Correlator Block and Delay Unit Control Computer Command Language

David Loone 7 Jan, 1993
Ron Heald 1 Apr, 1995

1. Overview

This document describes two computer systems used in the Australia Telescope National Facility (ATNF) correlator. The Block Control Computer (BCC) provides control for the Correlator Blocks. Each block consists of a Block Bus Switch board, a Block Distributor board, and 8 Correlator Module boards. Together they correlate the data for a single baseline, i.e. one pair of antennas. Three blocks normally reside in a rack and are controlled by a single BCC.

The Delay Unit Control Computer (DUCC) controls up to 24 delay units and recirculators. The delay units insert an artificial delay into the signal path to compensate for the different distances from antenna to source. The recirculators can used whenever the input data rate is below that at which the individual correlators operate (8 MHz in 1-bit mode or 4 MHz in 2-bit mode). By *recirculating* the data the number of correlated lags is increased resulting in better spectral resolution.

These computer systems are quite similar in design differing only in the set of commands they execute and their status displays. Both use an Intel 80386 or 80486 computer and run the pSOS operating system from Integrated Systems. Both receive commands from and respond to the host Correlator Control Computer (CCC). This communication is via a RS-232 serial line or ethernet using the TCP/IP protocol.

When communicating with the BCC system the serial port can only be used to control block 0. When the network communication is used commands for Correlator Blocks 0, 1, and 2 are accepted on network ports 4000_{10}, 4001_{10}, and 4002_{10}, respectively. The DUCC system has 7 communication ports which accept commands: the serial port plus 6 network ports 4000_{10} through 4005_{10}. Each of these can control any or all of the 24 delay units and recirculators.

Command Format

Commands are passed from the CCC to the BCC and DUCC systems as ASCII text. A command consists of a line of text beginning with a period (“.”) character followed by a two letter command and any command arguments. The command line is terminated by a carriage return character (ASCII 13), a new-line character (ASCII 10), or both. Some commands require an input data block containing extra information required by the command. This input data block immediately follows the command line. In addition, some commands return an output data block containing information obtained by the command.

Lines returned by the system to the CCC are terminated by a carriage return character (ASCII 13) followed by a new-line (ASCII 10). After any command has been executed (whether or not any input or output data blocks were involved) the system returns a line containing a single hexadecimal number which is an error code. If the command was successfully completed this error code will be zero. A non-zero error code indicates that the command could not be completed and the value indicates the nature of the error (see Return Error Codes below).

The majority of this document is used to describe the individual commands. In the descriptions the first line gives the command name and its basic function. The second line gives the command and its arguments. Each argument is listed in italic and optional arguments are enclosed in square brackets ([...]). The bullet (•) character indicates a space or tab character. The next lines contain a description of the "type" of each argument. A list of bold characters enclosed in parenthesis means that argument may assume any of the emboldened characters. Two numbers separated by two periods (..) means that argument may accept any numerical value between or including the two values. Numerical values separated by commas means that the argument may assume any of the numerical values listed. All numerical values given are in hexadecimal. (The one exception to this rule is the ETD specification given in the .EE and .LT commands.) Normal type is used for characters passed from the CCC to the system, bold type for the characters passed from the system to the CCC.

For example, the .SM command is used to select the Correlator Modules which will be accessed in following commands. The following is an example of the use of this command.

```
.SM • 5 • 6
0
```

Input Data Blocks

Some commands require extra input data. This is supplied in an *input data block*. An input data block is one or more lines of ASCII text with each line containing a single value. Each line is terminated by a carriage return character (ASCII 13), a new-line character (ASCII 10), or both. The block is terminated by a line containing only the tilde character ("~", ASCII 126). The command error code is returned after the end of block marker is received by the system.

The system will read as much data as it requires from an input data block ignoring any excess. If too much or not enough information is in the input data block then an error is returned.

For example the .DP command is used to set the delay setting of a previously defined block of delay units. The following is an example of the use of this command and illustrates the use of an input data block. The delay unit block contains three delay units and the three values given in the input data block will each be used to set the corresponding delay unit.

```
.DP • 0
3F49
2746
104C
~
0
```

Output Data Blocks

For the system to send data to CCC an *output data block* is used. An output data block consists first of a line containing only the percent sign (“%”, ASCII 37), followed by one or more lines of text, and finally terminated by a line containing only the tilde character ("~", ASCII 126). The error code for the command is returned after the block has been sent. The end of block line will always be returned even when an error occurs and no data is sent. If an error occurs while the data for the output data block is being generated then the output data block is shortened by sending the end of block line and error code immediately.

The CCC software should read all required information from an output data block then discard the rest of the data up until the end of block line. This convention allows backward compatible upgrades to the system.

For example the .GT command is used to obtain the current atomic time. The following is an example of the use of this command and illustrates the use of an output data block.

```
.GT
%
F04D16EF33EF0
~
0
```

Front Panel Indicators

There are 10 LED indicators on the front of the Correlator Block Bus Switch board of which three are used by the BCC. These provide information about the current state of the BCC software and hardware. The functions of these indicators are:

• CPU OK (yellow). Indicates the BCC is functioning properly by being on and off during alternate seconds.

• EVT GEN (yellow). Will be lit whenever the software is generating data for the event generator.

• BCC WAIT (yellow). Will be lit whenever the BCC is waiting for a command from CCC, or inversely will be off whenever it is executing a command.

Return Error Codes

At the end of execution of any command the system returns an error code to the host computer. Error codes consist of 16-bit unsigned numbers (four hexadecimal digits). The possible error codes are given below.

Error	**Description**
0000	Ok. The command completed successfully.
7001	IllegalCommand. A line starting with a period (".") has been received but the next two characters were not recognised as a legal command.
7002	MissingArgument. A command line argument which is required by the given command was not present in the command line.
7003	IllegalArgument. A command line argument either contained an illegal character and could not be converted, or contained an illegal value.
7004	IllegalMode. The command line contained an illegal mode.
7005	MemoryAllocError. An error occurred when the system attempted to allocate a memory block.
7006	DataBlockValueError. An input data block element either contained illegal characters and could not be converted, or contained an illegal value.
7007	MissingDataBlockElement. There were not enough lines in the input data block following the command.
7008	TooManyDataBlockElement. There were too many lines in the input data block following the command.
7009	ExceededRecursiveLimit. Calls to the system shell were nested too deeply. This can only occur when using the .EX command.
700A	FileNotFound. The given disk file name was not found. This can only occur when using the .EX command.
700B	ProgFileNotFound. The given memory file name was not found. This can only occur when using the .PA, .PD, .PE, and .PX commands.

700C	ProgFileCRCError. A CRC error occurred when reading the given memory file. This can only occur when using the .PA, .PD, .PE, and .PX commands.
700D	NonExistentMemory. The interface status register indicated an attempt was made to access a memory location which does not exist.
700E	MemoryAddressOverflow. The interface status register indicated that the address register overflowed while accessing a memory location.
700F	UnknownETD. The Event Timing Description for the given index has not been defined with .LT command. This can only occur when using the .EE command.
7010	BATOffLine. The event generator indicates the BAT clock signal is not present or broken.
7011	EGFIFOError. An event generator FIFO memory fault occurred.
7012	EGPreambleError. The system failed to find the preamble to the event generator version/serial number string.
7013	EGSerNumTooLong. The event generator version/serial number string was longer than expected.
7014	ETDTooLong. The given Event Timing Description was too long. This can only occur when using the .LT command.
7015	BadETDDesc. A error occurred while the .LT command was parsing an Event Timing Description. This can only occur when using the .LT command.
7016	NoEventGen. No event generator is present in the system.
7018	NonexistentModule. The addressed Correlator Module is not present.
7019	IllegalModuleCombination. Only one Correlator Module can be selected for the command.
701A	DelayUnitOwned. A delay unit in the given delay unit block is already assigned. This can only occur when using the .DB command.
701B	UnknownDelayTable. The given delay unit block index has not be defined with the .DB command.
701C	OutOfDelayBulkTables. All available delay unit blocks have been used. This can only occur when using the .DB command.
701D	ETDQueueOverflow. The message queue where ETD processing requests are placed is full. This usually indicates previous ETD processing has not been completed. This can only occur when using the .EE command.

2. Correlator Block Commands

The following is a description of the commands which are available only on the Block Control Computer (BCC) system. They are used to control and monitor the Correlator Block.

All of these commands (except .SM) take Correlator Block bus control away from the external transfer bus. When this happens the host computer cannot access the block. The commands should not be executed when there is a possibility that CCC requires the bus.

.AA .AA

Access Accumulator Memory.

.AA • *mode* • *address* • *length*
mode: (**R**, **W**)
address: 0..1FFF
length: 1..2000

This command provides access to the Accumulator Memory of the currently selected Correlator Module(s) and the PC Module. This memory consists of 8K x 24-bit words and is accessed using addresses 0 through $1FFF_{16}$. The *address* argument provides the start address for the data transfer. The *length* argument is the number of words to be transferred. This memory resides in the lower part of the total module memory all of which can be accessed using the .AM command.

The *mode* argument determines whether the data transfer is to or from the Accumulator Memory. If *mode* is **R** then data is transferred from the Accumulator Memory to the host by an output data block. If *mode* is to **W** then data is transferred from the host to the Accumulator Memory by an input data block. In either case the data block consists of *length* 24-bit unsigned hexadecimal integers. Any combination of Correlator Modules or the PC Module by itself may be selected for a write operation, but only any single module may be selected for a read. See the .SM command below

Examples:

To transfer the contents of the first 400_{16} locations in the Accumulator Memory to the host computer using an output data block the following command would be used:

```
.AA • R • 0 • 400
```

To clear the first two Accumulator Memory locations the following command and input data block would be used.

```
.AA • W • 0 • 2
0
0
~
```

.AM .AM

Access module memory.

.AM • *mode* • *address* • *length*
mode: (**R**, **W**)
address: 0..FFFF
length: 1..10000

This command provides access to the memory of the currently selected Correlator Module(s) and the PC Module. This includes the Accumulator Memory, Address Translation Table (ATT) memory, various registers, and several empty spaces. The memory is organised as of 32K x 16-bit words. The *address* argument provides the start address for the data transfer. For historical reasons byte addresses 0 to $FFFF_{16}$ are used. The least significant bit of *address* is discarded. The *length* argument is the number of words to be transferred.

The *access-mode* argument determines the direction of the transfer. If *mode* is **R** then data is transferred from the Module Memory to the host by an output data block. If *mode* is **W** then data is transferred from the host to the Module Memory by an input data block. In either case the data block consists of *length* 16-bit words. Any combination of Correlator Modules or the PC Module by itself may be selected for a write operation, but only any single module may be selected for a read. See the .SM command below.

. BA .BA

Set block autocorrelation mode.

.BA • *group0-sel* • *group1-sel*
group0-sel: (**C**, **X**, **Y**, **S**)
group1-sel: (**C**, **X**, **Y**, **S**)

This command controls the gating of the four input IFs coming into the Block. The four IFs are grouped into pairs which are each capable of being switched for autocorrelation. The *group0-sel* parameter controls IF lines 0 and 1. The *group1-sel* parameter controls IF lines 2 and 3. A value of **C** puts that group into cross-correlation mode. A value of **X** or **Y** switches that input channel onto both of the block backplane channels giving autocorrelation of that input channel. A value of **S** gives cross-correlation of the two input channels but they are swapped as they go onto the block backplane.

.ER .ER

Enable/disable pseudo-random data source.

.ER • *enable*
enable: (**Y**, **N**)

A pseudo-random data source may be switched on to provide the Correlator Block with random data for test purposes. If *enable* is equal to **N** then the data source will be switched off and the data for the Correlator Block will come from the data input. If *enable* is equal to **Y** then the pseudo-random data source will be switched on.

.MS .MS

Master sync select.

.MS • *master-sync*
master-sync: 0..4

This command selects the master sync for the phase locked loop circuit on the Block Distributor. This can be one of four external clock signals associated with the IF inputs coming from the delay unit, or an internal oscillator. This command selects which of these five sources is used. Setting the source to 0-3 selects one of the 4 external sources while 5 selects the internal oscillator.

.MT .MT

Perform memory test on module memory.

.MT • *memory*
memory: 0..2

This command performs a memory test on memory in the currently selected Correlator Module. Only one module may be selected. The memory tested depends upon the *memory* parameter. The valid values for *memory* are:

0	Accumulator Memory
1	Address Translation Table Memory
2	Event Generator Translation Table Memory

Several memory tests are performed on the selected memory. The test results are returned in real time as the tests are completed. If a test fails the results are returned and then the next test is attempted. The tests are:

Zeros test	All memory is set to zero then read and checked.
Ones test	All memory is set to one then read and checked.
Address test	Each memory location is set to its address then all memory is read and checked.
Walking ones test	Write first memory location with increasing integral powers of two (walking one) each time reading back and checking. Then repeat for all other memory locations.
Alias test	Write all memory with zeros, then for each address on an address bit boundary all ones are written and all of memory is checked to see that the value does not occur anywhere else.

The command returns an output data block containing five lines of text. These are:

zeros-success [• *zeros-fail-address* • *zeros-fail-data*]
ones-success [• *ones-fail-address* • *ones-fail-data*]
address-success [• *address-fail-address* • *address-fail-data*]
walking-ones-success [• *walking-ones-fail-address* • *walking-ones-data* • *walking-ones-fail-data*a]
alias-success [• *alias-test-address*s • *alias-fail-address*]
where:

zeros-success, ones-success, address-success, walking-ones-success, alias-success: (**T**, **F**), equal to **T** if the corresponding test was successful, or **F** is the test found a fault.

zeros-fail-address, ones-fail-address, address-fail-address, walking-ones-fail-address, alias-fail-address: 0..FFFF, The address at which the corresponding test failed. Not present if test was successful.

zeros-fail-data, ones-fail-data, address-fail-data, walking-ones-fail-data, alias-fail-data: The data read from the *fail-address* which caused the test to fail. Not present if the test was successful.

walking-ones-data: The data written to the *fail-address* when the test failed. Not present if the test was successful.

walking-test-address: The address where all ones were written when the test failed. Not present if the test was successful.

.PA .PA

Access Address Translation Table (ATT) data.

.PA • $ *mode* [: *file-name*]
mode: (**D**, **G**, **L**, **R**, **W**, **Z**)
file-name: *file-name-type*

This command is used to access Address Translation Table (ATT) data. A system of memory resident *files* can be used hold ATT data. The ATT can be written directly from the contents of a file thus eliminating the need for an input data block and greatly reducing the needed processing. Command modes are available for reading and writing the files, for transferring their contents to the ATT, and for listing all known files.

The *mode* argument determines what action is taken and whether the *file-name* argument is required. The possible values for *mode* are:

D: direct. An input data block containing ATT data is expected. Its contents are written into the ATT of the selected Correlator Module(s) immediately. The *file-name* argument is not required but if given the data is also written there.

G: get. If the *file-name* argument is present then it contents are returned in an output data block. If the *file-name* argument is not present then the ATT of the selected Correlator Module is read and the contents returned in an output data block.

L: list. A list of all ATT data files is returned in an output data block. The *file-name* argument is not required.

R: read. The ATT of the selected Correlator Module(s) is written from the ATT programming file given by *file-name*. The *file-name* argument is required.

W: write. An input data block containing ATT data is expected. Its contents are written to the file given by *file-name*. This mode does not alter the contents of the ATT. The *file-name* argument is required.

Z: default. The ATT of the selected Correlator Module(s) is written with system default data. The *file-name* argument is not required.

An input or output data block for the .PA command defines the contents of the ATT. It consists of 1024 lines each containing a 16-bit unsigned hexadecimal number. The first word of the data section corresponds to the lowest memory location of the ATT.

Any combination of Correlator Modules may be selected for a write operation, but only any single module may be selected for a read. See the .SM command below.

A directory listing output data block (obtained when *mode*=**L**) consists of a number of lines each one corresponding to a directory entry. The format of the lines is as follows:

date • *time* • *count* • *file-name*

The *date* and *time* fields indicate when the file was last accessed. The format of *date* is "dd/mm/yy" giving the day, month and year, respectively. The format of *time* is "hh:mm:ss" giving the hour, minute, and second, respectively. The *count* field gives the number of times the file has been accessed. The *file-name* field can be any number of characters in length and contains the name of the file.

.PD .PD

Access DELAY programming data.

.PD • $ *mode* [: *file-name*]
mode: (**D**, **G**, **L**, **R**, **W**)
file-name: *file-name-type*

This command is used to access DELAY chip programming data. A system of memory resident *files* can be used hold DELAY data. The DELAY chip can be written directly from a file thus eliminating the need for input data block and greatly reducing the processing. Command modes are available for reading and writing the files, for transferring their contents to the DELAY chip, and for listing all known files.

The value of *mode* determines what action is taken and whether the *file-name* argument is required. The possible values for *mode* are:

D: direct. An input data block containing DELAY data is expected. Its contents are used to program the DELAY chips of the selected Correlator Module(s) immediately. The *file-name* argument is not required but if given the data is also written there.

G: get. If the *file-name* argument is present then it contents are returned in an output data block. If the *file-name* argument is not present then the DELAY chips of the selected Correlator Module are read and the contents returned in an output data block.

L: list. A list of all DELAY programming files is returned in an output data block. The *file-name* argument is not required.

R: read. The DELAY chips of the selected Correlator Module(s) are written from the DELAY programming file given by *file-name*. The *file-name* argument is required.

W: write. An input data block containing DELAY programming data is expected. Its contents are written to the file given by *file-name*. This mode does not alter the contents of the DELAY chips. The *file-name* argument is required.

An input or output data block for the .PD command defines the DELAY programming data. It consists of 36 lines each containing a 7-bit unsigned hexadecimal number. The first 18 lines contains the programming information for DELAY chip 0. The second 18 lines contains the programming information for DELAY chip 1. For a description of the format of the 18 data lines corresponding to a DELAY chip see the description of the .PX command below.

Any combination of Correlator Modules may be selected for a write operation, but only any single module may be selected for a read. See the .SM command below.

The directory listing output data block (obtained when *mode*=**L**) has the same format as described under the .PA command above.

.PE .PE

Access Event Generator Translation Table (EGTT) data.

.PE • $ *mode* [: *file-name*]
mode: (**D**, **G**, **L**, **R**, **W**, **Z**)
file-name: *file-name-type*

This command is used to access Event Generator Translation Table (EGTT) data. A system of memory resident *files* can be used hold EGTT data. The EGTT can be written directly from a file thus eliminating the need for input data block and greatly reducing the processing. Command modes are available for reading and writing the files, for transferring their contents to the EGTT, and for listing all known files.

The *mode* argument determines what action will be taken and whether the *file-name* argument is required. The possible values for *mode* are:

D: direct. An input data block containing EGTT data is expected. Its contents are written into the EGTT of the selected Correlator Module(s) immediately. The *file-name* argument is not required but if given the data is also written there.

G: get. If the *file-name* argument is present then it contents are returned in an output data block. If the *file-name* argument is not present then the EGTT of the selected Correlator Module is read and the contents returned in an output data block.

L: list. A list of all EGTT data files is returned in an output data block. The *file-name* argument is not required.

R: read. The EGTT of the selected Correlator Module(s) is written from the EGTT programming file given by *file-name*. The *file-name* argument is required.

W: write. An input data block containing EGTT data is expected. Its contents are written to the file given by *file-name*. This mode does not alter the contents of the EGTT. The *file-name* argument is required.

Z: default. The EGTT of the selected Correlator Module(s) is written with system default data. The *file-name* argument is not required.

An input or output data block for the .PE command defines the contents of the EGTT. It consists of 2048 lines each containing a 8-bit unsigned hexadecimal number. The first word of the data section corresponds to he lowest memory location of the EGTT.

Any combination of Correlator Modules may be selected for a write operation, but only any single module may be selected for a read. See the .SM command below.

The directory listing output data block (obtained when *mode*=**L**) has the same format as described under the .PA command above.

.PM .PM

Reads from or writes to the Multibeam Correlator registers.

.PM • *address* • [*data0*] • [*data1*] • [*data2*] •........[*data31*]
address: Base address of Correlator Module
data0 - dataN: Register address and data to load into register.

This command reads from or writes to the registers of the designated Correlator Module. To Write the command must supply the address and at least 1 register/address value. The data is made up of 2 fields: bits 0 - 15 contain data, bits 16 - 20 contain the register number and bits 21 - 31 are 0; for example, the data 123456, puts the data 3456 into register 12.

To read supply only the Correlator Module base address. An output data block is returned with 33 entries. Same data encoding is used as with the write. the 33rd entry is the serial number and it takes the register number 3F.

.PR .PR

Set EPS or MCL registers in all Correlator Modules.

.PR • *mode* • *data0* • *data1* • *data2* • *data3* ·*data4* ·*data5* ·*data6* ·*data7*
mode: (**E**, **M**)
data0: *datum for module 0*
data1: *datum for module 1*
...
data7: *datum for module 7*

This command sets the contents of either the EPS or MCL registers in all Correlator Modules at once. The is much faster than performing the equivalent series of .SM and .AM commands. A mode of **E** indicates the EPS registers are to be written, **M** indicates the MCL registers. If any datum has bits set in the upper word, ie is greater than 65,535, the corresponding module is NOT written.

.PX .PX

Access XCELL programming data.

.PX • $ *mode* [: *file-name*]
mode: (**D**, **G**, **R**, **L**, **W**)
file-name: *file-name-type*

This command is used to access XCELL programming data. A system of memory resident *files* can be used hold XCELL data. The XCELL can be written directly from a file thus eliminating the need for input data block and greatly reducing the processing. Command modes are available for reading and writing the files, for transferring their contents to the XCELL, and for listing all known files.

The value of *mode* determines what action will be taken and whether the *file-name* argument is required. The possible values for *mode* are:

D: direct. An input data block containing XCELL data is expected. Its contents are used to program the XCELL chips of the selected Correlator Module(s) immediately. The *file-name* argument is not required but if given the data is also written there.

G: get. If the *file-name* argument is present then it contents are returned in an output data block. If the *file-name* argument is not present then the XCELL chips of the selected Correlator Module are read and the contents returned in an output data block.

L: list. A list of all XCELL programming files is returned in an output data block. The *file-name* argument is not required.

R: read. The XCELL chips of the selected Correlator Module(s) are written from the XCELL programming file given by *file-name*. The *file-name* argument is required.

W: write. An input data block containing XCELL programming data is expected. Its contents are written to the file given by *file-name*. This mode does not alter the contents of the XCELL chips. The *file-name* argument is required.

An input or output data block for the .PX command defines the XCELL programming data. It consists of 288 lines each containing a 7-bit unsigned hexadecimal number. The first 18 lines contain the programming data for XCELL chip 15. The last 18 lines contains the programming data for XCELL chip 0. The format of the 18 numbers corresponding to a single XCELL are:

Word	**Stage**	**Program bit function**						
		6	**5**	**4**	**3**	**2**	**1**	**0**
0	SwDelayY	ZD	SD2	SD1	SD0	X	X	X
1	VarDelayY1	ZD	Casc	VD4	VD3	VD2	VD1	VD0
2	VarDelayY2	ZD	Casc	VD1	VD0	X	X	X
3	VarDelayY3	ZD	Casc	VD2	VD1	VD0	X	X
4	VarDelayY4	ZD	Casc	VD1	VD0	X	X	X
5	VarDelayY5	ZD	Casc	VD4	VD3	VD2	VD1	VD0
6	VarDelayY6	ZD	Casc	VD1	VD0	X	X	X
7	VarDelayY7	ZD	Casc	VD2	VD1	VD0	X	X
8	VarDelayY8	ZD	Casc	VD1	VD0	X	X	X
9	SwDelayX	ZD	SD2	SD1	SD0	X	X	X
10	VarDelayX1	ZD	Casc	VD4	VD3	VD2	VD1	VD0
11	VarDelayX2	ZD	Casc	VD1	VD0	X	X	X
12	VarDelayX3	ZD	Casc	VD2	VD1	VD0	X	X
13	VarDelayX4	ZD	Casc	VD1	VD0	X	X	X
14	VarDelayX5	ZD	Casc	VD4	VD3	VD2	VD1	VD0
15	VarDelayX6	ZD	Casc	VD1	VD0	X	X	X
16	VarDelayX7	ZD	Casc	VD2	VD1	VD0	X	X
17	VarDelayX8	ZD	Casc	VD1	VD0	X	X	X

Any combination of Correlator Modules may be selected for a write operation, but only any single module may be selected for a read. See the .SM command below.

The directory listing output data block (obtained when *mode*=**L**) has the same format as described under the .PA command above.

.SM .SM

Select module(s).

.SM • [*modnum* [• *modnum* [• *modnum* . . .]]]
modnum: 0..7

This command selects the indicated module(s) for access by subsequent commands. If more than one module number is given then all those modules are selected and only write operations may be carried out on the modules

.ST .ST

Return the status of the Correlator Block.

.ST • *modnum*
modnum: 0..C

This command returns the current status of some aspect of the block. Values of $0_{16}..7_{16}$ correspond to Correlator Modules 0 through 7, 8_{16} to the PC Module, 9_{16} to the block distributor, A_{16} to the Block Control Computer itself, B_{16} to the event generator, and C_{16} to all the Correlator Modules. The returned output data block contains one or more lines of which describe a particular aspect of the hardware. The number and interpretation of the lines depend upon the piece of hardware for which the status information is requested.

If *modnum* is in the range $0_{16}..7_{16}$ (Correlator Modules) the returned data block contains the following line of data:

reconfiguration • *delay-status* • *correlation-mode* • *channel-select* • *bit-mode* •
sel-x • *sel-y* • *eg-filter* • *att-page* • *error-status* • *recirc* • *serial-number*

where:

reconfiguration: 0..7, the reconfiguration of the module being tested.

delay-status: (**N**, **F**, **I**, **L**), the delay status of the module. A value of **N** means that it has no delaying function. A value of **F** means that it is first in a concatenated set. A value of **B** means that it is the intermediate element of a concatenated set. A value of **L** means that it is the last of a concatenated set.

correlation-mode: (**A**, **C**), the correlation mode of the modules. A value of **A** means that the module is in autocorrelation mode. A value of **C** means that the module is in cross-correlation mode.

channel-select: (**X**, **Y**), the selected channel for autocorrelation and module concatenation.

bit-mode: 0, 1, the bit mode (1 or 2) of the module.

sel-x: 0..3, the backplane IF channel selected for the module X channel.

sel-y: 0..3, the backplane IF channel selected for the module Y channel.

eg-filter: 0..FF, the event generator filter value.

att-page: 0..7, the currently selected ATT page number.

error-status: 0..7FF, the error status register of the module.

serial-number: 0..FFF, the serial number of the module from the serial number PROM.

If *modnum* is 8_{16} (PC Module) the return data block contains the following line of data:

NOT IMPLEMENTED

If *modnum* is 9_{16} (Block Distributor) the return data block contains the following line of data:

NOT IMPLEMENTED

If *modnum* is A_{16} (Block Control Computer), the return data block contains the following line of data:

selected-modules • *modules-present* • *terminator* • *block-id* • *module-error* • *serial-number*

where:

selected-modules: set of 0..8, the currently selected modules. Bits 0..7 indicate which Correlator Modules are currently selected. Bit 8 is high if the PC Module or Block Terminator is currently selected.

modules-present: set of 0..9, the modules installed in the block. Bits 0..7 indicate which Correlator Modules are present in the block. Bit 8 is high if the PC Module or Block Terminator is present. Bit 9 is high if the Block Distributor is present.

terminator: (**T**, **F**), equal to **T** if the PC Module is a terminator board.

block-id: 0..FF, the Block Id for the Sky Transfer bus as set on the backplane.

module-error: (**T**, **F**), equal to **T** if one of the modules with error flag enable is indicating an error.

serial-number: 0..FFFF, the serial number of the switch board from the serial number PROM.

If *modnum* is B_{16} (event generator), the return data block contains the following line of data:

NOT IMPLEMENTED

If *modnum* is C_{16} (all Correlator Modules) the return data block contains a line for each Correlator Module which is present. Each line contains the following data:

module-number • *reconfiguration* • *delay-status* • *correlation-mode* • *channel-select* • *bit-mode* • *sel-x* • *sel-y* • *eg-filter* • *att-page* • *error-status* • *recirc* • *serial-number*

where:

module-number : 0..7, the module number to which the remainder of the line applies.

The remaining line fields are as described above under Correlator Modules.

3. Delay Unit Commands

The following is a description of the commands which are available only on the Delay Unit Control Computer (DUCC). They are used to control and monitor the delay units and recirculators.

There are two methods of accessing the delay units. The simplest way is to write delay setting information to individual delay units using a single command for each unit. The commands for accessing individual delay units are .DD for setting the delay setting, .MF for setting the mode, and .SD for returning the status.

The second method of accessing the delay units is to write delay settings to a previously defined block of delay units using one command. This reduces command execution and communications traffic overhead. Presently up to the number of delay units of such blocks can be defined. The commands for accessing a block of delay units are .DB for defining the block, .DM for setting the modes, .DP for setting the delay setting and .DS for returning the status.

.DB .DB

Define multiple delay units.

.DB • [*block*]
block: 0..17

This command defines which delay units will be used in subsequent block data unit manipulations. An input data block following the command is used to define the delay units. Each of the lines contains a number in the range 0..17 which defines one delay unit. Checking is done to ensure that delay units are present and do not appear in any other delay unit block.

The information in the input data block is stored for later use by the .DS, .DM and .DP commands. These commands act on the delay units defined in the .DB command. For example, if the following .DB command is executed:

```
.DB 0
5
6
7
15
~
```

Then a subsequent .DP 0 command would accept a data block containing 4 data elements representing the delay setting for delay units 5, 6, 7, and 21_{10} respectively.

.DD .DD

Set delay unit data.

.DD • *delay-unit* • *delay*
delay-unit: 0..17
delay: 0..FFFF

This command sets the delay of a single delay unit. The delay setting takes effect on the next integration cycle.

.DM .DM

Set mode of a block of delay units.

.DM • [*block*]
block: 0..17

This command sets the modes of the block of delay units previously defined by a .DB command. An input data block is required which contains delay unit mode values (one per line) in one to one correspondence to the delay unit numbers given in the block. The modes of the delay units defined by the .DB command are then set in to the values given in this command. The mode settings come into effect at the beginning of the next integration cycle. The *block* argument specifies one of the delay unit blocks defined by the .DB command.

.DP .DP

Set the delay setting of a block of delay units.

.DP • [*block*]
block: 0..17

This command sets the delay settings of the block of delay units previously defined by a .DB command. An input data block is required which contains delay unit delay setting values (one per line) in one to one correspondence to the delay unit numbers in the block. The delay settings of the delay units defined by the .DB command are then set in to the values given in this command. The delay settings come into effect at the beginning of the next integration cycle. The *block* argument specifies one of the delay unit blocks defined by the .DB command.

.DS .DS

Return the status of delay units/recirculators.

.DS • mode • [*block*]
mode: (**A**, **B**, **R**)
block: 0..17

This command returns the status of some aspect of the delay units and/or recirculators. The value of *mode* determines what action will be taken and whether the *block* argument is needed. The returned output data block contains one or more lines of which describe a particular aspect of the hardware. The number and interpretation of the lines depend upon the piece of hardware for which the status information is requested. The possible values for *mode* are:

A: Report all delay and recirculators unit numbers which are present. To be reported the delay units must either belong to the caller or not be allocated. The output data block contains the following two lines of data:
D [*delay-unit*] • ... • [*delay-unit*]
R [*recirculator-unit*] • ... • [*recirculator-unit*]
where:
delay-unit: 0..23, delay unit number.

B: Report the status of the block of delay units previously defined by a .DB command. The output data block contains a number of text lines each corresponding to the status of a single delay unit in the block. There is one additional line which contains the times at which the status was read and the delays were last written. The *block* argument is required and specifies one of the delay unit blocks defined by the .DB command. Each delay unit status line consists of the following:
bin-status ·o delay-unit-status ·o delay-unit-mode ·o delay-unit-setting
where:
bin-status: 0,1, the current status of the bin associated with the delay unit specified by *delay-unit*.
delay-unit-status: 0..3, the current status of the delay unit specified by *delay-unit*.
delay-unit-mode: 0..3, the current mode of the delay unit specified by *delay-unit*.
delay-unit-setting: 0..FFFF, the current delay setting of the delay unit specified by *delay-unit*.

The *bin-status* and *delay-unit-status* values are the status values latched at the end of the previous integration cycle.

The last line contains the following data:
status-bat o delay-bat
where:

current-bat: 0..FFFF FFFF FFFF, the current time.

delay-bat: 0..FFFF FFFF FFFF, the time the delays were last written. This refers to the last .DP command, not the last .DD command..

R: Report all delay unit numbers which are present. To be reported the delay units must either belong to the caller or not be allocated. The output data block contains a line for each delay unit reported.

.ED .ED

Access event generator distribution module (Z22).

.ED • *mode* • *address* • *length*
mode: (**R**, **W**)
address: 0..11
length: 1..12

This command accesses the memory of the event generator distribution module. The *address* argument provides the start address for the data transfer. The *length* argument is the number of words to be transferred. The possible values for *mode* are:

R: read. Read the specified memory and return the results in an output data block.

W: write. Write the specified memory with the contents of the following input data block.

.MF .MF

Set delay unit mode.

.MF • *delay-unit* • *mode*
delay-unit: 0..17
mode: 0..3

This command sets the mode of a single delay unit. The mode takes effect on the next integration cycle.

.RM .RM

Set the recirculating mode and the event signal source of a Recirculating Channel.

.RM • *recirculator-unit* • *mode* • *source*
recirculator-unit: 0..17
mode: 0..3
source: 0..3

The recirculator is used when the bandwidth of the signal is less than 8 MHz (1-bit sampling) or 4 MHz (2-bit sampling).

An event generator signal is required for synchronisation. This signal can be one of the four event signals (EG0 ... EG3) on the delay unit backplane as specified by the *source.*

Example:

```
.RM 2 1 0
```

Sets recirculator unit 2 to mode 1 (1 MHz bandwidth in 2-bit sampling mode) with EG0 as the synchronisation signal.

.RO .RO

Access recirculator channel offset data.

.RO • *mode* • *recirc-unit* • [*length*]
mode: (**R, W**)
recirc-unit: 0..17
length:: 0..10

This command loads the offset data into the specified Recirculator Channel. There are sixteen 16 bit words in each Channel where the offset data is loaded. Each recirculating mode requires a different number of offset data. Mode 0 requires two words, mode 1 requires four words, mode 2 requires eight words, and mode 3 all sixteen words. Offset data must be loaded before running the Recirculator, and because it will be in the “set” state after executing the .RO command must always be followed by the .RM command. This will turn the recirculator channel into the “run” state.

The offset data consists of 1 sign bit and 15 magnitude bits. The most-significant bit is the sign bit and when set indicates the value is negative. The remaining 15 bits are the absolute delay value. The actual value passed in the command is the number of samples divided by 8.

When a read operation is performed the returned information takes the following form:

```
offset-data
.
.
~
```

where: offset-data: 0 .. FFFF, the offset data stored in the corresponding location.

Examples:

```
.RO W 3 4
0
80
8080
100
~
```

Load four words of offset data to Recirculator channel 3. This will produce actual delay offsets of 0, +1024, -1024, +2048 samples.

```
.RM 3 1 0
```

Set the channel running in mode to 1.

```
.RO R 3
```

Read back all sixteen words of the offset data from Recirculator Channel 3.

.SD .SD

Return delay unit status.

.SD • *delay-unit*
delay-unit: 0..17

This command returns an output data block containing the status of a single delay unit. If *delay-unit* is not present then the data block contains information on the delay system and has the following form:

delays-present
where:

delays-present: 0..FFFF, a bit-map of the currently present delay units in the delay system. High bits in *delays-present* correspond to delay units which are present.

If *delay-unit* is present then the data block contains data on one of the delay units and is of the following form:

bin-status-o delay-unit-status o delay-unit-mode o delay-unit-setting
where:

bin-status: 0..3, the current status of the bin associated with *delay-unit*.

delay-unit-status: 0..3, the current status of the *delay-unit*.

delay-unit-mode: 0..3, the current mode of the *delay-unit*.

delay-unit-setting: 0..FFFF, the current delay setting of the *delay-unit*.

The *bin-status* and *delay-unit-status* values are the status values latched at the end of the previous integration cycle.

4. Event Generator Commands

The event generator is a custom hardware device designed and built at the ATNF. It can generate up to 16 timing signals and a clocking signal. It includes a frame grabber used to obtain information from the Binary Atomic Time (BAT) clock, aka the Hunt clock. The following is a description of the commands which are available on both the Block Control Computer (BCC) and Delay Unit Control Computer (DUCC) systems when equipped with this device.

To control the event generator's timing signals an **Event Timing Description** (ETD) is used. An ETD is a recursive data structure which defines a stream of events for the event generator. An ETD acts like a program which is followed to generate the data (time and events) to send to the event generator. It consists of the ETD commands listed below. Of special importance is the Sequence command which allows an ETD to be recursive. It is used to create loops within an ETD.

The ETD processor contains 8 event and 8 time registers which can be manipulated by the ETD. Each event register holds a 16-bit unsigned number, and each time register hold a 64-bit unsigned number. The registers are accessed through the processing of the ETD. Event register 0 has special meaning in the ETD commands and is referred to as the accumulator. The accumulator can also be accessed by specifying $0 in the same way as the other registers. There is also a carry register which is used in processing the *Increment* command. It can be cleared using the *ClearCarry* command.

The input data block used to define an ETD consists of a number of text lines. Each line contains a command and any needed arguments separated by spaces or tabs in the same way as command lines.

All specifications of time in an ETD definition are of a data type called *reduced-BAT* . The integer part of a reduced-BAT time represents the least significant 48-bits of atomic time in microseconds. It can contain up to 12 hexadecimal digits. The integer part may optionally be followed by a decimal point (".") and up to 8 hexadecimal digits giving the microsecond fraction. Actual event times are truncated to the nearest microsecond but the fractional part is retained to minimize accumulated errors.

All specifications of events in an ETD definition are of a data type called *event-def*. Such a value can take two forms: It can be a literal consisting of a hexadecimal number in the range 0..FFFF, or it can be the contents of an event register. An event register is specified by a "$" character followed by a hexadecimal number in the range 0..8. For example, $6 refers to register 6. For compatibility with previous software versions literals maybe prefixed with the # character.

The time offset argument of the event definition (‘e’) and sequence (‘s’) commands, and the period argument of the sequence (‘s’) command can also take two forms. They can be a reduce-BAT literal, or the contents of a time register. A time register is specified by a "$" character followed by a hexadecimal number in the range 0..8. For example, $6 refers to register 6.

Examples:

This ETD will generate a 1Hz square wave with 50% duty cycle on the least significant bit of the event generator output. The square will be generated for 32767_{10} complete cycles starting at the time given in the .EE command.

```
S • 0 • 2 • FFFE • 7A120
X • 0001
E • 0 • $0
```

This ETD will generate a 10 millisecond period square wave for the first half of each second, and a 20 millisecond period wave of 8 millisecond width pulses for the second half of each second on output bit 0. Output 1 wil be low for the first half of each second and high for the second half. It will run for 65535 seconds and output bit 1 will be left high.

```
S • 0 • 2 • FFFF • F4240
S • 0 • 2 • 32 • 2710
E • 0 • 1
E • 1388 • 0
S • 7A120 • 2 • 16 • 4E20
E • 0 • 3
E • 1F40 • 2
```

ETD Commands

And:

A.• *and-value*

where:

and-value (event-def) is a number which is ANDed with the current contents of the accumulator and the result is placed back into the accumulator.

This command performs an AND function of *and-value* and the current contents of the accumulator and places the result back into the accumulator.

ClearCarry:

C

This command clears the carry register. The carry register is used in conjunction with the ETD Increment command.

EventDefinition:

E.• *time-offset.*• *event* [• *mask*]

where:

time-offset (reduced-bat) is the offset from the current base time at which this event is to occur.

event (event-def) is a value which is output by the event generator at the time of the event.

mask (event-spec) is a value which defines the output bits of the Event Generator which are to be changed by the EventDefinition.

This is the fundamental definition of an event which is generated at the time calculated by the elements of the ETD in which it is contained. If *mask* is omitted then all bits in event are used. If *mask* is present then bits in *mask* which are low are not altered on the Event Generator output, ie they remain as they were in the previous event. After the event is produced, the accumulator contains the event.

Get:

G.• *value*
where:
value (event-def) is a number which is placed into the accumulator.

This command places the *value* (either a literal or the contents of one of the event registers) into the accumulator.

Increment:

I.• *mask* [• *increment*]
where:
mask (event-def) is a bit pattern which determines the field of the accumulator which is to be accumulated. The argument must contain a single cluster of high bits.
increment (event-def) is the amount by which the *mask* field of the accumulator is to be increased. If *increment* is omitted then it defaults to one.

The value of *increment* and the carry register is added to a bit field in the accumulator determined by the value of *mask*. The result is placed back into the field of the accumulator and the carry is placed into the carry register. The carry register may be cleared using the ETD command ClearCarry.

Using the carry register, non-contiguous fields in the accumulator may be incremented by an arbitrary amount. The carry register is first cleared and then the least significant field is incremented. The rest of the fields are then incremented by zero in order from the least to the most significant.

Not:

N

This command performs an NOT function of the current contents of the accumulator and places the result back into the accumulator.

Or:

O • *or-value*
where:
or-value (event-def) is a number which is ORed with the current contents of the accumulator and the result is placed back into the accumulator.

This command performs an OR function of *or-value* and the current contents of the accumulator and places the result back into the accumulator.

Put:

P • *register*
where:
register (event-def) is a register into which the contents of the accumulator are deposited. The argument must be in the register form of *event-def.*

This command places the contents of the accumulator into one of the event registers.

Sequence:

S • *time-offset* • *elements* • *repetitions* • *period*
where:

time-offset (reduced-BAT) is the offset from the current base time at which the sequence is to begin.
elements (0..FFFF) is the number of ETDs following this line which form the list to be executed.
repetitions (event-def) is the number of times the following list of ETDs is to be repeated. Note that this argument is of type *event-def*, so that the number or repetitions can be independent of the actual ETD loaded, ie it can be retrieved from a register.
period (reduced-BAT) is the time to be taken for one repetition. This must be equal to or greater than the time actually taken to execute all ETDs of the sequence once.

The Sequence command provides the control structure for the ETD. Following it should be a list of *elements* ETDs which are executed sequentially *repetitions* times.

When an ETD is found to consist of a sequence, a base time variable is created. The initial value for the base time is the current value for the inherited base time plus the value of the *time-offset* for this sequence. Each time the sequence is repeated, the value of the *period* argument is added to the base time. The base time is used as the base for the *time-offset* arguments of all the ETDs within the list. The execution of a sequence does not alter the value of the inherited base time. The inherited base time at the top level of the ETD comes from the ETD execution command.

XOr:

X • *xor-value*
where:
xor-value (event-def) is a number which is XORed with the current contents of the accumulator and the result is placed back into the accumulator.

This command performs an XOR function of *xor-value* and the current contents of the accumulator and places the result back into the accumulator.

.EE .EE

Execute an Event Timing Description (ETD).

.EE [• *etd-spec* [• *start-time*]]
etd-spec: 0..50_{10}
start-time: 0..FFFFFFFFFFFF [.0..FFFFFFFF]

The .EE command executes a Event Timing Description (ETD) which was previously loaded using the .LT command. The *start-time* argument specifies the time (in reduced BAT with optional fractional part) at which the ETD is to begin execution. If the *start-time* argument is not present then the ETD begins approximately 1 second after the command is received. The *etd-spec* argument specifies which ETD buffer is to be used. If it is not given it is assumed to be zero. An ETD which has been loaded into memory may be executed any number of times using the .EE command. The .EE command returns the error code immediately and the ETD is executed in the background.

.EI .EI

Initialise the event generator.

.EI

This command initialises the event generator. It clears the event generator outputs, stops the generation of events, and clears the software registers and all ETD buffers. This command should be executed before any commands which deal with the event generator.

.GT **.GT**

Get the current atomic time and accumulated leap seconds (DUTC).

.GT

This command returns the current atomic time and DUTC. The command uses the event generator to grab a frame from the clock and then decodes the time and DUTC from it. The output data block consists of a single line containing a 64-bit unsigned hexadecimal number representing the Modified Julian Date (MJD) in microseconds, followed by a space, followed by a hexadecimal number representing the DUTC.

.LT **.LT**

Load an Event Timing Description (ETD).

.LT [• *etd-spec*]
etd-spec: 0..50_{10}

This command loads an Event Timing Description (ETD). The ETD is contained in an input data block following the command. It contains the control information for the event generator outputs (see the introduction to this section). This command does not execute the ETD but simply parses it into a memory data structure. Once in memory an ETD may be executed using the .EE command which also sets the start time for the ETD. In this way an ETD may be executed any number of times using a different starting time for each run.

A number of ETDs may be present in the system at any time and the *etd-spec* argument specifies which of the ETD buffers is to hold the ETD. If it is not given it is assumed to be zero. If the given ETD buffer contains a previous definition it is overwritten.

5. Utility Commands

The following is a description of utility commands which are available on both the Block Control Computer (BCC) and Delay Unit Control Computer (DUCC) systems.

.EX **.EX**

Execute commands from a disk file.

.EX • *file-name*
file-name: *file-name-type*

This command executes commands taken from the given disk file.

.IP **.IP**

Get or Set the IP address of the node.

.IP ·[*ip-address* ·*subnet-mask*]
ip-address: ip address in 'dot' notation

subnet-mask: subnet mask in ‘dot’ notation

If no parameters supplied, commands returns the IP address, and subnet mask in a standard data block.

This command executes commands taken from the given disk file.

.TI .TI

No operation.

.TI • <any number of dummy arguments>

This command does nothing except return a zero error code. Previously this command set the system time.