SHF Communication Technologies AG

GPIB Programming Manual

SHF EA 40 GIG Error Analyzer

GPIB PROGRAMMING MANUAL VERSION 1.01

SHF EA 40 GIG 40 GBit/s Error Analyzer

SHF Communication Technologies AG
the bandwidth company

Amalienstrasse 14 • 12247 Berlin
Telephone ++49 30/7 72 05 10 • Fax ++49 30/753 10 78
Email: mail@shf-communication.de
http://www.shf-communication.de

14th December 2001

Contents

Chapter 1

Introduction

This manual gives a general information on how to control the SHF EA 40 GIG using the rear panel GPIB interface. Descriptions of the actual commands for the SHF EA 40 GIG are found in the following chapters.

Programming information is specific to the SHF EA 40 GIG, and assumes that the user is already familiar with the GPIB. If you are not familiar with GPIB, then refer to the following books:

- The International Institute of Electrical and Electronic Engineers. *IEEE Standard 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation.* New York, NY, 1987

- The International Institute of Electrical and Electronic Engineers. *IEEE Standard 488.2-1987, IEEE Standard Codes, Formats, Protocols and Common Commands For Use with ANSI/IEEE Std 488.1-1987.* New York, NY, 1987

To obtain a copy of either of these two documents, write to:

The International Institute of Electrical and Electronic Engineers, Inc.
345 East 47th Street
New York, NY 10017
USA.

Chapter 2

SHF EA 40 GIG GPIB Capabilities

The SHF EA 40 GIG interfaces to the GPIB as defined by the IEEE Standards 488.1 and 488.2. The table shows the functional subset that is implemented for the SHF EA 40 GIG.

Mnemonic	*Function*
SH1	Complete source handshake capability
AH1	Complete acceptor handshake capability
T6	Basic talker; serial poll; unaddressed to talk if addressed to listen; no talk only
L4	Basic listener; unaddressed to listen if addressed to talk; no listen only
SR1	Complete service request capability
RL0	No remote/local capability
PP0	No parallel poll capability
DC1	Device clear capability
DT1	Device trigger capability (accepted but ignored)
C0	No controller capability
E2	Tristate outputs (except the handshake line)

Table 1: GPIB Capabilities

Chapter 3

GPIB Command Syntax

The instrument can be controlled through the GPIB interface using commands and queries. This section describes the syntax of these commands and queries. It also describes the conventions the instrument uses to process them. The section entitled *Command Set* lists the commands and queries themselves.

A set of commands can be used to control the operations and functions of the instrument from the external interface (GPIB).

This manual describes commands and queries using the Backus-Naur Form (BNF) notation given in table 2.

Symbol	*Meaning*
<>	Defined element (e.g., <arg>)
::=	Is defined as (e.g., <arg> ::= argument)
|	Exclusive OR
{ }	One of this group is required
[ ]	Optional item
...	Previous element(s) may be repeated

Table 2: BNF-Symbols

Command Structure

Command language messages are composed of set commands (also referred to as simply *commands*) and query commands (also referred to as *queries*). Set commands tell the instrument to take a specific action. Queries ask the instrument to return information about its state.

Commands are composed of syntactic elements:

<*header*> ::= The command name; if it ends with a question mark, the command is a query.

<*delimiter*> ::= A space, colon (:), comma (,), or semi-colon (;) which brakes the message into segments for the instrument to process.

<*link*> ::= A command sub-function. Not all commands have links.

<*argument*> ::= A quantity, quality, restriction, or limit associated with the header or link.

Figure 1 shows the four syntactic elements.

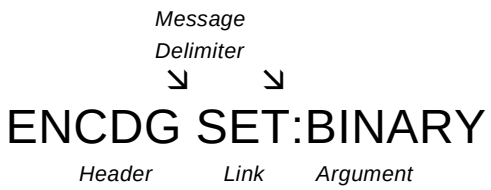

Figure 1: Example of Syntax Elements

The command can be written in upper case or in lower case.

The components can be typed in full, or they can be shortened. The full version is in the command table in both lower and upper case letters. The shortened version of a command is that part which is shown in upper case.

Example: The command

SOURce:POWer:STATe ON

can also be written in lower case as

source:power:state on

or it can be shortened and written as

sour:pow:stat on

Chapter 4

Common Commands

The IEEE 488.2 standard has a list of reserved commands, called common commands. Some of these commands must be implemented by any instrument using the standard, others are optional. The SHF EA 40 GIG implements all the necessary commands. This chapter describes the implemented commands.

Common Status Information

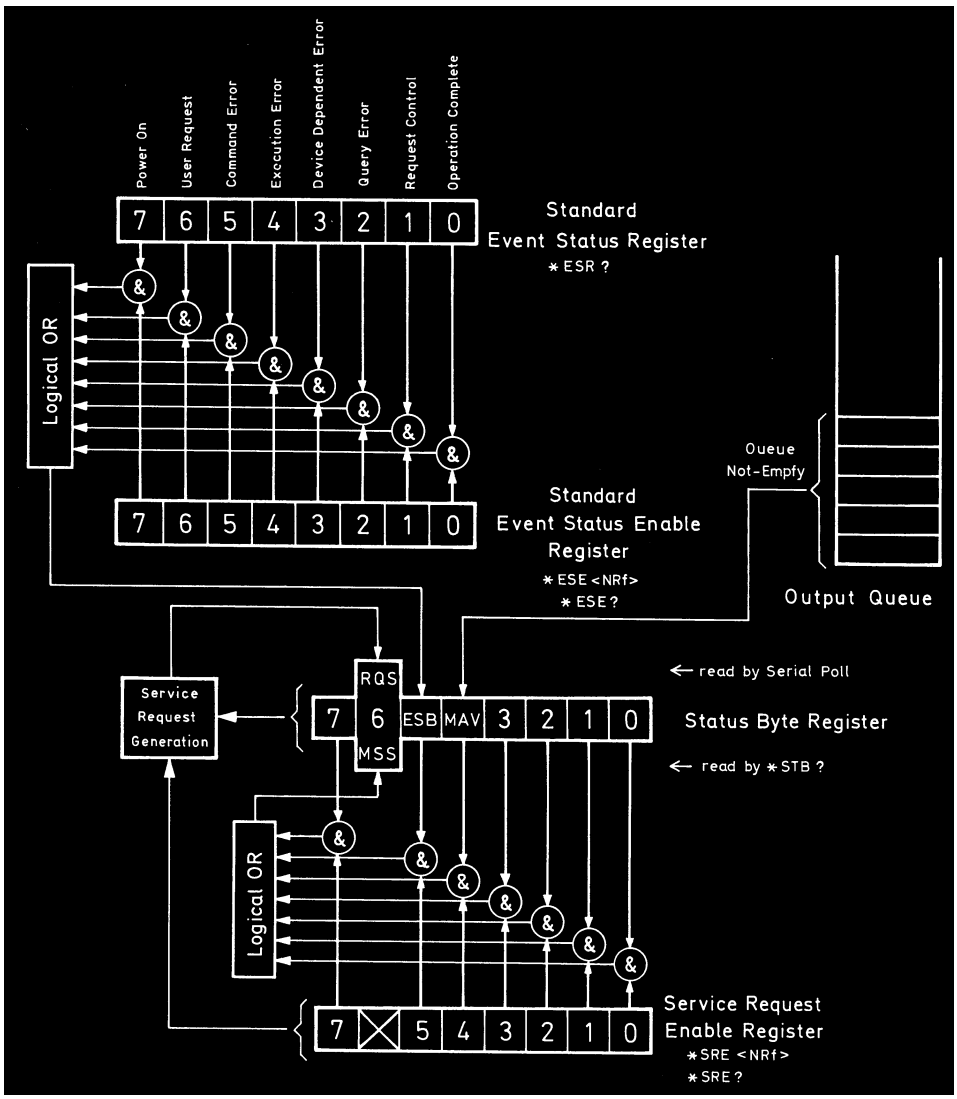

Figure 2: Common Status Registers

There are four registers involved in the status information available from the instrument when you use the common commands. Two of these are status-registers and two are enable-registers. These registers conform to the *IEEE Standard 488.2-1987*. You can find further descriptions of these registers under "*ESE", "*ESR?", "*SRE" and "*STB?".

Figure 2 shows how the registers are organized.

Note

Unused bits in any of the registers return 0 when you read them.

SRQ

A service request (SRQ) is forced when a bit in the Status Byte register goes from 0 > 1 AND the corresponding Service Request Enable Mask bit is set.

If an SRQ is forced, the Request Service (RQS) bit is set to 1. This bit remains at 1 until read by a serial poll, even if the reason or condition that caused the service request no longer exists. Similarly, if a serial poll reads the RQS it is reset to 0, even if the condition that caused the service request still exists. The serial poll command transfers the value of the Status Byte register to a variable.

Input Queue

The input queue is a FIFO queue (first-in first-out) and is 1024 bytes long.

Output Queue

The output queue is a FIFO queue (first-in first-out) and is 10 output messages long. The Message Available (MAV) bit is set in bit four of the Status Byte register whenever the output queue is not empty.

Error Queue

The error queue is a FIFO queue (first-in first-out) and is 30 errors long. That is, the oldest error is the first error to be read.

Common Command Summary

Command	*Function*
***CLS**	Clear Status Command
***ESE**	Standard Event Status Enable Command
***ESE?**	Standard Event Status Enable Query
***ESR?**	Standard Event Status Register Query
***IDN?**	Identification Query
***OPC**	Operation Complete Command
***OPC?**	Operation Complete Query
***RST**	Reset Command
***SRE**	Service Request Enable Command
***SRE?**	Service Request Enable Query
***STB?**	Read Status Byte Query
***TRG**	Trigger Command
***TST?**	Self-Test Query
***WAI**	Wait-to-Continue Command

Table 3: Common Command Summary

***CLS**

Clear Status Command

Syntax: `*CLS`

Definition: The *CLS command clears all the event registers summarized in the Status Byte register. With the exception of the output queue, all queues that are summarized in the Status Byte register are emptied. The error queue is also emptied. Neither the Standard Event Status Enable register, nor the Service Request Enable register are affected by this command. After the *CLS command the instrument is left in the idle state. The command does not alter the

instrument setting. *OPC / *OPC? actions are canceled.

Example

```
OUTPUT 722; "*CLS"
```

***ESE**

Standard Event Status Enable Command

Syntax

```
*ESE < value >
0 ≤ value ≤ 255
```

Definition

The *ESE command sets bits in the Standard Event Status Enable register and thus enables the corresponding bits in the Standard Event Status register (see table 4). A 1 in a bit in the enable register enables the same bit in the status register. The value is sent as an integer.

The register is cleared at power-on. The *RST and *CLS commands do not change the register.

Bits	*Mnemonics*	*Decimal Value*
7	Power On	128
6	Not used	0
5	Command Error	32
4	Execution Error	16
3	Device Dependent Error	8
2	Query Error	4
1	Not used	0
0	Operation Complete	1

Table 4: The Standard Event Status (Enable) Register

Command Errors

They indicate that a syntax error has been detected by the parser, this could be

⇔ A syntax error.

⇔ A semantic error (unrecognized command)

The Command Error bit is set in the Standard Event Status Register.

Execution Errors

They indicate that a syntax error has been detected by the part of the instrument which controls the command execution.

⇔ Data is out of range.

⇔ The command could not be executed because of hardware limitations.

The Execution Error bit is set in the Standard Event Status Register.

Device Dependent Errors

They indicate a failure due to an abnormal hardware or firmware condition. These codes are also used for the results of the self-test. The device dependant error bit is set in the Standard Event Status Register.

Query Errors

They indicate that a problem has been detected in the output queue. This could be:

⇔ Trying to read from a queue while the queue is empty

⇔ or Data lost from the output queue.

The Query Error bit is set in the Standard Event Status Register.

Related Commands `*ESE?`

Example `OUTPUT 722; "*ESE 21"`

***ESE?**

Standard Event Status Enable query.

Syntax `*ESE?`

Response `< value >`
`0 ≤ value ≤ 255`

Definition The *ESE? query returns the contents of the Standard Event Status Enable register (see "*ESE" for information on this register). The value is returned as an integer.

Related Commands `*ESE`

Example
```
OUTPUT 722; "*ESE?"
ENTER 722; A$
```

***ESR?**

Standard Event Status Register query.

Syntax `*ESR?`

Response `< value >`
`0 ≤ value ≤ 255`

Definition The *ESR? returns the contents of the Standard Event Status register (see " *ESE" for information on this register). The register is cleared after being read. The value is returned as an integer.

Example
```
OUTPUT 722; "*ESR?"
ENTER 722; A$
```

***IDN?**

Identification query

Syntax `*IDN?`

Response SHF AG, EA40GIG, < serial number >, Revision 1.0

- SHF AG : manufacturer
- EA40GIG: instrument model number
- < serial number > : serial number; 0: means that serial number is not provided
- 1.0 : firmware revision level

Definition The *IDN? query gets the instrument identification.

Example

```
OUTPUT 722; "*IDN?"
ENTER 722; A$
```

***OPC** Operation Complete Command

Syntax

```
*OPC
```

Definition

The *OPC command parses all program message units in the input queue and sets the operation complete bit in the Standard Event Status register, when the contents of the input queue have been processed.

The following actions cancel the *OPC command (and put the instrument into Operation Complete, Command Idle State):

⇒ Power-on

⇒ *CLS

⇒ *RST

Related Commands

```
*OPC?, *WAI
```

Example

```
OUTPUT 722; "CLS;*ESE 1;*SRE 32"
OUTPUT 722; "*OPC"
```

***OPC?** Operation Complete query

Syntax

```
*OPC?
```

Response

```
< value >
value = 1
```

Definition

The *OPC? command parses all program message units in the input queue, sets the operation complete bit in the Standard Event Status register, and places an ASCII "1" in the output queue, when the contents of the input queue have been processed.

The following actions cancel the *OPC? query (and put the instrument into Operation Complete, Command Idle State):

⇒ Power-on

⇒ *CLS

⇒ *RST

Related Commands `*OPC, *WAI`

Example

```
OUTPUT 722; "*OPC?"
ENTER 722; A$
```

Note

that this query prevents the use of the input queue because it is immediately followed by an ENTER command. The enter command will not execute until it receives data from the output queue.

***RST** Reset Command

Syntax `*RST`

Definition The *RST command sets the instrument to reset setting (standard setting) stored in ROM.

Pending *OPC / *OPC? actions are canceled.

Instrument state: the instrument is placed in the idle state awaiting a command.

The *RST command clears the key queue. The following states are not changed:

- ⇒ GPIB (interface) state
- ⇒ Instrument interface address
- ⇒ Output queue
- ⇒ Service Request Enable register (SRE)
- ⇒ Standard Event Status Enable register (ESE)

The following table lists the commands and parameters of the reset state.

SHF EA 40 GIG Reset State (Standard Setting)
Internal Bitrate 39.98 GBit/s
Clock ratio half
Gating mode repeat
Gating period bits
Gating period 1 GBit
Threshold = 0 mV
2^{31}-1
Polarity CCITT
GPIB address 20
Time to screensave 30 min
Start measure input disabled

Table 5: Reset State (Standard Setting)

Example

```
OUTPUT 722; "*RST"
```

***SRE**

Service Request Enable command

Syntax

```
*SRE < value >
0 ≤ value ≤ 255
```

Definition

The *SRE command sets bits in the Service Request Enable register. A "1" in a bit inside an enable register enables the corresponding bits in the status register. The value is sent as an integer.

The register is cleared at power-on. The *RST and *CLS commands do not change the register.

Bits	Mnemonics	Decimal Value
7	Not used	0
6	Request Service (RQS), Master Summary Status (MSS)	64
5	Event Status Bits (ESB)	32
4	Message Available (MAV)	16
3	Not used	0
2	Not used	0
1	Not used	0
0	Not used	0

Table 6: The Service Request Enable Register / Status Byte Register

Related Commands `*SRE?, *STB?`

Example

```
OUTPUT 722; "*SRE 48"
```

***SRE?**

Service Request Enable query.

Syntax `*SRE?`

Response

```
< value >
0 ≤ value ≤ 255
```

Definition The *SRE? query returns the contents of the Service Request Enable register (see "*SRE" for information on this register). The value is returned as an integer.

Related Commands `*SRE, *STB?`

Example

```
OUTPUT 722; "*SRE?"
ENTER 722; A$
```

***STB?**

Status Byte query.

Syntax

```
*STB?
```

Response

```
< value >
0 ≤ value ≤ 255
```

Definition

The *STB? query returns the contents of the Status Byte register (see " *SRE" for information on this register). The register is programmed via the Service Request Enable Mask. For setting this mask, see " *SRE". The value is returned as an integer.

The Master Summary Status (MSS) bit is true when any enabled bit of the STB register is set (excluding bit 6).

The Status Byte register including the master summary bit, MSS, is not directly altered because of an *STB? query.

Related Commands

```
*SRE, *SRE?
```

Example

```
OUTPUT 722; "*STB?"
ENTER 722; A$
```

***TST?**

Self-Test query

Syntax

```
*TST?
```

Response

```
< value >
value = 0 | 1
A value of zero means no errors.
```

Definition

The *OPC? command parses all program message units in the input queue, sets the operation complete bit in the Standard Event Status register, and places an ASCII "1" in the output queue, when the contents of the input queue have been processed.

The following actions cancel the *OPC? query (and put the instrument into Operation Complete, Command Idle State):

⇒ Power-on

⇒ *CLS

⇒ *RST

Related Commands

```
*OPC, *WAI
```

Example

```
OUTPUT 722; "*TST?"
ENTER 722; A$
```

***WAI** Wait Command

Syntax `*WAI`

Definition The *WAI command prevents the instrument from executing any further commands until the current command has finished executing. All pending operations are completed during the wait period.

Related Commands `*OPC, *OPC?`

Example

```
OUTPUT 722; "*WAI"
```

Chapter 5

Input Commands

This set of commands control the delay line, the external start measurement control and the value of the input threshold on the EA 40 GIG.

Command	*Parameters/Results*
INPut:DELay	<value>
INPut:DELay?	<value>
INPut:THReshold	<value>
INPut:THReshold?	<value>
INPut:MEAsure	{ENABle|DISABle}
INPut:MEAsure?	{ENABle|DISABle}

Table 7: Input Command Summary

INPut:DELay

This input command is used to set the value of the input delay line

Syntax
```
INPut:DELay <numeric value>
-80 ≤ value ≤ 80
```

Example
```
OUTPUT 722; "INP:DEL 78"
```

INPut:CLKdelay?

This input command is used to query the value of the input delay line

Syntax
```
INPut:DELay?
```

Response
```
< value >
-80 ≤ value ≤ 80
```

Example
```
OUTPUT 722; "INP:DEL?"
ENTER 722; A$
```

INPut:THReshold

This input command is used to set the level of the input threshold voltage

Syntax
```
INPut:THReshold <numeric value>
-400 ≤ value ≤ 400
```

Example `OUTPUT 722; "INP:THR 53"`

INPut:THReshold? This input command is used to query the level of the input threshold voltage

Syntax `INPut:THReshold?`

Response `< value >`
`-400 ≤ value ≤ 400`

Example
```
OUTPUT 722; "INP:THR?"
ENTER 722; A$
```

INPut:MEAsure This input command is used to enable or disable the start measurement input on the front of the instrument.

Syntax `INPut:MEAsure` {ENABle|DISABle}

Example `OUTPUT 722; "INP:MEA DISAB"`

INPut:MEAsure? This input command is used to query whether the start measurement input is enabled or disabled.

Syntax `INPut:MEAsure?`

Response `< ENAB | DISAB >`

Example
```
OUTPUT 722; "INP:MEA?"
ENTER 722; A$
```

Chapter 6

Pattern Commands

This set of commands relate to the properties of the PRBS pattern which is detected by the SHF EA 40 GIG.

Command	*Parameters/Results*
PATTern:SELect	{PRBS7|PRBS15|PRBS23|PRBS31}
PATTern:SELect?	{PRBS7|PRBS15|PRBS23|PRBS31}
PATTern:POLarity	{CCITT|INVerted}
PATTern:POLarity?	{CCITT|INVerted}

Table 8: Pattern Command Summary

PATTern:SELect

This command is used to select the type of PRBS pattern to be detected by the analyzer.

Syntax
```
PATTern:SELect
{PRBS7|PRBS15|PRBS23|PRBS31}
```

Example
```
OUTPUT 722; "PATT:SEL PRBS15"
```

PATTern:SELect?

This command is used to query which PRBS pattern is detected by the analyzer.

Syntax
```
PATTern:SELect?
```

Response
```
<PRBS7 | PRBS15 | PRBS23 | PRBS31>
```

Example
```
OUTPUT 722; "PATT:SEL?"
ENTER 722; A$
```

PATTern:POLarity

This command is used to set the polarity of the pattern

Syntax
```
PATTern:POLarity {CCITT|INVerted}
```

Example
```
OUTPUT 722; "PATT:POL INV"
```

PATTern:POLarity? This command is used to query the polarity of the pattern.

Syntax `PATTern:POLarity?`

Response `< CCITT | INV >`

Example

```
OUTPUT 722; “PATT:POL?”
ENTER 722; A$
```

Chapter 7

Gating Commands

This set of commands relates to the gating functions of the EA 40 GIG.

Command	*Parameters/Results*
GATing:MODe	{SINgle|REPeat}
GATing:MODe?	{SINGLE|REPEAT}
GATing:PERiod	{BITS|TIME}
GATing:PERiod?	{BITS|TIME}
GATing:RANge	<value>
GATing:RANge?	<value>

Table 9 Gating Command Summary

GATing:MODe

This command is used to switch the gating mode between a single measurement and repeat measurements.

Syntax `GATing:MODe {SINgle|REPeat}`

Example
```
OUTPUT 722; "GAT:MOD SIN"
```

GATing:MODe?

This command is used to query whether the gating mode is set to single or repeat measurements

Syntax `GATing:MODe?`

Response `< SINGLE | REPEAT >`

Example
```
OUTPUT 722; "GAT:MOD?"
ENTER 722; A$
```

GATing:PERiod

This command switches the gating period between bits and time.

Syntax `GATing:PERiod {BITS|TIME}`

Example
```
OUTPUT 722; "GAT:PER TIME"
```

GATing:PERiod?

This command is used to query whether the gating priod is seet to measure for a given time, or a given number of bits.

Syntax `GATing:PERiod?`

Response `< BITS | TIME >`

Example
```
OUTPUT 722; "GAT:PER?"
ENTER 722; A$
```

GATing:RANge

The gating period command, ie by time/bits should be used first. The numerical value of this command is dependent on the setting of the gating period

Syntax `GATing:RANge <value>`

Example
```
OUTPUT 722; "GAT:RAN "
```

GATing:RANge?

This command is used to query value of the gating range.

Syntax `GATing:RANge?`

Response `< value >`

Example
```
OUTPUT 722; "GAT:RAN?"
ENTER 722; A$
```

Chapter 8

Clock Commands

This chapter covers the commands which are related to the clock on the EA 40 GIG.

Command	*Parameters/Results*
CLOCK:INPut	{INTernal|EXTernal}
CLOCK:INPut?	{INTernal|EXTernal}
CLOCK:RATio	{HALF|FULL}
CLOCK:RATio?	{HALF|FULL}
CLOCK:BITrate	<value>
CLOCK:BITrate?	<value>

Table 10 Clock Command Summary

CLOCK:INPut

This command is used to switch between the internal clock and external clock input.

Syntax `CLOCK:INPut {INTernal|EXTernal}`

Example
```
OUTPUT 722; "CLOCK:INP INT"
```

CLOCK:INPut?

This command is used to query which clock, internal or external, is being used.

Syntax `CLOCK:INPut?`

Response `< INT | EXT >`

Example
```
OUTPUT 722; "CLOCK:INP?"
ENTER 722; A$
```

CLOCK:RATio

This command is used to switch the clock between half clock and full clock

Syntax `CLOCK:RATio {HALF|FULL}`

Example
```
OUTPUT 722; "CLOCK:RAT HALF"
```

CLOCK:RATio?

This command is used to query whether the clock ratio is set ot full or half clock.

Syntax `CLOCK:RATio?`

Response `< HALF | FULL >`

Example
```
OUTPUT 722; “CLOCK:RAT?”
ENTER 722; A$
```

CLOCK:BITrate

This command is used to set the internal bitrate.

Syntax `CLOCK:BITrate < value >`

Response `value = < 1.24e9 | 2.49e9 | 4.98e9 | 9.95e9 | 19.91e9 | 39.81 >`

Example `OUTPUT 722; “CLOCK:BIT 9.95e9”`

CLOCK:BITrate?

This command is used to query the selected internal bitrate.

Syntax `CLOCK:BITrate?`

Response `< 1.24e9 | 2.49e9 | 4.98e9 | 9.95e9 | 19.91e9 | 39.81 >`

Example
```
OUTPUT 722; “CLOCK:BIT?”
ENTER 722; A$
```

Chapter 9

Fetch commands

The FETch is used to return measurement values from the Error Analyzer. The FETch command returns the result for the current gating period.

If a valid result is not available, then the string "1E30" is returned (e.g. in the case of an overflow).

For the interpretation of the results obtained using the FETch query commands, the currently selected gating parameters have to be taken into account.

Command	*Parameters/Results*
FETch:SENse:ERRor:MUX?	< 0 | 1 >
FETch:SENse:ERRor:A?	< 1E30 | value >
FETch:SENse:ERRor:C?	< 1E30 | value >
FETch:SENse:ERRor:ALL?	< 1E30 | value >

Table 11: FETch Command Summary

FETch:SENse:ERRor:MUX?

These errors might occur in case of an input data stream, which is generated using multiple PRB sequences interleaved in a wrong sequence.

Syntax
```
FETch:SENse:ERRor:MUX?
```

Response
```
< "0" | "1" >
"0": no multiplexing errors
"1": multiplexing errors occurred
```

Related Commands
```
FETch:SENse:ERRor:A?
FETch:SENse:ERRor:B?
FETch:SENse:ERRor:C?
FETch:SENse:ERRor:D?
FETch:SENse:ERRor:ALL?
```

Example

```
OUTPUT 722;
"FETch:SENse:ERRor:MUX?"
ENTER 722; A$
```

FETch:SENse: ERRor:A?

This query command reports the total number of errors accumulated in channel A during the previous gating period.

Syntax

```
FETch:SENse:ERRor:A?
```

Response

```
< "1E30" | count >
```

Related Commands

```
FETch:SENse:ERRor:MUX?
FETch:SENse:ERRor:B?
FETch:SENse:ERRor:C?
FETch:SENse:ERRor:D?
FETch:SENse:ERRor:ALL?
```

Example

```
OUTPUT 722; "FETch:SENse:ERRor:A?"
ENTER 722; A$
```

FETch:SENse: ERRor:B?

This query command reports the total number of errors accumulated in channel B during the previous gating period.

Syntax

```
FETch:SENse:ERRor:B?
```

Response

```
< "1E30" | count >
```

Related Commands

```
FETch:SENse:ERRor:MUX?
FETch:SENse:ERRor:A?
FETch:SENse:ERRor:C?
FETch:SENse:ERRor:D?
FETch:SENse:ERRor:ALL?
```

Example

```
OUTPUT 722; "FETch:SENse:ERRor:B?"
ENTER 722; A$
```

FETch:SENse: ERRor:C?

This query command reports the total number of errors accumulated in channel C during the previous gating period.

Syntax

```
FETch:SENse:ERRor:C?
```

Response

```
< "1E30" | count >
```

Related Commands

```
FETch:SENse:ERRor:MUX?
FETch:SENse:ERRor:A?
FETch:SENse:ERRor:B?
FETch:SENse:ERRor:D?
```

```
                    FETch:SENse:ERRor:ALL?
Example             OUTPUT 722; "FETch:SENse:ERRor:C?"
```

FETch:SENse: ERRor:D?

This query command reports the total number of errors accumulated in channel D during the previous gating period.

```
Syntax              FETch:SENse:ERRor:D?

Response            < "1E30" | count >

Related Commands    FETch:SENse:ERRor:MUX?
                    FETch:SENse:ERRor:A?
                    FETch:SENse:ERRor:B?
                    FETch:SENse:ERRor:C?
                    FETch:SENse:ERRor:ALL?
Example         OUTPUT 722; "FETch:SENse:ERRor:D?"
```

FETch:SENse: ERRor:ALL?

This query command reports the bit error ratio (BER, errors per bit) which has been measured during the last gating period.

```
Syntax              FETch:SENse:ERRor:ALL?

Response            < "1E30" | value >

Related Commands    FETch:SENse:ERRor:MUX?
                    FETch:SENse:ERRor:A?
                    FETch:SENse:ERRor:B?
                    FETch:SENse:ERRor:C?
                    FETch:SENse:ERRor:D?
Example             OUTPUT                          722;
                    "FETch:SENse:ERRor:ALL?"
                    ENTER 722; A$
```

Chapter 10

Eye Commands

These commands relate to the measurement and capture of an eye profile from the EA 40 GIG

Command	*Parameters/Results*
EYE:CONtour	{1e-6|1e-7|1e-8|1e-9|1e-10|1e-11|1e-12}
EYE:CONtour?	see text

Table 12 Eye Command Summary

EYE:CONtour

This command starts an eye contour measurement with the error limit defined by `value.`

Syntax

```
EYE:CONtour < value >

value = 1e-6, 1e-7, 1e-8, 1e-9,
1e-10, 1e-11, 1e-12
```

Example

```
OUTPUT 722; "EYE:CON 1E-10"
```

EYE:CONtour?

With this query, an array of 161 integer pairs of type LONG (4 Bytes) is returned. Each pair represents the upper and lower threshold voltage limit at which the previously chosen bitrate limit has not been exceeded. The first pair corresponds to the -80 ps delay line setting, the second pair to the -79 ps delay line setting and so on up to 80 ps in 1 ps steps. An appropriate data structure to extract the data from the received ASCII string should be set up. A value of 1E9 for the upper limit or -1E9 for the lower limit means that this point is not valid, i.e. this point is outside the eye.

Syntax

```
EYE:CONtour?
```

Example of typical code for the EYE:CONtour? function:

```
CStdioFile File;
char buffer[255];
int i;
```

```
int ud;     // variable for unit descriptor
int adr;    // GPIB Address

struct eadata {
   long upper;
   long lower;
} ea_eye[161];

adr=20;

File.Open("eyecont.dat",CFile::modeCreate|CFile::modeWrit
e|CFile::typeText);
   // open the device
ud=ibdev(0,adr,0,T10s,1,0);
if (ud < 0) {
   gpiberr("ibdev Error");
      // Take the device offline.
   ibloc(ud);
   ibonl(ud,0);
   return(1); // return error occurred
}
   // send query
ibwrt (ud,"EYE:CONTOUR?",12);
if (ibsta & ERR) {
   gpiberr("ibwrt Error");
      // Take the device offline.
   ibloc(ud);
   ibonl(ud,0);
   return(1); // return error occurred
}
   // receive data
ibrd(ud,(char*)ea_eye,1288);
if (ibsta & ERR) {
   gpiberr("ibrd Error");
   ibloc(ud);
   ibonl(ud,0);
   return(1);  // return error occurred
}
   // go to local operation
ibloc(ud);
   // take the device offline
ibonl(ud,0);

for (i=0;i<161;i++) {
   if (ea_eye[i].upper<1E9) {
      sprintf(buffer,"%d\t%ld\n",i-80,ea_eye[i].upper);
      File.WriteString(buffer);
   }
   if (ea_eye[i].lower>-1E9) {
      sprintf(buffer,"%d\t%ld\n",i-80,ea_eye[i].lower);
      File.WriteString(buffer);
   }
}
File.Close();
return(0); // return no error
```

Chapter 11

Burst commands

The BURst commands control the burst gating function of the EA 40 GIG. They allow the feature to be activated and deactivated as well as setting the length of the burst, the number of cycles to be measured and the timeout.

Command	*Parameters/Results*
BURst:GATing	{ENABle|DISABle}
BURst:GATing?	{ENABle|DISABle}
BURst:LENgth	<value>
BURst:LENgth?	<value>
BURst:CYCles	<value>
BURst:CYCles?	<value>
BURst:TIMeout	<value>
BURst:TIMeout?	<value>

Table 13: FETch Command Summary

BURst:GATing

This command is used to activate and deactivate the burst gating.

Syntax `BURst:GATing {ENABle|DISABle}`

Example
```
OUTPUT 722; "BUR:GAT ENAB"
```

BURst:GATing?

This command is used to query the status of the burst gating

Syntax `BURst:GATing?`

Response `{ENABle|DISABle}`

Example
```
OUTPUT 722; "BUR:GAT?"
ENTER 722; A$
```

BURst:LENgth

This command is used to set the length of a data burst in µs

Syntax `BURst:LENgth < value >`

`10 ≤ value ≤ 10240µs in 10µs steps`

Example
```
OUTPUT 722; "BUR:LEN 9950"
```

BURst:LENgth?

This command is used to query the length of a data burst

Syntax `BURst:LENgth?`

Response `10 ≤ value ≤ 10240µs`

Example
```
OUTPUT 722; "BUR:LEN?"
ENTER 722; A$
```

BURst:CYCles

This command sets the number of burst cycles.

Syntax `BURst:CYCles < value >`

`1 ≤ value ≤ 262140`

Example
```
OUTPUT 722; "BUR:CYC 1000"
```

BURst:CYCles?

This command queries the number of burst cycles set.

Syntax `BURst:CYCles?`

Response `1 ≤ value ≤ 262140`

Example
```
OUTPUT 722; "BUR:CYC?"
ENTER 722; A$
```

BURst:TIMeout

This command is used to set the maximum waiting time for a burst

Syntax `BURst: < value >`

`105 ≤ value ≤ 26880`

Example
```
OUTPUT 722; "BUR:TIM "
```

BURst:TIMeout? This command is used to query the maximum waiting time for a burst

Syntax `BURst:TIMeout?`

Response `105 ≤ value ≤ 26880`

Example
```
OUTPUT 722; “BUR:TIM?”
ENTER 722; A$
```

Chapter 12

Audio commands

These commands switch the audio support on and off, as well as controlling the volume level and the audio sync loss function.

Command	*Parameters/Results*
AUDio:SUPport	{ON|OFF}
AUDio: SUPport?	{ON|OFF}
AUDio:VOLume	<value>
AUDio: VOLume?	<value>
AUDio:SYNcloss	{ON|OFF}
AUDio: SYNcloss?	{ON|OFF}

Table 13: FETch Command Summary

AUDio:SUPport

This command switches the audio support on and off

Syntax `AUDio:SUPport {ON|OFF}`

Example
```
OUTPUT 722; “AUD:SUP ON”
```

AUDio:SUPport?

This command queries the status of the audio support

Syntax `AUDio:SUPport?`

Response `{ON|OFF}`

Example
```
OUTPUT 722; “AUD:SUP?”
ENTER 722; A$
```

AUDio:VOLume This command sets the volume level

Syntax `AUDio:VOLume <value>`

`0 ≤ value ≤ 15`

Example
```
OUTPUT 722; "AUD:VOL 3"
```

AUDio:VOLume? This command queries the setting of the volume level

Syntax `AUDio:VOLume?`

Response `0 ≤ value ≤ 15`

Example
```
OUTPUT 722; "AUD:VOL?"
ENTER 722; A$
```

AUDio:SYNcloss This command is used to switch on the audio sync loss

Syntax `AUDio:SYNcloss {ON|OFF}`

Example
```
OUTPUT 722; "AUD:SYN ON"
```

AUDio:SYNcloss? This command is used to query the status of the audio sync loss

Syntax `AUDio:SYNcloss?`

Response `{ON|OFF}`

Example
```
OUTPUT 722; "AUD:SYN?"
ENTER 722; A$
```