```
; ********************** START OF COADDER  PROGRAM  ***********************

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

; Define some useful DSP register locations
RST_ISR EQU     $0      ; Hardware reset interrupt
BUF_STR EQU     $80     ; Starting address of buffers in X:
BUF_LEN EQU     $20     ; Length of buffers
COM_BUF EQU     BUF_STR         ; Starting address of command buffer in X:
COM_TBL EQU     COM_BUF+BUF_LEN ; Starting address of command table in X:
ADR_TBL EQU     COM_TBL+BUF_LEN ; Table with SRAM addresses for pixels
APL_ADR EQU     $10             ; Start of application code, $10 for now
APL_LEN EQU     $2F0-APL_ADR    ; Length of application code area
ROM_OFF EQU     $4AAB           ; Offset address into timing board ROM
                                ;   so it gets placed at $6001 = 24k into
                                ;   32k ROM
SRAM_AD EQU     $5F0    ; Address in timing board SRAM where coadder code is
X_TABLE EQU     0       ; Starting address of X: memory table

; DSP specific addresses
BCR     EQU     $FFFE   ; Bus (=Port A) Control Register -> Wait States
PCC     EQU     $FFE1   ; Port C Control Register
PCDDR   EQU     $FFE3   ; Port C Data Direction Register
PCD     EQU     $FFE5   ; Port C Data Register
CRA     EQU     $FFEC   ; SSI Control Register A
CRB     EQU     $FFED   ; SSI Control Register B
SSISR   EQU     $FFEE   ; SSI Status Register
SSIRX   EQU     $FFEF   ; SSI Receiver
SSITX   EQU     $FFEF   ; SSI Transmitter
PBC     EQU     $FFE0   ; Port B Control Register
PBDDR   EQU     $FFE2   ; Port B Data Direction Register
PBD     EQU     $FFE4   ; Port B Data Register
PCTL    EQU     $FFFD   ; PLL control register
IPR     EQU     $FFFF   ; Interrupt Priority Register
SCR     EQU     $FFF0
SSR     EQU     $FFF1
SCCR    EQU     $FFF2
STXL    EQU     $FFF4
SRXL    EQU     $FFF4
SSI_TDE EQU     6       ; SSI Transmitter data register empty
SSI_RDF EQU     7       ; SSI Receiver data register full
RCV_ERR EQU     2       ; Place in X: memory for SSI errors

; Board addresses
RDFIFO  EQU     $C000   ; Read A/D FIFO into DSP
WRCAFIFO EQU    $C001   ; Write from DSP to coadder FIFO
RSTFIFO EQU     $C002   ; Reset both A/D and coadder FIFOs
RSTSRAM EQU     $8000   ; Start 32-bit mode pointer with D0-D23 (X: memory)

; Parallel port bit numbers, Port B
SRAMADDR16      EQU     4       ; SRAM address line 16
SRAMADDR17      EQU     5       ; SRAM address line 17
SRAMADDR18      EQU     6       ; SRAM address line 18
SRAMADDR19      EQU     7       ; SRAM address line 19
GAIN            EQU     8       ; A/D gain
EF              EQU     9       ; Bit number of incomimng FIFO empty flag
HF              EQU    10       ; Bit number of incoming FIFO half full flag

; Define a little table for X: memory entries
NCOADDS EQU     X_TABLE
NUTR    EQU     NCOADDS+1
CDS     EQU     NUTR+1
NFS     EQU     CDS+1
NCOLS   EQU     NFS+1
NROWS   EQU     NCOLS+1
NREADS  EQU     NROWS+1

; More flags, Port C
M24_32          EQU     2       ; 24- or 32-bit coaddition, Port C
CAHF            EQU     5       ; Cadder FIFO half full flag, on Port C
MEM_ERR         EQU     3       ; Set if memory testing error
```

```
;*****************************************************************************
;                                                                            *
;    Permanent address register assignments                                  *
;        R1 - Pointer to commands received pending processing                *
;        R2 - Pointer to processed commands                                  *
;        R3 - Pixel address of frame being coadded                           *
;        R4 - Pixel address of frame being transmitted                       *
;        R5 - RDFIFO address = $C000                                         *
;        R6 - WRCAFIFO address = $C001                                       *
;                                                                            *
;*****************************************************************************


;*******************************************************************************
;****  COADDER CODE IN SRAM PROGRAM SPACE  ****
;*******************************************************************************
; Specify the load address for the coadder code
        IF      @SCP("DOWNLOAD","HOST")
        ORG     P:RST_ISR,P:$5F0                ; Download address
        ELSE
        ORG     P:RST_ISR,P:RST_ISR+ROM_OFF     ; ROM address
        ENDIF

; Special address for two words for downloading over coadder SCI port
        DC      END_ADR                 ; Number of boot code words
        DC      $0                      ; Starting address of boot code

; After RESET jump to initialization code
        IF      @SCP("DOWNLOAD","HOST")
        ORG     P:RST_ISR,P:$5F0+2              ; Download address
        ELSE
        ORG     P:RST_ISR,P:RST_ISR+ROM_OFF+2   ; ROM address
        ENDIF

        JMP     <INIT           ; This is the interrupt service for RESET
        NOP
        DC      0,0,0,0

; Put the ID words for this version of the ROM code. It is placed at
;   the address of the SWI = software interrupt, which we never use.
        DC      $000000         ; Institution
                                ; Location
                                ; Instrument
        DC      $030004         ; Version 3.00, board #4 = coadder

; Space for ISRA and ISRB. P:$ISRA = $8
        DC      0,0,0,0

; This SSI interrupt service routine receives timing board data
        MOVEP   X:SSIRX,X:(R1)+  ; Put the word in the SSI receiver buffer
        NOP

; The SSI interrupts to here when there is an error.
        JSR     CLR_SSI

; Initialization of the DSP - system register, serial link, interrupts
;   Starting address = P:$10
INIT    MOVEC   #$08,OMR        ; Operating Mode Register - Single chip mode
                                ;   suitable for use with the OnCE port, with
                                ;   internal Y: memory disabled

        MOVEP   #$7F0002,X:PCTL ; MFO to 11=2, DFO to 3=0, XTLD=1, PSTP=1,
                                ;   PEN=1, COD0 to 1=0, CSRC=1 and CKOS=1
                                ;   CKOUT enabled to 3 x BUS-CLK = 75 MHz

; Port B
        MOVEP   #$0,X:PBC       ; Port B Control Register enabling port B
                                ;   pins as general purpose I/O.
        MOVEP   #$01F0,X:PBDDR  ; SRAMADDR16-19, GAIN are outputs
```

```
        MOVEP   #$0000,X:PBD    ; SRAMADDR16-19 = 0
        BSET    #GAIN,X:PBD     ; Gain = 1 for 0 to -10V A/D  (BSET, low gain)
;       BCLR    #GAIN,X:PBD     ; Gain = 0 for 0 to -5V A/D   (BCLR, high gain)

; Port C
        MOVEP   #$6002,X:CRA    ; SSI programming
        MOVEP   #$BD20,X:CRB    ; SSI programming
        MOVEP   #$01C8,X:PCC    ; Enable SSI, not SCI
        MOVEP   #$0004,X:PCDDR  ; PC2 = M24/32 is an output

; Set external bus access speeds
        MOVEP   #$2281,X:BCR    ; Wait states for external memory accesses
                                ;    Ext. X:, Ext. Y, Ext. P:, Ext. I/O
        MOVEP   #$2000,X:IPR    ; Service SSI interrups only, level 1
        BCLR    #M24_32,X:PCD   ; 24-bit (CLR) or 32-bit (SET) coaddition

; Clear the high address lines to the SRAM to zero
        BCLR    #SRAMADDR16,X:PBD       ; SRAM address line = 0
        BCLR    #SRAMADDR17,X:PBD       ; SRAM address line = 0
        BCLR    #SRAMADDR18,X:PBD       ; SRAM address line = 0
        BCLR    #SRAMADDR19,X:PBD       ; SRAM address line = 0

; Set up software tables and registers
        MOVE    #<COM_BUF,R1    ; Starting address of command buffer
        CLR     A   R1,R2
        MOVE    #31,M1          ; Input buffers are circular, modulo 32
        MOVE    M1,M2
        DO      #32,ZERO_X      ; Zero receiver buffer
        MOVE    A,X:(R1)+
ZERO_X
        MOVE    #RDFIFO,R5      ; Its in a register for faster execution
        MOVE    #WRCAFIFO,R6    ; Its in a register for faster execution

; Write the command table to X: memory
        MOVE    #TABLE,R0
        MOVE    #COM_TBL,R7
        DO      #(END_TABLE-TABLE),L_COM
        MOVE    P:(R0)+,X0
        MOVE    X0,X:(R7)+
L_COM

        ANDI    #$FC,MR         ; Enable interrupt processing

; Start processing commands here
START   MOVE    R1,X0           ; Pointer to received commands
        MOVE    R2,A            ; Pointer to processed commands
        CMP     X0,A
        JEQ     <START          ; Keep waiting if there are no new commands

; Check the header (S,D,N) for self-consistency
CHK_HDR MOVE    X:(R2),Y0
        MOVE    #$FCF0F8,A1     ; Test for S.LE.3 and D.LE.$F and N.LE.7
        AND     Y0,A
        JNE     <BAD_HDR        ; Test failed, discard candidate header
        MOVE    #$030300,A1
        AND     Y0,A            ; Test for S.NE.0 or D.NE.0
        JEQ     <BAD_HDR        ; Test failed, discard candidate header
        MOVE    #7,A1
        AND     Y0,A            ; Test for N.GE.1
        JNE     RD_COM
BAD_HDR MOVE    (R2)+           ; Increment past bad header
        JMP     <START          ; Test failed, discard candidate header

; Read all the words of the command before processing them
RD_COM  MOVE    A,Y0            ; Y0 = NWORDS
RD_WORD MOVE    R1,A            ; R1 = pointer to received commands
        MOVE    R2,X0           ; R2 = pointer to processed commands
        SUB     X0,A
        JGE     <L_C32
        MOVE    #>32,X1         ; Correct for the circular buffer
        ADD     X1,A
L_C32   CMP     Y0,A            ; Has it been incremented to RCV_BUF + NWORDS?
```

```
        JLT     <RD_WORD         ; No, keep looking for more

;  Process the receiver entry - is it in the command table ?
COMMAND MOVE    X:(R2)+,A        ; Save the header
        MOVE    A,X:<HDR
        MOVE    X:(R2)+,A        ; Get the command buffer entry
        MOVE    #<COM_TBL,R0     ; Get command table starting address
        DO      #NUM_COM,END_COM ; Loop over command table
        MOVE    X:(R0)+,X1       ; Get the command table entry
        CMP     X1,A  X:(R0),R7  ; Does receiver = table entries address?
        JNE     <NOT_COM         ; No, keep looping
        ENDDO                    ; Restore the DO loop system registers
        JMP     (R7)             ; Jump execution to the command
NOT_COM MOVE    (R0)+            ; Increment the register past the table address
END_COM

;*******************************************************************************
;  It's not in the command table - send an error message
ERROR   MOVE    #'ERR',X0        ; Send the message - there was an error
        JMP     <FINISH1         ; This protects against unknown commands

; Send a reply packet - header and reply
FINISH  MOVE    #'DON',X0        ; Send a DONE message as a reply
FINISH1 MOVE    #$040202,A       ; Only reply to the timing board
        JSR     <XMT_TIM
        MOVE    X0,A
        JSR     <XMT_TIM
        JMP     <START           ; Go get the next command

;*******************************************************************************
; Test Data Link - simply return value received after 'TDL'
TDL     MOVE    X:(R2)+,X0               ; Get data value
        MOVE    #$040002,A               ; Reply to the host
        JSR     <XMT_TIM
        MOVE    X0,A
        JSR     <XMT_TIM
        JMP     <START                   ; Go get the next command

; Get the four arguments from the timing board
GET_ARGUMENTS
        MOVE    X:(R2)+,X0
        MOVE    X0,X:<NCOADDS            ; Number of coadds
        MOVE    X:(R2)+,X0
        MOVE    X0,X:<NUTR               ; Number of up-the-ramp exposures
        MOVE    X:(R2)+,X0
        MOVE    X0,X:<CDS                ; Correlated double sampling or not
        MOVE    X:(R2)+,X0
        MOVE    X0,X:<NFS                ; Number of Fowler samples
        MOVE    X:(R2)+,X0
        MOVE    X0,X:<NCOLS              ; Number of Cols
        MOVE    X:(R2)+,X0
        MOVE    X0,X:<NROWS              ; Number of Rows
        MOVE    X:(R2)+,X0
        MOVE    X0,X:<NREADS             ; Number of Reads
        RTS

;*******************************************************************************
; This is "Initialize coadder board". Zero out 512 x 512 pixel of SRAM
ICA     MOVE    A,X:RSTFIFO              ; Reset A/D and coadder FIFOs
        MOVE    X0,X:RSTSRAM             ; Initialize the Ping-Pong data pointer
        BSET    #M24_32,X:PCD            ; Always 32-bit (SET) coaddition
        BCLR    #SRAMADDR17,X:PBD
        JSR     <INIT_FRAME              ; This is the reset frame
        BSET    #SRAMADDR17,X:PBD
        JSR     <INIT_FRAME              ; This is the expose frame
        JMP     <START

INIT_FRAME
        BCLR    #SRAMADDR16,X:PBD        ; A16 = 0
        JSR     <CLR_MEM
        BSET    #SRAMADDR16,X:PBD        ; A16 = 1
        JSR     <CLR_MEM
```

```
        RTS

; Zero out one 64k = 256 x 256 pixel page of SRAM = 64 k words
CLR_MEM MOVE    #$10000,X0              ; 64K words of SRAM memory at a time
        CLR     A
        MOVE    A1,R3
        DO      X0,ZERO_SRAM
        MOVE    A,Y:(R3)
        MOVE    A,Y:(R3)+               ; Initialize D31-D00 of coadder memory
ZERO_SRAM
        RTS

; Always store the  reset frame in SRAMADDR17 = 0
; always store the expose frame in SRAMADDR17 = 1



;********************************************************************************
; This is the coadder routine. Up-the-ramp is not supported.
RDC     MOVE    A,X:RSTFIFO             ; Reset A/D and coadder FIFOs
        MOVE    X0,X:RSTSRAM            ; Initialize the Ping-Pong data pointer
        MOVE    #$00FFFF,X1             ; Set D23-D16 = 0
        MOVEP   #$1181,X:BCR            ; Wait states for ext. memory accesses
                                        ;    Ext. X:, Ext. Y, Ext. P:, Ext. I/O
        JSR     <GET_ARGUMENTS          ; Get all the arguments for this call

        DO      X:<NCOADDS,L_COADDS
        JCLR    #0,X:<CDS,FS_RESET      ; Don't coadd reset frame if CDS = 0
        BCLR    #SRAMADDR17,X:PBD
        DO      X:<NFS,FS_RESET         ; Number of Fowler samples per frame
        JSR     <COADD_FRAME            ; This is the reset frame
        NOP
FS_RESET
        BSET    #SRAMADDR17,X:PBD
        DO      X:<NFS,FS_EXPOSE
        JSR     <COADD_FRAME            ; This is the expose frame
        NOP
FS_EXPOSE
        NOP
L_COADDS

        MOVEP   #$2281,X:BCR            ; Wait states for ext. memory accesses
        JMP     <START

; Coadd one 256 x 512 pixel frame
COADD_FRAME
        BCLR    #SRAMADDR16,X:PBD       ; A16 = 0
        JSR     <COADD                  ; one 256x256 pix frame
        BSET    #SRAMADDR16,X:PBD       ; A16 = 1
        JSR     <COADD                  ; another 256x256 frame
        NOP
        RTS

; Define a macro so in-line code can be simplified
COADD_ONE_PIXEL MACRO
        AND     X1,A  Y:(R0),B1
        MOVE    Y:(R0),B2       ; Get coadded D31-D24 from SRAM
        ADD     A,B  X:(R5),A1
        MOVE    B1,Y:(R0)       ; Write D23-D00 to SRAM
        MOVE    B2,Y:(R0)+      ; Write D31-D24 to SRAM, increment pixel pointer
        ENDM

; Coadd 1/2 a frame = 128k pixels (32-bit coaddition)
COADD   CLR     A
        MOVE    A,R0            ; R0 = SRAM address
        MOVE    X0,X:RSTSRAM    ; Initialize the Ping-Pong data pointer
        MOVE    #16384,X0
        DO      X0,COADD_4      ; Read all the image minus four pixels
        JCLR    #EF,X:PBD,*     ; Wait for the FIFO to be not empty
        MOVE    X:(R5),A1       ; Read from the A/D FIFO
        COADD_ONE_PIXEL         ; Macro defined immediately above
        COADD_ONE_PIXEL
```

```
        COADD_ONE_PIXEL
        AND     X1,A  Y:(R0),B1
        MOVE    Y:(R0),B2
        ADD     A,B             ; Don't read FIFO, because its not ready yet
        MOVE    B1,Y:(R0)
        MOVE    B2,Y:(R0)+
COADD_4
        RTS


; This sends 512x512x2 16-bit samples (512x512 32-bit pixels) to the FIFO ******
;*******************************************************************************
; Transmit the coadded image to the CAFIFO for reading by the timing board.
;   Data is always transmitted in 32-bits per pixel format.
TCA     MOVE    A1,X:RSTFIFO            ; Reset A/D and coadder FIFOs
        MOVE    A1,X:RSTSRAM            ; Initialize the Ping-Pong data pointer
        JSET    #0,X:<CDS,L_3
        BSET    #SRAMADDR17,X:PBD       ; CDS = 0
        JSR     <XMIT_FRAME
        JMP     <START
L_3     BCLR    #SRAMADDR17,X:PBD       ; CDS = 1 => Yes correlated double
        JSR     <XMIT_FRAME             ;  sampling
        BSET    #SRAMADDR17,X:PBD
        JSR     <XMIT_FRAME
        JMP     <START

XMIT_FRAME
        BCLR    #SRAMADDR16,X:PBD       ; A16 = 0
        JSR     <WR_CAFIFO              ; dump one 256x256 (32-bit) pix page per board
        BSET    #SRAMADDR16,X:PBD       ; A16 = 1
        JSR     <WR_CAFIFO              ; dump another 256x256 (32-bit) pix page per board
        RTS

WR_CAFIFO                               ; Transmit a 64k (256x256 32-bit) pixel image
        CLR     A
        MOVE    A1,R0
        DO      #1024,L_TCA2
        JCLR    #CAHF,X:PCD,*           ; Wait if the FIFO is half full
        DO      #64,L_TCA3
        MOVE    Y:(R0),A0               ; Get bits D23-D00 of coadded image
        MOVE    A0,B0                   ; Write bits D15-D00 to B0
        MOVE    Y:(R0)+,A1              ; Get bits D31-D24 of coadded image
        REP     #16                     ; Shift bits D31-D16 into position
        ASR     A
        MOVE    A0,X:(R6)               ; Write bits D31-D16 to CAFIFO (first 16 bits into FIFO)
        MOVE    B0,X:(R6)               ; Write bits D15-D00 to CAFIFO (second 16 bits into FIFO)
L_TCA3
        NOP
L_TCA2
        RTS



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

; Clear error condition and interrupt on SSI receiver
CLR_SSI MOVEP   X:SSISR,X:RCV_ERR ; Read SSI status register
        MOVEP   X:SSIRX,X:RCV_ERR ; Read receiver register to clear error
        RTI

; This sends 512x512 16-bit samples (512x512 16-bit pixels) to the FIFO ********
;*******************************************************************************
; This is the pass through (PT) readout code
READ_PT JSR     <GET_ARGUMENTS          ; Get all the arguments for this call
;       MOVE    A,X:RSTFIFO             ; Reset the A/D and coadder FIFOs

;       DO      X:<NCOADDS,L_NCOADDS
```

```
;       JCLR    #0,X:<CDS,L1_NFS        ; One frame per coadd if CDS = 0

;       DO      X:<NFS,L1_NFS           ; Loop over number of Fowler samples
;       MOVE    A,X:RSTFIFO             ; Reset the A/D and coadder FIFOs
;       JSR     <READ_PT_FRAME          ; This is the first frame
;       NOP
;L1_NFS

;       DO      X:<NUTR,L_NUTR
;       DO      X:<NFS,L2_NFS           ; Loop over number of Fowler samples
        DO      X:<NREADS,L2_NFS        ; Loop over NREADS
        CLR     A
        MOVE    A,X:RSTFIFO             ; Reset the A/D and coadder FIFOs

        JSR     <READ_PT_FRAME          ; This is the second frame
        NOP
L2_NFS
        NOP
;L_NUTR
;       NOP

;L_NCOADDS

        JMP     <START

READ_PT_FRAME
                                        ; 1024 rows by 512 cols for HAWAII 1RG (2 amp)
        MOVE    X:<NROWS,A1
;       LSR     A1
        DO      A1,L_ROWS               ;       NROWS/2
        MOVE    X:<NCOLS,A1
        LSR     A1
        DO      A1,L_COLS               ;       NCOLS
        JCLR    #EF,X:PBD,*             ; Wait for the pixel data
        MOVE    X:(R5),X0               ; Read one 16-bit pix from the A/D FIFO (per board) PIX (
amp) 0
        MOVE    X0,X:(R6)               ; Write one 16-bit pix to output FIFO (per board)
        MOVE    X:(R5),X0               ; PIX (amp) 1
        MOVE    X0,X:(R6)
        MOVE    X:(R5),X0               ; PIX (amp) 2
        MOVE    X0,X:(R6)
        MOVE    X:(R5),X0               ; PIX (amp) 3
        MOVE    X0,X:(R6)

L_COLS  NOP
L_ROWS  NOP
        RTS

;******************************************************************************
;*** Command table. This is different from the TIM/TIMBOOT command table ***
;******************************************************************************
; Command table is in P: memory so it downloads from the timing board
NUM_COM EQU     6                       ; Number of commands
TABLE   DC      'ICA',ICA               ; Initialize coadder board
        DC      'RDC',RDC               ; Read image, after exposure
        DC      'TCA',TCA               ; Transmit coadder image to timing board
        DC      'TDL',TDL               ; Test Data Link
        DC      'RPT',READ_PT           ; Read pass through
        DC      'ERR',ERROR
END_TABLE
;******************************************************************************
;*** End of Command table. ***
;******************************************************************************

; Last address of program so number of program words can be calculated
END_ADR

;******************************************************************************
;****  END OF COADDER CODE  *************
;******************************************************************************
```

```
; Check for overflow
        IF      @CVS(N,*)>$6FF
        WARN    'Internal P: memory overflow!'
        ENDIF

        ENDSEC                          ; End of section COADD
```