TNG Tip-Tilt Servo Loop
DSP Software

Luca Fini[*,1], Piero Ranfagni[*,2]

June 1996

[*] Osservatorio Astrofisico di Arcetri

[1] e-mail: *lfini@arcetri.astro.it*
[2] e-mail: *ranfagni@arcetri.astro.it*

Arcetri Technical Report N° 4/1996
Firenze, Giugno 1996

Abstract

The TNG Tip-Tilt Servo-Loop subsystem [1] is the signal processing chain which, based on the evaluation of the image motion, computes the correction to be applied to a flat steering mirror to compensate for tip-tilt components. The heart of the Servo-Loop is a DSP 56001 based board running the **TNG_AO** *program which implements the compensation algorithm.*

In the following pages we report the complete source code of the TNG_AO program with extensive comments.

In order to maintain a close relationship between the code and the related comments the latter have been included as comment lines into the assembler source files which are automatically processed by a small procedure to generate the final text of this report.

Sommario

Il sottosistema Servo-Loop del sistema di controllo del Tip-Tilt di TNG [1] è costituito dalla catena di elaborazione che, sulla base di una stima del movimento immagine, calcola la correzione angolare da applicare ad uno specchio rigido e piano per compensare le componenti di tip-tilt. Il cuore del sottosistema consiste in una scheda che alloggia un DSP 56001 sul quale viene eseguito il programma **TNG_AO** *che implementa l'algoritmo di compensazione.*

Nelle pagine che seguono riportiamo il codice completo del programma TNG_AO dotato di ampi commenti.

Allo scopo di mantenere allineato il codice e la sua descrizione, il testo relativo è stato incluso sotto forma di linee di commento nei files sorgente assembler; questi vengono processati mediante un semplice script per estrarre automaticamente il testo di questo report.

Contents

1 Introduction **1**
- 1.1 How to Read this Document . . . 1
- 1.2 *Tip-Tilt* Servo Loop Architecture . . . 1
  - 1.2.1 Hardware Architecture . . . 1
  - 1.2.2 Loop Implementation and Timing . . . 2
- 1.3 The DSP . . . 2
  - 1.3.1 DSP Board Architecture . . . 2
  - 1.3.2 Modulus Addressing . . . 3
  - 1.3.3 Fractional Arithmetic . . . 3
- 1.4 Interfacing with the Host and with other AO System Components . . . 4
  - 1.4.1 Interaction with the VME Host . . . 4
  - 1.4.2 Reading Counters . . . 5
  - 1.4.3 Controlling the Mirror DACs . . . 5
- 1.5 Software Development . . . 5
  - 1.5.1 Conditional Assembly . . . 5
  - 1.5.2 Interactions with the Monitor Program . . . 6
  - 1.5.3 Register Usage . . . 6

2 The DSP Software Organization **6**

3 Constant Definitions **7**
- 3.1 TNG_AO Code Constants . . . 7
  - 3.1.1 Memory Map Related Constants . . . 7
  - 3.1.2 VME Communication Block . . . 8
  - 3.1.3 Miscellaneous Constants . . . 10
- 3.2 DSP Housekeeping Constants . . . 11
  - 3.2.1 I/O General Constants . . . 11
  - 3.2.2 Port B Detailed Programming Constants . . . 11
  - 3.2.3 Port C Detailed Programming Constants . . . 12
  - 3.2.4 Exception Processing Related Constants . . . 13

4 MACRO Definitions **13**
- 4.1 CLRACC: Clear Accumulation Block . . . 14
- 4.2 DISINTS: Disable All Interrupts . . . 14
- 4.3 READW: Read a Word from the Host Interface . . . 14
- 4.4 WRERR: Write an Error Message . . . 14
- 4.5 RDCNT: Read APD Counters . . . 15
- 4.6 WRDAC: Write to Mirror DAC . . . 17

5 Memory Organization **18**
- 5.1 P Memory Usage . . . 19
- 5.2 X and Y Memory Usage . . . 20

6 Initialization 23
6.1 DSP Status Initialization 23
6.2 TNG_AO Variables and Buffers Initialization 25
6.3 Utility Subroutines 26

7 Host Command Processing 27
7.1 Control Commands 27
7.1.1 STRTMON: Start Monitor Code 27
7.1.2 LDPARAM: Load Loop Parameters 28
7.1.3 LDCOEFF: Load Filter Coefficients 28
7.1.4 LDSAMPL : Load Mirror Drive Samples 29
7.1.5 STPLOOP: Stop the Loop 30
7.2 Data Access Commands 30
7.2.1 SNDIDNT: Send Back Identification String (Banner) 30
7.2.2 DWNLBUF: Download the Data from the Run-Time Data Buffer 31
7.2.3 WRERROR: Write Last Error Code to Host Interface 32
7.3 Loop Initialization Commands 33
7.3.1 OPNLOOP: Start the "Open Loop Mode" 33
7.3.2 CLSLOOP: Start the "Closed Loop Mode" 34
7.3.3 MRDRIVE: Start the "Mirror Drive Mode" 34
7.4 Loop Initialization Common Code 35
7.5 Subroutines 37
7.5.1 STOP: Terminate the Current Loop 37
7.5.2 STIMR: Preset the Interrupt Timer 37

8 Main Loop 38
8.1 The VME Communication Block 38

9 Timer Interrupt Servicing Routine 40
9.1 *Tip-Tilt* Components Evaluation 40
9.2 Filters 47
9.2.1 North-South Channel 48
9.2.2 East-West Channel 49
9.3 End of Timer Interrupt Service 51
9.3.1 Reset Counters 51
9.3.2 Returning from the Timer Interrupt 51

10 Interrupt Vectors Table 51
10.1 Host Command Table 52
10.2 Timer Interrupt Vector 52

A The DSP Bootstrap Process 53

B Symbol List 54

List of Figures

1 Input/Output Port Bits Usage Specification 4
2 Status Word Description 9
3 TNG_AO Memory Map 19
4 Loop data storage organization 31
5 Loop data transmission order 31
6 Loop Code Structure (one channel) 33
7 *Tip-tilt* Components Reference 41

1 Introduction

In the following pages we report the complete source code of the TNG *tip-tilt* Servo Loop DSP software together with extensive comments and explanations needed to fully understand the algorithms employed and their implementation.

In this introductive section we have gathered some miscellaneous information which may be useful to understand the following descriptions. These include some notes on the TNG *tip-tilt* servo-loop subsystem architecture, a brief resume of the DSP internal architecture (mainly related to the numerical representation and operations), some information about the interfacing with the other system components and a number of notes about the software implementation. References to other documents including DSP manuals are quoted when appropriate.

1.1 How to Read this Document

This document is generated from the source files of the TNG_AO program which contain extensive comments with LaTeXformatting commands. The files as such can be directly fed to the DSP56000 Macro Assembler [7] to generate the executable program; the LaTeXsource is obtained by a simple preprocessing procedure.

Sections from 2 to 10 describe the TNG_AO code and approximately reflect the internal structure of the code itself.

When the document is formatted the actual source code appears in fragments appropriately interspersed in the text which are typed in a different typeface than the text to be easily identified. Each fragment of code begins with a separation line which also contains the name of the source file to ease in locating portions of the code.

All the figures included in the document have been made by using ASCII characters: they are not quite pleasant to look at, but have the advantage to be readable also when lookin at the source files.

Appendix A contains a brief description of the bootstrap process and the related code organization of the TNG_AO software.

Finally appendix B contains a table in which all symbols are listed, alphabetically ordered, with their location in memory.

1.2 *Tip-Tilt* Servo Loop Architecture

The *tip-tilt* servo loop architecture is fully covered elsewhere [1], but an overall description may be useful for the reader in order to better understand many details of the following dissertation.

1.2.1 Hardware Architecture

The TNG *tip-tilt* Servo Loop is made up of three main parts:

- The **Quad Cell** is made of four Avalanche Photodiodes where the light of the reference star spot is directed by the optical assembly. The photodiodes counts are accumulated in four counters which can be read from the DSP (see below).

- The **System Controller**, a VME based host system which also includes a DSP board which implements the *tip-tilt* correction algorithm.

- The **Steering Mirror**, a voice coil driven flat mirror which applies corrections to the beam. It receives position commands from the DSP board.

The DSP board is a commercial board (DBV56H by Loughborough) holding a Motorola DSP 56001 processor which can communicate with the host computer (an MC 68040 based CPU) via the VME bus.

The host computer controls the loop operation by properly setting the parameters and monitors the proper functioning based on diagnostic data received from the DSP.

The system software is made up of two parts: the host (or supervisor) software, written in C language and described elsewhere [3], and the DSP code which implements the time tight *tip-tilt* evaluation algorithm, written in DSP 56001 Macro Assembler [7] and is described in detail in the following pages.

1.2.2 Loop Implementation and Timing

In the normal operation of the loop a single step is started when an interrupt is received by the DSP internal timer. At each step the DSP software performs the following actions:

- Read the values of the four APD counters.
- Compute the two *tip-tilt* components.
- Filter the two *tip-tilt* components.
- Apply the computed values to the Steering Mirror.

During the above process a set of diagnostic data are gathered. In the idle time between the end of the step processing and the next interrupt from the timer the DSP can transmit the diagnostic data to the host.

To allow a correct operation of the servo loop the interval between two subsequent interrupts from the timer must be at least long enough to allow the above cycle to be completed. An approximation of the cycle execution time can be obtained from the following relation:

$$T = 0.2847N + 43.14$$

Where T is the cycle time in microseconds, and N the length (number of coefficients) of the filter. The relation is valid for a 27 MHz DSP clock.

1.3 The DSP

A complete description of the DSP 56001 features can be found in the related documentation [6, 4]. Here we resume only some pieces of information which may be useful for the understanding of the TNG_AO code.

1.3.1 DSP Board Architecture

The DBV56H board [4] is a VME based board hosting a 27 MHz DSP 56001 Digital Signal Processor and provided with a total of 32 Kw of Program memory and 64 Kw of data memory. The data memory is divided into two blocks (referred to as X and Y memories) with parallel

access capabilities which together with the parallel execution of some ALU instructions yield to a very efficient execution of code [6].

The board makes some of the DSP I/O ports available, both for communication with the VME host and for interaction with external subsystems. One of the DSP serial port is dedicated to an RS232 interface to connect to a PC for software development and debugging.

The DBV56H board comes with a EPROM containing a debugging monitor which allows code downloading and debug interaction via a serial line connected to a PC hosting the development system.

1.3.2 Modulus Addressing

The Modulus Addressing capabilities of the DSP 56001 address computation logic are used estensively in the TNG_AO code and deserve some introduction.

Besides the usual register indirect memory addressing modes the DSP 56001 is provided with an address modifier register (usually referred to as `Mx`) which can be used to implement circular buffers with sizes in the range 2–32768. When the DSP address computation ALU computes an address resulting from any operation on an address register `Rx`, the computation is performed with modulo arithmetic, where the modulus is the value of the corresponding `Mx` register plus one. This causes the resulting address value to remain within an address range of size `Mx+1`. The lower boundary (base address) must have zeros in the k LSBs, where $2^k \geq Mx+1$, and therefore must be a multiple of 2^k. The DSP56000 Macro Assembler provides a memory storage definition instruction which properly alignes memory areas which are to be used as circular buffers.

1.3.3 Fractional Arithmetic

The DSP 56001 ALU directly supports fractional arithmetic [5]. This mainly affects the multiplication because the sum is analogous to the usual integer one.

A fraction F is a number whose magnitude satisfies the inequality: $0.0 \leq |F| < 1.0$. The binary word is interpreted as having a binary point after the most significant bit (MSB). The MSB is, as usual, interpreted as sign. The range of numbers which can be represented in this form using N-bit two's complement is: $-1.0 \leq F \leq 1.0 - 2^{-(N-1)}$.

The multiplication instruction used in the following code multiplies two fractions stored into two 24 bit registers and the result is added to the content of a 48 bit accumulator, extended with an 8 bit extension register to protect against overflow.

Two different machine instructions may be used for multiplication: `MAC`, which operates as described above, and `MACR`, which also rounds the result in the most significant word of the accumulator while clearing the least significant word. This operation is usually employed as the last one in the vector multiplication iteration loop to obtain the maximum precision for the desired 24 bits fractional result.

In order to obtain the correct values in our implementation of the FIR filter (which must be representable as fractional 24 bits values) the filter coefficients must be properly set so that the final result (the output of the filter) doesn't cause overflow in the extension bits of the accumulator.

1.4 Interfacing with the Host and with other AO System Components

The DSP board interfaces with the VME Host by the DSP Host Port, as explained below, and with the APD counters and with the steering mirror electronics through the on-board 24 bit I/O port. The DSP board also provides an RS232 compatible serial port for interaction with the development system.

A detailed description of the APD counter board which also contains the logic for writing to the mirror DACs, can be found in the related documentation [2].

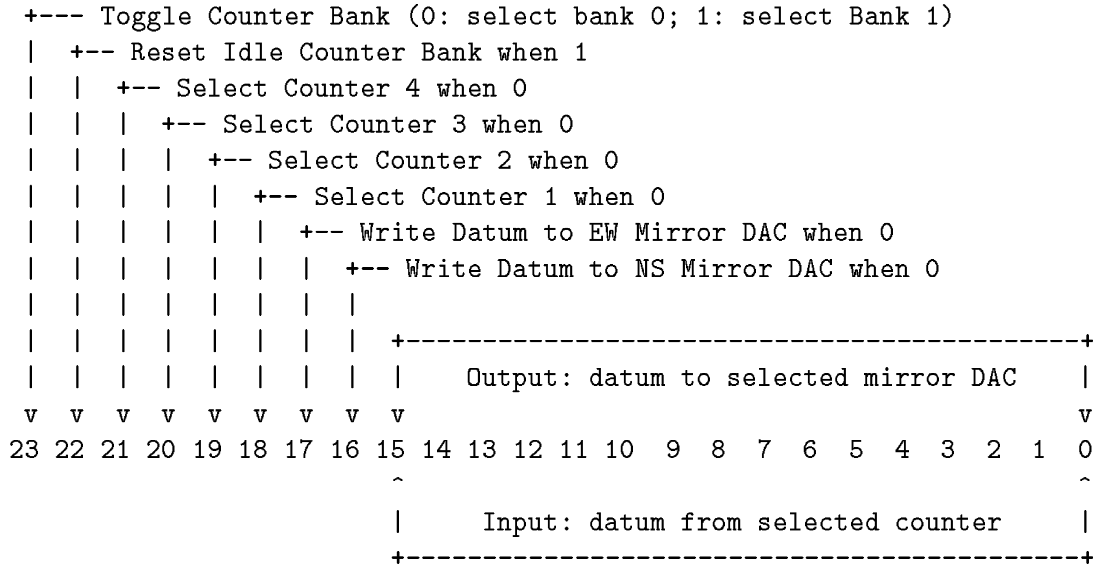


Figure 1: Input/Output Port Bits Usage Specification

The 24 bit I/O port is a bidirectional port seen as a memory-mapped "external" peripheral in the DSP data address space (see [6] section 9 and [4]). In figure 1 we report the specification of I/O port bits usage. The most significant 8 bits are used for control purposes and the remaining 16 bits are used for data input and output. The port is used both for reading from the counter banks and for driving the steering mirror.

1.4.1 Interaction with the VME Host

The VME bus interface is implemented by the DSP Host Port and is seen as a memory-mapped "internal" peripheral occupying three 24-bit words in data memory space (see [6] section 10).

The interface consists of three registers. 1) Host Control Register (HCR), a read-write 8-bit register from the DSP point of view corresponding to a read-only ISR register from the VME host point of view. 2) Host Status Register (HSR) a read-only 8-bit register from the DSP point of view corresponding to two registers (ICR and CVR) from the host side. 3) Data Register, a 24 bits read-write register from the DSP point of vies corresponding to a 32 bit read-write register, addressed as four separate bytes from the host side (the upper byte always is read as 0

by the host).

The Host Interface is used in the TNG_AO code for data exchange, for some auxiliary synchronization, and to allow the host to control the DSP operations. Data exchange is performed through the Data Register, with synchronization through the HCR and HSR registers; auxiliary synchronization is performed by some flag bits in the HCR; DSP control is performed by Host Commands issued through the CVR-HSR registers.

1.4.2 Reading Counters

At each cycle the counter bank selection line (bit 23 as shown in figure 1) is toggled so that the bank which was previously counting is ready for readout, while the other starts counting (it was previously reset to zero at the end of the preceding cycle). Data are read out by setting the corresponding line to 0 (bits 18-21, one for each counter), while maintaining the other lines at the previous state, performing a read operation from the I/O port and resetting the line to 1. The operation is coded as macro `RDCNTS` (see section 4).

This cycle is repeated four times to read the four values. Finally the bank counters are reset to 0 by setting to 1 the related output bit (22) for a period whose length depends on a configuration parameter (see `DELAY` in section 3 which defines the number of repetitions of a `NOP` instruction.

1.4.3 Controlling the Mirror DACs

The mirror position is sent to the DACs by outputting the required integer value to the 16 least significant bits of the output port and then cycling the related control bit (bit 16 for the NS channel and bit 17 for the EW channel). The control bit is set to zero for a period depending on the parameter `DELAY` already quoted above. The operation is coded as macro `WRDAC` (see section 4).

1.5 Software Development

The TNG_AO DSP software has been developed by means of the DSP56000 Macro Assembler [7] running on a PC and the on-board monitor program distributed with the DSP board.

In the following section are gathered some details regarding the use of some assembler capabilities, the interaction with the on-board monitor, and the like.

1.5.1 Conditional Assembly

In order to ease the development both of the preliminary version of the TNG_AO software and of the final version to be hosted on the board EPROM, we use the conditional assembly capabilities of the DSP56000 Macro Assembler. Here follows a list of the symbols we employ in the code:

- **EPROM_VER**. Defined to generate the final version of the code to be burnt into EPROM. It includes some code which could conflict with the Monitor.
- **TIMINT**. Defined to allow the timer to actually send interrupts. During the development phase it is sometimes useful to switch off the interrupt timer and simulate it with an explicit

command from the host which allows to examine the system after a single shot [1].

- **NOCNT**. If defined the code will not include the instructions to actually read the values of the APD counters; count values are instead got from four location in memory. This allows to debug the algorithm by feeding the system with constant input values.

1.5.2 Interactions with the Monitor Program

In the development version of the TNG_AO code the DSP board hosts the Monitor EPROM while the TNG_AO code is loaded via the serial line from the development system. The board is configured to load and start the Monitor at boot (see also appendix A).

In the final version of the code both the Monitor and the TNG_AO code will reside in EPROM and will be loaded into memory at boot, but the control will start from the TNG_AO code. The monitor will anyway be restartable via a proper host command for testing and final debugging.

1.5.3 Register Usage

In order to obtain the maximum time efficiency we have maintained to a minimum the need of saving the status of registers at the end of each section of code. This obviously puts some constraints on the usage of registers which are described in the following.

`R0` (together with `M0` and `N0`). Pointer to the communication buffer during transmission to the host.

`R2` (together with `M2` and `N2`). Pointer to the circular input buffer for the FIR filter.

`R3` (together with `M3` and `N3`). Pointer to the big data buffer where run-time values are stored (see sections 7 and 5.2).

`R5` (together with `M5` and `N5`). Pointer to the circular buffers from where data to feed the steering mirror are taken (either the output of the filter, or the tables of mirror drive values, or the zeroed buffer as explained in section 7).

`R6` (together with `M6` and `N6`). Pointer to the circular buffer of filter coefficients.

`R7` (together with `M7` and `N7`). Pointer to the accumulation buffer.

2 The DSP Software Organization

For ease of management we have divided the DSP code into various sections, stored into single files, which are included from the following piece of code with the include directive.

The order of the following include directives must not be modified because in some instances the control passes from one block of code to the other simply because they are stacked one after another.

[1] The board interface allows the host program to issue any interrupt from the VME interface, not only the twelve "host commands" described in section 7.

```
        OPT     CEX,MEX
        PAGE    255,49,3,3
        TITLE   'DSP 56001 TNG_AO_M'
;---------------------------------------------------------------------------
;              TNG Tip-Tilt System DSP 56K Assembler Code                  +
;                L. Fini, P.Ranfagni Vers. 1.11 Aug 1996                   +
;          Serial Loading Version using Monitor  V5.20 20/02/91            +
; O S S E R V A T O R I O   A S T R O F I S I C O   D I   A R C E T R I +
;---------------------------------------------------------------------------

        opt     mu
                                ;---------------------------------------
        include 'ao_equ'        ; TNG_AO equates and constants
        include 'ao_macro'      ; Macros
        include 'ao_mem'        ; TNG_AO workspace memory area

                                ;---------------------------------------
        include 'ao_init'       ; TNG_AO Initialization
        include 'ao_com'        ; TNG_AO Command processing

        include 'ao_int'        ; TNG_AO Timer interrupt service - Start
        include 'ao_filt'       ; TNG_AO Timer interrupt service - Filter
        include 'ao_endi'       ; TNG_AO Timer interrupt service - End

                                ;---------------------------------------
        include 'ao_vec'        ; TNG_AO exception vectors table
        end
```

3 Constant Definitions

In this section we have gathered together all the definitions related to constants both for the TNG_AO code and for DSP housekeeping purposes.

3.1 TNG_AO Code Constants

The following definitions are related to the TNG_AO software and are subdivided into a few separated subsection for ease of reference.

3.1.1 Memory Map Related Constants

The following definitions are related to memory areas used by the TNG_AO software.

Notes:

1. The first part of both X and Y memories are used as holding buffers for real time data gathered during the loop[2]. Because the buffers are used as circular buffers to store the most recent data, they must start at an address boundary which is a multiple of a power of

[2]The detailed usage of such buffers is described elsewhere (see section 5.2).

two greather than or equal to the buffer length. Allocating them at address \$0000 allows any possible length of the circular buffer[3].

2. The value BUF_ORG is set so that in the following memory areas some others properly aligned circular buffers can be fitted, as detailed in the code.

Source file: ao_equ.asm

```
                            ; MEMORY MAP ASSIGNMENTS
                            ; P_Memory
P_ROM   EQU     $800        ; Program area start address
P_RAM   EQU     $1800       ; Data area start address
                            ;----------------------------------------------------
                            ; Due to circular buffer allocation X_RAM and Y_RAM
                            ; MUST both be $0 !!!
X_RAM   EQU     $0000       ; AO_M serial version XRam variables start address
Y_RAM   EQU     $0000       ; AO_M serial version YRam variables start address

BUF_ORG EQU     $6D00       ; This value has been computed so that after this
                            ; in both X and Y memories we may have:
                            ;       254 locations for general purpose variables
DUMMY   EQU     $6DFE       ;         2 locations for dummy circular buffer
                            ;    2x 256 locations for circular buffer
                            ;      4096 locations for circular buffer
                            ; without vasting any space
                            ;
                            ; NB: To allow double store operations in the accumu-
                            ;     lation buffers, the two blocks in X and Y
                            ;     memories MUST be aligned at GEN_PUR
```

3.1.2 VME Communication Block

The VME Communication Block is the data area used to store and accumulate run-time information gathered during the loop and periodically transmitted to the VME Host. The costants defined in the following fragment of code are used to ease the addressing of the single words in the Communication Block.

The following values are offsets in the Communication block.

Source file: ao_equ.asm

```
STATUS  EQU     $0              ; Status Word (see below).
C1_SUM  EQU     $1              ; Current sum for Counter #1
C2_SUM  EQU     $2              ; Current sum for Counter #2
C3_SUM  EQU     $3              ; Current sum for Counter #3
C4_SUM  EQU     $4              ; Current sum for Counter #4
NS_SUM  EQU     $5              ; Current sum for North-South read out
EW_SUM  EQU     $6              ; Current sum for East-West read out
NS_CORR EQU     $7              ; Current sum for North-South correction
```

[3] See the description of circular buffer implementation capabilities of the DSP 56001 in section 1.3.2.

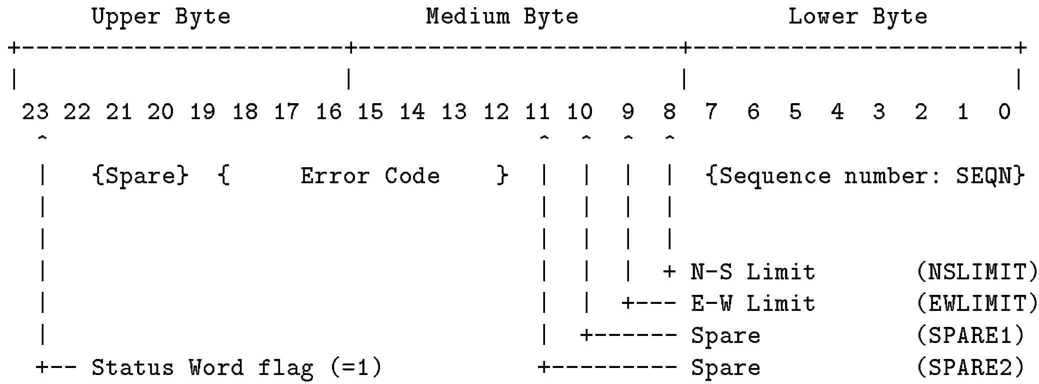


Figure 2: Status Word Description

```
EW_CORR EQU     $8              ; Current sum for East-West correction

                                ; Status Word bit flags
NSLIMIT EQU     $8
EWLIMIT EQU     $9
```

With reference to figure 2 here follows a description of each field.

- **SEQN** (Bits 0-7) - Sequence Number: Byte A value which is incremented every time a new block is written. It can be used by the host to check whether any block has been lost.
- **NSLIMIT** (Bit 8) - N-S Limit Reached: Flag Set when the North-South Filter output value reaches the representation limit in any iteration. The Filter algorithm is implemented with DSP 56001 MACR instruction which accumulates the output in a 56 bit accumulator. The final filter output must, instead, be representable as a 24 bit fractional value (see section 1.3.3). If this doesn't happen this bit is set so that the host can be made aware of it and subsequently modify the filter coefficient to avoid the problem.
- **EWLIMIT** (Bit 9) - E-W Limit Reached: Flag Set when the North-South Filter output value reaches the representation limit in any iteration. See NSLIMIT flag for details.
- **SPARE1** (Bit 10) - Currently unused Set when the East-West Correction value reaches the upper limit in any of the iterations.
- **SPARE2** (Bit 11) - Currently unused Set when the East-West Correction value reaches the lower limit in any of the iterations.
- **ERRC** (Bits 12-19) - Error Code Usually zero, set to a value specifying the error condition (see values in the relevant code fragment).

Source file: ao_equ.asm

```
                                 ; ERROR CODES
E_NOERR EQU     $000000          ; No error
E_NOPAR EQU     $001000          ; Parameters have not been loaded.
E_NOCOE EQU     $002000          ; Filter coefficients have not been loaded.
E_NOSAM EQU     $003000          ; Mirror drive samples have not been loaded.
E_OVTIM EQU     $004000          ; Time not enough to finish elementary cycle
E_STOPD EQU     $005000          ; Cycle stopped by host
E_MAXCY EQU     $006000          ; Max number of cycles reached
```

3.1.3 Miscellaneous Constants

Here follows some constants used to address specific bits in a variable used for various synchronization purposes.

Source file: ao_equ.asm

```
                                 ; SEMAPHORS AND FLAGS
TMR_SEM EQU     $0               ; Timer interrupt overlapping flag
TRN_SEM EQU     $1               ; Transmission synchronization flag
LOO_SEM EQU     $2               ; Loop is active flag
```

Here follows the definition of constants related to size of buffers, etc. and various other definitions.

Source file: ao_equ.asm

```
                                 ; DIMENSIONING PARAMETERS
MAXFILT EQU     256              ; Maximum filter length

                                 ;-------------------------------------
                                 ; MISCELLANEOUS
LFEED   EQU     $A               ; ASCII LineFeed code

CNTSWTC EQU     23               ; Counter switch bit
RES_CNT EQU     22               ; Counter reset bit

NS_DAC  EQU     16               ; Bit to write to NS DAC
EW_DAC  EQU     17               ; Bit to write to EW DAC

OPEN    EQU     $0               ; Loop mode flag bit in P:LMODE

MONITOR EQU     $7C00            ; Monitor program entry point

DELAY   EQU     10               ; Delay cycle length for line stabulization

WAITACK EQU     1500             ; Number of checks on the error acknowledgment
                                 ; flag
```

3.2 DSP Housekeeping Constants

The following constants are used for the programming of peripherals, exception processing and the like. More details can be found in section 6 which deals with the TNG_AO software initialization.

3.2.1 I/O General Constants

Here we define a number of constants of general use: address of control registers, peripherals, etc.

Source file: ao_equ.asm

```
                                ; EQUATES FOR I/O PORT PROGRAMMING
                                ; Register Addresses
BCR     EQU     $FFFE           ; Port A Bus Control Register
PBC     EQU     $FFE0           ; Port B Control Register
PBDDR   EQU     $FFE2           ; Port B Data Direction Register
PBD     EQU     $FFE4           ; Port B Data Register
PCC     EQU     $FFE1           ; Port C Control Register
PCDDR   EQU     $FFE3           ; Port C Data Direction Register
PCD     EQU     $FFE5           ; Port C Data Register

                                ;-------------------------------------
                                ; EXTERNAL PERIPHERAL ADDRESSES
PORTOUT EQU     $FFC0           ; 16 BIT output port
PORTIN  EQU     $FFC0           ; 16 BIT input port
XIRQA   EQU     $FFC8           ; IRQA source select
XIRQB   EQU     $FFC9           ; IRQB source select
SAMP    EQU     $FFCA           ; Sample clock source
INVERT  EQU     $FFCB           ; Inversion of external interrupts
```

3.2.2 Port B Detailed Programming Constants

The B port is used as Host Interface Communication port HI.

Source file: ao_equ.asm

```
                                ; EQUATES FOR HOST INTERFACE COMMUNICATIONS
                                ; Register Addresses
HCR     EQU     $FFE8           ; Host Control Register
HSR     EQU     $FFE9           ; Host Status Register
HTX     EQU     $FFEB           ; (To) Host Transmit Data Register
HRX     EQU     $FFEB           ; (From) Host Receive Data Register

                                ; HI Control Register Bit Flags
HRIE    EQU     $0              ; Host Receive Interrupt Enable
HTIE    EQU     $1              ; Host Transmit Interrupt Enable
HCIE    EQU     $2              ; Host Command Interrupt Enable
HF2     EQU     $3              ; Host Flag 2
HF3     EQU     $4              ; Host Flag 3
```

```
                                ; HI Status Register  Bit Flags
HRDF    EQU     $0              ; Host Receive Data Full
HTDE    EQU     $1              ; Host Transmit Data Empty
HCP     EQU     $2              ; Host Command Pending
HF0     EQU     $3              ; Host Flag 0
HF1     EQU     $4              ; Host Flag 1
```

3.2.3 Port C Detailed Programming Constants

The C port is the serial communication interface. It is not free for usage by the TNG_AO code because it's used for the interaction with the Monitor software during the development phase.

Source file: ao_equ.asm

```
                                ; EQUATES FOR SERIAL COMMUNICATIONS INTERFACE

                                ; Register Addresses
SRXL    EQU     $FFF4           ; SCI Receive Data Register (low)
SRXM    EQU     $FFF5           ; SCI Receive Data Register (middle)
SRXH    EQU     $FFF6           ; SCI Receive Data Register (high)
STXL    EQU     $FFF4           ; SCI Transmit Data Register (low)
STXM    EQU     $FFF5           ; SCI Transmit Data Register (middle)
STXH    EQU     $FFF6           ; SCI Transmit Data Register (high)
STXA    EQU     $FFF3           ; SCI Transmit Data Address Register
SCR     EQU     $FFF0           ; SCI Control Register
SSR     EQU     $FFF1           ; SCI Status Register
SCCR    EQU     $FFF2           ; SCI Clock Control Register

                                ; SCI Control Register Bit Flags

WDS     EQU     $3              ; Word Select Mask
WDS0    EQU     0               ; Word Select 0
WDS1    EQU     1               ; Word Select 1
WDS2    EQU     2               ; Word Select 2
SBK     EQU     4               ; Send Break
WAKE    EQU     5               ; Wake-up Mode Select
RWI     EQU     6               ; Receiver Wake-up Enable
WOMS    EQU     7               ; Wired-OR Mode Select
RE      EQU     8               ; Receiver Enable
TE      EQU     9               ; Transmitter Enable
ILIE    EQU     10              ; Idle Line Interrupt Enable
RIE     EQU     11              ; Receive Interrupt Enable
TIE     EQU     12              ; Transmit Interrupt Enable
TMIE    EQU     13              ; Timer Interrupt Enable

                                ; SCI Status Register Bit Flags

TRNE    EQU     0               ; Transmitter Empty
TDRE    EQU     1               ; Transmit Data Register Empty
RDRF    EQU     2               ; Receive Data Register Full
```

```
IDLE    EQU     3               ; Idle Line
OR      EQU     4               ; Overrun Error
PE      EQU     5               ; Parity Error
FE      EQU     6               ; Framing Error
R8      EQU     7               ; Received Bit 8

                                ;----------------------------------------
                                ; serial port clock division ratio for
                                ; various baud rates
                                ; div = [(clock_rate/64) / baud_rate] - 1

CLKDIV   EQU      43            ; 27.000 Mhz, 9600 bauds
```

3.2.4 Exception Processing Related Constants

Here are gathered all the constants related to exception processing.

Source file: ao_equ.asm

```
                                ; EQUATES FOR EXCEPTION PROCESSING

IPR     EQU     $FFFF           ; Interrupt Priority Register (Peripherals)

PIC     EQU     $CC00           ; Peripherals Interrupt code - Priority 2
                                ; on host interface, unchanged SCI

                                ;----------------------------------------
                                ; wait states for memory and i/o
                                ; range 0 - 15  (0  = no wait states,
                                ;               15 = 15 wait states)

XWAIT   EQU     0               ; X Memory wait state value
YWAIT   EQU     0               ; Y Memory wait state value
PWAIT   EQU     0               ; P Memory wait state value
IOWAIT  EQU     1               ; I/O wait state value (as in Monitor!!)

                                ; the total wait state configuration value
                                ; is the combination of the above values

WAITS   EQU     (XWAIT*$1000)+(YWAIT*$100)+(PWAIT*$10)+IOWAIT

HOSTVEC EQU     $24             ; Start of Host command area in interrupt
                                ; vector
TIMVEC  EQU     $1C             ; Timer interrupt area in interrupt vector
```

4 MACRO Definitions

The following macros are used in various sections of the TNG_AO code.

4.1 CLRACC: Clear Accumulation Block

The Accumulation Block is one of the two Data Block used for status transmission to the host, namely the one currently used for status data accumulation, while the other is being transmitted to the host.

Source file: **ao_macro.asm**

```
CLRACC  MACRO                   ; Clear current Accumulation block
        MOVE    P:ACC_PTR,R0    ; Load current CMB pointer

        MOVE    #0,X0
        DO      #CM2_BLX-CM1_BLX,_END
        MOVE    X0,X:(R0)+
_END
        ENDM
```

4.2 DISINTS: Disable All Interrupts

This macro is used to disable all interrupts (in our case both the timer interrupt and the interrupt from host commands) at the beginning of the servicing of an host commend to ensure that a following command issued by the host, or an interrupt from the timer do not interrupt the current host command processing before it is finished.

Source file: **ao_macro.asm**

```
DISINTS MACRO                   ; Macro to disable both interrupt timer and
                                ; host commands
        BCLR    #TMIE,X:SCR     ; disable interrupt from timer
        BCLR    #HCIE,X:HCR     ; disable Host commands
        ENDM
```

4.3 READW: Read a Word from the Host Interface

This macro is used to read a word from the Host Interface and store it in a given location.

Source file: **ao_macro.asm**

```
READW   MACRO   ADDRESS         ; Macro to read a word from the VME and
                                ; store it into ADDRESS
        JCLR    #HRDF,X:HSR,*   ; Wait for Data register full
        MOVEP   X:HRX,ADDRESS   ; Move word to address
        ENDM
```

4.4 WRERR: Write an Error Message

This macro is used to store the error code in a memory location and sets the error flag. Prior of reading any word from the DSP the Host checks for the error flag. If an error is signaled it breaks

the normal data reading cycle, gets the error message by issuing the related Host Command (see WRERROR in section 7), acknowledges the error reading, and acts accordingly.

Source file: ao_macro.asm

```
WRERR   MACRO   CODE            ; Write an error code to VME
        MOVE    CODE,X0
        NOP
        MOVE    X0,Y:LAST_ER    ; Save last error code (Debug)
        BSET    #HF3,X:<<HCR    ; Set Error flag

        ENDM
```

4.5 RDCNT: Read APD Counters

This macro gets a count value from Counter x accumulates the sum of counts in CxSUM, computes the equalized count value and stores it into CxTMP. Where x [equal to 1,2,3,4], indicates the particular counter. The logic of counter banks control is explained in detail in section 1.4.2.

Notes:

1. The APD counters board [2] has two counter banks which must be swapped so that when reading from one bank, the other will continue to accumulate counts. The macro controls the swapping of counter banks by means of the Most significant bit of the control word; because we actually use four different words preset to the right values to address both the counter bank and the particular counter (argument CxSEL) the bank selecting bit is toggled at the beginning of the macro, but only the first invocation of the macro actually toggles the status of the output line; the others are only necessary to maintain the line at the same state. The same mechanism is used when we apply a fifth command word necessary to clear the counters after reading (see section 9).

2. Due to the fixed point fractional arithmetic used by the DSP 56001 (see also section 1.3.3), the algorithm needs to be explained in some deeper detail.

   Each counter value is stored into DSP memory as it is read from the counter. Those values in fractional arithmetic appears as if multiplied by a factor 2^{-23} because the fractional point is assumed to be at the right of the most significant bit of the word.

   Sums and differences are simple to compute because they aren't affected by the fractional representation of numbers. When multiplying or dividing, instead, care must be used to take into account the underflow. In fractional arithmetics numbers are always lower than 1, so that when computing multiplications you cannot have overflow, but you may loose precision; on the contrary, when computing divisions, you must take care of possible overflow.

   In the following code data are properly scaled in order to avoid both problems. E.g.: each counter value is multiplied by two, and then by an equalization factor precomputed as part of the system calibration process. The values of the equalization coefficients (which are necessarily less than one, due to the fractional fixed point representation of numbers)

may be properly chosen in order to avoid to loose precision. I.e.: the coefficient related to the "best" counter (the one with the higher quantum efficiency) may be set to 0.5 (the resulting value combined with the factor 2 is thus 1.0) while the other coefficients will have values (in the range [0.5-1.0]) computed in order to cancel the differences in efficiency[4]. Overflow may not occur because we have 16 bits counters stored into 24 bits words so that we have 8 guard bits, more than enough to also avoid overflow when subsequently summing the four values.

3. To allow preliminary debugging of the code we have introduced the conditional compilation symbol `NOCNT` to generate code which doesn't actually read from counters, but returns constant values: if `NOCNT` is not zero, the macro initializes the input value with the content of argument `Sx`, ignoring the counter readout.

Source file: **ao_macro.asm**

```
RDCNT   MACRO   CxSEL,CxSUM,CxTMP,CxEQL,Sx,xBUF

        ; Arguments:

        ;            1           2           3           4
        ; CxSEL:    CNT1SEL,    CNT2SEL,    CNT3SEL,    CNT4SEL
        ; CxSUM:    C1_SUM,     C2_SUM,     C3_SUM,     C4_SUM
        ; CxTMP:    X:CN1_TMP,  Y:CN2_TMP,  X:CN3_TMP,  Y:CN4_TMP
        ; Sx:       CNST_10,    CNST_15,    CNST_20,    CNST_25
        ; xBUF      Y:(R3)      X:(R3)+     Y:(R3)      X:(R3)+

        ; At the end of the macro register B contains the equalized
        ; counter value (CxTMP)

        BCHG    #CNTSWTC,Y:CxSEL        ; Switch counter banks
        MOVE    Y:CxSEL,X1              ; Get Counter select command word
        MOVEP   X1,Y:PORTOUT            ; Select Counter x

;               Prepare offset
        MOVE    #CxSUM,N7
        MOVE    P:MASK_16,Y0

        REP     #DELAY                  ; Wait for line status to stabilize
        NOP

;               A1=Cx(raw)
        MOVEP   Y:PORTIN,A              ; Read Counter x

        IF      NOCNT                   ; Include this code to simulate counters
        MOVE    P:Sx,A                  ; SIMUL: This MOVE simulates a count
                                        ;        from the input counter
        ENDIF
```

---

[4] We have implicitly assumed that no APD with a difference in quantum efficiency from the other of the group greater than 50% will be used.

```
        AND     Y0,A                    ; Mask 8 Msbits
        ASL     A                       ; Multiply by 2

;               X0=Cx(raw)
        MOVE    A1,X0
;                               A=CxSUM (prepare to accumulate)
        MOVE                    X:(R7+N7),A

        MOVE    X0,xBUF                 ; Save input value into buffer

;               A=CxSUM+Cx
        ADD     X0,A

;               B=0     X0=EQx (prefetch)
        CLR     B       X:CxEQL,Y0

;               B=Cx*EQx        Save CxSUM
        MACR    X0,Y0,B         A,X:(R7+N7)

;               Save Cx(equal)
        MOVE    B,CxTMP

        MOVE    Y:CSUMTMP,X0    ; Load previous sum (4 counters)
        ADD     X0,B            ; Add current (equaliz.) count
        MOVE    B,Y:CSUMTMP     ; Save sum

                                ; Note: the code calling this macro uses
                                ; the result of previous ADD to check if
                                ; the sum of the four counters is zero
        ENDM
```

4.6 WRDAC: Write to Mirror DAC

This macro is used to write a word to the mirror DAC.

Source file: **ao_macro.asm**

```
WRDAC   MACRO   DAC             ; Output data into DAC
                                ; DAC is a bit address to select
                                ; the channel
                                ; X0 contains the Datum
        MOVE    Y:CNRESET,Y0

        CLR     A
        CMP     X0,A
        JGE     _OK_DAC

        MOVE    #$ffff,A
        CMP     X0,A
        JLT     _OK_DAC

;               A=Out
```

```
        MOVE    X0,A

_OK_DAC
;               A1=OutWord
        OR      Y0,A

        MOVE    A1,Y:PORTOUT            ; Put datum in output port
        BCLR    DAC,A1                  ; Clear Write bit.

        REP     #DELAY                  ; Wait for a stable line status
        NOP

        MOVE    A1,Y:PORTOUT            ; Put datum to output port

        REP     #DELAY                  ; Wait for a stable line status
        NOP

        BSET    DAC,A1                  ; Set Write bit
        MOVE    A1,Y:PORTOUT            ; Put datum to output port

        REP     #DELAY                  ; Wait for a stable line status
        NOP

        ENDM
```

5 Memory Organization

In the following section the organization of the memory areas is described in detail.

The DSP 56001 has three memory areas usually named P, X, Y. Each of them, in the configuration chosen for the DBV56H board, is made up of 32 K of 24 bits words.

The P (Program) memory (or Pram) is dedicated to executable code [5]. The first 64 words of the Pram are reserved to the interrupt vectors.

Note that to allow the use of the Monitor program during the development phase, the upper part of the Pram (addresses: $7C00-$7FFF) is reserved to the monitor code and cannot be used by any program.

Following the 32Kw PRAM, in the full addressing space of 64 Kw, there is a further 32 Kw area where the bootstrap EPROM is mapped (addresses: $8000-$FFFF).

The data memory is divided into two memory banks, X and Y which can be used to support parallel instructions: two machine instructions can be executed concurrently provided some conditions are met, namely the instructions use different ALU's and fetch/store arguments from/to different memory banks.

The full 32 Kw of X and Y memory space are available to user programs. Starting from address $8000 up to address $FFBF in the full 64 Kw X and Y addressing space there is an empty space (no memory available). In the area between addresses $FFC0 and $FFFF the internal (X memory) and external (Y memory) peripherals are mapped.

A complete list of the symbols used by the TNG_AO code can be found in Appendix B.

[5] If the code is stored in an EPROM it is copied into Pram prior of execution as part of the bootstrap process. The bootstrap process is described in some detail in Appendix A.

```
                X                  Y                       P
X_BUF +---------+ $0000 +---------+ Y_BUF     +---------+ $0000
      |         |       |         |           |         | Int. Vector
      |         |       |         |           +---------+ $0040
      |         |       |         |           |         |
      :  27904  :       :  27904  :           |         | Internal Ram
      :         :       :         :           |         |
      |         |       |         |           +---------+ $01FF
      |         |       |         |           |         |
  GEN +---------+ $6D00 +---------+ GEN       |         |
      |   256   |       |   256   |           +---------+ $0800
 I_EW +---------+ $6E00 +---------+ I_NS      |         |
      |   256   |       |   256   |           |   TNG   | P_ROM
 C_NS +---------+ $6F00 +---------+ C_EW      |         |
      |   256   |       |   256   |           +---------+ $1800
 O_EW +---------+ $7000 +---------+ O_NS      |         |
      |         |       |         |           :   TNG   : P_RAM
      :  4096   :       :  4096   :           |         |
      |         |       |         |           +---------+ $7C00
      |         |       |         |           | MONITOR |
      +---------+ $8000 +---------+           +=========+ $8000
      :  empty  :       :  empty  :           |  EPROM  |
      +---------+ $FFC0 +---------+           |         |
      | on chip |       |external |           |Ex. Load.| $C000
      |peripher.|       |peripher.|           |         |
      +---------+       +---------+           +---------+ $FFFF
```

Figure 3: TNG_AO Memory Map

5.1 P Memory Usage

The Pram used by TNG_AO software is globally divided into two areas: an area ($0800-$17FF) dedicated to the code, and therefore called P_ROM to stress the fact that it must not be written into; and an area for data variables ($1800-$7BFF) consequently named P_RAM. The following fraction of code defines the usage of the P_RAM area.

Source file: ao_mem.asm

```
        ORG     P:P_RAM                 ; TNG_AO P_Ram area
```

Here follow a few locations to be used as pointers or temporary storage. Note that this area is defined so that it is cleared at startup.

Source file: ao_mem.asm

```
P_CLNAREA                   ; Start of P memory area to clean
                            ;
```

```
CMB_PTR DS      1       ; Current VME Communication block pointer
ACC_PTR DS      1       ; Current Accumulation Block Pointer

SAV_Y1  DS      1       ; Store Y1 when entering timer interrupt

P_RAM_END
```

5.2 X and Y Memory Usage

Due to some architectural characteristics of the DSP 56001 the memory areas into X and Y ram must fulfill a number of alignment constraints.

- Two buffers used at the same time, if aligned, can be addressed by means of a single register.
- The DSP 56001 addressing logic provides the hardware support of circular buffers of any length. The buffer itself must anyway start at a memory addres which is a multiple of a power of two greather or equal than the buffer length (see section 1.3.2).

Due to the above constraints both the X and Y memories start with a big space reserved for two circular buffers used to store the output of the four counters and the filter outputs from the two channels, continuously during the loop. This allows the host to retrieve a fair amount of the “memory” of the system (see the related host function: DWNLOAD). The length of the buffers is set so that it is the biggest which leaves space for a number of smaller buffers at the end, as explained below. The buffer start at address $0 because it allows to use them as circular buffers.

Source file: ao_mem.asm

```
        ORG     X:X_RAM ; TNG_AO Serial Version X_Ram area
X_BUF   DSM     27904   ; Circular Buffer for input channel 1 and 2
                        ; This lenght MUST not be changed because uses the
X_BUF_END               ; entire free memory
```

Following the big buffers two aligned generic areas 256 word long, (GEN) used for general purpose variables are defined. Note that only the first 19 words of this area must be aligned in the two memory banks; the rest can be used at will.

Then two 256 word circular buffers for each memory bank are allocated; they are used for the FIR filters implementation. Note that the coefficient storage area and the corresponding samples storage area for each filter are allocated in different memory banks to allow the implementation of the filter loop by means of the DSP parallel instructions.

At the end of both X and Y memories are allocated two circular buffers 4096 word long mirror displacement values (two channels), preset by the host, are stored to be used to drive the mirror (see the related operation mode: MRDRIVE).

```
        ORG     X:BUF_ORG

X_VARS

SAVE_AX DS       1      ; Area to Save value of A0 on Interrupt

                        ;-----------------------------------------------
                        ; Variables, Flags and other stuff
                        ; The following Communication block areas MUST BE
                        ; aligned with the corresponding in Y memory !!
                        ; Note: the length of communication block MUST
                        ;       be the same as defined in the VME host
                        ;       source code !!
CM1_BLX DS      9       ; VME Comunication Block 1 storage.
CM2_BLX DS      9       ; VME Comunication Block 2 storage.

                        ; END OF ALIGNED AREA
                        ;-----------------------------------------------

SEMFRS  DS      1       ; Semaphors for various synchronozations
                        ; (See assignement of single bits in ao_mequ.asm)

CMB_SQN DS      1       ; Communication block sequence number

LD_CHK  DS      1       ; Loading complete flag
                        ; Bit 0: Parameters
                        ; Bit 1: Coefficients
                        ; Bit 2: Samples

LMODE   DS      1       ; Loop mode flag. It is used to select between
                        ; closed loop (mirror input=filter output) and
                        ; open loop/mirror drive (mirror input=from table)


N_COE   DS      1       ; Coefficent number storage
N_SAM   DS      1       ; Output samples number storage
T_FREQ  DS      1       ; Timer setting
N_AVE   DS      1       ; Accumulation number
MAXCYCL DS      1       ; Maximum number of cycles
N_SHIFT DS      1       ; Number of bit shifts for counters

CN1_EQL DS      1       ; Counter 1 equalization coefficient
CN2_EQL DS      1       ; Counter 2 equalization coefficient
CN3_EQL DS      1       ; Counter 3 equalization coefficient
CN4_EQL DS      1       ; Counter 4 equalization coefficient

MRFC_NS DS      1       ; Mirror NS command scaling factor
MROF_NS DS      1       ; Mirror NS command scaling offset
MRFC_EW DS      1       ; Mirror EW command scaling factor
MROF_EW DS      1       ; Mirror EW command scaling offset
```

```
INIT_NS DS      1       ; Preset value for filter memory
INIT_EW DS      1       ; Preset value for filter memory

N_CUR   DS      1       ; Current number of cycles

                                ;---------------------------------------------
                                ; The following variables are used as
                                ; temporaries by the Timer interrupt routine
CN1_TMP DS      1               ; Temporary storage for CNT1
CN3_TMP DS      1               ; Temporary storage for CNT3

C14_TMP DS      1               ; Temporary for CN1+CN4
C12_TMP DS      1               ; Temporary for CN1+CN2

        ORG     X:DUMMY ;-----------------------------------------------------
                        ; CIRCULAR BUFFERS
Z_BUFX  DSM     2       ; Dummy input buffer (preset to 0)
I_EW    DSM     MAXFILT ; Circular Buffer Mod MAXFILT for input channels E-W
C_NS    DSM     MAXFILT ; Circular Buffer Mod MAXFILT of filter coefficients N-S
O_NS    DSM     4096    ; Circular Buffer Mod 4096 for output channels N-S

X_RAM_END
```

Here follows the definition of the Y-memory map. Again we stress the fact that a number of alignement criteria between X and Y memories must be fullfilled.

Source file: ao_mem.asm

```
        ORG     Y:Y_RAM ; TNG_AO Serial Version Y_Ram area
Y_BUF   DSM     27904   ; Circular Buffer for input channel 1 and 2
                        ; This lenght MUST not be changed because uses the
                        ; entire free memory

        ORG     Y:BUF_ORG
Y_BUF_END

Y_VARS
                        ;-------------------------------------------------
                        ; Variables, Flags and other stuff
                        ; The following Communication block areas MUST BE
                        ; aligned with the corresponding in X memory !!
SAVE_AY DS      1       ; Area to Save value of A0 on Interrupt
CM1_BLY DS      9       ; VME Comunication Block 1 storage.
CM2_BLY DS      9       ; VME Comunication Block 2 storage.

                        ; END OF ALIGNED AREA
                        ;-------------------------------------------------

                        ;=========================================
                        ; The following variables are used as
```

```
                          ; temporaries by the Timer interrupt routine
CSUMTMP DS      1         ; Temporary storage for CN1+CN2+CN3+CN4
CN2_TMP DS      1         ; Temporary storage for CNT2
CN4_TMP DS      1         ; Temporary storage for CNT4
C34_TMP DS      1         ; Temporary for CN3+CN4
C23_TMP DS      1         ; Temporary for CN3+CN2

CNRESET DS      1         ; Counters reset word
CNT1SEL DS      1         ; CN1 Selection word
CNT2SEL DS      1         ; CN2 Selection word
CNT3SEL DS      1         ; CN3 Selection word
CNT4SEL DS      1         ; CN4 Selection word

CUR_AVE DS      1         ; Current accumulation number

LAST_ER DS      1         ; Storage for last error code

        ORG     Y:DUMMY ;-----------------------------------------------------
                          ; CIRCULAR BUFFERS
Z_BUFY  DSM     2         ; Dummy input buffer (preset to 0)
I_NS    DSM     MAXFILT ; Circular Buffer Mod MAXFILT for input channels N-S
C_EW    DSM     MAXFILT ; Circular Buffer Mod MAXFILT of filter coefficients E-W
O_EW    DSM     4096      ; Circular Buffer Mod 4096 for output channels E-W

Y_RAM_END
```

6 Initialization

Here starts the TNG_AO executable code. After the boot process (or manually directed by the monitor command, during the development phase) the program control is passed to the address `P_ROM` to begin the execution.

The following code is used at the startup of the system to initialize both the DSP to the required status and a number of variables and buffers.

Note that some parts of the code are assembled coditionally depending on the version of the code: the "monitor" version (which is configured so that it can be executed under the control of the DBV56H on board monitor) sometimes requires slightly different setting than the final "eprom" version. The condition symbol is called `EPROM_VER` and must be defined to generate the software version suited to be burnt into EPROM.

6.1 DSP Status Initialization

At the beginning of the P_ROM area is stored an identification string which can be sent to the host on request in order to identify the software revision. Note that the banner is always just following the main entry point so that is it easily retrievable in the memory map.

Then some locations are reserved for costants used throughout the code.

The following fraction of code is needed to initialize and preset the global DSP status: internal pram, peripherals, interrupt registers and vectors, bus control register, B port and, only for the EPROM version, the C port.

```
        ORG     P:P_ROM             ; TNG_AO CODE STARTING ADDRESS
        JMP     EXEC                ; Jump to begin of executable code

                                    ;------------------------------------------
                                    ; AREA FOR CONSTANTS
                                    ;------------------------------------------

                                    ;       B A N N E R
                                    ; DO NOT change the length of Banner !!!
BANNER                              ; The banner is exactly 19 words long
        IF EPROM_VER
        DC      'TNG Tip-Tilt / EV 1.12 Aug 1996 / L. Fini, P. Ranfagni',$0

        ELSE
        DC      'TNG Tip-Tilt / MV 1.12 Aug 1996 / L. Fini, P. Ranfagni',$0
        ENDIF

;
BANLEN   EQU  *-BANNER              ; Length of Banner in words

CNST_1  DC      1                   ; Constant 1
CNST_2  DC      2                   ; Constant 2
CNST_FF DC      $FF                 ; Constant $FF
CNST_M1 DC      -1                  ; Constant -1
CNST_10 DC      10                  ; Constant decimal 10
CNST_15 DC      15                  ; Constant decimal 15
CNST_20 DC      20                  ; Constant decimal 20
CNST_25 DC      25                  ; Constant decimal 25
MASK_16 DC      $ffff               ; 16 bits mask
MAXFRAC DC      $7fffff             ; Max positive fraction


EXEC                                ;----------------------------------------
        IF      EPROM_VER           ; Code not necessary in Monitor version
        MOVE    #0,X0               ; Bootstrap firmware loads boot extended
        MOVE    X0,R0               ; loader code in the first 512 words of
        DO      #$200,MCLEAN        ; internal PRAM, we purge it to free this
        MOVEM   X0,P:(R0)+          ; memory. Don't use in monitor version.
MCLEAN
        RESET                       ; reset on chip peripherals (belt and braces)
                                    ; don't use in Monitor version.

                                    ; External interrupts and sample clock.
        MOVEP   #0,Y:SAMP           ; DEFAULT SAMPLE CLOCK
        MOVEP   #0,Y:XIRQA          ; NO EXTERNAL IRQA
        MOVEP   #0,Y:XIRQB          ; NO EXTERNAL IRQB
        MOVEP   #0,Y:INVERT         ; NO EXTERNAL IRQ INVERSION

        ENDIF                       ;--------------------------------------------
```

```
                        ; CONFIGURE WAIT STATE PARAMETERS FOR MEMORY
                        ; AND I/O
        MOVEP   #WAITS,X:BCR    ; load wait state control value to BCR

                        ; LOAD HOST COMMAND INTERRUPT VECTORING CODE
                        ; INTO LOW RAM LEAVING MONITOR RELATED
                        ; INTERRUPTS UNCHANGED
        MOVE    #VSTART,R1      ; point R1 at external load address
        MOVE    #HOSTVEC,R0     ; point R0 at internal Ram destination
        DO      #VEND-VSTART,VL ; set up loop for transfer
        MOVE    P:(R1)+,X0      ; load a word
        MOVE    X0,P:(R0)+      ; and transfer it into low Ram
VL

;       IF EPROM_VER            ; The Port C is configured by the Monitor
;       JSR     PC_CFG          ; in any case.
;       ENDIF

        JSR     CLR_OUT         ; Clear Output registers

                        ; configure Port B for Host Interface
        MOVEP   #1,X:PBC        ; turn on host interface
        BSET    #HCIE,X:HCR     ; enable host command interrupts
```

6.2 TNG_AO Variables and Buffers Initialization

In the following lines of code we clear the memory areas used by the TNG_AO software and initialize various variables and pointers.

Source file: **ao_init.asm**

```
                        ; Clear P_RAM
        MOVE    #0,x0
        MOVE    #P_CLNAREA,R0
        DO      #P_RAM_END-P_CLNAREA,PCLEAN
        MOVE    X0,P:(R0)+
PCLEAN

        MOVE    #CM1_BLX,X0
        MOVE    X0,P:CMB_PTR    ; Initialize Commun. Block pointer
        MOVE    #CM2_BLX,X0
        MOVE    X0,P:ACC_PTR    ; Initialize Accumul. Block pointer

                        ; Clear X_RAM
        MOVE    #0,X0
        MOVE    #X_RAM_END-X_RAM,R1
        MOVE    #X_RAM,R0
        DO      R1,XCLEAN
        MOVE    X0,X:(R0)+
XCLEAN
```

```
                              ; Clear Y_RAM
        MOVE    #0,X0
        MOVE    #Y_RAM_END-Y_RAM,R1
        MOVE    #Y_RAM,R0
        NOP
        DO      R1,YCLEAN
        MOVE    X0,Y:(R0)+
YCLEAN
        NOP

                              ; INITIALIZE COUNTER SELECTION WORDS
        MOVE    #$3F,Y0
        MOVE    Y0,Y:CNRESET  ; NB: It's left Justified (==$3F0000)
        MOVE    #$3B,Y0
        MOVE    Y0,Y:CNT1SEL  ; NB: It's left Justified (==$3B0000)
        MOVE    #$37,Y0
        MOVE    Y0,Y:CNT2SEL  ; NB: It's left Justified (==$370000)
        MOVE    #$2F,Y0
        MOVE    Y0,Y:CNT3SEL  ; NB: It's left Justified (==$2F0000)
        MOVE    #$1F,Y0
        MOVE    Y0,Y:CNT4SEL  ; NB: It's left Justified (==$1F0000)

        MOVE    #$ffff,M6     ; Preset safe modulo for input buffers
        MOVE    #$ff,M2       ; Preset safe modulo for input buffers

        BCLR    #HF3,X:<<HCR  ; Clear error flag


                              ;-----------------------------------------
        MOVEP   #PIC,X:<<IPR  ; Set Interrupt priorities

        MOVE    #0,SR         ; Clear status register (including process
                              ; priority)

        MOVE    #$7C56,R0
        MOVE    #PIC,X0       ; Trucco per abilitare gli Host command
        MOVE    X0,P:(R0)
```

At the end of initialization we jump to a common point where an idle loop is used to wait for an interrupt due to a command issued by the host.

Source file: ao_init.asm

```
        JMP     WAIT_ST       ; Wait for Host Command
```

6.3 Utility Subroutines

The following lines of code define some utility subroutines used in the initialization procedure and somewhere else.

Source file: ao_init.asm

```
                            ;  Sub to configure Port C for Serial Comm.
                            ;  (not necessary in Monitor version)
PC_CFG
        MOVEP   #$0302,X:SCR    ;set 1-start 8-data 1-stop TX,RX active,Int off

        MOVEP   #CLKDIV,X:SCCR ;load clock control for 9600baud,
                               ; internal clk @ 27MHz
        MOVEP   #3,X:PCC        ;configure Port C bits 0,1 for RX and TX
        MOVEP   #7,X:PCDDR      ;set direction for RX,TX and CLK.
        RTS

                            ;-----------------------------------------
                            ; Sub to clear the output port (Now we put
                            ; a 0, maybe something different will be
                            ; needed in the future
CLR_OUT
        MOVE    #0,X0
        MOVEP   X0,Y:PORTOUT
        RTS
```

7 Host Command Processing

The host controls the DSP by means of commands issued through the host port. The host writes a byte to the Command Vector Register of the DSP-VME interface which fires one out of fourteen interrupts in order to transfer the control to the command processing code. This mechanism allows to implement up to fourteen different "host commands".

Each host command stops the loop currently active (if any) and disables the interrupt until the command processing is finished, so that no command is accepted until the previous one is completed.

The TNG_AO software defines nine commands: STRTMON, LDPARAM, LDCOEFF, LDSAMPL, STPLOOP, SNDIDNT, DWNLBUF, WRERROR, OPNLOOP, CLSLOOP, MRDRIVE, as described in the following sections.

For each command a suitable interface routine has been defined in the host side interface library [3].

7.1 Control Commands

7.1.1 STRTMON: Start Monitor Code

This command forces the control to go to the beginning of the Monitor code. It can be used in order to allow final debugging of the TNG_AO code or to test new versions of the code on the working system. The corresponding routine in the host library is `StartMon()` [3].

Source file: ao_com.asm

```
STRTMON                         ; START THE MONITOR
        DISINTS                 ; Disable all interrupts
        BCLR    #HF2,X:<<HCR    ; Signal to Host that DSP is busy
```

```
        BCLR    #HF3,X:<<HCR    ; Clear Error flag
        JSR     STOP            ; Stop everything
        JMP     MONITOR         ; Jump to monitor entry point
```

7.1.2 LDPARAM: Load Loop Parameters

Following this command the DSP waits for the host to send the set of parameters which define the characteristics of the loop (e.g.: integration time, filter length, etc.). The corresponding routine defined from the host side is `LDparam()` [3].

Source file: **ao_com.asm**

```
LDPARAM                         ; LOAD GENERAL PARAMETERS
        DISINTS                 ; Disable all interrupts
        BCLR    #HF2,X:<<HCR    ; Signal to Host that DSP is busy
        BCLR    #HF3,X:<<HCR    ; Clear Error flag
        JSR     STOP            ; Stop everything

        READW   X:T_FREQ        ; Read Internal timer setting
        READW   X:MAXCYCL       ; Read Maximum number of cycles
        READW   X:N_AVE         ; Read Number of averaging cycles

        READW   X:CN1_EQL       ; Read Counter 1 equalization coeff.
        READW   X:CN2_EQL       ; Read Counter 2 equalization coeff.
        READW   X:CN3_EQL       ; Read Counter 3 equalization coeff.
        READW   X:CN4_EQL       ; Read Counter 4 equalization coeff.

        READW   X:MROF_NS       ; Read Mirror NS command scaling offset
        READW   X:MRFC_NS       ; Read Mirror NS command scaling factor
        READW   X:MROF_EW       ; Read Mirror EW command scaling offset
        READW   X:MRFC_EW       ; Read Mirror EW command scaling factor

        READW   X:INIT_NS       ; Read NS initialization value
        READW   X:INIT_EW       ; Read EW initialization value

        BSET    #0,X:LD_CHK     ; Notify that loading has been done
        JMP     WAIT_ST         ; Return to wait state
```

7.1.3 LDCOEFF: Load Filter Coefficients

Following this command the DSP waits for the host to send the values of filters coefficients. Two copies of the filter coefficient vector are stored into memory (one in X memory and the other in Y memory, for the two *tip-tilt* channels) to allow the filter code to use the DSP 56001 parallel instructions. The corresponding routine defined from the host side is `LDcoeff()` [3].

Source file: **ao_com.asm**

```
LDCOEFF                         ; LOAD FILTER COEFFICIENTS
        DISINTS                 ; Disable all interrupts
```

```
        BCLR    #HF2,X:<<HCR     ; Signal to Host that DSP is busy
        BCLR    #HF3,X:<<HCR     ; Clear Error flag
        JSR     STOP             ; Stop everything

        MOVE    #N_COE,R0        ; Point to coeff number location
        READW   X:(R0)           ; Read number of coefficients
        MOVE    #C_NS,R0         ; Set coefficient bank initial address
        MOVE    X:N_COE,R1       ; Get Number of coefficients

        DO      R1,LCOE_E        ; Loop on coefficients
        JCLR    #HRDF,X:HSR,*    ; Wait for data register full
        MOVEP   X:HRX,X0         ; Get datum from host
        MOVE    X0,X:(R0)        ; Store into X_mem coeff. buffer
        MOVE    X0,Y:(R0)+       ; Store into Y_mem coeff. buffer
LCOE_E
        BSET    #1,X:LD_CHK      ; Notify that loading has been done
        JMP     WAIT_ST          ; Return to wait state
```

7.1.4 LDSAMPL : Load Mirror Drive Samples

Following this command the DSP waits for the host to send two vectors of values which will be used to generate a signal to drive the mirror (see also command MRDRIVE). The corresponding routine defined from the host side is `LDsampl()` [3].

Source file: ao_com.asm

```
LDSAMPL                          ; LOAD MIRROR DRIVE SAMPLES
        DISINTS                  ; Disable all interrupts
        BCLR    #HF2,X:<<HCR     ; Signal to Host that DSP is busy
        BCLR    #HF3,X:<<HCR     ; Clear Error flag
        JSR     STOP             ; Stop everything

        MOVE    #N_SAM,R0        ; Point to samples number location
        READW   X:(R0)           ; Read number of samples

                                 ; First: N-S samples
        MOVE    #O_NS,R1         ; Set samples bank initial address
        DO      X:(R0),LSNS_E    ; Loop on samples
        READW   Y:(R1)+
LSNS_E
                                 ; Then: E-W samples
        MOVE    #O_EW,R1         ; Set sample bank initial address
        DO      X:(R0),LSEW_E    ; Loop on samples
        READW   X:(R1)+
LSEW_E
        BSET    #2,X:LD_CHK      ; Notify that loading has been done
        JMP     WAIT_ST          ; Return to wait state
```

7.1.5 STPLOOP: Stop the Loop

This command interrupts any active loop mode and returns the DSP to the waiting state. The correspondiong routine in the host library is `StopLoop()` [3].

Source file: ao_com.asm

```
STPLOOP                         ; STOP THE LOOP
        DISINTS                 ; Disable all interrupts
        BCLR    #HF2,X:<<HCR    ; Signal to Host that DSP is busy
        BCLR    #HF3,X:<<HCR    ; Clear Error flag
        JSR     STOP            ; Stop everything

WAIT_ST MOVE    #0,SP           ; Clear stack pointer to clean pending
                                ; interrupts, do loops, etc
        MOVE    #0,SR
        BSET    #HCIE,X:HCR     ; Reenable Host commands
        BSET    #HF2,X:<<HCR    ; Signal to Host that DSP is  ready

        JMP     *               ; Wait next command
```

7.2 Data Access Commands

The commands listed in this section are used by the host to prompt for DSP data to be transmitted through the Host interface. Data transfers obtained by this functions are not to be mistaken with the Communication Block data periodically sent by the DSP when any loop mode is active.

7.2.1 SNDIDNT: Send Back Identification String (Banner)

After this command the DSP transmits to the host the identification string (see `BANNER` in section 5). The corresponding routine in the host library is `DSPident()` [3]

Source file: ao_com.asm

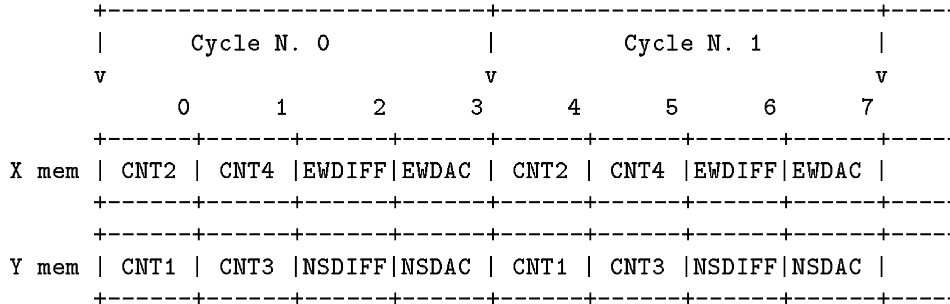
```
SNDIDNT                         ; DOWNLOAD BANNER TO HOST PORT
        DISINTS                 ; Disable all interrupts
        BCLR    #HF2,X:<<HCR    ; Signal to Host that DSP is busy
        BCLR    #HF3,X:<<HCR    ; Clear Error flag
                                ; Load start address
        MOVE    #BANNER,R0      ; save start address in pointer reg R0
                                ; Load download count
        MOVE    #BANLEN,N0      ; load count into reg N0
                                ; Download Program Memory
        DO      N0,BANEND       ; loop through data downloading until complete
        JCLR    #HTDE,X:HSR,*   ; Wait for register empty
        MOVEP   P:(R0)+,X:HTX   ; Write the word
BANEND

;  Wait for host command

        JMP     WAIT_ST
```

7.2.2 DWNLBUF: Download the Data from the Run-Time Data Buffer

During the loop operation the DSP accumulates data into two circular buffers (see `X_BUF` and `Y_BUF` in section 5). This command is used to transfer the content of the buffers to the host. After this command the DSP starts to transmit to the host the required data.



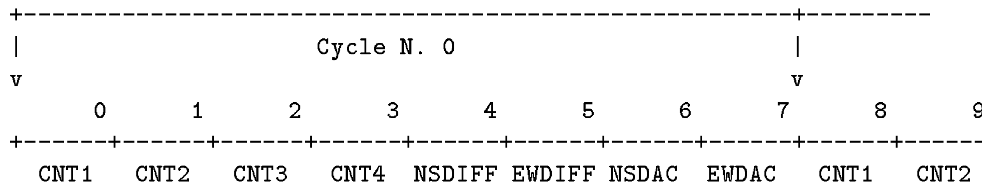
Where the values are defined as follows:

- `CNTx` The double of the value of counter `x` (we store the double of the count value because this simplifies slightly the code).
- `NSDIFF`, `EWDIFF` the normalized and rescaled values of the computed *tip-tilt* components (as a result of various rescaling operations, the computed values are actually one half of the *tip-tilt* components as explained in section 9.1). These are the values sent as inputs to the filters.
- `NSDAC`, `EWDAC` the outputs of the filters, properly scaled and offset to be fed to the Steering Mirror DAC's.

Figure 4: Loop data storage organization

Data are grouped in blocks of eight words, each related to one cycle of the loop. During the step processing data are stored into Y and X memory alternately as shown in figure 4. Note that when transmitting the stored data to the host, the two memories are addressed alternately, so that the resulting order of values in the transmitter data block is as shown in figure 5.

The corresponding routine in the host library is `DSPdownl()` [3]



Where the symbols have the same meaning as in figure 4.

Figure 5: Loop data transmission order

Source file: ao_com.asm

```
DWNLBUF                         ; Download internal buffer
        DISINTS                 ; Disable all interrupts
        BCLR    #HF2,X:<<HCR    ; Signal to Host that DSP is busy
        BCLR    #HF3,X:<<HCR    ; Clear Error flag
        JSR     STOP            ; Stop everything

        MOVE    #X_BUF_END-X_BUF,X0     ; preset buffer length
        JCLR    #HTDE,X:HSR,*   ; Wait for register empty
        MOVEP   X0,X:HTX        ; Write the word

        DO      X0,END_BUF
        JCLR    #HTDE,X:HSR,*   ; Wait for register empty
        MOVEP   Y:(R3),X:HTX    ; Write the word
        JCLR    #HTDE,X:HSR,*   ; Wait for register empty
        MOVEP   X:(R3)+,X:HTX   ; Write the word
END_BUF
        JMP     WAIT_ST         ; Wait next command
```

7.2.3 WRERROR: Write Last Error Code to Host Interface

This function can be used by the host to get the last error code. The host can know at any time that an error has been detected by the DSP by sensing the HF3 bit. After the command the DSP puts the error code onto the port then clears the error flag (HF3), so that the reading process can synchronize on it.

The corresponding routine in the host library is `DSPcheck()` [3].

Source file: ao_com.asm

```
WRERROR
        DISINTS                 ; Disable all interrupts
        BCLR    #HF2,X:<<HCR    ; Signal to Host that DSP is busy

        BCLR    #LOO_SEM,X:SEMFRS       ; Clear loop active flag

        IF TIMINT
        JSR     PC_CFG                  ; Reconfigure port C (and stop timer)
        ENDIF

        JCLR    #HF1,X:<<HSR,*  ; Wait for Flag from host

        MOVE    Y:LAST_ER,X0    ; Get last error code
        MOVEP   X0,X:HTX        ; Output onto port
        NOP
        BCLR    #HF3,X:<<HCR    ; Clear Error flag

        JMP     WAIT_ST         ; Wait next command
```

7.3 Loop Initialization Commands

The *tip-tilt* loop operation has three different modes: a) the **Open Loop** mode is essentially used to read the Quad Cell input while maintaining the mirror in a fixed position in order to provide the Host with data useful for the preset of the system prior of switching to Close Loop mode; b) the **Closed Loop** mode is used when correcting the *tip-tilt*; c) the **Mirror Drive** mode is used to drive the mirror with a known signal for various testing purposes.

The three modes are implemented essentially with the same code. The loop code may be thought of as a pipeline of three stages: 1) *Tip-tilt* components evaluation (see detailed description in section 9), 2) filtering (see description in section 9.2), 3) mirror control (see description in section 4.6), as shown in figure 6. Note that the figure only shows one of the two channels.

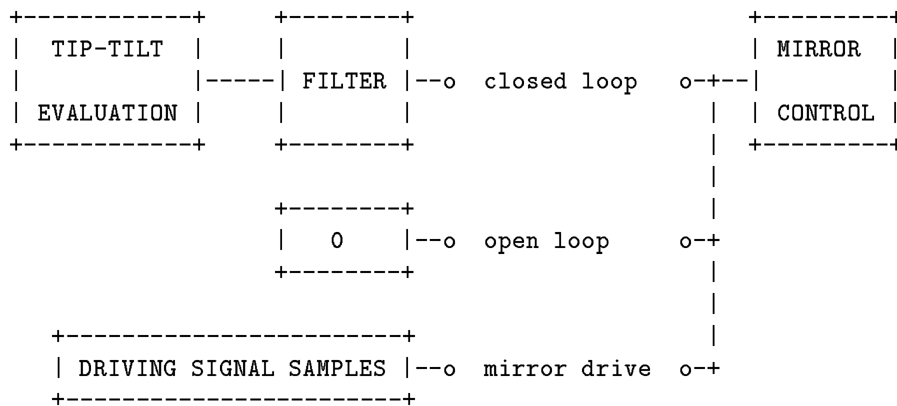


Figure 6: Loop Code Structure (one channel)

The three loop modes can thus be selected simply by choosing the right inputs to the mirror control code. In Closed Loop mode the inputs are taken from the outputs of the filters which are stored in the big circular buffers as described in section 7.2.2, in Open Loop mode they are forced to zero, and in mirror drive they are taken from two circular buffers of data samples preloaded with the suitable Host Command (see command LDSAMPL). The three modes thus only differ for the initialization code.

The corresponding routine in the host library is `StartLoop()` [3], called with the proper argument to select the loop mode.

7.3.1 OPNLOOP: Start the "Open Loop Mode"

In open loop mode the mirror input must be zero (The mirror actual position may be arbitrarily chosen by means of the offset values used in the mirror position computation). The code presets register pointers `R5`, `M5`, `N5` so that they point to a two components circular buffer which has been preset to zero.

Source file: ao_com.asm

```
OPNLOOP                          ; INITIALIZE OPEN LOOP MODE
```

```
        DISINTS                 ; Disable all interrupts
        BCLR    #HF2,X:<<HCR    ; Signal to Host that DSP is busy
        BCLR    #HF3,X:<<HCR    ; Clear Error flag
        BCLR    #OPEN,X:LMODE   ; Reset open loop flag

SETLOOP
        JSR     STOP            ; Stop everything

                                ; Register R5,M5,N5 are used for addressing
                                ; the input value to mirror.
                                ; In open loop the input for mirror control
                                ; is taken from a dummy circular buffer
                                ; of length two initialized to zero.
        MOVE    #Z_BUFX,R5
        MOVE    #1,M5           ; preset modulo register
        MOVE    #1,N5           ; Preset postdecrement register
        JMP     LP_COMMON       ; Go to loop common code
```

7.3.2 CLSLOOP: Start the "Closed Loop Mode"

In closed loop mode the input to the mirror control is taken from the output of the two filters. Register pointers R5, M5, and N5 are preset to point to the two aligned memory buffers where the output of the filter is stored.

Source file: ao_com.asm

```
CLSLOOP                         ; INITIALIZE CLOSED LOOP MODE
        DISINTS                 ; Disable all interrupts
        BCLR    #HF2,X:<<HCR    ; Signal to Host that DSP is busy
        BCLR    #HF3,X:<<HCR    ; Clear Error flag
        BCLR    #OPEN,X:LMODE   ; Reset open loop flag

        JSR     STOP            ; Stop everything

                                ;-----------------------------------------
                                ; Registers R5,M5,N5 are used for
                                ; addressing the input value to mirror
        MOVE    #X_BUF+3,R5
        MOVE    #X_BUF_END-X_BUF-1,M5   ; preset modulo register
        MOVE    #4,N5           ; Preset postdecrement register
        JMP     LP_COMMON       ; Go to loop common code
```

7.3.3 MRDRIVE: Start the "Mirror Drive Mode"

In mirror drive mode the input to the mirror control is taken from two circular buffers which have been preloaded with the driving signal samples.

Source file: ao_com.asm

```
MRDRIVE                         ; INITIALIZE MIRROR DRIVE LOOP MODE
        DISINTS                 ; Disable all interrupts
        BCLR    #HF2,X:<<HCR    ; Signal to Host that DSP is busy
        BCLR    #HF3,X:<<HCR    ; Clear Error flag
        BSET    #OPEN,X:LMODE   ; Signal Open loop mode
        JSR     STOP            ; Stop everything

                                ; set R5,M5,N5 to get input from samples table
        MOVE    X:N_SAM,R5
        NOP
        MOVE    (R5)-
        MOVE    R5,M5           ; Set modulo register

        MOVE    #O_NS,R5
        MOVE    #1,N5

        JMP     LP_COMMON
```

7.4 Loop Initialization Common Code

The following fragment of code is common to the three loop modes and completes the operations needed to prepare for loop activation.

Source file: ao_com.asm

```
LP_COMMON
        MOVE    #0,X0           ; Clear X buffer
        MOVE    #X_BUF_END-X_BUF,R1
        MOVE    #X_RAM,R0
        DO      R1,XCL1
        MOVE    X0,X:(R0)+
XCL1
        MOVE    #0,X0           ; Clear Y buffer
        MOVE    #Y_BUF_END-Y_BUF,R1
        MOVE    #Y_RAM,R0
        DO      R1,YCL1
        MOVE    X0,Y:(R0)+
YCL1
        MOVE    X:INIT_EW,X0    ; Preset filter memory (EW)
        MOVE    #MAXFILT,R1
        MOVE    #I_EW,R0
        DO      R1,PSET_X
        MOVE    X0,X:(R0)+
PSET_X

        MOVE    X:INIT_NS,X0    ; Preset filter memory (NS)
        MOVE    #MAXFILT,R1
        MOVE    #I_NS,R0
        DO      R1,PSET_Y
        MOVE    X0,Y:(R0)+
PSET_Y
```

```
        MOVE    #CM1_BLX,R0
        MOVE    R0,P:CMB_PTR    ; Preset CMB pointer
        MOVE    R0,P:ACC_PTR    ;
        CLRACC                  ; Clear Block
        MOVE    #CM2_BLX,R0
        MOVE    R0,P:ACC_PTR    ; Preset ACC pointer
        CLRACC                  ; Clear Block
        MOVE    #0,X0
        MOVE    X0,Y:CUR_AVE    ; Clear current accumulation number
        MOVE    X0,X:CMB_SQN    ; Clear Comm.Block sequence number

        MOVE    #0,R0
        MOVE    #LD_CHK,R1
        MOVE    R0,X:N_CUR      ; Clear current number of cycles
        JSET    #0,X:(R1),OK_OL ; Test if parameters have been loaded
        WRERR   #E_NOPAR        ; Send back error code
        JMP     WAIT_ST         ; return to wait state

OK_OL
                                ; ----------------------------------------
                                ; Registers R2,M2,N2 are used for input
                                ; buffer addressing and cannot be used
                                ; otherwise in timer interrupt routine
        MOVE    X:N_COE,R2      ; Preset Filter length
        NOP
        MOVE    (R2)-
        MOVE    R2,M2           ; Preset modulo register for samples
        MOVE    R2,M6           ; Preset modulo register for coefficients

        MOVE    #I_EW,R2        ; Preset input buffer pointer

                                ;-----------------------------------------
                                ; Registers R3,M3,N3 are used for input
                                ; long storage addressing
        MOVE    #X_BUF,R3
        MOVE    #X_BUF_END-X_BUF-1,M3   ; preset modulo register
        MOVE    #0,N3

                                ; Reset counters
        MOVE    Y:CNRESET,X0
        MOVEP   X0,Y:PORTOUT    ; First bank
        BCLR    #RES_CNT,X0
        MOVEP   X0,Y:PORTOUT
        BSET    #RES_CNT,X0
        MOVEP   X0,Y:PORTOUT

        BCHG    #CNTSWTC,X0     ; Switch counter banks

        MOVEP   X0,Y:PORTOUT    ; Second bank
        BCLR    #RES_CNT,X0
        MOVEP   X0,Y:PORTOUT
```

```
        BSET    #RES_CNT,X0
        MOVEP   X0,Y:PORTOUT

        MOVE    #0,X0           ; Clear flags
        MOVE    X0,X:SEMFRS

        BCLR    #HF3,X:<<HCR    ; Clear error flag
        BSET    #HF2,X:<<HCR    ; Signal to Host that DSP is  ready
                                ; ----------------------------------------
        JSR     STIMR           ; Preset Interrupt timer and fire it

        BSET    #HCIE,X:HCR     ; reenable host commands to allow to issue
                                ; a "manual interrupt"

        JMP     MAIN            ; Go to main loop
```

7.5 Subroutines

The following subroutines are used by the command processing code.

7.5.1 STOP: Terminate the Current Loop

This subprogram orderly terminates a loop mode. It is called before processing any host command to disable the loop process.

Source file: ao_com.asm

```
STOP                                    ; TERMINATE THE LOOP
        BCLR    #LOO_SEM,X:SEMFRS       ; Tset if a loop is active
        JCC     WASNTLOOP               ; Verify that a loop is active

        WRERR   #E_STOPD                ; Signal that the loop has been stopped

        IF TIMINT                       ; (see description above)
        JSR     PC_CFG                  ; Reconfigure port C (and stop timer)
        ENDIF

WASNTLOOP
        RTS                             ;-------------------------------------
```

7.5.2 STIMR: Preset the Interrupt Timer

This subprogram is called to program and set the internal timer which fires the processing of a loop step.

Source file: ao_com.asm

```
STIMR                           ; PRESET AND FIRE THE TIMER
        IF TIMINT               ; Set the symbol TIMINT to 1 to actually set
                                ; the timer interrupt. During debug the timer
```

```
                        ; is not set so that port C can be used with
                        ; the monitor. The timer interrupt is simulated
                        ; by sending the proper host command from the
                        ; host

        MOVEP   #0,X:SCR        ; Sub: Clear SCR (disable port C)
        MOVEP   X:T_FREQ,X:SCCR ; Preset timer setting

        ENDIF                   ;------------------------------------------

        MOVE    #TINT_1,R1      ;point R1 at external load address
        MOVE    #TIMVEC,R0      ;point R0 at internal Ram destination
                                ; Set Interrupt address in interrupt table
        DO      #2,TL           ;set up loop for transfer
        MOVE    P:(R1)+,X0      ;load a word
        MOVE    X0,P:(R0)+      ;and transfer it into low Ram
TL

        BSET    #LOO_SEM,X:SEMFRS       ; Signal loop active
        BCLR    #TMR_SEM,X:SEMFRS       ; Clear Timer interrupt flag

        IF TIMINT               ; See above

        BSET    #TMIE,X:SCR     ; Fire the timer !!

        ENDIF

        RTS
```

8 Main Loop

After the initialization the actual loop code is fired. The main loop provides for the periodic transmission of a block of data to the VME host. This task is periodically interrupted by the interrupt timer to process the next datum from the APD readout.

8.1 The VME Communication Block

The communication block (CMB) contains data gathered during the loop operation and periodically transmitted to the host. In order to limit the throughput on the communication channel to the VME, the data are accumulated for a given number of loop steps (`N_AVE`) and thus transmitted at a rate which is lower than the loop sampling rate.

The CMB is actually implemented by a double buffer so that the one which is ready for transmission (and consequently referred to as "Communication Block") can be sent while data are accumulated into the other (sometimes called "Accumulation Block"). The CMB consists of nine words defined as follows:

1. `STATUS`. A status word which contains the following information: a sequence number which is incremented every time a block is sent to the host, some flags and an error code (see detailed description in figure 2).

2. C1_SUM. The sum of N_AVE subsequent counts from counter C1.

3. C2_SUM. The sum of N_AVE subsequent counts from counter C2.

4. C3_SUM. The sum of N_AVE subsequent counts from counter C3.

5. C4_SUM. The sum of N_AVE subsequent counts from counter C4.

6. NS_SUM. The sum of N_AVE subsequent values of the North-South tilt component.

7. EW_SUM. The sum of N_AVE subsequent values of the East-West tilt component.

8. NS_CORR. The sum of N_AVE subsequent values of the North-South correction component (actually the corresponding filter output).

9. EW_CORR. The sum of N_AVE subsequent values of the East-West correction component (actually the corresponding filter output).

The above pieces of information can be continuously monitored by the host in order to evaluate the status of the loop and the quality of the correction.

Source file: ao_com.asm

```
MAIN                                ; MAIN TIP-TILT LOOP.
                                    ; The Main Loop consists of continuous
                                    ; transmission of the ready Communication
                                    ; Block to the Host Interface.
                                    ; This loop is periodically interrupted by
                                    ; the high priority timer interrupt.
                                    ;-----------------------------------------

WAIT_F  BTST    #TRN_SEM,X:SEMFRS           ; Wait for Comm. block ready
        JCC     WAIT_F
                                            ; Increment CMB sequence counter
        MOVE    X:CMB_SQN,A                 ; Get the counter
        MOVE    P:CMB_PTR,R0                ; Get Accumulation Pointer

;               Load LSByte mask
        MOVE    P:CNST_FF,Y1

;               Mask LSByte     Load CMB_PTR
        AND     Y1,A            X:(R0),Y1

;               Incl.CMB_SQN
;               in status word
        OR      Y1,A

;               Save Status word
        MOVE    A1,X:(R0)

                                    ;-----------------------------------------
WRITEB                              ; Now transmit communication block to VME
                                    ; We use R0: it cannot be used in
```

```
                                ; Timer interrupt code

        DO      #CM2_BLX-CM1_BLX,WREND
        JCLR    #HTDE,X:HSR,*   ; Wait for register empty
        MOVEP   X:(R0)+,X:HTX   ; Write the word
WREND

        BCLR    #TRN_SEM,X:SEMFRS       ; Signal that CMB has been transmitted

        JMP     MAIN                    ; Return to Main loop begin
```

9 Timer Interrupt Servicing Routine

This fragment of code is invoked by the periodic timer interrupt to process the next loop step. The loop step processing is logically divided into three substeps: 1) *Tip-tilt* components evaluation, 2) filtering, 3) mirror control. In the following sections we will describe in detail each substep.

To optimize the call time we want to have a minimum number of registers to save onto stack, so we reserve some registers to timer routine which are not used in other parts of the code.

The timer interrupt servicing routine needs the exclusive use of the following registers:

```
Accumulators        : B
ALU Input registers : X,Y (Y1 is used elsewhere so it's saved)
Addressing registers: R2,R3,R7,R6
Offset registers    : N2,N3,N7,N6
Modifier registers  : M2,M3,M7,M6
```

The other parts of the software MUST NOT use these registers which maintain status information required by the algorithm.

9.1 *Tip-Tilt* Components Evaluation

When the interrupt is fired by the timer the control goes to the following section of code where the values accumulated during the sampling interval are read and the two *tip-tilt* components are evaluated.

In order to estabilish a reference we have named the two components "North-South" and "East-West", but the reference is arbitrary in that the system operations don't depend in any way on the actual orientation of the quad cell with respect to the sky. In figure 7 the relationship between APD's and corresponding *tip-tilt* components is shown.

At the beginning of the timer interrupt servicing code we firstly disable servicing of other interrupts (namely Host Commands); then we check that this interrupt didn't interrupt the processing of the previous timer interrupt, due to a value of the interval too low to allow the complete processing of the step. Then we check whether the Communication Buffer is ready to be sent (that is `N_AVE` sums have been accumulated), in this case we swap the Communication

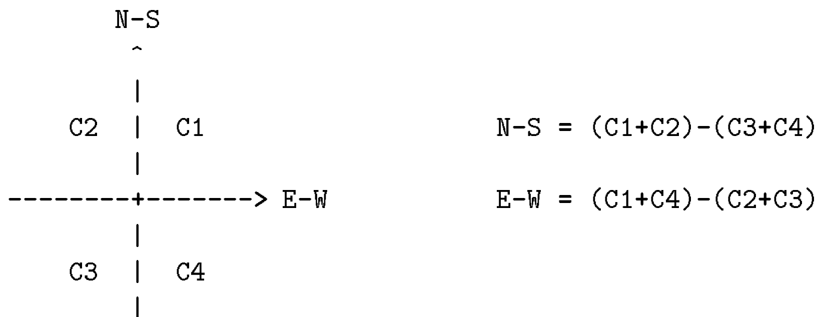


Figure 7: *Tip-tilt* Components Reference

Buffers. For the sake of clarity in the following pages we call "Accumulation Buffer" the buffer where sums are being accumulated and "Communication Buffer" the buffer being transmitted..

Source file: ao_int.asm

```
TIO_L
        BCLR    #HCIE,X:HCR               ; Disable host command while servicing
                                          ; the timer interrupt
        MOVE    Y1,P:SAV_Y1               ; Save Y1
        MOVE    A,L:SAVE_AX               ; Save A

        BSET    #TMR_SEM,X:SEMFRS         ; Test (and set) Timer interrupt flag
                                          ; to ensure that the interrupt from
        JCC     OK                        ; the timer  doesn't overlap a current
                                          ; timer cycle

        WRERR   #E_OVTIM                  ; Send error code

        JMP     WAIT_ST                   ; return to wait state

                                          ;-------------------------------------
OK                                        ; Test Communication buffer (and switch)
        MOVE    X:N_AVE,X0
        MOVE    Y:CUR_AVE,A
        MOVE    P:CNST_1,Y1               ; Preset a constant 1 for subsequent use

        .IF     A <GE> X0                 ; Test Current number of sums.

                                          ; Swap accumulation buffers
        BTST    #TRN_SEM,X:SEMFRS         ; Check that Transmission is finished
        JCS     E_TEST                    ; CMB transmission unfinished:
                                          ; do not swap
        MOVE    P:ACC_PTR,X1
        TFR     X1,B
        MOVE    P:CMB_PTR,X1
        MOVE    X1,P:ACC_PTR
```

```
        MOVE    B,P:CMB_PTR
        BSET    #TRN_SEM,X:SEMFRS       ; Signal that CMB is ready

E_TEST                                  ; Now clear new accum buffer
        CLRACC
;               Clear CURACC (A)        Fetch CMB_SQN
        CLR     A                       X:CMB_SQN,B     ; get the counter

        ADD     Y1,B                    ; increment CMB_SQN
        MOVE    B1,X:CMB_SQN            ; save it
        .ENDI

;               Increm. CUR_AVE
        ADD     Y1,A

;               Save CUR_AVE
        MOVE    A,Y:CUR_AVE
```

Now the four Counter values are read from the counter board, multiplied by the corresponding equalization factors and stored in temporary locations. Notes:

1. Prior of reading the counter a check is made to verify whether the loop must be stopped because it has reached the required number of steps (see section 7.1.2 for a description of this feature).
2. The code is actually implemented as a macro (RDCNT: a detailed description can thus be found in section 4).

Source file: ao_int.asm

```
RD_CNTRS
        .IF     R3 <GT> X:MAXCYCL

                                ;---------------------------------------------
                                ; Stop the loop if the number of cycles
                                ; has been reached
        BCLR    #HF2,X:<<HCR    ; Signal to Host that DSP is busy

        BCLR    #LOO_SEM,X:SEMFRS       ; Clear loop active flag

        WRERR   #E_MAXCY

        IF TIMINT               ; (see description above)
        JSR     PC_CFG          ; Reconfigure port C (and stop timer)
        ENDIF

        JMP     WAIT_ST         ;---------------------------------------------


        .ENDI
```

```
        MOVE    P:ACC_PTR,R7            ; Preset pointer to accum buffer

        MOVE    #0,X0
        MOVE    X0,Y:CSUMTMP            ; Clear 4 counters sum

        RDCNT   CNT1SEL,C1_SUM,X:CN1_TMP,CN1_EQL,CNST_15,Y:(R3)
        RDCNT   CNT2SEL,C2_SUM,Y:CN2_TMP,CN2_EQL,CNST_15,X:(R3)+
        RDCNT   CNT3SEL,C3_SUM,X:CN3_TMP,CN3_EQL,CNST_15,Y:(R3)
        RDCNT   CNT4SEL,C4_SUM,Y:CN4_TMP,CN4_EQL,CNST_25,X:(R3)+
```

In order to normalize the values of the computed *tip-tilt* components for the total number of counts, we compute the reciprocal of the sum of the four equalized counter values. Note that, due to the use of fractional arithmetic and to the previous multiplication by two and by the equalization coefficients (see Section 4), if we call N_i the integer counter values, E_i the equalization coefficients, C_i, the equalized counter values in fractional representation as computed by macro RDCNT, ($i = 1, 2, 3, 4$) and R the reciprocal, we have:

$$C_i = (N_i 2^{-23}) 2 E_i = N_i E_i 2^{-22}$$

$$R = \frac{1}{\sum_{i=1}^{4} N_i E_i} = \frac{2^{-22}}{\sum_{i=1}^{4} C_i}$$

Note that we can use a single quadrant division algorithm because the value of the operands is always positive.

Source file: ao_int.asm

```
        JEQ     SUMZERO                 ; Test if the sum of 4 counters is zero

;               Clear Carry
        AND     #$FE,CCR

;               Get CN1+CN2+CN3+CN4
        MOVE    Y:CSUMTMP,X0

        MOVE    P:CNST_2,B              ; B=2**(-22)
        REP     #24
        DIV     X0,B                    ; Compute the reciprocal
        JMP     STREC
SUMZERO
        MOVE    P:MAXFRAC,B0
STREC
        MOVE    B0,Y:CSUMTMP            ; Save reciprocal
```

After the computation the counters just read are cleared (for a description of the mechanism of counter banks addressing see the description of macro RDCNT in section 4).

```
                                        ; Prepare to reset counters
        BCHG    #CNTSWTC,Y:CNRESET      ; Switch counter bank address
                                        ; in the command word
        MOVE    Y:CNRESET,X0
        MOVEP   X0,Y:PORTOUT
```

In the following fragment of code we actually compute the *tip-tilt* components. The order adopted in the computation has been chosen in order to optimize the use of parallel instructions of the DSP.

Source file: ao_int.asm

```
        MOVE    X:CN1_TMP,X0    ; X0=CN1_TMP
        MOVE    Y:CN4_TMP,B     ; B=CN4_TMP

                                        ;------------------------------------
                                        ; Now we compute the sums: C14, C34,
                                        ; C23, C12
;               A=C14           Y0=CN4
        ADD     X0,B            B,Y0

;                               Save C14
        MOVE                    B,X:C14_TMP

;               B=CN3
        MOVE    X:CN3_TMP,B

;               B=C34           X0=CN3
        ADD     Y0,B            X:CN3_TMP,X0

;                               A=CN2
        MOVE                    Y:CN2_TMP,A

;               A=C23           Y0=CN2
        ADD     X0,A            A,Y0
;                                               Save C34
        MOVE                                    B,Y:C34_TMP

;                               B=CN1
        MOVE                    X:CN1_TMP,B

;               B=C12                           Save C23
        ADD     Y0,B                            A,Y:C23_TMP

;                               Save C12
        MOVE                    B,X:C12_TMP

;                               Y0=C34
        MOVE                    Y:C34_TMP,Y0
```

```
                                                        ; Now we compute the differences
;               preset offset to NS_SUM
        MOVE    #NS_SUM,N7

;               B=12-34(NS)     A=C14
        SUB     Y0,B            X:C14_TMP,A

;               B=0     Y0=(NSdiff*2)
        CLR     B       B,Y0                            ; Normalize

;                       Y1=1/SUMCNTs
        MOVE            Y:CSUMTMP,Y1
```

The normalized difference must be properly scaled in order not to loose precision and avoid overflows. We have coded the algorithm very efficiently based on the following considerations. We want to compute:

$$NS_{diff} = S(C_1 + C_2 - C_3 - C_4)R$$

$$EW_{diff} = S(C_1 + C_4 - C_2 - C_3)R$$

where S is a proper scale factor which allows the fractional representation of the resulting *tip-tilt* component.
Substituting the value of R:

$$NS_{diff} = S\frac{C_1 + C_2 - C_3 - C_4}{\sum_{i=1}^{4} C_i}2^{-22}$$

$$EW_{diff} = S\frac{C_1 + C_4 - C_2 - C_3}{\sum_{i=1}^{4} C_i}2^{-22}$$

Note that NS_{diff} and EW_{diff} must be less than 1, so that they can be represented in fractional notation. We have:

$$\frac{C_1 + C_2 - C_3 - C_4}{\sum_{i=1}^{4} C_i} \leq 1$$

$$\frac{C_1 + C_4 - C_2 - C_3}{\sum_{i=1}^{4} C_i} \leq 1$$

thus:

$$(C_1 + C_2 - C_3 - C_4)R \leq 2^{-22}$$

$$(C_1 + C_4 - C_2 - C_3)R \leq 2^{-22}$$

As a consequence we have chosen $S = 2^{21}$ as a safe scaling factor.

In order to perform the required scaling in an efficient way we first divide the result by 8 (shifting three times to the right) and then we move the least significant word of the 48 bits accumulator to the most significant word (this is equivalent to a multiplication by a factor 2^{24}). The values of registers resulting from the above process ensure that we have no overflow, we don't loose bits in the 24 bits representation of the value and that the sign is correctly maintained.

Source file: ao_int.asm

```
;               B0=(NSdiff*2)
        MAC     Y0,Y1,B
        ASR     B               ; Divide by two
        ASR     B               ; Divide by two
        ASR     B               ; Divide by two

;               Save I_NS  (get the Least Significant half of B)
        MOVE    B0,Y:(R2)

        MOVE    B0,Y:(R3)               ; Save difference to big buffer

;                       Y0=NS_SUM (prefetch)
        MOVE            X:(R7+N7),Y0

;                       X0=C23
        MOVE            Y:C23_TMP,X0

;               A=14-23(EW)     B=I_NS (fetched again to have it in B1)
        SUB     X0,A            Y:(R2),B

;               A=0     X0=EWdiff
        CLR     A       A,X0

;               A=Normalized EWdiff
        MAC     X0,Y1,A
        ASR     A               ; Divide by two
        ASR     A               ; Divide by two
        ASR     A               ; Divide by two

;               B=NS_SUM+I_NS           Save I_EW
        ADD     Y0,B                    A0,X:(R2)

        MOVE    A0,X:(R3)+              ; Save difference to Big buffer

;               Save NS_SUM
        MOVE    B,X:(R7+N7)

;                       prefetch offset to EW_SUM
        MOVE            #EW_SUM,N7

        MOVE    X:(R2),A        ; A=I_EW (fetched again to have it in A1)

;               X0=EW_SUM
```

```
        MOVE    X:(R7+N7),X0

;               A=EW_SUM+I_EW
        ADD     X0,A

;               Save EW_SUM
        MOVE    A,X:(R7+N7)

        MOVE    (R2)+                   ; (R2)->Sample(t-N_COE)
```

As a result of the above computation we have the two *tip-tilt* components:

$$NS_{diff} = 2^{21} \frac{C_1 + C_2 - C_3 - C_4}{\sum_{i=1}^{4} C_i} 2^{-22}$$

$$EW_{diff} = 2^{21} \frac{C_1 + C_4 - C_2 - C_3}{\sum_{i=1}^{4} C_i} 2^{-22}$$

Thus:

$$NS_{diff} = 2^{-1} \frac{C_1 + C_2 - C_3 - C_4}{\sum_{i=1}^{4} C_i}$$

$$EW_{diff} = 2^{-1} \frac{C_1 + C_4 - C_2 - C_3}{\sum_{i=1}^{4} C_i}$$

and substituting the values for C_i:

$$NS_{diff} = 2^{-1} \frac{2^{-22}(N_1E_1 + N_2E_2 - N_3E_3 - N_4E_4)}{2^{-22} \sum_{i=1}^{4} N_iE_i}$$

$$EW_{diff} = 2^{-1} \frac{2^{-22}(N_1E_1 + N_4E_4 - N_2E_2 - N_3E_3)}{2^{-22} \sum_{i=1}^{4} N_iE_i}$$

The factor 2^{-1} is the net scaling factor resulting from the various scaling operations needed in the algorithm in order to maintain the maximum precision.

The last instruction of the this section is needed to prepare for the following filter computation. By incrementing `R2` we have it pointing to the oldest input sample (the one which must be multiplied for the first filter coefficient).

9.2 Filters

In the following section we implement the two FIR filters which produce the two components of the corrections to be applied to the mirror.

At the end of the previous section register `R2` points to the oldest input value for both input channels (the two input components are in fact stored in two aligned memory buffers in X and Y memories.

Note also that the controls from previous section gets here simply because this one immediately follows it (i.e.: the order of includes must not be modified).

The filter code has been derived from the example provided in [6], page B-4, and uses both the hardware implemented circular buffer management and the parallel execution of addition and multiplication capabilities of the DSP 56001. The filter code for the two channels is logically identical and differs only for X and Y memory addressing.

Warning: while the input samples buffer must be necessarily implemented as circular buffer, there is no logical need for the coefficient buffer to be circular (the filter coefficient pointers always starts from the same coefficient at every cycle). During the debugging phase, anyway, we noticed that if the buffers are not both circular with the same modulus the filter loop doesn't work properly.

9.2.1 North-South Channel

We do the filtering of the North-South component first. Remember that R6 points to the beginning of the coefficient vector and R2 to the oldest input sample. After N_COE iterations of the filter algorithm the two registers will have again the same value.

Source file: **ao_filt.asm**

```
                                    ; NORTH-SOUTH CHANNEL FILTERING
        MOVE    #C_NS,R6

        MOVE    M2,X1               ; X1=N_coeff-1


FIR_NS

        MOVE    #NS_CORR,N7         ; Preset pointer to ACB

;       Clear accum.    Load 1st coeff          Load 1st sample
        CLR     A       X:(R6)+,X0              Y:(R2)+,Y0

                                                        ; ------------
        REP     X1                                      ; Filter cycle
        MAC     x0,y0,A X:(R6)+,X0      Y:(R2)+,Y0      ; (N-1 steps)
                                                        ; ------------

;                               prefetch Previous sum
        MACR    X0,Y0,A         X:(R7+N7),X1            ; Last step with
                                                        ; rounding

                                                ; ACCUMULATE FILTER OUTPUT
        BCLR    #6,SR                           ; Clear limit flag to prepare
                                                ; to verify overflow
;                       X0=Filter output
        ADD     X1,A    A,X0

        JLC     CLR1                            ; Check overflow
        BSET    #NSLIMIT,X:(R7)                 ; Set LIMIT flag
```

```
CLR1
;                       Save partial sum
        CLR     A       A,X:(R7+N7)
```

The output of the filter is then scaled and offset to adapt to the DAC dynamic range. The scale and offset parameters are preset by the host (see section 7.1.2).

Source file: ao_filt.asm

```
;               Load scaling factor
        MOVE    X:MRFC_NS,Y0

;                               Scale filter output
        MACR    X0,Y0,A

        MOVE    X:MROF_NS,Y0    ; Fetch output offset
        ADD     Y0,A            ; Add offset


;               X0=Scaled output
        MOVE    A,X0

        MOVE    X0,Y:(R3)               ; Store output in buffer

                                        ;------------------------------
        MOVE    Y:(R5),X0               ; Get input for mirror
                                        ; Write to DAC
        WRDAC   #NS_DAC
```

9.2.2 East-West Channel

Then we process the East-West component. As a result of the circular buffer implementation of both input and coefficient buffers after the N_COE iterations of the first filter registers R2 points again to the oldest input sample and register R6 points to the first filter coefficient.

Source file: ao_filt.asm

```
        MOVE    #C_EW,R6                ; EAST-WEST CHANNEL FILTERING
        MOVE    M2,X1                   ; X1=N_coeff-1

FIR_EW
        MOVE    #EW_CORR,N7             ; Preset pointer to ACB

;       Clear accum.            Load 1st coeff  Load 1st sample
        CLR     A               Y:(R6)+,Y0      X:(R2)+,X0

        REP     X1
        MAC     x0,y0,A Y:(R6)+,Y0      X:(R2)+,X0
```

```
;                               Prefetch previous sum
        MACR    X0,Y0,A         X:(R7+N7),X1




                                                ; ACCUMULATE FILTER OUTPUT
        BCLR    #6,SR                           ; Clear limit flag to prepare
                                                ; to verify overflow
;                       X0=Filter output
        ADD     X1,A    A,X0

        JLC     CLR2                            ; Check overflow
        BSET    #EWLIMIT,X:(R7)                 ; Set LIMIT flag

CLR2
;                       Save partial sum
        CLR     A       A,X:(R7+N7)
```

The output of the filter is scaled and offset to adapt to the DAC dynamic range. The scale and offset parameters are preset by the host (see section 7.1.2).

Source file: **ao_filt.asm**

```
;                       Load scaling factor
        MOVE            X:MRFC_EW,Y0

;                               Scale filter output
        MACR    X0,Y0,A


        MOVE    X:MROF_EW,Y0            ; Fetch output offset
        ADD     Y0,A
                                        ; Accumulate filter output
;                       X0=Scaled output
        MOVE            A,X0


        MOVE    X0,X:(R3)+              ; Store output in buffer


                                        ;-----------------------------
        MOVE    X:(R5)+N5,X0            ; Get input for mirror
                                        ; Write to DAC
        WRDAC   #EW_DAC
```

At the end of filter section register R2 points again to the oldest input sample which will be overwritten by the value read in the next cycle.

9.3 End of Timer Interrupt Service

The following fraction of code performs the final operations to end the processing of a single step of the loop. It terminates with an RTI instruction in order to return properly from the interrupt initiated process.

9.3.1 Reset Counters

After the correction values are written to the mirror DAC's, the last read counter bank is reset to be ready for the next cycle.

Source file: ao_endi.asm

```
                                    ; RESET COUNTERS
        MOVE    Y:CNRESET,X0        ; Load output word
        BSET    #RES_CNT,X0         ; Set "reset" bit
        MOVEP   X0,Y:PORTOUT        ; Output to port
        BCLR    #RES_CNT,X0         ; Clear "reset" bit

        REP     #DELAY              ; Wait for line stabilization
        NOP

        MOVEP   X0,Y:PORTOUT        ; Output to port

        REP     #DELAY              ; Wait for line stabilization
        NOP
```

9.3.2 Returning from the Timer Interrupt

Here we perform the housekeeping operations required prior of returning from the Timer Interrupt Servicing Routine.

Source file: ao_endi.asm

```
                                    ; END OF INTERRUPT SERVICE ROUTINE
        MOVE    P:SAV_Y1,Y1         ; restore Y1
        MOVE    L:SAVE_AX,A         ; Restore A

        BCLR    #TMR_SEM,X:SEMFRS   ; Signal that Timer routine
                                    ; has finished
        BSET    #HCIE,X:HCR         ; reenable host commands
        RTI                         ; Return from Interrupt
```

10 Interrupt Vectors Table

Here follows a piece of code needed to define the Interrupt Vector Table for the Host Command processing and for the Timer. The table generated here is simply a copy which is moved to the right address (starting at $24) during the initialization of the TNG_AO software (see section 6).

The active Interrupt Vector Table has a fixed position in P Memory starting at P:$0 up to P:$3F (see [6], pag. 6-4). The first eighteen vectors are reserved for various system uses; we use ten out of thirteen available vectors for Host Command Processing (see section 7 and we overwrite vector $1C to address the Timer Interrupt Servicing routine (See section 9).

10.1 Host Command Table

We define here the ten Host Command Processing addresses.

Source file: **ao_vec.asm**

```
                                ;       Host command table
VSTART
        JMP     >STRTMON        ; cod=12, jump to Monitor launch
        JMP     >LDPARAM        ; cod=13, jump to parameters configuration
        JMP     >LDCOEFF        ; cod=14, jump to coefficients configuration
        JMP     >LDSAMPL        ; cod=15, jump to samples configuration
        JMP     >OPNLOOP        ; cod=16, jump to open loop
        JMP     >CLSLOOP        ; cod=17, jump to closed loop
        JMP     >MRDRIVE        ; cod=18, jump to mirror drive
        JMP     >DWNLBUF        ; cod=19, download memory buffers
        JMP     >STPLOOP        ; cod=1A, jump to stop function
        JMP     >SNDIDNT        ; cod=1B, jump to download banner routine
        JMP     >WRERROR        ; cod=1c, write error code
VEND
```

10.2 Timer Interrupt Vector

Source file: **ao_vec.asm**

```
TINT_1  JSR     >TIO_L          ; Timer interrup vector
TINT_1E
```

A The DSP Bootstrap Process

In order to help the reader to understand some aspects of the DSP code we resume here the description of the Bootstrap process executed by the DSP at startup or when a reset signal is received.

The DSP Operating Modes are set so that the boot sequence is as follows[6]:

1. The control logic maps the bootstrap ROM in program memory at address `$0000`. In this state the read operations come from the ROM and can address 32 words, while the write operations go to the program RAM (Pram) with full address capabilities.

2. Program execution starts at location `P:$0000`. The ROM contains the code necessary to copy onto address `P:$0000-$01FF` 512 words read from a byte-wide EPROM starting at address `P:$C000`.

3. After the loading the ROM program removes the bootstrap ROM from the program memory map and re-enable read/write access to the program RAM, then executes a `JMP #<00` instruction to start the execution of the user program, again at address `P:$0000`.

4. The following step will be different in the development version of the TNG_AO code and in the final one:

   (a) In the development version the EPROM provided with the DBV56H DSP board contains a second level bootstrap routine which loads the monitor program (also stored in the same EPROM) in the upper part of the Pram and starts it.

   (b) In the final version the EPROM will contain a slightly different version of second level bootstrap, together with the Monitor and the TNG_AO code. The second level bootstrap will load the monitor code as in the development version and then the TNG_AO code and will pass the control to the latter. The monitor code will remain in program memory to be re-enabled by means of the related host command (see command `STRTMON` in section 7).

[6] For a complete coverage of the DSP 56001 operating modes and the various bootstrapping options see [6], section 3.3.5.

B Symbol List

In the following table are listed all the symbols used in the TNG_AO code together with the indication of the memory area, the absolute address the data type and the length. Dta types are: DATA (storage locations), CONST (initialized storage locations), MOD (storage locations aligned for circular addressing).

Symbol	Mem	Address	Type	Length
ACC_PTR	P	1801	DATA	1
C12_TMP	X	6D2B	DATA	1
C14_TMP	X	6D2A	DATA	1
C23_TMP	Y	6D17	DATA	1
C34_TMP	Y	6D16	DATA	1
CM1_BLX	X	6D01	DATA	9
CM1_BLY	Y	6D01	DATA	9
CM2_BLX	X	6D0A	DATA	9
CM2_BLY	Y	6D0A	DATA	9
CMB_PTR	P	1800	DATA	1
CMB_SQN	X	6D14	DATA	1
CN1_EQL	X	6D1D	DATA	1
CN1_TMP	X	6D28	DATA	1
CN2_EQL	X	6D1E	DATA	1
CN2_TMP	Y	6D14	DATA	1
CN3_EQL	X	6D1F	DATA	1
CN3_TMP	X	6D29	DATA	1
CN4_EQL	X	6D20	DATA	1
CN4_TMP	Y	6D15	DATA	1
CNRESET	Y	6D18	DATA	1
CNST_1	P	0815	CONST	1
CNST_10	P	0819	CONST	1
CNST_15	P	081A	CONST	1
CNST_2	P	0816	CONST	1
CNST_20	P	081B	CONST	1
CNST_25	P	081C	CONST	1
CNST_FF	P	0817	CONST	1
CNST_M1	P	0818	CONST	1
CNT1SEL	Y	6D19	DATA	1
CNT2SEL	Y	6D1A	DATA	1
CNT3SEL	Y	6D1B	DATA	1
CNT4SEL	Y	6D1C	DATA	1
CSUMTMP	Y	6D13	DATA	1
CUR_AVE	Y	6D1D	DATA	1
C_EW	Y	6F00	MOD	256
C_NS	X	6F00	MOD	256

```
INIT_EW    X    6D26    DATA       1
INIT_NS    X    6D25    DATA       1
I_EW       X    6E00    MOD      256
I_NS       Y    6E00    MOD      256
LAST_ER    Y    6D1E    DATA       1
LD_CHK     X    6D15    DATA       1
LMODE      X    6D16    DATA       1
MASK_16    P    081D    CONST      1
MAXCYCL    X    6D1B    DATA       1
MAXFRAC    P    081E    CONST      1
MRFC_EW    X    6D23    DATA       1
MRFC_NS    X    6D21    DATA       1
MROF_EW    X    6D24    DATA       1
MROF_NS    X    6D22    DATA       1
N_AVE      X    6D1A    DATA       1
N_COE      X    6D17    DATA       1
N_CUR      X    6D27    DATA       1
N_SAM      X    6D18    DATA       1
N_SHIFT    X    6D1C    DATA       1
O_EW       Y    7000    MOD     4096
O_NS       X    7000    MOD     4096
SAVE_AX    X    6D00    DATA       1
SAVE_AY    Y    6D00    DATA       1
SAV_Y1     P    1802    DATA       1
SEMFRS     X    6D13    DATA       1
T_FREQ     X    6D19    DATA       1
X_BUF      X    0000    MOD    27904
Y_BUF      Y    0000    MOD    27904
Z_BUFX     X    6DFE    MOD        2
Z_BUFY     Y    6DFE    MOD        2
```

References

[1] L. Fini, P. Ranfagni, “The TNG Tip-Tilt Servo-Loop”. TNG Technical Report N. 54, Dec. 1995.

[2] V. Biliotti, L. Fini, “TNG Tip-Tilt Interface/Counter Board”. Arcetri Technical Report in preparation.

[3] L. Fini, P. Ranfagni, “TNG Tip-Tilt System Supervisor Library Reference Manual. Version 2.1”. Arcetri Technical Report N. 2/1996, Firenze, Jun. 1996.

[4] “DBV56H, DSP56001 VME Board User Manual. Version 2.01”. Loughborough Sound Images Ltd., May 1994.

[5] A. Chrysafis, S. Lansdowne, “Fractional and Integer Arithmetic Using the DSP56000 Family of General Purpose Digital Signal Processors”. Motorola Inc., 1988.

[6] “DSP56000/DSP56001 Digital Signal Processor User’s Manual”. (DSP56000UM/AD, Rev. 2), Motorola Inc., 1990.

[7] “Motorola DSP56000 Macro Assembler Reference Manual”. Motorola, Inc., 1987.