```
       COMMENT *

This file is used to generate DSP code for the second generation
        timing boards to operate one 512 x 512 pixel quadrant of
        a HAWAII-1R  array with two coadder boards.

    *

    PAGE    132     ; Printronix page width - 132 columns

; Define a section name so it doesn't conflict with other application programs
        SECTION TIM_HAWAII_1RG

; Include a header file that defines global parameters
        INCLUDE "../DSPlib/timhdr.asm"

APL_NUM EQU     0               ; Application number from 0 to 3
CC      EQU     TIMREV4+NGST+COADDER

SCR     EQU     $FFF0   ; SCI Control Register
SSR     EQU     $FFF1   ; SCI Status Register
SCCR    EQU     $FFF2   ; SCI Clock Control Register
WRSCI   EQU     $FFF4   ; Write least significant byte to SCI port
WRSCI0  EQU     $FFF4   ; Write least significant byte to SCI port
WRSCI1  EQU     $FFF5   ; Write middle significant byte to SCI port
WRSCI2  EQU     $FFF6   ; Write most significant byte to SCI port
SRAM_AD EQU     $5F0    ; Address in timing board SRAM where coadder code is
P_COADD EQU     $E001   ; Start of coadder code in EEPROM
RST_CA  EQU     3       ; Timing board resets coadder board (Rev 4B, which IDTL has)
                        ;    NOTE error in docs! should be 4B=3 3B=5

; Software status bit definitions for the program only
RST_MODE EQU    16      ; 0 for global reset, 1 for row-by-row reset
CDS_MODE EQU    17      ; 0 for single sampling, 1 for CDS
PTR_MODE EQU    18      ; 0 for Coadder arithmetic, 1 for pass through mode
UTR_MODE EQU    19      ; 0 for normal, 1 up-the-ramp readout
BO_MODE  EQU    20      ; 0 for normal, 1 Bright Object




;******************************************************************************
;****  CLOCKING CODE IN SRAM PROGRAM SPACE  ****
;******************************************************************************
;  Specify execution and load addresses
        IF      @SCP("DOWNLOAD","HOST")
        ORG     P:APL_ADR,P:APL_ADR             ; Download address
        ELSE
        ORG     P:APL_ADR,P:APL_NUM*N_W_APL     ; EEPROM address
        ENDIF


;******************************************************************************
; Normal readout of array with global reset
RD_ARRAY
        JSR     <PCI_READ_IMAGE ; Tell the PCI card the number of pixels to expect
        ; first, calculate some numbers
        MOVE    Y:<NCOLS,A1             ; Get total number of cols,
        ASR     A                       ; Divide by two (2 quads wide)
        MOVE    A1,Y:<TEMP              ; store temporarily

        DO      #1500,DLY_ON             ;   This delay needs to be here - not sure why
        JSR     <PAL_DLY
        NOP
DLY_ON

        BSET    #WW,X:PBD               ; Set WW to 1 for 16-bit image data


        MOVE    #VSOURCE_ON,R0           ; Turn VSOURCE ON
        JSR     <CLOCK
```

```
        MOVE    #FRAME_INIT,R0         ; Initialize the frame for readout
        JSR     <CLOCK



; use this block for ROW skipping (this is burst-mode subarray)

        MOVE    Y:<STARTROW,A
        MOVE    X:ONE,X0
        CMP     X0,A                            ;   if # of rows to skip is zero, then don't
        JEQ     <NOROWSKIP
        DEC     A
        DO      A,ROWSKIP
        MOVE    #CLOCK_ROW_FAST,R0
        JSR     <CLOCK
        NOP
ROWSKIP
        NOP
NOROWSKIP



; The first two pix reads have no XFER or SXMIT because the A/D data are not valid
; (pipeline delay in the ADS937 A/D converter)

L_READ_FIRST_TWO                        ; clock (no read) first 2 pix of first row

        MOVE    #READ_ON,R0            ; Turn Read ON
        JSR     <CLOCK


        JSET    #RST_MODE,X:STATUS,RR_RD1
        MOVE    #CLOCK_ROW,R0            ; if not row-by-row reset
        JSR     <CLOCK
        JMP     <RR_RD2
RR_RD1  MOVE    #CLOCK_ROW_RESET,R0      ; if row-by-row reset
        JSR     <CLOCK
RR_RD2  NOP


        ; for 1RG, need 2 HCLK pulses before the first real pixel in each row
        DO      #2,SKIP_0       ; This only affects the first row
        MOVE    #CLOCK_ONLY,R0  ; load clock only
        JSR     <CLOCK          ; execute
        NOP
SKIP_0


; use this block for COL skipping - this is done once here for the special
; first row, then again inside the column loop  (this is for burst-mode subarray)
        MOVE    Y:<STARTCOL,A
        MOVE    X:ONE,X0
        CMP     X0,A                            ;   if # of cols to skip is zero, then don't
        JEQ     <NOCOLSKIP_1
        DEC     A
        DO      A,COLSKIP_1
        MOVE    #CLOCK_ONLY_FAST,R0
        JSR     <CLOCK
        NOP
COLSKIP_1
        NOP
NOCOLSKIP_1



        JSET    #BO_MODE,X:STATUS,RR_BO       ; if BO mode selected, jump to the BO method
        ; 2 pix, so do this twice
        DO      #2,FIRST_TWO_PIX         ;Loop two times
        MOVE    Y:<CLK_COL_AND_READ_NO_XFER,R0  ;load waveform to fill A to D pipeline
```

```
        JSR     <CLOCK          ; clock it
        NOP
FIRST_TWO_PIX
        JMP     <REMAIN         ; skip over the following BO mode block


; *************************************************************************************************
********
; This is BO mode. Since BO reads twice per pix, do this once with NO_XFER, then once with XFER t
o get 2 REAL pix

; first thing to do in BO mode is calculate the number of iterations for
; each of the delay loops

; n = ((time - 2) * 25.0)

RR_BO   MOVE    X:<EXP_TIM,A
        MOVE    X:<TWO,Y0
        SUB     Y0,A

        MOVE    A,Y0
        MOVE    #0,A
        DO      #25,SUM_LOOP ; integer multiply by 25
        ADD     Y0,A
        NOP
SUM_LOOP
        MOVE    A,Y0    ; this is n for the exposure delay loop

; pix 1  #########################################################
        MOVE    #CLOCK_ONLY,R0     ; clock to pixel
        JSR     <CLOCK
        NOP
        MOVE    #PULSE_RESET,R0         ; strobe the reset on this row
        JSR     <CLOCK
        NOP

        ; this loop takes ((n*40e-9) + 2000.0e-9) s to execute
        ; n=200 is 10 us
        ; put a settling time delay here (10 us)
        ;
        DO      #200,SET_DLY1               ; Each step of this loop is 1 us
        NOP
SET_DLY1



        JCLR    #CDS_MODE,X:STATUS,BO_RD_END_1
        CLR     B
        DO      Y:<NFS,BO_RD_END_1                          ; Fowler sampling

        ; first BO read(s)

        ; first 2 reads get no XFER (pipeline delay)
        MOVE    X:<TWO,Y1
        CMP     Y1,B
        JGE     W_XFER                  ; if two reads already done, go to W_XFER
        MOVE    #A2D_ONLY_NO_XFER,R0
        JSR     <CLOCK
        MOVE    X:<ONE,X0
        ADD     X0,B
        JMP     <RDONE
        ; Any reads (fowlers) beyond  2 here get XFERed
W_XFER  MOVE    #A2D_ONLY,R0
        JSR     <CLOCK
RDONE


        NOP
BO_RD_END_1
```

```
; Implement up-the-ramp readout
        DO      Y:<IUTR,BO_L_UP_THE_RAMP_1


        ; delay loop goes here for exptime in BO mode
        ; this is the number to delay by -  X:<EXP_TIM
        ; in BO mode, this is assumed to be microseconds, not
        ; milliseconds as in NORMAL mode.
        ;
        ; the number in the do loop should be:
        ; n = ((time - 2e-6)/40e-9)
        ; where time is in seconds
        ;
        ; or if time is in microseconds:
        ; n = ((time - 2) * 25.0)
        ;
        ; exposure time should be an integer number of microseconds
        ; and min exptime is 3 microseconds.
        ;
        ; actual time per pix is:
        ; t = time + 22us + settling_time
        ;
        ; for BO mode though, the commanded exptime should be the
        ; full desired exposure time.
        ;

        DO      Y0,EXP_DLY1               ; Each step of this loop is 1 us
        NOP
EXP_DLY1


; Sample the signal level NFS times
        DO      Y:<NFS,BO_RD_EXP_1                           ; Fowler sampling

        ; second BO read

        ; first 2 reads get no XFER (pipeline delay), so continue to check
        MOVE    X:<TWO,Y1
        CMP     Y1,B
        JGE     W_XFER1          ; if two reads already done, go to W_XFER
        MOVE    #A2D_ONLY_NO_XFER,R0
        JSR     <CLOCK
        MOVE    X:<ONE,X0
        ADD     X0,B
        JMP     <RDONE1
        ; Any reads (fowlers) beyond  2 here get XFERed
W_XFER1 MOVE    #A2D_ONLY,R0
        JSR     <CLOCK
RDONE1


        NOP
BO_RD_EXP_1
        NOP
BO_L_UP_THE_RAMP_1
        NOP


; pix 2  #########################################################
        MOVE    #CLOCK_ONLY,R0     ; clock to pixel
        JSR     <CLOCK
        NOP
        MOVE    #PULSE_RESET,R0          ; strobe the reset on this row
        JSR     <CLOCK
        NOP

        ; put a settling time delay here (10 us)
        DO      #200,SET_DLY2              ; Each step of this loop is 1 us
        NOP
SET_DLY2
```

```
        JCLR    #CDS_MODE,X:STATUS,BO_RD_END_2
        DO      Y:<NFS,BO_RD_END_2                          ; Fowler sampling

        ; first BO read(s)
        MOVE    #A2D_ONLY,R0
        JSR     <CLOCK

        NOP
BO_RD_END_2

; Implement up-the-ramp readout
        DO      Y:<IUTR,BO_L_UP_THE_RAMP_2

        ; delay loop goes here for exptime in BO mode
        ; this is the number to delay by -  X:<EXP_TIM
        DO      Y0,EXP_DLY2               ; Each step of this loop is 1 us
        NOP
EXP_DLY2

; Sample the signal level NFS times
        DO      Y:<NFS,BO_RD_EXP_2                          ; Fowler sampling

        ; second BO read
        MOVE    #A2D_ONLY,R0
        JSR     <CLOCK

        NOP
BO_RD_EXP_2
        NOP
BO_L_UP_THE_RAMP_2
        NOP

; *************************************************************************************************
********




REMAIN  MOVE    Y:<TEMP,A1      ;total number of cols/2
        MOVE    X:<ONE,X1
        SUB     X1,A            ; subtract one
        SUB     X1,A            ; subtract one
        SUB     X1,A            ; subtract one four subs is wrong but gets them to line up
        DO      A1,L_READ_THE_REST      ; clock-and-read the remaining (510) pix in the first row
 (one at a time)
        JSET    #BO_MODE,X:STATUS,RR_BO1      ; if BO mode selected, jump to the BO method
        MOVE    Y:<CLK_COL_AND_READ,R0
        JSR     <CLOCK
        JMP     <OVER_BO1       ; jump over BO mode

; *****************************************************************************
; this is BO mode
RR_BO1  MOVE    #CLOCK_ONLY,R0     ; clock to pixel
        JSR     <CLOCK
        NOP
        MOVE    #PULSE_RESET,R0         ; strobe the reset on this row
        JSR     <CLOCK
        NOP

        ; put a settling time delay here (10 us)
        DO      #200,SET_DLY3               ; Each step of this loop is 1 us
        NOP
SET_DLY3



        JCLR    #CDS_MODE,X:STATUS,BO_RD_END_3
        DO      Y:<NFS,BO_RD_END_3                          ; Fowler sampling

        ; first BO read(s)
        MOVE    #A2D_ONLY,R0
```

```
        JSR     <CLOCK

        NOP
BO_RD_END_3

; Implement up-the-ramp readout
        DO      Y:<IUTR,BO_L_UP_THE_RAMP_3


        ; delay loop goes here for exptime in BO mode
        ; this is the number to delay by -  X:<EXP_TIM
        DO      Y0,EXP_DLY3                ; Each step of this loop is 1 us
        NOP
EXP_DLY3

; Sample the signal level NFS times
        DO      Y:<NFS,BO_RD_EXP_3                            ; Fowler sampling

        ; second BO read
        MOVE    #A2D_ONLY,R0
        JSR     <CLOCK

        NOP
BO_RD_EXP_3
        NOP
BO_L_UP_THE_RAMP_3
        NOP
; *****************************************************************************
OVER_BO1
        NOP
L_READ_THE_REST


; do this at the end to clock into the "next" 64 column block at the end of each full row.
;       MOVE    #CLOCK_ONLY,R0
;       JSR     <CLOCK

;        MOVE    #READ_OFF,R0            ; Turn Read OFF
;        JSR     <CLOCK


        CLR     A
        MOVE    Y:<NROWS,A1
        DEC     A                       ; NROWS - 1
        DO      A1,FRAME                ; clock-and-read the remaining (1023) rows per half

L_READ_NORMALLY
        JSET    #RST_MODE,X:STATUS,RRR_RD1
        MOVE    #CLOCK_ROW,R0           ; if not row reset
        JSR     <CLOCK
        JMP     <RRR_RD2
        ; **** this should only happen on the first readout. Need to fix this for row-by-row ****
***
RRR_RD1 MOVE    #CLOCK_ROW_RESET,R0     ; if row-by-row reset
        JSR     <CLOCK
RRR_RD2 NOP

        MOVE    #READ_ON,R0            ; Turn Read ON
        JSR     <CLOCK

        ; for 1RG, need 2 HCLK pulses before the first real pixel in each row
        ; This affects the relative start col of each row the same in all reads
        ; It looks like 2 HCLKs is correct, if smaller the last two pixels
        ; per row are identical.
        DO      #2,SKIP_1       ; comment
        MOVE    #CLOCK_ONLY,R0  ; comment
        JSR     <CLOCK          ; comment
        NOP                     ;
SKIP_1


; use this block for COL skipping  (this is for burst-mode subarray)
```

```
        MOVE    Y:<STARTCOL,A
        MOVE    X:ONE,X0
        CMP     X0,A                            ;   if # of cols to skip is zero, then don't
        JEQ     <NOCOLSKIP
        DEC     A
        DO      A,COLSKIP
        MOVE    #CLOCK_ONLY_FAST,R0
        JSR     <CLOCK
        NOP
COLSKIP
        NOP
NOCOLSKIP

        MOVE    Y:<TEMP,X0              ; total cols/2
        DO      X0,L_READ_LAST_TWO_PIXELS ; clock-and-read (512) pix per row, 1 at a time
        JSET    #BO_MODE,X:STATUS,RR_BO2      ; if BO mode selected, jump to the BO method
        MOVE    Y:<CLK_COL_AND_READ,R0
        JSR     <CLOCK
        JMP     <OVER_BO2       ; jump over BO mode

; ******************************************************************************
; this is BO mode
RR_BO2  MOVE    #CLOCK_ONLY,R0      ; clock to pixel
        JSR     <CLOCK
        NOP
        MOVE    #PULSE_RESET,R0         ; strobe the reset on this row
        JSR     <CLOCK
        NOP


        ; put a settling time delay here (10 us)
        DO      #200,SET_DLY4               ; Each step of this loop is 1 us
        NOP
SET_DLY4


        JCLR    #CDS_MODE,X:STATUS,BO_RD_END_4
        DO      Y:<NFS,BO_RD_END_4                           ; Fowler sampling

        ; first BO read(s)
        MOVE    #A2D_ONLY,R0
        JSR     <CLOCK

        NOP
BO_RD_END_4

; Implement up-the-ramp readout
        DO      Y:<IUTR,BO_L_UP_THE_RAMP_4

        ; delay loop goes here for exptime in BO mode
        ; this is the number to delay by -  X:<EXP_TIM
        DO      Y0,EXP_DLY4                 ; Each step of this loop is 1 us
        NOP
EXP_DLY4

; Sample the signal level NFS times
        DO      Y:<NFS,BO_RD_EXP_4                           ; Fowler sampling

        ; second BO read
        MOVE    #A2D_ONLY,R0
        JSR     <CLOCK

        NOP
BO_RD_EXP_4
        NOP
BO_L_UP_THE_RAMP_4
        NOP
; ******************************************************************************
OVER_BO2
        NOP
L_READ_LAST_TWO_PIXELS
```

```
        MOVE    #READ_OFF,R0            ; Turn Read OFF
        JSR     <CLOCK          ; clock it

        NOP
FRAME
        MOVE    #CLOCK_ROW,R0           ; extra one at end of frame
        JSR     <CLOCK          ; clock it see page 33 of 1RG manual

        MOVE    #VSOURCE_OFF,R0         ; Turn VSOURCE OFF
        JSR     <CLOCK

        ; 2 pix, do this twice
;       DO      #2,LAST_TWO_PIX ; load two
;       MOVE    Y:<LAST_TWO_PIXELS,R0 ; put in R0
;       JSR     <CLOCK          ; clock it
;       NOP
;LAST_TWO_PIX

        ; this seems to work better then the above block for the 1R to get the last two pix.
        MOVE    Y:<CLK_COL_AND_READ,R0  ; comment
        JSR     <CLOCK                  ; comment
        MOVE    Y:<CLK_COL_AND_READ,R0 ; comment
        JSR     <CLOCK          ; comment

        JSR     <PAL_DLY                ; Wait for serial data transmission
        BCLR    #WW,X:PBD               ; Clear WW to 0 for non-image data
        RTS


;******************************************************************************
; Row by Row reset (in Rockwell-speak)
RESET_ARRAY
        MOVE    #VSOURCE_OFF,R0            ; Turn VSOURCE OFF
        JSR     <CLOCK
        MOVE    #FRAME_INIT,R0
        JSR     <CLOCK
        DO      #1024,L_RESET_ARRAY
        MOVE    #CLOCK_ROW_RESET,R0
        JSR     <CLOCK
        NOP
L_RESET_ARRAY
        RTS


;******************************************************************************
; Continuously reset and read array, checking for commands every four rows
CONT_RST
        MOVE    #VSOURCE_OFF,R0            ; Turn VSOURCE OFF  (default for autoflush)
        JSR     <CLOCK
        MOVE    #FRAME_INIT,R0
        JSR     <CLOCK
        DO      #256,L_RESET            ; total cols/4
        JSR     <CLK_FPA
        JSR     <GET_RCV                ; Look for a new command every 4 rows
        JCC     <NO_COM                 ; If none, then stay here
        ENDDO
        JMP     <CHK_SSI
NO_COM  NOP
L_RESET
        JMP     <CONT_RST

;******************************************************************************
; A simple clocking subroutine just to save some memory since its called twice
CLK_FPA
        DO      #512,L_CLK_FPA
        MOVE    #CLOCK_ROW_RESET,R0
        JSR     <CLOCK
        NOP
L_CLK_FPA
        RTS
```

```
;******************************************************************************
;****************************    SUBROUTINE    ***********************
;  Core subroutine for clocking out array charge
CLOCK   MOVE    Y:(R0)+,X0              ; # of waveform entries
        MOVE    Y:(R0)+,A               ; Start the pipeline
        DO      X0,CLK1                 ; Repeat X0 times
        MOVE    A,X:(R6) Y:(R0)+,A      ; Send out the waveform
CLK1
        MOVE    A,X:(R6)                ; Flush out the pipeline
        RTS                             ; Return from subroutine
;**************************  END  OF  SUBROUTINE  ********************
;******************************************************************************



;******************************************************************************
;****  END OF CLOCKING CODE  *************
;******************************************************************************

; Check for program overflow
        IF      @CVS(N,*)>=$2F0
        WARN    'Application P: program is too large!'  ; Make sure program
        ENDIF                                           ;  will not overflow




;******************************************************************************
;  ****************  PROGRAM CODE IN SRAM PROGRAM SPACE    *******************
;******************************************************************************
; Put all the following code in SRAM, starting at P:$2F0.
        IF      @SCP("DOWNLOAD","HOST")
        ORG     P:$2F0,P:$2F0   ; Download address
        ELSE
        ORG     P:$2F0,P:APL_NUM*N_W_APL+APL_LEN ; ROM address
        ENDIF

;******************************************************************************
START_EXPOSURE

        MOVE    X:<ONE,X0                       ; If up-the-ramp is selected
        MOVE    Y:<NUTR,A                       ;   then pass-through and
        CMP     X0,A                            ;   correlated double sample
        JEQ     <ADR_SEL                        ;   are both required
        SUB     X0,A
        MOVE    A,Y:<IUTR                       ; Subtract 1 from NUTR because
        BSET    #PTR_MODE,X:STATUS              ;   one frame will be done by
        BSET    #CDS_MODE,X:STATUS              ;   CDS = 1

ADR_SEL MOVE    #CLOCK_COL_AND_READ,X0          ; Coadder values
        MOVE    #READ_LAST_TWO_PIXELS,X1
        MOVE    #CLOCK_COL_AND_READ_NO_XFER,Y0
        JCLR    #PTR_MODE,X:STATUS,INIT_CA
        MOVE    #CLOCK_COL_AND_READ_PT,X0       ; Pass through values
        MOVE    #SXMIT_LAST_TWO_PIXELS,X1
        MOVE    #CLOCK_COL_AND_READ_PT_NO_XFER,Y0

; Initialize the coadder board - clear the FIFOs and zero the SRAM
INIT_CA MOVE    X0,Y:<CLK_COL_AND_READ
        MOVE    X1,Y:<LAST_TWO_PIXELS
        MOVE    Y0,Y:<CLK_COL_AND_READ_NO_XFER
        MOVE    #$020F02,X0             ; Header for all coadder boards
        JSR     <XMT_CA
        MOVE    #'ICA',X0               ; Initialize
        JSR     <XMT_CA

; Start the coadder board accumulating A/D image data or passing it through
```

```
        MOVE    #$020F06,X0             ; Header for all coadder boards
        JSR     <XMT_CA

        ; select coadd or pass-through mode here
        MOVE    #'RDC',X0               ; Read frames in coadd mode
        JCLR    #PTR_MODE,X:STATUS,S_COADD
        MOVE    #'RPT',X0               ; Read frames in pass-through mode
S_COADD JSR     <XMT_CA

        MOVE    Y:<NCOADDS,X0           ; Send the number of coadded frames
        JSR     <XMT_CA
        MOVE    Y:<NUTR,X0              ; Send the number of up-the-ramp frames
        JSR     <XMT_CA
        CLR     A
        JCLR    #CDS_MODE,X:STATUS,CDS_NOT
        BSET    #0,A
CDS_NOT MOVE    A,X0                    ; Send CDS value, 0 or 1
        JSR     <XMT_CA
        MOVE    Y:<NFS,X0               ; Send the number of Fowler samples
        JSR     <XMT_CA
        MOVE    Y:<NCOLS,X0             ; Send the number of Columns
        JSR     <XMT_CA
        MOVE    Y:<NROWS,X0             ; Send the number of Rows
        JSR     <XMT_CA
        MOVE    Y:<NREADS,X0            ; Send the number of reads
        JSR     <XMT_CA


; Transmit 'DON' back to host computer to indicate exposure start
        MOVE    #$020102,X0
        JSR     <XMT_FO
        MOVE    #'IIA',X0
        JSR     <XMT_FO

; In coadder mode put the PCI board and fiber optics in 16-bit mode later
        JCLR    #PTR_MODE,X:STATUS,COADD

; Start the big coadd loop
COADD   DO      Y:<NCOADDS,L_COADD

; Reset the array once for each coadd
        JSR     <RESET_ARRAY

; Sample the reset level NFS times if in CDS mode
        JCLR    #CDS_MODE,X:STATUS,RD_END
        DO      Y:<NFS,RD_END                           ; Fowler sampling
        JSR     <RD_ARRAY                               ; Read the array
        NOP
RD_END

; Implement up-the-ramp readout
        DO      Y:<IUTR,L_UP_THE_RAMP

; Expose the array
        MOVE    #ST_COAD,R7             ; Jump to ST_COAD after exposure
        JMP     <EXPOSE                 ; Start exposure countdown

; Sample the signal level NFS times
ST_COAD DO      Y:<NFS,RD_EXP                           ; Fowler sampling
        JSR     <RD_ARRAY                               ; Read the array
        NOP
RD_EXP
        NOP
L_UP_THE_RAMP                           ; Up the ramp loop


        NOP
L_COADD                                 ; Main coadd loop

;       ;; This is a total hack workaround to get the number of pixels to
;       ;; be right in the image. For some reason getting the reads to
;       ;; line up ends up with too few pixels. This should be removed
```

```
;       ;; when we figure out what is going on.
;       MOVE    Y:<NREADS,X0              ; number of reads
;       DO      X0,EXTRA_READS ; Read a few times to finish up the image
;        MOVE    Y:<CLK_COL_AND_READ,R0 ; comment
;        JSR     <CLOCK                 ; comment
;        MOVE    Y:<CLK_COL_AND_READ,R0 ; comment
;        JSR     <CLOCK                 ; comment
;       NOP
;EXTRA_READS
;       NOP



; In coadder mode jump to the transmit routine in SRAM
        JCLR    #PTR_MODE,X:STATUS,TRANSMIT_COADDED_IMAGE

; In PT mode, finish up and return to START
        JSR     <PAL_DLY                ; Wait for the last serial data
        BCLR    #WW,X:PBD               ; Clear WW to 0 for 32-bit commands
        JMP     <START                  ; Wait for a new host computer command

; Transmit the coadded image from the coadder board to the host computer
TRANSMIT_COADDED_IMAGE
        MOVE    #$020F02,X0             ; Header for all coadder boards
        JSR     <XMT_CA
        MOVE    #'TCA',X0               ; Transmit the coadded frame
        JSR     <XMT_CA
        JSR     <PAL_DLY                ; Wait for coadder to start

; Alert the PCI interface board that images are coming soon
        MOVEP   #$020002,Y:WRFO         ; Send header word to the FO transmitter
        REP     #20                     ; Delay a bit for the transmission
        NOP
        MOVEP   #'RDA',Y:WRFO
        REP     #20                     ; Delay a bit for the transmission
        NOP
        BSET    #WW,X:PBD               ; Set WW = 1 for 16-bit image data

XMT_DLY EQU     $4A0000                 ; Delay for serial transmitter

; This is for a single 512x512 quad assuming all 8 outputs are on that quad (aka Aladdin) *******
*****
; Transmit the coadded 512x512 pixel frame to the host computer, 32-bits/pixel
        MOVE    #>8,A                   ; Not-CDS value = (512 x 512) / 8       (loop 8 times)
        JCLR    #CDS_MODE,X:STATUS,XMIT
        MOVE    #>16,A                  ; CDS value = 2 x (512 x 512) / 8       (loop 16 times)
XMIT    DO      A,L_XTIM1               ;   32-bits per pixel
        MOVE    #$1000,X0               ; For two coadder boards transmit       (loop 4096 times)
        DO      X0,L_TXIM2              ;   eight pixels per loop
        MOVE    #$00F060,A              ; Write SXMIT out
        MOVE    A,X:(R6)                ;     send 4 16-bit pix, board 1
        MOVE    #XMT_DLY,X0             ; Delay for pixel transmission
        MOVE    X0,X:(R6)
        MOVE    #$00F060,A
        MOVE    A,X:(R6)                ;     send 4 16-bit pix, board 1
        MOVE    #XMT_DLY,X0
        MOVE    X0,X:(R6)
        MOVE    #$00F0E4,A
        MOVE    A,X:(R6)                ;     send 4 16-bit pix, board 2
        MOVE    #XMT_DLY,X0
        MOVE    X0,X:(R6)
        MOVE    #$00F0E4,A
        MOVE    A,X:(R6)                ;     send 4 16-bit pix, board 2
        MOVE    #XMT_DLY,X0
        MOVE    X0,X:(R6)
L_TXIM2 NOP
L_XTIM1
        JSR     <PAL_DLY                ; Wait for the last serial data
        BCLR    #WW,X:PBD               ; Clear WW to 0 for 32-bit commands
        JMP     <START                  ; Wait for a new host computer command
```

```
;******************************************************************************
;  Update the DACs
SET_DAC DO      Y:(R0)+,SET_L0          ; Repeat X0 times
        MOVEP   Y:(R0)+,X:SSITX         ; Send out the waveform
        JSR     <PAL_DLY                ; Wait for SSI and PAL to be empty
        NOP                             ; Do loop restriction
SET_L0
        RTS                             ; Return from subroutine

;******************************************************************************
; Set the exposure time
SET_EXT MOVE    X:(R4)+,X0      ; Get third word of command = exposure time
        MOVE    X0,X:<EXP_TIM   ; Write to magic address
        MOVE    X0,X:<TGT_TIM
        JMP     <FINISH         ; Send out 'DON' reply

;******************************************************************************
; Set the pixel clock time (DLY1 in waveforms.asm)
SET_PIX MOVE    X:(R4)+,A1      ; hold the new delay in a temp location
        REP     #12             ; commanded delay is just the left 12 bits of a 24-bit number
        LSL     A1
        MOVE    A1,Y:<TEMP      ; (its the third word of command)

        MOVE #WAVE_START,R0     ; Loop over the entire wave table
        JSR     <NEW_DLY

; Set the new delay time
        MOVE    Y:<TEMP,X0      ; move the new delay time from the temp location
        MOVE    X0,Y:PIX_TIM    ; Write to magic address
        JMP     <FINISH         ; Send out 'DON' reply


; A subroutine to subtract the old delay, and add the new delay
NEW_DLY MOVE    Y:(R0)+,A               ; # of waveform entries
        INC     A                       ; plus 1
        MOVE    A,X0
        DO      X0,L1                   ; Repeat X0 times
        MOVE    Y:(R0),A                ; move wave into A
        MOVE    Y:PIX_TIM,B             ; move old delay into B

        REP     #16                     ; Compare the delay in the waveform to the
        LSR     A                       ; stored old delay. If not equal, then just
        REP     #16                     ; skip this wave and go to the next. This is for
        LSR     B                       ; the video delays and such, which have a different
        CMP     B,A                     ; timing
        MOVE    Y:(R0),A                ; move clean wave back into A
        MOVE    Y:PIX_TIM,X1            ; move old delay back into X1
        JNE     <NOSUB

        SUB     X1,A                    ; subtract old delay
        MOVE    Y:<TEMP,X1              ; move new delay into X1
        ADD     X1,A                    ; add new delay
NOSUB   MOVE    A,Y:(R0)+               ; move result into wave and point to next wave
        NOP
L1
        RTS                             ; Return from subroutine

;******************************************************************************
; Abort exposure - not implemented
ABORT_EXP
        JMP     <FINISH         ; Issue a 'DON' reply

;******************************************************************************
; Abort readout - not implemented
ABORT_READOUT
        JMP     <START          ; Do not issue a 'DON' reply

;******************************************************************************
; Delay for serial writes to the PALs and DACs by 8 microsec
PAL_DLY DO      #300,PAL_DLY_END        ; Wait 8 usec for serial data transmission
        NOP
PAL_DLY_END
```

```
        RTS

;******************************************************************************
; 1 us delay - this is a utility for BO mode mainly
US_DLY  DO      #37,US_DLY_END
        NOP
US_DLY_END
        RTS

;******************************************************************************
; Transmit a word to the coadder board over the SSI link
XMT_CA  JCLR    #SSI_TDE,X:SSISR,*      ; Continue if SSI XMT register is empty
        MOVEP   X0,X:SSITX              ; Write to SSI buffer
        RTS

;******************************************************************************
; Wait for a 'DON' reply from the coadder board
WT_DON  JCLR    #SSI_RDF,X:SSISR,*
        MOVEP   X:SSIRX,A
        MOVE    Y:<CA_HDR,X0
        CMP     X0,A
        JNE     <WT_DON
        JCLR    #SSI_RDF,X:SSISR,*
        MOVEP   X:SSIRX,A
        MOVE    X:<DON,X0
        CMP     X0,A
        JNE     <WT_DON
        RTS

;******************************************************************************
; Set a particular bias or video offset value
;     SBN  #BOARD  #DAC  "type of board" voltage
;                               #BOARD is from 0 to 15
;                               #DAC_number is from 0 to 47
;                               'type of board' is 'CLK' or 'VID'
;                               #voltage is from 0 to 4095
SET_BIAS_NUMBER                 ; Set bias number
        JSR     <SER_ANA        ; Set SSI to analog board communication
        MOVE    X:(R4)+,Y0      ; First argument is board number, 0 to 15
        MOVE    Y0,A
        REP     #20
        LSL     A
        MOVE    A,X0
        MOVE    X:(R4)+,Y1      ; Second argument is DAC number
        MOVE    Y1,Y:<TEMP      ; also store temporarily
        MOVE    Y1,A
        REP     #14
        LSL     A
        OR      X0,A
        MOVE    X:(R4)+,B       ; Third argument is 'VID' or 'CLK' string
        MOVE    #'VID',X0
        CMP     X0,B
        JNE     <CLK_DRV
        BSET    #19,A1          ; Set bits to mean video processor DAC
        BSET    #18,A1
        JMP     <VID_BRD
CLK_DRV MOVE    #'CLK',X0
        CMP     X0,B
        JNE     <ERR_SBN
VID_BRD MOVE    A,X0
        MOVE    X:(R4)+,A       ; Fourth argument is voltage value, 0 to $fff
        MOVE    #$000FFF,X1     ; Mask off just 12 bits to be sure
        AND     X1,A
        OR      X0,A

; Update DAC table in memory with new value
        MOVE    #'CLK',X0
        CMP     X0,B
        JNE     VB

CB      MOVE    X:<TWO,X0       ; check to be sure board # is ge 2, otherwise exit with err
        MOVE    Y0,B1
```

```
        CMP     X0,B
        JLT     <ERROR

        MOVE    Y0,B1
        ASR     B                ; divide board# by 2
        MOVE    X:<ONE,X0
        SUB     X0,B             ; minus one - now CLK2 is 0 and CLK4 is 1

        MOVE    B1,X0
        MOVE    #$30,B1
        MOVE    B1,Y0
        MOVE    #0,B
        MPY     X0,Y0,B
        ASR     B
        MOVE    B0,X0
        MOVE    #0,B
        MOVE    Y:<TEMP,Y1
        ADD     X0,B             ; add mem offset for given board
        ADD     Y1,B             ; add DAC #
        MOVE    #DACS,X1
        ADD     X1,B             ; add first CLK board DAC value, which is stored in
        MOVE    X:<ONE,X1        ;   DACS+1
        ADD     X1,B
        MOVE    B,R2             ; move mem pointer into R2
        JMP     <WRDAC

VB      MOVE    #OFFSETS,X1      ; first VID board DAC value is stored here
        MOVE    Y0,B             ; Jump ahead to the selected offset
        LSL     B                ; (board# * 4) for VID
        LSL     B
        ADD     X1,B             ;   + OFFSETS
        ADD     Y1,B             ;   + DAC number for this board (0-3)
        MOVE    B,R2
        NOP
WRDAC   MOVE    A,Y:(R2)         ; Write the number to the DAC table in memory

; Write the number to the DAC itself
        MOVEP   A,X:SSITX        ; Write the number to the DAC itself
        JSR     <PAL_DLY         ; Wait for the number to be sent
        JSR     <SER_CA          ; Return SSI to coadder board communication
        JMP     <FINISH

; If invalid board type, then return wth error
ERR_SBN MOVE    X:(R4)+,A        ; Read and discard the fourth argument
        JSR     <SER_CA          ; Return SSI to coadder board communication
        JMP     <ERROR



;******************************************************************************
; Get a particular bias or video offset value
;    GBN  #BOARD  #DAC  "type of board" voltage
;                                #BOARD is from 0 to 15
;                                #DAC_number is from 0 to 47
;                                'type of board' is 'CLK' or 'VID'
GET_BIAS_NUMBER                  ; Get bias number
        MOVE    X:(R4)+,Y0       ; First argument is board number, 0 to 15
        MOVE    Y0,A
        REP     #20
        LSL     A
        MOVE    A,X0
        MOVE    X:(R4)+,Y1       ; Second argument is DAC number
        MOVE    Y1,Y:<TEMP       ; also store temporarily
        MOVE    Y1,A
        REP     #14
        LSL     A
        OR      X0,A
        MOVE    X:(R4)+,B        ; Third argument is 'VID' or 'CLK' string

; get value from DAC table in memory
        MOVE    #'CLK',X0
        CMP     X0,B
```

```
        JNE     GVB

        MOVE    X:<TWO,X0       ; check to be sure board # is ge 2, otherwise exit with err
        MOVE    Y0,B1
        CMP     X0,B
        JLT     <ERROR

GCB     MOVE    Y0,B1
        ASR     B               ; divide board# by 2
        MOVE    X:<ONE,X0
        SUB     X0,B            ; minus one - now CLK2 is 0 and CLK4 is 1

        MOVE    B1,X0
        MOVE    #$30,B1
        MOVE    B1,Y0
        MOVE    #0,B
        MPY     X0,Y0,B
        ASR     B
        MOVE    B0,X0
        MOVE    #0,B
        MOVE    Y:<TEMP,Y1
        ADD     X0,B            ; add mem offset for given board
        ADD     Y1,B            ; add DAC #
        MOVE    #DACS,X1
        ADD     X1,B            ; add first CLK board DAC value, which is stored in
        MOVE    X:<ONE,X1       ;   DACS+1
        ADD     X1,B
        MOVE    B,R2            ; move mem pointer into R2
        JMP     <RDDAC

GVB     MOVE    #OFFSETS,X1     ; first VID board DAC value is stored here
        MOVE    Y0,B            ; Jump ahead to the selected offset
        LSL     B               ; (board# * 4) for VID
        LSL     B
        ADD     X1,B            ;   + OFFSETS
        ADD     Y1,B            ;   + DAC number for this board (0-3)
        MOVE    B,R2
        NOP
RDDAC   MOVE    Y:(R2),B        ; retrieve value from DAC table
        MOVE    #$000FFF,X1     ; Mask off just the 12 bits containing the $000-$FFF value
        AND     X1,B
        MOVE    B,X0            ; X0 now holds the requested value
                                ; Send a reply of value in X0 - header and reply - to host
        MOVE    X:<HDR,A        ; Get header of incoming command
        MOVE    X:<SMASK,Y0     ; This was the source byte, and is to
        AND     Y0,A  X:<TWO,Y0 ;    become the destination byte
        REP     #8              ; Shift right one byte, add it to the
        LSR     A  Y0,X:<NWORDS ;     header, and put 2 as the number
        ADD     Y0,A  X:<SBRD,Y0 ;     of words in the string
        ADD     Y0,A            ; Add source board's header, set X1 for above
        MOVE    A,X:(R3)+       ; Put header on the transmitter stack
        MOVE    X0,X:(R3)+      ; Put value of XO on the transmitter stack
        JMP     <PRC_RCV        ; Go transmit these words

;*******************************************************************************
; This is where the pixel clock is selected for the diagnostic jack
;     on the clock driver board
;
; Specify the MUX value to be output on the clock driver board
; Command syntax is  SMX  #clock_driver_board #MUX1 #MUX2
;                               #clock_driver_board from 0 to 15
;                               #MUX1, #MUX2 from 0 to 23
SET_MUX JSR     <SER_ANA        ; Set SSI to analog board communication
        MOVE    X:(R4)+,A       ; Clock driver board number
        REP     #20
        LSL     A
        MOVE    #$003000,X0
        OR      X0,A
        MOVE    A,X1            ; Move here for storage

; Get the first MUX number
        MOVE    X:(R4)+,A       ; Get the first MUX number
```

```
        JLT     ERR_SM1
        MOVE    #>24,X0         ; Check for argument less than 32
        CMP     X0,A
        JGE     ERR_SM1
        MOVE    A,B
        MOVE    #>7,X0
        AND     X0,B
        MOVE    #>$18,X0
        AND     X0,A
        JNE     <SMX_1          ; Test for 0 <= MUX number <= 7
        BSET    #3,B1
        JMP     <SMX_A
SMX_1   MOVE    #>$08,X0
        CMP     X0,A            ; Test for 8 <= MUX number <= 15
        JNE     <SMX_2
        BSET    #4,B1
        JMP     <SMX_A
SMX_2   MOVE    #>$10,X0
        CMP     X0,A            ; Test for 16 <= MUX number <= 23
        JNE     <ERR_SM1
        BSET    #5,B1
SMX_A   OR      X1,B1           ; Add prefix to MUX numbers
        MOVE    B1,Y1

; Add on the second MUX number
        MOVE    X:(R4)+,A       ; Get the next MUX number
        JLT     ERR_SM2
        MOVE    #>24,X0         ; Check for argument less than 32
        CMP     X0,A
        JGE     ERR_SM2
        REP     #6
        LSL     A
        MOVE    A,B
        MOVE    #$1C0,X0
        AND     X0,B
        MOVE    #>$600,X0
        AND     X0,A
        JNE     <SMX_3          ; Test for 0 <= MUX number <= 7
        BSET    #9,B1
        JMP     <SMX_B
SMX_3   MOVE    #>$200,X0
        CMP     X0,A            ; Test for 8 <= MUX number <= 15
        JNE     <SMX_4
        BSET    #10,B1
        JMP     <SMX_B
SMX_4   MOVE    #>$400,X0
        CMP     X0,A            ; Test for 16 <= MUX number <= 23
        JNE     <ERR_SM2
        BSET    #11,B1
SMX_B   ADD     Y1,B            ; Add prefix to MUX numbers

        MOVEP   B1,X:SSITX
        JSR     <PAL_DLY        ; Delay for all this to happen
        JSR     <SER_CA         ; Return SSI to coadder board communication
        JMP     <FINISH
ERR_SM1 MOVE    X:(R4)+,A
ERR_SM2 JSR     <SER_CA         ; Return SSI to coadder board communication
        JMP     <ERROR

;******************************************************************************
; Test coadder memory
TEST_CA_MEMORY
        MOVE    X:SSIRX,A               ; Dummy read to clear SSI receiver
        MOVE    #$020F02,X0             ; Header for all coadder boards
        JSR     <XMT_CA                 ; Transmit command to coadder boards
        MOVE    #'TCM',X0               ; Test Coadder Memory
        JSR     <XMT_CA

;******************************************************************************
; Power off
PWR_OFF BCLR    #CDAC,X:<LATCH          ; Clear all DACs
        BCLR    #ENCK,X:<LATCH          ; Disable DAC output switches
```

```
        MOVEP   X:LATCH,Y:WRLATCH
        BSET    #LVEN,X:PBD             ; LVEN = HVEN = 1 => Power reset
        BSET    #PWRST,X:PBD            ; PWRST = 1 => Power reset
        BSET    #HVEN,X:PBD
        MOVE    #TST_RCV,X0
        MOVE    X0,X:<IDL_ADR
        JMP     <FINISH

;*******************************************************************************
; Power on sequence. This gives nice RC turn-on times of 0.5 milliseconds
;       Open output switches (ENCK)
;       Power ON, delay
;       Zero DACs and switches, delay
;       Close output switch (ENCK)
;       Load correct voltages into DACs
; Power on
PWR_ON  JSR     <SER_ANA                ; Enable SSI for analog boards
        BSET    #CDAC,X:<LATCH          ; Disable clearing of all DACs
        BCLR    #DUALCLK,X:<LATCH       ; Each half of clk drv is independent
        BCLR    #ENCK,X:<LATCH          ; Open DAC output switches
        MOVEP   X:LATCH,Y:WRLATCH
        BSET    #LVEN,X:PBD             ; LVEN = HVEN = 1 => Power reset
        BSET    #PWRST,X:PBD            ; PWRST = 1 => Power reset
        BSET    #HVEN,X:PBD

; Now ramp up the low voltages (+/- 6.5V, 16.5V)
        BCLR    #LVEN,X:PBD             ; LVEN = Low => Turn on +/- 6.5V,
        BCLR    #PWRST,X:PBD            ;    +/- 16.5V
        MOVE    X:<THREE,A              ; Delay three milliseconds to settle
        JSR     <CON_DELAY

; First, zero all clock driver switches and DACs
        MOVE    #$002000,X0     ; bottom of clock board
        MOVE    X0,X:(R6)
        MOVE    #$003000,X0     ; top of clock board
        MOVE    X0,X:(R6)
        MOVE    #$004000,X0     ; bottom of bias board
        MOVE    X0,X:(R6)
        MOVE    #$005000,X0     ; top of bias board
        MOVE    X0,X:(R6)
        MOVE    #ZERO_BIASES,R0
        JSR     <SET_DAC
        MOVE    X:<THREE,A
        JSR     <CON_DELAY

; Close output switch before writing correct voltages to DACs to the voltages
;   turn on with the RC time constant of the RC filters on the DAC outputs
        BSET    #ENCK,X:<LATCH          ; Enable DAC output switches
        MOVEP   X:LATCH,Y:WRLATCH

; Turn on the DACS
        MOVE    #DACS,R0                ; Get starting address
        JSR     <SET_DAC
        MOVE    X:<THREE,A              ; Delay three milliseconds to settle
        JSR     <CON_DELAY

; Enable continuous reset mode
        MOVE    #CONT_RST,X0            ; Put array in continuous reset mode
        MOVE    X0,X:<IDL_ADR

; Enable the SSI for coadder board communication
        JSR     <SER_CA                 ; Enable SSI for the coadder board
        JMP     <FINISH

;*******************************************************************************
; Move Coadder code from timing board ROM to coadder DSP memory over the SCI
LOAD_COADDER
        JSR     <CA_RST

; Configure SCI port for a baud rate of 50 MHz / 64 / 2 = 390 kbaud
        MOVEP   #$0007,X:PCC    ; Enable SCI pins RXD, TXD and SCLK
        MOVEP   #$0302,X:SCR    ; Enable RXD and TXD
```

```
        MOVEP   #$1001,X:SCCR   ; SCI clock = 16 x baud rate
        JSR     <PAL_DLY

; Send the boot code to the coadder
        MOVE    #P_COADD,R0     ; Starting P: address in EEPROM
        DO      #3,L_COADDER
        MOVE    P:(R0)+,A2      ; Get one byte from EEPROM
        REP     #8
        ASR     A               ; Shift right 8 bits
        NOP
L_COADDER
        MOVE    X:<TWO,X0
        ADD     X0,A            ; A1 = Number of words + 2
        MOVE    #P_COADD,R0     ; Starting P: address in EEPROM
        DO      A1,P_MOVE       ; Loop over A1 words
        DO      #3,P_LOOP       ; Write 3 bytes per word
        JCLR    #0,X:SSR,*      ; Wait for transmitter to be empty
        MOVEP   P:(R0)+,X:WRSCI ; Transfer one byte from EEPROM to SCI port
P_LOOP  NOP
P_MOVE
        DO      #2400,*+3       ; Wait for the last word to transmit
        NOP
        JSR     <SER_CA         ; Enable SSI for the coadder board
        JMP     <FINISH

;******************************************************************************
; Move Coadder code from timing board SRAM to coadder DSP memory over the SCI
DOWNLOAD_COADDER
        JSR     <CA_RST

; Configure SCI port for a baud rate of 50 MHz / 64 / 2 = 390 kbaud
        MOVEP   #$0007,X:PCC    ; Enable SCI pins RXD, TXD and SCLK
        MOVEP   #$0302,X:SCR    ; Enable RXD and TXD
        MOVEP   #$1001,X:SCCR   ; SCI clock = 16 x baud rate
        JSR     <PAL_DLY

; Send the boot code to the coadder
        MOVE    #SRAM_AD,R0     ; Starting P: address in SRAM = $500
        MOVE    X:<TWO,X0
        MOVE    P:(R0),A        ; Get number of words from SRAM
        ADD     X0,A            ; A1 = Number of words + 2
        DO      A1,PROGRAM_MOVE ; Loop over number of words
        JCLR    #0,X:SSR,*      ; Wait for transmitter to be empty
        MOVEP   P:(R0),X:WRSCI0 ; Transfer one byte from SRAM to SCI port
        JCLR    #0,X:SSR,*      ; Wait for transmitter to be empty
        MOVEP   P:(R0),X:WRSCI1 ; Transfer one byte from SRAM to SCI port
        JCLR    #0,X:SSR,*      ; Wait for transmitter to be empty
        MOVEP   P:(R0)+,X:WRSCI2 ; Transfer one byte from SRAM to SCI port
PROGRAM_MOVE
        DO      #2400,*+3       ; Wait for the last word to transmit
        NOP
        JSR     <SER_CA         ; Enable SSI for the coadder board
        JMP     <FINISH

;******************************************************************************
; Reset the coadder board to force it to re-boot
CA_RST  BCLR    #RST_CA,X:<LATCH
        MOVEP   X:LATCH,Y:WRLATCH       ; Clear reset coadder bit
        DO      #600,*+3                ; Delay by RESET* time = 100 microsec
        NOP
        BSET    #RST_CA,X:<LATCH
        MOVEP   X:LATCH,Y:WRLATCH       ; Set reset coadder bit
        RTS

;******************************************************************************
; Enable serial communication to the analog boards
SER_ANA BSET    #0,X:PBD        ; Set H0 for analog boards SSI
        MOVEP   #$0000,X:PCC    ; Software reset of SSI
        BCLR    #10,X:CRB       ; Change SSI to continuous clock for analog
        MOVEP   #$0160,X:PCC    ; Re-enable the SSI
        RTS
```

```
;******************************************************************************
; Enable serial communication to the coadder board
SER_CA  MOVEP   #$0000,X:PCC     ; Software reset of SSI
        BSET    #10,X:CRB        ; Change SSI to gated clock for utility board
        MOVEP   #$01E8,X:PCC     ; Enable the SSI
        MOVEP   #$0003,X:PCDDR   ; Disable PC2 = SCLK output
        BCLR    #0,X:PBD         ; Clear H0 for utility board SSI
        RTS

;******************************************************************************
; Simple, do nothing command to exit RUN mode
STP     JMP     <FINISH

;******************************************************************************
; Short delay for settling times
CON_DELAY                                ; Alternate entry for camera on delay
        MOVE    A,X:<TGT_TIM
        CLR     A                        ; Zero out elapsed time
        MOVE    A,X:<EL_TIM
        BSET    #0,X:TCSR                ; Enable DSP timer
CNT_DWN JSET    #0,X:TCSR,CNT_DWN        ; Wait here for timer to count down
        RTS

;******************************************************************************
; Let the host computer read the controller configuration
READ_CONTROLLER_CONFIGURATION
        MOVE    Y:<CONFIG,X0             ; Just transmit the configuration
        JMP     <FINISH1

;******************************************************************************
COADDER_TDL
        MOVE    X:SSIRX,A                ; Dummy read to clear SSI receiver
        MOVE    #$020F03,X0              ; Header for all coadder boards
        JSR     <XMT_CA                  ; Transmit command to coadder boards
        MOVE    #'TDL',X0                ; Test Coadder Memory
        JSR     <XMT_CA
        MOVE    X:(R4)+,X0               ; Argument
        JSR     <XMT_CA
        JCLR    #SSI_RDF,X:SSISR,*       ; Wait for a reply from the coadder
        MOVE    X:SSIRX,A                ; Read the SSI word
        MOVE    #$040002,X0              ; Desired header from the timing board
        CMP     X0,A
        JNE     <ERROR
        JCLR    #SSI_RDF,X:SSISR,*       ; Wait for a reply from the coadder
        MOVE    X:SSIRX,X0               ; Read the SSI word
        JMP     <FINISH1

;******************************************************************************
INIT_COADDER
        MOVE    X:SSIRX,A                ; Dummy read to clear SSI receiver
        MOVE    #$020F02,X0              ; Header for all coadder boards
        JSR     <XMT_CA                  ; Transmit command to coadder boards
        MOVE    #'ICA',X0                ; Test Coadder Initialization
        JSR     <XMT_CA
        JCLR    #SSI_RDF,X:SSISR,*       ; Wait for a reply from the coadder
        MOVE    X:SSIRX,A                ; Read the SSI word
        MOVE    #$040202,X0              ; Desired header from the timing board
        CMP     X0,A
        JNE     <ERROR
        JCLR    #SSI_RDF,X:SSISR,*       ; Wait for a reply from the coadder
        MOVE    X:SSIRX,X0               ; Read the SSI word
        JMP     <FINISH1

;******************************************************************************
; Specify either global or row-by-row reset
SELECT_RESET_MODE
        MOVE    X:(R4)+,X0               ; Get the command argument
        JSET    #0,X0,SRM_SET
        BCLR    #RST_MODE,X:STATUS       ; Global reset is in effect
        JMP     <FINISH
SRM_SET BSET    #RST_MODE,X:STATUS       ; Row-by-row reset is in effect
        JMP     <FINISH
```

```
;******************************************************************************
; Specify either normal or Bright Object (BO) mode
SELECT_BO_MODE
        MOVE    X:(R4)+,X0              ; Get the command argument
        JSET    #0,X0,SBO_SET
        BCLR    #BO_MODE,X:STATUS       ; NORMAL mode is in effect
        JMP     <FINISH
SBO_SET BSET    #BO_MODE,X:STATUS       ; BO mode is in effect
        JMP     <FINISH

;******************************************************************************
; Set number of Fowler samples per frame
SET_NUMBER_OF_FOWLER_SAMPLES
        MOVE    X:(R4)+,X0
        MOVE    X0,Y:<NFS               ; Number of Fowler samples
        JMP     <FINISH

;******************************************************************************
; Specify readout either before and after exposure (CDS) or only after
SELECT_SAMPLING_MODE
        MOVE    X:(R4)+,X0              ; Get the command argument
        JSET    #0,X0,CDS_SET
        BCLR    #CDS_MODE,X:STATUS      ; Single sampling is in effect
        JMP     <FINISH
CDS_SET BSET    #CDS_MODE,X:STATUS      ; Double correlated sampling is
        JMP     <FINISH                 ;  in effect

;******************************************************************************
; Specify whether the coadder board does arithmetic (PTR = 0) or whether
;   it just passes the data right through
SELECT_PASS_THROUGH_READOUT
        MOVE    X:(R4)+,X0              ; Get the command argument
        JSET    #0,X0,PTR_SET
        BCLR    #PTR_MODE,X:STATUS      ; Coadder board arithmetic is
        JMP     <FINISH                 ;  in effect
PTR_SET BSET    #PTR_MODE,X:STATUS      ; Pass through readout mode is
        JMP     <FINISH                 ;  in effect

;******************************************************************************
; Set number of coadds per frame
SET_NUMBER_OF_COADDS
        MOVE    X:(R4)+,X0
        MOVE    X0,Y:<NCOADDS
        JMP     <FINISH

;******************************************************************************
; Set number of exposures for up-the-ramp readout mode; also, pass-through
;   mode is always implemented in up-the-ramp.
SET_UP_THE_RAMP
        MOVE    X:(R4)+,X0
        MOVE    X0,Y:<NUTR
        MOVE    X:<ONE,X0       ; load one into X0
        MOVE    X0,Y:<IUTR      ; reset this so that Fowler and CDS work after UTR
        JMP     <FINISH

; Alert the PCI interface board that images are coming soon
PCI_READ_IMAGE
        MOVE    #$020104,X0
        JSR     <XMT_FO
        MOVE    #'RDA',X0
        JSR     <XMT_FO
        MOVE    Y:<NCOLS,X0     ; load ncols into X0
        MOVE    Y:<NREADS,X1    ; load nreads into X1
        MPY     X0,X1,A
        ASR     A               ; first shift right to get rid of extra shift
        ;;                      caused by multiply.
        ASR     A               ; Now shift right twice to divide by 4
        ASR     A               ; Note that the PCI card will multiply ncol
;;                              by eight so we are really passing ncols*nreads*2
;;                              to account for the extra reads of temp and analog
        ;;                      outputs.
```

```
        MOVE    A0,X0
;       MOVE    X0,Y:<STARTCOL  ;  debugging
        JSR     <XMT_FO
        MOVE    Y:<NROWS,X0     ; load nrows
        JSR     <XMT_FO
        RTS

XMT_FO  MOVE    X0,Y:WRFO
        REP     #15
        NOP
        RTS

;******************************************************************************
; Send the device type back to the PCI/VOODOO
GET_DEVICE_TYPE
        MOVE    X:<ONE,X0       ; X0 now holds the requested value (0=H1R, 1=H1RG)
                                ; Send a reply of value in X0 - header and reply - to host
        MOVE    X:<HDR,A        ; Get header of incoming command
        MOVE    X:<SMASK,Y0     ; This was the source byte, and is to
        AND     Y0,A  X:<TWO,Y0 ;    become the destination byte
        REP     #8              ; Shift right one byte, add it to the
        LSR     A  Y0,X:<NWORDS ;     header, and put 2 as the number
        ADD     Y0,A  X:<SBRD,Y0 ;    of words in the string
        ADD     Y0,A            ; Add source board's header, set X1 for above
        MOVE    A,X:(R3)+       ; Put header on the transmitter stack
        MOVE    X0,X:(R3)+      ; Put value of XO on the transmitter stack
        JMP     <PRC_RCV        ; Go transmit these words

;******************************************************************************
; Send a serial command to an H1RG register
;   SER  #REGISTER  #DATA
;                               #REGISTER is from 1 to 13 (1-D)
;                               #DATA (up to 12 bits)
SERIAL_COMMAND

        MOVE    X:(R4)+,Y0      ; First argument is REGISTER NUMBER
        CLR     A
        MOVE    Y0,A1
        MOVE    X:(R4)+,X1      ; Second argument is number of data bits in this register
;        REP     #12
        REP     X1              ; shift address over N bits to the left
        LSL     A
        MOVE    A1,X0

        MOVE    X:<TWO,A1               ; add 4 to N data bits to get the total word length
        ASL     A
        ADD     X1,A
        MOVE    A1,X1           ; store in X1


        MOVE    X:(R4)+,Y1      ; Third argument is DATA
        MOVE    Y1,A1
        OR      X0,A1           ; OR REG # with DATA to build N-bit command word
        MOVE    A1,Y:<TEMP

        MOVE    #CSB_OFF,R0     ; Set CSB LO
        JSR     <CLOCK

        MOVE    Y:<TEMP,A1              ; X0 now contains the command word

        ; now loop over all bits and send out 1's or 0's
;       DO      #16,SENDBITS
        DO      X1,SENDBITS

        MOVE    Y:<TEMP,A1
        LSL     A
        MOVE    A1,Y:<TEMP
        MOVE    A1,B1
        MOVE    #$010000,Y0
        AND     Y0,B
        MOVE    B1,X0
        MOVE    X:<ZERO,Y0
```

```
        CMP     Y0,B
        JGT     <SET_1
        MOVE    #CLOCK_SERIAL_ZERO,R0   ; Send a 0
        JSR     <CLOCK
        JMP     NEXT_BT
SET_1   MOVE    #CLOCK_SERIAL_ONE,R0    ; Send a 1
        JSR     <CLOCK
NEXT_BT
        NOP
SENDBITS

        MOVE    #CSB_ON,R0     ; Set CSB HI - all bits are in...this latches the value
        JSR     <CLOCK

        JMP     <FINISH

;******************************************************************************
;*** Command table. There MUST be exactly 32 commands entered here. ***
;******************************************************************************
        IF      @SCP("DOWNLOAD","HOST")
        ORG     X:COM_TBL,X:COM_TBL                     ; Download address
        ELSE
        ORG     P:COM_TBL,P:APL_NUM*N_W_APL+APL_LEN+$200 ; EEPROM address
        ENDIF
        DC      'SEX',START_EXPOSURE    ; Voodoo and CCDTool start exposure
        DC      'STP',STP               ; Exit continuous RUN mode
        DC      'PON',PWR_ON            ; Turn on all camera biases and clocks
        DC      'POF',PWR_OFF           ; Turn +/- 15V power supplies off
        DC      'LCA',LOAD_COADDER      ; Coadder code comes from timing ROM
        DC      'DCA',DOWNLOAD_COADDER  ; Coadder code comes from timing SRAM
        DC      'SBN',SET_BIAS_NUMBER
        DC      'GBN',GET_BIAS_NUMBER
        DC      'SMX',SET_MUX           ; Set MUX number on clock driver board
        DC      'DON',START             ; Nothing special
        DC      'SET',SET_EXT           ; Set exposure time
        DC      'AEX',ABORT_EXP         ; Abort exposure
        DC      'ABR',ABORT_READOUT
        DC      'RCC',READ_CONTROLLER_CONFIGURATION
        DC      'TCM',TEST_CA_MEMORY    ; Test coadder memory
        DC      'CAT',COADDER_TDL       ; Test coadder data link
        DC      'ICA',INIT_COADDER

; Special NGST commands
        DC      'SRM',SELECT_RESET_MODE
        DC      'CDS',SELECT_SAMPLING_MODE
        DC      'SFS',SET_NUMBER_OF_FOWLER_SAMPLES
        DC      'SPT',SELECT_PASS_THROUGH_READOUT
        DC      'SNC',SET_NUMBER_OF_COADDS
        DC      'SUR',SET_UP_THE_RAMP
        DC      'SPC',SET_PIX           ; Set the pixel clock time
        DC      'SBO',SELECT_BO_MODE
        DC      'TYP',GET_DEVICE_TYPE
        DC      'SER',SERIAL_COMMAND

        DC      0,START,0,START,0,START,0,START
        DC      0,START

; the remaining commands (40 total) are in ../DSPlib/timboot.asm
;******************************************************************************
;*** End of Command table ***
;******************************************************************************


;******************************************************************************
;   ****************  END OF PROGRAM CODE  ******************
;******************************************************************************


        IF      @SCP("DOWNLOAD","HOST")
        ORG     Y:0,Y:0         ; Download address
        ELSE
        ORG     Y:0,P:          ; EEPROM address continues from P: above
```

```
        ENDIF

TEMP    DC      0               ; Any temporary storage
NCOLS   DC      $400            ; Number of columns
NROWS   DC      $400            ; Number of rows
NREADS  DC      0               ; Number of reads
STARTCOL        DC      1               ; Starting col of readout
STARTROW        DC      1               ; starting row of readout
NCOADDS DC      1               ; Number of frames to coadd, set in SNC
NUTR    DC      1               ; Number of up-the-ramp frames
IUTR    DC      1               ; Number of up-the-ramp frames, minus 1
NFS     DC      1               ; Number of Fowler samples
CLK_COL_AND_READ DC     0       ; Address of clock and read waveform
LAST_TWO_PIXELS DC 0            ; Address of last two pixels waveform
CLK_COL_AND_READ_NO_XFER DC 0   ; Address of special first waveform
CA_HDR  DC      $040202         ; Coadder header for replies
CONFIG  DC      CC              ; Controller configuration

; Put all the waveform and DC bias definitions in a separate file
        INCLUDE "waveforms.asm"

        ENDSEC                  ; End of section TIM_HAWAII_1RG

; Include the coadder source
        INCLUDE "coadd.asm"

;  End of program
        END
```