

TNG Tip-Tilt System  
Supervisor Library Reference Manual  
Version 2.1

Giugno 1996

Luca Fini(\*), Piero Ranfagni(\*)

(\*) Osservatorio Astrofisico di Arcetri

Arcetri Technical Report N. 2/1996  
Firenze, Giugno 1996

## INTRODUCTION

The following manual describes a library of C subprograms which may be used as building blocks for the development of the TNG Tip-Tilt system software on the Supervisor.

The manual is related to version 2.x of the library which has been modified with respect to the previous one in order to be compatible with the final version of the TNG\_AO DSP code.<p>

The Supervisor is a 68040 based board which communicates through the VME bus with a DSP 56001 based board (Loughborough DBV56H) where the Tip-Tilt loop is implemented. The purpose of the library is to make the development of the tip-tilt loop control software easier both for the implementation in the laboratory and for the final version at the telescope.

The manual is divided in two parts: Part I describes the routines used to control the DSP board; Part II describes the routines used to communicate with the APD Motherboard through a serial line.

This software has been implemented by Luca Fini(1) and Piero Ranfagni(2) at the Osservatorio Astrofisico di Arcetri as part of the development of the Tip-Tilt correction system for the Galileo National Telescope (TNG).

(1) E-Mail: [lfini@arcetri.astro.it](mailto:lfini@arcetri.astro.it)

(2) E-Mail: [ranfagni@arcetri.astro.it](mailto:ranfagni@arcetri.astro.it)

## PART I

### DSP CONTROL MODULES

#### IMPLEMENTATION NOTES

1. All required symbols, macros and function prototypes are defined in the include file:

```
dsplib.h
```

2. Every program using the library MUST include a global section used to initialize properly the pointers to the VME board. This may be done by defining the symbol MAIN before including dsplib.h, only in the main program (or anyway, in only one of the program modules), as in the following example:

```
#define MAIN  
#include "dsplib.h"
```

3. Software Versions

The DSP software has two flavours: a version used for development purposes and the final version to be freezed into the DBV56H board EPROM. The two version are, obviously, functionally identical, but may be different for some details. E.g.: a hardware reset in development version will reload into the program RAM of the DSP the content of development EPROM (i.e.: the Monitor) and the Tip-Tilt software must be reloaded interactively with the proper monitor command and then interactively started, while in the final version a reset will directly reload the TipTilt software and start it. So in the development version it is not handy that the program running on the VME host issues the hardware reset at the beginning, but it is better to use a manual reset.

This example shows that there are also some sligth differences in the software from the VME host point of view.

All the software developed for the TNG Tip-Tilt system will cope with development and final versions differences by means of conditional compilation sections of code. To compile a version of the code ready for the final version the compiler must define the preprocessor symbol:

```
EPROM_VERSION
```

Otherwise a development version is generated.



DSPCMB      DSP Communication block structure

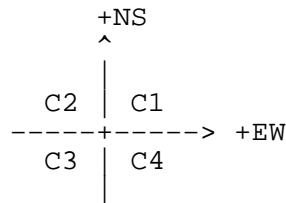
This structure is used to receive the Communication Block from the DSP  
It is used as argument of the routine DSPreadb()

```
typedef struct {
    unsigned Code;           DSP status code
    unsigned Seqn;          Comm. block sequence number
    unsigned Guard;         Guard bits
    unsigned Values[CMBDBLEN]; Comm. block data buffer
} DSPCMB;
```

The following symbols may be used to index values in the CMB data buffer:

```
#define C1_SUM            0            Accumulated sum of counter 1
#define C2_SUM            1            Accumulated sum of counter 2
#define C3_SUM            2            Accumulated sum of counter 3
#define C4_SUM            3            Accumulated sum of counter 4
#define NS_SUM            4            Accumulated sum of N-S differences
#define EW_SUM            5            Accumulated sum of E-W differences
#define NS_COR            6            Accumulated sum of N-S correction
#define EW_COR            7            Accumulated sum of E-W correction
```

Note: the counters are assumed to be organized as the four quadrants of a cartesian plane with NS direction along the abscissa and EW along the ordinate as in the following scheme:



So:

$$\begin{aligned} \text{N-S Difference} &= (C1+C2)-(C3+C4) \\ \text{E-W Difference} &= (C1+C4)-(C3+C2) \end{aligned}$$

## SUBPROGRAM LIST

DSPcheck	Check for error status
DSPclose	Clears the communications between the VME host and the DSP
DSPdownl	Reads a block of data out of the DSP buffer memory
DSPident	Reads identification information from the DSP
DSPreadb	Reads a communication block from the DSP
DSPreadw	Reads a 24 bit word from the DSP
DSPready	Wait for the DSP to signal it is ready
DSPwriteb	Writes a vector of 24 bit words to the DSP
HdReset	Hard reset.
LDcoeff	Loads the filter coefficients to DSP
LDparam	Loads the loop parameters to the DSP
LDsampl	Loads the mirror drive samples to the DSP
StartLoop	Starts a loop mode
StartMon	Start the DSP Monitor Program
StopLoop	Stops any loop mode
dbl2dsp	Convert Double precision into Fixed point DSP format
dsp2dbl	Convert Fixed point DSP format into double precision
timerUs	Converts timer control word into microseconds
usTimer	Generate the timer control word for the DSP



DSPclose Clears the communications between the VME host and the DSP

Source file: dspio.c

This routine clears the communication through the VME between the host and the DSP.

Currently it is likely of no use, in that a following reset of the DSP will do the same, but it is good programming practice to call it at the end of a task

USES: Nothing

int DSPclose() Returns 0 on success

DSPdownl Reads a block of data out of the DSP buffer memory

Source file: dspcmd.c

The DSP loop modes can be preset so that data are stored in the DSP internal memory buffers at each timer cycle. This routine transfers the content of the buffers to the VME host memory.

The organization of data in the return buffer are as follows:

CNT1 CNT2 CNT3 CNT4 DIF\_NS DIF\_EW COR\_NS COR\_EW

Note: Counts (CNT1-4) and correction values to be sent to the DACs Computed tip-tilt values (DIF\_NS, DIF\_EW) are represented in the DSP 56001 internal fixed point fractionary format (COR\_NS,COR\_EW) are represented as 24 bits unsigned integers.

After the request, the DSP will stop the current Loop mode and send back as many blocks as available starting from the most remote in the past up to the latest. If enough space is not available in the caller provided buffer only as many blocks as available are transferred into it.

USES: DSPreadw()

```
int DSPdownl(Buf,MaxLen,
             ActLen)
```

Returns 0 on success; (-1) on error.  
The error code is stored in global variable DSPerrno.

Note: error code NOSPACERR signals that the caller didn't provide enough space for the full sequence. In this case the number of data words transferred will be MaxLen and the argument ActLen will contain the number required for a full transfer.

```
int Buf[];
int MaxLen;
int *ActLen;
```

Output data buffer (caller provided)  
Buffer maximum length  
Returns number of data read or required space on overflow (see above)

DSPident Reads identification information from the DSP

Source file: dspcmd.c

This routine requires the DSP to send back some information which identifies the software version.

USES:

char *DSPident()	Returns a pointer to identification string Returns NULL if an error condition occurs. The error code is put in global variable DSPerrno.
------------------	---

DSPreadb Reads a communication block from the DSP

Source file: dspio.c

The routine waits until a full communication block has been received from the DSP and returns it in a proper structure.

See the Introduction section for a description of the content of the data block sent back by the DSP.

USES: DSPreadw()

int DSPreadb(DSPcmb)	Returns completion code
	0 No error
	(-1) Error code from DSP (The value can be read with a call to DSPcheck())
DSPCMB *DSPcmb;	Pointer to a DSPCMB structure to receive the DSP Communication block

DSPreadw Reads a 24 bit word from the DSP

Source file: dspio.c

The routine waits until a word is available on the interface, then reads it and returns a 24 bits unsigned value.

DSP words are read by the VME host as single bytes at three non contiguous locations.

USES: Nothing

int DSPreadw()

Returns the word read from the DSP.  
If the DSP signals an error condition then the reading is interrupted, and the value (-1) is returned. The error code can be then read with a call to DSPcheck().

DSPready Wait for the DSP to signal it is ready

Source file: dspcmd.c

This routine waits for the DSP ready signal, then initializes the Host port from the VME side. It is used at the beginning of the main program or after a hard reset command (HdReset()) to avoid problems due to the time needed by the DSP after reset to reload the code into program RAM and preset everything for Tip-Tilt operation.

This routine uses bit HF2 of the ISR

USES: nothing

int DSPready() Returns 0 on success

DSPwriteb Writes a vector of 24 bit words to the DSP

Source file: dspio.c

The routine sequentially writes the 24 bits words from a buffer to the DSP.

Synchronization is performed by means of the TXDE bit of the DSP-VME interface ISR register. This bit is set when the I/O register is empty.

DSP words are written by the VME host as single bytes at three non contiguous locations. Care is taken to maintain the correct byte order.

No check is performed on the values to write, if they're greater than 24 bits the upper byte is simply not sent.

USES: Nothing

int DSPwriteb(Buf,len)	Returns the number of words successfully written. Note: currently no error check is provided, the routine can return only when all the words have been sent.
int Buf[];	Buffer holding the input words
int len;	length of buffer



LDcoeff Loads the filter coefficients to DSP

Source file: dspcmd.c

This routine loads a new set of filter coefficients to the DSP. If called when the DSP is in any looping mode the loop is stopped. After loading the DSP remains in wait state.

USES: DSPwriteb(), dbl2dsp()

```
int LDcoeff(Num,Coeffs)           Returns 0 on success
int    Num;                       Number of coefficients. It must be in the
                                   range: [2,256]
double Coeffs[];                  vector holding filters coefficients

                                   All coefficients must be in the range:
                                   [-1.0,1.0)

                                   The coefficient are ordered in vector in
                                   "time" order, i.e.: the first coefficient
                                   multiplies the sample which is farthest
                                   in the past.
```

LDparam Loads the loop parameters to the DSP

Source file: dspscmd.c

This routine loads new loop parameters to the DSP. If called when the DSP is in any looping mode the loop is stopped. After loading the DSP remains in wait state.

USES: DSPwriteb(), usTimer()

```
int LDparam(Time,MaxCycles,      Returns 0 on success
            NAVg,
            CN1_equ, CN2_equ,
            CN3_equ, CN4_equ,
            MRofsNS, MRfctNS,
            MRofsEW, MRfctEW,
            NSinit, EWinit )
```

double Time; Counter sampling time in microsec.  
This value must be grether than the minimum time needed by the DSP to perform the computation of a single step of the loop  
For a 27 MHz clock the step time can be evaluated with the following relation:

$$T = 0.2847 N + 43.14$$

Where N is the length of the filter (number of coefficients)

int MaxCycles; If set to zero any loop mode started will run until it is explicitly stopped by another command. If greater than zero, the loop will be stopped after the specified number of cycles.

Note: This value is limited by the dimension of the internal buffer where real time data are stored.  
The current limit is: 6796 cycles.

int NAVg; Number of averaging cycles  
The number of cycles during which real time data are summed up prior of sending back the result to the host

double CN1\_equ; Equalization coefficient for counter 1  
double CN2\_equ; Equalization coefficient for counter 2  
double CN3\_equ; Equalization coefficient for counter 3  
double CN4\_equ; Equalization coefficient for counter 4

All equalization coefficients must be in the range: [-1.0,1.0)

int MRofsNS;                   Mirror drive offset value. N-S component  
Range: [0,65535]

int MRfctNS;                   Mirror drive gain    value. N-S component  
Range: [-8388607,+8388607]

int MRofsEW;                   Mirror drive offset value. E-W component  
Range: [0,65535]

int MRfctEW;                   Mirror drive gain    value. E-W component  
Range: [-8388607,+8388607]

double NSinit;                 NS-Filter memory initialization value  
double EWinit;                 EW-Filter memory initialization value

When changing the loop characteristics on the fly, if the filter memory is re-initialized to zero, a spike may be generated on the output to the mirror. To avoid this undesired effect we allow to preinitialize all the filter memory to a constant different from zero (actually the average of a proper number of the last inputs to the filters, available in the VME communication block, can be used)

To understand the role of above factors one must refer to the closed loop computation algorithm:

1 - Integer counter values (CNTn) are scaled as follows:

$$C_n = 2^{(-23)} * CNT_n * C_{Nn\_equ}$$

where: C<sub>n</sub>           - Equalized value on counter n  
C<sub>Nn\\_equ</sub>       - Equalization coefficient for counter n  
CNT<sub>n</sub>           - Integer count read from counter n  
                  (N = 1,2,3,4)

2 - The Normalized differences are computed:

$$DIF\_NS = 2^{(-1)} (C1+C2-C3-C4) / (C1+C2+C3+C4)$$
$$DIF\_EW = 2^{(-1)} (C1+C4-C2-C3) / (C1+C2+C3+C4)$$

3 - DIF<sub>xx</sub> are passed through the filter.

4 - The output of filter OUT<sub>xx</sub> is then scaled:

$$COR\_xx = ( (OUT\_xx * MRfctxx) + MRofsxx)$$

where: COR<sub>xx</sub>       - 16 bit unsigned value sent to DAC xx  
                  range: [0,65535]  
OUT<sub>xx</sub>           - Output from filter xx

MRfctxx - Gain factor for DAC xx  
MRofsxx - Offset for DAC xx  
(xx = NS, EW)

LDsAMPL Loads the mirror drive samples to the DSP

Source file: dspcmd.c

This routine loads a new set of mirror drive samples to the DSP. If called when the DSP is in any looping mode the loop is stopped. After loading the DSP remains in wait state.

USES: DSPwriteb()

```
int LDsAMPL(Num,NSsmp1,EWsmp1) Returns 0 on success
int Num; Number of coefficients. It must be in the
range: [0,4096]
int NSsmp1[]; vector holding N-S samples
int EWsmp1[]; vector holding E-W samples
Integers contained in the two buffers are
sent untouched to the DA converter (or to
the mirror digital channel). The caller
must set them up properly for the purpose

The length of the two vectors must be the
same in that there is no way to drive the
two components with two different rates
```

StartLoop Starts a loop mode

Source file: dspcmd.c

This routine Starts a loop mode of the DSP. loop mode selectors are defined in DSPlib.h

When the DSP is in loop mode can be stopped by calling StopLoop() or any other command

USES:

```
int StartLoop(Mode)
int Mode;
```

Returns 0 on success  
Mode selector. It must be one of:

OPNLOOPCMD  
CLSLOOPCMD  
MRDRIVECMD

StartMon Start the DSP Monitor Program

Source file: dspcmd.c

This routine restrts the Monitor program resident in the DSP memory. After this command the DSP control goes to the Monitor and interaction with the DSP can be performed through the DBV56H resident serial line.

USES: nothing

int StartMon() Returns 0 on success

StopLoop Stops any loop mode

Source file: dspcmd.c

This routine Stops the loop mode currently active. No other operation is performed, and the DSP returns to WAIT mode.

USES:

int StopLoop() Returns 0 on success

dbl2dsp      Convert Double precision into Fixed point DSP format

Source file: cfcvt.c

This routine transform a double precision value in the Range: [-1.0,1.0) Into the proper internal representation of the DSP 56001 (24 bits, signed fixed point with the binary point left justified)

USES: Nothing

int dbl2dsp(Value)                      Returns the value converted into 24 bits integers (all values will look positive to a 32 bits processor, so there is no conflict with the error return).

  If a range error occurs returns (-1)

double Value;                          Value to be converted (range: [-1.0,1.0) )

Note: For the user convenience the value 1.0 is accepted as a valid input and converted to the closest representable number: 0.99999988.

dsp2dbl Convert Fixed point DSP format into double precision

Source file: cfcvt.c

This routine transform a value from the DSP 56001 fixed point format into a double precision value in the Range: [-1.0,1.0)

USES: Nothing

double dsp2dbl(Value)	Returns the value converted into double precision number in the range [-1.0,1.0)
unsigned Value;	Value to be converted (24 bit integer)

timerUs      Converts timer control word into microseconds

Source file: dsptimer.c

This routine returns the exact time value (in microseconds) corresponding to the given DSP 56001 internal Timer code.

Note: see routine usTimer for a description of the DSP 56001 internal timer code.

USES: Nothing

double timerUs(Tcode)	Returns time interval in microseconds returns (-1.0) on error
int Tcode;	DSP 56001 Timer code

usTimer      Generate the timer control word for the DSP

Source file: dsptimer.c

The DSP 56001 internal timer has two resolution steps: high time resolution (2.3704 microsec. steps) and low resolution (18.963 microsec. steps). High resolution may be used for a maximum time interval of: 9.709 ms and low resolution for a maximum of 77.68 ms.

NOTE: the above values are computed for a clock frequency of 27.0 MHz

The routine usTimer approximates the time interval (given as a double in microseconds) with the maximum resolution available and yields the code needed to program the timer for the given interval.

The code only sets the CD bits (the 12 LSB's) and possibly the SCP bit, the other bits of the word are set to 0.

USES: Nothing

int usTimer(Msecs)	Returns the timer control code or (-1) on error
double   Msecs;	Interval value (microsec) This value must be in the range: [1.18,77680.0]

PART II

SERIAL COMMUNICATION ROUTINES

COMMANDS

=====

```
#define DSPNOERROR    0           Normal completion return
                              *****
                              Read commands

#define HV            0x0
#define HVBIASLIK    0x1
#define VQUENCH      0x2
#define TEMP_APD     0x3
#define TEMP_SLIK    0x4
#define VREF         0x5
#define CHECK        0x6
#define ALL          0xf
#define EEPROM       0x8
#define OP_MODE      0x9
#define FAULT        0xa

                              *****
                              Write commands

#define STBY         0x0
#define TEST_MC      0x1
#define TEST_QC      0x2
#define EXTENSION    0x3
#define WRITING      0x4
#define SHTDWN       0x5
#define SILENT       0x6
#define NORMAL       0x7
```

## SUBPROGRAM LIST

SERclose	Closes the communication with the serial port
SERopen	Opens the communication with the serial port
SERrcvd	Receives from the Motherboard a reply to a read command
SERrdcmd	Sends a read command to the Motherboard
SERwrcmd	Sends a write command to the Motherboard

SERclose Closes the communication with the serial port

Source file: serial.c

This routine opens the communication with the APD motherboard through the serial line

USES: Nothing

int SERclose() Returns 0 on success



SERrcvd      Receives from the Motherboard a reply to a read command

Source file: serial.c

This routine reads from the Motherboard the data block sent as a reply to a read command (see SERrdcmd()).

USES: Nothing

```
int SERrcvd(Buffer)                      Returns the number of data items
                                         returns (-1) on system error. See the
                                         variable errno for error code.
                                         returns (-2) if SERopen has not been called
                                         returns (-3) on i/o errors
                                         returns (-4) if first char is not SYNC
                                         returns (-5) for data type mismatch
                                         returns (-6) for checksum not verified

int Buffer[MAX_RCVBUF];                  Receive buffer
```

SERrdcmd Sends a read command to the Motherboard

Source file: serial.c

This routine sends a read command to the Motherboard after the call the routine SERrcvd() can be called to read the reply.

USES: SERsend

```
int SERrdcmd(code,apd,          Returns 0 on success
                rm,tout)       returns (-1) on system error. See the
                                variable errno for error code.
                                returns (-2) if SERopen has not been called
int code;                       Function code. It must be one of the
                                following symbols defined in serial.h:
                                HV
                                HVBIASLIK
                                VQUENCH
                                TEMP_APD
                                TEMP_SLIK
                                VREF
                                CHECK
                                ALL
                                EEPROM
                                OP_MODE
                                FAULT
int apd;                          APD selector (1,2,3,4)
int rm;                             =0 Request only
                                    =1 Actual read
int tout;                           =0 timeout disabled
                                    =1 timeout enabled
```

SERwrcmd Sends a write command to the Motherboard

Source file: serial.c

This routine sends a write command to the Motherboard

USES: SERsend

```
int SERwrcmd(opmode,apd,          Returns 0 on success
              thv,extn,data)      returns (-1) on system error. See the
                                  variable errno for error code.
                                  returns (-2) if SERopen has not been called
int opmode;                       Operating mode. It must be one of the
                                  following symbols defined in serial.h:

                                  STBY          APDs are on with HV under break-
                                                down voltage
                                  TEST_MC      Start microcontroller self test
                                  TEST_QC      Start test for given APD
                                                controller
                                  EXTENSION    Set extension mode
                                  WRITING     Write data into controllers
                                  SHTDWN      Set shutdown state
                                  SILENT      Disable serial writing from the
                                                motherboard
                                  NORMAL      Set normal status

int apd;                           APD selector (1,2,3,4)

int thv;                            =0 TBD
                                    =1 TBD

int extn;                           Extension bits
int data;                           data bits
```