Timer/Counter/ Analyzers

PM6680B, PM6681, PM6681R, PM6685 & PM6685R

Programming Manual

Table of Contents

1 Getting Started

Finding Your Way Through This Manual. . 1-2
Manual Conventions 1-3
Setting Up the Instrument 1-4
Interface Functions 1-5

2 Bus Commands for the Benchtop User

Default settings (after *RST). 2-8

3 Introduction to SCPI

What is SCPI? 3-2
How does SCPI Work in the Instrument? . 3-4
Program and Response Messages 3-8
Command Tree 3-11
Parameters 3-12
Macros. 3-15
Status Reporting System. 3-18
Error Reporting 3-19
Initialization and Resetting. 3-21

4 Programming Examples

Introduction 4-2

GW-Basic for National Instruments PC-IIA 4-3

Setting up the interface 4-3
1. Limit Testing 4-4
3. Frequency Profiling 4-5
4. Fast Sampling 4-7
5. Status Reporting 4-9
6. Statistics 4-11

'C' for National Instruments PC-IIA 4-13

1. Limit Testing 4-14
2. REAL Data Format 4-15
3. Frequency Profiling 4-17
4. Fast Sampling 4-19
6. Statistics 4-21

5 Instrument Model

Introduction 5-2
Measurement Function Block 5-3
Other Subsystems. 5-4
Order of Execution. 5-4
MEASurement Function 5-5

6 Using the Subsystems

Introduction 6-2
Calculate Subsystem. 6-3
Calibration Subsystem. 6-4
Configure Function 6-5
Format Subsystem 6-6
Time Stamp Readout Format 6-6
Input Subsystems 6-7
Measurement Function 6-9
Output Subsystem. 6-12
Sense Command Subsystems 6-14
Status Subsystem 6-15
Trigger/Arming Subsystem 6-30

7 How to Measure Fast

Introduction 7-3
Rough Trigger Subsystem Description . . . 7-4
Some Basic Commands 7-5
Basic Measurement Method 7-7

4822 872 20081
August 2000

General Speed Improvements. 7-8
40000 measure- ments/second. 7-11
Supervising a Process. 7-12
Speed Summary 7-17

8 Error Messages

9 Command Reference

Abort. 9-3
:ABORt 9-4
Arming Subsystem 9-5
:ARM :COUNt 9-6
:ARM :DELay. 9-7
:ARM :ECOunt. 9-7
:ARM :LAYer2 9-8
:ARM :LAYer2 :SOURce 9-8
:ARM :SLOPe 9-9
:ARM :SOURce 9-9
:ARM :STOP :DELay. 9-10
:ARM :STOP :ECOunt. 9-10
:ARM :STOP :SLOPe 9-11
:ARM :STOP :SOURce 9-11
Calculate Subsystem 9-13
:CALCulate :AVERage :COUNt. 9-14
:CALCulate :AVERage :STATe 9-14
:CALCulate :AVERage :TYPE. 9-15
:CALCulate :DATA?. 9-15
:CALCulate :IMMediate 9-16
:CALCulate :LIMit 9-16
:CALCulate :LIMit :FAIL?. 9-17
:CALCulate :LIMit :LOWer. 9-17
:CALCulate :LIMit :LOWer :STATe 9-18
:CALCulate :LIMit :UPPer 9-18
:CALCulate :LIMjt :UPPer :STATe. 9-19
:CALCulate :MATH 9-20
:CALCulate :MATH 9-21
:CALCulate :MATH :STATe. 9-21
:CALCulate :STATe. 9-22
Calibration Subsystem 9-23
:CALibration :INTerpolator :AUTO. 9-24
Configure Function 9-25
:CONFigure :<Measuring Function> 9-26
:CONFigure :ARRay :<Measuring Function> 9-27
Diagnostics Subsystem 9-29
:DIAGnostic:CALibration:INPut[1|2]:HYSTeresis 9-30
Display Subsystem 9-31
:DISPlay :ENABle 9-32
Fetch Function. 9-33
:FETCh? 9-34
:FETCh :ARRay?. 9-35
Format Subsystem 9-37
:FORMat 9-38
:FORMat 9-38
:FORMat :FIXed 9-39
:FORMat :SREGister. 9-39
:FORMat :TINFormation 9-40
Initiate Subsystem. 9-41
:INITiate :CONTinuous 9-42
:INITiate. 9-42
Input Subsystems 9-43
:INPut«[1]|2» :COUPling 9-44
:INPut«[1]|2» :ATTenuation. 9-44
:INPut :HYSTeresis 9-45
:INPut :FILTer 9-45
:INPut :HYSTeresis :AUTO 9-46
:INPut«[1]|2» :IMPedance 9-47
:INPut«[1]|2» :LEVel 9-47
:INPut :LEVel. 9-48
:INPut :LEVel :AUTO. 9-49
:INPut :LEVel :AUTO. 9-50
:INPut«[1]|2|4» :SLOPe. 9-51
:INPut2:COMMon 9-51
Measurement Function 9-53
:MEASure :<Measuring Function>? 9-56
:MEASure :ARRay :<Measuring Function>? 9-57
:MEASure:MEMory<N>?. 9-58
:MEASure:MEMory? 9-58
:MEASure_«:DCYCle/:PDUTycycle» . . . 9-59
EXPLANATIONS OF THE MEASURING FUNCTIONS 9-59
:MEASure :FREQuency?. 9-60
:MEASure :FREQuency :BURSt? 9-61
:MEASure :FREQuency :PRF? 9-62
:MEASure :FALL :TIME?. 9-63

:MEASure :FREQuency :RATio? 9-63
:MEASure [:VOLT] :MAXimum? 9-64
:MEASure [:VOLT] :MINimum? 9-64
:MEASure :NWIDth? 9-65
:MEASure :PWIDth? 9-65
:MEASure_«:PDUTycycle/ :DCYCle»? 9-66
:MEASure_«:NDUTycycle»? 9-66
:MEASure :PERiod? 9-67
:MEASure :PHASe? 9-67
:MEASure [:VOLT] :PTPeak? 9-68
:MEASure :RISE :TIME? 9-68
:MEASure :TINTerval? 9-69
:MEASure :TOTalize :ACCumulated? 9-70
:CONFigure :TOTalize :CONTinuous 9-71
:MEASure :TOTalize :GATed? 9-72
:MEASure :TOTalize :SSTop? 9-72
:MEASure :TOTalize :TIMed? 9-73
Memory Subsystem 9-75
:MEMory :DELete :MACRo 9-76
:MEMory :FREE :SENSe? 9-76
:MEMory :FREE :MACRo? 9-77
:MEMory :NSTates? 9-77
Output Subsystem 9-79
:OUTPut 9-80
:OUTPut :SCALe 9-80
Read Function 9-81
:READ? 9-82
:READ:ARRay? 9-83
Sense Command Subsystem 9-85
:ACQuisition :APERture 9-87
:ACQuisition :APERture 9-87
:ACQuisition :HOFF: ECOunt 9-88
:ACQuisition :HOFF 9-88
:ACQuisition :HOFF :TIME 9-89
:ACQuisition :HOFF :MODE 9-89
:ACQuisition :RESolution 9-90
:ACQuisition :RESolution 9-90
:AVERage :MODE 9-91
:AVERage :COUNt 9-91
:FREQuency :RANGe :LOWer 9-92
:AVERage :STATe 9-92
:FUNCtion 9-93
:INTernal :FORMat 9-95
:ROSCillator :SOURce 9-96
:SDELay 9-96
:TOTalize :GATE 9-97
:VOLTage:GATed:STATe 9-97
Status Subsystem 9-99
:STATus :DREGister0? 9-100
:STATus :DREGister0 :ENABle 9-100
:STATus :OPERation :CONDition? 9-101
:STATus :OPERation :ENABle 9-102
:STATus:OPERation? 9-103
:STATus :PRESet 9-103
:STATus :QUEStionable :CONDition? 9-104
:STATus :QUEStionable? 9-105
:STATus :QUEStionable :ENABle 9-105
System Subsystem 9-107
:SYSTem :COMMunicate: GPIB: ADDRess 9-108
:SYSTem :ERRor? 9-108
:SYSTem :PRESet 9-109
:SYSTem :SDETect 9-109
:SYSTem :SET 9-110
:SYSTem :TIME :ELAPsed? 9-110
:SYSTem :TOUT 9-111
:SYSTem :TOUT :TIME 9-111
:SYSTem :UNPRotect 9-112
:SYSTem :VERSion? 9-112
Test Subsystem 9-113
:TEST:CHECk 9-114
:TEST :SELect 9-114
Trigger Subsystem 9-115
:TRIGger:COUNt 9-116
Common Commands 9-117
*CLS 9-118
*DMC 9-119
*EMC 9-120
*ESE 9-121
*ESR? 9-122
*GMC? 9-122
*IDN? 9-123
*LMC? 9-123
*OPC 9-124
*LRN? 9-124

∗OPC? 9-125
∗OPT? 9-125
∗PSC 9-126
∗PMC 9-126
∗PUD 9-127
∗RCL 9-127
∗RMC 9-128
∗RST 9-128
∗SAV 9-129
∗SRE 9-130
∗STB? 9-131
∗TRG 9-131
*TST? 9-132
∗WAI 9-132

10 Index

Chapter 1

Getting Started

Finding Your Way Through This Manual

You should use this Programming Manual together with the PM6680B/1/5 Operators Manual. That manual contains specifications for the counter and explanations of the possibilities and limitations of the different measuring functions.

Sections

The chapters in this manual are divided into three sections aimed at different levels of reader knowledge.

The 'General' Section, which can be disregarded by the users who know the IEEE-488 and SCPI standards:

- *Chapter 2 Bus Commands for the Benchtop User gives bus commands for the front panel keys.*
- *Chapter 3 Introduction to SCPI explains syntax data formats, status reporting, etc.*

The Practical Section of this manual contains:

- *Chapter 4, Programming Examples, with examples of typical programs for a wide variety of applications. These programs are written in GW-basic and C.*

The 'Programmers Reference' Section of this manual contains:

- *Chapter 5, Instrument Model explains how the instrument looks from the bus. This instrument is not quite the same as the one used from the front panel.*
- *Chapter 6 Using the Subsystems explains more about each subsystem.*
- *Chapter 7, How to Measure Fast is a set of measuring situations which the user is often confronted with when programming a counter. This chapter also contains information about how to use the more complex subsystem.*
- *Chapter 8, Error Messages contains a list of all error messages that can be generated during bus control.*
- *Chapter 9, Command Reference, This chapter gives complete information on all commands. The subsystems and commands are sorted alphabetically.*

Index

You can also use the index to get an overview of the commands. The index is also useful when looking for additional information on the command you are currently working with.

Manual Conventions

Syntax Specification Form

This manual uses the EBNF (Extended Backus-Naur Form) notation for describing syntax. This notation uses the following types of symbols:

■ Printable Characters:

Printable characters such as Command headers, etc., are printed just as they are, e.g. period means that you should type the word PERIOD.

The following printable characters have a special meaning and will only be used in that meaning: # ' " () : ; *
Read Chapter 3' Introduction to SCPI' for more information.

■ Non-printable Characters:

Two non-printable characters are used:

␣ *indicates the space character (ASCII code 32).*

↵ _ *indicates the new line character (ASCII code 10).*

■ Specified Expressions: < >

Symbols and expressions that are further specified elsewhere in this manual are placed between the <> signs.
For example <Dec. data.>. The following explanation is found on the same page: "Where <Dec. data> is a four-digit number between 0.1 and 8*10-9.

■ Alternative Expressions Giving Different Result:

Alternative expressions giving different results are separated by |. For example, On|Off means that the function may be switched on or off.

■ Grouping: « »

Example: FORMat␣«ASCII|REAL» specifies the command header FORMat followed by a space character and either ASCII or REAL.

■ Optionality: [ ]

An expression placed within [ ] is optional.

Example: `[:VOLT]:FREQuency`

means that the command FREQuency may or may not be preceded by :VOLT.

■ Repetition: { }

An expression placed within { } can be repeated zero or more times.

■ Equality: =

Equality is specified with =
Example: <Separator>= ,

Mnemonic Conventions

■ Truncation Rules

All commands can be truncated to shortforms. The truncation rules are as follows:

- The shortform is the first four characters of the command.
- If the fourth character in the command is a vowel, then the shortform is the first three characters of the command. This rule is not

used if the command is only four characters.

– If the last character in the command is a digit, then this digit is appended to the shortform.

Examples:

Longform	Shortform
:MEASURE	:MEAS
:NEGATIVE	:NEG
:DREGISTER0	:DREG0
:EXTERNAL4	:EXT4

The shortform is always printed in CAPITALS in this manual: :MEASure, :NEGative, :DREGister0, :EXTernal4 etc.

■ Example Language

Small examples are given at various places in the text. These examples are not in BASIC or C, nor are they written for any specific controller. They only contain the characters you should send to the counter and the responses that you should read with the controller.

Example:

SEND→ `MEAS:FREQ?`

This means that you should program the controller so that it addresses the counter and outputs this string on the GPIB.

READ← `1.234567890E6`

This means that you should program the controller so that it can receive this data from the GPIB, then address the counter and read the data.

Setting Up the Instrument

Setting the GPIB Address

The address switches on the rear panel of the counter are set to 10 when it is delivered. The address used is displayed when the instrument is turned on.

If you want to use another bus address, you can set these switches to any address between 0 and 30 as shown in the following table.

Address	Switch Settings	Address	Switch Settings
0	00000	16	10000
1	00001	17	10001
2	00010	18	10010
3	00011	19	10011
4	00100	20	10100
5	00101	21	10101
6	00110	22	10110
7	00111	23	10111
8	01000	24	11000
9	01001	25	11001
10	**01010**	26	11010
11	01011	27	11011
12	01100	28	11100
13	01101	29	11101
14	01110	30	11110
15	01111		

The address can also be set via a GPIB command or from the AUX MENU on the PM6680B/1/5. The set address is stored in nonvolatile memory and remains until you change it.

Power-on

When turned on, the counter starts with the setting it had when turned off.

■ Standby

When the counter is in REMOTE mode, you cannot switch it off. You must first enable Local control by pressing LOCAL.

Testing the Bus

To test that the instrument is operational via the bus, use *IDN? to identify the instrument and *OPT? to identify which options are installed. (See 'System Subsystem' , *IDN? and *OPT?)

Interface Functions

What can I do with the Bus?

All the capabilities of the interface for the PM6680B-series are explained below.

■ Summary

Description,	Code
Source handshake,	SH1
Acceptor handshake,	AH1
Control function,	C0
Talker Function,	T6
Listener function,	L4
Service request,	SR1
Remote/local function,	RL1
Parallel poll,	PP0
Device clear function,	DC1
Device trigger function,	DT1
Bus drivers,	E2

■ SH1 and AH1

These simply mean that the counter can exchange data with other instruments or a controller using the bus handshake lines: DAV, NRFD, NADC.

■ Control Function, C0

The counter does not function as a controller.

■ Talker Function, T6

The counter can send responses and the results of its measurements to other devices or to the controller. T6 means that it has the following functions:

- Basic talker.
- No talker only.
- It can send out a status byte as response to a serial poll from the controller.
- Automatic un-addressing as a talker when it is addressed as a listener.

■ Listener Function, L4

The counter can receive programming instructions from the controller. L4 means that it has the following functions:

– Basic listener.

– No listen only.

– Automatic un-addressing as listener when addressed as a talker.

■ Service Request, SR1

The counter can call for attention from the controller, e.g., when a measurement is completed and a result is available.

■ Remote/Local, RL1

You can control the counter manually (locally) from the front panel or remotely from the controller. The LLO, local-lock-out function, can disable the LOCAL button on the front panel.

■ Parallel Poll, PP0

The counter does not have any parallel poll facility.

■ Device Clear, DC1

The controller can reset the counter via interface message DCL (Device clear) or SDC (Selective Device Clear).

■ Device Trigger, DT1

You can start a new measurement from the controller via interface message GET (Group Execute Trigger).

■ Bus Drivers, E2

The GPIB interface has tri-state bus drivers.

Chapter 2

Bus Commands for the Benchtop User

:INP:FILT_ON|OFF

Switches on or off the 100kHz LP-filter

:INP:IMP_50|1E6

Sets the input impedance 50Ω or 1MΩ

:INP:SLOP_POS|NEG

Positive or negative trigger slope

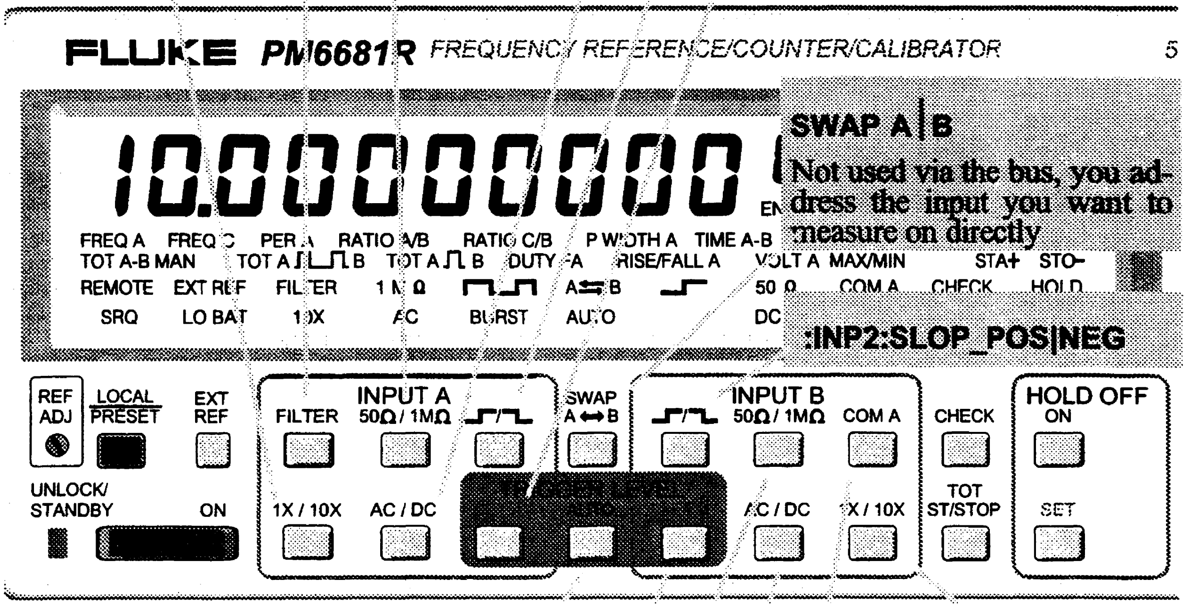
:INP:ATT_1|10

Attenuation 1X or 10X

:INP:COUP_AC|DC

:INP:LEV_<level>

Level can be set to between –5.1 to + 5.1 V when attenuator is set to 1X, and –51 to + 51 V when attenuator is set to 10X


FLUKE PM6681R FREQUENCY REFERENCE/COUNTER/CALIBRATOR
5
10.0000000000
FREQ A FREQ C PER A RATIO A/B RATIO C/B P WIDTH A TIME A-B
TOT A-B MAN TOT A ⎍ B TOT A ⎍ B DUTY A RISE/FALL A VOLT A MAX/MIN STA+ STO-
REMOTE EXT REF FILTER 1 MΩ A⇆B 50 Ω COM A CHECK HOLD
SRQ LO BAT 10X AC BURST AUTO DC
REF ADJ
LOCAL PRESET
EXT REF
UNLOCK/ STANDBY
ON
INPUT A
FILTER
50Ω / 1MΩ
1X / 10X
AC / DC
SWAP A↔B
TRIGGER LEVEL
AUTO
INPUT B
50Ω / 1MΩ
COM A
AC / DC
1X / 10X
CHECK
TOT ST/STOP
HOLD OFF
ON
SET

SWAP A|B

Not used via the bus, you address the input you want to measure on directly

:INP2:SLOP_POS|NEG

:INP:LEV:AUTO_ON|OFF|ONCE

:INP2:LEV:AUTO_ON|OFF|ONCE

Note that AUTO is selected individually for A and B inputs

:INP2:ATT_1|10

:INP2:COM_MON|OFF

:INP2:COUP_AC|DC

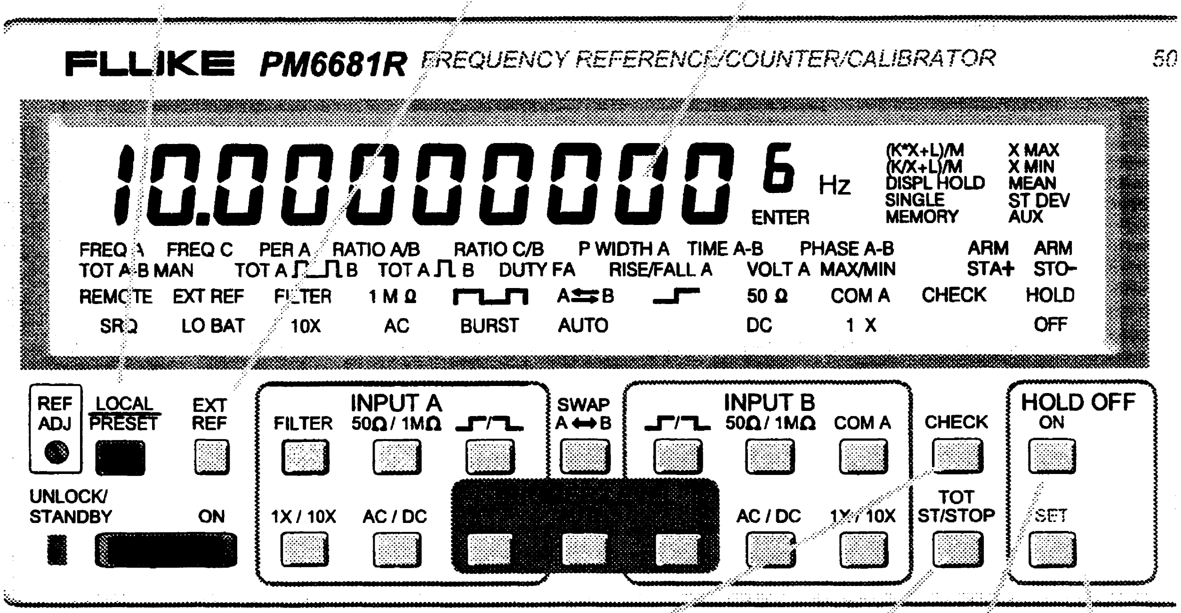
:INP2:IMP_50|1E6

:INP2:LEV_<level>

:DISPL:ENAB_ON|OFF

:ROSC:SOUR_INT|EXT*

:SYST:PRES or *RST
Presets the counter to default



:TEST:CHEC_ON|OFF

:TOT:GAT_ON|OFF*

:ACQ:HOFF_ON|OFF*

:ACQ:HOFF:TIME_<time>
Time can be set between
200E–9 and 1.6 *

* These commands are from the SENSE subsystem

:FUNC_"function_channel,channel" *

Function and channel is explained on page 2-6

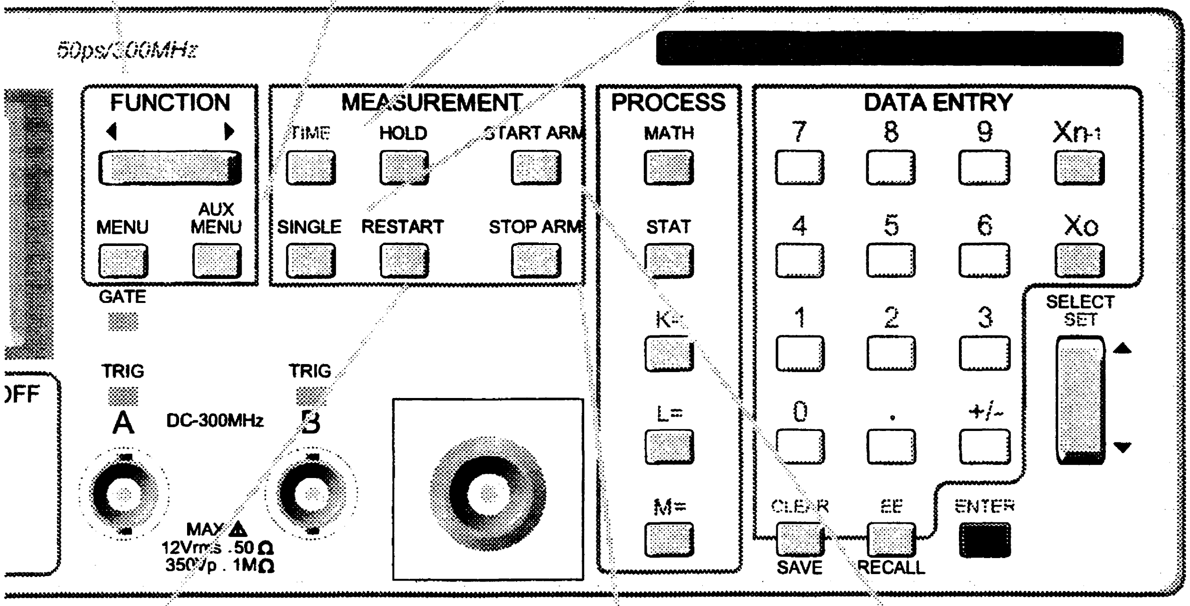
:ACQ:APER_<time>

Time can be set to:
0.8E–6, 1.6E–6, 3.2 E–6,
6.4E–6 12.8E–6
and 50E–6 to 400 *

The functions in the auxiliary menu tree are found in many different subsystem command trees, for instance the No. of samples for statistics is in the Calculate subsystem

:AVER:STAT_OFF|ON

OFF gives SINGLE
ON gives AVER-



:READ?

Starts a measurement and requests result

:ARM:SOUR_EXT2|EXT4

Switches on start arming on input B(2) or E(4).

:ARM:SOUR_IMM

Switches off start arming

:ARM:SLOP_POS|NEG

:ARM:STOP:SOUR_EXT2|EXT4

Switches on stop arming on input B(2) or E(4).

:ARM:STOP:SOUR_IMM

Switches off stop arming

:ARM:STOP:SLOP_POS|NEG

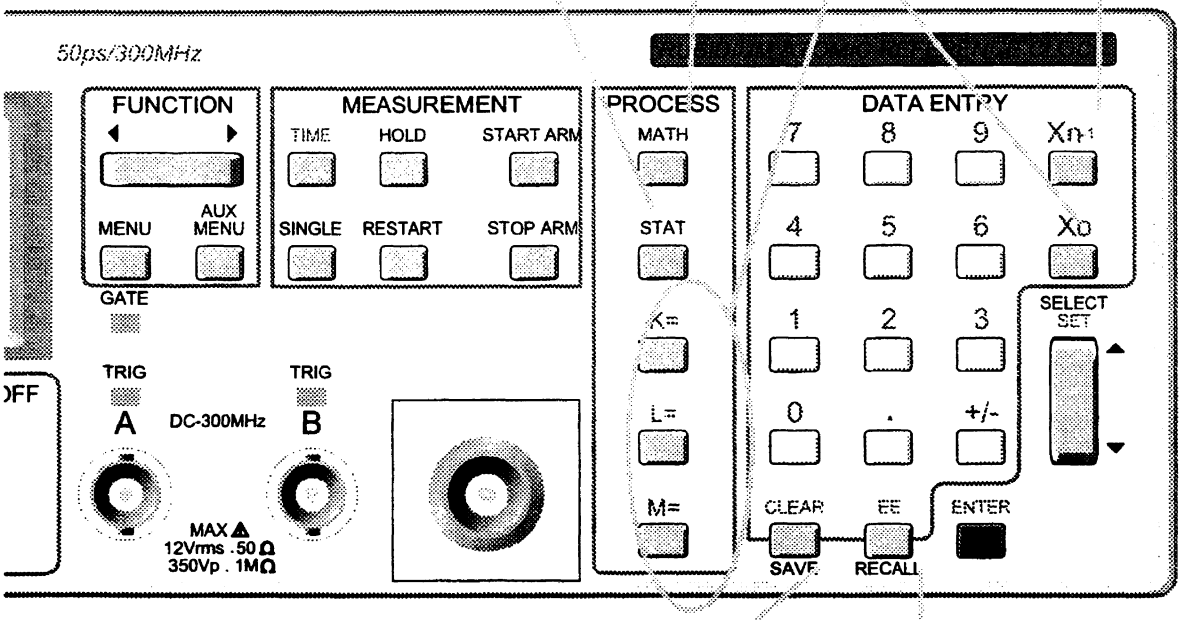
* These commands are from the SENSE subsystem

:CALC:MATH_ (<expression>)
Expression is mathematical expression containing +, –, *, /, and XOLD
:CALC:MATH:STAT_ON|OFF

XOLD
in a mathematical expression
gives the same result ar pressing Xn-1

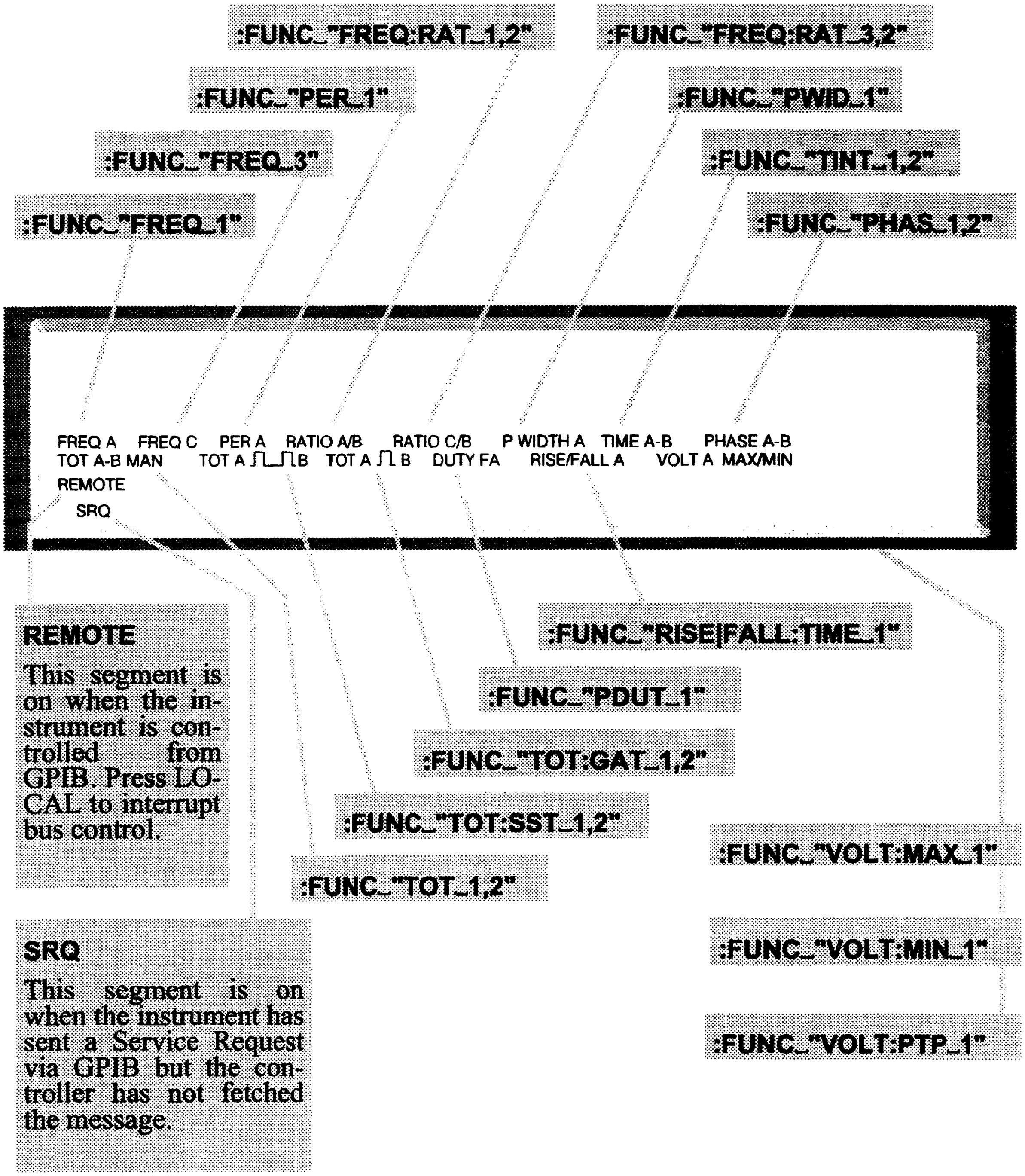
:CALC:AVER:TYPE_MAX|MIN|SDEV|MEAN
Selects statistical function
:CALC:AVER:STAT_ON|OFF

Not used via bus; enter constants directly in the mathematical expression



***SAV_<memory location>**
Memory location can be any No. between 0 and 19

***RCL_<memory location>**



:FUNC_"FREQ:RAT_1,2"

:FUNC_"FREQ:RAT_3,2"

:FUNC_"PER_1"

:FUNC_"PWID_1"

:FUNC_"FREQ_3"

:FUNC_"TINT_1,2"

:FUNC_"FREQ_1"

:FUNC_"PHAS_1,2"

REMOTE

This segment is on when the instrument is controlled from GPIB. Press LOCAL to interrupt bus control.

:FUNC_"RISE|FALL:TIME_1"

:FUNC_"PDUT_1"

:FUNC_"TOT:GAT_1,2"

:FUNC_"TOT:SST_1,2"

:FUNC_"VOLT:MAX_1"

:FUNC_"TOT_1,2"

SRQ

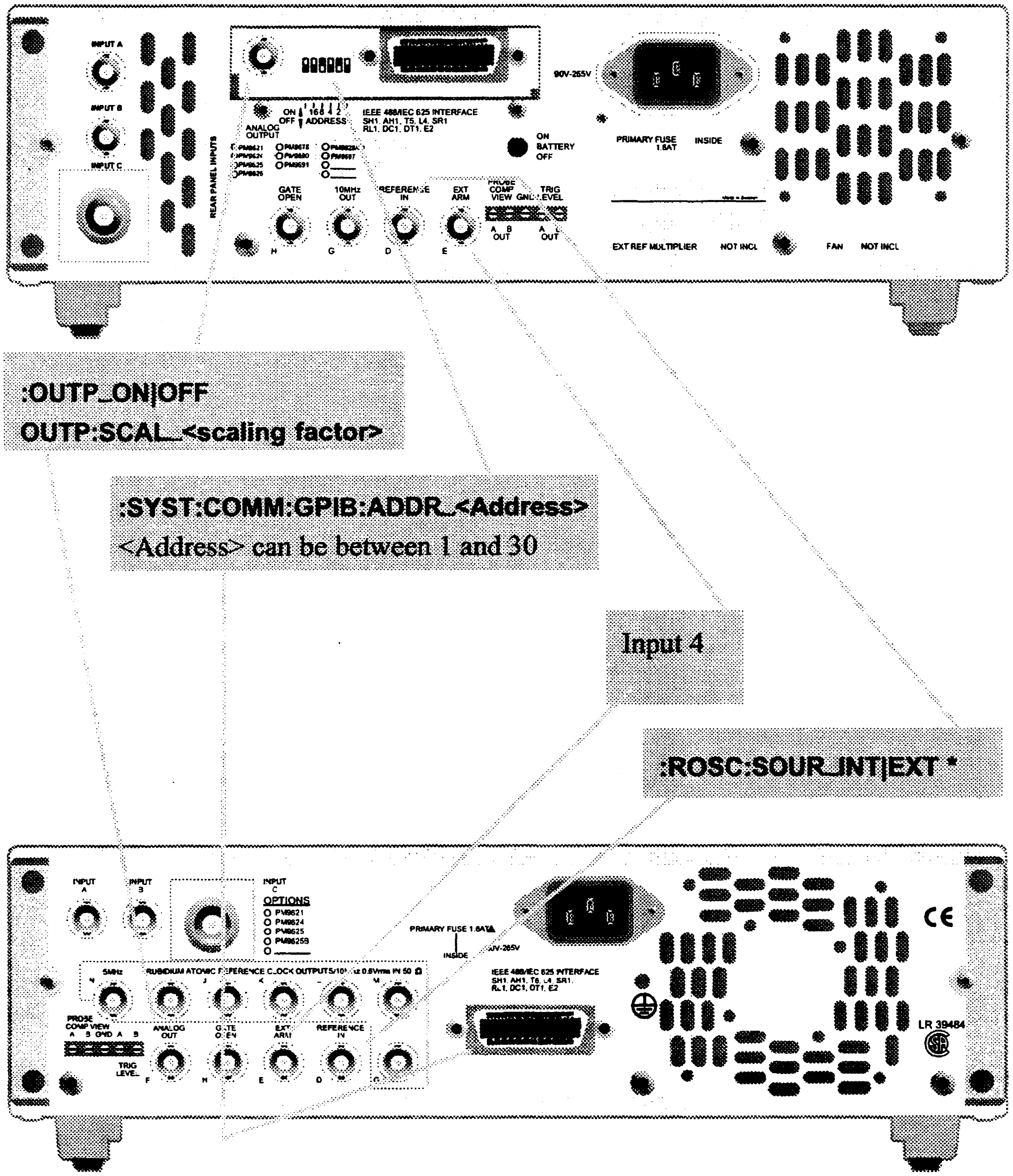
This segment is on when the instrument has sent a Service Request via GPIB but the controller has not fetched the message.

:FUNC_"VOLT:MIN_1"

:FUNC_"VOLT:PTP_1"

All commands on this page are from the SENSE subsystem

PM6680B



:OUTP_ON|OFF
OUTP:SCAL_<scaling factor>

:SYST:COMM:GPIB:ADDR_<Address>
<Address> can be between 1 and 30

Input 4

:ROSC:SOUR_INT|EXT *

PM6681R

* This command is from the SENSE subsystem

Default settings (after *RST)

PARAMETER	VALUE/ SETTING
Input A:	
Trigger level	AUTO
Impedance	1 MΩ
Manual Trigger level (Controlled by autotrigger)	0V
Manual Attenuator (Controlled by autotrigger)	1X
Coupling	AC
Trigger slope	Pos
Filter	OFF
Input B:	
Trigger level	AUTO
Impedance	1 MΩ
Manual Trigger level (Controlled by autotrigger)	0V
Manual Attenuator (Controlled by autotrigger)	1X
Coupling	DC
Trigger slope	Pos
Common	OFF
Arming:	
Start	OFF
Stop	OFF
Delay	Start, Time, OFF
Channel	Ext Arm Input E
Statistics:	
Statistics	OFF

PARAMETER	VALUE/ SETTING
Mathematics	OFF
Sample size in Statistics	100
Sample size in Time Interval Average	100
Mathematical constants:	
K= and M=	1
L=	0
Miscellaneous:	
Function	FREQ A
Timeout	100 ms, OFF
Measuring time	100 µs
Check	OFF
Single cycle	OFF
Analog output control	OFF
Hold Off	Time, OFF
Memory Protection (Memory 10 to19)	Not changed by reset
Auxiliary functions	All switched OFF
Blank LSD	OFF

Chapter 3

Introduction to SCPI

What is SCPI?

SCPI (Standard Commands for Programmable Instruments) is a standardized set of commands used to remotely control programmable test and measurement instruments. The CNT-8X firmware contains the SCPI. It defines the syntax and semantics that the controller must use to communicate with the instrument.

This chapter is an overview of SCPI and shows how SCPI is used in Fluke Frequency Counters and Timer/Counters.

SCPI is based on IEEE-488.2 to which it owes much of its structure and syntax. SCPI can, however, be used with any of the standard interfaces, such as GPIB (=IEC625/IEEE-488), VXI and RS-232.

Reason for SCPI

For each instrument function, SCPI defines a specific command set. The advantage of SCPI is that programming an instrument is only function dependent and no longer instrument dependent. Several different types of instruments, for example an oscilloscope, a counter and a multimeter, can carry out the same function, such as frequency measurement. If these instruments are SCPI compatible, you can use the same commands to measure the frequency on all three instruments, although there may be differences in accuracy, resolution, speed, etc.

Compatibility

SCPI provides two types of compatibility: Vertical and horizontal.

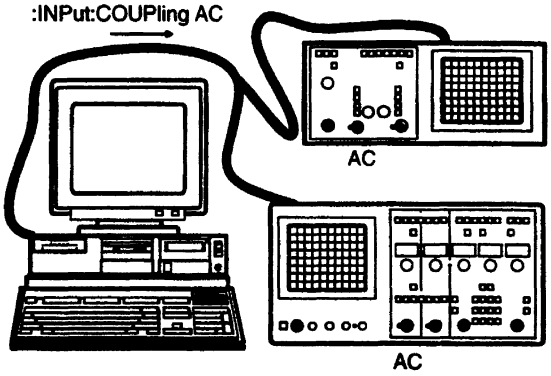


Figure 3-1 Vertical

This means that all instruments of the same type have i'dentical controls. For eample, oscilloscopes will have the same controls for timebase, triggers and voltage settings

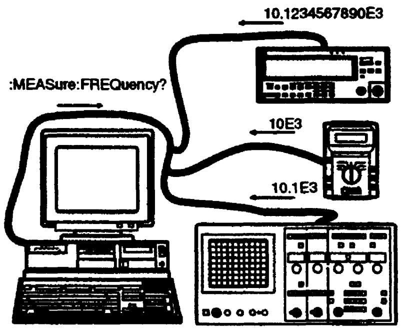


Figure 3-2 Hoizontal

This means that instruments of different types that performs the same functions have the same commands. For example, a DMM, an oscilloscope, and a counter can all measure frequency with the same commands

Management and Maintenance of Programs

SCPI simplifies maintenance and management of the programs. Today changes and additions in a good working program are hardly possible because of the great diversity in program messages and instruments. Programs are difficult to understand for anyone other than the original programmer. After some time even the programmer may be unable to understand them.

A programmer with SCPI experience, however, will understand the meaning and reasons of a SCPI program, because of his knowledge of the standard. Changes, extensions, and additions are much easier to make in an existing application program. SCPI is a step towards portability of instrument programming software and, as a consequence, it allows the exchange of instruments.

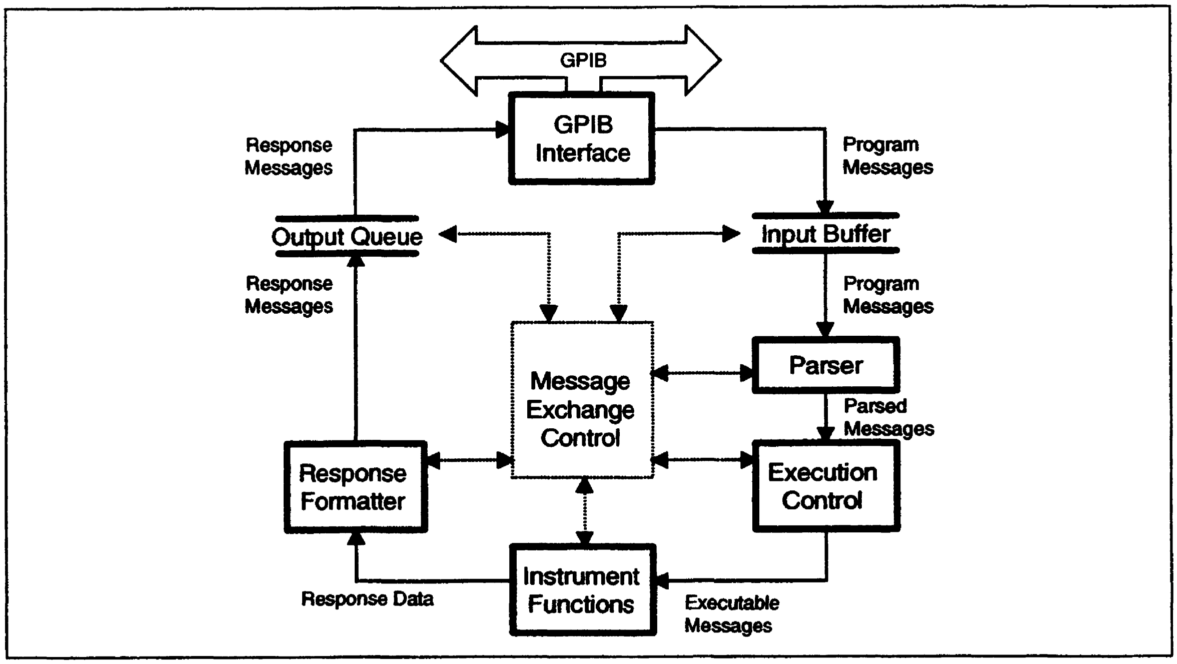


Figure 3-3 *Overview of the firmware in a SCPI instrument.*

How does SCPI Work in the Instrument?

The functions inside an instrument that control the operation provide SCPI compatibility. Figure 3-3 shows a simplified logical model of the message flow inside a SCPI instrument.

When the controller sends a message to a SCPI instrument, roughly the following happens:

- The GPIB controller addresses the instrument as listener.
- The GPIB interface function places the message in the Input Buffer.
- The Parser fetches the message from the Input Buffer, parses (decodes) the message, and checks for the correct syntax. The instrument reports incorrect syntax by sending command errors via the status system to the controller. Moreover, the parser will detect if the controller requires a response. This is the case when the input message is a query (command with a "?" appended).

The Parser will transfer the executable messages to the Execution Control block in token form (internal codes). The Execution Control block will gather the information required for a device action and will initiate the requested task at the appropriate time. The instrument reports execution errors via the status system over the GPIB and places them in the Error Queue.

- When the controller addresses the instrument as talker, the instrument takes data from the Output Queue and sends it over the GPIB to the controller.

Message Exchange Control protocol

Another important function is the Message Exchange Control, defined by IEEE 488.2. The Message Exchange Control protocol specifies the interactions between the several functional elements that exist between the GPIB functions and the device-specific functions, see Figure 3-3 .

The Message Exchange Control protocol specifies how the instrument and controller should exchange messages. For example, it specifies exactly how an instrument shall handle program and response messages that it receives from and returns to a controller.

This protocol introduces the idea of commands and queries; queries are program messages that require the device to send a response. When the controller does not read this response, the device will generate a Query Error. On the other hand, commands will not cause the device to generate a response. When the controller tries to read a response anyway, the device then generates a Query Error.

The Message Exchange Control protocol also deals with the order of execution of program messages. It defines how to respond if Command Errors, Query Errors, Execution Errors, and Device-Specific errors occur. The protocol demands that the instrument report any violation of the IEEE-488.2 rules to the controller, even when it is the controller that violates these rules.

The IEEE 488.2 standard defines a set of operational states and actions to implement the message exchange protocol. These are shown in the following table:

State	Purpose
IDLE	Wait for messages
READ	Read and execute messages
QUERY	Store responses to be sent
SEND	Send responses
RESPONSE	Complete sending responses
DONE	Finished sending responses
DEADLOCK	The device cannot buffer more data

Action,	Reason
Unterminated,	The controller attempts to read the device without first having sent a complete query message
Interrupted,	The device is interrupted by a new program message before it finishes sending a response message

Protocol Requirements

In addition to the above functional elements, which process the data, the message exchange protocol has the following characteristics:

- The controller must end a program message containing a query with a message terminator before reading the response from the device (address the device as talker). If the controller breaks this rule, the device will report a query error (unterminated action).
- The controller must read the response to a query in a previously (terminated) program message before sending a new program message. When the controller violates this rule, the device will report a query error (interrupted action).
- The instrument sends only one response message for each query message. If the query message resulted in more than one answer, all answers will be sent in one response message.

■ Order of Execution

Deferred Commands

Execution control collects commands until the end of the message, or until it finds a query or other special command that forces execution. It then checks that the setting resulting from the commands is a valid one: No range limits are exceeded, no coupled parameters are in conflict, etc. If this is the case, the commands are executed in the sequence they have been received; otherwise, an execution error is generated, and the commands are discarded.
This deferred execution guarantees the following:

- All valid commands received before a query are executed before the query is executed.
- All queries are executed in the order they are received.
- The order of execution of commands is never reversed.

■ Sequential and Overlapped Commands

There are two classes of commands: sequential and overlapped commands. All commands in the CNT-8X counters are sequential, that is one command finishes before the next command executes.

Remote Local Protocol

■ Definitions

Remote Operation

When an instrument operates in remote, all local controls, except the local key, are disabled.

Local Operation

An instrument operates in local when it is not in remote mode as defined above.

Local Lockout

In addition to the remote state, an instrument can be set to remote with 'local lockout'. This disables the return-to-local button. In theory, the state local with local lockout is also possible; then, all local controls except the return-to-local key are active.

The Counter in Remote Operation

When the Counter is in remote operation, it disables all its local controls except the LOCAL key.

The Counter in Local Operation

When the Counter is in local operation the instrument is fully programmable both from the front panel and from the bus. If a bus message arrives while a change is being entered from the front panel, the front panel entry is interrupted and the bus message is executed.

We recommend you to use Remote mode when using counters from the bus. If not, the counter measures continously and the initiation command :INIT will have no effect.

Program and Response Messages

The communication between the system controller and the SCPI instruments connected to the GPIB takes place through Program and Response Messages. A Program Message is a sequence of one or more commands sent from the controller to an instrument. Conversely, a Response Message is the data from the instrument to the controller.

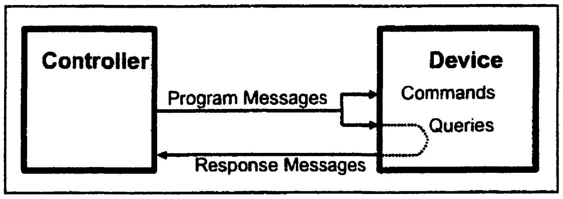


Figure 3-4 *Program and response messages.*

The GPIB controller instructs the device through program messages. The device will only send responses when explicitly requested to do so; that is, when the controller sends a query. Queries are recognized by the question mark at the end of the header, for example: *IDN? (requests the instrument to send identity data).

Syntax and Style

■ Syntax of Program Messages

A command or query is called a program message unit. A program message unit consists of a header followed by one or more parameters, as shown in Figure 3-5 .

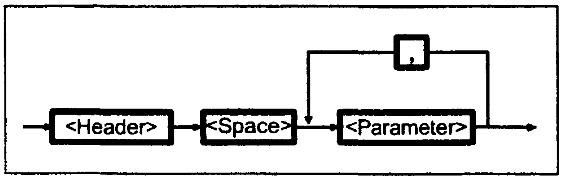


Figure 3-5 *Syntax of a Program Message Unit.*

One or more program message units (commands) may be sent within a simple program message, see Fig. 3-6.

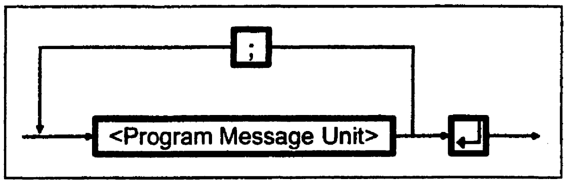


Fig 3-6 *Syntax of a terminated Program Message.*

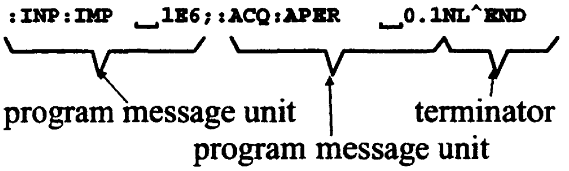
The ↵ is the pmt (program message terminator) and it must be one of the following codes:

↵	NL^END	This is <new line> code sent concurrently with the END message on the GPIB.
	NL	This is the <new line> code.
	<dab>^END	This is the END message sent concurrently with the last data byte <dab>.

NL is the same as the ASCII LF (<line feed> = ASCII $10_{decimal}$). The END message is sent via the EOI-line of the GPIB. The ^ character stands for 'at the same time as'.

Most controller programming languages send these terminators automatically, but allow changing it. So make sure that the terminator is as above.

Example of a terminated program message:



This program message consists of two message units. The unit separator (semicolon) separates message units.

Basically there are two types of commands:

Common Commands

The common command header starts with the asterisk character (*), for example *RST.

SCPI Commands

SCPI command headers may consist of several keywords (mnemonics), separated by the colon character (:).

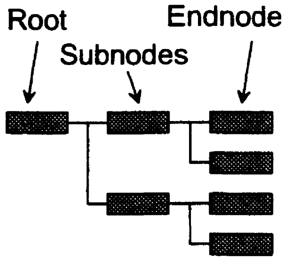


Figure 3-7 *The SCPI command tree.*

Each keyword in a SCPI command header represents a node in the SCPI command tree. The leftmost keyword (INPut in the previous example) is the root level keyword, representing the highest hierarchical level in the command tree.

The keywords following represent subnodes under the root node. See 'COMMAND TREE' on page 3-10 for more details of this subject.

Forgiving Listening

The syntax specification of a command is as follows:

ACQuisition:APERture␣<numeric value>

Where: ACQ and APER specify the shortform, and ACQuisition and APERture specify the longform. However, ACQU or APERT are not allowed and cause a command error.

In program messages either the long or the shortform may be used in upper or lower case letters. You may even mix upper and lower case. There is no semantic difference between upper and lower case in program messages. This instrument behavior is called forgiving listening.

For example, an application program may send the following characters over the bus:

SEND→ `iNp:ImP␣1E6`

The example shows the shortform used in a mix of upper and lower case

SEND→ `Input:Imp␣1E6`

The example shows the a mix of long and shortform anda mixe of upper and lower case.

Notation Habit in Command Syntax

To clarify the difference between short and longform, the shortform in a syntax specification is shown in upper case letters and the remaining part of the longform in lower case letters.

Notice however, that this does not specify the use of upper and lower case characters in the message that you actually sent. Upper and lower case letters, as used in syntax specifications, are only a notation convention to ease the distinction between long and shortform.

■ Syntax of Response Messages

The response of a SCPI instrument to a query (response message unit) consists of one or more parameters (data elements) as the following syntax diagram shows. There is no header returned.

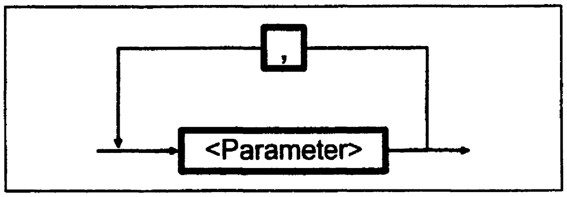


Figure 3-8 *Syntax of a Response Message Unit.*

If there are multiple queries in a program message, the instrument groups the multiple response message units together in one response message according to the following syntax:

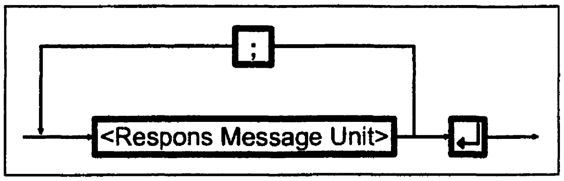


Fig 3-9 *Syntax of a Terminated Response Message.*

The response message terminator (rmt) is always NL^END, where:

NL^END is <new line> code (equal to <line feed> code = ASCII 10 decimal) sent concurrently with the END message. The END message is sent by asserting the EOI line of the GPIB bus.

Responses:

A SCPI instrument always sends its response data in shortform and in capitals.

Example:

You program an instrument with the following command:

SEND→ `:ROSCillator:SOURce␣EXTernal`

Then you send the following query to the instrument:

SEND→ `:ROSCillator:SOURce?`

The instrument will return:

READ← `EXT`

response in shortform and in capitals.

Command Tree

Command Trees like the one below are used to document the SCPI command set in this manual. The keyword (mnemonic) on the root level of the command tree is the name of the subsystem. The following example illustrates the Command Tree of the INPut1 subsystem.

```
<HEADER>                 Parameters
:INPut[1]
 ├→:IMPedance            ␣<Numeric value>|MAX|MIN
 └→:FILTer
     └→[:LPASs]
         └→[:STATe]  ␣<Boolean>
```

Figure 3-10 *Example of an INPut subsystem command tree.*

☞ *The keywords placed in square brackets are optional nodes. This means that you may omit them from the program message.*

Example:

SEND→ `INPUT1:FILTER:LPASS :STATE␣ON`

is the same as

SEND→ `INPUT:FILTER␣ON`

Moving down the Command Tree

The command tree shows the paths you should use for the command syntax. A single command header begins from the root level downward to the 'leaf nodes' of the command tree. (Leaf nodes are the last keywords in the command header, before the parameters.)

- **Example:**

SEND→ `INPut:EVENt:HYSTeresis`

Where: INPut is the root node and HYSTeresis is the leaf node.

Each colon in the command header moves the current path down one level from the root in the command tree. Once you reach the leaf node level in the tree, you can add several leaf nodes without having to repeat the path from the root level.

Just follow the rules below:

– Always give the full header path, from the root, for the first command in a new program message.

– For the following commands within the same program message, omit the header path and send only the leaf node (without colon).

☞ *You can only do this if the header path of the new leaf-node is the same as that of the previous one. If not, the full header path must be given starting with a colon.*

Command header = Header path + leaf node

– Once you send the pmt (program message terminator), the first command in a new program message must start from the root.

- **Example:**

SEND→ `INPut:EVENt:HYSTeresis MIN;LEVel␣0.5`

This is the command where:

INPut:EVENt is the header path and :HYSTeresis is the first leaf-node and LEVel is the second leaf node because LEVel is also a leaf-node under the header path INPut:EVENt.

There is no colon before LEVel!

Parameters

Numeric Data

Decimal data are printed as numerical values throughout this manual. Numeric values may contain both a decimal point and an exponent (base 10).

These numerals are often represented as NRf (NR = NumeRic, f = flexible) format.

■ Keywords

In addition to entering decimal data as numeric values, several keywords can exist as special forms of numeric data, such as MINimum, MAXimum, DEFault, STEP, UP, DOWN, NAN (Not A Number), INFinity, NINF (Negative INFinity). The Command Reference chapters explicitly specify which keywords are allowed by a particular command. Valid keywords for the CNT-8X counters are MAXimum and MINimum.

MINimum

This keyword sets a parameter to its minimum value.

MAXimum

This keyword sets a parameter to its maximum value.

The instrument always allows MINimum and MAXimum as a data element in commands, where the parameter is a numeric value. MIN and MAX values of a parameter can always be queried.

Example:

```
SEND→ INP:LEV?␣MAX
```

This query returns the maximum range value.

■ Suffixes

You can use suffixes to express a unit or multiplier that is associated with the decimal numeric data. Valid suffixes are s (seconds), ms (milliseconds), mohm (megaohm), kHz (kilohertz), mV (millivolt).

Example:

```
SEND→ :SENS:ACQ:APER␣100ms
```

Where: ms is the suffix for the numeric value 100.

Notice that you may also send ms as MS or mS. MS does still mean milliseconds, not Mega Siemens!

Response messages do not have suffixes. The returned value is always sent using standard units such as V, S, Hz, unless you explicitly specify a default unit by a FORMat command.

Boolean Data

A Boolean parameter specifies a single binary condition which is either true or false.

Boolean parameters can be one of the following:

– ON or 1 means condition true.

– OFF or 0 means condition false.

■ Example

```
SEND→ :SYST:TOUT_ON or
      :SYST:TOUT_1
```

This switches timeout monitoring on.
A query, for instance :SYSTem:TOUT?, will return 1 or 0; never ON or OFF.

Expression Data

You must enclose expression program data in parenthesis (). Three possibilities of expression data are as follows:

- <numeric expression data> <parameter list>
- <channel list>

An example of <numeric expression data> is: (X – 10.7E6) This subtracts a 10.7 MHz intermediate frequency from the measured result.

An example of <parameter list> is: (5,0.02) This is a list of two parameters; the first one is 5 and the second one 0.02.

An example of <channel list> is: (@3),(@1) This specifies channel 3 as the main channel and channel 1 as the second channel.

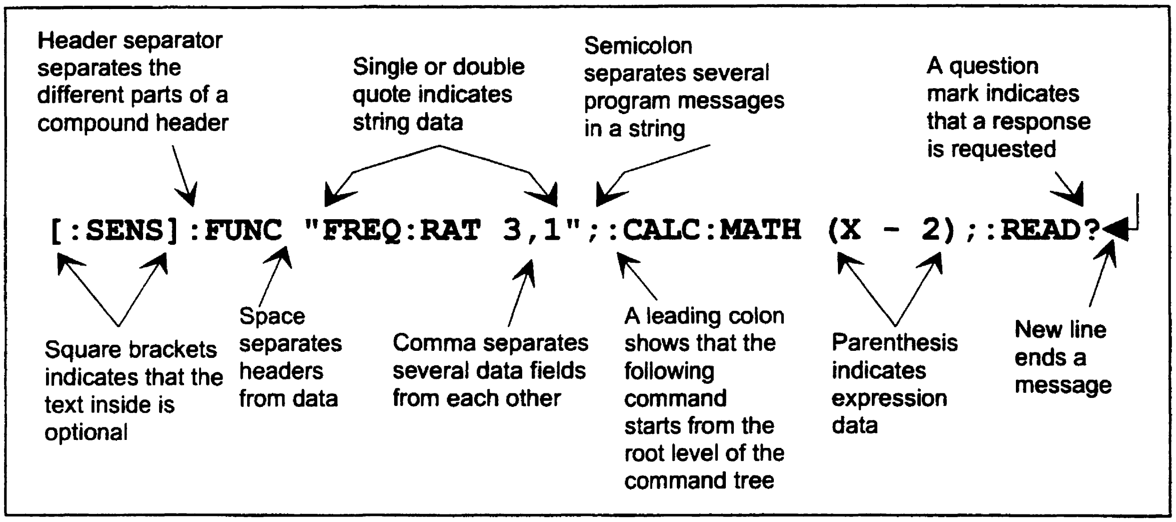
Other Data Types

Other data types that can be used for parameters are the following:

- String data: Always enclosed between single or double quotes, for example "This is a string" or 'This is a string.'
- Character data: For this data type, the same rules apply as for the command header mnemonics. For example: POSitive, NEGative, EITHer.
- Non-decimal data: For instance, #H3A for hexa decimal data.
- Block data: Used to transfer any 8-bit coded data. This data starts with a preamble that contains information about the length of the parameter.

  Example:
  ```
  #218INP:IMP_50;SENS_10
  ```

Summary



```
[:SENS]:FUNC "FREQ:RAT 3,1";:CALC:MATH (X - 2);:READ?↵
```

Header separator separates the different parts of a compound header
Single or double quote indicates string data
Semicolon separates several program messages in a string
A question mark indicates that a response is requested
Square brackets indicates that the text inside is optional
Space separates headers from data
Comma separates several data fields from each other
A leading colon shows that the following command starts from the root level of the command tree
Parenthesis indicates expression data
New line ends a message

Macros

A macro is a single command, that represents one or several other commands, depending on your definition. You can define 25 macros of 40 characters in the counter. One macro can address other macros, but you cannot call a macro from within itself (recursion). You can use variable parameters that modify the macro.

Use macros to do the following:

– Provide a shorthand for complex commands.

– Cut down on bus traffic.

Macro Names

You can use both commands and queries as macro labels. The label cannot be the same as common commands or queries. If a macro label is the same as a CNT-8X command, the counter will execute the macro when macros are enabled (*EMC␣1) and it will execute the CNT-8X command when macros are disabled (*EMC␣0).

Data Types within Macros

The commands to be performed by the macro can be sent both as block and string data.

String data is the easiest to use since you don't have to count the number of characters in the macro. However, there are some things you must keep in mind:

Both double quote (") and single quote (') can be used to identify the string data. If you use a controller language that uses double quotation marks to define strings within the language (like BASIC) we recommend that you use block data instead, and use single quotes as string identifiers within the macro.

When using string data for the commands in a macro, remember to use a different type of string data identifiers for strings within the macro. If the macro should for instance set the input slope to positive and select the period function, you must type:

```
":Inp:slope␣pos;:Func␣'PER␣1'"
```

or

```
':Inp:slope␣pos;:Func␣"PER␣1"'
```

Define Macro Command

*DMC assigns a sequence of commands to a macro label. Later when you use the macro label as a command, the counter will execute the sequence of commands.

Use the following syntax:

*DMC <macro-label>, <commands>

■ Simple Macros

Example:

```
SEND→ *DMC␣'MyInputSetting'
      #255:INP:IMP␣50;HYST␣1
      ;LEV␣0.55;:INP:HYST:AUTO
      ␣0;
```

This example defines a macro MyInputSetting, which sets the impedance to 50 Ω, sets the sensitivity to 1V, the trigger level to +0.55V, and switches off auto sensitivity and auto trigger level.

■ Macros with Arguments

You can pass arguments (variable parameters) with the macro. Insert a dollar sign ($) followed by a single digit in the range 1 to 9 where you want to insert the parameter. See the example below.

When a macro with defined arguments is used, the first argument sent will replace any occurrence of $1 in the definition; the second argument will replace $2, etc.

Example:

```
SEND→ *DMC␣'AUTO',#247
      :INP:HYST:AUTO␣$1;
      :INP:IMP␣$2
```

This example defines a macro AUTO, which takes two arguments, i.e., auto «ON|OFF|ONCE» ($1) and impedance «50|1E6» ($2) .

```
SEND→ AUTO␣OFF,50
```

Switches off both auto sensitivity and auto trigger level and sets the input impedance to 50Ω.

Deleting Macros

Use the *PMC (purge macro) command to delete all macros defined with the *DMC command. This removes all macro labels and sequences from the memory. To delete only one macro in the memory, use the :MEMory:DELete:MACRo command.

You cannot overwrite a macro; you must delete it before you can use the same name for a new macro.

Enabling and Disabling Macros

■ *EMC Enable Macro Command

When you want to execute a CNT-8X command or query with the same name as a defined macro, you need to disable macro execution. Disabling macros does not delete stored macros; it just hides them from execution.

Disabling: `*EMC␣0` disables all macros.
Enabling: `*EMC␣1`

■ *EMC? Enable Macro Query

Use this query to determine if macros are enabled.

Response:

1 macros are enabled
0 macros are disabled

How to Execute a Macro

Macros are disabled after *RST, so to be sure, start by enabling macros with *EMC 1. Now macros can be executed by using the macro labels as commands.

■ Example:

```
SEND→ *DMC␣'LIMITMON','
      :CALC:STAT␣ON;
      :CALC:LIM:STAT␣ON;
      :CALC:LIM:LOW:DATA
      $1;STAT␣ON;
      :CALC:LIM:UPP:DATA
      $2;STAT␣ON'
SEND→ *EMC␣1
```

Now sending the command

```
SEND→ LIMITMON␣1E6,1.1E6
```

will switch on the limit monitoring to alarm between the limits 1 MHz and 1.1 MHz.

Retrieve a Macro

■ *GMC? Get Macro Contents Query

This query gives a response containing the definition of the macro you specified when sending the query.

Example using the above defined macro:

```
SEND→ *GMC?_'LIMITMON'
READ← #292:CALC:STAT
        ON;:CALC:LIM:STAT ON;
        :CALC:LIM:LOW:DATA
        $1;STAT_ON;
        :CALC:LIM:UPP:DATA
        $2;STAT_ON'
```

■ *LMC? Learn Macro Query

This query gives a response containing the labels of all the macros stored in the Timer/Counter.

Example:

```
SEND→ *LMC?
 READ←"MYINPSETTING","LIMITMON
      "
```

Now there are two macros in memory, and they have the following labels: “MYINPSETTING” and “LIMITMON”.

Status Reporting System

Introduction

Status reporting is a method to let the controller know what the counter is doing. You can ask the counter what status it is in whenever you want to know.

You can select some conditions in the counter that should be reported in the Status Byte Register. You can also select if some bits in the Status Byte should generate a Service Request (SRQ).
(An SRQ is the instrument's way to call the controller for help.)

Read more about the Status Subsystem in Chapter 6.

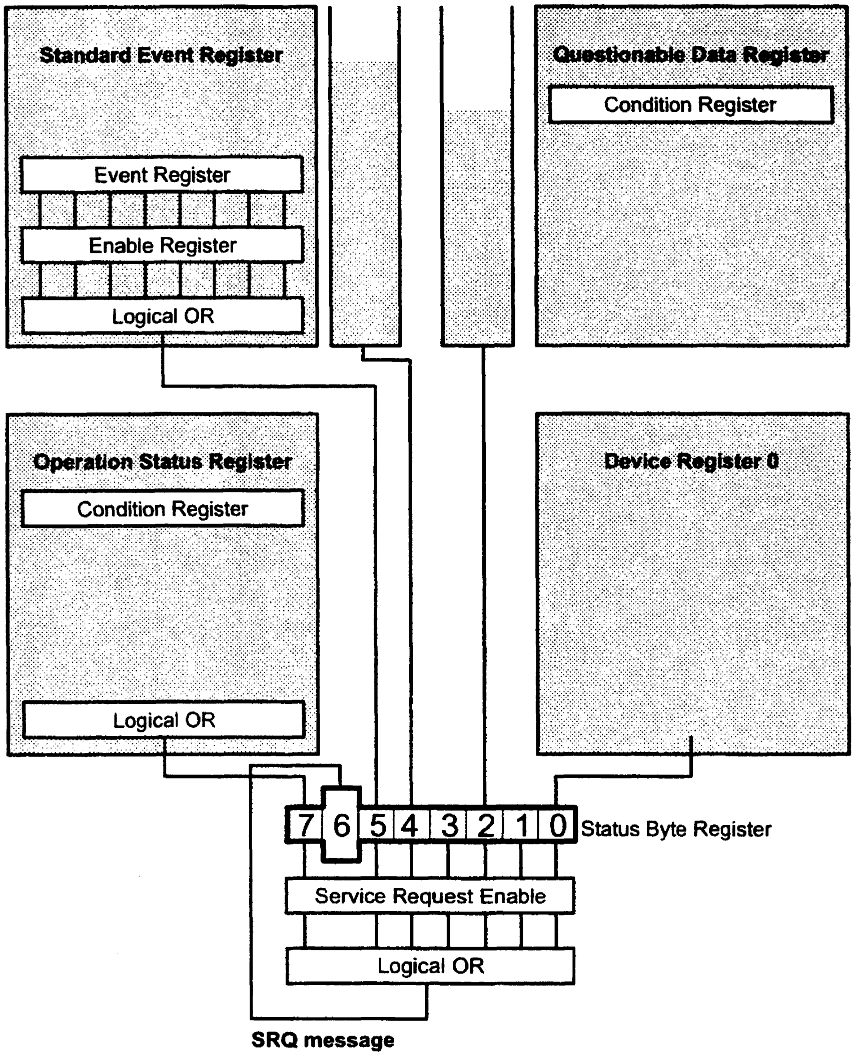


Figure 3-11 *CNT-8X Status register structure.*

Error Reporting

The counter will place a detected error in its Error Queue. This queue is a FIFO (First-In First-Out) buffer. When you read the queue, the first error will come out first, the last error last.

If the queue overflows, an overflow message is placed last in the queue, and further errors are thrown away until there is room in the queue again.

■ Detecting Errors in the Queue

Bit 2 in the Status Byte Register shows if the instrument has detected errors. It is also possible to enable this bit for Service Request on the GPIB. This can then interrupt the GPIB controller program when an error occurs.

■ Read the Error/Event Queue

This is done with the :SYSTem:ERRor? query.

Example:

```
SEND→ :SYSTem:ERRor?
READ← -100,␣"Command␣Error"
```

The query returns the error number followed by the error description.

Further description of all error numbers can be found in the Error Messages chapter

If more than one error occurred, the query will return the error that occurred first. When you read an error you will also remove it from the queue. You can read the next error by repeating the query. When you have read all errors the queue is empty, and the :SYSTem:ERRor? query will return:

0, "No error"

When errors occur and you do not read these errors, the Error Queue may overflow. Then the instrument will overwrite the last error in the queue with the following:

–350, "Queue overflow"

If more errors occur, they will be discarded.

■ Standardized Error Numbers

The instrument reports four classes of standardized errors in the Standard Event Status and in the Error/Event Queue as shown in the following table:

Error Class	Range of Error Numbers	Standard Event Register
Command Error	–100 to –199	bit 5 - CME
Execution Error	–200 to –299	bit 4 - EXE
Device- specific Error	–300 to –399 +100 to +32767	bit 3 - DDE
Query Error	–400 to –499	bit 2 -QYE

■ Command Error

This error shows that the instrument detected a syntax error.

■ Execution Error

This error shows that the instrument has received a valid program message which it cannot execute because of some device specific conditions.

■ Device-specific Error

This èrror shows that the instrument could not properly complete some device specific operations.

■ Query Error

This error will occur when the Message Exchange Protocol is violated, for example, when you send a query to the instrument and then send a new command without first reading the response data from the previous query. Also, trying to read data from the instrument without first sending a query to the instrument will cause this error.

Initialization and Resetting

Reset Strategy

There are three levels of initialization:

- Bus initialization
- Message exchange initialization
- Device initialization

■ Bus Initialization

This is the first level of initialization. The controller program should start with this which initializes the IEEE-interfaces of all connected instruments. It puts the complete system into remote enable (REN-line active) and the controller sends the interface clear (IFC) command. The command or the command sequence for this initialization is controller and language dependent. Refer to the user manual of the system controller in use.

■ Message Exchange Initialization

Device clear is the second level of initialization. It initializes the bus message exchange, but does not affect the device functions.

Device clear can be signaled either with DCL to *all* instruments or SDC (Selective device-clear) only to the addressed instruments. The instrument action on receiving DCL and SDC is identical, they will do the following:

- Clear the input buffer.
- Clear the output queue.
- Reset the parser.
- Clear any pending commands.

The device-clear commands will not do the following:

- Change the instrument settings or stored data in the instrument.
- Interrupt or affect any device operation in progress.
- Change the status byte register other than clearing the MAV bit as a result of clearing the output queue.

☞ *Many older IEEE-instruments, that are not IEEE-488.2 compatible returned to the power-on default settings when receiving a device-clear command. IEEE-488.2 does not allow this.*

When to use a Device-clear Command

The command is useful to escape from erroneous conditions without having to alter the current settings of the instrument. The instrument will then discard pending commands and will clear responses from the output queue. For example; suppose you are using the Counter in an automated test equipment system where the controller program returns to its main loop on any error condition in the system or the tested unit. To ensure that no unread query response remains in the output queue and that no unparsed message is in the input buffer, it is wise to use device-clear. (Such remaining responses and commands could influence later commands and queries.)

■ Device Initialization

The third level of initialization is on the device level. This means that it concerns only the addressed instruments.

*The *RST Command*

Use this command to reset a device. It initializes the device-specific functions in the Counter.

The following happens when you use the *RST command:

- You set the Counter-specific functions to a known default state. The *RST condition for each command is given in the command reference chapters.
- You disable macros.
- You set the counter in an idle state (outputs are disabled), so that it can start new operations.

*The *CLS Command*

Use this command to clear the status data structures. See 'Status Reporting system' in this chapter.

The following happens when you use the *CLS command:

- The instrument clears all event registers summarized in the status byte register.
- It empties all queues, which are summarized in the status byte register, except the output queue, which is summarized in the MAV bit.

Chapter 4

Programming Examples

Introduction

Each program example in this chapter is written for IBM-PC compatible computers equipped with the National Instruments PC-IIA. In addition to that, many of the examples are written in both 'GW-BASIC' and 'C'.

Even if you do not have these interface board or use these computer languages, look at the examples anyway. They give you a good insight on how to program the instrument efficiently.

☞ *To be able to run these programs without modification, the address of your counter must be set to 10.*

Example 1. Limit Testing

Example 2. REAL Data Format

Example 3. Frequency Profiling

Example 4. Fast Sampling

Example 5. Status Reporting

Example 6. Statistics, this example is only for PM6680B and PM6681

GW-Basic for National Instruments PC-IIA

Setting up the interface

All these programs start with a declaration containing three lines of setup information for the interface. This declaration must be merged with the programs prior to running them. The declaration is printed below, but it is also available as a file on the diskettes delivered with your interface. The file name is DECL.BAS.

```
20 CLEAR ,60000! : IBINIT1=60000! : IBINIT2=IBINIT1+3 : BLOAD
“bib.m”,IBINIT1
30 CALL
IBINIT1(IBFIND,IBTRG,IBCLR,IBPCT,IBSIC,IBLOC,IBPPC,IBBNA,
IBONL,IBRSC,IBSRE,IBRSV,IBPAD,IBSAD,IBIST,IBDMA,IBEOS,IBTMO,IBEO
T, IBRDF,IBWRTF,IBTRAP,IBDEV,IBLN)
40 CALL
IBINIT2(IBGTS,IBCAC,IBWAIT,IBPOKE,IBWRT,IBWRTA,IBCMD,IBCMDA,
IBRD,IBRDA,IBSTOP,IBRPP,IBRSP,IBDIAG,IBXTRC,IBRDI,IBWRTI,IBRDIA,
IBWRTIA,IBSTA%,IBERR%,IBCNT%)
```

1. Limit Testing

This program uses limit testing to check that the frequency is above a preset value.

```
50 CNTNAME$ = "DEV10"
60 CALL IBFIND (CNTNAME$, CNT%)
70 '
80 '
90 ' — Set continuous frequency measurement —
100 WRT$ = "*RST; *CLS; :FUNC 'FREQ 1'; :INIT:CONT ON"
110 CALL IBWRT (CNT%, WRT$)
120 '
130 ' — Enable limit monitoring, limit 1 MHz —
140 WRT$ = ":CALC:LIM ON; LIM:UPP 1E6; UPP:STATE ON"
150 CALL IBWRT (CNT%, WRT$)
160 WRT$ = ":STAT:DREG0:ENAB 2; *SRE 1"
170 CALL IBWRT (CNT%, WRT$)
180 '
190 ' — Wait until the limit is passed —
200 PRINT "Waiting for limit to be passed"
210 MASK% = &H800
220 CALL IBWAIT (CNT%, MASK%)
230 '
240 ' — Read status and device status register —
250 CALL IBRSP (CNT%, SPR%)
260 '
270 ' — Read frequency —
280 WRT$ = "READ?"
290 CALL IBWRT (CNT%, WRT$)
300 MSG$ = SPACE$(255)
310 CALL IBRD (CNT%, MSG$)
320 PRINT "Frequency = "; LEFT$(MSG$, IBCNT%)
330 WRT$ = ":STAT:DREG0:EVEN?"
340 CALL IBWRT (CNT%, WRT$)
350 MSG$ = SPACE$(255)
360 CALL IBRD (CNT%, MSG$)
370 '
380 ' — Disable continuous measurement —
390 WRT$ = ":INIT:CONT OFF"
400 CALL IBWRT (CNT%, WRT$)
410 END
```

3. Frequency Profiling

Frequency profiling visualizes frequency variations for a certain time. This program gives an output file called:
PROFILE.DAT. If this file is imported to a spreadsheet program, for instance Excel, you can create a graph like the one in the figure below.

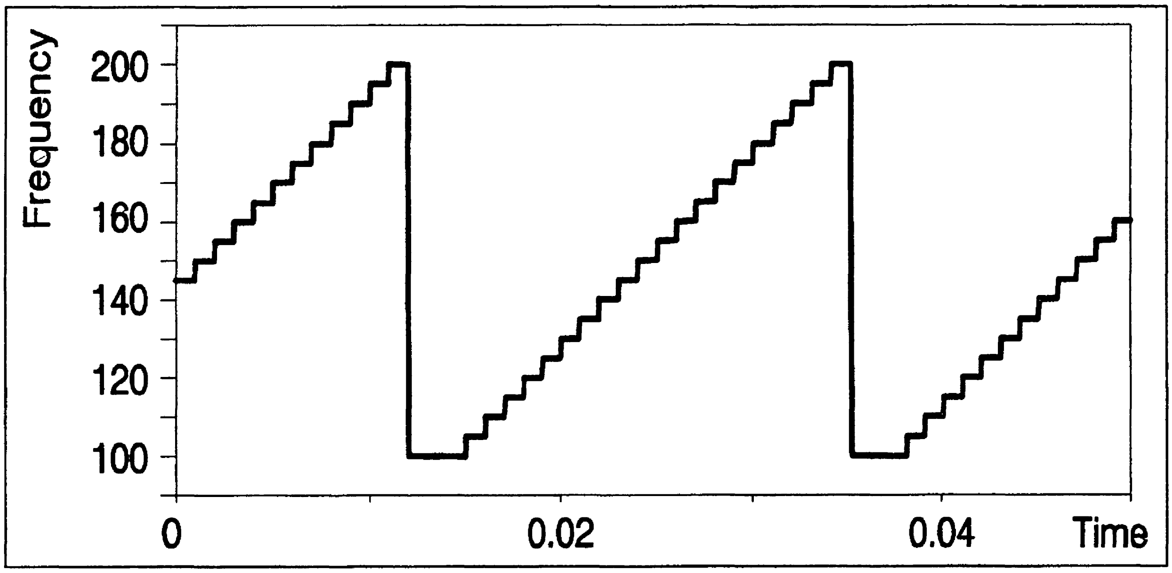


Figure 4-1 *This figure is the results of frequency profiling on a sweep generator.*

```
50 '
60 OPEN "O", 1, "PROFILE.DAT"
70 CNTNAME$ = "DEV10"
80 CALL IBFIND (CNTNAME$, CNT%)
90 '
100 '
110 ' — Enable arming, etc. —
120 WRT$ = ":TRIG:COUN 1; :ARM:COUN 1; SOUR EXT4"
130 CALL IBWRT(CNT%, WRT$)
140 WRT$ = ":INP:LEV:AUTO ONCE
150 CALL IBWRT(CNT%, WRT$)
160 WRT$ = ":DISP:ENAB OFF; :ACQ:APER 1E-6"
170 CALL IBWRT(CNT%, WRT$)
180 '
190 ARMDELAY = .0000002
```

```
200 '
210 ' ==== CAPTURE PROFILE =====
220 '
230 PRINT “Profiling”
240 '
250 FOR I=0 TO 999
260    ' — Set arming delay time --
270    WRT$ = “:ARM:DEL” + STR$(ARMDELAY)
280    CALL IBWRT(CNT%, WRT$)
290    '
300    ' — Measure and read result --
310    WRT$ = “READ?”
320    CALL IBWRT(CNT%, WRT$)
330    MSG$ = SPACE$(255)
340    CALL IBRD(CNT%, MSG$)
350    '
360    ' — Write arming delay time and result to file --
370    PRINT#1, STR$(ARMDELAY), LEFT$(MSG$, INSTR(MSG$,
       CHR$(10)))
380    '
390    ' — Increase arming delay --
400    ARMDELAY = ARMDELAY + .0000001
410 NEXT I
420 '
430 WRT$ = “:DISP:ENAB ON”
440 CALL IBWRT(CNT%, WRT$)
450 '
460 CLOSE 1
470 END
```

4. Fast Sampling

This program makes a quick array measurement and stores the results in the internal memory of the counter. Then it writes the results to a file called MEAS.DAT. The measurement results as a function of the samples can be visualized in a spreadsheet program such as Excel.

```
50 '
60 OPEN "O", 1, "MEAS.DAT"
70 CNTNAME$ = "DEV10"
80 CALL IBFIND (CNTNAME$, CNT%)
90 '
100 '
110 ' — Clear status —
120 WRT$ = "*CLS"
130 CALL IBWRT(CNT%, WRT$)
140 '
150 ' — Enable 1000 measurement with maximum speed —
160 WRT$ = ":TRIG:COUN 1000; :ARM:COUN 1"
170 CALL IBWRT (CNT%, WRT$)
180 WRT$ = ":INP:LEV:AUTO ONCE; :CAL:INT:AUTO OFF"
190 CALL IBWRT (CNT%, WRT$)
200 WRT$ = ":DISP:ENAB OFF; :INT:FORM PACKED"
210 CALL IBWRT (CNT%, WRT$)
220 WRT$ = ":ACQ:APER MIN; :AVER:STAT OFF"
230 CALL IBWRT (CNT%, WRT$)
240 '
250 ' — Enable SRQ on operation complete —
260 WRT$ = "*ESE 1; *SRE 32"
270 CALL IBWRT (CNT%, WRT$)
280 '
290 ' — Start measurement —
300 PRINT "Measuring"
310 WRT$ = "INIT; *OPC"
320 CALL IBWRT (CNT%, WRT$)
330 '
340 ' — Wait for operation complete —
350 MASK = &H800
360 CALL IBWAIT (CNT%, MASK)
370 '
380 ' — Read status and event status register —
390 CALL IBRSP (CNT%, SPR%)
400 WRT$ = "*ESR?"
410 CALL IBWRT (CNT%, WRT$)
420 MSG$ = SPACE$(255)
430 CALL IBRD (CNT%, MSG$)
```

```
440 '
450 PRINT "Fetching result"
460 '
470 FOR I=0 TO 999
480   ' — Fetch one result —
490   WRT$ = "FETCH?"
500   CALL IBWRT (CNT%, WRT$)
510   MSG$ = SPACE$(255)
520   CALL IBRD (CNT%, MSG$)
530   '
540   ' — Write result to file —
550   PRINT#1, I, LEFT$(MSG$, INSTR(MSG$, CHR$(10)))
560 NEXT I
570 '
580 WRT$ = ":DISP:ENAB ON"
590 CALL IBWRT(CNT%, WRT$)
600 '
610 CLOSE 1
620 END
```

5. Status Reporting

This program sets up the status reporting for Service Request on ‘Message Available’ and ‘Command’, ‘Execution’, or Query’ errors.

The program reads a command from the controller keyboard and sends it to the counter, then it checks the status byte using Serial Poll. It determines the reason for Service Request, and reads query responses and error messages.

```
50 CNTNAME$ = “DEV10"
60 CALL IBFIND (CNTNAME$, CNT%)
70 '
80 '
90 ' — CLEAR STATUS —
100 WRT$ = “*cls”
110 CALL IBWRT (CNT%, WRT$)
120 '
130 ' — SET EVENT STATUS ENABLE —
140 ' Enable Command Error, Execution Error and Query Error
150 WRT$ = “*ese 52"
160 CALL IBWRT (CNT%, WRT$)
170 '
180 ' — SET SERVICE REQUEST ENABLE —
190 ' Enable Service Request on Event Status and Message
      Available
200 WRT$ = “*sre 48"
210 CALL IBWRT (CNT%, WRT$)
220 '
230 ' ======== MAIN LOOP ====================================
240 WHILE 1
250    '
260    ' — ENTER COMMAND STRING AND SEND TO COUNTER —
270    LINE INPUT “Enter command string (<CR> to end):”, CMD$
280    IF CMD$ = “” GOTO 760
290    CMD$ = CMD$
300    CALL IBWRT (CNT%, CMD$)
310    ' WAIT for execution
320    FOR I=1 TO 1000
330       CALL IBRSP (CNT%, SPR%)
340       IF SPR% AND 16 THEN GOTO 380
350    NEXT I
360    '
370    ' — READ STATUS BYTE —
380    IF SPR% <> 0 THEN PRINT “Status byte = ”; SPR%
390    ELSE GOTO 750
400       '
```

```
410      ‘ — CHECK MESSAGE AVAILABLE BIT —
420      WHILE SPR% AND 16
430         PRINT “  Message available bit set”
440         MSG$ = SPACE$(255)
450         CALL IBRD (CNT%, MSG$)
460         LFPOS = INSTR(MSG$, CHR$(10))
470         IF LFPOS <> 0 THEN PRINT “Response = ” LEFT$(MSG$,
            LFPOS)
480         IF LFPOS = 0 THEN PRINT “Response = ”; MSG$
490         CALL IBRSP (CNT%, SPR%)
500      WEND
510      ‘
520      ‘ — CHECK EVENT STATUS BIT —
530      IF NOT SPR% AND 32 GOTO 750
540         PRINT “  Event status bit set”
550         WRT$ = “*esr?”
560         CALL IBWRT (CNT%, WRT$)
570         ESR$ = SPACE$(255)
580         CALL IBRSP (CNT%, SPR%)
590         CALL IBRD (CNT%, ESR$)
600         ESR% = VAL(ESR$)
610         IF ESR% AND 32 THEN PRINT “    Command error”
620         IF ESR% AND 16 THEN PRINT “    Execution error”
630         IF ESR% AND 4  THEN PRINT “    Query error”
640         ‘
650         ‘ — READ ERROR MESSAGES —
660         WRT$ = “syst:err?”
670         ERRMESS$ = SPACE$(255)
680         CALL IBWRT (CNT%, WRT$)
690         CALL IBRD (CNT%, ERRMESS$)
700         WHILE NOT INSTR(ERRMESS$, “No error”) <> 0
710            PRINT LEFT$(ERRMESS$, INSTR(ERRMESS$, CHR$(10)))
720            CALL IBWRT (CNT%, WRT$)
730            CALL IBRD (CNT%, ERRMESS$)
740         WEND
750 WEND
760 PRINT “PROGRAM TERMINATED”
770 END
```

6. Statistics

(Only for PM6680B and PM6681)

In this example, the counter makes 10000 measurements and uses the statistical functions to determine MAX, MIN, MEAN, and Standard Deviation. All four results are sent to the controller.

```
50 CNTNAME$ = “DEV10"
60 CALL IBFIND (CNTNAME$, CNT%)
70 '
80 '
90 WRT$ = “*RST; *CLS; *SRE 16; :FUNC ‘Freq 1’; :ACQ:APER MIN”
100 CALL IBWRT (CNT%, WRT$)
110 WRT$ = “:INP:LEV:AUTO Off”
120 CALL IBWRT (CNT%, WRT$)
130 '
140 ' — Enable statistics on 10000 measurements —
150 WRT$ = “:CALC:AVER:STAT ON; COUN 10000"
160 CALL IBWRT (CNT%, WRT$)
170 '
180 ' ==== Start measurement ====
190 WRT$ = “:Init; *OPC?”
200 CALL IBWRT (CNT%, WRT$)
210 '
220 ' — Wait for operation complete (MAV) —
230 PRINT “WAITING FOR MEASUREMENT TO GET READY”
240 MASK% = &H800
250 CALL IBWAIT (CNT%, MASK%)
260 '
270 ' — Read status and response —
280 CALL IBRSP (CNT%, SPR%)
290 MSG$ = SPACE$(255)
300 CALL IBRD (CNT%, MSG$)
310 '
320 ' — Maximum —
330 WRT$ = “:CALC:AVER:TYPE MAX; :CALC:IMM?”
340 CALL IBWRT (CNT%, WRT$)
350 MSG$ = SPACE$(255)
360 CALL IBRD (CNT%, MSG$)
370 PRINT “MAXIMUM = ”; LEFT$(MSG$, IBCNT%)
380 '
390 ' — Minimum —
400 WRT$ = “:CALC:AVER:TYPE MIN; :CALC:IMM?”
410 CALL IBWRT (CNT%, WRT$)
420 MSG$ = SPACE$(255)
```

```
430 CALL IBRD (CNT%, MSG$)
440 PRINT “MINIMUM =”; LEFT$(MSG$, IBCNT%)
450 '
460 ' — Mean —
470 WRT$ = “:CALC:AVER:TYPE MEAN; :CALC:IMM?”
480 CALL IBWRT (CNT%, WRT$)
490 MSG$ = SPACE$(255)
500 CALL IBRD (CNT%, MSG$)
510 PRINT “MEAN =”; LEFT$(MSG$, IBCNT%)
520 '
530 ' — Standard deviation —
540 WRT$ = “:CALC:AVER:TYPE SDEV; :CALC:IMM?”
550 CALL IBWRT (CNT%, WRT$)
560 MSG$ = SPACE$(255)
570 CALL IBRD (CNT%, MSG$)
580 PRINT “STANDARD DEVIATION =”; LEFT$(MSG$, IBCNT%)
590 END
```

'C' for National Instruments PC-IIA

1. Limit Testing

This program uses limit testing to check that the frequency is above a preset value.

```
#include “decl.h”
#include <stdio.h>
#include <process.h>
main ()
{
  int                                   Counter, Status, i;
  char                                  InString[80];

  Counter = ibfind(“DEV10");

  /*Set continuous frequency measurement*/
  ibwrt(Counter, “*RST; *CLS; :FUNC ‘Freq 1’; :INIT:CONT ON”,
41);

  /*Enable limit monitoring, limit 1 MHz*/
  ibwrt(Counter, “:CALC:LIM ON; LIM:UPP 1E6; UPP:STAT ON”, 38);
  ibwrt(Counter, “:STAT:DREG0:ENAB 2; *SRE 1", 26);

  /*Wait until the limit is passed*/
  printf(“Waiting for limit to be passed\n”);
  ibwait(Counter, RQS);

  /**Read status and device status register**/
  ibrsp(Counter, &Status);
  ibwrt(Counter, “:STAT:DREG0:EVEN?”, 17);
  ibrd(Counter, InString, 80);

  /*Read frequency**/
  ibwrt(Counter, “READ?”, 5);
  ibrd(Counter, InString, 80);
  InString[ibcnt] = ‘\0’;
  printf(“Frequency = %s\n”, InString);

  /*Disable continuous measurement*/
  ibwrt(Counter, “:INIT:CONT OFF”, 14);

  exit(0);
}
```

2. REAL Data Format

This program uses the REAL data format to speed up the measurement.

/* IEEE 488.2 binary real format follows the ‘little-endian’ format with the most-significant byte first and the least-significant byte last. Intel processors use the ‘big-endian’ format, with the least-significant byte first, so we have to reverse the byte order of the incoming block when running on a PC (Intel processor).

```
#include “decl.h”
#include <stdio.h>
#include <process.h>
#include <conio.h>
main ()
{
  int                                  Counter, i;
  char                                 InString[80];
  double                               DoubleFreq;
  Counter = ibfind(“DEV10");

  /*Make the counter output it’s result in real format*/
  ibwrt(Counter, “:FORM REAL”, 10);

  /*Make continuous measurements until a key is hit*/
  do {

    /*Make a measurement and read the result*/
    ibwrt(Counter, “READ?”, 5);
    ibrd(Counter, InString, 80);

    /*Assign the bytes 3...10 of InString to DoubleFreq bytes
    7...0.
    The format of InString is #18******** , where “********”
    represents the value.*/
    for (i=0; i<8; i++)
      ((unsigned char *)&DoubleFreq)[7-i] = InString[3+i];

    /*Print the result*/
    printf(“%le\n”, DoubleFreq);
  } while (!kbhit());

  /*Restore ascii output format*/
  ibwrt(Counter, “:FORM ASCII”, 11);

  exit(0);
}
```

3. Frequency Profiling

Frequency profiling visualizes frequency variations for a certain time. This program gives an output file called:
PROFILE.DAT. If this file is imported to a spreadsheet program, such as Excel, you can create a graph like the one in the figure below.

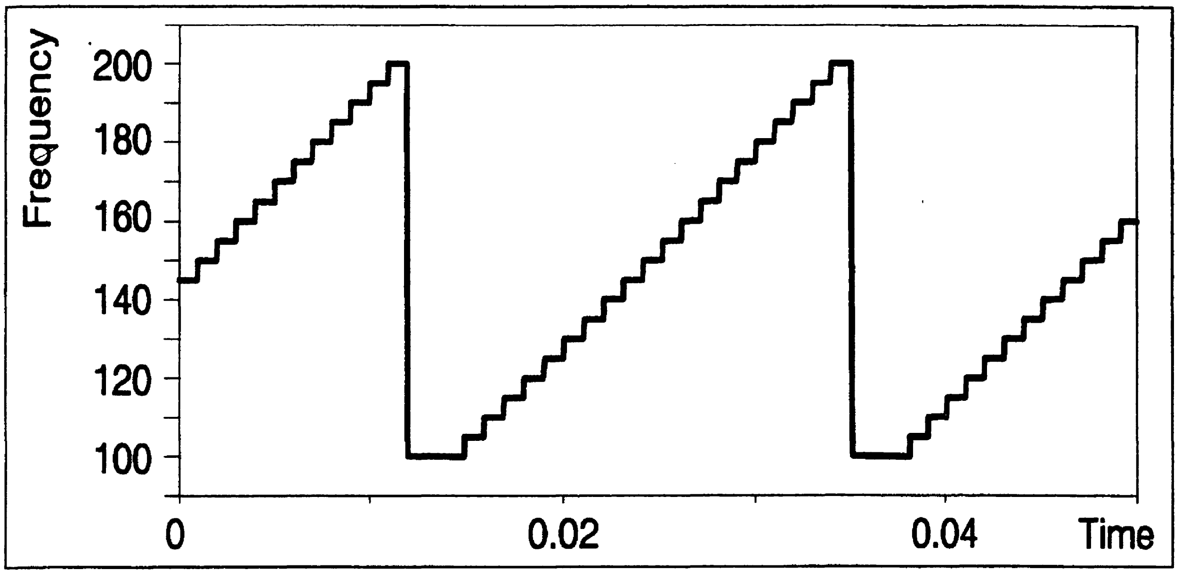


Figure 4-2 *This figure is the results of frequency profiling on a sweep generator.*

```
#include "decl.h"
#include <stdio.h>
#include <process.h>
#include <string.h>

main ()

{
  int                                     Counter, i;
  char                                    ArmString[80],
InString[80];
  double                                  ArmDelay;
  FILE                                    *ofp;

  if (ofp = fopen("PROFILE.DAT", "w")) {
    Counter = ibfind("DEV10");

    /*Enable arming, etc.*/
    ibwrt(Counter, ":TRIG:COUN 1; :ARM:COUN 1", 25);
  ibwrt(Counter, “:INP:LEV:AUTO ONCE”, 18);
  ibwrt(Counter, “:DISP:ENAB OFF; :ACQ:APER 1E-6", 30);

  ArmDelay=200e-9;

  /*CAPTURE PROFILE*/

  Printf(”Profiling”);

  for (i=0; i<1000; i++) {
   /*Set arming delay time*/
   sprintf(ArmString, “:ARM:DEL %le”, ArmDelay);
   ibwrt(Counter, ArmString, strlen(ArmString));

   /*Measure and read result*/
   ibwrt(Counter, “READ?”, 5);
   ibrd(Counter, InString, 80);
   InString[ibcnt] = ‘\0’;

   /*Write arming delay time and result to file*/
   fprintf(ofp, “%le, %s”, ArmDelay, InString);

   /*Increase arming delay*/
   ArmDelay += 100e-9;
  }

  ibwrt(Counter, “:DISP:ENAB ON”, 13);

  /*Close file*/
  Fclose(ofp);

 } else
  printf(“CANT OPEN FILE”);

 exit(0);
}
```

4. Fast Sampling

This program makes a quick array measurement and stores the results in the internal memory of the counter. Then it writes the results to a file called MEAS.DAT. The measurement results as a function of the samples can be visualized in a spreadsheet program, such as Excel.

```
#include “decl.h”
#include <stdio.h>
#include <process.h>
#include <string.h>

main ()

{
  int                                   Counter, Status, i;
  char                                  InString[80];
  FILE                                  *ofp;

  if (ofp = fopen(“MEAS.DAT”, “w”)) {

    Counter = ibfind(“DEV10");

    /*Clear status*/
    ibwrt(Counter, “*CLS”, 4);

    /*Enable 1000 measurement with maximum speed*/
    ibwrt(Counter, “:TRIG:COUN 1000; :ARM:COUN 1", 28);
    ibwrt(Counter, “:INP:LEV:AUTO ONCE; :CAL:INT:AUTO OFF”, 37);
    ibwrt(Counter, “:DISP:ENAB OFF; :INT:FORM PACKED”, 32);
    ibwrt(Counter, “:ACQ:APER MIN; :AVER:STAT OFF”, 32);

    /**Enable SRQ on operation complete**/
    ibwrt(Counter, “*ESE 1; *SRE 32", 15);

    /*Start measurement*/
    printf(“Measuring\n”);
    ibwrt(Counter, “INIT; *OPC”, 10);

    /*Wait for operation complete*/
    ibwait(Counter, RQS);

    /**Read status and event status register**/
    ibrsp(Counter, &Status);
    ibwrt(Counter, “*ESR?”, 5);
    ibrd(Counter, InString, 80);
    printf(“Fetching result”);

    for (i=0; i<1000; i++) {
      /*Fetch one result*/
      ibwrt(Counter, “FETCH?”, 6);
      ibrd(Counter, InString, 80);
      InString[ibcnt] = ‘\0’;

      /*Write result to file*/
      fprintf(ofp, “%d, %s”, i, InString);
    }

    ibwrt(Counter, “:DISP:ENAB ON”, 13);

    /*Close file*/
    Fclose(ofp);

  } else
    printf(“CANT OPEN FILE”);

  exit(0);
}
```

6. Statistics

(Only for PM6680B and PM6681)

In this example, the counter makes 10000 measurements and uses the statistical functions to determine MAX, MIN, MEAN, and Standard Deviation. All four results are sent to the controller.

```
#include “decl.h”
#include <stdio.h>
#include <process.h>

main ()

{
 int                                    Counter, Status, i;
 char                                   InString[80];

 Counter = ibfind(“DEV10");
 ibwrt(Counter, “*CLS; *SRE 16", 13);
 ibwrt(Counter, “*RST; :FUNC ‘Freq 1’; :ACQ:APER MIN”, 38);
 ibwrt(Counter, “:INP:LEV:AUTO OFF”, 17);

 /*Enable statistics on 10000 measurements*/
 ibwrt(Counter, “:CALC:AVER:STAT ON; COUN 10000", 30);
 ibwrt(Counter, “:TRIG:COUN 10000", 16);

 /*Start measurement*/
 ibwrt(Counter, “:Init; *OPC?”, 12);

 /*Wait for operation complete (MAV)*/
 printf(“Waiting for measurement to get ready\n”);
 ibwait(Counter, RQS);

 /**Read status and response**/
 ibrsp(Counter, &Status);
 ibrd(Counter, InString, 80);

 /*Read maximum value**/
 ibwrt(Counter, “:CALC:AVER:TYPE MAX; :CALC:IMM?”, 31);

 ibrd(Counter, InString, 80);
 InString[ibcnt] = ‘\0’;
 printf(“Maximum = %s\n”, InString);
 /*Read minimum value*/
 ibwrt(Counter, “:CALC:AVER:TYPE MIN; :CALC:IMM?”, 31);
 ibrd(Counter, InString, 80);
 InString[ibcnt] = ‘\0’;
 printf(“Minimum = %s\n”, InString);

 /*Read mean value*/
 ibwrt(Counter, “:CALC:AVER:TYPE MEAN; :CALC:IMM?”, 32);
 ibrd(Counter, InString, 80);
 InString[ibcnt] = ‘\0’;
 printf(“Mean = %s\n”, InString);

 /*Read standard deviation value*/
 ibwrt(Counter, “:CALC:AVER:TYPE SDEV; :CALC:IMM?”, 32);
 ibrd(Counter, InString, 80);
 InString[ibcnt] = ‘\0’;
 printf(“Standard deviation = %s\n”, InString);

 exit(0);
}
```

This side is intentionally left blank.

Chapter 5

Instrument Model

Introduction

The figure below shows how the instrument functions are categorized. This instrument model is fully compatible with the SCPI generalized instrument model.

The generalized SCPI instrument model, contains three major instrument categories as shown in the following table:

Function	**Instrument type**	**Examples**
Signal acquisition	Sense instruments	Voltmeter, Oscilloscope, Counter
Signal generation	Source instruments	Pulse generator, Power supply
Signal routing	Switch instruments	Scaners,(de)-multiplexers

An instrument may use a combination of the above functions. The CNT-8X counters belong to the signal acquisition category, and only that category is described in this manual.

The instrument model in Figure 5-1 defines where elements of the counter language are assigned in the command hierarchy. The major signal function areas are shown broken into blocks. Each of these blocks are major command sub-trees in the counter command language.

The instrument model also shows how measurement data and applied signals flow through the instrument. The model does not include the administrative data flow associated with queries, commands, performing calibrations etc.

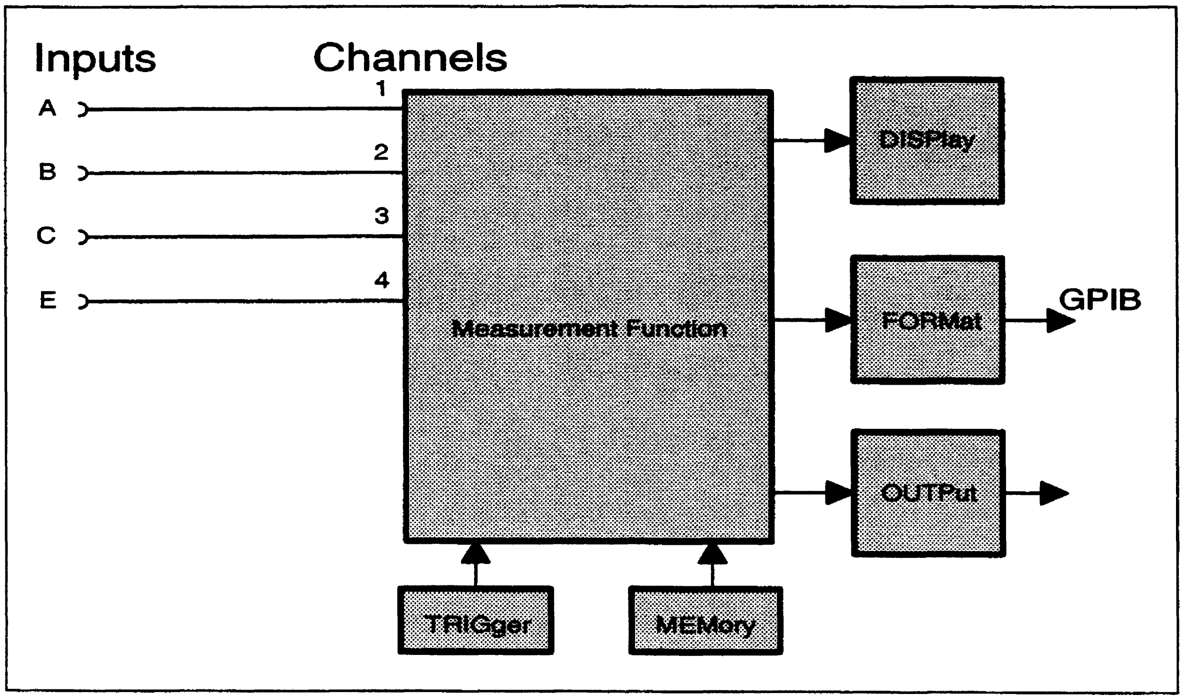


Figure 5-1 *CNT-8X Instrument model. Note that Input B (channel 2) is not available on PM6685.*

Measurement Function Block

The measurement function block converts the input signals into an internal data format that is available for formatting into GPIB bus data. The measurement function is divided into three different blocks: INPut, SENSe and CALCulate. See Figure 5-3.

■ INPut

The INPut block performs all the signal conditioning of the input signal before it is converted into data by the SENSe block. The INPut block includes coupling, impedance, filtering etc.

■ SENSe

The SENSe block converts the signals into internal data that can be processed by the CALCulate block. The SENSe commands control various characteristics of the measurement and acquisition process. These include: gate time, measurement function, resolution, etc.

■ CALCulate

The CALCulate block performs all the necessary calculations to get the required data. These calculations include: calibration, statistics, mathematics, etc.

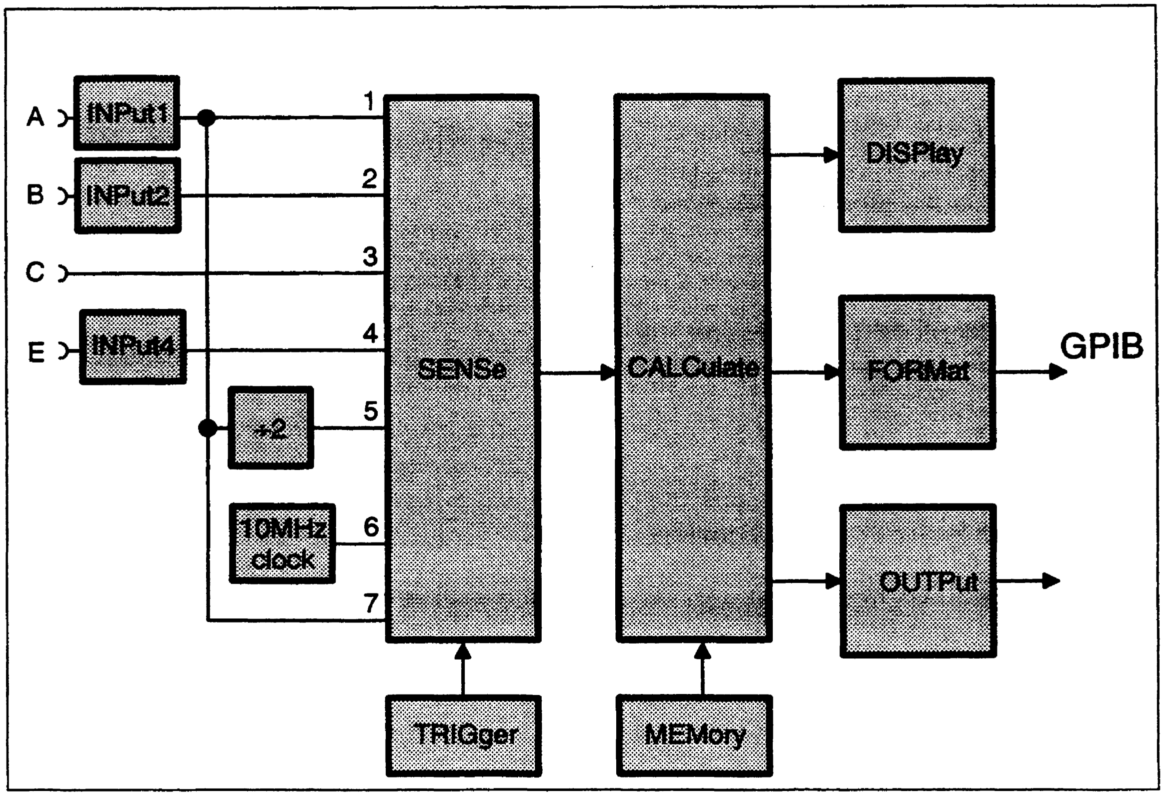


Figure 5-2 CNT-8X Measurement model. Note that Input B (channel 2) is not available on PM6685

Other Subsystems

In addition to the major functions (subsystems), there are several other subsystems in the instrument model.

The different blocks have the following functions.

■ CALibration

This subsystem controls the calibration of the interpolators used to increase the resolution of the CNT-8X counters.

■ DISPlay

Commands in this subsystem control what data is to be present on the display and whether the display is on or off.

■ FORMat

The FORMat block converts the internal data representation to the data transferred over the external GPIB interface. Commands in this block control the data type to be sent over the external interface.

■ MEMory

The MEMory block holds macro and instrument state data inside the counter.

■ OUTPut

This subsystem controls the analog output available in the CNT-8X counters.

■ STATus

This subsystem can be used to get information about what is happening in the instrument at the moment.

■ Synchronization

This subsystem can be used to synchronize the measurements with the controller.

■ SYSTem

This subsystem controls some system parameters like timeout.

■ TEST

This subsystem tests the hardware and software of the counter and reports errors.

■ TRIGger

The trigger block provides the counter with synchronization capability with external events. Commands in this block control the trigger and arming functions of the Timer/ Counter.

Order of Execution

- All commands in CNT-8X counters are sequential, i.e., they are executed in the same order as they are received.
- If a new measurement command is received when a measurement is already in progress, the measurement in progress will be aborted unless `*WAI` is used before the command.

MEASurement Function

In addition to the subsystems of the instrument model, which controls the instrument functions, SCPI has signal-oriented functions to obtain measurement results. This group of MEASure functions has a different level of compatibility and flexibility. The parameters used with commands from the MEASure group describe the signal you are going to measure. This means that the MEASure functions give compatibility between instruments, since you don't need to know anything about the instrument you are using. See Figure 5-3.

■ MEASure?

This is the most simple command to use, but it does not offer much flexibility. The `MEASure?` query lets the counter configure itself for an optimal measurement, start the data acquisition, and return the result.

■ CONFigure; READ?

The `CONFigure` command makes the counter choose an optimal setting for the specified measurement. `CONFigure` may cause any device setting to change.

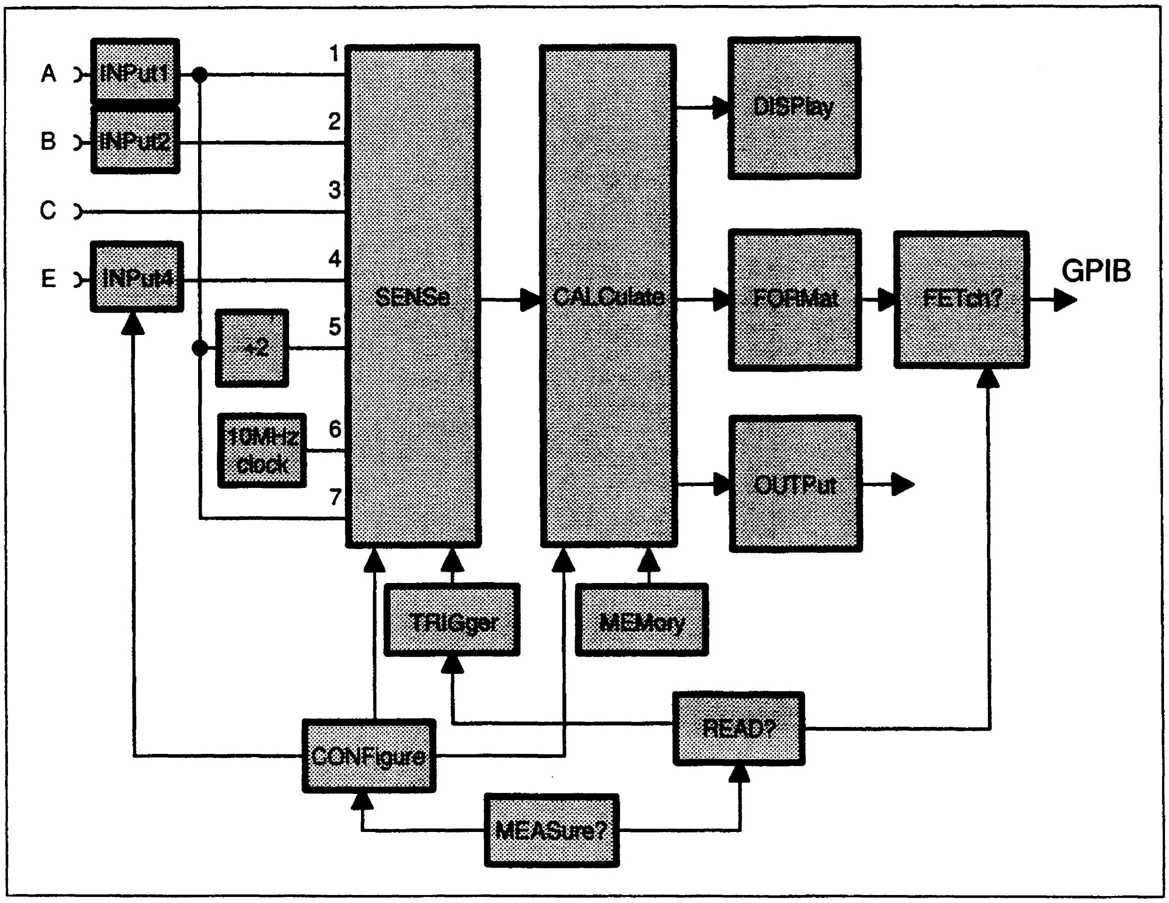


Figure 5-3 *CNT-8X Measurement Function*
Note that Input B (channel 2) is not available on PM6685.

READ? starts the acquisition and returns the result.

This sequence does the same as the MEASure command, but now it is possible to insert commands between CONFigure and READ? to adjust the setting of a particular function (called fine tuning). For instance, you can set an input attenuator at a required value.

- **CONFigure; INITiate;FETCh?**

The READ? command can be divided into the INITiate command, which starts the measurement, and the FETCh? command, which requests the instrument to return the measuring results to the controller.

Versatility of Measurement Commands	
MEASure?	Simple to use,, few additional possibilities.
CONFigure READ?	Somewhat more difficult,, but some extra possibilities.
CONFigure INITiate FETCh?	Most difficult to use,, but many extra features.

Chapter 6

Using the Subsystems

Introduction

Although SCPI is intended to be self explanatory, we feel that some hints and tips on how to use the different subsystems may be useful. This chapter does not explain each and every command, but only those for which we believe extra explanations are necessary.

Calculate Subsystem

The calculate subsystem processes the measuring results. Here you can recalculate the result using mathematics, make statistics (not PM6685) and set upper and lower limits for the measuing result that the counter itself monitors and alerts you when the limits are exceeded.

■ Mathematics

The mathematic functions are the same as on the front panel.

■ Statistics

The PM6680B and PM6681 can calculate and display the MIN, MAX, MEAN and standard deviation of a given number of samples. The statistic functions are the same as on the front panel.

■ Limit Monitoring

Limit monitoring makes it is possible to get a service request when the measurement value falls below a lower limit or rises above an upper limit. Two status bits are defined to support limit monitoring. One is set when the results are greater than the UPPer limit, the other is set when the result is less than the LOWer limit. The bits are enabled using the standard `*SRE` command and `:STAT:DREG0:ENAB`. Using both these bits, it is possible to get a service request when a value passes out of a band (UPPer is set at the upper band border and LOWer at the lower border) OR when a measurement value enters a band (LOWer set at the upper band border and UPPer set at the lower border).
Turning the limit monitoring calculations on/off will not influence the status register mask bits which determine whether or not a service request will be generated when a limit is reached. Note that the calculate subsystem is automatically enabled when limit monitoring is switched on. This means that other enabled calculate sub-blocks are indirectly switched on.

Calibration Subsystem

The interpolators used to increase the resolution of the measurement result in the counter must be calibrated to maintain the highest possible accuracy of the counter.

The calibration method of the PM6681 differs from the method used in PM6680B and PM6685.

■ PM6680B, PM6685

The intepolators are automatically calibrated before each measurement. This procedure takes only a fraction of a second, but to increase speed, you can turn off the auto calibration.

■ PM6681

In PM6681, the interpolators are factory calibrated. Calibration must be performed only after repair and can be performed at your local Service centers.

If the calibration is lost for any reason, the counter will show =CAL. LOSt=.

By pressing PRESET you can bypass this message and use the counter anyway, however you must press the front panel key. No bus command takes you past this error message.

This is so that you cannot bypass the message by mistake, and run a test system without a calibrated instrument.

Configure Function

The CONFigure command sets up the counter to make the same measurements as the MEASure query, but without initiating the measurement and fetching the result. Use configure when you want to change any parameters before making the measurement.

Read more about Configure under MEASure.

Format Subsystem

Time Stamp Readout Format

It is not trivial to decide how time stamped measurements are to be presented on the bus. If the 'ADIF' format defined by SCPI is adopted, it should be adopted for all data readout, and switched on and off by the already standardized :FORMat:DINTerchange command. This format covers the appropriate readout format for time stamped measurements well, so when it is selected as output format, there is not any problem. But the user may still decide not to use the ADIF format, so we need a solution to the readout problem whether or not we decide to implement ADIF. The chosen one is as follows:

For :FETCh:SCALar?, :READ:SCALar? and :MEASure:SCALar?, the readout will consist of two values instead of one. The first will be the measured value, and the next one will be the timestamp value, given in seconds in the NR2 format ddd.ddddddddd (12 digits).

In :FORMat ASCii mode, the result will be given as a floating-point number (NR3 format) followed by integers (NR1 format). In :FORMat REAL mode, the result will be given as an eight-byte block containing the floating-point measured value, followed by a four-byte block containing the integer timestamp count, where each count represents 125 nanoseconds.

When doing readouts in array form, with :FETCh :ARRay?, :READ :ARRay? or :MEASure :ARRay?, the response will consist of alternating measurement values and timestamp values, formatted the same way as for scalar readout. All values will be separated by commas.

Input Subsystems

PM6685

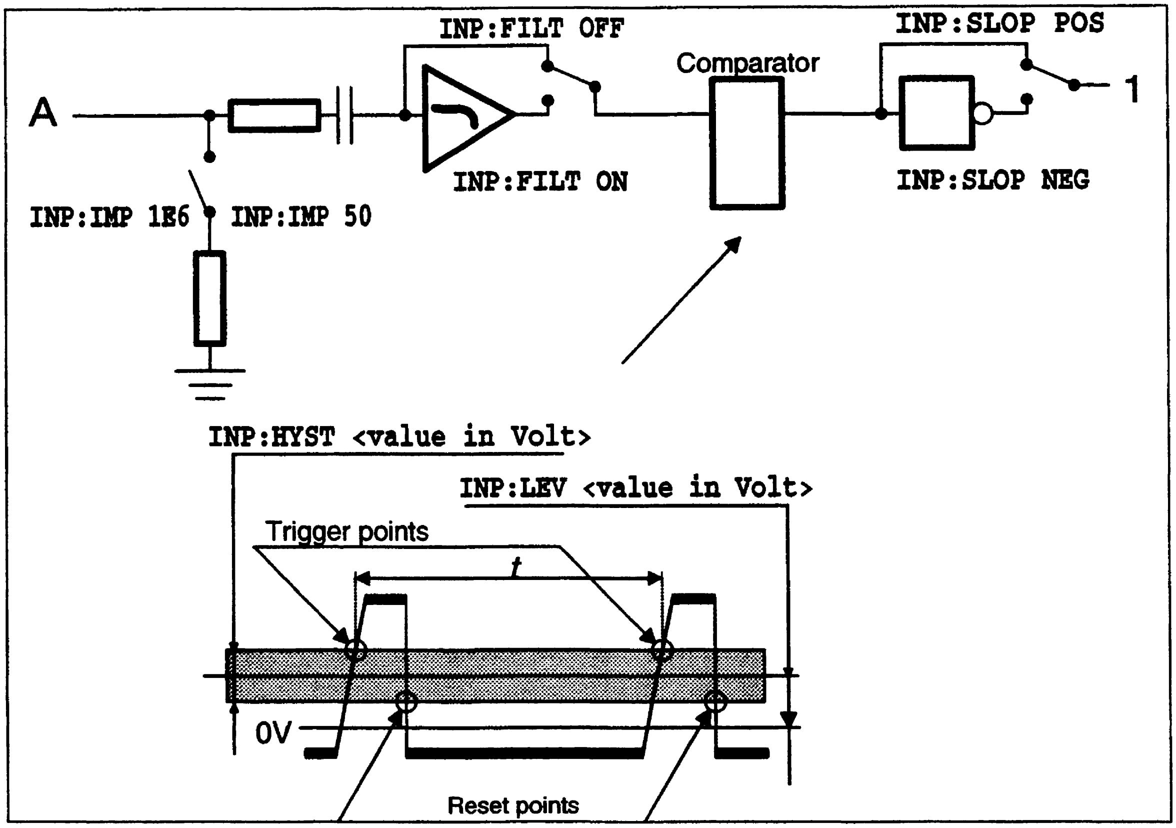


Figure 6-1 *Summary of PM6685 input amplifier settings.*

PM6680B/PM6681

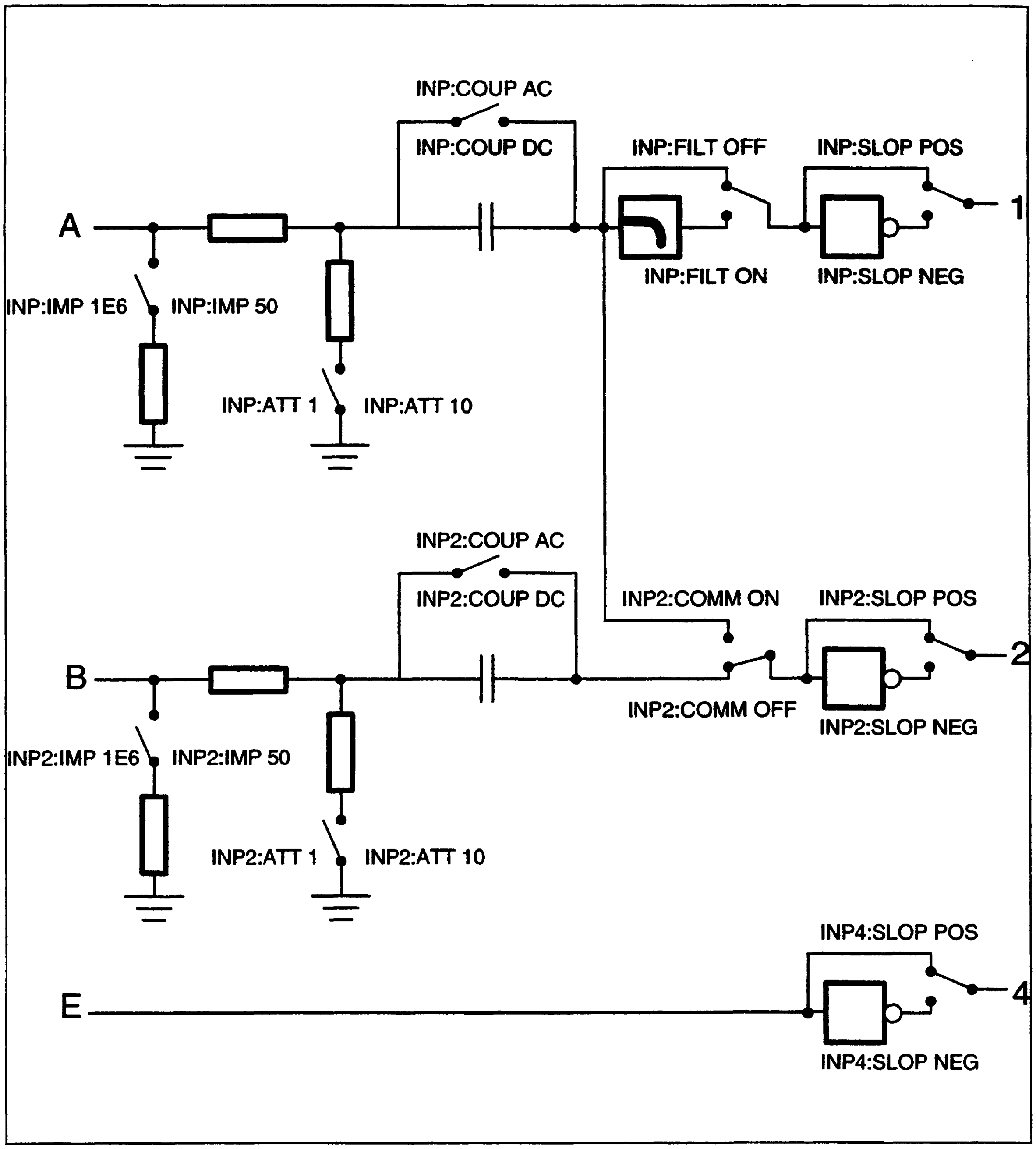


Figure 6-2 *Summary of PM6680B / PM6681 input amplifier settings.*

Measurement Function

The Measure function group has a different level of compatibility and flexibility than other commands. The parameters used with commands from the Measure group describe the signal you are going to measure. This means that the Measure functions give compatibility between instruments, since you don't need to know anything about the instrument you are using.

MEASure?

This is the most simple query to use, but it does not offer much flexibility. The MEASure? query lets the instrument configure itself for an optimal measurement, starts the data acquisition, and returns the result.

■ Example:

SEND→ `MEASure:FREQ?`

This will execute a frequency measurement and the result will be sent to the controller. The instrument will select a setting for this purpose by itself, and will carry out the required measurement as "well" as possible; moreover, it will automatically start the measurement and send the result to the controller.

You may add parameters to give more details about the signal you are going to measure, for example:

SEND→ `MEASure:FREQ?␣20␣MHz,1`

Where: 20 MHz is the expected value, which can, of course, also be sent as 20E6, and 1 is the required resolution. (1 Hz)

Also the channel numbers can be specified, for example:

```
SEND→ MEASure:FREQ?␣(@3)
SEND→ MEASure:FREQ?␣20E6,
        1,(@1)
```

CONFigure; READ?

The CONFigure command causes the instrument to choose an optimal setting for the specified measurement. CONFigure may cause any device setting to change. READ? starts the acquisition and returns the result.

This sequence operates in the same way as the MEASure command, but now it is possible to insert commands between CONFigure and READ? to fine tune the setting of a particular function. For example, you can change the input impedance from 1 MΩ to 50 Ω.

- **Example:**

SEND→ `CONFigure:FREQ⌴2E6,1`

2E6 is the expected value
1 is the required resolution (1Hz)

SEND→ `INPut:IMPedance50⌴OHM`

Sets input impedance to 50 Ω

SEND→ `READ?`

Starts the measurement and returns the result.

CONFigure;INITiate;FETCh?

The READ? command can be divided into the INITiate command, which starts the measurement, and the FETCh? command, which requests the instrument to return the measuring results to the controller.

- **Example:**

SEND→ `CONFigure:FREQ⌴20E6,1`

20E6 is the expected signal value
1 is the required resolution

SEND→ `INPut:IMPedance1E⌴6`

Sets input impedance to 1 MΩ

SEND→ `INITiate`

Starts measurement

SEND→ `FETCh?`

Fetches the result

Versatility of measurement commands	
MEASure?	Simple to use, few additional possibilities.
CONFigure READ?	Somewhat more difficult, but some extra possibilities.
CONFigure INITiate FETCh?	Most difficult to use, but many extra features.

Output Subsystem

The analog output is turned off as a default. You turn it on/off and set the scaling factor under ANALOG OUT in the aux menu.

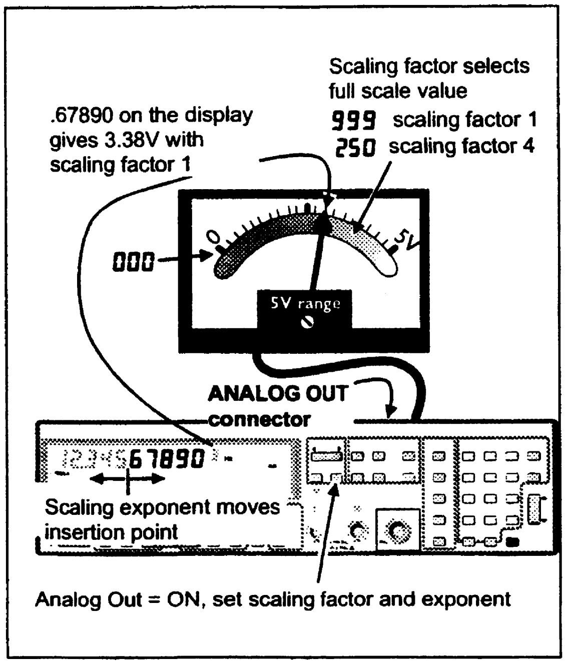

Scaling factor selects full scale value
999 scaling factor 1
250 scaling factor 4
.67890 on the display gives 3.38V with scaling factor 1
000
0
5V
5V range
ANALOG OUT connector
67890
Scaling exponent moves insertion point
Analog Out = ON, set scaling factor and exponent

Figure 6-3 *The analog output function.*

Scaling Factor

The scaling factor has two functions:

- Its exponent selects which digits to output on the analog output.
- Its value sets what reading should represent full scale.

As default, the scaling factor is 1 (1E0). This means that the full scale value is 0.999 and the analog output converts the fraction (digits to the right of the decimal point) to a voltage.

The scaling factor should be:

$$Scaling\ factor = \frac{1}{full\ scale\ value}$$

where full scale value is the value for which you want the analog output to output its maximum voltage (5 V).

Example:

- Take a measurement result, for instance: 12.34567890 E+6 Hz
- Represent this result without exponent: 12345678.90 Hz
- Multiply this value with the scaling factor, for instance 0.001. 12345.67890
- Take the fractional part of the result: .67890

– This is the value that will determine the output voltage; .00 will give 0 V and .99 will give 5 V. This means that the reading will give:
.67890*5=3.3945 V.
This is ouput as 3.38 V due to the 0.02 V resolution of the analog output.

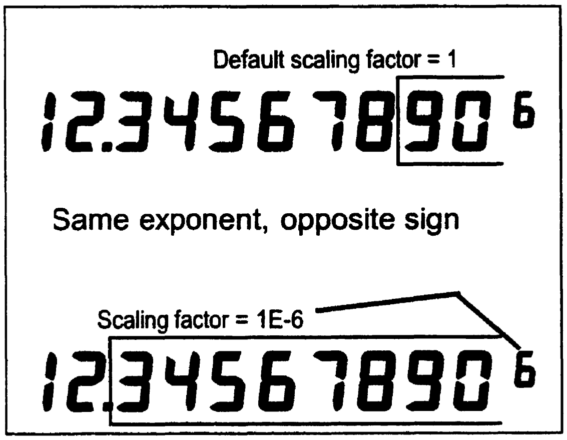


Figure 6-4 *To use the shown decimal point as reference, set the exponent of the scaling factor to the same value as the exponent of the measurement result but with opposite sign.*

■ Resolution

The analog output range is 0 to 5 V in 250 steps, so one step is 0.02 V. If the scaling factor is 1, one such step is taken each time the display changes with X.004, and if the scaling factor is 4, one step is taken each time the display changes with X.001.

The X in the above paragraph can be any digit and does not influence the output voltage. If the display changes from 0.996 to 1.000, the voltage drops from 4.98 V to 0V. If the display value increases further, the output voltage starts to increase again; see .

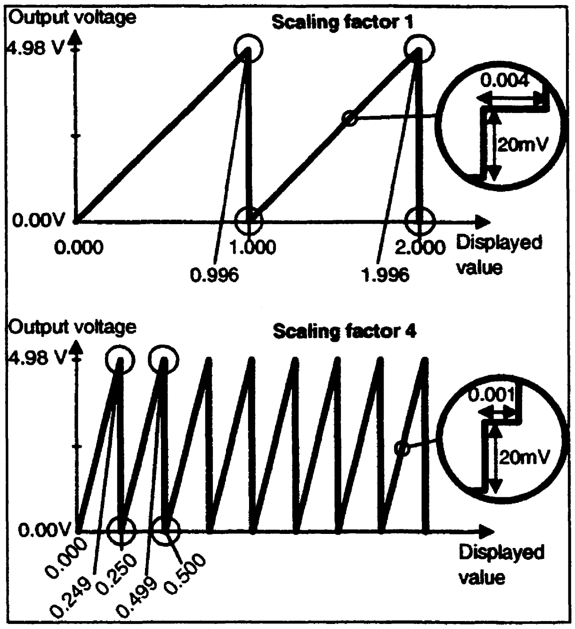


Figure 6-5 *Output voltage versus displayed value for two different scaling factors.*

Sense Command Subsystems

Depending on application, you can select different input channels and input characteristics.

■ Switchbox

In automatic test systems, it is difficult to swap BNC cables when you need to measure on several measuring points. With PM6680B/1 you can select from three different basic inputs (A, B and E), on which the counter can measure directly without the need for external switching devices. With PM6685 you can select from two different basic inputs (A and E).

■ Prescaling

For all measuring functions except frequency, the maximum input A frequency is 160 MHz.

To extend the range for frequency measurements, PM6680B and PM6685 can divide (scale) the input A frequency by two, while PM6681 scales by four.

When using channel 1, the counter automatically selects this scaling factor when measuring FREQ A, giving 300 MHz max frequency for PM6681 and PM6685, and 225 MHz for PM6680B.

For all other measuring functions, and for frequency if you select negative slope, the counter does not divide the signal and the max repetition rate is 160 MHz.

Status Subsystem

Introduction

Status reporting is a method to let the controller know what the counter is doing. You can ask the counter what status it is in whenever you want to know.

You can select some conditions in the counter that should be reported in the Status Byte Register. You can also select if some bits in the Status Byte should generate a Service Request (SRQ).
(An SRQ is the instrument's way to call the controller for help.)

Status Reporting Model

■ The Status Structure

The status reporting model used by CNT-8X is standardized in IEEE 488.2 and SCPI, so you will find similar status reporting in most modern instruments. Figure 6-6 shows an overview of the complete CNT-8X status register structure. It has four registers, two queues, and a status byte:

- The **Standard Event Register** reports the standardized IEEE 488.2 errors and conditions.
- The **Operation Status Register** reports the status of the CNT-8X measurement cycle (see also ARM-TRIG model, page 6-27).
- The **Questionable Data Register** reports when the output data from the CNT-8X may not be trusted.
- The **Device Register 0** reports when the measuring result has exceeded preprogrammed limits.
- The **Output Queue status** reports if there is output data to be fetched.
- The **Error Queue status** reports if there are error messages available in the error queue.
- The **Status Byte** contains eight bits. Each bit shows if there is information to be fetched in the above described registers and queues of the status structure.

Using the Registers

Each status register monitors several conditions at once. If something happens to any one of the monitored conditions, a summary bit is set true in the Status Byte Register.

Enable registers are available so that you can select what conditions should be reported in the status byte, and what bits in the status byte should cause SRQ.

A register bit is TRUE, i.e., something has happened, when it is set to 1. It is FALSE when set to 0.

Note that all event registers and the status byte records positive events. That is when a condition changes from inactive to active, the bit in the event register is set true. When the condition changes from active to inactive, the event register bits are not affected at all.

When you read the contents of a register, the counter answers with the decimal sum of the bits in the register.

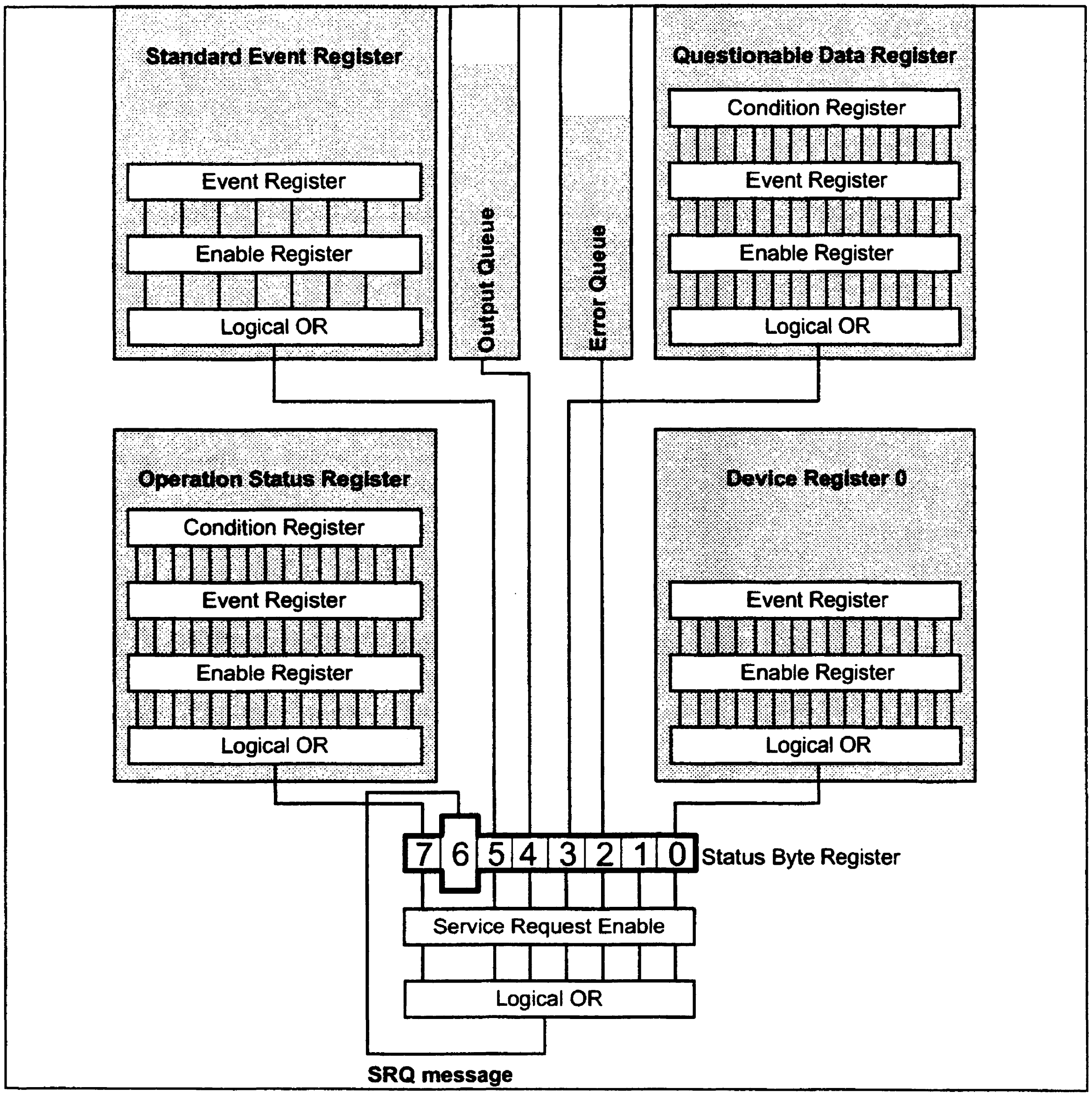


Figure 6-6 *CNT-8X Status register structure.*

Example:

The counter answers 40 when you ask for the contents of the Standard Event Status Register.

- Convert this to binary form. It will give you 101000.
- Bit 5 is true showing that a command error has occurred.
- Bit 3 is also true, showing that a device dependent error has occurred.

Use the same technique when you program the enable registers.

- Select which bits should be true.
- Convert the binary expression to decimal data.
- Send the decimal data to the instrument.

Clearing/Setting all bits

- You can clear an enable register by programming it to zero. You can set all bits true in a 16-bit event enable register by programming it to 32767 (bit 16 not used).
- You set all bits true in 8-bit registers by programming them to 255 (Service Request Enable and Standard Event Enable.)

■ Using the Queues

The two queues, where CNT-8X stores output data and error messages, may contain data or be empty. Both these queues have their own status bit in the Status Byte. If this bit is true there is data to be fetched.

When the controller reads data, it will also remove the data from the queue. The queue status bit in the status byte will remain true for as long as the queue holds one or more data bytes. When the queue is empty, the queue status bit is set false.

Status of the Output Queue (MAV)

The MAV (message available) queue status message appears in bit 4 of the status byte register. It indicates if there are bytes ready to be read over the GPIB in the GPIB output queue of the instrument. The output queue is where the formatted data appears before it is transferred to the controller.

The controller reads this queue by addressing the instrument as a talker. The command to do this differs between different programming languages. Examples are IOENTERS and IBREAD.

Status of the Error Message Queue (EAV)

The EAV (error message available) queue status message appears in bit 2 of the status byte register. Use the `:SYSTem:ERRor?` query to read the error messages. Chapter 21 explains all possible error messages .

■ Using the Status Byte

The status byte is an eight bit status message. It is sent to the controller as a response to a serial poll or a *STB? query, see Figure 6-7. Each bit in the status byte contains a summary message from the status structure. You can select what bits in the status byte should generate a service request to alert the controller.

When a service request occurs, the SRQ-line of the GPIB will be activated. Whether or not the controller will react on the service request depends on the controller program. The controller may be interrupted on occurrence of a service request, it may regularly test the SRQ-line, it may regularly make serial poll or *STB?, or the controller may not react at all. The preferred method is to use SRQ because it presents a minimum of disturbance to the measurement process.

Selecting Summary Message to Generate SRQ

The counter does not generate any SRQ by default. You must first select which summary message(s) from the status byte register should give SRQ. You do that with the Service Request Enable command `*SRE <bit mask>`.

Example:

```
*SRE˽16
```

This sets bit 4 (16=2^4) in the service request enable register (see Figure 6-8). This makes the instrument signal SRQ when a message is available in the output queue.

RQS/MSS

The original status byte of IEEE 488.1 is sent as a response to a serial poll, and bit 6 means requested service, RQS.

IEEE 488.2 added the `*STB?` query and expanded the status byte with a slightly different bit 6, the MSS. This bit is true

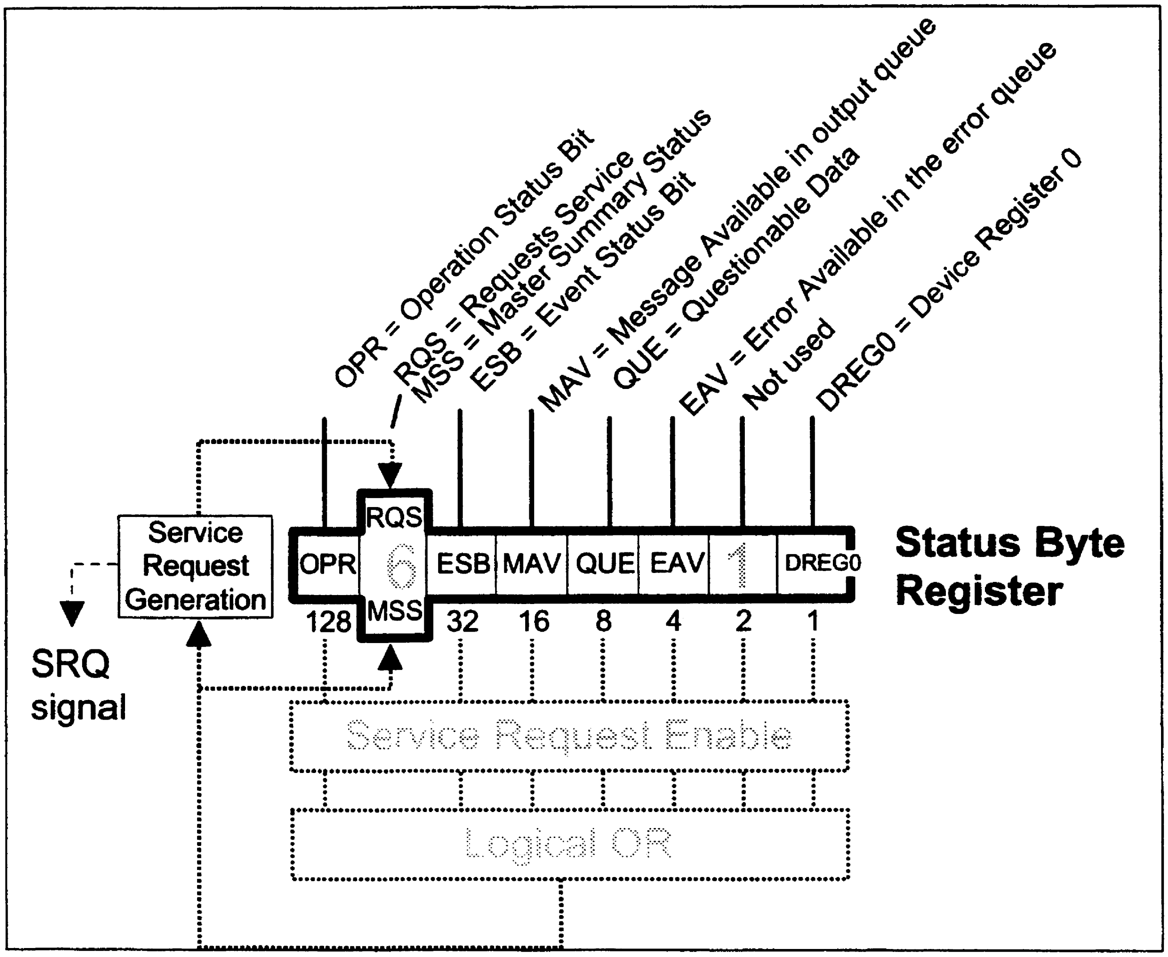

OPR = Operation Status Bit
RQS = Requests Service
MSS = Master Summary Status Bit
ESB = Event Status Bit
MAV = Message Available in output queue
QUE = Questionable Data
EAV = Error Available in the error queue
Not used
DREG0 = Device Register 0
Service Request Generation
SRQ signal
RQS
OPR
6
MSS
ESB
MAV
QUE
EAV
1
DREG0
128
32
16
8
4
2
1
Status Byte Register
Service Request Enable
Logical OR

Figure 6-7 *The status byte bits.*

as long as there is unfetched data in any of the status event registers.

- The Requested Service bit, RQS, is set true when a service request has been signalled. If you read the status byte via a Serial Poll, bit 6 represents RQS. Reading the status byte with a serial poll will set the RQS bit false, showing that the status byte has been read.
- The Master Summary Status bit, MSS, is set true if any of the bits that generates SRQ is true. If you read the status byte using `*STB?`, bit 6 represents MSS. MSS remains true until all event registers are cleared and all queues are empty.

Setting up the Counter to Report Status

Include the following steps in your program when you want to use the status reporting in CNT-8X:

- `*CLS` Clears all event registers and the error queue
- `*ESE <bit mask>` Selects what conditions in the Standard Event Status register should be reported in bit 5 of the status byte
- `:STATus:OPERation:ENABle <bit mask>` Selects which conditions in the Operation Status register should be reported in bit 7 of the status byte
- `:STATus:QUEStionable:ENABle <bit mask>` Selects which conditions in the Questionable Status register should be reported in bit 3 of the status byte
- `:STATus:DREGister0:ENABle <bit mask>` Selects which conditions in Device Register 0 should be reported in bit 0 of the status byte
- `*SRE <bit mask>` Selects which bits in the status byte should cause a Service Request

A programming example using status reporting is available in the Programming Examples in chapter 4.

Reading and Clearing Status

■ Status Byte

As explained earlier, you can read the status byte register in two ways:

Using the Serial Poll (IEEE-488.1 defined).

- Response:
  - Bit 6: RQS message, shows that the counter has requested service via the SRQ signal.
  - Other bits show their summary messages
  - A serial poll sets the RQS bit FALSE, but does not change other bits.

*Using the Common Query *STB?*

- Response:
  - Bit 6: MSS message, shows that there is a reason for service request.
  - Other bits show their summary messages.
  - Reading the response will not alter the status byte.

■ Status Event Registers

You read the Status Event registers with the following queries:

- `*ESR?` Reads the Standard Event Status register
- `:STATus:OPERation?` Reads the Operation Status Event register
- `:STATus:QUEStionable?` Reads the Questionable Status Event register
- `:STATus:DREGister0?` Reads Device Event register

When you read these registers, you will clear the register you read and the summary message bit in the status byte.

You can also clear all event registers with the `*CLS` (Clear Status) command.

■ Status Condition Registers

Two of the status register structures also have condition registers: The Status Operation and the Status Questionable register.
The condition registers differ from the event registers in that they are not latched. That is, if a condition in the counter goes on and then off, the condition register indicates true while the condition is on and false when the condition goes off. The Event register that monitors the same condition continues to indicate true until you read the register.

- `:STATus:OPERation:CONDition?` Reads the Operation Status Condition register
- `:STATus:QUEStionable:CONDition?` Reads the Questionable Status Condition register

Reading the condition register will not affect the contents of the register.

Why Two Types of Registers?

Let's say that the counter measures continuously and you want to monitor the measurement cycle by reading the Operation Status register.

Reading the Event Register will always show that a measurement has started, that waiting for triggering and bus arming has occurred and that the measurement is stopped. This information is not very useful.

Reading the Condition Register on the other hand gives *only* the status of the measurement cycle, for instance "Measurement stopped".

Although it is possible to read the condition registers directly, we recommend that you use SRQ when monitoring the measurement cycle. The measurement cycle is disturbed when you read condition registers.

■ Summary:

The way to work when writing your bus program is as follows:

Set up

- Set up the enable registers so that the events you are interested in are summarized in the status byte.
- Set up the enable masks so that the conditions you want to be alerted about generate SRQ. It is good practice to generate SRQ on the EAV bit. So, enable the EAV-bit via `*SRE`.

Check & Action

- Check if an SRQ has been received.
  - Make a serial poll of the instruments on the bus until you find the instrument that issued the SRQ (the instrument that has RQS bit true in the Status Byte).

- When you find it, check which bits in the Status Byte Register are true.
- Let's say that bit 7, OPR, is true. Then read the contents of the Operation Status Register. In this register you can see what caused the SRQ.
- Take appropriate actions depending on the reason for the SRQ.

Standard Status Registers

These registers are called the *standard* status data structure because they are mandatory in all instruments that fulfill the IEEE 488.2 standard.

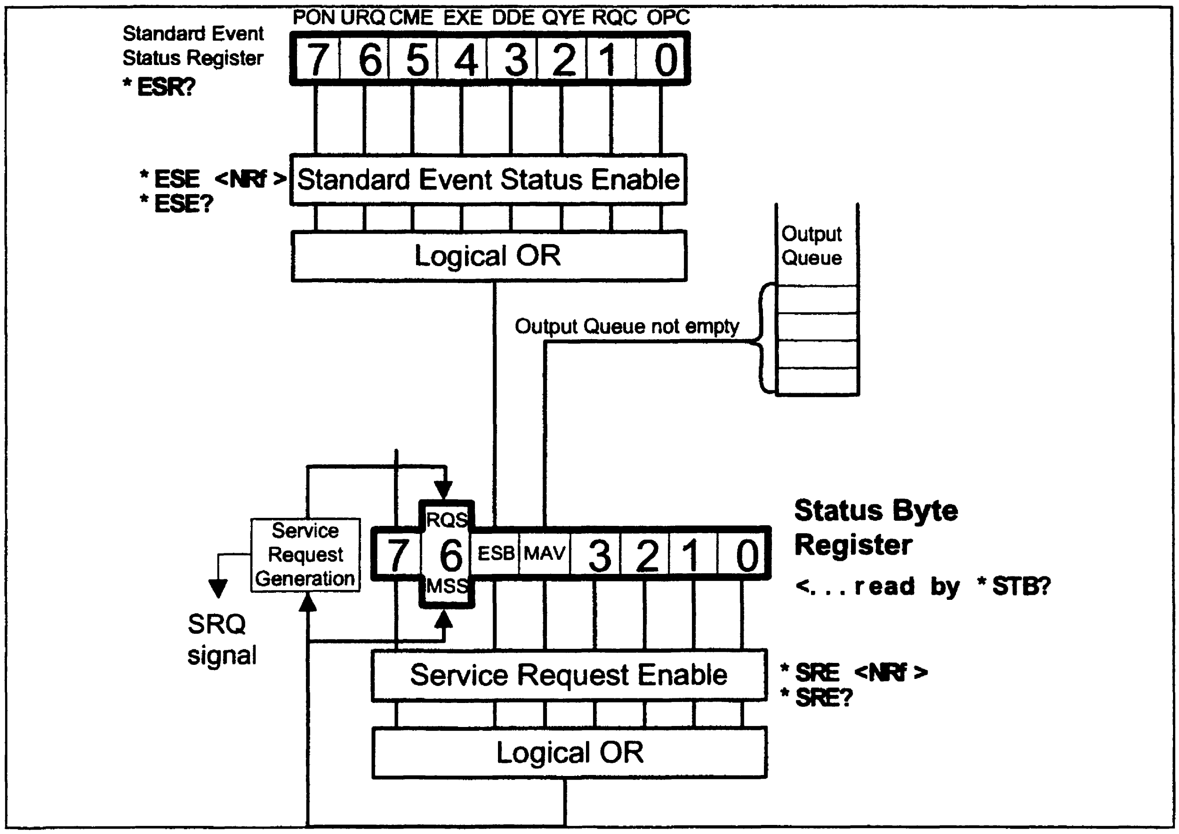


Figure 6-8 *Standard status data structures, overview.*

Standard Event Status Register

Bit 7 (weight 128) — Power-on (PON)

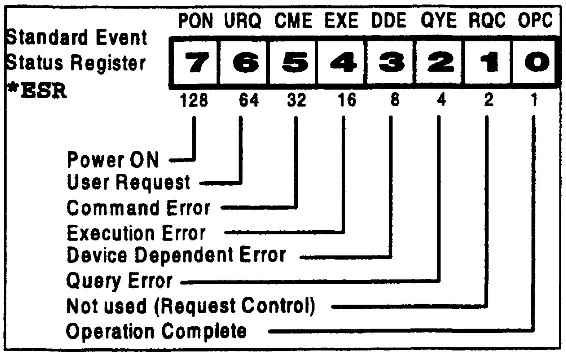


Figure 6-9 *Bits in the standard event status register*

Shows that the counter's power supply has been turned off and on (since the last time the controller read or cleared this register).

Bit 6 (weight 64)—User Request (URQ)

Shows that the user has pressed a key on the front panel of CNT-8X (except LOCAL/PRESET). The URQ bit will be set regardless of the remote local state of the counter. The purpose of this signal is, for example, to call for the attention of the controller by generating a service request.

Bit 5 (weight 32) — Command Error (CME)

Shows that the instrument has detected a command error. This means that it has received data that violates the syntax rules for program messages.

Bit 4 (weight 16) — Execution Error (EXE)

Shows that the counter detected an error while trying to execute a command. (See 'Error reporting' on page 3-17.) The command is syntactically correct, but the counter cannot execute it, for example because a parameter is out of range.

Bit 3 (weight 8) — Device-dependent Error (DDE)

A device-dependent error is any device operation that did not execute properly because of some internal condition, for instance error queue overflow. This bit shows that the error was not a command, query or execution error.

Bit 2 (weight 4) — Query Error (QYE)

The output queue control detects query errors. For example the QYE bit shows the unterminated, interrupted, and deadlock conditions. For more details, see 'Error reporting' on page 3-17.

Bit 1 (weight 2)—Request Control (RQC)

Shows the controller that the device wants to become the active controller-in-charge. Not used in the CNT-8X.

Bit 0 (weight 1) — Operation Complete (OPC)

The counter only sets this bit TRUE in response to the operation complete command (*OPC). It shows that the counter has completed all previously started actions.

Summary, Standard Event Status Reporting

*ESE <bit mask>

Enable reporting of Standard Event Status in the status byte.

*SRE 32

Enable SRQ when the Standard Event structure has something to report.

ESR?

Reading and clearing the event register of the Standard Event structure.

SCPI-defined Status Registers

CNT-8X has two 16-bit SCPI-defined status structures: The operation status and the questionable data structure. These group is 16-bits wide while the status byte and the standard status groups are 8-bits wide.

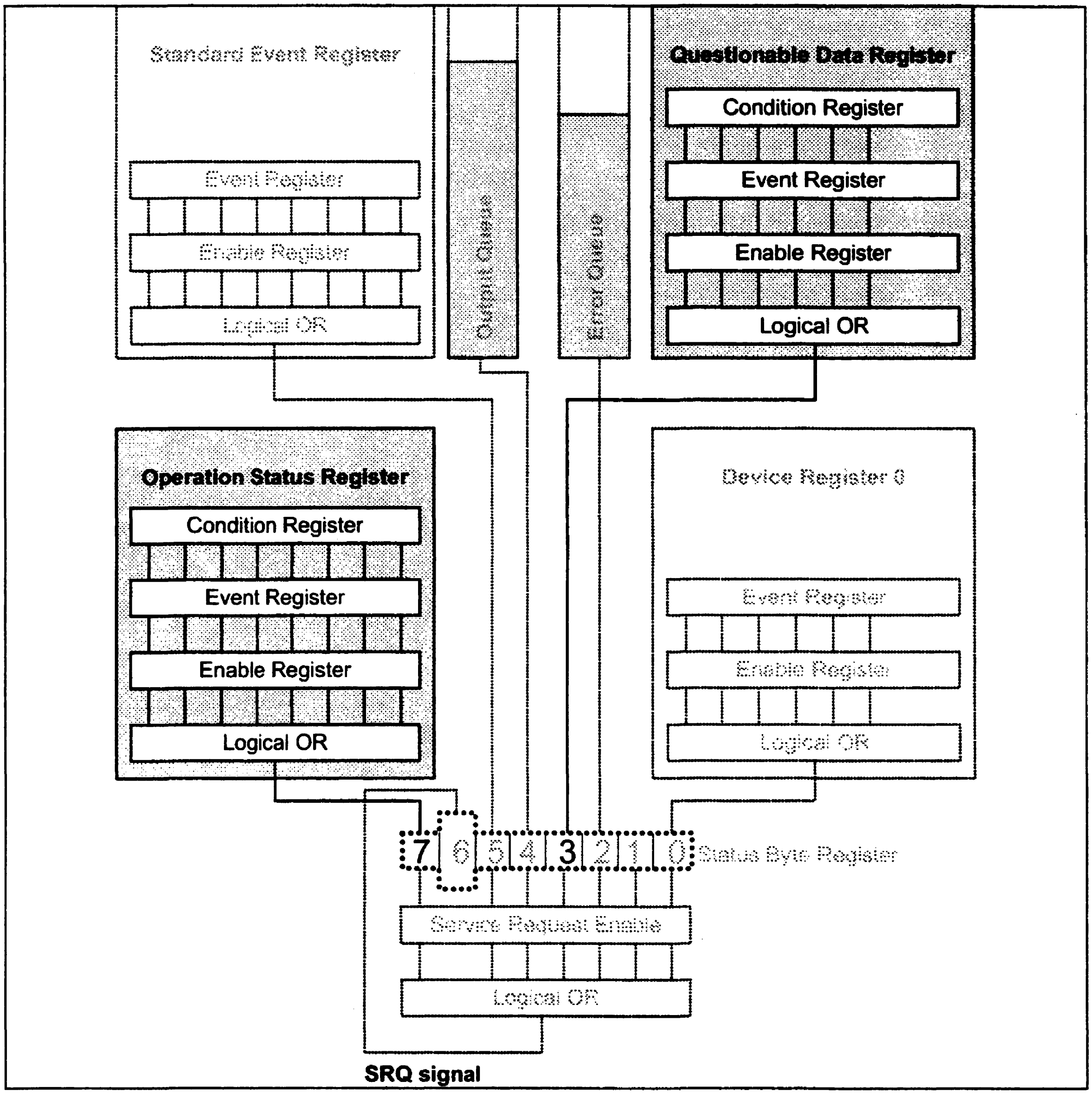


Figure 6-10 *Status structure 7, Operation Status Group, and Status structure 3, Questionable Data Group are SCPI defined.*

■ Operation Status Group

This group reports the status of the CNT-8X measurement cycle.

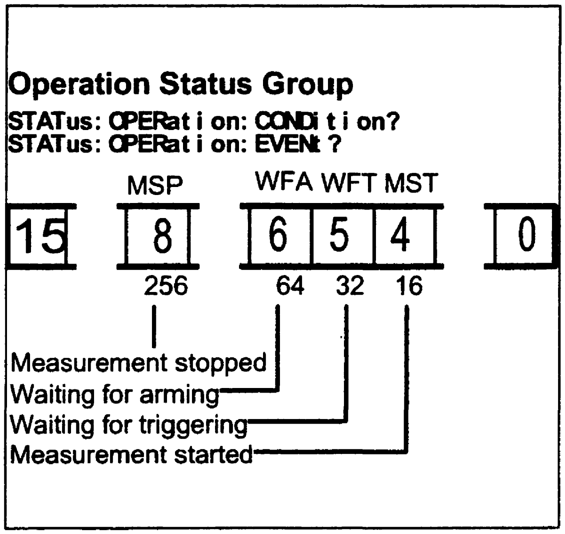


Figure 6-11 *Bits in the Opeation Status Register.*

Bit 8 (weight 256) — Measurement Stopped (MSP)

This bit shows that the counter is not measuring. It is set when the measurement, or sequence of measurements, stops.

Bit 6 (weight 64) — Wait for Bus Arming (WFA)

This bit shows that the counter is waiting for bus arming in the arm state of the trigger model.

Bit 5 (weight 32) — Waiting for Trigger and/or External Arming (WFT)

This bit shows when the counter is ready to start a new measurement via the trigger control option (488.2), for the shortest possible trigger delay. The counter is now in the wait for the trigger state of the trigger model.

Bit 4 (weight 14) — Measurement Started (MST)

This bit shows that the counter is measuring. It is set when the measurement or sequence of measurements start.

■ Summary, Operation Status Reporting

:STAT:OPER:ENAB

Enable reporting of Operation Status in the status byte.

**SRE 128*

Enable SRQ when operation status has someting to report.

:STAT:OPER?

Reading and clearing the event register of the Operation Status Register structure

:STAT:OPER:COND?

Reading the condition register of the Operation Status Register structure.

Questionable Data/Signal Status Group

This group reports when the output data from the CNT-8X may not be trusted.

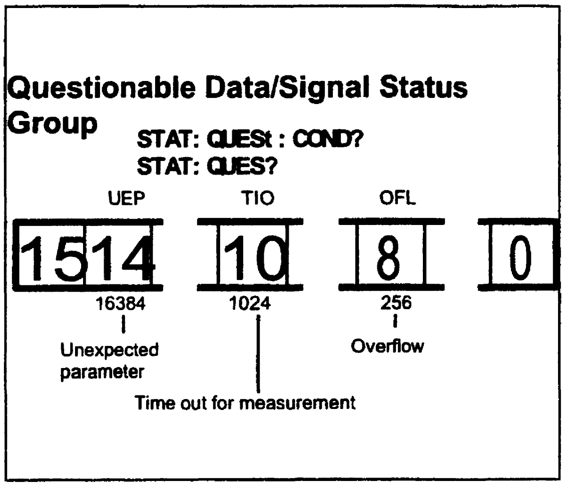


Figure 6-12 *Bits in Questionable data register.*

Bit 14 (weight 16384) — Unexpected Parameter (UEP)

This bit shows that CNT-8X has received a parameter that it cannot execute, although the parameter is valid according to SCPI. This means that when this bit is true, the instrument has not performed a measurement exactly as requested.

Bit 10 (weight 1024) — Timeout for Measurement (TIO)

The counter sets this bit true when it abandons the measurement because the internal timeout has elapsed, or no signal has been detected.
See also `:SYST:TOUT` and `:SYST:SDET`.

Bit 8 (weight 256) Overflow (OFL)

The counter sets this bit true when it cannot complete the measurement due to overflow.

■ Summary, Questionable Data/Signal Status Reporting

:STAT:QUES:ENAB <bit mask>

Enable reporting of Questionable data/signal status in the status byte.

**SRE 8*

Enable SRQ when data/signal is questionable.

:STAT:QUES?

Reading and clearing the event register of the Questionable data/signal Register structure.

:STAT:QUES:COND?

Reading the condition register of Questionable data/signal Register structure.

Device-defined Status Structure

CNT-8X has one device-defined status structure called the Device Register 0. It summarizes this structure in bit 0 of the status byte. Its purpose is to report when the measuring result has exceeded preprogrammed limits.

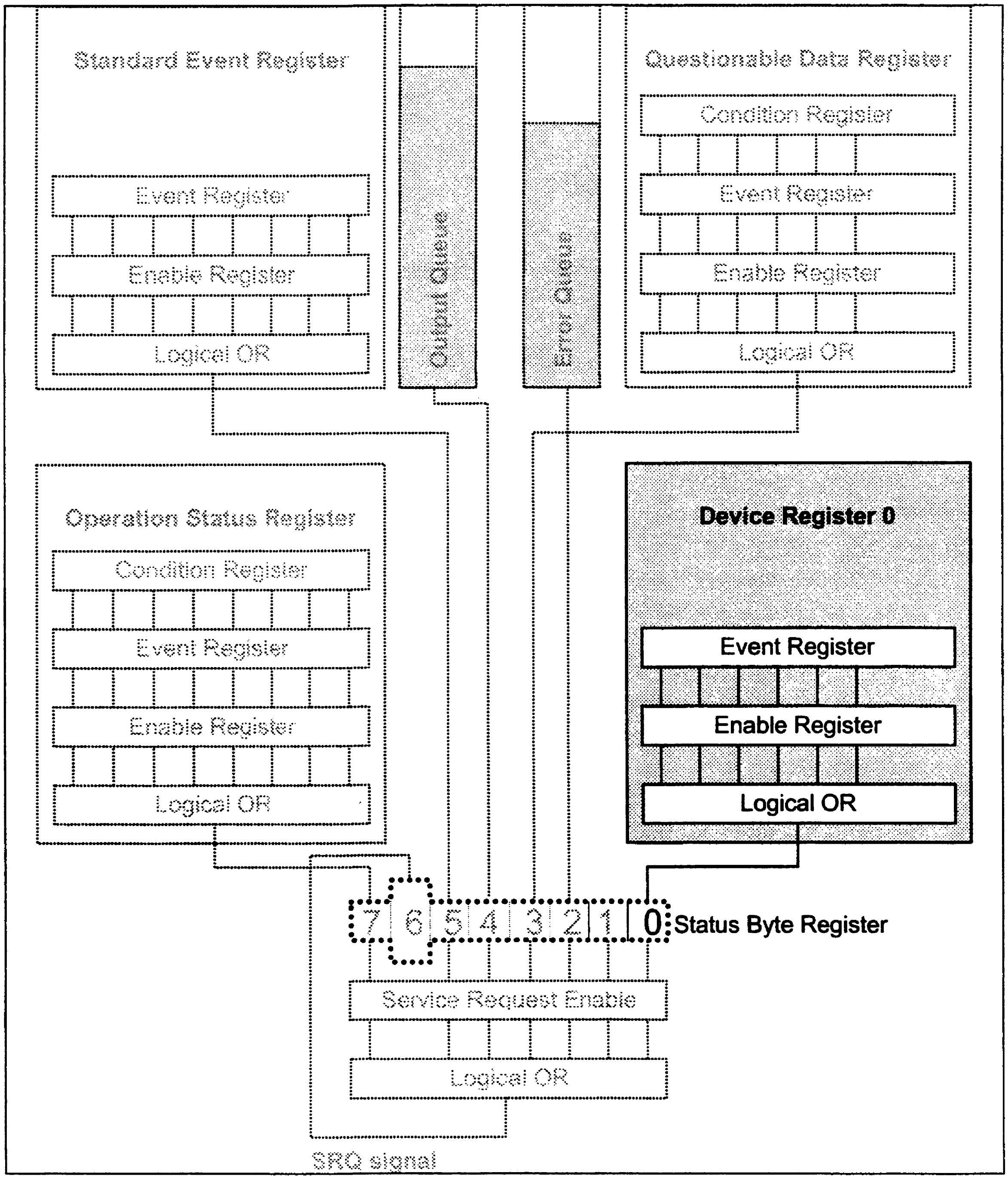


Figure 6-13 Device-defined status data structures (model).

You set the limits with the following commands in the calculate subsystem. `:CALCulate:LIMit:UPPer` and `:CALCulate:LIMit:LOWer`

An example on how to use limit monitoring is available in Chapter 4, 'Program Examples.'

Bit Definition

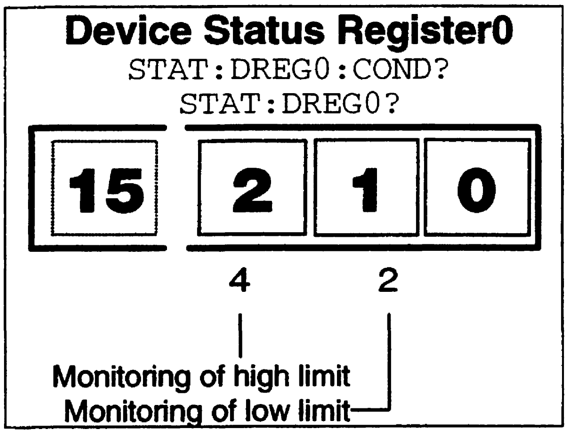


Figure 6-14 *Bits in the Device Status Register number 0.* `:STATus:DREGister0?` *Reads out the contents of the Device Status event Register 0 and clears the register.*

Bit 2 (weight 4) — Monitor of Low Limit

This bit is set when the low limit is passed from above.

Bit 1 (weight 2) — Monitor of High Limit

This bit is set when the high limit is passed from below.

■ Summary, Device-defined Status Reporting

:STAT:DREG0:ENAB <bit mask>

Enable reporting of device-defined status in the status byte.

**SRE 1*

Enable SRQ when a limit is exceeded.

:STAT:DREG0?

Reading and clearing the event register of Device Register structure 0.

– If bit 1 is true, the high limit has been exceeded.

– If bit 2 is true, the low limit has been exceeded.

Power-on Status Clear

Power-on clears all event enable registers and the service request enable register if the power-on status clear flag is set TRUE (see the common command `*PSC`.)

■ Preset the Status Reporting Structure

You can preset the complete status structure to a known state with a single command, the `STATus:PRESet` command, which does the following:

– Disables all bits in the Standard Event Register, the Operation Status Register, and the Questionable Data Register

– Enables all bits in Device Register 0

– Leaves the Service Request Enable Register unaffected.

Trigger/Arming Subsystem

The SCPI TRIGger subsystem enables synchronization of instrument actions with specified internal or external events. The following list gives some examples.

Instrument Action

Some examples of events to synchronize with are as follows:

- measurement
- bus trigger
- external signal level or pulse
- 10 occurrences of a pulse on the external trigger input
- other instrument ready
- signal switching
- input signal present
- 1 second after input signal is present
- sourcing output signal
- switching system ready

The ARM-TRIG Trigger Configuration

Figure 6-15 gives a typical trigger configuration, the ARM-TRIG model. The configuration contains two event-detection layers: the 'Wait for ARM' and 'Wait for TRIG' states.

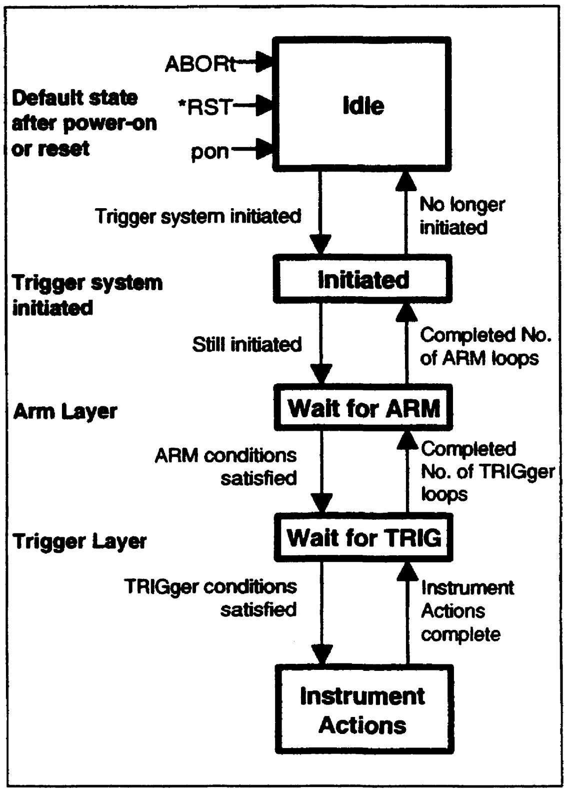


Figure 6-15 *Generalized ARM-TRIG model.*

This trigger configuration is sufficient for most instruments. More complex instruments, such as the CNT-8X, have more ARM layers.

The 'Wait for TRIG' event-detection layer is always the last to be crossed before instrument actions can take place.

Structure of the IDLE and INITIATED States

When you turn on the power or send `*RST` or `:ABORT` to the instrument, it sets the trigger system in the IDLE state; see .

The trigger system will exit from the IDLE state when the instrument receives an `INITiate:IMMediate`. The instrument will pass directly through the INITIATED state downward to the next event-detection layers (if the instrument contains any more layers).

The trigger system will return to the INITIATED state when all events required by the detection layers have occurred and the instrument has made the intended measurement. When you program the trigger system to `INITiate:CONTinuous` ON, the instrument will directly exit the INITIATED state moving downward and will repeat the whole flow described above. When `INITiate:CONTinuous is OFF`, the trigger system will return to the IDLE state.

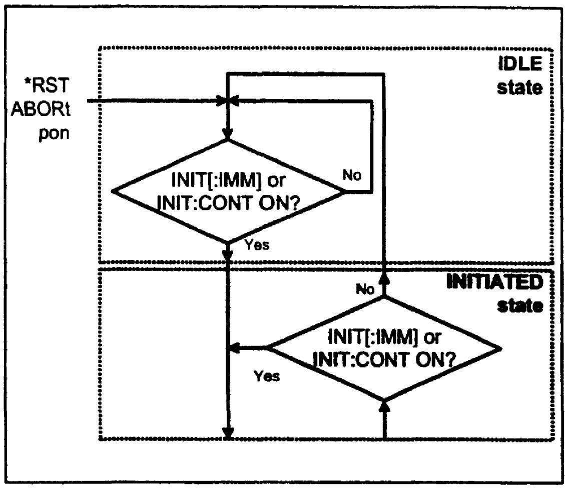


Figure 6-16 *Flow diagram of IDLE and INITIADED layers.*

■ Structure of an Event-detection Layer

The general structure of all event-detection layers is identical and is roughly depicted by the flow diagram in

In each layer there are several programmable conditions, which must be satisfied to pass by the layer in a downward direction:

■ Forward Traversing an Event-detection Layer

After initiating the loop counters, the instrument waits for the event to be detected. You can select the event to be detected by using the <layer>:SOURce command. For example: `:ARM:LAYer2:SOURce BUS`

You can specify a more precise characteristic of the event to occur. For example: `:ARM:LAYer:DELay 0.1`

You may program a certain delay between the occurrence of the event and entering into the next layer (or starting the device actions when in the TRIGger layer). This delay can be programmed by using the <layer>:DELay command.

■ Backward Traversing an Event-detection Layer

The number of times a layer event has to initiate a device action can be programmed by using the <layer>:COUNt command. For example: `:TRIGger:COUNt 3` causes the instrument to measure three times, each measurement being triggered by the specified events.

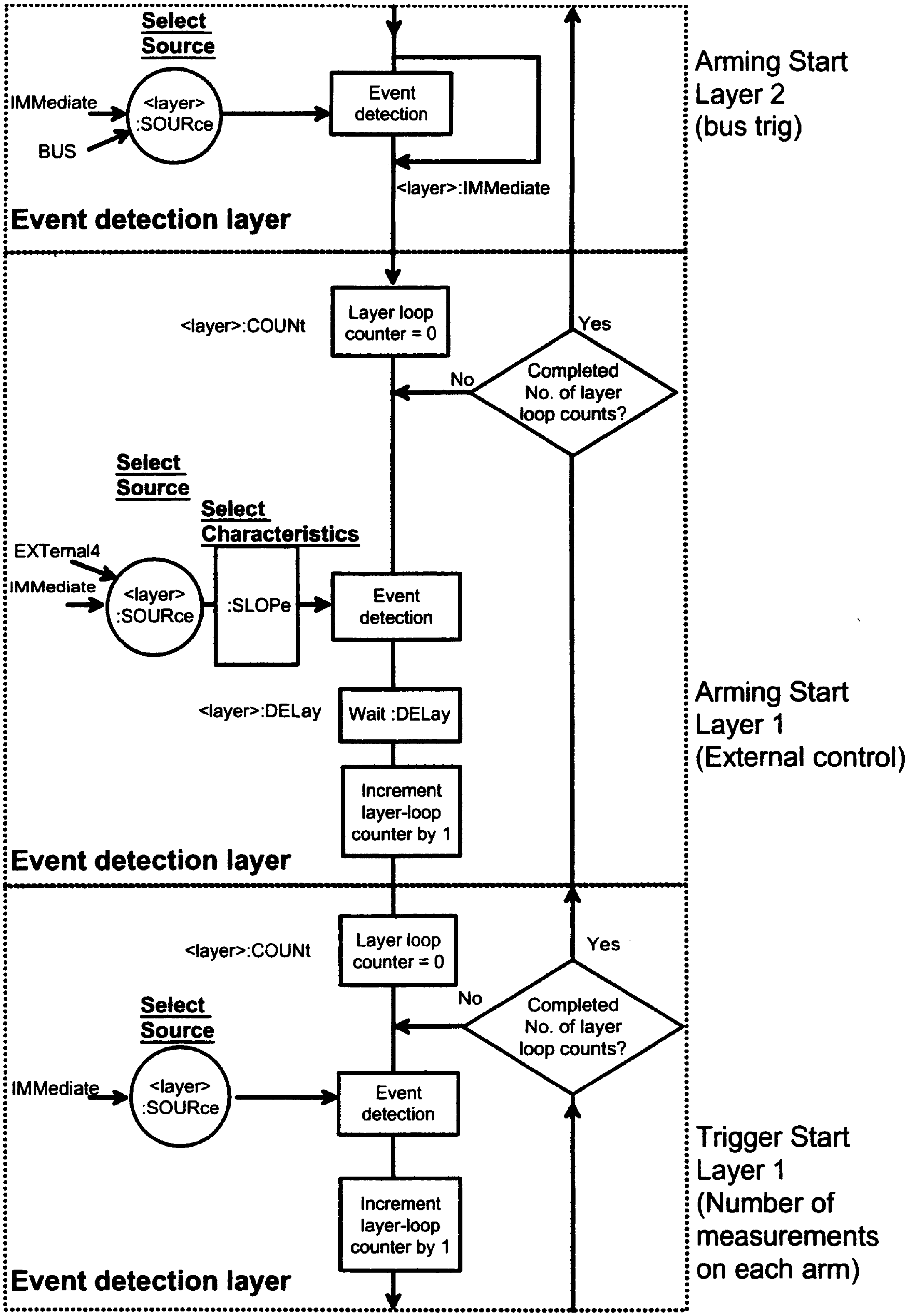
Triggering

■ *TRG Trigger Command

The trigger command has the same function as the Group Execute Trigger command GET, defined by IEEE 488.1.

*When to use *TRG and GET*

The `*TRG` and the GET commands have the same effect on the instrument. If the Counter is in idle, i.e., not parsing or executing any commands, GET will execute much (≈ 20 µs) faster than `*TRG` (≈ 4 ms) since the instrument must always parse `*TRG`.


Select Source
IMMediate
BUS
<layer> :SOURce
Event detection
<layer>:IMMediate
Event detection layer
Arming Start
Layer 2
(bus trig)
<layer>:COUNt
Layer loop counter = 0
Yes
No
Completed No. of layer loop counts?
Select Source
Select Characteristics
EXTernal4
IMMediate
<layer> :SOURce
:SLOPe
Event detection
<layer>:DELay
Wait :DELay
Increment layer-loop counter by 1
Event detection layer
Arming Start
Layer 1
(External control)
<layer>:COUNt
Layer loop counter = 0
Yes
No
Completed No. of layer loop counts?
Select Source
IMMediate
<layer> :SOURce
Event detection
Increment layer-loop counter by 1
Event detection layer
Trigger Start
Layer 1
(Number of
measurements
on each arm)

Chapter 7

How to Measure Fast

Introduction

The CNT-8X counters can complete a measurement cycle in many different ways, each with its own advantage. This means that your first step is to select a basic "measurement scenario" based on the requirements of the measurement. This chapter contains some measurement scenarios that you can choose from.

These counters can measure with impressive speed if you program them correctly. You will find guidelines for speed improvements in each of the described measurement scenarios.

Controller Synchronization

The start of measurements can either be individually or block synchronized by the controller. The instrument-to-controller synchronization deals with how to start a measurement or sequence of measurements and to read data in the most efficient way. You can also synchronize the measurement with the measuring object more accurately by using external control (arming), but this is not described here.

Measurement Cycle Synchronization

It is a good practice to check that the measurement proceeds as planned when the controller has started a measurement, or block of measurements.

■ Start

If the input signal fails, or there is no arming etc., the measurement cycle will not start.

Timeout

Turn on timeout and set the time longer than the expected measurement cycle. Then wait for the timeout period, and take actions if you got timeout.

If the measurement time is long, you may have to wait many seconds or even minutes until timeout, just to learn that the measurement never started.

Measurement started

Before starting a measurement, set up the status reporting system so that you get a Service Request on Measurement-in-progress, bit 4 in the Operation Status Event Register. Check this with serial poll after a reasonable time when the measurement ought to be started, lets say after 100ms (time dependent on input signal frequency). If the bit is true, continue. If false, abort the measurement and check the signals, alert the operator etc.

■ Stop

You must also know when the measurement is completed in order to read out the results. Should you read results or send other commands before the measurement is completed, the measurement will be interrupted.

You can of course let the controller wait until you are absolutely certain that the result is ready, before you fetch it. But it is better to use *OPC to get an Operation Complete status message, or *OPC? to get an ASCii "1" in the output queue, when the measurement is ready.

*OPC and *OPC? are common commands described on page 9-124 and 9-125.

*OPC reports when operation is complete, via the Status Subsystem described on page 6-14.

Rough Trigger Subsystem Description

The trigger subsystem is the functional part of the CNT-8X that controls the start and stop of measurements. This is the function that the controller interacts with when it controls the measurement sequence.

A simplified model of the CNT-8X's trigger subsystem is a state-machine with four different states. These states are as follows:

IDLE State

– The counter waits for new commands. It is not measuring

WAIT_FOR_BUS_ARM State

– The counter is ready to receive a bus arming signal, GET or ✱TRG

WAIT_ FOR_MEASUREMENT_TO_ START State

– The counter waits for the input signal triggering to start the measurement or block of measurements. If the counter uses arming, it is waiting for the specified arming event.

MEASUREMENT State

– The counter measures. It monitors the hardware and controls the measurement time. If block measurement mode is used, (`ARM:COUN` or `TRIG:COUN`_≥2) the counter stays in this state until all measurements inside the block has been made.

Different actions cause the trigger subsystem to change between the different states. The transitions are shown in . The status is reflected in status byte :STATus:OPERation:CONDition.

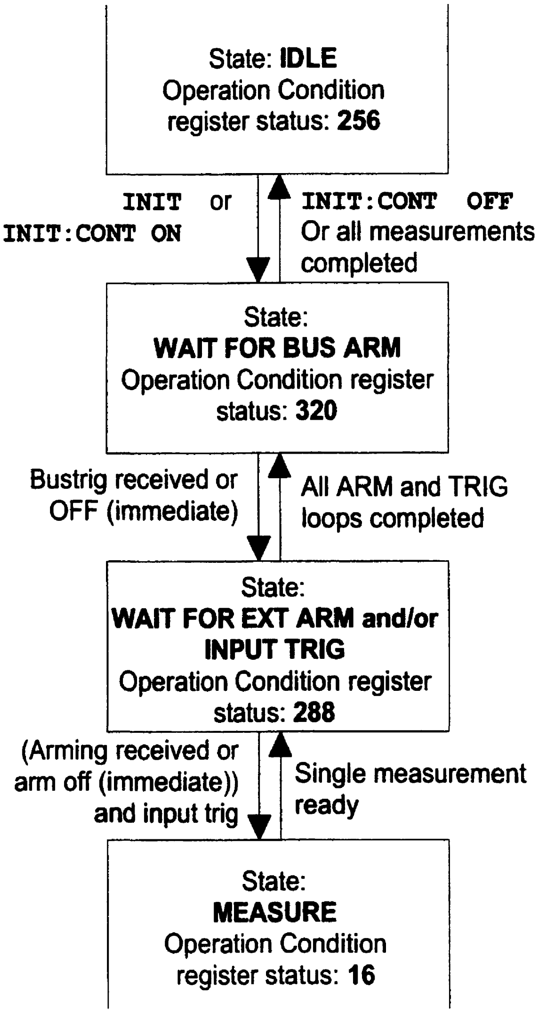


Figure 7-1 *Trigger subsystem states.*

Some Basic Commands

Here follows a description of some basic CNT-8X commands that control the measurement sequence.

CONFigure

The CONFigure command sets up the counter to do the measurement specified by the parameters of the command. The command gives a limited number of parameter options such as:

- Measurement function
- Measurement channel
- Number of measurements and sometimes also the following:
- Measuring time
- Trigger level

The counter sets up the rest of its functions in the best way for the requested measurement. This means that any instrument setting may be changed by this command.

Examples:

- Set up to measure frequency:
  `CONF:FREQ`
- Set up to do 100 frequency measurements:
  `CONF:ARRay:FREQ␣(100)`
- Set up to do 100 frequency measurements on the A-channel:
  `CONF:ARRay:FREQ␣(100),(@1)`
- Set up to do 100 frequency measurements on the A-channel. Expected frequency 10 MHz that should be measured with a resolution of 1 Hz.
  `CONF:ARRay:FREQ␣`
  `(100),10e6,1,(@1)`

INITiate

The INITiate command will normally start a measurement or measurement sequence and store the result internally in the CNT-8X. However the actual action is to change the state of the trigger subsystem from "idle" to "wait_for_bus_arming". The result of changing the state of the trigger subsystem depends on the programming of this subsystem. For example it could be programmed to do the following:

- Make 1000 measurements.
- Wait for a GET/*TRG and then start a measurement.
- Wait for a GET and then make 534 measurements.
- Wait for an arming pulse and make one measurement.
- Wait for an arming pulse and make 234 measurements.

INITiate :CONTinuous

This command sets the counter in a mode where it continues with a new measurement immediately after it has finished the previous one. This is done by not returning the trigger subsystem to the "idle" state.

ABORt

This command stops the current measurement (if any), and sets the trigger subsystem to the "idle" state. This means that the counter is only waiting for new commands.

FETCh?

The FETCh query retrieves measurement data. It could either be a single value (SCALar) or a series of values (ARRay).

Examples:

- Get one measurement: `FETCh?`
- Get 100 measurements: `FETCh:ARRay?␣100`

The number of measurements is defined by the setting of the ARM and TRIG counters. The ARM counter can be set directly by the :CONF and :MEAS commands. The :FETCh:ARRay? query parameter only decides how many measurement results to read out.

READ?

This command simply means to start a measurement or measurement sequence and read data.

This query is identical to: `ABORt;INITiate;FETCh?`

This means that the counter starts a measurement (single or array) after it has aborted any previous measurements. It also returns the result.

Examples:

- Start one measurement and fetch result: `READ?`
- Start measurements and fetch 100 results: `READ:ARRay?␣100`

MEASure?

This query is identical to: `CONFigure;READ?`

This means that the command sets up the counter and starts a measurement/measurement sequence.

Examples:

- Make a frequency measurement: `MEAS:FREQ?`
- Make 100 frequency measurements: `MEAS:ARRay:FREQ?␣(100)`
- Make 100 frequency measurements on the A-channel: `MEAS:ARRay:FREQ? (100),(@1)`
- Make 100 frequency measurements on the A-channel. The expected frequency to be measured is 10MHz with a resolution of 1 Hz. `MEAS:ARRay:FREQ? (100),10e6,1,(@1)`

MEAS:MEM1?, MEAS:MEM? 10

Memory Recall, Measure and Fetch Result

This command is only for PM6681. Use it when you want to measure *several parameters fast*, i.e., switch quickly between measurement functions.

`MEAS:MEM1?` recalls the contents of memory 1 and reads out the result, `MEAS:MEM2?` recalls the contents of memory two and reads out the result etc.

The equivalent command sequence is `*RCL1;READ?`

The allowed range for <N> is 1 to 9. Use the somewhat slower `MEAS:MEMory?␣N` command if you must use memories 10 to 19.

	TIMING	
	Data Format	
Command	ASCII	REAL
`MEAS:MEM1?`	7.9 ms	6.7 ms
`MEAS:MEM?␣1`	9.1 ms	8.0 ms
`*RCL␣ 1;READ?`	10.1 ms	8.9 ms

Basic Measurement Method

A basic measurement method for a system composed of signal sources, measuring object, and measuring devices will be a simple step-by-step procedure. This procedure goes as follows:

Step 1: Set up signal sources

Step 2: Set up measurement devices

Step 3: Trigger measurement devices

Step 4: Read data

Step 5: Evaluate data

The above procedure may be repeated as many times as required.

The methods described here deal with how you should do steps 3 and 4 in the best and most efficient way with the CNT-8X.

Individually Synchronized Measurements

This is a method that you should use when you need to start each measurement externally from the controller. The most probable reason that you should use individually synchronized measurements is that you need to evaluate data in real time and make decisions depending on the acquired data. An example of this could be to tune an oscillator by measuring the output frequency and adjust the oscillator depending on the measured frequency.

Of the many available ways to do this with the CNT-8X, three should be mentioned: `READ?`, `INIT:CONT` and GET and `MEASure?`

■ READ?

The READ? query provides a basic mechanism for this. It ensures that the measurement is started after the counter receives the command. It will also send back the result. The READ? query should be preceded by setting up the counter by using either `CONFigure` or individual programming commands. This command should be used if *no special speed requirements exists.*

■ INIT:CONT and GET

In this method the trigger function is continuously initiated by the command `INITiate:CONTinuous␣1`. This gives you the minimal firmware overhead if you don't change settings in the counter. Set up the counter either by using `CONFigure`, or by using individual programming commands before starting the measuring sequence. Setting up includes switching on the "wait for bus trigger" function with the following command:

```
ARM:START:LAY2:SOURce␣BUS.
```

As default, the counter starts a measurement and sends the result to the controller when receiving a GET or a *TRG command. This method is the *fastest way to make individually synchronized measurements.*

■ MEASure?

The `MEASure?` query sets up the counter, ensures that the measurement is started after the command is received, and also sends the result to the controller. This command has the highest possible degree of compatibility to other instruments; however the command reprograms the counter, and often you need to set up the counter by yourself. This is primarily why we recommend the `READ?` query.

Block Synchronized Measurements

In the block synchronized mode, the controller only starts a sequence of measurements. The counter then measures, without any controller intervention, at the highest possible speed. It "dumps" the results into internal memory and reads them out for evaluation later. This method gives the highest possible data capture speed.

■ READ:ARRay?

This is the basic method for starting up a measurement sequence and reading the data. Set up the counter using CONFigure or individual programming commands before sending the `READ:ARRay?` query. This method will make the measurements with a high measurement rate. The speed depends on a number of individual measurement parameters; see also "General speed improvements" below. The counter stores the data in its internal memory and when it has captured all data, it transfers the resulting array to the controller.

■ INIT + GET + FETCH:ARRay?

The `READ:ARRay?` method has one drawback, it includes some unwanted firmware overhead between when the counter receives the command and it starts the first measurement. This can be solved by setting up the counter to wait for a GET before it starts the measurement sequence. The default actions for GET include sending a single result (the first value) when the counter has completed the sequence. This makes it possible to let the controller wait for the Message AVailable status bit to find out when the data capture is ready. You can read the complete array by using the `FETCH:ARRay?` command. So if, for example, the array size is 4, GET gives the first result in the array and `FETC:ARR?␣3` fetches result two, three, and four.

■ MEAS:ARRay?

The MEASure:ARRay? query ensures that the measurement sequence is started after the command is received. It will also send back the results. It also includes setting up the counter. This is the command that has the highest possible degree of compatibility with other instruments; however, this command reprograms the counter and often you need to program the counter yourself. This is why we recommend using the `READ:ARRay?` query.

General Speed Improvements

The CNT-8X has many options to improve measurement speed. Here you will get a list of actions that you can use to improve the measurement speed. Most of these commands *decrease the average dead time*. The dead time is the time between measurements, that is, from stop to the next start. These actions are all general, that is, they affect the rate of measurements for all measurement methods given above; however, they are especially valuable for the block synchronized measurements. In this mode, the dead time can be as low as 120µs (PM6681).

AUTO

One of the most timesaving commands you can use with the CNT-8X is `:INPut:LEVel:AUTO␣OFF`. This will save the time it takes to determine the trigger levels. (About 50ms/measurement in PM6680B and PM6685, and 85ms in PM6681.)

Display Control

The display can be switched off with the command `DISPlay:ENABle␣OFF`. When you switch off the display, the counter loses the measurement data resolution information. This means that the counter always sends all digits independently of whether or not they are significant. This command reduces the dead time by about 7ms.

GPIB Data Format

You can select the format of the result sent on the GPIB using the FORMat command. Two options exist: ASCii and REAL. The REAL format saves a lot of time both in the instrument and the controller for converting data. However, when the counter uses the REAL format, you lose the measurement data resolution information. It sends the REAL format as a block data element. This means that it sends data as:
#18< 8 bytes real>.

The <8 bytes real> is a double precision binary floating-point code according to IEEE 488.2 / IEEE 754.

Time Measurement Resolution

The basic time measurement resolution can be selected by using the command `SENSe:ACQ:RES␣«HIGH|LOW»`. When you set the resolution to low it will be 100ns for PM6680B and PM6685, and 80ns for PM6681. Instead of the high resolution, which is 0.25ns for PM6680B and PM6685, and 0.05ns for PM6681.

If the counter does real-time calculations and you switch to low resolution, the measurement dead time decreases about 0.6 to 0.9ms. If the counter does not do real-time calculations, then it saves only about 0.05ms per measurement cycle. If the counter does real time calculations with the display switched on, then you can save up to 2ms by selecting the lower resolution.

Automatic Interpolator Calibration (PM6680B/85)

The time interpolation technique achieves the high time resolution. The counter automatically calibrates these interpolators once per measurement cycle. You can control this automatic calibration by using the command:
CAL:INT:AUTO␣«ON|OFF|ONCE».
When you switch calibration off, the measurement cycle time decreases.

Disabling this calibration makes the counter more sensitive to temperature changes. The measurement values may drift away, which results in a larger inaccuracy. However, when the instrument has been switched on for more than 20 minutes and the ambient temperature is stable within ± 5°C, this is no problem. You can also easily recalibrate the interpolators by using the CAL:INT:AUTO ONCE command.

Block Measurements

When dumping measurement results into the internal memory, it is important to program the arming and triggering counters in the best way. For maximum measurement rate use the block armed mode. To do this set:
:ARM:STARt:LAYer:COUNt␣1
and
TRIG:COUNt␣<N>
where <N> is the number of measurements in a block.

Real Time Calculation

Normally the counter calculates the results in "real time." This means that for each measurement, the counter immediately calculates the result based on the raw data information in various counting registers. It needs to do this in order to display the result, make mathematical calculations, limit testing and statistical calculations. It is possible to defer the calculations until the controller requests these values. The counter intermediately stores the measurement data in a packed format. This is done with the command:
:SENSe:INTernal:FORMat␣ PACKed.

This is the most important command when you want to improve the measurement rate for block synchronized measurements.

Note: If you want a very high speed you must set `:AVER:STATE␣OFF` *and* `:ACQ:APER␣ MIN.`

40000 measurements/second

(Only PM6681)

PM6681 can make measure every period of a signal with up to 40 000 Hz. This is called "Back-to-Back" period measurements and in only available via GPIB. The high speed is obtained when the PM6681 measures low-resolution measurements directly to its internal memory. That memory can store 6143 measurement results. When full, the measurement must be stopped, and the results fetched by the controller.)

Note also that some functions are disabled to obtain high measurement speed: You cannot use external arm/trig or hold-off. Statistics is also disabled.

Example: 1000 back-to-back periods

`:SENS:FUNC␣"PER␣1"`
Select period as measurement function

`:INP:COUP␣DC` *Select DC coupling*

`:INP:LEV:AUTO␣OFF` *Turn off Auto Trigger*

`:INP:LEV␣1` *Set fixed trigger level*

`:SENS:ACQ:RES␣LOW` *Select low resolution/high speed measurements*

`:SENS:INT:FORM␣PACK` *Suspends the result calculation until the capture is ready*

`:TRIG:COUN␣1000;:ARM:COUN␣1` *Set up PM6681 for 1000 measurements*

`:INIT` *Start a capture*

`:FETC:ARR?␣1000` *Fetch the 1000 results from the internal PM6681 memory*

Supervising a Process

One typical use of a counter in the industry is to measure a parameter and alert the adjusting machinery when the parameter gets close to the correct value. The machinery now slows down for an accurate final adjustment of the parameter, and stops the adjustment procedure when the value is correct.

An example of such a procedure is when a laser adjusts the value of a resistor that is connected to an oscillator. You measure the frequency of the oscillator and the laser cuts the resistor until the oscillator oscillates at the correct frequency.

Obvious Method

The most obvious way to do this may be as follows:

- Let the counter measure the frequency.
- Send the result to the controller.
- Let the controller, that controls both the laser cutter and the counter, decide when to slow down the cutting procedure, and eventually switch the laser off when the correct frequency is obtained.

This method works fine for slow processes but the bus transfer rate of the counter limits the measuring speed to around 125 measurements/s for PM6680B and PM6685, and 250 measurements/s for PM6681. If all speed increasing actions are taken, and only around 10 measurements/s if no speed increasing actions taken.

Optimal Method

An experienced CNT-8X programmer knows that he can increase the process speed to over 300 measurements/second, by letting the counter do more and the controller less of the job:

- Set up the counter to measure continuously with low resolution, without displaying or reading out any results.
- Set up limit monitoring so that the counter issues a service request when the frequency reaches the limit where the laser should slow down.
- Proceed with one of the following:
- Alternative 1: The program slows down the laser and recalls new narrower limits from the internal counter memory and selects higher resolution.
- Alternative 2: The program slows down the laser and reprograms the counter to make high resolution measurements and reports each measurement result to the controller.
- The controller stops the process when the desired result is obtained.

See also the limit monitoring programming example in Chapter 4.

Speed Summary

The following table summarizes the time that can be gained when fine tuning the measurement process.

The normal dead time between frequency measurements is approximately as follows:

48.8 ms for the PM6680B.
85 ms for the PM6681.
75 ms for the PM6685.

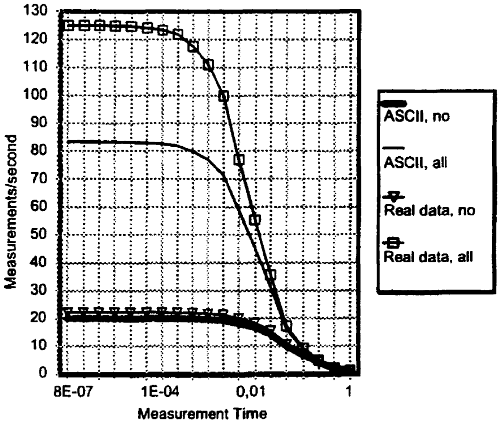
If you should read out the measuring result to the controller add read out time to each result. (Consult your controller manual.)

Individually Synchronized Measurements											
Speed Improvement Actions						Dead Time Between Measurements Including Transfer to Controller					
None, normal READ?	AUTOtrigger OFF	INT:CONT and GET	Display OFF	CAL:INT:AUTO OFF	100ns resolution	ASCII Data Format			Real Data Format		
						PM6680B	PM6681	PM6685	PM6680B	PM6681	PM6685
✔						Approx. 50 ms	Approx. 85 ms*	Approx. 75 ms	Approx. 45 ms	Approx. 85 ms*	Approx. 75 ms
	✔					19 ms	10 ms	23 ms	15 ms	9 ms	20 ms
	✔	✔				12 ms	9 ms	20 ms	9 ms	8 ms	17 ms
	✔	✔	✔	✔		12 ms	5.5 ms	12 ms	9 ms	4.2 ms	9 ms
	✔	✔	✔	✔	✔	12 ms	5.5 ms	12 ms	8 ms	4.2 ms	8 ms

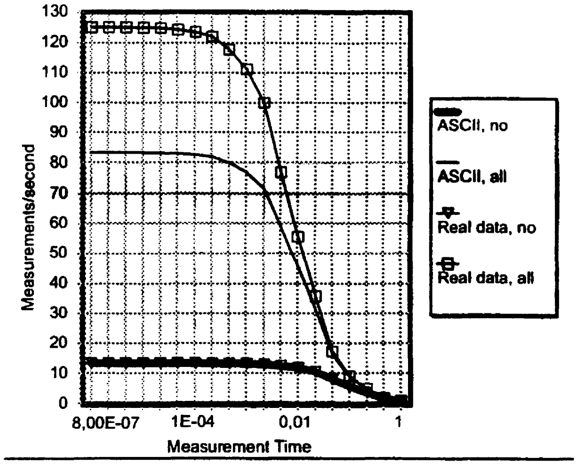
Turning the real-time calculations on/off will not affect the dead time because the calculations are still done inside the measurement loop during the output of data.

* *It takes longer time for PM6681 to determine trigger levels than for PM6680B, why? The reason is that PM6681 must find the correct level from 16 times as many triggerlevel setting steps than thePM6680B/85 (1.25 mV steps versus 20mV steps).*

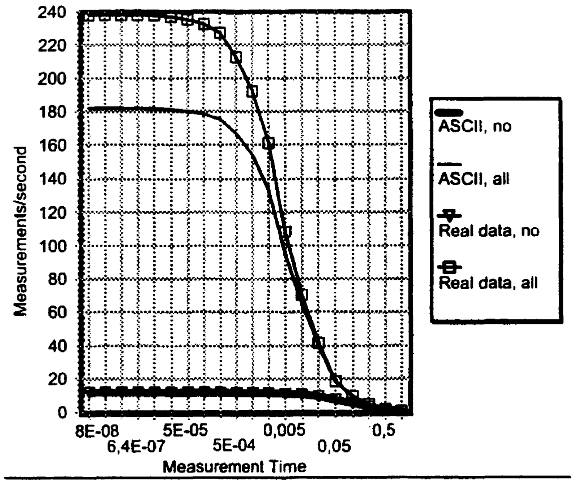
Speed, Individually sync. measurements
PM6680B



Speed, Individually sync. measurements
PM6685



Speed, Individually sync. measurements
PM6681



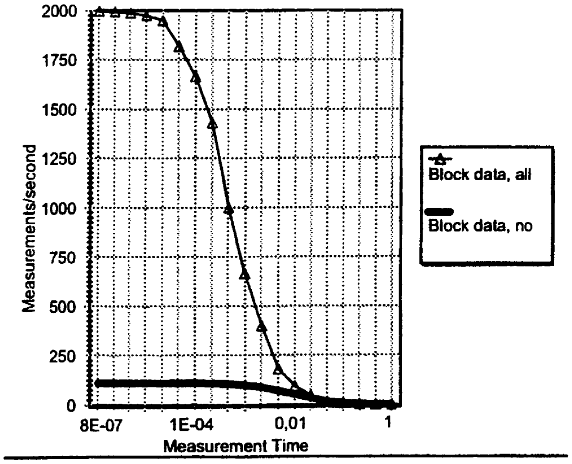
Block Synchronized Measurements **(:ARM:STARt:LAYer:COUNt 1 and TRIG:COUNt <N>)**					
Speed Improvement Actions			**Dead Time Between Measurements**		
CAL:INT:AUTO OFF Display OFF	Realtime calculations OFF ([SENS]:INT:FORM PACK)	Low resolution ([SENS]:ACQ:RES LOW)	PM6680B	PM6681	PM6685
			9 ms	4.5 ms	9 ms
✓			2 ms	1.3 ms	2.5 ms
✓	✓		0.5 ms	0.12 ms	0.6 ms
✓	✓	✓	0.5 ms	0.025 ms*	0.6 ms

1: The GPIB format command will not affect the dead time for the block synchronized mode because the counter captures all data before transferring it to the controller.

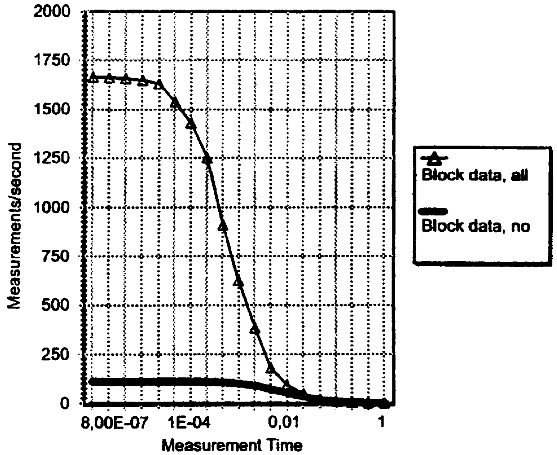
2: Switching the real time calculations on/off In the block synchronized mode will significantly decrease the dead time; however, the time for calculations (2 ms for PM6680B/85 and 1 ms for PM6681) is added to the transfer time.

* *In PM6681, low resolution is used for Back-to-Back period measurements.*
The measuring time has no effect in this mode. Only the Timestamps are used.

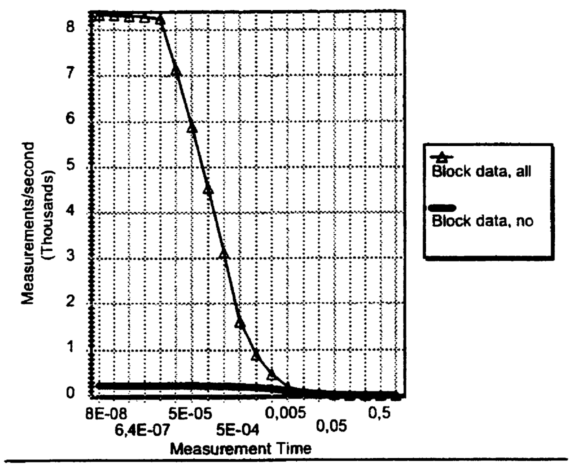
Speed, Block sync. measurements
PM6680B



Speed, Block sync. measurements
PM6685



Speed, Block sync. measurements
PM6681



Calculating the Measurement Speed

When opimizing your program for speed, add the measuring time you use to the dead time, and subtract the time gain for the timesaving commands you intend to use; all times should be expressed in seconds:

$$\frac{1}{Meas.time + DeadTime - \sum TimeGain} =$$

= number of measurements / second.

Where:

Meas.time *is the measurement time you use.*

Deadtime *is the deadtime between measurements after preset. see page 7-11.*

Timegain *is the timegain in the table on page 7-15.*

Timesaving Commands	Time Gain in ms			Sacrifice
	PM6680B	PM6681	PM6685	
	Freq	Freq	Freq	
FREQ:RANG:LOW␣MAX	(23)		(23)	10kHz lower freq. limit for AUTO. This Timesaving is only possible when AUTO is on.
		(55)		50kHz lower freq. limit for AUTO. This Timesaving is only possible when AUTO is on.
INP:LEV:AUTO␣OFF	40	70	52	You have to set trigger levels manually.
DISP:ENAB␣OFF	5.4	3.5	7.7	Only the controller can read the result.
CAL:INT:AUTO␣OFF	0.22	N.A.	0.22	You must instruct the counter to calibrate the interpolators once in a while to maintain accuracy.
SENS:ACQ:RES␣LOW	0.81	0 / 0.1	0.81	Works up to about 40kHz. The resolution of each measurement drops to 100ns for PM6680B/85 and 80ns for PM6681 (PM6681 Only: Gives Back-to-back measurements in period, i.e. every period in a block is measured)
TRIG:COUNT␣2100	0.69	N.A.	0.69	No sacrifice, the program loop in the counter gets shorter, saving time.
TRIG:COUNT␣6143	N.A.	4.2	N.A.	
INT:FORM␣PACK	1.15	1.2	1.15	You cannot use limit monitoring, mathematics etc. in the CALC subsystem, nor the Display or the Output subsystems.
TRIG:COUNT␣1000(or more) STAT:OPER:ENAB␣0 ARM:STA:LAY2:SOUR␣ IMM INP:LEV:AUTO␣OFF (All together)	N.A.	0.13	N.A.	These commands (all together) will increase measurement speed the last step from about 4000 to over 8000 measurements/s
FORM:TINF␣OFF	N.A.	0	N.A.	No timestamping possible. This only influences the read. Time stamps are always registred internally.

☞ *All these time gain estimates are approximations valid for frequency A measurements and may be changed without notice. The time gain/loss depends on measuring function.*

Single "Speed Switch" Command for PM6680B/85

Since many parameters must be set to get the highest measuring speed, it is simpler if you use the macro function:

Send the following lines to turn on macros; define one macro called FASTFREQ and one macro called SLOWFREQ.

```
SEND→ *EMC␣1
SEND→ *DMC␣'FastFreq',
      ":ACQ:APER␣MIN;
      :AVER:STAT␣OFF;
      :INP:LEV:AUTO OFF;
      :DISP:ENAB␣OFF;
      :CAL:INT:AUTO␣OFF;
      :SENS:ACQ:RES␣LOW"

SEND→ *DMC␣'SlowFreq',
      ":ACQ:APER␣200␣ms;
      :AVER:STAT␣ON;
      :INP:LEV:AUTO␣ON;
      :DISP:ENAB␣ON;
      :CAL:INT:AUTO␣ON;
      :SENS:ACQ:RES␣HIGH"
```

Now you just have to send FASTFREQ to the counter to get high measurement speed for frequency measurements, and SLOWFREQ to return to normal measuring speed.

Note that these macros include all speed-increasing commands from the table on the previous page. Omit the ones you do not want to use in your application and the ones that do not apply to your counter.

Single "Speed Switch" Command for PM6681

Since many parameters must be set to get the highest measuring speed, it is simpler if you use the macro function:

Send the following lines to turn on macros; define one macro called FASTFREQ and one macro called SLOWFREQ.

```
SEND→ *EMC␣1

SEND→ *DMC␣'FastFreq',
      ":ACQ:APER␣MIN;
      :AVER:STAT␣OFF;
      :INP:LEV:AUTO␣OFF;
      :DISP:ENAB␣OFF;
      :INT:FORM␣PACK;
      :SENS:ACQ:RES␣LOW;
      :FORM:TINF␣OFF;
      :TRIG:COUNT␣6143;
      :STAT:OPER:ENAB␣0;
      :ARM:STA:LAY2:SOUR␣IMM"

SEND→ *DMC␣'SlowFreq',
      ":ACQ:APER␣200 ms;
      :AVER:STAT␣ON;
      :INP:LEV:AUTO␣ON;
      :DISP:ENAB␣ON;
      :INT:FORM␣REAL;
      :SENS:ACQ:RES␣HIGH;
      :FORM:TINF␣ON;
      :TRIG:COUNT␣1;
      :STAT:OPER:ENAB␣1;
      :ARM:STA:LAY2:SOUR␣BUS"
```

Now you just have to send FASTFREQ to the counter to get high measurement speed for frequency measurements, and SLOWFREQ to return to normal measuring speed.

Note that these macros include all speed-increasing commands from the table on the previous page. Omit the ones you do not want to use in your application.

Chapter 8

Error Messages

Read the Error/Event Queue

You read the error queue with the `:SYSTem:ERRor?` query.

Example:

SEND→ `:SYSTem:ERRor?`
READ← `-100, "Command Error"`

The query returns the error number followed by the error description.

If more than one error occurred, the query will return the error that occurred first. When you read an error, you will also remove it from the queue. You can read the next error by repeating the query. When you have read all errors, the queue is empty, and the `:SYSTem:ERRor?` query will return:

0, "No error"

When errors occur and you do not read these errors, the Error Queue may overflow. Then the instrument will overwrite the last error in the queue with:

–350, "Queue overflow"

If more errors occur they will be discarded.

Read more about how to use error reporting in the Introduction to SCPI chapter

Command Errors		
Error Number	**Error Description**	**Description/Explanation/Examples**
0	No error	
–100	Command error	This is the generic syntax error for devices that cannot detect more specific errors. This code indicates only that a Command Error defined in IEEE-488.2, 11.5.1.1.4 has occurred.
–101	Invalid character	A syntactic element contains a character which is invalid for that type; for example, a header containing an ampersand, SETUP&. This error might be used in place of errors –114, –121, –141, and perhaps some others.
–102	Syntax error Syntax error; unrecognized data	An unrecognized command or data type was encountered; for example, a string was received when the counter does not accept strings.
–103	Invalid separator	The parser was expecting a separator and encountered an illegal character; for example, the semicolon was omitted after a program message unit, *EMC1:CH1:VOLTS5.
–104	Data type error	The parser recognized a data element different than one allowed; for example, numeric or string data was expected but block data was encountered.

Command Errors

Error Number	Error Description	Description/Explanation/Examples
–105	GET not allowed	A Group Execute Trigger was received within a program message (see IEEE-488.2, 7.7).
–108	Parameter not allowed	More parameters were received than expected for the header; for example, the ∗EMC common command accepts only one parameter, so receiving ∗EMC0,,1 is not allowed.
–109	Missing parameter	Fewer parameters were received than required for the header; for example, the ∗EMC common command requires one parameter, so receiving ∗EMC is not allowed.
–110	Command header error	An error was detected in the header. This error message is used when the counter cannot detect the more specific errors described for errors –111 though –119.
–111	Header separator error	A character that is not a legal header separator was encountered while parsing the header; for example, no space followed the header, thus ∗GMC"MACRO" is an error.
–112	Program mnemonic too long	The header contains more than 12 characters (see IEEE-488.2, 7.6.1.4.1).
–113	Undefined header	The header is syntactically correct, but it is undefined for this specific counter; for example, ∗XYZ is not defined for any device.
–114	Header suffix out of range	Indicates that a non-header character has been encountered in what the parser expects is a header element.
–120	Numeric data error Numeric data error; overflow from conversion Numeric data error; underflow from conversion Numeric data error; not a number from conversion	This error, as well as errors –121 through –129, are generated when parsing a data element that appears to be of a numeric type. This particular error message is used when the counter cannot detect a more specific error.

Command Errors

Error Number	Error Description	Description/Explanation/Examples
–121	Invalid character in number	An invalid character for the data type being parsed was encountered; for example, an alpha in a decimal numeric or a "0" in octal data.
–123	Exponent too large	The magnitude of the exponent was larger than 32000 (see IEEE-488.2, 7.7.2.4.1).
–124	Too many digits	The mantissa of a decimal numeric data element contained more than 255 digits excluding leading zeros (see IEEE-488.2, 7.7.2.4.1).
–128	Numeric data not allowed	A legal numeric data element was received, but the counter does not accept it in this position for the header.
–130	Suffix error	This error as well as errors –131 through –139 is generated when parsing a suffix. This particular error message is used when the counter cannot detect a more specific error.
–131	Invalid suffix	The suffix does not follow the syntax described in IEEE-488.2, 7.7.3.2, or the suffix is inappropriate for this counter.
–134	Suffix too long	The suffix contained more than 12 characters (see IEEE-488.2, 7.7.3.4).
–138	Suffix not allowed	A suffix was encountered after a numeric element that does not allow suffixes.
–140	Character data error	This error as well as errors 141 through –149 is generated when parsing a character data element. This particular error message is used when the counter cannot detect a more specific error.
–141	Invalid character data	Either the character data element contains an invalid character or the particular element received is not valid for the header.
–144	Character data too long	The character data element contains more than 12 characters (see IEEE-488.2, 7.7.1.4).
–148	Character data not allowed	A legal character data element was encountered where prohibited by the counter.
–150	String data error	This error as well as errors –151 through –159 is generated when parsing a string data element. This particular error message is used when the counter cannot detect a more specific error.

Command Errors

Error Number	Error Description	Description/Explanation/Examples
–151	Invalid string data	A string data element was expected, but was invalid for some reason (see IEEE-488.2, 7.7.5.2); for example, an END message was received before the terminal quote character.
	Invalid string data; unexpected end of message	
–158	String data not allowed	A string data element was encountered but was not allowed by PM6685 at this point in parsing.
–160	Block data error	This error as well as errors –161 through –169 is generated when parsing a block data element. This particular error message is used when PM6685 cannot detect a more specific error.
–161	Invalid block data	A block data element was expected, but was invalid for some reason (see IEEE-488.2, 7.7.6.2); for example, an END message was received before the length was satisfied.
–168	Block data not allowed	A legal block data element was encountered but was not allowed by the counter at this point in parsing.
–170	Expression data error	This error as well as errors –171 through –179 is generated when parsing an expression data element. This particular error message is used if the counter cannot detect a more specific error.
	Expression data error; floating-point underflow	The floating-point operations specified in the expression caused a floating-point error.
	Expression data error; floating-point overflow	
	Expression data error; not a number	
	Expression data error; different number of channels given	Two channel list specifications, giving primary and secondary channels for 2-channel measurements, contained a different number of channels.

Command Errors		
Error Number	**Error Description**	**Description/Explanation/Examples**
−171	Invalid expression data	The expression data element was invalid (see IEEE-488.2, 7.7.7.2); for example, unmatched parentheses or an illegal character were used.
	Invalid expression data; bad mnemonic	A mnemonic data element in the expression was not valid.
	Invalid expression data; illegal element	The expression contained a hexadecimal element not permitted in expressions.
	Invalid expression data; unexpected end of message	End of message occurred before the closing parenthesis.
	Invalid expression data; unrecognized expression type	The expression could not be recognized as either a mathematical expression, a data element list or a channel list.
−178	Expression data not allowed	A legal expression data was encountered but was not allowed by the counter at this point in parsing.
−180	Macro error	This error as well as errors −181 through −189 is generated when defining a macro or executing a macro. This particular error message is used when the counter cannot detect a more specific error.
−181	Invalid outside macro definition	Indicates that a macro parameter placeholder ($<number) was encountered outside of a macro definition.
−183	Invalid inside macro definition	Indicates that the program message unit sequence, sent with a *DDT or *DMC command, is syntactically invalid (see IEEE-10.7.6.3).
−184	Macro parameter error	Indicates that a command inside the macro definition had the wrong number or type of parameters.
	Macro parameter error; unused parameter	The parameter numbers given are not continuous; one or more numbers have been skipped.
	Macro parameter error; badly formed placeholder	The'$' sign was not followed by a single digit between 1 and 9.
	Macro parameter error; parameter count mismatch	The macro was invoked with a different number of parameters than used in the definition.

Execution errors		
Error Number	**Error Description**	**description/explanation/examples**
–200	Execution error	This is the generic syntax error for devices that cannot detect more specific errors. This code indicates only that an Execution Error as defined in IEEE-488.2, 11.5.1.1.5 has occurred.
–210	Trigger error	
–211	Trigger ignored	Indicates that a GET, *TRG, or triggering signal was received and recognized by the counter but was ignored because of counter timing considerations; for example, the counter was not ready to respond.
–212	Arm ignored	Indicates that an arming signal was received and recognized by the counter but was ignored.
–213	Init ignored	Indicates that a request for a measurement initiation was ignored because another measurement was already in progress.
–214	Trigger deadlock	Indicates that the trigger source for the initiation of a measurement is set to GET and subsequent measurement query is received. The measurement cannot be started until a GET is received, but the GET would cause an INTERRUPTED error.
–215	Arm deadlock	Indicates that the arm source for the initiation of a measurement is set to GET and subsequent measurement query is received. The measurement cannot be started until a GET is received, but the GET would cause an INTERRUPTED error.
–220	Parameter error	Indicates that a program-data-element related error occurred. This error message is used when the counter cannot detect the more specific errors –221 to –229.
–221	Settings conflict Settings conflict; PUD memory is protected Settings conflict; invalid combination of channel and function	Indicates that a legal program data element was parsed but could not be executed due to the current counter state (see IEEE-488.2, 6.4.5.3 and 11.5.1.1.5.)

Execution errors		
Error Number	**Error Description**	**description/explanation/examples**
–222	Data out of range	Indicates that a legal program data element was parsed but could not be executed because the interpreted value was outside the legal range as defined by the counter (see IEEE-488.2, 11.5.1.1.5.).
	Data out of range; exponent too large	The expression was too large for the internal floating-point format.
	Data out of range; below minimum	Data below minimum for this function/parameter.
	Data out of range; above maximum	Data above maximum for this function/ parameter.
	Data out of range; (Save/recall memory number)	A number outside 0 to 19 was given for the save/recall memory.
–223	Too much data	Indicates that a legal program data element of block, expression, or string type received that contained more data than the counter could handle due to memory or related counter-specific requirements.
	Too much data; *PUD string too long	
	Too much data;String or block too long	
–224	Illegal parameter value	Used where exact value, from a list of possible values, was expected.
–230	Data corrupt or stale	Possibly invalid data; new reading started but not completed since last access.
–231	Data questionable	
	Data questionable; one or more data elements ignored	One or more data elements sent with a MEASure or CONFigure command was ignored by the counter.
–240	Hardware error	Indicates that a legal program command or query could not be executed because of a hardware problem in the counter. Definition of what constitutes a hardware problem is completely device specific. This error message is used when the counter cannot detect the more specific errors described for errors –241 through –249.

Execution errors		
Error Number	**Error Description**	**description/explanation/examples**
–241	Hardware missing Hardware missing; (prescaler)"	Indicates that a legal program command or query could not be executed because of missing counter hardware; for example, an option was not installed. Definition of what constitutes missing hardware is completely device specific.
–254	Media full	Indicates that a legal program command or query could not be executed because the media was full; for example, there is no room on the disk. The definition of what constitutes a full media is device specific.
–258	Media protected	Indicates that a legal program command or query could not be executed because the media was protected; for example, the write-protect tab on a disk was present. The definition of what constitutes protected media is device specific.
–260	Expression error	Indicates that an expression-program data-element-related error occurred. This error message is used when the counter cannot detect the more specific errors described for errors –261 through –269.
–261	Math error in expression	Indicates that a syntactically correct expression program data element could not be executed due to a math error; for example, a divide-by-zero was attempted.
–270	Macro error	Indicates that a macro-related execution error occurred. This error message is used when the counter cannot detect the more specific error described for errors –271 through –279.
	Macro error; out of name space	No room for any more macro names.
	Macro error; out of definition space	No room for this macro definition.
–271	Macro syntax error	Indicates that a syntactically correct macro program data sequence, according to IEEE-488.2 10.7.2, could not be executed due to a syntax error within the macro definition (see IEEE-488.2, 10.7.6.3)
–272	Macro execution error	Indicates that a syntactically correct macro program data sequence could not be executed due to some error in the macro definition (see IEEE-488.2, 10.7.6.3)

Execution errors		
Error Number	**Error Description**	**description/explanation/examples**
–273	Illegal macro label	Indicates that the macro label defined in the *DMC command was a legal string syntax, but could not be accepted by the counter (see IEEE-488.2, 10.7.3 and 10.7.6.2); for example, the label was too long, the same as a common command header, or contained invalid header syntax.
–274	Macro parameter error	Indicates that the macro definition improperly used a macro parameter place holder (see IEEE-488.2, 10.7.3).
–275	Macro definition too long	Indicates that a syntactically correct macro program data sequence could not be executed because the string or block contents were too long for the counter to handle (see IEEE-488.2, 10.7.6.1).
–276	Macro recursion error	Indicates that a syntactically correct macro program data sequence could not be executed because the counter found it to be recursive (see IEEE-488.2, 10.7.6.6).
–277	Macro redefinition not allowed	Indicates that a syntactically correct macro label in the *DMC command could not be executed because the macro label was already defined (see IEEE-488.2, 10.7.6.4).
–278	Macro header not found	Indicates that a syntactically correct macro label in the *GMC? query could not be executed because the header was not previously defined.

Standardized Device specific errors		
Error Number	**Error Description**	**description/explanation/examples**
–300	Device specific error	This code indicates only that a Device-Dependent Error as defined in IEEE-488.2, 11.5.1.1.6 has occurred. Contact your local service center.
–311	Memory error	Indicates that an error was detected in the counter's memory. Contact your local service center.
–312	PUD memory lost	Indicates that the protected user data saved by the *PUD command has been lost. Contact your local service center.
–314	Save/recall memory lost	Indicates that the nonvolatile calibration data used by the *SAV? command has been lost. Contact your local service center.
–330	Self-test failed	Contact your local service center.
–350	Queue overflow	A specific code entered into the queue in lieu of the code that caused the error. This code indicates that there is no room in the queue and an error occurred but was not recorded.

Query errors		
Error Number	**Error Description**	**description/explanation/examples**
–400	Query error	This code indicates only that a Query Error as defined in IEEE-488.2, 11.5.1.1.7 and 6.3 has occurred.
–410	Query INTERRUPTED	Indicates that a condition causing an INTERRUPTED Query error occurred (see IEEE-488.2, 6.3.2.3); for example, a query was followed by DAB or GET before a response was completely sent.
	Query INTERRUPTED; in send state	The additional information indicates the IEEE-488.2 message exchange state where the error occurred.
	Query INTERRUPTED; in query state	
	Query INTERRUPTED; in response state	
–420	Query UNTERMINATED	Indicates that a condition causing an UNTERMINATED Query error occurred (see IEEE-488.2, 6.3.2.2); for example, the counter was addressed to talk and an incomplete program message was received.
	Query UNTERMINATED; in idle state	The additional information indicates the IEEE-488.2 message exchange state where the error occurred
	Query UNTERMINATED; in read state	
	Query UNTERMINATED; in send state	
–430	Query DEADLOCKED	Indicates that a condition causing an DEADLOCKED Query error occurred (see IEEE-488.2, 6.3.1.7); for example, both input buffer and output buffer are full and the counter cannot continue.
–440	Query UNTERMINATED after indefinite response	Indicates that a query was received in the same program message after an query requesting an indefinite response was executed (see IEEE-488.2, 6.5.7.5.7.)

CNT-8X Device specific errors (leading 1 only for PM6681)		
Error Number	**Error Description**	**description/explanation/examples**
(1)100	Device operation gave floating-point underflow	A floating-point error occurred during a counter operation.
(1)101	Device operation gave floating-point overflow	A floating-point error occurred during a counter operation.
(1)102	Device operation gave 'not a number'	A floating-point error occurred during a counter operation.
(1)110	Invalid measurement function	The counter was requested to set a measurement function it could not make.
(1)120	Save/recall memory protected	An attempt was made to write in a protected memory.
(1)130	Unsupported command	Indicates a mismatch between bus and counter capabilities.
(1)131	Unsupported boolean command	
(1)132	Unsupported decimal command	
(1)133	Unsupported enumerated command	
(1)134	Unsupported auto command	
(1)135	Unsupported single shot command	
(1)136	Command queue full; last command discarded	The counter has an internal command queue with room for about 100 commands. A large number of commands arrived fast without any intervening query.
(1)137	Inappropriate suffix unit	A suffix unit was not appropriate for the command. Recognized units are Hz (Hertz), s (seconds), Ohm (Ω) and V (Volt).
(1)138	Unexpected command to device execution	A command reached counter execution which should have been handled by the bus.
(1)139	Unexpected query to device execution	A query reached counter execution which should have been handled by the bus.

CNT-8X Device specific errors (leading 1 only for PM6681)		
Error Number	**Error Description**	**description/explanation/examples**
(1)150	Bad math expression format	Only a fixed, specific math expression is recognized by the counter, and this was not it.
(1)160	Measurement broken off	A new bus command caused a running measurement to be broken off.
(1)170	Instrument set to default	An internal setting inconsistency caused the instrument to go to default setting.
(1)190	Error during calibration	An error has occurred during calibration of the instrument.
(1)191	Hysteresis calibration failed	The input hysteresis values found by the calibration routine was out of range. Did you remember to remove the input signal?
(1)200	Message exchange error	An error occurred in the message exchange handler (generic error).
(1)201	Reset during bus input	The instrument was waiting for more bus input, but the waiting was broken by the operator.
(1)202	Reset during bus output	The instrument was waiting for more bus output to be read, but the waiting was broken by the operator.
(1)203	Bad message exchange control state	An internal error in the message exchange handler.
(1)204	Unexpected reason for GPIB interrupt	A spurious GPIB interrupt occurred, not conforming to any valid reason like an incoming byte, address change, etc.
(1)205	No listener on bus when trying to respond	This error is generated when the counter is an active talker, and tries to send a byte on the bus, but there are no active listeners. (This may occur if the controller issues the device talker address before its own listener address, which some PC controller cards has been known to do)
(1)210	Mnemonic table error	An abnormal condition occurred in connection with the mnemonics tables (generic error).
(1)211	Wrong macro table checksum found	The macro definitions have been corrupted (could be loss of memory).
(1)212	Wrong hash table checksum found	The hash table has been corrupted. Could be bad memory chips or address logic. Contact your local service center.
(1)213	RAM failure to hold information (hash table)	The memory did not retain information written to it. Could be bad memory chips or address logic. Contact your local service center.

CNT-8X Device specific errors (leading 1 only for PM6681)		
Error Number	Error Description	description/explanation/examples
(1)214	Hash table overflow	The hash table was too small to hold all mnemonics. Ordinarily indicates a failure to read (RAM or ROM) correctly. Contact your local service center.
(1)220	Parser error	Generic error in the parser.
(1)221	Illegal parser call	The parser was called when it should not be active.
(1)222	Unrecognized input character	A character not in the valid IEEE488.2 character set was part of a command.
(1)223	Internal parser error	The parser reached an unexpected internal state.
(1)230	Response formatter error	Generic error in the response formatter.
(1)231	Bad response formatter call	The response formatter was called when it should not be active.
(1)232	Bad response formatter call (eom)	The response formatter was called to output an end of message, when it should not be active.
(1)233	Invalid function code to response formatter	The response formatter was requested to output data for an unrecognized function.
(1)234	Invalid header type to response formatter	The response formatter was called with bad data for the response header (normally empty)
(1)235	Invalid data type to response formatter	The response formatter was called with bad data for the response data.
(1)240	Unrecognized error number in error queue	An error number was found in the error queue for which no matching error information was found.

See also Error Messages in Appendix 1 of the Operators Manual.

This page is intentionally left blank.

Chapter 9

Command Reference

This page is intentionally left blank.

Abort

:ABORt

:ABORt

PM6680B/81/85

Abort Measurement

The ABORt command terminates a measurement. The trigger subsystem state is set to “idle-state”.

Type of command:

Aborts all previous measurements if `*WAI` is not used.

Complies to standards: SCPI 1991.0, confirmed.

Arming Subsystem

```
:ARM
[ :STARt / :SEQuence [ 1 ] ]
        :LAYer2
              :[ IMMediate ]
              :SOURce      ␣   BUS | IMMediate
        [ :LAYer [ 1 ] ]
              :COUNt       ␣   <Numeric value> | MIN | MAX
              :DELay                          ␣   <Numeric value> | MIN | MAX
              :ECOunt      ␣   <Numeric value> | MIN | MAX
              :SLOPe       ␣   POSitive|NEGative
              :SOURce      ␣   EXTernal2 | External4 | IMMediate
:STOP / SEQuence2
        [ :LAYer [ 1 ] ]
        :DELay                                      ␣   <Numeric value> | MIN | MAX        (Only PM6680B /
                                                     PM6681)
        :ECOunt                          ␣   <Numeric value> | MIN | MAX        (Only PM6680B / PM6681)
        :SLOPe                           ␣   POSitive | NEGative
        :SOURce                          ␣   EXTernal2 | EXTernal4 | IMMediate | TIMerf
```

:ARM :COUNt

PM6680B/81/85

⌴«<Numeric value>|MIN|MAX»

No. of Measurements on each Bus arm

This count variable controls the upward exit of the "wait-for-bus-arm" state (`:ARM:STARt:LAY1`). The counter loops the trigger subsystem downwards COUNt number of times before it exits to the idle state.

This means that a COUNt No. of measurements can be done for each Bus arming or INITiate.

The actual number of measurements made on each INIT *is equal to:* (:ARM:START:COUNT)*(:TRIG:START:COUNT)

Parameters:

<Numeric value> is a number between 1 and 65 535. (1 switches the function OFF.)

MIN gives 1

MAX gives 65 535

Returned format: <Numeric value>↵

Example:

SEND→ `:ARM:COUN⌴100`↵

***RST condition:** 1

Complies to standards: SCPI 1991.0, confirmed

PM6680B/81/85

:ARM :DELay

⌴ «<Numeric value> | MIN | MAX»

Delay after External Start Arming

This command sets a delay between the pulse on the arm input and the time when the counter starts measuring. The delay is only active when the following is selected:

`:ARM:STARt:SOURce 8 EXT4.`

Range: 200 ns to 1.67 s.

The optional node [:FIXed] is only accepted by PM6681.

Parameters:

<Numeric value> is a number between $200*10^{-9}$ and 1.67s.

MIN gives 0 which switches the delay OFF.

MAX gives 1.67 s

Returned format: <Numeric value>↵

Example:
SEND→ :ARM:DEL ⌴0.1↵

***RST condition:** 0

Complies to standards: SCPI 1991.0, confirmed.

PM6680B PM6681

:ARM :ECOunt

⌴ «<Numeric value> | MIN | MAX»

External Events before Start Arming

This command sets the number of negative edges required on the B-input (EXT2) before the counter starts measuring (Start Arming Delay by events). Start Arming delay by events cannot be used at the same time as stop Arming delay by events `(:ARM:STOP:ECO)`.

The delay is only active when `:ARM:START:SOUR ⌴EXT2|EXT4` *is selected.*

Only one of the delays: `:ARM:STAR:DEL`, `:ARM:STOP:DEL`, `:ARM:STAR:ECO`, and `:ARM:STOP:ECO` can be used at a time. When you program this delay, the other three delays will be reset to their `*RST` values.

Parameters:

<Numeric value> is a number between 2 and 16 777 215. 1 switches the delay by events OFF.
SEND→ `:ARM:ECO ⌴25`↵

Returned format: <Numeric value>↵

***RST condition:** 1

Complies to standards: SCPI 1991.0, confirmed.

:ARM :LAYer2

PM6680B/81/85

Bus Arming Override

This command overrides the waiting for bus arm, provided the source is set to bus. When this command is issued, the counter will immediately exit the "wait-for-bus-arm" state.

The counter generates an error if it receives this command when the trigger subsystem is not in the "wait-for-bus-arm" state.

If the Arming source is set to Immediate, this command is ignored.

Example:

SEND→ `:ARM:LAY2↵`

Complies to standards: SCPI 1991.0, confirmed.

:ARM :LAYer2 :SOURce

⌴ «BUS | IMMediate»

PM6680B/81/85

Bus Arming On/Off

Switches between Bus and Immediate mode for the "wait-for-bus-arm" function, (layer 2). GET and `*TRG` triggers the counter if Bus is selected as source.

If the counter receives GET/ `*TRG` when not in "wait-for-bus-arm" state, it ignores the trigger and generates an error.

It also generates an error if it receives GET/ `*TRG` and bus arming is switched off (set to IMMediate).

Returned format: BUS|IMM↵

Example:

SEND→ `:ARM:LAY2:SOUR ⌴ BUS↵`

Complies to standards: SCPI 1991.0, confirmed.

PM6680B/81/85

:ARM :SLOPe
␣ «POSitive|NEGative»

External Arming Start Slope

Sets the slope for the start arming condition.

Returned format: POS|NEG↵

Example:
SEND→ `:ARM:SLOP ␣ NEG↵`

***RST condition:** POS

Complies to standards: SCPI 1991.0, confirmed.

PM6680B/81/85

:ARM :SOURce
␣ «EXTernal2 | EXTernal4 | IMMediate»

External Arming Start Source

Selects channel 4 (Input E) as arming input, or switches off the start arming function. When switched off the DELay is inactive.

Parameters:

EXTernal2 is input B (Only PM6680B/81)

EXTernal4 is input E

IMMediate is Start arming OFF

Returned format: EXT2 | EXT4 | IMM↵

Example:
SEND→ `:ARM:SOUR ␣ EXT4↵`

***RST condition:** IMM

Complies to standards: SCPI 1991.0, confirmed.

:ARM :STOP :DELay

PM6680B PM6681

⌣ «<Numeric value> | MIN | MAX»

Delay after External Stop Arming

This command sets a delay between stop slope of the pulse on the arm input and the time when the counter stops measuring. The delay is only active when the following is selected:
`:ARM:STOP:SOURce ⌣ EXT2|EXT4.`

Range: 200 ns to 1.67 s.

The optional node [:FIXed] *is only accepted by PM6681.*

Parameters:

<Numeric value> is a number between $200*10^{-9}$ *and 1.67s.*

MIN gives 0 which switches the delay OFF.

MAX gives 1.67 s

Returned format: <Numeric value>↵

Example:
SEND→ `:ARM:STOP:DEL ⌣ 0.1↵`

***RST condition:** 0

Complies to standards: SCPI 1991.0, confirmed.

:ARM :STOP :ECOunt

PM6680B/81/85

⌣ «<Numeric value> | MIN | MAX»

External Events before Stop Arming

This command sets the number of stop slopes are required on the external stop arming source before the counter stop measuring (Stop Arming Delay by events). Stop Arming delay by events cannot be used at the same time as start Arming delay by events `(:ARM:START:ECO).`

The delay is only active when `:ARM:STOP:SOUR EXT2|EXT4` *is selected.*

Only one of the delays: `:ARM:STAR:DEL, :ARM:STOP:DEL, :ARM:STAR:ECO, and :ARM:STOP:ECO` can be used at a time. When you program this delay, the other three delays will be reset to their `*RST values.`

Parameters:

<Numeric value> is a number between 2 and 16 777 215. 1 switches the delay by events OFF.
SEND→ `:ARM:STOP:ECO ⌣ 25↵`

Returned format: <Numeric value>↵

***RST condition:** 1

Complies to standards: SCPI 1991.0, confirmed.

PM6680B/81/85

:ARM :STOP :SLOPe

␣ «POSitive | NEGative»

External Stop Arming Slope

Sets the slope for the stop arming condition.

Returned format: POS|NEG↵

Example:

SEND→ `:ARM:STOP:SLOP ␣ NEG↵`

***RST condition:** POS

Complies to standards: SCPI 1991.0, confirmed.

PM6680B/81/85

:ARM :STOP :SOURce

␣ «EXTernal2 | EXTernal4 | IMMediate»

External Stop Arming Source

Selects between channel 2 (Input B) and channel 4 (Input E) as stop arming input, or switches off the stop arming function.

Parameters:

EXTernal2 is input B (Only PM6680B/81)

EXTernal4 is input E

IMMediate is Stop arming OFF

Returned format: EXT4|IMM↵

Example:

SEND→ `:ARM:STOP:SOUR ␣ EXT4↵`

***RST condition:** IMM

Complies to standards: SCPI 1991.0, confirmed.

This page is intentionally left blank.

Calculate Subsystem

```
:CALCulate
 :STATe                                         ␣  ON|OFF
 :DATA?
 :IMMediate
 :MATH
        [:EXPRession]                     ␣  <Numeric expression>
        :STATe                                  ␣  ON|OFF
        :AVERage
             [:STATe]                 ␣  ON|OFF
             :TYPE                        ␣  MIN|MAX|SDEViation|MEAN
             :COUNt                   ␣  <Numeric value>|MIN|MAX
 :LIMit
        [:STATe]␣  ON|OFF
             :UPPer
                          [:DATA]     ␣  <Numeric value>|MIN|MAX
                          :STATe          ␣  ON|OFF
             :LOWer
                          [:DATA]     ␣  <Numeric value>|MIN|MAX
                          :STATe          ␣  ON|OFF
        :FAIL?
```

:CALCulate :AVERage :COUNt

PM6680B PM6681

⎵ < No. of samples>

Sample Size for Statistics

Sets the number of samples to use in statistics sampling.

Parameters: <No. of samples> is a number in the range of 1 to 65535.

Returned format: < No. of samples>↵

***RST condition:** 100

:CALCulate :AVERage :STATe

PM6680B PM6681

⎵ < Boolean >

Enable Statistics

Switches On/Off the statistical function. Note that the CALCulate subsystem is automatically enabled when the statistical functions are switched on. This means that other enabled calculate sub-blocks are indirectly switched on. The statistics must be enabled before the measurements are performed. When the statistical function is enabled, the counter will keep the trigger subsystem initiated until the `:CALC:AVER:COUNT` variable is reached. This is done without any change in the trigger subsystem settings. Consider that the trigger subsystem is programmed to perform 1000 measurements when initiated. In such a case, the counter must make 10000 measurements if the statistical function requires 9500 measurements because the number of measurements must be a multiple of the number of measurements programmed in trigger subsystem (1000 in this example).

Parameters

<Boolean> = (1/ON | 0/OFF)

Returned format: <1|0↵

***RST condition:** OFF

PM6680B PM6681

:CALCulate :AVERage :TYPE

˽ «MAX|MIN|MEAN|SDEViation»

Statistical Type

Selects the statistical function to be performed.

☞ *You must use* :CALC:DATA? *to read the result of statistical operations.* :READ?, :FETC? *will only send the results that the statistical operation is based on.*

Parameters:

MAX returns the maximum value of all samples taken under `:CALC:AVER` control.

MIN returns the minimum value of all samples taken under `:CALC:AVER` control.

MEAN returns the mean value of the samples taken: $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$

SDEV Returns the standard deviation: $s = \sqrt{\frac{1}{n-1}\left(\sum x_i^2 - \frac{1}{n}\left(\sum x_i\right)^2\right)}$

Returned format: MAX|MIN|MEAN|SDEV↵

***RST condition:** MEAN

PM6680B/81/85

:CALCulate :DATA?

Fetch calculated data

Fetches data calculated in the post processing block. Use this command to fetch the calculated result without making a new measurement.

Returned Format:

<Decimal data>↵

Example for PM6685:

SEND→ `:CALC:MATH:STAT ˽ ON;:CALC:MATH ˽ (X ˽ - ˽ 10.7E6);:INIT; *OPC`

Wait for operation complete

SEND→ `:CALC:DATA?`

READ← `<Measurement ˽ result ˽ minus ˽ 10.7E6>`

Example for PM6680B/81

SEND→ `:CALC:MATH:STAT˽ON;:CALC:MATH˽(((1˽*˽X)˽-˽10.7E6)˽/˽1);:init; *OPC`

Wait for operation complete

SEND→ `:CALC:DATA?`

READ← `<Measurement ˽ result ˽ minus ˽ 10.7E6>`

***RST condition:**

Event, no `*RST` condition.

Complies to standards: SCPI 1991.0, Confirmed

:CALCulate :IMMediate

PM6680B/81/85

Recalculate Data

This event causes the calculate subsystem to reprocess the statistical function on the sense data without reacquiring the data. Query returns this reprocessed data.

This command is not very useful in PM6685, but is accepted to maintain compatibility with the other counters in the CNT-8X series of counters.

Returned format: <Decimal data>↵

Where: <Decimal data> is the recalculated data.

Example:

```
SEND→ :CALC:AVER:STAT ␣ ON;TYPE ␣ SDEV;:INIT;*OPC
    Wait for operation complete

SEND→ :CALC:DATA?
READ← <Value ␣ of ␣ standard ␣ deviation>
SEND→ :CALC:AVER:TYPE ␣ MEAN
SEND→ :CALC:IMM?
READ← <Mean ␣ value>
```

***RST condition:** Event, no *RST condition.

Complies to standards: SCPI 1991.0, Confirmed.

:CALCulate :LIMit

PM6680B/81/85

␣ <Boolean>

Enable Monitoring of Parameter Limits

Turns On/Off the limit-monitoring calculations.

Limit monitoring makes it is possible to get a service request when the measurement value falls below a lower limit, or rises above an upper limit.

Two status bits are defined to support limit-monitoring. One is set when the results are greater than the UPPer limit, the other is set when the result is less than the LOWer limit. The bits are enabled using the standard *SRE command and :STAT:DREG0:ENAB. Using both these bits, it is possible to get a service request when a value passes out of a band (UPPer is set at the upper band border and LOWer at the lower border) OR when a measurement value enters a band (LOWer set at the upper band border and UPPer set at the lower border).

Turning the limit-monitoring calculations On/Off will not influence the status register mask bits, which determine whether or not a service request will be generated when a limit is reached. Note that the calculate subsystem is automatically enabled when limit-monitoring is switched on. This means that other enabled calculate sub-blocks are indirectly switched on.

Parameters <Boolean> = (1/ON | 0/OFF)

Returned format: 1|0↵

***RST condition:** OFF

See also: Example 1 in Chapter 4 deals with limit-monitoring.

Complies to standards: SCPI 1991.0, confirmed.

PM6680B/81/85

:CALCulate :LIMit :FAIL?

Limit Fail

Returns a 1 if the limit testing has failed (the measurement result has passed the limit), and a 0 if the limit testing has passed.

The following events reset the fail flag:

– Power-on

– `*RST`

– A `:CALC:LIM:STAT OFF` → `:CALC:LIM:STAT ␣ON` transition

– Reading a 1 with this command.

Returned format: 1| 0↵

Example:

SEND→ `SENS:FUNC ␣ FREQ;:CALC:LIM:STAT ␣ ON;:CALC:LIM ␣ :HIGH␣ 1E3;READ?;*WAI;:CALC:LIM:FAIL?`

READ← `1`

if frequency ia above 1kHz, otherwize 0

Complies to standards: SCPI 1991.0, confirmed.

PM6680B/81/85

:CALCulate :LIMit :LOWer

␣ «<Decimal data>|MAX|MIN»

Set Low Limit

Sets the value of the 'Lower Limit' , i.e., the lowest measurement result allowed before the counter generates a 1 that can be read with `:CALCulate:LIMit:FAIL?`, or by reading the corresponding status byte.

Parameters

Parameter range: $-9.9*10^{+37}$ to $+9.9*10^{+37}$.

Returned format: < Decimal data>↵

***RST condition:** 0

Complies to standards: SCPI 1991.0, confirmed.

:CALCulate :LIMit :LOWer :STATe

PM6680B/81/85

␣ <Boolean>

Check Against Lower Limit

Selects if the measured value should be checked against the lower limit.

Parameters <Boolean> = (1/ON | 0/OFF)

Returned format: 1| 0 ↵

***RST condition:** 0

Complies to standards: SCPI 1991.0 confirmed.

:CALCulate :LIMit :UPPer

PM6680B/81/85

␣ «<Decimal data>|MAX|MIN»

Set Upper Limit

Sets the value of the 'Upper Limit', i.e., the highest measurement result allowed before the counter generates a 1 that can be read with `:CALCulate:LIMit:FAIL?`, or by reading the corresponding status byte.

Parameters

Range: $-9.9*10^{+37}$ to $+9.9*10^{+37}$

Returned format: <Decimal data>↵

***RST condition:** 0

Complies to standards: SCPI 1991.0, confirmed.

:CALCulate :LIMit :UPPer :STATe ␣<Boolean>

Check Against Upper Limit

Selects if the measured value should be checked against the upper limit.

Parameters <Boolean> = (1/ON | 0/OFF)

Returned format: 1| 0 ↵

***RST condition:** 0

Complies to standards: SCPI 1991.0, confirmed.

:CALCulate :MATH

␣ (<expression>)

PM6685

Select Mathematical Expression

Defines the mathematical expression used for mathematical operations. This function equals the nulling function from the front panel.

☞ *The data type <expression data> must be typed within parentheses.*

☞ *The operand must be surrounded by space characters.*

Parameters

<expression> is: (X + K) ***No deviations are allowed from this form.***

K can be any positive or negative numerical constant within the range –9.9E+37 to +9.9E+37
X is the measurement result.

Returned format: <expression>↵ Where <expression> is the expression selected.

Example This example subtracts 10700000 from the measurement result.

```
SEND→:CALC:MATH ␣ (X ␣ - ␣ 10.7E6)
```

Example 2 This example defines the mathematical expression, enables postprocessing and mathematics, make a measurement, and fetches the result:

```
SEND→:CALC:MATH ␣ (X ␣ - ␣ 10.7E6);MATH:STATE ␣ ON;:READ?
```

***RST condition:** $(X - 1.0000\,E{+}7)$

Complies to standards: SCPI 1991.0 Confirmed.

PM6680B PM6681

:CALCulate :MATH

␣ (<expression>)

Select Mathematical Expression

Defines the mathematical expression used for mathematical operations. This function equals the nulling function from the front panel.

The data type <expression data> must be typed within parentheses.

Parameters

<expression> is one of the following two mathematical expressions:

*((K␣*_X)␣+␣L)␣/␣M or ((K␣/␣X)␣+␣L)␣/␣M* ***No deviations are allowed.***

K, L and M can be any positive or negative numerical constant, or use XOLD for the last, previously measured value.

Each operand must be surrounded by space characters.

Example

SEND→`:CALC:MATH ␣ (((1 ␣ * ␣ X) ␣ - ␣ 0) ␣ / ␣ XOLD)`

This example gives a relative result from the last measuring result.

***RST condition:**

(((1 * X) + 0) ⁄ 1) (No calculation)

Returned format: <expression>↵

Complies to standards: SCPI 1991.0 Confirmed.

PM6680B/81/85

:CALCulate :MATH :STATe

␣ <Boolean>

Enable Mathematics

Switches on/off the mathematical function. Note that the CALCulate subsystem is automatically enabled when MATH operations are switched on. This means that other enabled calculate sub-blocks are indirectly switched on. Switching off mathematics, however, does not switch off the CALCulate subsystem.

Parameters:

<Boolean> = (1/ON | 0/OFF)

Returned syntax: 0|1

Example

SEND→`:CALC:MATH:STAT ␣ 1`

This example switches on mathematics.

***RST condition:** OFF

Complies to standards: SCPI 1991.0, confirmed.

:CALCulate :STATe

PM6680B/81/85

␣ <Boolean>

Enable Calculation

Switches on/off the complete post-processing block. If disabled, neither mathematics or limit-monitoring can be done.

Parameter

<Boolean> = (1/ON | 0/OFF)

SEND→ `:CALC:STAT ␣ 1`

Switches on Post Processing.

Returned format: 1|0↵

***RST condition:** OFF

Complies to standards: SCPI 1991.0, Confirmed

Calibration Subsystem

```
:CALibration
     :INTerpolator
                :AUTO          ␣  <Boolean>|ONCE          (Only PM6680B, PM6685)
```

☞ *PM6681 has factory calibrated interpolators, and calibration cannot be changed by the operator.*
Calibration of the PM6681 input hysteresis is done in the Diagnostis subsystem.

:CALibration :INTerpolator :AUTO

PM6680B/85

␣ <Boolean>| ONCE

Calibration of Interpolator

The PM6680B/85 are reciprocal counters that uses an interpolating technique to increase the resolution. In time measurements, for example, interpolation increases the resolution from 100 ns to 0.25 ns.

The counter calibrates the interpolators automatically once for every measurement when this command is ON. When this command is OFF, the counter does no calibrations but uses the values from the last preceding calibration. The intention of this command is to turn off the auto calibration for applications that dump measurements into the internal memory. This will increase the measurement speed.

Parameters

<Boolean> = (1 | ON / 0 | OFF)

Returned format: 1|0↵

***RST condition:** ON

See also:

Chapter 6, 'How to Measure Fast'.

Configure Function

Set up Instrument for Measurement

```
:CONFigure
        [:SCALar]<Measuring Function>     ␣  <Parameters>,(<Channels>)]
        :ARRay<Measuring Function>        ␣  (<Array Size>)[,<Parameters>,(<Channels>)]
```

The array size for :MEASure *and* :CONFigure, *and the channels, are expression data that must be in parentheses ( ).*

Measuring Function, Parameters and Channels are explained on page 9-54.

The counter uses the default Parameters and Channels when you omit them in the command.

:CONFigure :<Measuring Function> [␣ <parameters>[,(<channels>)]]

PM6680B/81/85

Configure the counter for a single measurement

Use the configure command instead of the measure query when you want to change other settings, for instance, the input settings before making the measurement and fetching the result.

The :CONFigure command controls the settings of the Input, Sense and Trigger subsystems in the counter in order to make the best possible measurement. It also switches off any calculations with :CALC:STATE ␣ OFF.

:READ? or :INITiate;:FETCh? will make the measurement and read the resulting measured value.

Since you may not know exactly what settings the counter has chosen to configure itself for the measurement, send an *RST before doing other manual set up measurements.

Parameters

<Measuring Function>, <Parameters> and <Channels> are defined on page 9-54. The optional parameter :SCALar means that one measurement is to be done.

Returned format: <String>↵

<String> contains the current measuring function and channel. The response is a <String data element> containing the same answer as for `[:SENSe]:FUNCtion?`.

Example:

SEND→ `:CONF:FREQ:RAT␣(@3),(@1)`

Configures the counter for freq. ratio C/A.

See also: 'Explanations of the Measuring Functions' starting on page 9-59.

Complies to standards: SCPI 1991.0, confirmed.

80B/81/85

:CONFigure :ARRay :<Measuring Function>

␣ (<array size>)[,<parameters> [,(<channels>)]]

Configure the counter for an array of measurements

The `:CONFigure:ARRay` command differs from the `:CONFigure` command in that it sets up the counter to perform the number of measurements you choose in the <array size>.

To perform the selected function, you must trigger the counter with the `:READ:ARRay?` or `:INITiate;:FETCh:ARRay?` queries.

Parameters <array size> sets the number of measurements in the array (1 to 2500).

<Measuring Function>, <Parameters>, and <Channels> are defined on page 9-54.

Example:

```
SEND→ :CONF:ARR:PER ␣ (7),5E-3,1E-6,(@4)
```

This example sets up the counter to make seven period measurements. The expected result is 5 ms, and the required resolution is 1 µs. The EXT ARM input is the measuring input.

To make the measurements and fetch the seven measurement results:

```
SEND→ :READ:ARR? ␣ 7

READ← 5.23421E-3,5.12311E-3,5.87526E-3, ␣
      5.50345E-3,5.33901E-3,5.25501E-3, ␣ 5.03571E-3
```

Complies to standards: SCPI 1991.0, confirmed.

This page is intentionally left blank.

Diagnostics Subsystem

```
:DIAGnostic
      :CALibration
            :INPut[1]
                        :HYSTeresis␣   OFF | ONCE
            :INPut2
                        :HYSTeresis␣   OFF | ONCE
```

:DIAGnostic:CALibration:INPut[1|2]:HYSTeresis 81
␣ «OFF | ONCE»

Input comparator hysteresis calibration

These two commands measure and save the hysteresis levels of the input comparator. This makes it possible to achieve a trig level accuracy of 2.5 mV, which is important in measurement functions such as phase, to get the best possible results.

☞ *Since the calibration compensates for the temperature drift of the input amplifier, it should be made at the same temperature as the accurate measurement is to be made at.*

Before sending these commands, be sure to disconnect any signal leads from the input connector of the input you want to calibrate.

If error code 1191 is generated, the calibration constants are out of range and you must calibrate again. Check that no cables are connected to input A or input B before recalibrating.

When the input calibration procedure can be done without error codes, the calibration is correct.

Example:

SEND→ `:DIAG:CAL:INP:HYST ␣ ONCE`

This string calibrates both input A and input B.

Returned format: OFF ↵

When queried, these commands always return OFF .

***RST condition:** *RST does not affect these calibration data.

Display Subsystem

```
:DISPlay
        :ENABle⎵   ON OFF
```

:DISPlay :ENABle

PM6680B/81/85

␣ < Boolean >

Display State

Turns On/Off the updating of the entire display section. This can be used for security reasons or to improve the GPIB speed, since the display does not need to be updated. Turning off the display reduces the dead time between measurements by about 7 ms.

When the display is turned off, the information about the measurement resolution is lost. That is, the counter will always send a full 12 digit mantissa independent of the measurement resolution.

Parameters:

Where <Boolean> = (1 / ON | 0 / OFF)

Returned format: 1|0 ↵

***RST condition:** ON

See also: Chapter 6, 'How to Measure Fast'.

Complies to standards: SCPI 1991.0, confirmed.

Fetch Function

```
:FETCh
          [:SCALar]?
          :ARRay?⎵  <Array Size>|MAX
```

:FETCh?

PM6680B/81/85

Fetch One Result

The fetch query retrieves one measuring result from the measurement result buffer of the counter without making new measurements. Fetch does not work unless a measurement has been made by the `:INITiate`, `:MEASure?`, or `:READ?` commands.

If the counter has made an array of measurements, `:FETCh?` fetches the first measuring results first. The second `:FETCh?` fetches the second result and so on. When the last measuring result has been fetched, fetch starts over again with the first result.

The same measuring result can be fetched again and again, as long as the result is valid, i.e., until the following occurs:

- `*RST` is received.
- an `:INITiate,` ⌴:MEASure or `:READ` command is executed
- any reconfiguration is done.
- an acquisition of a new reading is started.

If the measuring result in the output buffer is invalid but a new measurement has been started, the fetch query completes when a new measuring result becomes valid. If no new measurement has been started, an error is returned.

Where the optional `:SCALar` means that one result is retrieved.

Returned format: <data>↵

The format of the returned data is determined by the format commands `:FORMat` and `:FORMat:FIXed`.

Complies to standards: SCPI 1991.0, confirmed.

:FETCh :ARRay?

␣ «<fetch array size>|MAX»

Fetch an Array of Results

:FETCh:ARRay? query differs from the :FETCh? query by fetching several measuring results at once.

An array of measurements must first be made by the commands. :INITiate, :MEASure:ARRay? or :CONFigure:ARRay; :READ?

If the array size is set to a positive value, the first measurement made is the first result to be fetched.

When the counter has made an array of measurements, :FETCh:ARRay? ␣10 fetches the first 10 measuring results from the output queue. The second :FETCh:ARRay? ␣10 fetches the result 11 to 20, and so on. When the last measuring result has been fetched, fetch:array starts over again with the first result.

In totalizing for instance, you may want to read the last measurement result instead of the first one. This is possible if you set the array size to a negative number. Example: :FETCh:ARRay? ␣–5 fetches the last five results. The output queue pointer is not altêred when the array size is negative. That is, the example above always gives the last five results every time the command is sent.

:FETCh:ARRay? ␣–1 is useful to fetch intermediate results in free-running or array measurements without interrupting the measurement.

Parameters

:ARRay means that an array of retrievals are done for each :FETCh command.
<fetch array size> is the number of retrievals in the array. This number must not exceed the number of measuring results in the measurement result buffer. The <SIZE> parameter maximum limit is depending on the :SENSe:INTernal:FORMat command as follows:

Format	Measuring function	Array Size	
		PM6680B/85	*PM6681*
Real:	All functions	2048	7019
Packed:	Frequency, Period, Ratio Totalize	2166	6143
	Pulse Width	764	4466
	Time-Interval, Rise/Fall time		4466
	Phase, Duty Cycle,Volt		7019
	Low resolution Frequency and Period		8191
	Low Res. Time-Interval and Pulse Width		4095

MAX means that all the results in the output buffer will be fetched.

Returned format: <data>[,<data>]↵

The format of the returned data is determined by the format commands `:FORMat` and `:FORMat:FIXed`.

Example:

If `:MEAS:ARR:FREQ? ␣(4)` *gives the results* `1.1000,1.2000,1.3000,1.4000`
`:FETC:ARR ␣2` *fetches the results* `1.1000,1.2000`
`:FETC:ARR ␣2` *once more fetches the results* `1.3000,1.4000`
`:FETC:ARR ␣-1` *always fetches the last result* `1.4000`

Complies to standards: SCPI 1991.0, confirmed.

Format Subsystem

```
:FORMat
     [:DATA]                     ␣  ASCii REAL[, <Numeric value> | AUTO]
     :FIXed                      ␣  ON OFF
     :SREGister                  ␣  ASCii | BINary | HEXadecimal | OCTal
     :TINFormation[:STATe]       ␣  <Boolean>
```

:FORMat

PM6680B/85

⎵ «ASCii|REAL»

Response Data Type

Sets the format in which the result will be sent on the bus.

Parameters

ASCii will send the measurement result in ASCii form.<sign><mantissa value>E<sign><exponent value>

<sign> = + or –

<mantissa value> = 1 to 12 digits (depending on measuring resolution) plus one decimal point.

<exponent value> = 1 to 3 digits

REAL will send the result in binary IEEE Double Precision floating-point format in a block-data element. #18<8 bytes real>. The <8 bytes real> is a double precision binary floating-point response according to IEEE488.2/IEEE754. This means that the eight bytes are sent in the following order:

First byte: <sign><7 MSB of the exponent>

Second byte: <4 LSB of the exponent><4 MSB of the fraction>

Third through eight byte: <48 LSB of the fraction>

Returned format: ASC|REAL↵

***RST condition:** ASCii

Complies to standards: SCPI 1991.0, confirmed.

:FORMat

PM6681

⎵ «ASCii|REAL»[, <Numeric value> | AUTO]

Response Data Type

Sets the format in which the result will be sent on the bus.

This command is identical to the above described command for the PM6680B/85, except for the optional length parameter.

Parameters:

ASCii: The length controls the number of digits in the mantissa and may be set to values from 2 to 12 or AUTO.

AUTO: The length will be controlled by the resolution of each measurement result. Auto will be ignored when `:INTernal:FORMat` ⎵ PACKed or `:DISPlay:ENABled` ⎵ OFF is selected.

REAL: The length parameter is ignored, 'reals' are always output in 8 byte format.

Returned format: ASC|REAL, <Numeric value> | AUTO↵

***RST condition:** ASCii, AUTO

See also: :FORMat :TINFormation command

Complies to standards: SCPI 1991.0, confirmed.

PM6680B/81/85

:FORMat :FIXed

␣ <Boolean>

Response Data Format

Sets the ASCii format to fixed. This results in the following response format:

<sign><mantissa value>E<sign><exponent value>

Where:

<sign> = +|–
<mantissa value> = 12 digits plus one decimal point.
<exponent value> = 3 digits

Parameters <Boolean> = (1 / ON | 0 / OFF)

The counter will add leading zeroes when the measurement resolution is less than 12 digits.

Returned format: 1|0 ↵

***RST condition:** OFF

PM6681

:FORMat :SREGister

␣ «ASCii | BINary | HEXadecimal | OCTal»

Data Type for Status Messages

This command selects the data type of the response to queries for any CONDition, EVENt and ENABle register. This includes the IEEE 488.2 status register queries.

Parameters:

ASCii	The data is transferred as ASCii bytes in NR1 format.
HEXadecimal	The data is encoded as non-decimal numeric, base 16, preceded by '#H' as specified in IEEE 488.2
OCTal	The data is encoded as non-decimal numeric base 8, preceded by '#Q' as specified in IEEE 488.2
BINary	The data is encoded as non-decimal numeric, base 2, preceded by '#B' as specified in IEEE 488.2

Returned format: ASCii | BINary | HEXadecimal | OCTal ↵

***RST condition:** ASCii

:FORMat :TINFormation

PM6681

␣ Boolean

Timestamping On/Off Timestamping;On/Off

This command turns on/off the time stamping of measurements. Time stamping is always done at the start of a measurement with a resolution of 125 ns, and is saved in the measurement buffer together with the measurement result.

The setting of this command will affect the output format of the MEASure, READ and FETCh queries.

For :FETCh:SCALar?, :READ:SCALar? and :MEASure:SCALar? the readout will consist of two values instead of one. The first will be the measured value and the next one will be the timestamp value.

In :FORMat ASCii mode, the result will be given as a floating-point number (NR3 format) followed by the timestamp in seconds in the NR2 format ddd.ddddddddd (12 digits). In :FORMat REAL mode, the result will be given as an eight-byte block containing the floating-point measured value, followed by a four-byte block containing the integer timestamp count, where each count represents 125 nanoseconds.

When doing readouts in array form, with :FETCh :ARRay?, :READ :ARRay?, or :MEASure :ARRay? , the response will consist of alternating measurement values and timestamp values, formatted the same way as for scalar readout. All values will be separated by commas.

Parameters <Boolean> = (1 / ON | 0 / OFF)

Returned format: 1|0 ↵

***RST condition:** OFF

Initiate Subsystem

```
:INITiate
    [:IMMediate ]
    :CONTinuous␣  ON | OFF
```

:INITiate

PM6680B/81/85

Initiate Measurement

The :INITιατε command initiates a measurement. Executing an :INITiate command changes the counter's trigger subsystem state from "idle-state" to "wait-for-bus-arm-state" (see Figure 6-15). The trigger subsystem will continue to the other states, depending on programming. With the *RST setting, the trigger subsystem will bypass all its states and make a measurement, then return to idle state. See also 'How to use the Trigger Subsystem' at the end of this chapter.

Complies to standards: SCPI 1991.0, confirmed.

:INITiate :CONTinuous

PM6680B/81/85

␣ <Boolean>

Continuously Initiated

The trigger system could continuously be initiated with this command. When Continuous is OFF, the trigger system remains in the "idle-state" until Continuous is set to ON or the :INITiate is received. When Continuous is set to ON, the completion of a measurement cycle immediately starts a new trigger cycle without entering the "idle-state", i.e., the counter is continuously measuring and storing response data.

Returned format: <Boolean>↵

***RST condition:** OFF

Complies to standards: SCPI 1991.0, confirmed.

Input Subsystems

■ INPUT A

```
:INPut[1]
        :ATTenuation                      ⌴ <Numeric value>|MIN|MAX  (1|10)     (Not PM6685)
        :COUPling                         ⌴ AC|DC                              (Not PM6685)
        :IMPedance                        ⌴ <Numeric value>|MIN|MAX
        [:EVENt]
               :HYSTeresis                ⌴ «<Decimal data>|MAX |MIN»          (Only PM6685)
                              :AUTO       ⌴ ON|OFF|ONCE                        (Only PM6685)
               :LEVel                     ⌴ <Numeric value>|MIN|MAX
                              :AUTO       ⌴ ON|OFF|ONCE
        :SLOPe                            ⌴ POS|NEG
        :FILTer
               [:LPASs]
                              [:STATe]    ⌴ ON|OFF
```

■ INPUT B (Not PM6685)

```
:INPut2
        :ATTenuation                      ⌴ <Numeric value>|MIN|MAX  (1|10)
        :COUPling                         ⌴ AC|DC
        :IMPedance                        ⌴ <Numeric value>|MIN|MAX
        [:EVENt]
               :LEVel                     ⌴ <Numeric value>|MIN|MAX
                              :AUTO       ⌴ ON|OFF|ONCE
        :SLOPe                            ⌴ POS|NEG
        :COMMon                           ⌴ ON|OFF
```

■ INPUT E

```
:INPut4
        [:EVENt]
               :SLOPe                     ⌴ POS|NEG
```

:INPut«[1]|2» :ATTenuation

PM6680B PM6681

⌣ «<Numeric value>|MAX|MIN»

Attenuation

Attenuates the input signal with 1 or 10. The attenuation is automatically set if the input level is set to AUTO.

Parameters:

<Numeric values> ® 5, and MIN gives attenuation 1.
<Numeric values> > 5, and MAX gives attenuation 10.

Returned format:

1.00000000000E+000|1.00000000000E+001 ↲

Example for Input A (1)

SEND→ `:INP:ATT ⌣ 10`

Example for Input B (2)

SEND→ `:INP2:ATT ⌣ 10`

***RST condition** Input A (1) and Input B (2): 1 (but set by autotrigger since AUTO is on after `*RST`. (`:INP:LEV:AUTO ⌣ON`).

Complies to standards: SCPI 1991.0, confirmed.

:INPut«[1]|2» :COUPling

PM6680B PM6681

⌣ «AC|DC»

AC/DC Coupling

Selects AC coupling (normally used for frequency measurements), or DC coupling (normally used for time measurements).

Returned format: AC|DC↲

Example for Input A (1)

SEND→ `:INP:COUP ⌣ DC`

Example for Input B (2)

SEND→ `:INP2:COUP ⌣ AC`

***RST condition**

Input A (1): AC

Input B (2): DC

Complies to standards: SCPI 1991.0, confirmed.

PM6680B/81/85

:INPut :FILTer

⌣ <Boolean>

Low Pass Filter

Switches on or off the low pass filter on input 1 (A). It has a cutoff frequency of 100 kHz.

Parameters:

<Boolean> is (1 / ON | 0 / OFF)

Returned format: 1|0↵

***RST condition** OFF

Complies to standards: SCPI 1991.0, confirmed.

PM6685

:INPut :HYSTeresis

⌣ «<Decimal data>|MAX |MIN»

Sensitivity

The sensitivity setting on the front panel is called HYSTeresis from the bus. The range is 27.12 mV to 75.4 V. This setting has no effect unless autosensitivity is turned off, see the following page.

Note that the sensitivity setting is coupled with the hysteresis setting according to the formula:

$$\frac{\textit{Trigger Level} + \textit{Hysteresis}}{2} < 37.7057\ldots$$

Parameters: <Decimal data> is a number between 27.12E-3 and 75.4.

MAX gives +75.4 V, MIN gives +2.7 mV

When using MAX as data, the counter always tries to set the hysteresis to +75.4 V. Unless the Trigger level is set to 0, this setting is impossible, and the counter will return an error message.

Returned format: <Decimal data>

Example:

SEND→ `:INP:HYST ⌣ 0.5;HYST:AUTO ⌣ 0`

This example sets the sensitivity to 0.5 V and switches off autosensitivity.

***RST condition** 0.65 V (but controlled by Autotrigger since AUTO is on after `*RST`)

:INPut :HYSTeresis :AUTO

PM6685

␣ «<Boolean>|ONCE»

Auto Sensitivity

AUTO from the front panel turns on both auto sensitivity (hysteresis) and auto waveform compensation(trigger level). From the bus there are two commands, one for auto hysteresis and one for auto trigger level. However, the function of these commands are identical. Both commands turn on/off the hysteresis and the trigger level simultaneously.

The auto sensitivity function normally sets the hysteresis to 33% of Vpp. However, two exceptions exists: Pulse Width and Duty Cycle, where the hysteresis is set to min.

If you have a stable amplitude, use the :AUTO ␣ ONCE, and the autotrigger will determine sensitivity once and then set fixed levels.

Parameters <Boolean> = (1/ON | 0/OFF)

ONCE means that AUTO first switches ON to check the signal. After determining suitable sensitivity and trigger level setting, it programs these values as if they where manually set. It ends by switching off AUTO. Using ONCE instead of AUTO ON improves measuring speed.

Returned format: 1|0↵

Example:

SEND→ `:INP:HYST:AUTO ␣ OFF`

This example switches off AUTO, enabling manual sensitivity and trigger level setting.

***RST condition** ON

PM6680B/81/85

:INPut«[1]|2» :IMPedance
␣ «<Decimal data>|MAX|MIN»

Input Impedance

The impedance can be set to 50 Ω or 1 MΩ.

Parameters

MIN or <Decimal data> that rounds off to 50 or less, sets the input impedance to 50

MAX or <Decimal data> that rounds off to 1001 or more, sets the impedance to 1 MΩ.

Returned format:

5.00000000000E+001|1.00000000000E+6↵

Example for Input A (1)

SEND→ `:INP:IMP ␣ 50`

Sets the input A impedance to 50 Ω.

Example for Input B (2) (Only for PM6680B/81)

SEND→ `:INP2:IMP ␣ 50`

Sets the input B impedance to 50 Ω.

***RST condition** 1 MΩ

Complies to standards: SCPI 1991.0, confirmed.

PM6680B PM6681

:INPut«[1]|2» :LEVel
␣ «<Decimal data>|MAX|MIN»

Fixed Trigger Level

Input A and input B can be individually set to autotrigger or to fixed trigger levels of between –5 V and +5 V in steps of 0.02 V (1.25mV for PM6681). If the attenuator is set to 10X, the range is –50 V and +50 V in 0.2 V(12.5mV steps).

For autotrigger, see the following page.

Parameters: <Decimal data> is a number between –5 V and +5 V if att=1X and between –50 V and +50 V if att=10X.

MAX gives +50 V and MIN gives –50 V

When using MAX and MIN as data, the counter always tries to set the trigger level to +50 V and –50 V. If the attenuator is set to 1X, it is impossible to set this trigger level, and the counter will return an error message.

Returned format: <Decimal data>↵

Example for Input A (1)

SEND→ `:INP:LEV ␣ 0.01`

Example for Input B (2)

SEND→ `:INP2:LEV ␣ 2.0`

***RST condition** 0 (but controlled by Autotrigger since AUTO is on after `*RST`)

:INPut :LEVel

⌴ «<Decimal data>|MAX|MIN»

PM6685

Waveform compensation

The three-position waveform compensation on the front panel is not available from the bus. Instead, you can set the trigger level, that is, the level on which the hysteresis band is centered. How to set the trigger level depends on the duty cycle and the peak-to-peak voltage of the signal.

Trigger level = V_{pp} * (0.5 - *Duty factor*)

This setting has no effect unless autosensitivity is turned off, see the following page.

Parameters

<Decimal data> is a number between approximately –37.7 V and +37.7 V.

MAX gives +37.7 ... V and MIN gives –37.7... V

Note that the ⌴ :INP:LEV command is coupled with the `:INP:HYST` command. See page 9-45.

Returned format: <Decimal data>↲

Example:

SEND→ `:INP:LEV ⌴ 3.75;LEV:AUTO ⌴ 0`

This example sets the trigger level to 3.75 V and switches off auto trigger level.

***RST condition** 0 (but controlled by Autotrigger since AUTO is on after `*RST`)

:INPut :LEVel :AUTO

␣ «<Boolean>|ONCE»

Autotrigger

If set to AUTO, the counter automatically controls both the trigger level and the attenuation[1]. If you have a stable amplitude, use the `:AUTO` ␣ ONCE, and the autotrigger will determine the trigger level once and then set a fixed level.

From the bus, input A and input B are always set to autotrigger individually.

Parameters:

<Boolean> = (1/ON | 0/OFF)

ONCE means that the autotrigger switches on, checks the signal, stores the trigger levels as manually set levels, and then switches off auto. This improves measuring speed.

Example for Input A (1)

SEND→ `:INP:LEV:AUTO ␣ OFF`

Example for Input B (2)

SEND→ `:INP2:LEV:AUTO ␣ ON`

Returned format: 1|0↵

***RST condition** ON

[1] The autotrigger function normally sets the trigger levels to 50 % of the signal amplitude. Two exceptions exists however:
Rise/Fall time measurements: Here the input 1 (A) trigger level is set to 10% and the Input 2 (B) trigger level is set to 90% of the amplitude.
Variable Hysteresis mode (channel 7): The input 1 (A) trigger level is set to 75% and the Input 2 (B) trigger level is set to 25% of the amplitude

:INPut :LEVel :AUTO

PM6685

⎵ «<Boolean>|ONCE»

Autotrigger

:INPut:AUTO?

If auto is on, the counter automatically controls the trigger level[1] and the hysteresis. If you have a stable amplitude, use the `:AUTO ONCE`, and auto will determine the trigger level once, and then set fixed levels.

Parameters

<Boolean> = (1/ON | 0/OFF)

ONCE means that AUTO first switches ON to check the signal. After determining suitable sensitivity and trigger level setting, it programs these values as if they where manually set. It ends by switching off AUTO.

Using ONCE instead of AUTO ON improves measuring speed.

Returned format: 1|0↵

Example:

SEND→ `:INP;LEV:AUTO ⎵ OFF`

This example switches off AUTO, enabling the programmed trigger level setting.

***RST condition** ON

[1] The autotrigger measure peak to peak level, and sets the lower level of the hysteresis band to 33%, and the upper level to 66% of the value, (for pulse and duty factor measurement, both levels are set to 50%).

PM6680B/81/85

:INPut«[1]|2|4» :SLOPe ␣ «POS|NEG»

Trigger Slope

Selects if the counter should trigger on a positive or a negative transition. Selecting negative slope is useful when measuring negative pulse width and negative duty cycle.

When you select negative slope, the counter always uses the non-prescaled mode, limiting the maximum input frequency to 160 MHz. This can be useful when you want to make fast frequency measurements: Using positive slope, the counter needs two input cycles to make a SINGLE frequency measurement, but when set to negative slope, only one input cycle is required.

Returned format: POS | NEG ↵

Example for Input A (1)

```
SEND→ :INP:SLOP ␣ POS
```

Example for Input B (2) **(Only for PM6680B/81)**

```
SEND→ :INP2:SLOP ␣ NEG
```

Example for Input E (4)

```
SEND→ :INP4:SLOP ␣ NEG
```

***RST condition** POS

Complies to standards: SCPI 1991.0, confirmed.

PM6680B PM6681

:INPut2:COMMon ON|OFF

When on, the signal on input A is fed both to Channel 1 and Channel 2. The interconnection is made before the filter on input A.

Parameters

<Boolean> = (1/ON | 0/OFF)

ON means that the signal on input A is fed both to Channel 1 and Channel 2. The input signal on input B is not used in the measurement. But the signal on input B is terminated by the input impedance of the counter (50Ω or 1MΩ).

OFF means that inputs A and B works separated from each other.

Returned format: 1|0↵

Example:

```
SEND→ :INP2:COMM ON
```

This example switches on common, feeding the same signal to both channel 1 and channel 2.

***RST condition** OFF

This page is intentionally left blank.

Measurement Function

Set up the Instrument, Perform Measurement, and Read Data

```
   :MEASure
          [:SCALar]<Measuring Function>?
 [<Parameters>][,(<Channels>)]
          :ARRay<Measuring Function>?              (<Array Size>)[,<Parameters>][,(<Channels>)]
          :MEMory?
 [<N>]
          :MEMory<N>?
```

The array size for :MEASure *and* :CONFigure, *and the channels, are expression data that must be in parentheses ( ).*

The default channels, which the counter uses when you omit the channels in the command, are printed in italics in the channel list on the following pages.

If you want to check what function and channels the counter is currently using, send :CONF?
This query gives the same answer as :FUNC? *in the SENSe subsystem*

```
:MEASure|:CONFigure
    [:VOLTage]
        [:SCALar]
            :FREQuency
                [:CW]?              [_   [<expected value>[,<resolu-
                                         tion>],][(@1|@2|@3|@4|@5|@6|@7)]]
                :RATio?         [_   [<exp. value>[,<resol.>],][(@1|@2|@3|@4),(@1|@2|@3|@4)]]
                :BURSt?         [_   [<exp. value>[,<resol.>],][(@1|@2|@3|@4|@5|@6|@7)]]
                :PRF?           [_   [<exp. value>[,<resol.>],][(@1|@2|@3|@4|@5|@6|@7)]]
            :PERiod?            [_   [<exp. value>[,<resol.>],][(@1|@2|@3|@4|@5|@6|@7)]]
            :TINTerval?         [_   [<exp. value>[,<resol.>],][(@1|@2|@4),(@1|@2|@4)]]
            :PHASe?             [_   [<exp. value>[,<resol.>],][(@1|@2),(@1|@2)]]
            :NWIDth?            [_   [<exp. value>[,<resol.>],][(@1|@2|@4)]]
            :PWIDth?            [_   [<exp. value>[,<resol.>],][(@1|@2|@4)]]
            :DCYCle|PDUTycycle?  [_    [<exp. value>[,<resol.>],][(@1|@2|@4)]]
            :NDUTycycle?        [_   [<exp. value>[,<resol.>],][(@1|@2|@4)]]
            :RISE:TIME?[_    [<lower thresh.>[,<upper thresh.>[,<exp. value>[,<resol.>]]],][(@1|@2)]]
            :FALL:TIME?[_    [<lower thresh.>[,<upper thresh.>[,<exp. value>[,<resol.>]]],][(@1|@2)]]
            :MAXimum?  _        [_   (@1|@2)]
            :MINimum?           [_   (@1|@2)]
            :PTPeak?            [_   (@1|@2)]
            :TOTalize           _

                :GATed?[            [_(@1|@2|@4),(@1|@2|@4)]
                :TIMed?         [_   [<Time for gate open>,][ (@1|@2|@4),(@1|@2|@4)]]
                :ACCumulated?[_   [<Time for gate open>,][ (@1|@2|@4),(@1|@2|@4)]]
                :SSTop?         [_   (@1|@2|@4),(@1|@2|@4)]
                [:CONTinuous*][_   [(@1|@2|@4),(@1|@2|@4)]
        :ARRay
            :FREQuency
                [:CW]?     _   (<Size>)[,[<expected value>[,<resolution>],] [(@1|@2|@3|@4|@5|@6|@7)]]
                :RATio?    _   (<Size>)[,[<exp. value>[,<resol.>],][(@1|@2|@3|@4),(@1|@2|@3|@4)]]
                :BURSt?    _   (<Size>)[,<exp. value>[,<resol.>],][(@1|@2|@3|@4)]]
                :PRF?      _   (<Size>)[,<exp. value>[,<resol.>],][(@1|@2|@3|@4)]]
            :PERiod?       _   (<Size>)[,<exp. value>[,<resol.>],][(@1|@2|@3|@4)]]
            :TINTerval?    _   (<Size>)[,[<exp. value>[,<resol.>],][(@1|@2|@4),(@1|@2|@4)]]
            :PHASe?        _   (<Size>)[,[<exp. value>[,<resol.>],][(@1|@2),(@1|@2)]]
            :NWIDth?       _   (<Size>)[,[<exp. value>[,<resol.>],][(@1|@2|@4)]]
            :PWIDth?       _   (<Size>)[,[<exp. value>[,<resol.>],][(@1|@2|@4)]]
            :DCYCle|PDUTycycle?      (<Size>)[,[<exp. value>[,<resol.>],][(@1|@2|@4)]]
            :NDUTycycle?  _   (<Size>)[,[<exp. value>[,<resol.>],][(@1|@2|@4)]]
            :RISE:TIME?   _   (<Size>)[,[<lower thr..>[,<upper thr.>[,<exp. value>[,<resol.>]]],][(@1|@2)]]
            :FALL:TIME?   _   (<Size>)[,[<lower thr.>[,<upper thr.>[,<exp. value>[,<resol.>]]],][(@1|@2)]]
            :MAXimum?     _   (<Size>)[,[(@1|@2)]
            :MINimum?     _   (<Size>)[,[(@1|@2)]
            :PTPeak?      _   (<Size>)[,[(@1|@2)]
            :TOTalize?    _

                :GATed?  _   (<Size>)[,[(@1|@2|@4),(@1|@2|@4)]]
                :TIMed?  _   (<Size>)[,[<Time for gate open>,][(@1|@2|@4),(@1|@2|@4)]]
                :ACCumulated?   (<Size>)[,[<Time for gate open>,][(@1|@2|@4),(@1|@2|@4)]]
                :SSTop?      (<Size>)[,[(@1|@2|@4),(@1|@2|@4)]]
                 [:CONTinuous*]     _   (<Size>)[,[(@1|@2|@4),(@1|@2|@4)]]
```

```
:MEASure|:CONFigure
 [:VOLTage]
      [:SCALar]
          :FREQuency
                      [:CW]?                    [␣  [<expected value>[,<resolu-
                                                     tion>]][,(@1|@3|@4|@5|@6)]]
                      :RATio?           [␣  [<exp. value>[,<resol.>]][,(@1|@3|@4|@5|@6),
                                            (@1|@3|@4|@5|@6)]]
                      :BURSt?           [␣  [<expected value>[,<resolution>]][,(@1|@3|@4)]]
                      :PRF?                     [␣  [<expected value>[,<resolution>]][,(@1|@3|@4)]]
          :PERiod?                      [␣  [<expected value>[,<resolution>]][,(@1|@3|@4)]]
          :NWIDth?                      [␣  [<threshold>[,(@1|@4)]]
          :PWIDth?                      [␣  [<threshold>[,(@1|@4)]]
          :PDUTycycle|DCYCle?[␣   [<threshold>][,(@1|@4)]]
          :NDUTycycle?                          [␣  [<threshold>][,(@1|@4)]]
          :TOTalize*
                      [:CONTinuous]*[␣   (@0|@1|@4)[,(@0|@1|@4)]

      [:ARRay]
          :FREQuency
                      [:CW]?␣   (<Size>)[,[<expected value>[,<resolution>]][,(@1|@3|@4|@5|@6)]]
                      :RATio?␣   (<Size>)[␣,[<exp. value>[,<resol.>]][,(@1|@3|@4),(@1|@3|@4)]]
                      :BURSt?␣   (<Size>)[␣,[<expected value>[,<resolution>]][,(@1|@3|@4)]]
                      :PRF?                 ␣   (<Size>)[,[<expected value>[,<resolution>]] [,(@1|@3|@4)]]
          :PERiod?     ␣   (<Size>)[,[<expected value>[,<resolution>]][,(@1|@3|@4)]]
          :NWIDth?     ␣   (<Size>)[,[<threshold>[,(@1|@4)]]
          :PWIDth?     ␣   (<Size>)[,[<threshold>[,(@1|@4)]]
          :PDUTycycle|DCYCle?␣   (<Size>)[,[<threshold>][,(@1|@4)]]
          :NDUTycycle?                      ␣   (<Size>)[,[<threshold>][,(@1|@4)]]
          :TOTalize*
                      [:CONTinuous*]␣   (<Size>)[,[(@0|@1|@4)[,(@0|@1|@4)]]
```

* Only for :CONFigure

(@0) means that the input is disabled (Only PM6685)
(@1) means input A
(@2) means input B (Not available on PM6685)
(@3) means input C (HF-input option)
(@4) means input E (Rear panel arming input)
(@5) means input A prescaled by 2
(@6) means the internal reference
(@7) means input A with the variable hysteresis mode (Only PM6680B and PM6681)

:MEASure :<Measuring Function>?

PM6680B/81/85

[⌴ [<parameters>][,(<channels>)]]

Make one measurement

The measure query makes a complete measurement, including configuration and readout of data. Use measure when you can accept the generic measurement without fine tuning.

When a CONFigure *command or* MEASure? *query is issued, all counter settings are set to the* *RST *settings., except those specified as <parameters> and <channels> in the* CONFigure *command or* MEASure? *query.*

You cannot use the `:MEASure?` query for `:TOTalize:CONTinuous`, since this function measures without stopping (continuously forever).

The `:MEASure?` query is a compound query identical to:
`:ABORt;:CONFigure:`<Meas_func>`;:READ?`

Parameters:

<Measuring Function>, <Parameters> and <Channels> are defined on page 9-54. You may omit <parameters> and <Channels>, which are then set to default.

Returned format: <data>↵

Where: The format of the returned data is determined by the format commands: :FORMat and :FORMat:FIXed.

Example:

SEND→ `:MEAS:FREQ? ⌴ (@3)`
READ← `1.78112526833E+009`

This example measures the frequency on the C-input and outputs the result to the controller.

Type of command: Aborts all previous measurement commands if *WAI is not used.

See also: 'Explanations of the Measuring Functions' starting on page 9-59.

Complies to standards: SCPI 1991.0, confirmed.

80B/81/85 :MEASure :ARRay :<Measuring Function>?

[_ (<array size>)[,[<parameters>] [,(<channels>)]]

Make an array of measurements

The :MEASure:ARRay query differs from the :MEASure query in that it performs the number of measurements you decide in the <array size> and sends all the measuring results in one string to the controller.

☞ *The array size for* :MEASure *and* :CONFigure, *and the channels, are expression data that must be in parentheses ( ).*

The :MEASure:ARRay query is a compound query identical to:
:ABORt;:CONFigure:ARRay:<Meas-func> _ (<array-size>); :READ:ARRay? _ (<array-size>)

Parameters:

<array size> sets the number of measurements in the array.

Returned format:

<Measuring result>{[,<measuring result>]}↵

Example:

SEND→ :MEAS:ARR:FREQ? _ (10)

Ten measuring results will be returned.

Type of command:

Aborts all previous measurement commands if not *WAI is used, see page 9-132.

Complies to standards: SCPI 1991.0, confirmed.

:MEASure:MEMory<N>? PM6681

Memory Recall, Measure and Fetch Result

Use this command when you want to measure *several parameters fast.*

`:MEAS:MEM1?` recalls the contents of memory 1 and reads out the result, `:MEAS:MEM2?` recalls the contents of memory two and reads out the result etc.

The equivalent command sequence is `*RCL1;READ?`

The allowed range for <N> is 1 to 9. Use the somewhat slower `:MEAS:MEMory? N` command described below if you must use memories 10 to 19.

	TIMING	
	Data Format	
Command	ASCii	REAL
`:MEAS:MEM1?`	7.9 ms	6.7 ms
`:MEAS:MEM? 1`	9.1 ms	8.0 ms
`*RCL 1;READ?`	10.1 ms	8.9 ms

Returned format:

<measurement result>↵

Complies to standards: SCPI 1991.0, confirmed

:MEASure:MEMory? PM6681

␣ <N>

Memory Recall, Measure and Fetch Result

Same as above command but somewhat slower. Allows use of all memories (1 to 19).

Example: `:MEAS:MEM ␣ 13`

This example recalls the instrument setting in memory number 13, makes a measurement, and fetches the result.

Complies to standards: SCPI 1991.0, confirmed

EXPLANATIONS OF THE MEASURING FUNCTIONS

This sub-chapter explains the various measurements that can be done with `:MEASure` and `:CONFigure;:READ`. Only the queries for single measurements using the measure command are given here, but all of the information is also valid for the `:CONFigure` command and for both scalar (single) and array measurements.

PM6680B/81/85

:MEASure_«:DCYCle/:PDUTycycle»

[˽ [<threshold>] [,(@«1|2|4|6»)]]

Positive Duty Cycle

Traditional duty cycle measurement is performed. That is, the ratio between the on time and the off time of the input pulse is measured.

Parameters

<threshold> parameter sets the trigger levels in volts. If omitted, the auto trigger level is set to 50 percent of the signal.

(@«1|2|4|6») is the channel to measure on:

(@1) means input A

(@2) means input B (Only PM6680B and PM6681)

(@4) means input E (Rear panel arming input)

(@6) means the internal reference

If you omit the channel, the instrument measures on input A (@1).

Example:

SEND→ `:MEAS:PDUT?`

READ← `+5.097555E-001`

In this example, the duty cycle is 50.97%

Complies to standards: SCPI 1991.0, confirmed.

:MEASure :FREQuency?

PM6680B/81/85

[⌴ [<expected value>[,<resolution>]] [,<(@«1|2|3|4|5|6|7»)>]]

Frequency

Traditional frequency measurements. The counter uses the <expected value> and <resolution> to calculate the Measurement Time `(:SENSe:ACQuisition:APERture)`.

Example:

SEND→ `:MEAS:FREQ? ⌴ (@3)`

READ← `1.78112526833E+009`

This example measures the frequency at input C.

The channel is expression data and it must be in parentheses ( ).

Parameters:

<expected value> is the expected frequency,

<resolution> is the required resolution.

<(@«1|3|4|5|6|7»)> is the channel to measure on:

(@1) means input A[1]
(@2) means input B (Only PM6680B and PM6681)
(@3) means input C (HF-input option)
(@4) means input E (Rear panel arming input)

(@5) means input A prescaled by 2
(@6) means the internal reference
@7) means input A with the variable hysteresis mode (Only PM6680B and PM6681)

If you omit the channel, the instrument measures on input A (@1).

1 The A input is always prescaled by 2 when measuring Frequency A and prescaled by 1 for all other functions.

Complies to standards: SCPI 1991.0, confirmed.

:MEASure :FREQuency :BURSt?

[␣ [<expected value>[,<resolution>]] [,<(@«1|2|3|4|5|6|7»)>]]

Burst Carrier Frequency

Measures the carrier frequency of a burst. The burst duration must be less than 50% of the pulse repetition frequency (PRF).

How to measure bursts is described in detail in the Operators Manual.

The counter uses the <expected value> and <resolution> to select a Measurement Time (`[:SENSe]:ACQuisition:APERture`), and then sets the sync delay (`[:SENSe]:SDELay`) to 1.5 * Measurement Time.

Parameters:

<expected value> is the expected carrier frequency,
<resolution> is the required resolution, e.g., 1 gives 1Hz resolution.

<(@«1|2|3|4|5|6|7»)> is the channel to measure on:
(@1) means input A
(@2) means input B (Only PM6680B and PM6681)
(@3) means input C (HF-input option)
(@4) means input E (Rear panel arming input)
(@5) means input A prescaled by 2
(@6) means the internal reference
(@7) means input A with the variable hysteresis mode (Only PM6680B/81)

If you omit the channel, the instrument measures on input A (@1).

Complies to standards: SCPI 1992.0, confirmed.

:MEASure :FREQuency :PRF?

PM6680B/81/85

[⌴ [<exp. val.>[,<res.>]][,<(@«1|2|3|4|5|6|7»)>]]

Pulse Repetition Frequency

Measures the PRF (Pulse Repetition Frequency) of a burst signal.The burst duration must be less than 50% of the pulse repetition frequency (PRF).

It is better to set up the measurement with the [:SENS]:FUNC ":FREQ:PRF" *command when measuring pulse repetition frequency. This command will allow you to set a suitable sync delay with the* [:SENSe]:Sync:DELay *command.*

How to measure bursts is described in detail in the Operators Manual.

Parameters: <exp. val.> is the expected PRF,

<res.> is the required resolution.

<(@«1|3|4|5|6»)> is the channel to measure on:

(@1) means input A
(@2) means input B (Only PM6680B and PM6681)
(@3) means input C (HF-input option)
(@4) means input E (Rear panel arming input)
(@5) means input A prescaled by 2
(@6) means the internal reference
(@7) means input A with the variable hysteresis mode (Only PM6680B/81)

If you omit the channel, the instrument measures on input A (@1).

The <expected value> and <resolution> are used to calculate the Measurement Time (`[:SENSe]:ACQuisition:APERture`). The Sync. Delay is always 10 µs (default value)

Complies to standards: SCPI 1992.0, confirmed.

PM6680B PM6681

:MEASure :FALL :TIME?

[␣[<lower threshold> [,<upper threshold>[,<expected value>[,<resolution>]]]] [,(@1)]]

Fall-time

The transition time from 90% to 10% of the signal amplitude is measured.

The measurement is always a single measurement and the Auto-trigger is always on, setting the trigger levels to 90% and 10 % of the amplitude. If you need an average transition time measurement, or other trigger levels, use the `:SENSe` subsystem and manually set trigger levels instead.

Parameters:

<lower threshold>, <upper threshold>, <expected value> and <resolution are all ignored by the counter

<(@1)> is the channel to measure on, i.e., input A

Complies to standards: SCPI 1991.0, confirmed.

PM6680B/81/85

:MEASure :FREQuency :RATio?

[␣[<expected value> [,<resolution>]][,<(@«1|2|3|4|5|6»)>,<(@«1|2|3|4|5|6»)>]]

Frequency Ratio

Frequency ratio measurements between two inputs.

Example:

SEND→ `:MEAS:FREQ:RAT? ␣ (@1),(@3)`

READ← `2.345625764333E+000`

This example measures the ratio between input A and input C.

The channel is expression data and it must be in parentheses ( ).

Parameters: <expected value> and <resolution> are ignored

<(@«1|2|3|4|5|6»)>,<(@«1|2|3|4|5|6»)> is the channels to measure on:

(@1) means input A

(@2) means input B (Only PM6680B and PM6681)

(@3) means input C (HF-input option)

(@4) means input E (Rear panel arming input)

(@5) means input A prescaled by 2

(@6) means the internal reference

If you omit the channel, the instrument measures between input A and input E.

Complies to standards: SCPI 1991.0, confirmed.

:MEASure [:VOLT] :MAXimum? [⌴(«@1|@2»)]

PM6680B PM6681

Positive Peak Voltage

This command measures the positive peak voltage with the input DC coupled.

Parameters:

(«@1|@2») is the channel to measure on

(@1) means input A

(@2) means input B

Complies to standards: SCPI 1991.0, confirmed.

:MEASure [:VOLT] :MINimum? [⌴(«@1|@2»)]

PM6680B PM6681

Negative Peak Voltage

This command measures the negative peak voltage with the input DC coupled

Parameters:

(«@1|@2») is the channel to measure on

(@1) means input A

(@2) means input B

Complies to standards: SCPI 1991.0, confirmed.

PM6680B/81/85

:MEASure :NWIDth?

[⌴ [<threshold>] [,<(@«1|2|4|6»)>]]

Negative Pulse Width

A negative pulse width measurement is performed.

This is always a single measurement. If you need an average pulse width measurement, use the `:SENSe` subsystem instead.

Parameters

<threshold> parameter sets the trigger levels in volts. If omitted, the auto trigger level is set to 50 percent of the signal.

<(@«1|2|4|6»)> is the channel to measure on:
(@1) means input A
(@2) means input B (Only PM6680B and PM6681)
(@4) means input E (Rear panel arming input)
(@6) means the internal reference

If you omit the channel, the instrument measures on input A.

Complies to standards: SCPI 1991.0, confirmed.

PM6680B/81/85

:MEASure :PWIDth?

[⌴ [<threshold>] [,<(@«1|2|4|6»)>]]

Positive Pulse Width

A positive pulse width measurement is performed.

This is always a single measurement. If you need an average pulse width measurement, use the `:SENSe` subsystem instead.

Parameters

<threshold> parameter sets the trigger levels in volts. If omitted, the auto trigger level is set to 50 percent of the signal.

<(@«1|2|4|6»)> is the channel to measure on:
(@1) means input A
(@2) means input B (Only PM6680B and PM6681)
(@4) means input E (Rear panel arming input)
(@6) means the internal reference

If you omit the channel, the instrument measures on input A.

Complies to standards: SCPI 1991.0, confirmed.

:MEASure_«:PDUTycycle/ :DCYCle»? PM6680B/81/85

[␣ [<threshold>] [,(@«1|2|4|6»)]]

Positive duty cycle: Duty Factor

Traditional duty cycle measurement is performed. That is, the ratio between the on time and the off time of the input pulse is measured.

Parameters

<threshold> parameter sets the trigger levels in volts. If omitted, the auto trigger level is set to 50 percent of the signal.

(@«1|2|4|6») is the channel to measure on:

(@1) means input A

(@2) means input B (Only PM6680B and PM6681)

(@4) means input E (Rear panel arming input)

(@6) means the internal reference

If you omit the channel, the instrument measures on input A (@1).

Example:

SEND→ `MEAS:PDUT?`

READ← `+5.097555E-001`

In this example, the duty cycle is 50.97%

Complies to standards: SCPI 1991.0, confirmed.

:MEASure_«:NDUTycycle»? PM6680B/81/85

[␣ [<threshold>] [,(@«1|2|4|6»)]]

Negative duty cycle: Duty Factor

Traditional duty cycle measurement is performed. That is, the ratio between the on time and the off time of the input pulse is measured.

Parameters

<threshold> parameter sets the trigger levels in volts. If omitted, the auto trigger level is set to 50 percent of the signal.

(@«1|2|4|6») is the channel to measure on:

(@1) means input A

(@2) means input B (Only PM6680B and PM6681)

(@4) means input E (Rear panel arming input)

(@6) means the internal reference

If you omit the channel, the instrument measures on input A (@1).

Example:

SEND→ `MEAS:PDUT?`

READ← `+5.097555E-001`

In this example, the duty cycle is 50.97%

Complies to standards: SCPI 1991.0, confirmed.

PM6680B/81/85

:MEASure :PERiod?

[␣ [<expected value> [,<resolution>]][,<(@«1|2|3|4|5|6|7»)>]]

Period

A traditional period measurement is performed.

The <expected value> and <resolution> are used to calculate the Measurement Time (`[:SENSe]:ACQuisition:APERture`).

Parameters:

<expected value> is the expected Period,

<resolution> is the required resolution,

<(@«1|2|3|4|5|6»)> is the channel to measure on:
(@1) means input A
(@2) means input B (Only PM6680B and PM6681)
(@3) means input C (HF-input option)
(@4) means input E (Rear panel arming input)
(@5) means input A prescaled by 2
(@6) means the internal reference
(@7) means input A with the variable hysteresis mode (Only PM6680B)PM6681)

If you omit the channel, the instrument measures on input A (@1).

Complies to standards: SCPI 1991.0, confirmed.

PM6680B PM6681

:MEASure :PHASe?

[␣ [<expected value>[,<resolution>]] [,(@«1|2»),(@«1|2»)]]

Phase

A traditional PHASe measurement is performed.

Parameters:

<expected value> and <resolution> are ignored by the counter

The first (@«1|2») is the start channel and the second (@«1|2») is the stop channel
(@1) means input A
(@2) means input B

If you omit the channel, the instrument measures between input A and input B.

Complies to standards: SCPI 1991-0, approved.

:MEASure [:VOLT] :PTPeak?

PM6680B PM6681

[␣ (@«1|2»)].

Peak-to-Peak Voltage

This command make measures the peak-to-peak voltage with the input DC coupled.

Parameters:

(@«1|2») is the channel to measure on

(@1) means input A

(@2) means input B

Complies to standards: SCPI 1991.0, confirmed.

:MEASure :RISE :TIME?

PM6680B PM6681

[␣ [<lower threshold> [,<upper threshold>[,<expected value>[,<resolution>]]]] [,(@1)]

Rise-time

The transition time from 10% to 90% of the signal amplitude is measured.The measurement is always a single measurement and the Auto-trigger is always on, setting the trigger levels to 10% and 90 % of the amplitude. If you need an average transition time measurement or other trigger levels, use the :SENSe subsystem and manually set trigger levels instead.

Parameters:

<lower threshold>, <upper threshold>, <expected value> and <resolution are all ignored by the counter

<(@1)> is the channel to measure on, i.e., input A

Complies to standards: SCPI 1991.0, confirmed.

:MEASure :TINTerval?

⌣ (@«1|2|4»),(@«1|2|4»)]

Time-Interval

Traditional time-interval measurements are performed. The trigger levels are set automatically, and positive slope is used. The first channel in the channel list is the start channel, and the second is the stop channel.

Parameters:

The first (@«1|2|4») is the start channel and the second (@«1|2|4») is the stop channel

(@1) means input A

(@2) means input B

(@4) means input E (Rear panel arming input)

If you omit the channel, input A is the start channel, and input B is the stop channel.

:MEASure :TOTalize :ACCumulated? PM6680B PM6681

[␣ <time for gate open>][,(@«1|2|4|5|6») [,(@«1|2|4|5|6»)]]

Totalize X gated by Y, accumulated

The counter totalizes the pulses on the primary channel. The totalizing starts when the gate signal on the secondary channel goes on and stops when the gate signal goes to off. The polarity of on/off is controlled via the `:INPut:SLOPe` command of the gate channel. The result is the sum of counts in all the gate openings that occur during a preset time <time for gate open>.

If you use the `:CONFigure` command, you can select if the counter should count positive or negative transitions with the `:INPut:SLOPe` command of the measuring channel.

Parameters: <time for gate open> is the time you want the totalizing to proceed. Range PM6680B: is 0.8E–6, 1.6E–6, 3.2E–6, 6.4E–6, 12.8E–6, and 50E–6 to 400 s Range PM6681 and 80E–9, 160E–9, 320E–9, 640E–9, 1.28E–6, and 20E–6 to 400 s.

The first <(@«1|2|4|5|6»)> is the channel to measure on.

The second <(@«1|2|4|5|6»)> is the gate channel.

(@1) means input A
(@2) means input B
(@4) means input E (rear panel arming input)
(@5) means input A prescaled by 2
(@6) means the internal reference

If you omit the channels, the instrument measures on input A with input B as the gate channel.

***RST condition:**

Time for gate open = 10 ms (`[:SENSe]ACQuisition:APERture`*)*

PM6680B/81/85

:CONFigure :TOTalize :CONTinuous

[_ (@«1|2|4|6»)][,(@«1|2|4|6»)]

Totalize Manually

This is a count/totalize function controlled from the GPIB interface using the command `SENS:TOT:GATE_ON|OFF`.

The counter counts up for each event on the primary input channel. and down on the secondary channel. The result is the difference between the primary and secondary channel. In addition to selecting totalizing, the `:CONF:TOT:CONT` command also selects positive trigger slope. If you want to count negative slopes on input A, send `:INPut:SLOPe_NEG` after the `:CONF:TOT:CONT` command.

Parameters

(@«1|2|4|6») is the primary (adding)channel:
,(@«1|2|4|6») is the secondary (subtracting) channel:

(@1) means input A

(@2) means input B (not PM6685)

(@4) means input E (rear panel arming input)

(@6) means the internal reference

Selecting the same channel as both primary and secondary disables the secondary channel.

This measurement cannot be done as a `:MEASure`, *it must be done as a* `:CONFigure` *followed by* `:INIT:CONT_ON`, *gate control with* `:SENS:TOT:GATE «ON|OFF»` *and completed with a* `:FETCh:ARR?` *<array size>.*

Example:

SEND→ `:CONF:TOT;:INP:SLOPe neg`

This example sets up the counter to totalize the negative slopes on Input A and disable the secondary channel. (Same as (@1),(@1).)

***RST condition** (@1),(@2) for PM6680B and PM6681, (@1),(@1) for PM6685

Normal Program Sequence for Totalizing on A	
`CONF:TOT:CONT_(@1),(@1)`	Set up the counter for totalize on A
`INIT:CONT_ON`	Initiate the counter continuously
`TOT:GATE_ON`	Start totalizing
`FETC:ARR?_-1`	Read intermediate results without stopping the totalizing
`TOT:GATE_OFF`	Stop totalizing
`FETC:ARR?_-1`	Fetch the final result from the totalizing

The `:FETCh:ARR?` command can take both positive and negative data. Positive data, for instance 10, outputs the first 10 measurements in the counter output buffer. Negative data, for instance –10, outputs the last ten results.

Intermediate results When totalizing you often want to read the intermediate result without stopping the totalizing process. `:FETC:ARR?_-1` outputs such a result.

:MEASure :TOTalize :GATed?

[⌣ (@«1|2|4|5|6») [,(@«1|2|4|5|6»)]]

PM6680B PM6681

Totalize X gated by Y

The counter totalizes the pulses on the primary channel. The totalizing starts when the gate signal on the secondary channel goes on and stops when the gate signal goes to off. The polarity of on/off is controlled via the `:INPut:SLOPe` command of the gate signal.

Select if the counter should count positive or negative transitions with the `:INPut:SLOPe` command of the measuring channel.

Parameters

The first <(@«1|2|4|5|6»)> is the channel to measure on, the second one is the gate channel:

(@1) means input A

(@2) means input B

(@4) means input E (rear panel arming input)

(@5) means input A prescaled by 2

(@6) means the internal reference

If you omit the channels, the instrument measures on input A with input B as the gate channel.

:MEASure :TOTalize :SSTop?

[⌣ ,(@«1|2|4|5|6») [,(@«1|2|4|5|6»)]]

PM6680B PM6681

Totalize X start/stop by Y

The counter totalizes the pulses on the primary channel. The totalizing starts when the gate signal on the secondary channel goes on and stops the next time the gate signal goes on. The polarity of ON is controlled via the `:INPut:SLOPe` command of the Start /stop channel.

Select if the counter should count positive or negative transitions with the `:INPut:SLOPe` command of the measuring channel.

Parameters:

The first <(@«1|2|4|5|6»)> is the channel to measure on, and the second one is the start/stop channel:

(@1) means input A

(@2) means input B

(@4) means input E (rear panel arming input)

(@5) means input A prescaled by 2

(@6) means the internal reference

If you omit the channels, the instrument measures on input A with input B is the start/stop channel.

:MEASure :TOTalize :TIMed?

[␣ [<time for gate open> [,(@«1|2|4»)][,(@«1|2|4»)]]]

Totalize X-Y During a Preset Time

This is a count/totalize function during a predefined time. The start/stop signal is generated by the counter and set by <time for gate open>.

The counter counts up for each event on the X-channel and down for each event on the Y-channel. The result is the difference between the two channels.

If you only want to Totalize on X, you must disable Y by setting both X and Y to the same channel or disconnecting the signal from Y.

Totalize –Y MANUAL, negative totalizing, is possible if you physically disconnect the signal on the X input.

Select if the counter should count positive or negative transitions with the INPut:SLOPe command of the channels.

Parameters:

<time for gate open> is the time you want the totalizing to proceed. The range is the same as for Measurement Time.

The first <(@«1|2|4»)> is the channel that counts up, and the second one is the channel that counts down:

(@1) means input A
(@2) means input B
(@4) means input E (rear panel arming input)

If you omit the channels, the instrument counts up on input A and down on input B.

Example:

SEND→ `:MEAS:TOT:TIM? ␣ 1,(@1),(@1)`

In this example the counter totalises the pulses on Channel 1 for one second. Any signals on channel 2 and 4 are ignored.

***RST condition:**

Time for gate open = 10 ms (`[:SENSe]:ACQuisition:APERture`*)*

This page is intentionally left blank.

Memory Subsystem

```
:MEMory
    :DELete
                :MACRo␣   '<Macro name>'
    :FREE
    :SENSe?
    :NSTates?
    :MACRo?
```

Related Common Commands:

```
*DMC
*EMC
*GMC?
*LMC?
*LRN?
*PMC
*RCL
*RMC
*SAV
```

:MEMory :DELete :MACRo

PM6680B/81/85

␣'<Macro name>'

Delete one Macro

This command removes an individual MACRo[1].

Parameters

'<Macro name>' is the name of the macro you want to delete.

<Macro name> is String data that must be surrounded by quotation marks.

See also:

`*PMC`, if you want to delete all macros.

[1] The proposed IEEE488.2 command `*RMC` (Remove Macro command) also works on PM6685. It preforms exactly the same action as `:MEMory:DELete:MACRo`. Note however that this command is not yet (1993) a certified IEEE488.2 command.

:MEMory :FREE :SENSe?

PM6681

Memory Free for results

This command gives information of the free memory available for sense data (measuring results) in the counter.

Returned format:

<Data positions available>, <Data positions in use>↵

PM6680B/81/85

:MEMory :FREE :MACRo?

Memory Free for Macros

This command gives information of the free memory available for MACRos in the counter. If no macros are specified, 1160 bytes are available.

Returned format:

<Bytes available>, <Bytes used>↵

Complies to standards: SCPI 1991.0, confirmed.

PM6680B/81/85

:MEMory :NSTates?

Memory States

The Number of States query (only) requests the number of *SAV/ *RCL instrument setting memory states available in the counter. The counter responds with a value that is one greater than the maximum that can be sent as a parameter to the *SAV and *RCL commands. (States are numbered from 0 to max–1.)

Returned format:

<the number of states available>↵

Complies to standards: SCPI 1991.0, confirmed

This page is intentionally left blank.

Output Subsystem

```
:OUTPut
     [:STATe]    ⎵  ON | OFF
     :SCALe      ⎵  <Numeric value>
```

:OUTPut

PM6680B/81/85

␣ <Boolean>

Enable Analog Out

This command switches on/off the analog output. See also `:OUTput:SCALe` command on the next page.

Parameters

<Boolean> = (1/ON | 0/OFF)

Returned format: <1|0>↵

Example:

Send→ `:OUTP ␣ 1`

Switches on the analog output.

***RST condition:** OFF

Complies to standards: SCPI 1991.0 confirmed.

:OUTPut :SCALe

PM6680B/81/85

␣ < Decimal data >

Scaling Factor, Analog Output

This command sets the scaling factor for the analog output. The measurement result is scaled after math, if math is used.

If you want a full-scale output for a specific readout, the formula is:

$$\textit{Scaling factor} = \frac{1}{\textit{full scale value}}$$

Parameters

<Decimal data> is the scaling factor. The range is –1020 to +1020.

Returned format: < Decimal data>↵

Example:

If you want full scale output (5 V) for a reading of 0.00359,

$$\textit{Scaling factor} = \frac{0.00359}{5} = 0.000718$$

Send→ `:OUTP:SCAL ␣ 718E-6`

***RST condition:** 1

Read Function

Perform Measurement and Read Data

```
:READ
    [:SCALar]?
    :ARRay?␣  <Array Size>|MAX
```

:READ?

PM6680B/81/85

Read one Result

The read function performs new measurements and reads out a measuring result without reprogramming the counter. Using the `:READ?` query in conjunction with the :CONFigure command gives you a measure capability where you can fine tune the measurement.

If the counter is set up to do an array of measurements, `:READ?` makes all the measurements in the array, stores the results in the output buffer, and fetches the first measuring result. Use `FETCh?` to fetch other measuring results from the output buffer. The `:READ?` query is identical to `:ABORt;:INITiate;:FETCh?`

Returned format: <data>↵

The format of the returned data is determined by the format commands `:FORMat` and `FORMat:FIXed`.

Example:

SEND→ `:CONF:FREQ;:INP:FILT ␣ ON;:READ?`

This example configures the counter to make a standard frequency measurement with the 100 kHz filter on. The counter is triggered, and data from the measurement are read out with the `:READ?` query.

SEND→ `:READ?`

This makes a new measurement and fetches the result without changing the programming of the counter.

Type of command: Aborts all previous measurement commands if `*WAI` is not used.

Complies to standards: SCPI 1991.0, confirmed.

:READ:ARRay?

␣ «<array size for FETCh>|MAX»

Read an array of results

The `:READ:ARRay?` query differs from the `:READ?` query by reading out several results at once after making the number of measurements previously set up by `:CONFigure:ARRay` ␣ or ␣ `:MEASure:ARRAy?`.

The `:READ:ARRay?` query is identical to:
`:ABORt;:INITiate;:FETCh:ARRay?`_<array size for FETCh>

The <array size for FETCh> *does not tell* :READ *to make that many measurements, only to fetch that many results.* :CONF:ARR, ␣ :MEAS:ARR, :ARM:LAY1:COUN or :TRIG:LAY1:COUN sets the number of measurements.

Parameters:

<array size for FETCh> sets the number of measuring results in the array. This size must be equal or less than the number of measurements specified with `:CONFigure`.

MAX means that all the results in the output buffer will be fetched.

Returned format: <data>[,<data>]↵

The format of the returned data is determined by the format commands `:FORMat` and `:FORMat:FIXed`.

SEND→ `:ARM:COUN ␣ 10;:READ:ARR? ␣ 5`

This example configures the counter to make an array of 10 standard measurements. The counter is triggered and data from the first five measurements are read out with the `:READ?` query.

Type of command: Aborts all previous measurement commands if `*WAI` is not used.

Complies to standards: SCPI 1991.0, confirmed.

This page is intentionally left blank.

Sense Command Subsystem

■ Sense Subsystem command tree for PM6680B and PM6681

```
[:SENSe]
:ACQuisition
        :APERture                  ␣  <meas time> | MIN | MAX
        :HOFF
              [:STATe]             ␣  ON | OFF
              :ECOunt              ␣  <hold off event count value> | MIN | MAX
              :MODE                ␣  TIME | EVENt
              :TIME                ␣  <hold off time value> | MIN | MAX
        :RESolution                ␣  HIGH | LOW
:AVERage
        :COUNt                     ␣  <Number of samples> | MIN | MAX

                                             ␣  TIME | COUNts
        :STATe                               ␣  ON|OFF
:FREQuency
        :RANGe
              :LOWer               ␣  <Minimum frequency for autotrigger> | MIN | MAX
:FUNCtion                          ␣  'Measuring function [ Primary channel [ , Secondary channel ] ] '
:INTernal
        :FORMat                    ␣  REAL | PACKed
:ROSCillator
        :SOURce                    ␣  INTernal | EXTernal
:TOTalize
        :GATE
              [:STATe]             ␣  ON | OFF
:VOLTage
        :GATed
              :STATe               ␣  ON | OFF
```

■ Sense Subsystem command tree for PM6685

```
[:SENSe]
 :FUNCtion             ␣  'Measuring function [␣ Primary channel [ , Secondary channel ] ] '
 :EVENt¹
        :LEVel         ␣  <Trigger level in Volts> | MIN | MAX
             :AUTO     ␣  ON | OFF | ONCE
        :HYSTeresis    ␣  <Sensitivity band in Volts> | MIN | MAX
             :AUTO     ␣  ON | OFF | ONCE
        :SLOPe         ␣  POS | NEG
 :ACQuisition
        :APERture      ␣  <Measurement Time> | MIN | MAX
        :HOFF²
             :TIME     ␣  <Hold off time> | MIN | MAX
 :AVERage
        :STATe         ␣  ON | OFF
 :ROSCillator
        :SOURce        ␣  INTernal | EXTernal
 :SDELay               ␣  <Burst sync. delay> | MIN | MAX
 :TOTalize
        :GATE
             [:STATe]  ␣  <ON | OFF
 :INTernal
        :FORMat        ␣  REAL | PACKed
```

1 Alias commands for commands in the Input subsystem.

2 Alias commands fot the: SDELay command for compatibility with the PM6680B.

PM6680B/85

:ACQuisition :APERture

␣ «<Decimal value > |MIN|MAX»

Set the Measurement Time

Sets the gate time for one measurement.

Parameters: <decimal value> is 0.8E–6, 1.6E–6, 3.2E–6, 6.4E–6, or 12.8E–6 and 50E–6 to 400s.
MIN gives 800 ns and MAX gives 400 s.
Measurement Times of 800 ns to 12.8 µs work in `:FREQ:CW`, `FREQ:BURST`, `:FREQ:PRF`, ␣:FREQ:RAT and `:PERiod`. If one of these short times is selected when the counter makes other measurements, it will use 50 µs.

If you want to switch between Average and Single measurements, use the :AVERage:STATe ␣ ON|OFF *in the Sense Subsystem.*

When Single is selected and an array measurement is done, the Measurement Time, set by `:Acquisition:APERture`, sets the time between the measurements in the array. This means that if you want a very high speed, you must set `:AVER:STATE` ␣ OFF and `:ACQ:APER` ␣ MIN.

Returned format: <Decimal value >↵

***RST condition:** 10 ms

SYST:PRESet condition: 200 ms

PM6681

:ACQuisition :APERture

␣ «<Decimal value > | MIN | MAX»

Set the Measurement Time

Sets the gate time for one measurement.

Measurement Times of 80 to 1280 ns work in `:FREQ:CW`, `FREQ:BURST`, `:FREQ:PRF`, ␣:FREQ:RAT and `:PERiod`. If one of these short times is selected when the counter makes other measurements, it will use 5 µs.

If you want to switch between Average and Single measurements, use the :AVERage:STATe␣ ON|OFF *in the Sense Subsystem.*

When Single is selected and an array measurement is done, the Measurement Time, set by `:Acquisition:APERture`, sets the time between the measurements in the array. This means that if you want a very high speed, you must set `:AVER:STATE` ␣ OFF and `:ACQ:APER` ␣ MIN.

Parameters: <decimal value> is 80, 160, 320, 640, 1280 ns and 20 µs to 400 s.
MIN gives 80 ns and MAX gives 400 s.

Returned format: <Decimal value >↵

***RST condition:** 10 ms

SYST:PRESet condition: 200 ms

:ACQuisition :HOFF

PM6680B PM6681

␣ <boolean>

Hold Off On/Off

Switches the Hold Off function On/Off.

Parameters:

<Boolean> = 1 / ON | 0 / OFF

Returned format: 1 | 0↵

***RST condition:** OFF

:ACQuisition :HOFF: ECOunt

PM6680B PM6681

␣ «<Decimal value>|MIN|MAX»

Hold Off, set event counter

Sets the Hold Off event value. The counter counts negative events on the B input (channel 2).

Parameters:

<decimal value> is a number in the range 2 to 16 777 215.

Returned format: <Decimal value>↵

***RST condition:** 100

PM6680B PM6681

:ACQuisition :HOFF :MODE

␣ «TIME|EVENt»

Hold Off Mode

Selects if triggering is going to be disabled for a preset time or for a preset number of events.

When set to event, the counter counts negative edges on the B input (channel 2).

This function is coupled to the `:ARM:START:DEL`, `:ARM:START:ECO`, `:ARM:STOP:DEL` and `:ARM:STOP:ECO`. The different delays must all be of the same type, (Time or Event). This means that setting one of them to Event delay causes the others to be set to Event delays.

Parameters:

TIME

EVENt

Returned format:

TIME| EVENt↵

***RST condition:**

TIME

PM6680B PM6681

:ACQuisition :HOFF :TIME

␣ «<Decimal value> |MIN|MAX»

Hold Off Time

Sets the Hold Off time value.

Parameters:

<Decimal data> = a number between 200E–9 and 1.6777215 for PM6680B, and between 40E–9 and 1.34217727 for PM6681.

Returned format:

<Decimal value>↵

***RST condition:**

10 µs for PM6680B and 1 µs for PM6681

:ACQuisition :RESolution

PM6680B

␣ «HIGH|LOW»

Resolution

Selects basic measurement mode for all time-related measurements.

Parameters:

HIGH: The resolution is the full 0.25 ns

LOW: The resolution is limited to a 100-ns clock. You can use this to increase the bus speed. Saves about 0.6 to 0.9 ms if the counter does real-time calculations, otherwise, only 0.05 ms.

Returned format:

HIGH|LOW↵

***RST condition:**

HIGH

:ACQuisition :RESolution

PM6681

␣ «HIGH|LOW»

Resolution

Turns off interpolator usage and also ignores the high resolution part of the count registers. Low Resolution functions only for Frequency, Period, Time-Interval and Pulse Width

Parameters:

HIGH: The resolution is the full 50 ps

LOW: The resolution is limited to 125 ns.

At low resolution no special arming and trig options are supported. There is no handling of Abort messages from the bus after the measurement series has been started. That means you cannot break off a low-resolution measurement series.

The results are based primarily on the timestamp values with 125-ns resolution. Single mode is forced on, and every period of a signal is measured. This mode is limited in frequency to <40 kHz for Frequency and Period, and <20 kHz for Time-Interval and Pulse Width. At 40 kHz the resolution is 1/400, or 2.6 digits.

Returned format:

HIGH|LOW↵

***RST condition:** HIGH

:AVERage :COUNt

␣ «<Decimal data>|MIN| MAX»

Average Samples

Sets the number of samples to use when doing time-interval averaging measurements in `:AVER:MODE ␣ COUN`. Applies to the functions:
PWIDTH, TIME, RISE and FALL TIME.

Parameters:

<Decimal data> is a number between 1and 65535.

Returned format: <Decimal data>↵

***RST condition:** 100

:AVERage :STATe

PM6680B/81/85

␣ <Boolean>

Average or Single?

Switch on/off the average function.

Parameters: <Boolean> = 1 | ON / 0 | OFF

ON means multiple period measurements for period related measurements and time-interval average for Time-Interval measurements.

OFF means that the counter measures on a single cycle. This is the same as when pressing the SINGLE key on the front panel.

When Single is selected and an array measurement is done, the Measurement Time, set by `:Acquisition:APERture`, sets the time between the measurements in the array. This means that if you want a very high speed yo must set `:AVER:STATE ␣OFF` and `:ACQ:APER ␣MIN.`

Returned format: <Boolean>↵

***RST condition:** ON

:FREQuency :RANGe :LOWer

PM6680B/PM6681

␣ «<Numeric value>|MIN|MAX»

High Speed Voltage Measurements

Use this command to speed up voltage measurements and Autotrigger functions when you don't need to measure on low frequencies.

Time to determine trigger levels				
Measuring function	**Min frequency limit (Default)**		**Max frequency limit**	
	PM6680B	*PM6681*	*PM6680B*	*PM6681*
Freq A	48 ms	85 ms	26 ms	30 ms
Time A-B	82 ms		38 ms	

Parameters:

<Numeric value> for PM6680B, 100 gives the lower frequency limit of 100 Hz and 10000 for a lower frequency limit of 10 kHz.
MIN gives 100 Hz frequency limit for PM6680B and 1Hz for PM6681.
MAX gives 10 kHz frequency limit forPM6680B and 50 kHz for PM6681.

Returned format: <Numeric value>↵

***RST condition:** 100

Complies to standards: SCPI 1991.0, confirmed.

:FUNCtion

⌴ '<Measuring function>[⌴<Primary channel> [,<Secondary channel>]]'

Select Measuring Function

Selects which measuring function is to be performed and on which channel(s) the instrument should measure.

Parameters:

<Measuring function> is the function you want to select, according to the SENSe subsystem command trees on page 9-85 and page 9-86.

<Primary channel> is the channel used in all single-channel measurements and the main channel in dual-channel measurements.

<Secondary channel> is the 'other' channel in dual-channel measurements. Only the primary channel may be programmed for all single channel measurements.

The measuring function and the channels together form one <String> that must be placed within quotation marks.

Returned format: "<Measuring function>_<Primary channel>[,<Secondary channel>]"↵

Example Select a pulse period measurement on input A (channel 1):

Send → `:FUNC ⌴ 'PER ⌴ 1'`

***RST condition:** FREQuency_1

Complies to standards: SCPI 1991.0, confirmed.

■ Functions and channels in PM6685

```
:FREQuency [ :CW ]                    [⌴  '1 | 3 | 4 | 5 | 6 ']
:FREQuency [ :CW ] :RATio             [⌴  '1 | 3 | 4 , 1 | 3 | 4 ']
:FREQuency :BURSt                     [⌴  '1 | 3 | 4 ']
:FREQuency :PRFrequency               [⌴  '1 | 3 | 4 ']
:PERiod                               [⌴  '1 | 3 | 4 ']
:NWIDth                               [⌴  '1 | 4 ']
:PWIDth                               [⌴  '1 | 4 ']
:PDUTycycle | DCYCle                  [⌴  '1 | 4 ']
:NDUTycycle                           [⌴  '1 | 4 ']
:TOTalize [ :CONTinuous ]             [⌴  '0 | 1 | 4 , 0 | 1 | 4 ']
```

■ Input channels PM6685

0 means that the input is disabled
1 means input A
3 means input C (HF-input option)
4 means input E (Rear panel arming input)
5 means input A prescaled by 2
6 means the internal reference

Functions and channels in PM6680B and PM6681

```
:FREQuency [ :CW ]               [␣ ‘(1|2 3|4|5|6|7)‘]
:FREQuency [ :CW ] :RATio        [␣ ‘1|2|3|4,1|2|3|4‘]
:FREQuency :BURSt                [␣ ‘1|2|3|4|5|6|7‘]
:FREQuency :PRF                  [␣ ‘1|2|3|4|5|6|7‘]

:PERiod                          [␣ ‘1|2|3|4|5|6|7‘]

:TINTerval                       [␣ ‘1|2|4,1|2|4|6‘]
:PHASe                           [␣ ‘1|2|6,1|2|6‘]

:NWIDth                          [␣ ‘1|2|4|6‘]
:PWIDth                          [␣ ‘1|2|4|6‘]

:DCYCle | PDUTycycle             [␣ ‘1|2|4|6‘]
:NDUTycycl                       [␣ ‘1|2|4|6‘]

:RISE:TIME                       [␣ ‘1|2‘]
:FALL:TIME                       [␣ ‘1|2‘]

:VOLT:MAXimum                    [␣ ‘1|2‘]
:VOLT:MINimum                    [␣ ‘1|2‘]
:VOLT:PTPeak                     [␣ ‘1|2‘]

:TOTalize:GATed                  [␣ ‘1|2|4|6,1|2|4|6‘]
:TOTalize:TIMed                  [␣ ‘1|2|4|6,1|2|4|6‘]
:TOTalize:ACCumulated            [␣ ‘1|2|4|6,1|2|4|6‘]
:TOTalize:SSTop                  [␣ ‘1|2|4|6,1|2|4|6‘]
```

Input channels PM6680B and PM6681

1 means input A

2 means input B

3 means input C (HF-input option)

4 means input E (Rear panel arming input)

5 means input A prescaled by 2

6 means the internal reference

7 means input A with the variable hysteresis mode

:INTernal :FORMat

«REAL|PACKed»

Internal Format

This command selects the internal data format of the measurement result from the SENSe block. The purpose of the command is to increase the measurement speed.

Parameters:

REAL means that the result is calculated in real-time after each measurement.

PACKed means that the raw measurement data is stored internally and the result is not calculated in real-time between measurements. The results are calculated later when they are sent to the controller. Since the result is not calculated, other blocks cannot use this data. That means that you cannot have the DISPlay, OUTPut, and CALCulate blocks switched on when using PACKed format.

The following measuring functions in PM6680B/81 cannot be used with PACKed format: Phase, Duty Cycle and Volt.

PM6685 cannot used PACKed format with Duty Cycle.

The internal format affects the number measuring results that the measurement result buffer can hold.

		Number of Results in Buffer	
Format	**Measuring Function**	PM6680B/85	PM6681
Real:	All functions	2048	7019
Packed:	Frequency, Period, Ratio Totalize	2166	6143
	Pulse Width, Time-Interval, Rise/Fall time	764	4466
	Phase, Duty Cycle, Volt	N.A.	N.A.
	Low resolution Frequency and Period	N.A.	8191
	Low Res. Time-Interval and Pulse Width	N.A.	4095

You must consider this when fetching results with the `:FETCh:ARRay` query.

***RST condition:** REAL

:ROSCillator :SOURce

PM6680B/81/85

⎵ «INT|EXT»

Select Reference Oscillator

Selects the signal from the external reference input as timebase instead of the internal timebase oscillator.

Returned format: <INT|EXT>↵

***RST condition:** INT

Complies to standards: SCPI 1991.0, confirmed.

:SDELay

PM6685

⎵ «<Numeric value>|MIN|MAX»

BURST/PRF Synchronization Delay

Sets the synchronization delay time used in FREQuency:BURSt | PRF measurements.

Parameter range: 200 ns to 1.6777215 s

***RST condition:** 10 ms

PM6680B/81/85

:TOTalize :GATE

␣ <Boolean>

Gate On/Off

Open/closes the gate for `:TOTalize[:CONTinuous]`.

☞ *Before opening the gate with this command, the counter must be in the 'continuously initiated' state ,* (`:INIT:CONT ␣ ON`)*or else the totalizing will not start.*

Parameters: <Boolean> = (1 / ON | 0 / OFF)

Returned format: <Boolean>↵

Example:

```
Send → :FUNC ␣ 'TOT ␣ 1'                    Selects totalizing
       on input A
Send → :INIT:CONT ␣ ON;TOT:GATE ␣ ON
```

This will initiate totalizing, reset the totalizing value to zero, and start totalizing.

```
Send → TOT:GATE ␣ OFF                        Stop totalizing
Read ← :FETCh:ARRay? ␣ -1                Read the final result
```

***RST condition:** OFF

PM6680B PM6681

:VOLTage:GATed:STATe

␣ <Boolean>

Gated Voltage Measurement

Selects the gated mode for the `:VOLTage:MAX|MIN|PTPeak` measuring functions and for the Autotrigger function.

The gated mode is useful for removing overshoot and undershoot. The gate signal is controlled by the `:ARM:STOP:SLOPe` and `:ARM:STOP:SOURce` commands. If channel 2 (B) is the source for the gating signal, all other characteristics of that channel can be used. When Gated Voltage is selected, the Stop Arming function is disabled from its normal stop arming usage. When gated voltage mode is selected, high enables measurement and low disables measurements. Use the slope if you want it the other way around.

Parameters:

<Boolean> = 1/ON|0/OFF

Returned format: 1|0 ↵

***RST condition:** OFF

This page is intentionally left blank.

Status Subsystem

```
:STATus
    :DREGister0
                :ENABle          ␣  <bit mask>
                [:EVENt]?
    :OPERation
                :CONDition?
                :ENABle          ␣  <bit mask>
                [:EVENt]?
    :QUEStionable
                :CONDition?
                :ENABle          ␣  <bit mask>
                [:EVENt]?
    :PRESet
```

■ Related Common Commands:

```
*CLS
*ESE        ␣  <bit mask>
*ESR?
*PSC        ␣  <bit mask>
*SRE        ␣  <bit mask>
*STB?
```

:STATus :DREGister0?

0
80B/81/85

Read Device Status Event Register

This query reads out the contents of the Device Event Register. Reading the Device Event Register clears the register. See Figure 6-14.

Returned format:

<dec.data> = the sum (between 0 and 6) of all bits that are true. See table below:

Bit No.	Weight	Condition
2	4	Last measurement below low limit.
1	2	Last measurement above high limit.

:STATus :DREGister0 :ENABle

0
80B/81/85

␣ <bit mask>

Enable Device Status Reporting

This command sets the enable bit of the Device Register 0.

Parameters:

<dec.data> = the sum (between 0 and 6) of all bits that are true. See table below:

Bit No.	Weight	Condition
2	4	Enable monitoring of low limit
1	2	Enable monitoring of high limit

Returned format: <bit mask>↵

80B/81/85

:STATus :OPERation :CONDition?

Read Operation Status Condition Register

Reads out the contents of the operation status condition register. This register reflects the state of the measurement process. See figure below. Note that bits 0 to 3, 7, and 9 to 15 are not used.

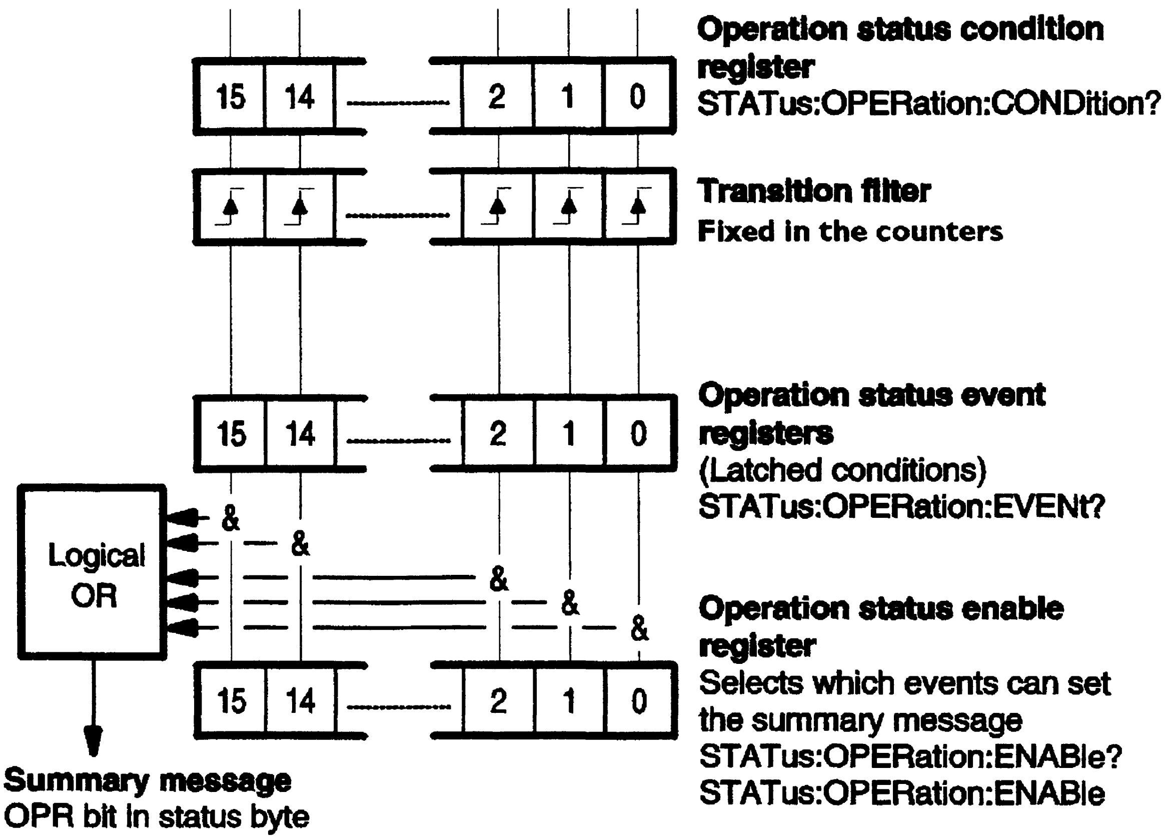
Returned Format:

<Decimal data> = the sum (between 0 and 368) of all bits that are true. See table below:

Bit No.	Weight	Condition
8	256	Not Measurement
6	64	Waiting for bus arming
5	32	Waiting for triggering and/ or external arming
4	16	Measurement

Complies to standards: SCPI 1991.0, confirmed

Device status continously monitored



:STATus :OPERation :ENABle

80B/81/85

⌴ <Decimal data>

Enable Operation Status Reporting

Sets the enable bits of the operation status enable register. This enable register contains a mask value for the bits to be enabled in the operation status event register. A bit that is set true in the enable register enables the corresponding bit in the status register. See figure on page 9-101.

An enabled bit will set bit #7, OPR (Operation Status Bit), in the Status Byte Register if the enabled event occurs. See also status reporting on page 3-14.

Power-on will clear this register if power-on clearing is enabled via `*PSC`.

Parameters: <dec.data> = the sum (between 0 and 368) of all bits that are true. See table below:

Bit No.	Weight	Condition
8	256	No measurement
6	64	Waiting for bus arming
5	32	Waiting for triggering and/or external arming
4	16	Measurement

Returned Format: <Decimal data>↵

Example:

SEND→ `:STAT:OPER:ENAB ⌴ 288`

In this example, waiting for triggering, bit 5, and Measurement stopped, bit 8, will set the OPR-bit of the Status Byte. (This method is faster than using `*OPC` if you want to know when the measurement is ready.)

Complies to standards: SCPI 1991.0, confirmed.

:STATus:OPERation?

80B/81/85

Read Operation Status, Event

Reads out the contents of the operation event status register. Reading the Operation Event Register clears the register. See figure on page 9-101.

Returned Format: <Decimal data>↵

<dec.data> = the sum (between 0 and 368) of all bits that are true. See table on page 9-102.

Complies to standards: SCPI 1991.0, confirmed.

PM6680B/81/85

:STATus :PRESet

Enable Device Status Reporting

This command has an SCPI standardized effect on the status data structures. The purpose is to precondition these toward reporting only device-dependent status data.

– It only affects enable registers. It does not change event and condition registers.

– The IEEE-488.2 enable registers, which are handled with the common commands `*SRE` and `*ESE` remain unchanged.

– The command sets or clears all other enable registers. Those relevant for this counter are as follows:

– It sets all bits of the Device status Enable Registers to 1.

– It sets all bits of the Questionable Data Status Enable Registers and the Operation Status Enable Registers to 0.

– The following registers never change in the counter, but they do conform to the standard `:STATus:PRESet` values.

– All bits in the positive transition filters of Questionable Data and Operation status registers are 1.

– All bits in the negative transition filters of Questionable Data and Operation status registers are 0.

Complies to standards: SCPI 1991.0, confirmed.

STATus :QUEStionable :CONDition?

PM6680B/81/85

Read Questionable Data/Signal Condition Register

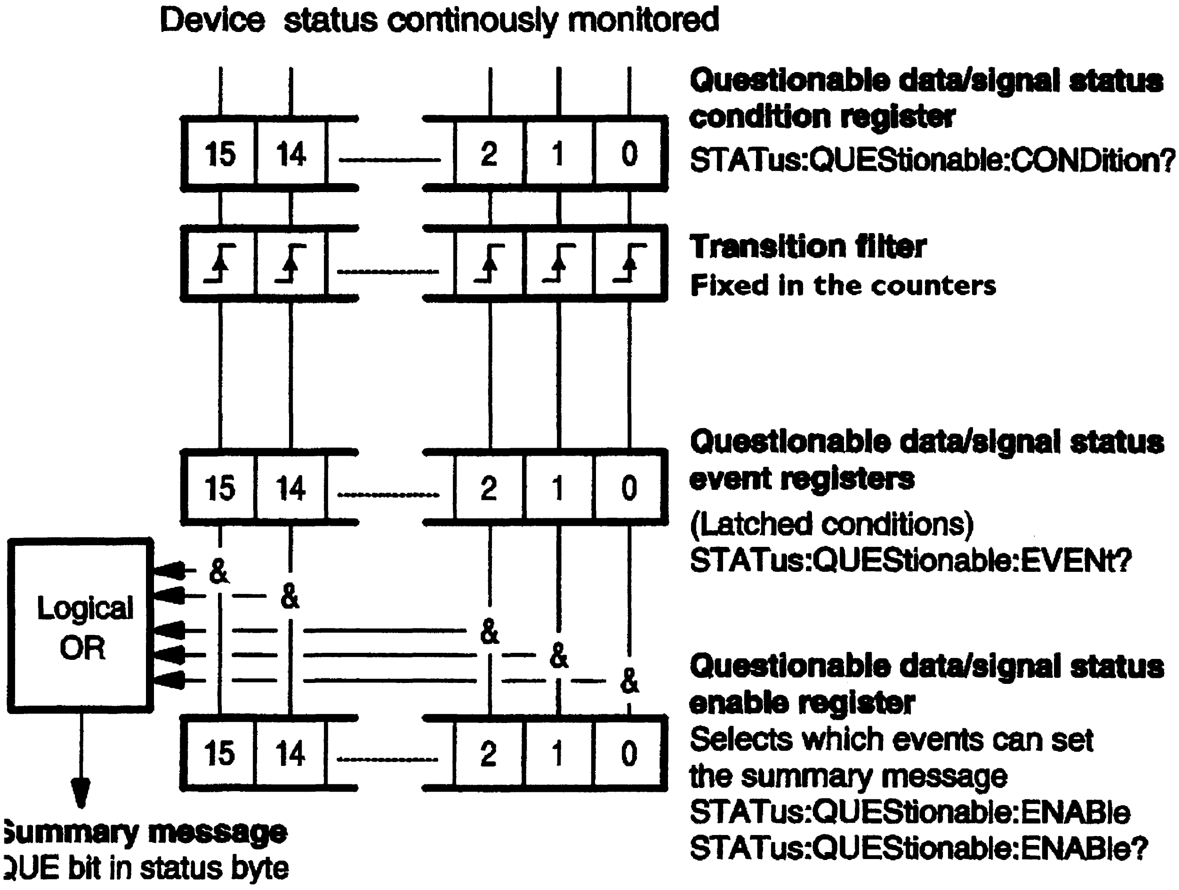
Reads out the contents of the status questionable condition register.

Returned Format:

<dec.data> = the sum (between 0 and 17920) of all bits that are true. See table below:

Bit No.	Weight	Condition
14	16384	Unexpected parameter
10	1024	Timeout or no signal detected
9	512	Overflow

Complies to standards: SCPI 1991.0, confirmed.


Device status continously monitored
15
14
2
1
0
Questionable data/signal status condition register
STATus:QUEStionable:CONDition?
Transition filter
Fixed in the counters
Questionable data/signal status event registers
(Latched conditions)
STATus:QUEStionable:EVENt?
Logical OR
&
Questionable data/signal status enable register
Selects which events can set the summary message
STATus:QUEStionable:ENABle
STATus:QUEStionable:ENABle?
Summary message
QUE bit in status byte

PM6680B/81/85

:STATus :QUEStionable :ENABle

␣ <Decimal data>

Enable Questionable Data/Signal Status Reporting

Sets the enable bits of the status questionable enable register. This enable register contains a mask value for the bits to be enabled in the status questionable event register. A bit that is set true in the enable register enables the corresponding bit in the status register. See figure on page 9-104.

An enabled bit will set bit #3, QUE (Questionable Status Bit), in the Status Byte Register if the enabled event occurs. See also status reporting on page 3-14.

Power-on will clear this register if power-on clearing is enabled via `*PSC`.

Parameters:

<dec.data> = the sum (between 0 and 17920) of all bits that are true. See the table on page 9-104.

Returned Format: <Decimal data> ↲

Example:

Send → `:STAT:QUES:ENAB ␣ 16896`

In this example, both 'unexpected parameter' bit 14, and 'overflow' bit 8, will set the QUE-bit of the Status Byte when a questionable status occurs.

Complies to standards: SCPI 1991.0, confirmed.

PM6680B/81/85

:STATus :QUEStionable?

Read Questionable Data/Signal Event Register

Reads out the contents of the status questionable event register. Reading the Status Questionable Event Register clears the register. See figure on page 9-104.

Returned Format:

<dec.data> = the sum (between 0 and 17920) of all bits that are true. See the table on page 9-104.

Complies to standards: SCPI 1991.0, confirmed.

This page is intentionally left blank.

System Subsystem

```
:SYSTem
        :COMMunicate
              :GPIB
                              :ADDRess          ⎵  <Numeric value> | MIN | MAX
        :ERRor?
        :PRESet
        :SYSTem:SDETect[:ENABle]                ⎵  ON | OFF                      (Only PM6685)
        :SET                                         ⎵  <Block data>
        :TIME
              :ELAPsed?
        :TOUT
              [:STATe]                          ⎵  ON | OFF
              :TIME                                  ⎵  <timeout value>
        :UNPRotect
        :VERSion?
```

■ Related common command:

```
*IDN?
*OPT?
*PUD ⎵  <arbitrary block program data>
*RST
```

SYSTem :COMMunicate: GPIB: ADDRess 80B/81/85

«<Numeric value>|MAX|MIN» [,«<Numeric value>|MAX|MIN»]

;et GPIB Address

This command sets the GPIB address. This selection overrides the switches on the rear panel of the counter. The set address is valid until a new address is set, either by bus command, switch setting, or via the front panel AUX-MENU.

arameters:

<Numeric value> is a number between 0 and 30.
MIN sets address 0.
MAX sets address 30.
[,<Numeric value>|MAX|MIN] sets a secondary address. This is accepted but not used in PM6681 and PM6685. PM6680B does not accept a secondary address.

`[:SELF]` ␣ This optional parameter is accepted by PM6681 and PM6685. PM6680B does not accept `[:SELF]`.

eturned format:> <Numeric value>↵

xample:

```
;END→ :SYST:COMM:GPIB:ADDR ␣ 12
```

'his example sets the bus address to 12.

omplies to standards: SCPI 1991.0, confirmed.

SYSTem :ERRor? PM6680B/81/85

Queries for an ASCii text description of an error that occurred. The error messages are placed in an error queue, with a FIFO (First In-First Out) structure. This queue is summarized in the Error AVailable (EAV) bit in the status byte.

eturned format:

<error number>,"<Error Description String>"↵

'There:

<Error Description String> = an error description as ASCii text.

ee also: Chapter 8, error messages.

omplies to standards: SCPI 1991.0, confirmed.

PM6680B/81/85

:SYSTem :PRESet

Preset

This command sets the counter to the same default settings as when the front panel key LOCAL/PRESET is pressed in local mode.

These are not exactly the same settings as after *RST, SYST:PRES *gives 200 ms Measurement Time and signal detection ON, while* *RST *gives 10-ms Measurement Time and signal detection OFF.*

See also: Default settings on page -.

Complies to standards: SCPI 1991.0, confirmed.

PM6685

:SYSTem :SDETect

␣ <Boolean>

Signal Detection

This command switches on or off the signal detection, that is, the ability to show NO SIGNAL, NO TRIG on the display.

When signal detection is enabled, the measurement attempt will be abandoned when the no signal is detected. A zero result will be sent to the controller instead of a measurement result, and the timeout bit in the STATus QUEStionable register will be set.

Returned format: «0|1»↲

Where:

0 means no signal detection
1 means signal detection ON.

***RST condition:** 0

SYSTem :SET

PM6680B/81/85

<Block data>

ead or Send Settings

Transmits in binary form the complete current state of the instrument. This data can be sent to the instrument to later restore this setting. This command has the same function as the `*LRN?` common command with the exception that it returns the data only as response to `:SYST:SET?`. The query form of this command returns a definite block data element.

arameters:

Block data> is the instrument setting previously retrieved via the `:SYSTem:SET?` query.

eturned format: <Block data>↵

here:

<Block data> is #292<92 data bytes> for PM6680B

<Block data> is #3104<104 data bytes> for PM6681

<Block data> is #276<76 data bytes> for PM6685

```
END→ :SYST:SET?
EAD← #2764...-_ ._ .Çä......d..._
       +................-.c-.............._ _ ?..............d
```

omplies to standards: SCPI 1991.0, confirmed.

SYSTem :TIME :ELAPsed?

PM6680B/81/85

ead On-time

Use this command if you want to know (in seconds) how long the counter has been on.

eturned format:

<String>=Power-on time.

For PM6680B and PM6685, this is the time elapsed since the last power-on.

For PM6681, this is the total elapsed time since the counter was new.

PM6680B/81/85

:SYSTem :TOUT

⌣ <Boolean

Timeout On/Off

This command switches on or off the timeout. When timeout is enabled, the measurement attempt will be abandoned when the time set with `:SYST:TOUT:TIME` has elapsed. A zero result will be sent to the controller instead of a measurement result and the timeout bit in the STATus QUEStionable register will be set.

Returned format: 0 means no timeout; 1 means that the timeout set by `:SYSTem:TOUT:TIME` is used.

Example:

SEND→ `:SYST:TOUT ⌣ 1;TOUT:TIME ⌣ 0.5;:STAT:QUES:ENAB ⌣ 1024;:*SRE ⌣ 8`

This example turns on timeout, sets the timeout to 0.5 s, enables status reporting of questionable data at timeout, and enables service request on questionable data.

```
SEND→ *STB?         If bit 3 in the status byte is set, read the
     questionable data status.
SEND→ :STAT:QUES:EVEN?           This query reads the ques-
     tionable data status.
READ← «1024|0»         1024 means timeout has occurred, and 0
     means no timeout.
```

***RST condition:** 0

PM6680B/81/85

:SYSTem :TOUT :TIME

⌣ «<Numeric value>|MIN|MAX

Timeout, Set

This command sets the timeout in seconds.

The timeout starts when a measurement starts and, if no result is obtained when the set timeout has elapsed the measurement is terminated.

Note that you must enable timeout using :SYST:TOUT_ON for this setting to take effect.

Parameters:

<Numeric value> is the timeout in seconds. The range is 0.1 to 25.5 s for PM6680B and PM6685. The range is 64 ms to 400 s for PM6681

MIN gives 0.1 s (64 ms for PM6681)

MAX gives 25.5 s (400 s for PM6681)

Returned format: <Numeric value>↵

***RST condition:** 0.1 (6.4E-2 for PM6681)

Complies to standards: SCPI 1991.0, confirmed.

SYSTem :UNPRotect

PM6680B/81/85

Inprotect

This command will unprotect the user data (set/read by *PUD) and front setting memories 10-19 until the next PMT (Program message terminator) or Device clear or Reset (*RST). This makes it necessary to send an unprotect command in the same message as for instance *PUD.

xample

end → :SYST:UNPR; *PUD ␣ #240Calibrated ␣ 1992-11-17, ␣ inventory No.1234

here:

# means that <arbitrary block program data> will follow.

2 means that the two following digits will specify the length of the data block

40 is the number of characters in this example

SYSTem :VERSion?

PM6680B/81/85

ystem Version

This query returns the SCPI system version that this instrument complies to.

eturned format:

<year>.<revision>↵

Where <year> is the year of publication of the complied standard and <revision> is the number of the SCPI standard.

xample

end → :SYST:VER?

ead← 1991.0

omplies to standards: SCPI 1991.0, confirmed.

Test Subsystem

```
:TEST
        :CHECk        ˽  ON | OFF
        :SELect       ˽  RAM | ROM | LOGic | DISPlay | ALL
```

■ Related common command:

*TST

TEST:CHECk

PM6680B/81/85

<Boolean>

ielect Check signal

This command connects the internal reference signal to the measuring logic, instead of an external measuring signal. This makes it possible to test all functions.

The frequency of the reference is 10 MHz for PM6680B and PM6685, and 100 MHz for PM6681.

arameters:

Boolean> = 1 / ON | 0 / OFF

1 and ON means test signal connected

0 and OFF means test signal disconnected.

Selecting channel 6 when entering measuring channel for :CONFigure :MEASure, *etc., also selects the reference.*

eturned format: 1|0↵

ST condition: 0

TEST :SELect

PM6680B/81/85

«RAM | ROM | LOGic | DISPlay | ALL»

ielect Self-tests

Selects which internal self-tests shall be used when self-test is requested by the *TST command.

eturned format:

«RAM | ROM | LOGic | DISPlay | ALL»↵

ST condition: ALL

Trigger Subsystem

```
:TRIGger
 [ STARt / :SEQuence [ 1 ] ]
      [ :LAYer [ 1 ] ]
           :COUNt␣   <Numeric value> | MIN | MAX
```

■ Related common command:

*TRG

TRIGger:COUNt

PM6680B/81/85

, «<Numeric value> | MIN | MAX»

lo. of Triggerings on each Ext Arm start

Sets how many measurements the instrument should make for each ARM:STARt condition, (block arming).

These measurements are done without any additional arming conditions before the measurement. This also means that stop arming is disabled for the measurements inside a block.

The actual number of measurements made on each INIT *equals to:*
(:ARM:START:COUN)*(:TRIG:START:COUNT)

'arameters:

:Numeric value> is a number between 1 and 65535.

MAX gives 65535

MIN gives 1

:xample:

;END→ `:TRIG:COUN ␣ 50`

:eturned format: <Numeric value>↵

RST condition: 1

:omplies to standards: SCPI 1991.0, confirmed.

Common Commands

*CLS		
*DMC	␣	<Macro label> , <Program messages>
*EMC	␣	<Decimal data>
*ESE	␣	<Decimal data>
*ESR?		
*GMC?	␣	<Macro label>
*IDN?		
*LMC?		
*LRN?		
*OPC		
*OPC?		
*OPT?		
*PMC		
*PSC	␣	<Decimal data>
*PUD	␣	<Arbitrary block program data>
*RCL	␣	<Decimal data>
*RMC	␣	<Macro name>
*RST		
*SAV	␣	<Decimal data>
*SRE	␣	<Decimal data>
*STB?		
*TRG		
*TST?		
*WAI		

*CLS

PM6680B/81/85

Clear Status Command

The *CLS common command clears the status data structures by clearing all event registers and the error queue. It does not clear enable registers and transition filters. It clears any pending *WAI, *OPC, and *OPC?.

xample:

Send → `*CLS`

Complies to standards: IEEE 488.2 1987.

*DMC

⌣ <Macro label> , <Program messages>

Define Macro

Allows you to assign a sequence of one or more program message units to a macro label. The sequence is executed when the macro label is received as a command or query. Twenty five macros can be defined at the same time, and each macro can contain an average of 40 characters.

If a macro has the same name as a command, it masks out the real command with the same name when macros are enabled. If macros are disabled, the original command will be executed.

If you define macros when macro execution is disabled, the counter executes the *DMC command fast, but if macros are enabled, the execution time for this command is longer.

Parameters:

<Macro label> = 1 to 12-character macro label. (String data must be surrounded by " " or ' as in the example below.)

<Program messages> = the commands to be executed when the macro label is received, both block data and string data formats can be used.

Example 1:

SEND→ `*DMC 'AMPLITUDE?',":FUNC ⌣ 'FREQ ⌣ 1';:INP:HYST:AUTO ONCE ⌣ ;:INP:HYST?;:INP:LEV?"`

This example defines a macro called amplitude?.

SEND→ `AMPLITUDE?`

The macro makes an AUTO ONCE and reads out the hysteresis and trigger level that auto selects. (Macros must be enabled; otherwise, the `:AMPLITUDE?` query will not execute, see ☛EMC)).

READ← `+3.46125461E-001;+3.64852399E-001`

Auto selects 33% of Vpp as hysteresis, so multiplying the first part of this reading by 3 will give you the signal amplitude (0.346*3=1.04 V in this example). You can also calculate positive peak voltage: $\frac{\textit{Hysteresis} * 3}{2} + \textit{Trigger Level}$ and negative peak voltage: Vp$\frac{\textit{Hysteresis} * 3}{2} - \textit{Trigger Level}$

Example 2:

SEND→ `*DMC ⌣ 'AUTOFILT',":INP:HYST:AUTO ⌣ $1;:INP:FILT ⌣ $1`

This example defines a macro AUTO which takes one argument, i.e., auto «ON|OFF|ONCE» ($1).

SEND→ `AUTOFILT ⌣ OFF`

Turns off both the auto function and the filter.

Complies to standards: IEEE 488.2 1987.

EMC

PM6680B/81/85

<Decimal data>

nable Macros

This command enables and disables expansion and execution of macros. If macros are disabled, the instrument will not recognize a macro although it is defined in the instrument. (The Enable Macro command takes a long time to execute.)

arameters:

Decimal data> = is 0 or 1. A value which rounds to 0 turns off macro execution. Any other value turns macro execution on.

☞ *Note that 1 or 0 is <Decimal data>, not <Boolean>!*
ON|OFF is `not` *allowed here!*

eturned format: «0|1» ↵

1 indicates that macro expansion is enabled.
0 indicates that macro expansion is disabled.

cample:

END→`*EMC ␣ 1`

Enables macro expansion and execution.

omplies to standards: IEEE 488.2 1987.

*ESE

⌴ <Decimal data>

Standard Event Status Enable

Sets the enable bits of the standard event enable register. This enable register contains a mask value for the bits to be enabled in the standard event status register. A bit that is set true in the enable register enables the corresponding bit in the status register. An enabled bit will set the ESB (Event Status Bit) in the Status Byte Register if the enabled event occurs. See also status reporting on page 3-14.

Parameters: <dec.data> = the sum (between 0 and 255) of all bits that are true.

Event Status Enable Register (1 = enable)

Bit	Weight	Enables
7	128	PON, Power-on occurred
6	64	URQ, User Request
5	32	CME, Command Error
4	16	EXE, Execution Error
3	8	DDE, Device Dependent Error
2	4	QYE, Query Error
1	2	RQC, Request Control (not used)
0	1	Operation Complete

Returned Format: <Decimal data> ↵

Example:

SEND→ `*ESE ⌴ 36`

In this example, command error, bit 5, and query error, bit 2, will set the ESB-bit of the Status Byte if these errors occur.

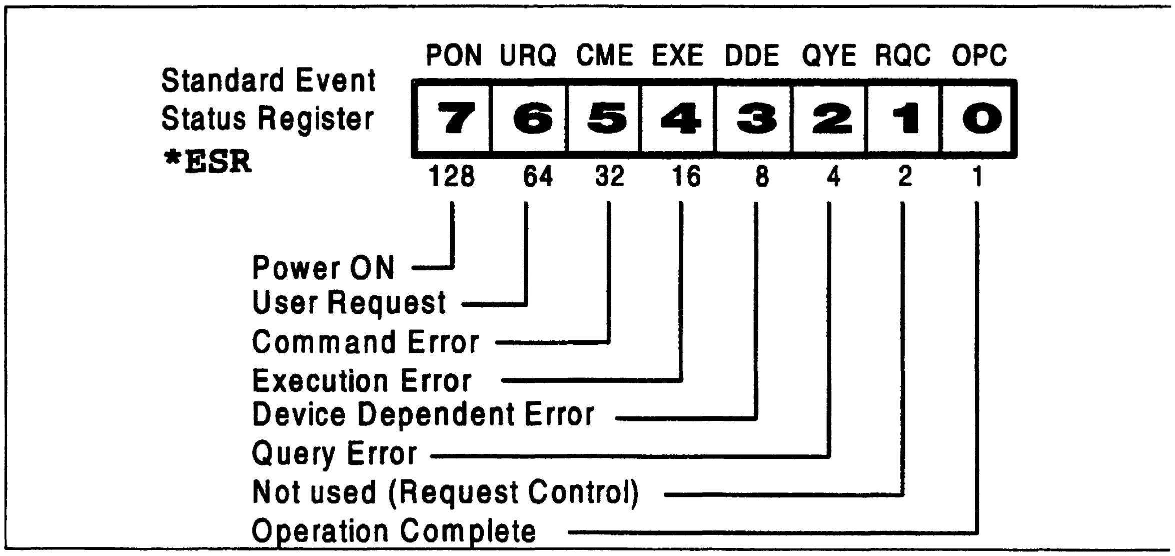


Figure 9-3 *Bits in the standard event status register.*

Complies to standards: IEEE 488.2 1987.

ESR?

PM6680B/81/85

vent Status Register

Reads out the contents of the standard event status register. Reading the Standard Event Status Register clears the register.

eturned Format:

dec.data> = the sum (between 0 and 255) of all bits that are true. See table on page 9-121.

omplies to standards: IEEE 488.2 1987.

GMC?

< macro label>

PM6680B/81/85

et Macro Definition

This command makes the counter respond with the current definition of the given macro label.

arameters:

Macro label> = the label of the macro for which you want to see the definition. (String data must be surrounded by " " or ' ' as in the example below.)

eturned Format: <Block data>↵

xample:

END→ `*GMC? ␣ 'AMPLITUDE?'`

Gives a block data response, for instance:

EAD←

#255:FUNC 'FREQ 1';:INP:HYST:AUTO ONCE;:INP:HYST?;INP:LEV?

omplies to standards: IEEE 488.2 1987.

PM6680B/81/85

*IDN?

Identification query

Reads out the manufacturer, model, serial number, Firmware level for main and GPIB program in an ASCii response data element. The query must be the last query in a program message.

Response is <Manufacturer> , <Model> , <Serial Number>, <Firmware Level>.

<Serial number> is not implemented in PM6680B and PM6685 and will always return a zero. Please look at the type plate at the rear panel of the counter if you are interested in the serial number. PM6681 returns the correct serial number.

Example

```
SEND →*IDN?
READ← Fluke, ␣ ␣ ␣ ␣ PM6685, ␣ 0, ␣ MAIN ␣ V1.01 ␣ 19 Nov ␣
      1992 ␣ / ␣ GPIB ␣ V1.12  ␣ ␣ 28 ␣ Oct ␣ 1992
```

Complies to standards: IEEE 488.2 1987.

PM6680B/81/85

*LMC?

Learn Macro

Makes the instrument send a list of string data elements, containing all macro labels defined in the instrument.

Returned Format:

<String> { ,<String> }↵

<String> = a Macro label. (String data will be surrounded by “ ” as in the example below.)

Example:

SEND→ *LMC?

May give the following response:

READ←"AUTOFILT","AMPLITUDE?"

Complies to standards: IEEE 488.2 1987.

'LRN?

PM6680B/81/85

.earn Device Setup

Learn Device Setup Query. Causes a response message that can be sent to the instrument to return it to the state it was in when the *LRN? query was made.

!eturned Format:

:SYST:SET_<Block data>↵

Vhere:

<Block data> is #292<92 data bytes> for PM6680B
<Block data> is #3104<104 data bytes> for PM6681
<Block data> is #276<76 data bytes> for PM6685

:xample

```
;END→ *LRN?
```

:omplies to standards: IEEE 488.2 1987.

'OPC

PM6680B/81/85

)peration Complete

The Operation Complete command causes the device to set the operation complete bit in the Standard Event Status Register when all pending selected device operations have been finished. See also Example 4 in Chapter 4.

:xample:

:nable OPC-bit

```
;END→ *ESE ␣ 1
```

;tart measurement (INIT). *OPC will set the operation complete bit in the status register when the ıeasurement is done.

```
;END→ :INIT;*OPC
```

Vait 1s for the measurement to stop. Read serial poll register, will reset service request

;POLL

:heck the Operation complete bit (0) in the serial poll byte. If it is true the measurement is ompleted and you can fetch the result.

```
;END→ FETCh?
```

'hen read the event status register to reset it:

```
;END→ *ESR?
```

' bit 0 is false, abort the measurement.

```
;END→ :ABORt
```

:omplies to standards: IEEE 488.2 1987.

PM6680B/81/85

*OPC?

Operation Complete Query

Operation Complete query. The Operation Complete query places an ASCii character 1 into the device's Output Queue when all pending selected device operations have been finished.

Returned Format: 1↵

See also:

Example 6 is Chapter 4.

Complies to standards: IEEE 488.2 1987.

PM6680B/81/85

*OPT?

Option Identification

Response is a list of all detectable options present in the instrument, with absent options represented with an ASCii '0'.

Returned format:

<Bus option>,<Prescaler option>↵

Where:

<Bus option> = GPIB
<Prescaler option> = 0|10|20
0 for prescaler option means that no prescaler is installed.

☞ *Oscillator type are not detectable and can therefore, not be reported.*

Complies to standards: IEEE 488.2 1987.

PMC

PM6680B/81/85

urge Macros

Removes all macro definitions.

‹ample: `*PMC`

ɘe also:

`:MEMory:DELete:MACRo ˽'<Macro-name>'` if you want to remove a single macro.

ɔmplies to standards: IEEE 488.2 1987.

PSC

<Decimal data>

PM6680B/81/85

ower-on Status Clear

Enables/disables automatic power-on clearing. The status registers listed below are cleared when the power-on status clear flag is 1. Power-on does not affect the registers when the flag is 0.

Service request enable register (*SRE)

Event status enable register (*ESE)

Operation status enable register (:STAT:OPER:ENAB)

Questionable data/signal enable register (:STAT:QUES:ENAB)

Device enable registers (:STAT:DREG0:ENAB)

*RST does not affect this power-on status clear flag.

ɪrameters: <Decimal data> = a number that rounds to 0 turns off automatic power-on clearing. Any other value turns it on.

ɘturned Format: «1 | 0» ↲

is enabled and 0 is disabled.

‹ample: `*PSC ˽ 1`

This example enables automatic power-on clearing.

ɔmplies to standards: IEEE 488.2 1987.

PM6680B/81/85

*PUD

˽<Arbitrary block program data>

Protected User Data

Protected user data. This is a data area in which the user may write any data up to 64 characters. The data can always be read, but you can only write data after unprotecting the data area. A typical use would be to hold calibration information, usage time, inventory control numbers, etc.

The content at delivery is: #234 FACTORY CALIBRATED ON: 19YY-MM-DD

– YY = year, MM = month, DD = day

Returned format: <Arbitrary block response data>↵

– Where:

<arbitrary block program data> is the data last programmed with *PUD.

Example

Send →
```
:SYST:UNPR; *PUD ˽ #240Calibrated ˽ 1993-07-16,  ˽ inven-
     tory ˽ No.1234
```

# means that <arbitrary block program data> will follow.

2 means that the two following digits will specify the length of the data block.

40 is the number of characters in this example.

Complies to standards: IEEE 488.2 1987.

PM6680B/81/85

*RCL

˽<Decimal data>

Recall

Recalls one of the up to 20 previously stored complete instrument settings from the internal nonvolatile memory of the instrument.

Memory number 0 contains the power-off settings for PM6685. For PM6681 memory number 0 contains the power-off settings until PRESET is pressed. After preset, memory 0 contains the pre-preset settings.

Parameters:

<Decimal data> = a number between 0 and 19.

Example:

SEND→
```
*RCL ˽ 10↵
```

Complies to standards: IEEE 488.2 1987.

RMC

PM6680B/81/85

'<Macro name>'

elete one Macro

This command removes an individual MACRo.

arameters:

'<Macro name>' is the name of the macro you want to delete.

<Macro name> is String data that must be surrounded by quotation marks.

e also:

`*PMC`, if you want to delete all macros.

RST

PM6680B/81/85

eset

The Reset command resets the counter. It is the third level of reset in a 3-level reset strategy, and it primarily affects the counter functions, not the IEEE 488 bus.

The counter settings will be set to the default settings listed on page -. All previous commands are discarded, macros are disabled, and the counter is prepared to start new operations.

ample: `*RST`

e also:

Default settings on page -.

mplies to standards: IEEE 488.2 1987.

Save

Saves the current settings of the instrument in an internal nonvolatile memory. Nineteen memory locations are available. Switching the power off and on does not change the settings stored in the registers.

Note that memory positions 10 to 19 can be protected from the front panel auxiliary menu. If this has been done, use the `:SYSTem:UNPRotect` command, then you can alter these memory positions.

Parameters

<Decimal data> = a number between 1 and 19.

Example:

SEND→ `*SAV ⌣ 10↵`

Complies to standards: IEEE 488.2 1987

SRE

PM6680B/81/85

<Decimal data>

ervice Request Enable

The Service Request Enable command sets the service request enable register bits. This enable register contains a mask value for the bits to be enabled in the status byte register. A bit that is set true in the enable register enables the corresponding bit in the status byte register to generate a Service Request.

arameters: <dec.data> = the sum (between 0 and 255) of all bits that are true. See table below:

Service Request Enable Register (1 = enable)

Bit	*Weight*	*Enables*
7	128	OPR, Operation Status
6	64	RQS, Request Service
5	32	ESB, Event Status Bit
4	16	MAV, Message Available
3	8	QUE, Questionable Data/Signal Status
2	4	EAV, Error Available
1	2	Not used
0	1	Device Status

eturned Format: <Integer>↵

here:

<Integer> = the sum of all bits that are set.

xample: `*SRE ␣ 16`

In this example, the counter generates a service request when a message is available in the output queue.

omplies to standards: IEEE 488.2 1987.

PM6680B/81/85

*STB?

Status Byte Query

Reads out the value of the Status Byte. Bit 6 reports the Master Summary Status bit (MSS), not the Request Service (RQS). The MSS is set if the instrument has one or more reasons for requesting service.

Returned Format:

<Integer> = the sum (between 0 and 255) of all bits that are true. See table below:

Status Byte Register (1 = true)			
Bit	*Weight*	*Name*	*Condition*
7	128	OPR	Enabled operation status has occurred.
6	64	MSS	Reason for requesting service
5	32	ESB	Enabled status event condition has occurred
4	16	MAV	An output message is ready
3	8	QUE	The quality of the output signal is questionable
2	4	EAV	Error available
1	2		Not used
0	1	DREG0	Enabled status device event conditions have occurred

See also: If you want to read the status byte with the RQS bit, use serial poll.

Complies to standards: IEEE 488.2 1987.

PM6680B/81/85

*TRG

Trigger

The trigger command `*TRG` starts the measurement and places the result in the output queue.

It is the same as:

```
:ARM:STARt:LAYer2:IMM; *WAI;:FETCh?
```

The Trigger command is the device-specific equivalent of the IEEE 488.1 defined Group Execute Trigger, GET. It has exactly the same effect as a GET after it has been received, and parsed by the counter.

However, GET is much faster than `*TRG`, since GET is a hardware signal that does not have to be parsed by the counter.

Example:

```
SEND→ :ARM:START:LAY2:SOURCE ⌴ BUS
SEND→ :INIT:CONT ⌴ ON
SEND→ *TRG
READ← +3.2770536E+004
```

Type of Command:

Aborts all previous measurement commands if not `*WAI` is used.

Complies to standards: IEEE 488.2 1987.

TST?

PM6680B/81/85

ielf Test

The self-test query causes an internal self-test and generates a response indicating whether or not the device completed the self-test without any detected errors.

eturned Format: <Integer>↵

'here:

<Integer> = a number indicating errors according to the table below.

<Integer> =	PM6680B Error	PM6681, PM6685 Error
0	No error	
1	RAM Failure	Display Failure
2	ROM 1 Failure	Logic Failure
4	Logic Failure	RAM Failure
8	Display Failure	Bus ROM Failure
16	Not used	ROM Bank 1 Failure
32	Not used	ROM Bank 2 Failure

omplies to standards: IEEE 488.2 1987

WAI

PM6680B/81/85

/ait-to-continue

The Wait-to-Continue command prevents the device from executing any further commands or queries until execution of all previous commands or queries has been completed.

cample:

END→ `:MEAS:FREQ?; *WAI;:MEAS:PDUT?`

In this example, `*WAI` makes the instrument perform both the frequency and the Duty Cycle measurement. Without `*WAI`, only the Duty Cycle measurement would be performed.

EAD← `+5.1204004E+002;+1.250030E-001`

omplies to standards: IEEE 488.2 1987.

Chapter 10

Index

Index

!

- Mohm 9-47
- 50 ohms 9-47

A

- Abort 7-4
  - Measurement 9-4
- AC|DC 2-2,9-44
- Accumulated, totalize X gated by Y 9-70
- Address
  - GPIB 9-108
  - Switches 1-4
- Analog
  - Filter 9-45
- Analog Out
  - Enable 9-80
  - Scaling factor 9-80
- Aperture 9-87
- Arbitrary block data 9-112
- Arming 9-8
  - Bus arm mode 9-8
  - Delay by events 9-7,9-10
  - Event count range 9-7,9-10
  - External Events before Start 9-7,9-10
  - Start delay 9-7
  - Start slope 9-9
  - Start source 9-9
  - Stop slope 9-11
  - Stop source 9-11
  - Subsystem 9-5
  - Wait for bus 6-23
- Array 7-4 - 7-5,7-7
  - Fetch 9-35
- ASCII
  - Data format 7-8,9-38
  - Fixed format 9-39
- Attenuation 2-2,9-44
- Auto
  - Attenuation 9-44
  - Gated voltage mode 9-97
  - Levels selected by 9-49
  - Off 9-45,9-48
  - Once 9-46,9-49 - 9-50
  - Power on clearing 9-126
  - Sensitivity 9-46
  - Speed 9-92
  - Trigger 9-50,9-97
  - Trigger level 9-49
  - Trigger On/Off 9-49
  - Triggering 2-2
- Auto calibration on/off 9-24
- Average
  - For a preset time 9-91
  - Measurements 9-92
  - Mode 9-91
  - On/Off 2-4,9-92
  - Sample size 9-91
  - Samples 9-91
  - Samples in averaging 9-91
  - State 9-92
- Average or Single? 9-92

B

Back-to-back · · · · · · · · · · · · · · · · · 7-15
  Period · · · · · · · · · · · · · · · · · 7-9,7-13
Block arming · · · · · · · · · · · · · · · · 9-116
Block data · · · · · · · · · · · · 3-12,7-8,9-38
Block measurements · · · · · · · · · · · · 7-8
Boolean · 3-11
Burst
  Carrier Frequency · · · · · · · · · · · · 9-61
  Repetition Frequency · · · · · · · · · 9-62
  Synchronization (PM6685) · · · · · 9-96
Bus
  Drivers · 1-6
Bus Arm · 9-6
  Exit · 9-8
  Mode · 9-8
  On/Off · 9-8
  Override · · · · · · · · · · · · · · · · · · · 9-8
Bus initialization · · · · · · · · · · · · · · · 3-19

C

Calculate
  Block · 5-3
  Enable · 9-22
  Mathematics · · · · · · · · · · · 9-20 - 9-21
  Reading data · · · · · · · · · · · · · · · · 9-15
  Real time · · · · · · · · · · · · · · · · · · · 7-9
  Subsystem · · · · · · · · · · · · · · · · · · 9-13
Calibration · · · · · · · · · · 7-8,9-112,9-127
  Of Comparators · · · · · · · · · · · · · · 9-30
  Of Interpolators · · · · · · · · · · · · · · 9-24
  Subsystem · · · · · · · · · · · · · · · · · · 9-23
Change function fast
  MEAS:MEM1? · · · · · · · · · · · · · · · · 7-5
Channel
  List · 3-12
  Selecting · · · · · · · · · · · · · · · 6-13,9-93
Character data · · · · · · · · · · · · · · · · 3-12
Check · 2-3
Check Against Lower Limit · · · · · · 9-18
Check Against Upper Limit · · · · · · 9-19
Check signal · · · · · · · · · · · · · · · · · 9-114
Clear Status · · · · · · · · · · · · · · · · · 9-118
Clearing
  status registers · · · · · · · · · · · · · 9-126
CME-bit · · · · · · · · · · · · · · · · · 6-21,9-121
Colon · 3-8,3-10
Command
  Error · · · · · · · · · · · · · · 3-4,3-17,9-121
  Error (CME) · · · · · · · · · · · · · · · · · 6-21
  Header · 3-10
  Tree · 3-10
Command Error (CME)
  Code list · 8-2
Command tree · · · · · · · · · · · · · · · · · · 3-8
Commands · · · · · · · · · · · · · · · · · · · 3-20
  *CLS · · · · · · · · · · · · · · · · · · 3-20,9-118
  *DMC · · · · · · · · · · · · · 3-13,9-117,9-119
  *EMC · · · · · · · · · · · · · · · · · 3-14,9-120
  *ESE · · · · · · · · · · · · · · · · 9-117,9-121
  *ESR? · 9-122
  *GMC? · · · · · · · · · · · · · · · · 3-15,9-122
  *IDN? · 9-123
  *LMC · · · · · · · · · · · · · · · · 9-117,9-123
  *LMC? · 3-15
  *LRN? · 9-124
  *OPT? · 9-125
  *PMC · · · · · · · · · · · · · 3-14,9-117,9-126
  *PSC · · · · · · · · · · · · · · · · 9-117,9-126
  *RCL · 9-127
  *RST · 9-117
  *SRE · · · · · · · · · · · · · · · · · 6-17,9-130
  *STB? · · · · · · · · · · · · · · · · 9-117,9-131
  *TRG · · · · · · · · · · · · · · · 6-29,9-8,9-131
  *TST? · · · · · · · · · · · · · · · · 9-117,9-132
  *WAI · 9-132
  :ABORt · 9-4
  :ACQuisition:APERture · · · · · · · · 9-87
  :Acquisition:HOFF · · · · · · · · · · · · 9-88
  :Acquisition:HOFF:ECOunt · · · · · 9-88
  :Acquisition:HOFF:MODE · · · · · · 9-89

:Acquisition:HOFF:TIMe 9-89
:Acquisition:RESolution 9-90
:ARM 9-8
:ARM:LAYer2:SOURce 9-8
:ARM:SEQuence:LAYer1:COUNt 9-6
:ARM:SEQuence:LAYer1:DELay 9-7,9-10
:ARM:SEQuence:LAYer1:SOURce 9-9
:ARM:SEQuence1:LAYer1:ECOunt 9-7,9-10
:ARM:SEQuence1:LAYer1:SLOPe 9-9
:ARM:SEQuence2:SLOPe 9-11
:ARM:SEQuence2:SOURce 9-11
:ARM:STARt 9-8
:ARM:STARt:LAYer1:COUNt 9-6
:ARM:STARt:LAYer1:DELay 9-7,9-10
:ARM:STARt:LAYer1:ECOunt 9-7,9-10
:ARM:STARt:LAYer1:SLOPe 9-9
:ARM:STARt:LAYer1:SOURce 9-9
:ARM:STOP:SLOPe 9-11
:ARM:STOP:SOURce 9-11
:AVERage:COUNts 9-91
:AVERage:MODE 9-91
:AVERage:STATe 9-92
:CALCulate :AVERage :COUNt 9-14
:CALCulate:AVERage:STATe 9-15
:CALCulate:AVERage:TYPE 9-15
:CALCulate:DATA 9-15
:CALCulate:IMMediate 9-16
:CALCulate:LIMit[:STATe] 6-3,9-16
:CALCulate:LIMit:FAIL 9-17
:CALCulate:LIMit:LOWer:STATe 9-18
:CALCulate:MATH:STATe 9-21
:CALCulate:STATe 9-22
:CALibration:INTerpolator:AUTO 9-24
:COMMunicate:GPIB:ADDRess 9-108
:CONFigure 9-26
:CONFigure:ARRay 9-27
:CONFigure:DCYCle 9-59,9-66
:CONFigure:FREQuency 9-60
:CONFigure:FREQuency: BURSt :PRF 9-62
:CONFigure:FREQuency:BURSt :CARRier 9-61
:CONFigure:FREQuency:RATio 9-63
:CONFigure:FTIMe 9-63
:CONFigure:MAXimum 9-64
:CONFigure:MINimum 9-64
:CONFigure:NDUTycycle 9-66
:CONFigure:NWIDth 9-65
:CONFigure:PDUTycycle 9-66
:CONFigure:PERiod 9-67
:CONFigure:PHASe 9-67
:CONFigure:PTPeak 9-68
:CONFigure:PWIDth 9-65
:CONFigure:RTIMe 9-63,9-68
:CONFigure:TINTerval 9-69
:CONFigure:TOTalize:ACCumulated 9-70
:CONFigure:TOTalize:CONTinuous 9-71
:CONFigure:TOTalize:GATed 9-72
:CONFigure:TOTalize:GATed? 9-72
:CONFigure:TOTalize:SSTop 9-72
:CONFigure:TOTalize:TIMed 9-73
:DISPlay:ENABle 9-32
:FETCh:ARRay? 9-34 - 9-35
:FETCh? 9-34
:FORMat:FIXed 9-39
:FREQuency:RANGe:LOWer 9-92
:FUNCtion 9-93
:INITiate 9-42
:INITiate:CONTinuous 9-42
:INPut:ATTenuation 9-44
:INPut:COUPling 9-44
:INPut:FILTer 9-45
:INPut:IMPedance 9-47
:INPut:LEVel 9-47
:INPut:LEVel:AUTO 9-49
:INPut:SLOPe 9-51
:MEASure:ARRay? 9-57
:MEASure:DCYCle? 9-59,9-66

- :MEASure:FREQuency: BURSt :PRF? 9-62
- :MEASure:FREQuency:BURSt :CARRier? 9-61
- :MEASure:FREQuency:RATio? 9-63
- :MEASure:FREQuency? 9-60
- :MEASure:FTIMe? 9-63
- :MEASure:MAXimum? 9-64
- :MEASure:MEMory? 9-58
- :MEASure:MEMory<N>? 9-58
- :MEASure:MINimum? 9-64
- :MEASure:NDUTycycle? 9-66
- :MEASure:NWIDth? 9-65
- :MEASure:PDUTycycle? 9-66
- :MEASure:PERiod? 9-67
- :MEASure:PHASe? 9-67
- :MEASure:PTPeak? 9-68
- :MEASure:PWIDth? 9-65
- :MEASure:RTIMe? 9-63,9-68
- :MEASure:TINTerval? 9-69
- :MEASure:TOTalize:ACCumulated? 9-70
- :MEASure:TOTalize:CONTinuous? 9-71
- :MEASure:TOTalize:GATed? 9-70,9-72
- :MEASure:TOTalize:SSTop? 9-72
- :MEASure:TOTalize:TIMed? 9-73
- :MEASure? 9-56
- :MEMory:DELete:MACRo 9-76,9-128
- :MEMory:FREE:MACRo? 9-77
- :MEMory:FREE:SENSe? 9-76
- :MEMory:NSTates? 9-77
- :READ:ARRay? 9-83
- :READ? 9-82
- :ROSCillator:SOURce 9-96
- :SDELay 9-96
- :SENSe:Acquisition:APERture 9-87
- :SENSe:Acquisition:HOFF 9-88
- :SENSe:Acquisition:HOFF:ECOunt 9-88
- :SENSe:Acquisition:HOFF:MODE 9-89
- :SENSe:Acquisition:HOFF:TIMe 9-89
- :SENSe:Acquisition:RESolution 9-90
- :SENSe:AVERage:COUNts 9-91
- :SENSe:AVERage:MODE 9-91
- :SENSe:AVERage:STATe 9-92
- :SENSe:FREQuency:RANGe:LOWer 9-92
- :SENSe:FUNCtion 9-93
- :SENSe:ROSCillator:SOURce 9-96
- :SENSe:SDELay 9-96
- :SENSe:TOTalize:GATE 9-97
- :SENSe:VOLTage:GATed:STATe 9-97
- :STATus:DREGister0? 9-100
- :STATus:OPERation:CONDition? 9-101
- :STATus:OPERation:ENABle 9-102
- :STATus:QUEStionable:CONDition? 9-104
- :STATus:QUEStionable:ENABle 9-105
- :STATus:QUEStionable? 9-105
- :SYSTem:ERRor? 9-108
- :SYSTem:PRESet 9-109
- :SYSTem:SET 9-110
- :SYSTem:TIME:ELAPsed 9-110
- :SYSTem:TOUT 9-111
- :SYSTem:UNPRotect 9-111 - 9-112
- :TEST:SELect 9-114
- :TOTalize:GATE 9-97
- :TRIGger [:SEQuence1]:COUNt 9-116
- :TRIGger[:STARt]:COUNt 9-116
- :VOLTage:GATed:STATe 9-97
- RCL 9-117
- SOC? 9-101
- SOEn 9-102
- SOEv? 9-103

Common Commands 3-8,9-117

Common via A 2-2

Comparator Calibration 9-30

Configure 5-5 - 5-6,7-4,9-26
- Array 9-27

  - Description 6-9
  - Function 9-25,9-53
  - Scalar 9-26
- Continuous
  - Period measurements 7-9,7-15
- Continuously Initiated 9-42
- Control function 1-5
- Controller synchronization 7-2
- Conventions 1-3
- Coupling
  - See AC/DC
- Cutoff frequency 9-45
- CW 9-61

D

- Data
  - Recalculate 9-16
- Data format 7-8
  - Internal 9-95
- Data Format 9-39
- Data Type .... 9-38 - 9-39
- DC coupling
  - See AC/DC
- DCL 3-19
- DDE-bit 6-21,9-121
- Deadlock 3-5
- Decimal data 3-11
- Declaration
  - For GPIB interface 4-3
- Default 2-3,9-128
  - Presetting the counter 9-109
- Deferred commands 3-5
- Define Macro 9-119
- Delay
  - After external start arming 9-7,9-10
  - After External Stop Arming 9-11
- Delete one Macro 3-15,9-76,9-128
- Device clear 1-6,3-19
- Device dependent Error (DDE) 6-21,9-121
- Device initialization 3-19
- Device Setup 9-124
- Device specific errors 3-4,3-18,8-13
  - Standardized 8-11
- Device Status 9-130
- Device Status Register
  - Enable 9-100
  - Event 9-100
  - No. 0 9-100
- Device Trigger, ... 1-6
- Diagnostics Subsystem 9-29
- Display
  - Enable 2-3,9-32
  - Off 9-32
  - On 9-32
  - State 9-32
  - Subsystem 9-31
- Double Precision floating point format 9-38
- Double quotes 3-12
- DREG0 9-131
- Dumping measurement results 7-9
- Duration
  - See Pulse width
- Duty cycle measurements 9-59
- Duty factor 9-95

E

- EAV .... 6-16,9-108,9-130 - 9-131
- Enable
  - Analog Out 9-80
  - Calculation 9-22
  - Display 9-32
  - Macros 9-120
  - Mathematics 9-21
  - Monitoring of Parameter Limits 9-16
  - Service Request 9-130
  - Standard Event Status 9-121
  - Statistics 9-14
- Error
  - ASCII description 9-108
  - Available 9-130

Clearing queue · · · · · · · · · · · · · 9-118
Command · · · · · · · · · · · · · · · · · · · 8-2
Device specific, code list· · · · · · · 8-13
Escape from condition· · · · · · · · · 3-19
Execution· · · · · · · · · · · · · · · · · · · 8-7
In self test · · · · · · · · · · · · · · · · · 9-132
Message available· · · · · · · · · · · · 6-16
Query, code list · · · · · · · · · · · · · · 8-12
Queue · · · · · · · · · · · · · · 3-17,6-16,8-2
Reporting · · · · · · · · · · · · · · · · · · · 3-17
Standardized device specific list · 8-11
Standardized numbers · · · · · · · · 3-17
ESB · · · · · · · · · · · · 9-121,9-130 - 9-131
Escape from erroneous conditions· 3-19
Event
Clearing registers · · · · · · · · · · · 9-118
Detection · · · · · · · · · · · · · · · · · · · 6-28
Read Device Status Event Register · 9-100
Status bit · · · · · · · · · · · · · 9-121,9-130
Status Register · · · · · · · · · · · · · 9-122
Events
Before Start Arming, External 9-7,9-10
Hold Off · 9-88
Example language · · · · · · · · · · · · · · 1-4
EXE-bit · · · · · · · · · · · · · · · · · 6-21,9-121
Execution
Control · 3-4
Error · · · · · · · · · · 3-4,3-18,6-21,9-121
Error code list · · · · · · · · · · · · · · · · 8-7
Expression · · · · · · · · · · · · · · 9-20 - 9-21
data · 3-12
Ext. ref. · 9-96
External Events before Start Arming · 9-7,9-10
External reference · · · · · · · · · · 2-3,9-96
on/off · 2-3

F

Fail
Limit · 9-17
Fall time · 9-91
Measurements· · · · · · · · · · · · · · · 9-63
Fast
Autotrigger· · · · · · · · · · · · · · · · · · 9-92
Period measurements · · · · · · · · · · 7-9
Recall and measure · · · · · · · · · · · 7-5
Fetch · 5-6,7-4
An Array of Results · · · · · · · · · · · 9-35
Array · 9-35
Calcutated Data· · · · · · · · · · · · · · 9-15
Description· · · · · · · · · · · · · · · · · · 6-10
Function · 9-33
One Result· · · · · · · · · · · · · · · · · · 9-34
Several measurement results · · · 9-35
Filter · 9-45
On/off · 2-2
Fixed data format · · · · · · · · · · · · · · 9-39
Fixed Trigger Level · · · · · · · · · · · · · 9-47
Format
Internal data · · · · · · · · · · · · · · · · · 9-95
Response Data · · · · · · · · · · · · · · 9-39
Subsystem· · · · · · · · · · · · · · · · · · 9-37
Formula
Mathematics · · · · · · · · · · · 9-20 - 9-21
Macro · · · · · · · · · · · · · · · · · · 9-76 - 9-77
Freerun · 9-42
Frequency
Low limit for volt/autotrig · · · · · · · 9-92
Measurement· · · · · · · · · · · · · · · · 9-60
Ratio measurements · · · · · · · · · · 9-63
Front panel memories· · · · · · · · · · 9-129
Function · 2-4
Change fast · · · · · · · · · · · · · · · · · · 7-5

G

Gate
On/Off · 9-97
Time · 9-87
Gate time · 9-87
Gated by Y
Accumulated, Totalize X · · · · · · · 9-70

Totalize X 9-72
Gated Voltage Measurement 9-97
GET 6-29,7-6,9-8,9-131
Get Macro 9-122
GPIB Address 1-4,9-108
Group Execute Trigger 9-131

H

Header path 3-10
Header separator 3-8
High Speed Period Measurements 7-9
High Speed Voltage Measurements 9-92
Hold Off 2-3
Event count 9-88 - 9-89
Event range 9-88
Events 9-88
Mode 9-89
On/Off 9-88
Setting time 9-89
Time 9-89
Time mode 9-89
Time range 9-89
Hysteresis 9-45

I

dentification query 9-123
dle state 6-28
FC 3-19
mmediate mode 9-8
mpedance 2-2,9-47
nitiate 3-19 - 3-20,5-6,7-4
Continuous 6-28,7-4
Continuously 7-6,9-42
Description 6-10
Immediate 6-28
Measurement 9-42
Subsystem 9-41
nitiated state 6-28
nput
AC/DC 9-44
Attenuation 2-2,9-44
Auto trigger 2-2
Common 2-2
Coupling 2-2,9-44
Filter 2-2
Impedance 2-2,9-47
Selecting 6-13
Selecting channel 9-93
Slope 2-2
Subsystems 9-43
Swap 2-2
Trigger level 2-2
INPut block 5-3
Instrument model 5-2
Interface clear 3-19
Internal format 9-95
Internal reference 2-3,9-96
Interpolators
Calibration of 9-24
Interrupted 3-5
Interval, Time 9-69

K

K, L and M 9-20 - 9-21
Keywords 3-11

L

Leading zeroes 9-39
Leaf node 3-10
Learn Device Setup 9-124
Learn Macro 9-123
Level
Fixed trigger 9-47
Set Trigger 9-48
Limit
Check lower 9-18
Check Upper 9-19
Enable 9-16
Enable monitoring 9-22
Example 4-4,4-14
Fail 9-17
Monitoring 6-26

Passed · · · 9-100
Set lower · · · 9-17
Set upper · · · 9-18
Listener function · · · 1-6
Local · · · 1-5
Control · · · 1-5
Lockout · · · 3-6
Operation · · · 3-6
Long form · · · 3-8
Low Pass Filter · · · 9-45
Lower case · · · 3-8
Lower Limit
Check · · · 9-18
Fail · · · 9-17
Set · · · 9-17

M

Macro · · · 3-13 - 3-15
Data types · · · 3-13
Define · · · 9-119
Delete · · · 3-15,9-76,9-128
Delete all · · · 9-126
Description · · · 3-13
Enable · · · 9-120
Get · · · 9-122
How to execute · · · 3-14
Learn · · · 9-123
Memory states · · · 9-77
Names · · · 3-13
Purge · · · 9-126
Mathematics · · · 2-5
Enable · · · 9-21 - 9-22
Select expression · · · 9-20 - 9-21
MAV · · · 3-19,9-130 - 9-131
MAX · · · 3-11,9-15
MEAN · · · 9-15
Measure · · · 5-5,7-5 - 7-6
Array · · · 9-57
Description · · · 6-9
Functions · · · 9-59
Once · · · 9-56
Scalar · · · 9-56
Volt neg. peak · · · 9-64
Volt peak · · · 9-64
Measurement
Abort · · · 9-4
Basic method · · · 7-6
Continuously initiated · · · 9-42
Fetch Results · · · 9-35
Function · · · 5-5
Gated Voltage · · · 9-97
High Speed Voltage · · · 9-92
Initiate · · · 9-42
No. of, on ext arm start · · · 9-116
No. on each bus arm · · · 9-6
Start sync. · · · 7-2
Started · · · 7-2
Started (MST) · · · 6-23
Status · · · 9-101 - 9-102
Stop sync. · · · 7-2
Stopped (MSP) · · · 6-23
Terminate · · · 7-4
Time · · · 9-87
Time, range · · · 9-87
Time, setting · · · 9-87
Trigger · · · 9-131
Measurement Function · · · 9-53
Measurement Time · · · 9-87
Setting · · · 9-87
Measuring
Burst CW · · · 9-61
Duty Cycle · · · 9-59,9-66
Fall Time · · · 9-63
Frequency · · · 9-60
Frequency ratio · · · 9-63
Input selection · · · 6-13
Period · · · 9-67
Phase · · · 9-67
PRF · · · 9-62
Pulse width · · · 9-65
Rise Time · · · 9-63,9-68
Select function · · · 9-26

  - Selecting function ············ 9-93
  - Terminate ··················· 9-4
  - Time-Interval ················ 9-69
  - Totalize X gated by Y ·········· 9-72
  - Transition time ··········· 9-63,9-68
- Measuring time ················ 2-4
  - Range ···················· 9-87
- Memory ···················· 2-5
  - Fast ····················· 9-58
  - Free for Macros ········ 9-76 - 9-77
  - Recall and measure fast ···· 7-5,9-58
  - Subsystem ················· 9-75
- Message
  - Available ················· 9-130
  - Exchange Control ············ 3-4
  - exchange initialization ········ 3-19
  - terminator ·················· 3-5
- MIN ······················· 9-15
- Mnemonic conventions ··········· 1-3
- Mnemonics ·················· 3-8
- Mode, Hold Off ················ 9-89
- Monitor
  - Limits ················· 6-3,9-16
  - Monitor of low limit ··········· 6-26
  - of high limit ················ 6-26
- MSP-bit ···················· 6-23
- MSS ······················ 9-131
- MST-bit ···················· 6-23
- Multiple measurements
  - See Array
- Multiple queries ················ 3-9

N

- Negative slope ················ 9-51
- Non-decimal data ·············· 3-12
- Notation habit ················ 3-9
- NRf ······················· 3-11
- Nulling ················ 9-20 - 9-21
- Numeric data ················· 3-11
- Numeric expression data ········ 3-12

O

- OFL-bit ···················· 6-24
- On/Off, Hold Off ··············· 9-88
- On-time Read ················ 9-110
- OPC-bit ···················· 6-21
- Operation
  - Complete ·················· 9-124
  - Complete (OPC) ········ 6-21,9-121
  - Complete Query ············ 9-125
- Operation Status
  - Bit ······················ 9-130
  - Bits in register ··············· 6-23
  - Condition ················· 9-101
  - Enable ··················· 9-102
  - Event ···················· 9-103
  - Group Overview ············· 6-22
- OPR ················· 9-130 - 9-131
- Optional nodes ················ 3-10
- Options
  - Identification ·············· 9-125
- Output queue ················ 6-16
- Output Subsystem ············· 9-79
- Overflow ···················· 6-24
  - Message ·················· 3-17
  - Status ··················· 9-104
- Override
  - Bus Arm ···················· 9-8

P

- Packed
  - Data format ················ 9-95
- Parallel poll, ················· 1-6
- Parameter list ················ 3-12
- Parenthesis ·················· 3-12
- Parser ······················ 3-4
- Peak-to-Peak
  - Voltage ·················· 9-68
- Period measurements ·········· 9-67
  - Back-to-back ················ 7-9
- Phase ····················· 9-67

- Pmt 3-7,3-10
- PON-bit 6-21,9-121
- Positive slope 9-51
- Power On 6-21,9-121
  - Status Clear 9-126
- Preset 2-3,9-109
  - Status at power on 9-126
  - Status registers 9-103
- Preset Time, totalize X-Y During a 9-73
- PRF 9-62
  - Synchronization (PM6685) 9-96
- Program message terminator 3-7,3-10
- Program messages 3-7
- Protected User Data 9-127
- Pulse
  - Repetition Frequency 9-62
  - Width 9-65,9-91
- Purge Macro 9-126

Q

- QUE 9-130 - 9-131
- Query
  - Error 3-4 - 3-5,3-18,6-21,8-12,9-121
  - Multiple 3-9
- Questionable Data/signal 9-130
  - Condition 9-104
  - Enable 9-105
  - Event 9-105
  - Status group 6-24
- Quotes 3-12
- QYE-bit 6-21,9-121

R

- Ratio 9-63
- Read 5-6,7-5 - 7-6
  - Array 9-83
  - Function 9-81
  - One Result 9-82
  - Scalar 9-82
- Read or Send Settings 9-110
- Real
  - Data format 7-8,9-38,9-95
- Recalculate Data 9-16
- Recall 2-5,7-5,9-58,9-127
- Reference
  - Selection 9-96
- REMOTE 1-5
- Remote operation 3-6
- Remote/local 1-6
- Remove
  - All macros 9-126
  - One macro 9-76,9-128
- Repetition 1-3
- Request Control (RQC) 6-21,9-121
- Request Service 9-130
- Reset 2-3,3-19,9-109,9-128
- Resolution 7-8,9-90
- Response
  - Data 3-9
  - Data Format 9-39,9-43
  - Data Type 6-7 - 6-8,9-38
  - Message 3-5
  - Message terminator 3-9
  - Messages 3-7
- Result
  - Fetch one 9-34
  - Reading 9-82
- Retrieve
  - Front panel setting 9-127
  - Measurement data 7-4
  - Measurement result 9-34
- Rise Time 9-91
  - Measurements 9-68
  - Trigger levels 9-49
- Rmt 3-9
- Root level 3-8
- Root node 3-8
- RQC-bit 6-21,9-121
- RQS 9-130
- RST 3-20

S

- Sample Size for Average 9-91
- Sample Size for Statistics 9-14
- Save 2-5,9-129
- Scaling Factor
  - Analog Output 6-11,9-80
- SCPI 3-2
  - Compliance of instrument 9-112
- SDC 3-19
- SDEV 9-15
- Select Mathematical Expression 9-20 - 9-21
- Selective device clear 3-19
- Self Test
  - Activate 9-132
  - Select 9-114
- Semicolon 3-8
- SEND 1-4
- SENSe block 5-3
- Sense Command Subsystem 9-85
- Sensitivity 9-45
  - Trigger level 9-45
- Sequential commands 3-5
- Service Request 3-17
  - Capability 1-6
  - Enable 9-130
- Set
  - Lower Limit 9-17
  - Upper Limit 9-18
- Settings
  - Reading 9-110
- Short form 3-8
- Signal Detection 9-109
- Single 2-4
  - Measurements 9-90
  - On/Off 9-92
  - Or Average? 9-92
- Single quotes 3-12
- Slope 2-2,9-51
  - Arming start 9-9
  - Stop arming 9-11
- Source
  - Start arming 9-9
  - Stop arming 9-11
- Speed
  - Autotrigger 9-92
  - Individually sync. meas. 7-6
  - Resolution 9-90
  - Summary 7-11
  - Voltage measurements, high 9-92
- Standard deviation 9-15
- Standard Event Status
  - Enable 9-121
- Standard event status register 6-21
- Standardized Device specific errors 8-11
- Standardized Error numbers 3-17,8-2
- Start arming
  - Delay 9-7,9-10
  - Delay by events 9-7,9-10
  - External Events before 9-7,9-10
  - Slope 9-9
- Start measurement 9-42
  - Sync. 7-2
- Start source
  - Arming 9-9
- Start/stop
  - Totalize 9-97
- Start/stop by Y, totalize X 9-72
- Statistics 2-5
  - Enable 9-14
  - Recalculating data 9-16
  - Sample size 9-14
  - Type 9-15
- Status
  - Clear 9-118
  - Clear data structures 3-20
  - Enable reporting 9-103
  - Enabling Standard Event Status 9-121
  - Event Status Register 9-122
  - Limit monitor 9-100
  - Measurement started 9-102

Measurement stopped · · · · · · · 9-102
Operation event· · · · · · · · · · · · · 9-103
Overflow · · · · · · · · · · · · · · · · · · 9-104
Preset · 9-103
Questionable Data/signal · · · · · 9-104
Questionable Data/signal, Event · 9-105
Register structure · · · · · · · · · · · · 3-16
Subsystem· · · · · · · · · · · · · · · · · · 9-99
Timeout · · · · · · · · · · · · · · · · · · · 9-104
Unexpected parameter · · · · · · · 9-104
Using the reporting · · · · · · · 3-16,6-14
Waiting for bus arming · · · · · · · 9-102
Waiting for triggering· · · · · · · · · 9-102
Status byte · · · · · · · · · · · · · · · 3-16,6-14
Bit 0 · 9-100
Bit 2 · 6-16
Bit 3· · · · · · · · · · · · · · · · 9-104 - 9-105
Bit 5· · · · · · · · · · · · · · · · 9-121 - 9-122
Bit 6 · 9-131
Bit 7· · · · · · · · · · · · · · · · 9-101 - 9-103
Query· 9-131
Reading· · · · · · · · · · · · · · · 6-16,9-131
Status reporting · · · · · · · · · · · 3-16,6-14
Stop Arming
Slope · 9-11
Source · 9-11
Stop Measurement · · · · · · · · · · · · · · 7-4
Sync. · 7-2
Store
Front panel settings· · · · · · · · · · 9-129
String data· · · · · · · · · · · · · · · · · · · 3-12
Subnodes · 3-8
Suffixes · 3-11
Summary
Instrument states· · · · · · · · · · · · · · 7-3
Measurement commands · · · · · · 6-10
Of command syntax · · · · · · · · · · 3-14
Of input amplifier settings· · · 6-7 - 6-8
Speed · · · · · · · · · · · · · · · · 7-11 - 7-16
Swap A-B · 2-2
Synchronization · · · · · · · · · · · · · 7-2,7-6
Syntax
and Style · 3-7
Summary · · · · · · · · · · · · · · · · · · · 3-14
System Subsystem· · · · · · · · · · · · · 9-107
System Version · · · · · · · · · · · · · · · · 9-112

T

Talker function· · · · · · · · · · · · · · · · · · 1-5
tatus
See Message Data Type
Terminate
Measurement · · · · · · · · · · · · · 7-4,9-4
Terminator· 3-8
50ohms/1Mohm· · · · · · · · · · · · · · 9-47
Test
Activating· · · · · · · · · · · · · · · · · · 9-132
Selecting internal self-test· · · · · 9-114
signal · 9-114
Subsystem· · · · · · · · · · · · · · · · · 9-113
Time
Hold Off · 9-89
Interval · · · · · · · · · · · · · · · · 9-69,9-91
Measure Rise · · · · · · · · · · · · · · · · 9-68
Read Elapsed · · · · · · · · · · · · · · · 9-110
Rise/fall · 9-91
Selecting Measurement Time · · · 9-87
Totalize X-Y During a Preset· · · · 9-73
Time out
For measurement (TIO) · · · · · · · 6-24
Timebase
External/internal · · · · · · · · · · · · · 9-96
Timeout · 7-2
On/Off· 9-111
Range· 9-111
Set · 9-111
Status· 9-104
TIO-bit· 6-24
Totalize
X-Y Manually · · · · · · · · · · · · · · · 9-71
Gate On/Off· · · · · · · · · · · · · · · · · 9-97

- Start/stop 2-3
- X gated by Y 9-72
- X gated by Y, accumulated 9-70
- X start/stop by Y 9-72

ransition time 9-91

rigger 6-29
- See Also Command: *TRG
- No. of, on ext arm start 9-116
- Slope 9-51
- Subsystem 9-14,9-115

rigger level 2-2
- Auto 9-49
- Fixed 9-45,9-47 - 9-48
- Set 9-45,9-48

runcation rules 1-3

ype, Statistical 9-15

U

EP-bit 6-24,9-104

nexpected parameter (UEP) 6-24
- Status 9-104

nit separator 3-8

nprotect 9-112

nterminated 3-5

pper case 3-8

pper Limit
- Check 9-19
- Fail 9-17
- Set 9-18

RQ-bit 6-21,9-121

ser data 9-112

ser request (URQ) 6-21,9-121

V

ariable hysteresis
- Auto levels 9-49

Volt
- Gated Measurements 9-97
- High Speed Measurements 9-92
- Negative Peak 9-64
- Peak 9-64
- Peak-to-Peak 9-68

W

WAI 5-4

Wait for bus arming (WFA) 6-23

Waiting for bus arming
- Status 9-102

Waiting for trigger and/or ext. arming (WFT) 6-23

Waiting for triggering
- Status 9-102

Wait-to-continue 9-132

Waveform compensation 9-48

WFA-bit 6-23

WFT-bit 6-23

X

X 3-8

X gated by Y, accumulated, totalize 9-70

X gated by Y, totalize 9-72

X start/stop by Y, totalize 9-72

X1/X10 attenuation 9-44

Xn-1 2-5

XOLD 9-21

X-Y During a Preset Time, totalize 9-73

Y

Y, accumulated, totalize X gated by 9-70

Y, totalize X gated by 9-72

Y, totalize X start/stop by 9-72