, 0 = not
CLEANUP_FM_OUTPUTS      EQU         $B

1 = ping pong buffers after the next calculateFM, 0 = not
FM_PP_AFTER_CALFM EQU         $C

1 = some non data output function requires servicing in
the BUSIO ISR, 0 = not
NON_DATA_ENABLED  EQU         $D

1 = reading 2nd half of windows, 0 = reading 1st half of windows
GET_REST_OF_WINDOWS     EQU         $E
```

```
1 = FM output has exceeded global storage, 0 = normal FM operation
FM_TERM_ADD_REACHED     EQU         $F

 N1 status bits definition
1 = output heartbeat response, 0 = Not
SEND_HEARTBEAT          EQU         $3

1 = doing RAM test, 0 = Not
RAM_TEST_ENABLED  EQU         $4

1 = outputting engineering mode data, 0 = Not
ENGINEERING_MODE  EQU         $5

1 = event data register interrupt is enabled, 0 = Not
EVENTS_ENABLED          EQU             $6

1 = Commanded to be the Klingon, 0 = Not
KLINGON_BIT             EQU             $7

1 = Last command OK, 0 = Not
LAST_COMMAND_STATUS     EQU             $8

1 = ping addresses being used, 0 = pong
PING_PONG               EQU             $9

1 = currently Holding Global Bus, 0 = Not
HAVE_BUS_BIT            EQU             $A

1 = time to load Image window constants
LOAD_GLOBAL_CONSTANTS   EQU             $B


      SCI stuff

control register init
11 bit multidrop
address bit wakeup
SCI sleeps until address bit set
transmitter disabled
receiver enabled
receiver interrupt enabled
```

```
NAME:
            Boot strap Loader Code

PURPOSE:
            Copy executables for the Blue processor from KAL
            powered Global Program Storage(GPS) RAM into the On
            Board RAM. These executables stored in GPS RAM
            are protected by Forward Error Correction.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            Jumps to address 0000h after code is loaded

INCLUDE FILES:
            "FecTab.asm"
            "FecCode.asm"

SUBROUTINES CALLED:
            FfwrdErrCrection -- this file.

CALLED FROM:
            Firmware internal to the DSP56001 after the reset
            signal removed from the processor.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012299      addition of standard header   jrmk


*******************************************************************
      Forward Error Correction table included here

      determine if this is blue1 or blue2 by
      sampling digital I/O pins.

      We're Blue1

      else jump to blue2 initializations

      Set blue1 address for Global status Word

      Set blue1 address for Global errors Word

      Set blue1 Global event rate storage address
```

```
set blue1 FEC address error array location

jump over blue2 initializations and
start loading executables



We're Blue2

Setup Blue2 FM constants pointer

Set blue2 address for Global status Word

Set blue2 address for Global errors Word

Set blue2 Global event rate storage address

set blue2 FEC address error array location

load executables


first initialize stack

enable access of program storage card

Initialize increment variable

Use Global Lookup Table to get location of flight Asm BIOS

Read starting address of asm code in GPS RAM

Read length of code to load

because of FEC we need to input the code in 16 word chunks
calculate the number of code chunks that need to be input

input the executables and FEC them:
      interrupts vectors into on chip RAM
      load blue.asm code into on chip RAM
      and on board memory

disable access of program storage card

Start this bad boy!!

Forward Error Correction code included here.
```

```
!
NAME:
```

# Definition of interrupt vectors

```
            used by Flight code
               ORG           P:$0000


MODIFICATION HISTORY:
      jmk101292   initial creation



******************************************************************
      Initialize interrupt vectors
      interrupt vectors

      reset/cold boot   @ P:$0000
      Stack error interrupt   @ P:$0002
      Trace interrupt   @ P:$0004
      Software interrupt      @ P:$0006
      int A, Global Bus I/O   @ P:$0008
      use a fast interrupt for
      interrupt B,store a raw photon
      ORG P:$000A
      Unused ssi interrupt receive data   @ P:$000C
      Unused ssi interrupt rec dat w/error    @ P:$000E
      Unused ssi interrupt transmit data  @ P:$0010
      Unused ssi interrupt xmit data w/error    @ P:$0012
      sci receive data  @ P:$0014
      sci rec data w/error    @ P:$0016
      fill all the remaining unused interrupt vectors with
      a jsr to the Illegal instruction interrupt handler


*****************************************************************************
            !!!!!!!!!!!!!!!START APPLICATION CODE HERE!!!!!!!!!!!!!!!
*****************************************************************************


******************************************************************
      Debugging option of pushing all executables off Chip Memory
      included here
```

```
NAME:
            Output_Data

PURPOSE:
            to update the images of the detector face stored in
            Small Word memory by incrementing the addresses
            calculated from the 24 bit event words received
            from the OM1.

CALLING SEQUENCE:
            Output_Data

INPUTS:
            Register N2 -- old processed event count
            Register N3 -- current processed event count
            Register R4 -- pointer to circular buffer of
                        calculated addresses to be incremented

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            Read_FM_Pixels -- this file

CALLED FROM:

RESTRICTIONS:
            Always output data before grabbing FM slice. If done
            in reverse order you could skew the arrival time of
            some photons in the FM window

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk


*******************************************************************
      calculate the number of events that need to be output
      number of photons to output = current-old

      update old count

      if 0 photons to output go home


      ok it's not Zero

      for system performance information:
      output calculated event rate to Proc area of global BW RAM
```

```
update all addresses contained in the output
circular buffer

see if it is time to sample the FM pixels

True,
      jsr to Read_FM_Pixels


see if there is FM data to be output

True,
initialize output registers


test for data to output
      if not :
            Clear output bit
            jmp to OutPutDataByeBye

data to output

see if fast mode data collected has exceeded
global storage allocated for output
If out of bounds...Flush it
If within bounds...Store it

if FM commanded off:
   output last address written to with FM data
Correct FM output address
(points to the Next location to fill)

output FM data event count

time to return
```

```
NAME:
            Read_FM_Pixels
PURPOSE:
            sample values stored in pixels selected for FM
            data collection

CALLING SEQUENCE:
            Read_FM_Pixels

INPUTS:
            FM_PIXELS_SAMPLED -- count of number of Valid FM Pixels
            FM_SAMPLE_ADDRESS_PT -- pointer to array of pixel addresses
            FM_SAMPLE_FP -- pointer to local storage for pixel values

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:


RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk


******************************************************************
      calculate number of pixels to grab
      put unsampled pixel count in A
      if less than or equal to maximum number of pixels to input
      then input that number. And clean up 'cause we're done.

      set flag indicating processing of FM data is required.

      put unsampled count in A, which is the number of pixels
      that have not been sampled

      more than max pixels to input,
      save updated number of pixels not sampled,

      put MAX in for number of pixels to sample in A
      setup source address buffer and destination data buffer

      input pixel values
      save source address buffer and destination data buffer pointers
```

```
NAME:
            CLC2

PURPOSE:
            Convert the serial stream of data from the OM1 into
            addresses that represent the location on the face of
            the MIC detector that had an enough charge deposited
            to trigger event processing.

CALLING SEQUENCE:
            None.

INPUTS:
            Raw serial data collected by the Fast Interrupt B routine.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            An Address in the Small Word Global Memory to increment.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            CalculateFM -- this file.

CALLED FROM:
            Default main background loop, entered into from the initialization
            code.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk


******************************************************************
      test for raw event dump mode, if enabled do not calculate addresses.
      loop forever doing nothing

      if not in raw event output mode
      test for need to process FM data

      if true jsr to "calculateFM"
      else
      see if there is events in the input buffer that need to be
      converted into addresses.

      move raw buffer fill pointer in A register
      move raw buffer empty pointer into B register and
      see if they are equal
      if equal jump back to raw event dump mode test
```

```
      else
      Calculate Global Address to Increment
            MSB                LSB
assuming photon -> 0 xbit yoff xoff wid
bit count          1   1   9    9    4
Note: xbit is no longer used, code removed 081198 jrmk

where the address =
      ((xoff + kx) * bb) + aa + yoff + ky

where aa, bb, kx, and ky are selected from 16 4-tuples based on the
value of wid.

      if debugging is enabled:
            test for generated address pointing to invalid addresses
            if true, output calculated address and in raw event
                  Global Error Word and Global Status Word, respectively.
                  the jump to raw event dump mode test
            if false, continue

      move output count into acc B
      increment output count
      put calculated address into output queue

      copy output count back to register
      do this last to prevent goof ups in output during interrupt

      test for MIC time stamp message
      if not a time stamp message jump to raw event dump mode test
      if it is a time stamp message but
      if FM is not enabled:
                  jump to raw event dump mode test
      If FM is enabled:
                  see if it is time to get another FM sample

      Time stamp messages are embedded in the MIC
      data stream, we test for them, when one is found the FM
      sample rate counter is decremented, when that count is zero it is
      time to request a sample of the FM pixels

      setup for FM Sample:
      put a zero in the sample time
      reset timer counter
      initialize number of pixels to sample
      initialize address array to use
      initialize data array to write to
      set sample required flag
      then jump to raw event dump mode test
```

NAME:

# CalculateFM

PURPOSE:

```
            Convert the number of photons collected between sequential
            samples of a pixel into a compressed data stream.

CALLING SEQUENCE:
            CalculateFM

INPUTS:
            Array of pixel values, sampled from Small Word RAM

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            CLC2

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk


******************************************************************
      save context

      is this the first pass at processing the current frame or the nth?

            if first, setup pointers and storage locations
            else, jump to calculate number of pixels to process

      figure out which is the current frame and which is the previous frame

      setup pointers to data:
            R3 should point to the new data
            R6 should point to the old data
            R7 should point to the output array

      Calculate the number of pixels to process

      if less than MAX_FM_PROCESS
            clear nth frame flag
            clear processing request flag
```

```
else
      set pixel count to process
Pack some FM data
      setup: see page 22 15Feb94 jmk lab notebook

test for new frame, ie. do not difference the data

w/differencing
subtract old pixel value form new if
there are counts, add them to the data array
count the number of events

w/o differencing
test new pixel value if
there are counts, add them to the data array
count the number of events

cleanup:
      set flag for nth processing pass
      save pointers to data:
      R3 points to the new data
      R6 points to the old data
      R7 points to the output array
      save the index of the last pixel with an event in it
      initialize ouput pointer
      restore registers

      if required,
            set ouput request
            clear the "no differencing" flag

if PROCESS_FM_DATA is set we are not done processing FM data
if it is cleared,
      clear a flag used in FM processing.
      request FM data output
      clear the "no differencing" flag
      Make the trailer
```

```
NAME:
```

# BUSIO -- Global bus interrupt handler

```
PURPOSE:
            Service periodic (~ 1 millisecond) interrupt that
            indicated opportunity to access Global Resources

CALLING SEQUENCE:
            None -- ISR

INPUTS:
            Control Bits in Packed word:
                  Global_bus_flags
                  Global_bus_flags2

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            Output_Data  -- This File
            Dump_Eng_data -- This File
            Grab_FM_Constants  -- This File
            Grab_Address_Constants  -- This File
            Mem_test -- This File

CALLED FROM:
            None - ISR.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012299      addition of standard header   jrmk


******************************************************************
      Check if we are the klingon
      if we're not the klingon lets get on with business
      if we're the klingon, is it time to give the bus back?
      Time to give the bus back!
      else
      save registers

      Set have bus flag

      increment bus cycle counter

      Update blue global semaphore and error flag
```

```
The onboard serial to parallel conversion circuit also
checks the parity of incoming 24 bit events
Test for parity errors since last 1/1024 second interrupt

if At least one parity error has occurred
wiggle Parity Error Latch Reset Bit
Increment Parity Error Count

if No parity errors or finished handling parity errors so
is there a Heartbeat request pending?
Yes,
      so write to Global RAM standard Heartbeat message
then check for RAM test request Pending
No heartbeat request,
then test for pending RAM testing request
True,
      jsr to Mem_test, upon return give back bus
Else see if there is a pending request to
input Image window constants
True,
      jsr to Grab_Address_Constants , upon return give back bus
NOTE:
      because of time required to input all constants the process
      is split between two bus cycles.
commented out 112398 when splitted grab between 2 busio cycles
else see if there is a pending request to
input Fast Mode window constants
True,
      jsr to Grab_FM_Constants, upon return give back bus
NOTE:
      because of time required to input all constants the process
      is split between two bus cycles.
else see if there is a pending request to
output raw data
True,
      jsr to Dump_Eng_data, upon return give back bus
if you add something after this uncomment the jmp
else see if there is a pending request to
output normal data
True,
      jsr to Output_Data, upon return give back bus
else, restore registers and return bus
If blue is not the Klingon give back the bus
else keep the bus and return from the ISR call
If we return the bus then Clear have bus flag
```

NAME:

# Dump_Eng_data

PURPOSE:

```
output raw 24 bit event words that have been collectd.
disable this mode after XXXXXX events have been output
```

CALLING SEQUENCE:

```
Dump_Eng_data
```

INPUTS:

None.

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

None.

FUNCTIONS IN THIS FILE:

None.

INCLUDE FILES:

None.

SUBROUTINES CALLED:

None.

CALLED FROM:

RESTRICTIONS:

None.

MODIFICATION HISTORY:

```
jmk101292   initial creation
012399      addition of standard header   jrmk
```

```
******************************************************************

Save registers
initialize registers used for outputting data
output no more than MAX_RAW_OUT events
setup increment variable
get next address to fill
save next address to write to
get last address to fill so we can test if we are done
if we're done
Exit engineering mode
Stop data acq. by disabling interrupts
flush input buffer by setting raw fill and empty pointers equal
and clean raw data mode indicator flag
restore registers
```

NAME:

# ESCIRec -- SCI Recevier interrupt with framing error

PURPOSE:

To handle framing errors in message words from other processors internal to the DPU. When error is detected any command fragment currently held is flushed.

CALLING SEQUENCE:

None.

INPUTS:

None.

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

None.

FUNCTIONS IN THIS FILE:

None.

INCLUDE FILES:

None.

SUBROUTINES CALLED:

None.

CALLED FROM:

Invoked by SCI Receiver hardware when a Framing Error is detected.

RESTRICTIONS:

None.

MODIFICATION HISTORY:

```
      jmk101292   initial creation
      012399      addition of standard header   jrmk
```

```
******************************************************************
      save registers
      flush input command buffer
      clear received byte count
      send receiver back to sleep
      clear start of collection flag
      reset fill and empty pointers
      note commanding error to be sent out on next heartbeat request
      restore registers
      Later Dude
```

```
NAME:
```

# SCIRec -- SCI receiver interrupt

```
PURPOSE:
            To input and format bytes onto complete SCI messages.

CALLING SEQUENCE:
            None.

INPUTS:
      ASSUME:
            order of xmission: Address, Length, CID, Low, Mid, High, Low ......

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            Invoked by SCI Receiver hardware when a byte is received.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk


******************************************************************
      Save registers
      are we currently collecting a message?
      if yes jump past initialization and go see
      what part of the message we are waiting for
      not collecting, check address to see if it's for us
      Is the message for this processor?
      Is the message for everyone?
      Is the message for all blue's?
      Not a message for us, put SCI back to Sleep and return
      is a message for us:
      Start receiving a new message by:
            clearing message length count
      Start receiving a new message by:
         clearing message length count
         set start of collection flag
      clear flags indicating:
            CID was input
            hi nibble received
            middle nibble received
```

```
      low nibble received
then return until next nibble is received
What part of the message are we waiting for?
      input command LENGHT
set up for next input, then return
      input Command ID
set up for next input, then return
      input Low Byte of next data word
set up for next input, then return
      input Middle Byte of next data word
set up for next input
test number of nibbles received against command length
if equal jump to command interpreter, else return
      input High Byte of next data word
setup for next input
test number of nibbles received against command length
if equal jump to command interpreter, else return
If full command received:
Put SCI back to Sleep
clear start of collection flag
execute command
restore registers
```

```
NAME:
```

# COMMAND_INTERPRETER

```
PURPOSE:
            to interpret and execute SCI commands sent from the White DSP

CALLING SEQUENCE:
            None. Inputs are passed through globals.

INPUTS:
            Current command is pointed to by the sci_empty_pointer.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            SCIRec -- This file

RESTRICTIONS:
      Called from the SCI interrupt, expecting A, B, X1, R7 and M7 to already
      be saved by the caller. And to be replaced by the caller.
      Command ID:
            $0A0 == Heart beat request
            $0A1 == Use Ping address space constants
            $0A2 == Use Pong address space constants
            $0A3 == Enable Klingon
            $0A4 == Disable Klingon
            $0A5 == Enable Events
            $0A6 == Disable Events
            $0A7 == Enable raw data output
            $0A8 == Disable FM data output
            $0A9 == Enable RAM test
            $0AA == Disable RAM test
            $0AB == Load image window constants
            $0AC == Load Fast Mode window constants
            $0AD == Enable FM data output
            $0AE == Echo ping address calculation constants
            $0AF == Echo pong address calculation constants

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk
```

# List of SCI Commands

```
Command ID:
            $0A0 == Heart beat request

Action:
      set flag indicating heartbeat has been requested


*****************************************************************
Command ID:
            SEND_COUNTS_TO_PING == Use Ping address space constants

Action:
      Ping/Pong image address calculation arrays
      change address register to point to Ping
      echo change in system status flags
      If FM enabled:
      force a FM slice
      FORCE SLICE:
      initialize output array fill and empty pointers.
      initialize number of pixels to sample
      initialize address array to use
      initialize data array to write to
      set sample required flag
      Set addresses to sample from
      Set destination for storage of FM data
      echo change in system status flags



*****************************************************************
Command ID:
            SEND_COUNTS_TO_PONG == Use Pong address space constants
Action:
      Ping/Pong image address calculation arrays
      change address register to point to Pong
      echo change in system status flags
      If FM enabled:
      force a FM slice
      FORCE SLICE:
      initialize output array fill and empty pointers.
      initialize number of pixels to sample
      initialize address array to use
      initialize data array to write to
      set sample required flag
      Set addresses to sample from
      Set destination for storage of FM data
      echo change in system status flags



*****************************************************************
Command ID:
            ENABLE_KLINGON == Allow this DSP to be the Klingon
Action:
      set bit in N1 requesting this DSP be the Klingon

      echo change in system status flags
```

```
*****************************************************************
Command ID:
            DISABLE_KLINGON == Make this DSP not the Klingon

Action:
      clear bit in N1 requesting this DSP be the Klingon

      echo change in system status flags


*****************************************************************
Command ID:
            ENABLE_EVENTS == Enable Event Interrupt

Action:
      Turn on event interrupt

      reset blue data interrupt(flush input buffer)
      enable interrupt B (store a raw photon)


*****************************************************************
Command ID:
            DISABLE_EVENTS == Disable Event interrupt

Action:
      Exit engineering mode
      Stop data acq. by disabling interrupts
      flush input buffer by setting raw fill and empty pointers equal
      and clean raw data mode indicator flag
       disable events processing


*****************************************************************
Command ID:
            ENABLE_ENGINEERING == Enable raw data output

Action:
      set bit in N1 requesting ENGINEERING_MODE

      setup raw data addresses
      signal start of raw event dump



*****************************************************************
Command ID:
            DUMP_LOCAL_RAM == Write contents of local RAM into 0x0FFFFF

Action:
      set bit in N1 requesting DUMP_LOCAL_RAM
      NOT USED



*****************************************************************
Command ID:
            ENABLE_RAM_TEST== Start testing global RAM
```

```
Action:
      set bit in N1 requesting RAM_TEST_ENABLED

       Make sure events are disabled.
       setup test parameters


*****************************************************************
Command ID:
            DISABLE_RAM_TEST == Disable test of global RAM before completion

Action:
      clear bit in N1 requesting RAM_TEST_ENABLED


*****************************************************************
Command ID:
            GET_DTW_PARAMETERS == Load image window constants

Action:
      set bit in N1 requesting LOAD_GLOBAL_CONSTANTS


*****************************************************************
Command ID:
            GET_BFAST_PARAMETERS == Load Fast Mode constants

Action:
      set bit in N1 requesting LOAD_FM_CONSTANTS


*****************************************************************
Command ID:
            ENBL_FAST_DATA_OUTPUT == enable FM data output

Action:
      If a fresh set of bfast has not been loaded since last
      FM exposure was terminated, send error message and
      return
      If number of time markers between FM samples == 0
      don't enable FM
      initialize pointer to which address array to use
      initialize output array fill and empty pointers.
      initialize FM sample rate counter
      clear index of last pixel where an event was seen
      zero out event count
      enable FM data acquisition
      indicate not to delta new and old data arrays
      indicate plenty of global storage for FM data
      set a flag used in FM processing.


*****************************************************************
Command ID:
            DSBL_FAST_DATA_OUTPUT == Disable FM data output

Action:
      disable FM
      clear flag indicating that we have a fresh set of bfast parameters
      Test to see if we are currently acquiring a final FM sample
```

```
      Nope, so:
      FORCE SLICE:
      initialize output array fill and empty pointers.
      initialize number of pixels to sample
      initialize address array to use
      initialize data array to write to
      set sample required flag
      Set addresses to sample from
      Set destination for storage of FM data

*****************************************************************
Command ID:
            $0AF == nothing yet

Action:
      None


 - what's the context of this next bit?
*****************************************************************
      update pointers and return



*****************************************************************
      Process bad command stuff
      Get pointer to receiver buffer
      point to after the bad command(s)
      indicate bad command
```

```
NAME:
            Grab_FM_Constants

PURPOSE:
            input from PROC Area of Big Word Memory all
            information required to collect and output
            FM Data

CALLING SEQUENCE:
            Grab_FM_Constants

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:


RESTRICTIONS:
       load Fast Mode constants:
      Format in Memory:
            Blue1 number of time markers between FM samples
            Blue1 Total Number of Pixels
            Blue1 Data Destination
            Blue1 Last Address Written to (loaded by disable FM)
            Blue1 Total number of Fast events
            Blue1 Pixel Address array (0..511)
            Blue2 number of time markers between FM samples
            Blue2 Total Number of Pixels
            Blue2 Data Destination
            Blue2 Last Address Written to (loaded by disable FM)
            Blue2 Total number of Fast events
            Blue2 Pixel Address array (0..511)
MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk


******************************************************************
      save registers
      setup increment
      read the delta between ping and pong starting addresses and
      store in Y0
      get the base address of the ping frame in global memory
```

```
are we in the middle of inputting pixel addresses?
No
calculate which DSP area to input from
input the Fast Mode sample rate
initialize delay between FM samples
Save the address before you check the sample time
If sample time == 0 return, configuration error
Replace the address, the sample time was non-zero
input the number of fastmode pixels to input
input the destination address of Fast Mode data
input the Last Available address for storage of Fast Mode data
increment past storage location for last address FM data stored in
save starting global address of pixel address array
initialize address array fill pointer
Yes this is the nth load of FM Constants
Initialize addresses array fill pointer for ping, R7
Initialize addresses array fill pointer for pong, R6
calculate the number of addresses to input this bus cycle
      if this pass read all of them clear flags
      else adjust counters and continue
initialize global source address, A
input addresses
calculate current fill pointer for pong by adding the
length of ping to the ping fill pointer
This pass finishes the load up!
save the state, ie. next global address to read from
and the next local address to write to
restore registers
```

NAME:

# Grab_Address_Constants

PURPOSE:

input from PROC Area of Big Word Memory all
information required to collect and output
Image Data

CALLING SEQUENCE:

INPUTS:

None.

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

None.

FUNCTIONS IN THIS FILE:

None.

INCLUDE FILES:

None.

SUBROUTINES CALLED:

None.

CALLED FROM:

RESTRICTIONS:

```
# of image windows (n)
 #1stuff1
 #1stuff2
 #1stuff3
 #1stuff4
 #1stuff5
 #1kx
 #1bb
 #1aa
 #1ky
   .
   .
   .
   .
 #nstuff1
 #nstuff2
 #nstuff3
 #nstuff4
 #nstuff5
 #nkx
 #nbb
 #naa
 #nky
# of Fast Mode windows (n)
FM Stuff .................
```

ASSUME: PROC_ADD_OF_FRAMESIZE points to the start of this information

```
       ASSUME: Address of PROC_ADD_OF_N_WINDOWS is $001b6a

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk


*******************************************************************
      Save Registers
      default to loading 16 windows, SU code will establish trash baskets in
      unused windows.
      which set to grab? first 8 or second 8
      restore registers
```

NAME:

# Mem_test:

PURPOSE:

Increment the last 265 locations on each
RAM board three times. During each global bus
a different RAM board is incremented.

CALLING SEQUENCE:

SCI command from White

INPUTS:

MEM_TEST_COUNT initialized by SCI command ENABLE_RAM_TEST
MEM_TEST_SWITCH initialized by SCI command ENABLE_RAM_TEST

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

The white processor is informed of the end of
test when the processor status in the heartbeat
reply indicates memory test is finished.

FUNCTIONS IN THIS FILE:

None.

INCLUDE FILES:

None.

SUBROUTINES CALLED:

None.

CALLED FROM:

RESTRICTIONS:

None.

MODIFICATION HISTORY:

```
      jmk101292   initial creation
      012399      addition of standard header   jrmk


*******************************************************************
      Save register
      test MEM_TEST_SWITCH to see which Global RAM Board we are testing

      if falls through then error condition, end of test
      else, if branched to, then end of test
      Test board 0x700000 and update pass count
      Test board 0x600000
      Test board 0x500000
      Test board 0x400000
      Test board 0x080000
      Test board 0x000000
      increment the assigned addressed
      update switch pointer
      restore registers
```

```
NAME:
```

# Illi -- Illegal interrupt

```
PURPOSE:
            ISR stub to provide indication that a non-valid op-code
            was executed. This can occur when programming errors
            cause the attempted execution of data areas.
            Used for debugging.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            Invoked by circuitry internal to the DSP56001.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk

******************************************************************
```

NAME:

# Swi -- Software Interrupt

PURPOSE:

ISR Stub for possible later implementation of the Software Interrupt Instruction.
Used for debugging. *Note: Not currently used.*

CALLING SEQUENCE:

None.

INPUTS:

None.

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

None.

FUNCTIONS IN THIS FILE:

None.

INCLUDE FILES:

None.

SUBROUTINES CALLED:

None.

CALLED FROM:

None.

RESTRICTIONS:

None.

MODIFICATION HISTORY:

```
jmk101292   initial creation
012399      addition of standard header   jrmk
```

*****************************************************************

```
NAME:
```

# Stkerr -- Stack error interrupt

```
PURPOSE:
            ISR stub to provide indication when a stack error occurs
            Used for debugging.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            Invoked by circuitry internal to the DSP56001.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk

*****************************************************************
```

```
NAME:
```

# Trace

```
PURPOSE:
            ISR stub.
            Used for debugging. Note: Not currently used.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            None.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk

******************************************************************
```

```
NAME:
```

# Ssi1 ------ This interface is NOT USED

```
PURPOSE:
            ISR for SSI Receiver input buffer full.
            If entered, will cause system to reset.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            None.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk

******************************************************************
```

```
NAME:
            Ssi2

PURPOSE:
            ISR for SSI receiver overflow error.
            If entered, will cause system to reset.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            None.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012299      addition of standard header   jrmk
```

```
NAME:
```

# Ssi3 ------ NOT USED

```
PURPOSE:
            ISR for SSI Transmitter output buffer empty.
            If entered, will cause system to reset. Note: Currently not used.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            None.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk

******************************************************************
```

```
NAME:
```

# Ssi4 -- NOT USED

```
PURPOSE:
            ISR for SSI Transmitter underflow error.
            If entered, will cause system to reset.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            None.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      102498      addition of standard header   jrmk

******************************************************************
```

```
NAME:
            start -- System Initialization

PURPOSE:
            To initialize red processor to a known state
            after power up or a reset.

CALLING SEQUENCE:
            None.

INPUTS:
            None.

OPTIONAL INPUT PARAMETERS:
            None.

OUTPUTS:
            None.

FUNCTIONS IN THIS FILE:
            None.

INCLUDE FILES:
            None.

SUBROUTINES CALLED:
            None.

CALLED FROM:
            Invoked when a reset interrupt is triggered.

RESTRICTIONS:
            None.

MODIFICATION HISTORY:
      jmk101292   initial creation
      012399      addition of standard header   jrmk


*******************************************************************

      reset peripheral interface
      initialize on chip wait state generator
      ext X, ext Y, ext P, ext I/O
      Initialize predefine registers
      initialize N0 to:
            FM using Ping addresses
      initialize N1 to:
      Last command OK
      Image using Ping
      initialize local copy of Global Blue Error Flag to 0
      If this processor is defined at power up to be the Klingon
      set the appropriate flag bit in N1
      initialize registers for calculation of addresses
      Make port B a parallel I/O port
      Initialize SCI port
      Flush SCI hardware Receiver buffer
      Set priority levels for the following interrupts:
      ENABLE SCI @ IPL = 1, A @ IPL = 0, B off
```

```
initialize SCI software buffers and flags
determine if this is blue1 or blue2 by
sampling digital I/O pins.
assume that we are blue or red
if we are Red, cause an unnatural death
if not Red, are we BLUE 1 or 2

We're Blue1
setup Blue1 FM constants pointer
setup blue1 heartbeat response
Set blue1 address for SCI
Set blue1 address for global status word
Set blue1 address for global errors word
Set blue1 global event rate storage address
set blue1 raw data dump addresses in Global RAM
go to initialization of registers
We're Blue2
Setup Blue2 FM constants pointer
setup blue2 heartbeat response
Set blue2 address for SCI
Set blue2 address for global status word
Set blue2 address for global errors word
Set blue2 global event rate storage address
set blue2 raw data dump addresses in global RAM

initialization of hardware registers
initialize parity and data receiver registers and buffers
If for any reason we load and use FfwrdErrCrection
non-ROM version set flag to indicate that
we should not fix the errors globally
(FFixGlobalErrors)
enable interrupts
return global bus by
wiggling Return Global Bus Bit
Jump to Main background loop to calculate address in global
memory of each photon
```

# FecCode.asm Detailed Design Document

```
FILE:
            FecCode.asm

PURPOSE:
            To check a 16 opcode block against the associated
            forward error correction bits.

FUNCTIONS IN THIS FILE:
            FfwrdErrCrection - C Callable.
            returns a "1" to a C-Routine if an error was corrected.

INCLUDE FILES:
            None

MODIFICATION HISTORY:
            121597      jrmk  creation
            060898      jrmk  header added

Calling requirements:
            ParityWord,      Storage 1 word
            Parity,          Storage 1 word
            ValueA,          Storage 1 word
            ValueANot,       Storage 1 word
            ValueT,          Storage 1 word
            ValueTNot,       Storage 1 word
            NextReadFromAddress, Storage 1 word
            assumed to be loaded with a pointer to next address
            in Program Storage RAM that will be read from
            (used to calculate address of opcode in error)
            FDataBlock,      Storage 16 words
            Address of buffer that program block to be verified was
            written into. (15 opcodes followed by 1 FEC word).

            FFECTable, Table of FEC masks, assumed already loaded
            if !WHITE_ERRORS_ADDRESS, address in global RAM where count of FEC
                  is kept.
            BlueFECErrorList,
            RedFECErrorList,
                  address in global RAM where list of
                  Program Storage RAM fixed addresses is
                  kept (circular buffer).


      091498      changed RedFECErrorList to FRedFECErrorList jrmk



** RCS $Id: FecCode.asm,v 1.6 1998/09/23 22:21:04 ho Exp $


******************************************************************
      9=number of masks that need to be ANDed with each of the 15 program
      words to FEC 360 bits (15 words * 24 bits)
```

```
for each program word: read the program word
.AND. it with each of it's masks (9) .XOR. them together
to generate the check word calculate the parity of the
check word and save that bit as the check bit for this row
of FEC masks.

repeat for all 15 program words to generate 15 check bits.
.OR. each check bit into a master check word then shift the word
1 position left before .ORing. the next check bit in.

Now that we have the parity word, that has been generated by ORing
together all of the parity bits generated by masking the program
words with successive rows of FEC masks, we shall generate a parity
bit for the parity word.

This will be done by taking the parityWord that was calculated on
the ground, stripping off it's parity bit, XORing it with the 15
program words, calculating the parity of the resulting word then
ORing that Parity bit into the ParityWord that was calculated here.

read the ParityWord that was calculated on the ground
and XOR it with the newly calculated parity word

if there have been zero errors the XOR product shaould be 0

if XOR product < $200 then there has been a multiple bit error
if XOR product > $200 then there has been a single bit error

so if there has not been an error, store the 15 program bytes
   if there has been an error, and it's a single bit error fix it
   if there has been a multiple bit error report it.

Calculate which bit is in error:
A contains syndrome bits
B is used to calculate which words is corrupted, and which bit

Subtract 24 from B and compare the difference to 24, if greater that
24 repeat the process. The number of times that you can subtract
24 tells you what word the error occurred in, the remainder what bit
was flipped

Fix the Bit:
      Y0 contains the number corresponding to the word in error
      B contains the number that corresponds to the bit in error
      in the word Y0
read the Program Word that contains the error
and bit change the errant location
generate a bit mask T
now to change the value of the bit that is in error
(A* .AND. T) .OR. (A .AND. T*)
replace the corrected word
save it for if we are scrubbing we will need it to fix the GPS RAM


Tell someone about the multiple bit error
by:
 indicate multi bit error by write to status word

increment FEC error count
calculate the address in memory where the error is and put it in
```

the Global Address Error Array

# Xmmon_std.asm Detailed Design Document

```
NAME:
            xmmom_std.asm

PURPOSE:
            Basic, assembly language functions called from 'C' to access global data
            and the assembly language routines that support them.

FUNCTIONS IN THIS FILE:
            read_int_array
            read_int
            write_int
            write_int_array
            zero_global

INCLUDE FILE:
            NONE

MODIFICATION HISTORY:
      6-Dec-95    Creation by jrmk & CH
      9-Jan-96  jrmk   add standard comment header
      15-feb-96   ch move zero_frame in
      18-nov-98   splitting of xmmom_std.asm into  xmmom_std.asm and
                  xmmom_std2.asm. xmmom_std.asm for asm code
                  that should operate out of the Zero Wait State On Chip RAM and,
                  xmmom_std2.asm for the asm code that does not need to be on chip.
                  Also included is zero_global.
                  This will help alleviate bus hogging conditions.
                        Fread_int         ;c callable --- removed
                        Fread_int_array   ;c callable --- removed
                        Fwrite_int_array  ;c callable --- removed
                        Fwrite_int        ;c callable --- removed
                        Fzero_frame       ;c callable --- removed
```

NAME:

read_int_array

PURPOSE:

CALLING SEQUENCE:

INPUTS:

```
Int base;
Int ysize;
Int yskip;
Int xsize;
Int *destination;
```

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

None.

FUNCTIONS IN THIS FILE:

None

INCLUDE FILES:

None.

SUBROUTINES CALLED:

None.

CALLED FROM:

RESTRICTIONS:

None.

MODIFICATION HISTORY:

*******************************************************************

NAME:

read_int

PURPOSE:

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:
None.

OUTPUTS:
None.

FUNCTIONS IN THIS FILE:
None

INCLUDE FILES:
None.

SUBROUTINES CALLED:
None.

CALLED FROM:

RESTRICTIONS:
None.

MODIFICATION HISTORY:

*****************************************************************

NAME:

write_int

PURPOSE:

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:
None.

OUTPUTS:
None.

FUNCTIONS IN THIS FILE:
None

INCLUDE FILES:
None.

SUBROUTINES CALLED:
None.

CALLED FROM:

RESTRICTIONS:
None.

MODIFICATION HISTORY:

*****************************************************************

NAME:

write_int_array

PURPOSE:

CALLING SEQUENCE:

INPUTS:

OPTIONAL INPUT PARAMETERS:
None.

OUTPUTS:
None.

FUNCTIONS IN THIS FILE:
None

INCLUDE FILES:
None.

SUBROUTINES CALLED:
None.

CALLED FROM:

RESTRICTIONS:
None.

MODIFICATION HISTORY:

*****************************************************************

NAME:

zero_global

PURPOSE:

Initialise a range of Global RAM to zero

CALLING SEQUENCE:

None.

INPUTS:

Packed bit flags stored in Fglobal_bus_flags

OPTIONAL INPUT PARAMETERS:

None.

OUTPUTS:

None.

FUNCTIONS IN THIS FILE:

None

INCLUDE FILES:

None.

SUBROUTINES CALLED:

None.

CALLED FROM:

RESTRICTIONS:

None.

MODIFICATION HISTORY:

*****************************************************************

Definitions

Big Word Memory: The Global RAM that is 24 bits wide. Global Address: 0x000000 through 0x0FFFFF

Blue1, Blue2: The two Motorola DSP56001 Digital Signal Processors tasked with converting the serial stream of data from the OM1 into addresses that represent the location on the face of the MIC detector that had an enough charge deposited to trigger event processing.

Chunk or Chunk of Executables: A set of sixteen 24 bit words stored in GPS RAM, that contain, in order, 15 DSP56001 opcodes and 1 FEC check word.

CID: Command I.D.

FEC: Forward Error Correction. This implementation uses a series of mask bit patterns to generate a check word that is embedded in the executables stored in GPS RAM (fifteen 24 bit opcodes followed by one 24 bit check word).

Fglobal_bus_flags: A 24 bit word that is used for storage of 24 flags. Each indicating: A state the software is in. Or a request for special processing.

Fglobal_bus_flags2: Same as Fglobal_bus_flags.

FM: Fast Mode

Global Program Storage RAM: KAL backed up 512K 24Bit RAM on the Global Bus which provides volatile program storage for all four processor boards internal to the DPU.

Global RAM: Global Variable storage area in On Board RAM

GPS RAM: see Global Program Storage RAM.

ISR: Interrupt Service Routine

KAL: Keep A Live. Power that is provided to keep volatile storage valid. In the DPU portion of the OM2 it is used to maintain the contents of the 512K 24 bit RAM location that are used as GPS RAM.

Klingon: The global bus is arbitrated in a round-robin fashion. Each processor gaining access to the global bus via an interrupt indicating it's availability. Each processor uses the global bus then returns it when it has finished accessing global resources. One of the processors is assigned the Klingon responsibility. The Klingon, after completing it's global bus transactions, exits the BUSIO ISR but DOES NOT relinquish the global bus to the next processor, rather it waits for an interrupt indicating a 1/1024 second time tick has passed. When it sees this millisecond interrupt it releases the global bus which is then passed from processor to processor until it arrives back to the Klingon processor (Yes, we do watch Star Trek).

loader.lut: A look up table located in GPS RAM that gives the starting address (in GPS RAM) and the length of every Swap Unit and Executable stored in GPS RAM.

MIC time stamp message: 0x800000. Two of these bit patterns are sent from the OM1 detector electronics at the end of every CCD frame. Counting the occurrence of these bit patterns is used to generate the delays between sampling the Fast Mode Pixels.

On Board RAM: 32K X 24Bit RAM on each of the four (4) processor boards (XMM-11F).

On Chip RAM: 256 X 24Bit zero wait state RAM built in the DSP56001 CPU.

SCI: Serial Communications Interface, a serial interface protocol that is supported by circuits contained in the DSP56001 processor from Motorola. The SCI link is an "Open Drain" interface, (ie. Several DSP56001s can have their respective SCI pins wired together to form a low data rate network.) in the OM2

Small Word Memory: The Global RAM that is only 16 bits wide. Global Address: 0x400000 through 0x7FFFFF.

SSI: Synchronous Serial Interface, a serial interface protocol that is supported by circuits contained in the DSP56001 processor from Motorola. The SSI interface is a half duplex three wire interface, (ex. NRZ Data, Clock, Envelope). This interface is used for communications between the ICU and DPU.

Swap Unit: A self contained unit of High Level Science Code, written in C, stored in it's executable form, that can be overlaid into a portion of On Board RAM and executed (i.e. the Swap Unit Track_Guide_Starts.c).