TABOR ELECTRONICS

PROTEUS
Infinite possibilities

Proteus Series Arbitrary Waveform Transceiver Programming Manual

Rev. 1.4

Warranty Statement

Products sold by Tabor Electronics Ltd. are warranted to be free from defects in workmanship or materials. Tabor Electronics Ltd. will, at its option, either repair or replace any hardware products which prove to be defective during the warranty period. You are a valued customer. Our mission is to make any necessary repairs in a reliable and timely manner.

Duration of Warranty

The warranty period for this Tabor Electronics Ltd. hardware is one year, except software and firmware products designed for use with Tabor Electronics Ltd. Hardware is warranted not to fail to execute its programming instructions due to defect in materials or workmanship for a period of ninety (90) days from the date of delivery to the initial end user.

Return of Product

Authorization is required from Tabor Electronics before you send us your product for service or calibration. Call your nearest Tabor Electronics support facility. A list is located on the last page of this manual. If you are unsure where to call, contact Tabor Electronics Ltd. Tel Hanan, Israel at 972-4-821-3393 or via fax at 972-4-821-3388. We can be reached at: support@tabor.co.il

Limitation of Warranty

Tabor Electronics Ltd. shall be released from all obligations under this warranty in the event repairs or modifications are made by persons other than authorized Tabor Electronics service personnel or without the written consent of Tabor Electronics.

Tabor Electronics Ltd. expressly disclaims any liability to its customers, dealers and representatives and to users of its product, and to any other person or persons, for special or consequential damages of any kind and from any cause whatsoever arising out of or in any way connected with the manufacture, sale, handling, repair, maintenance, replacement or use of said products. Representations and warranties made by any person including dealers and representatives of Tabor Electronics Ltd., which are inconsistent or in conflict with the terms of this warranty (including but not limited to the limitations of the liability of Tabor Electronics Ltd. as set forth above), shall not be binding upon Tabor Electronics Ltd. unless reduced to writing and approved by an officer of Tabor Electronics Ltd. This document may contain flaws, omissions, or typesetting errors. No warranty is granted nor liability assumed in relation thereto. The information contained herein is periodically updated and changes will be incorporated into subsequent editions. If you have encountered an error, please notify us at support@taborelec.com. All specifications are subject to change without prior notice. Except as stated above, Tabor Electronics Ltd. makes no warranty, express or implied (either in fact or by operation of law), statutory or otherwise; and except to the extent stated above, Tabor Electronics Ltd. shall have no liability under any warranty, express or implied (either in fact or by operation of law), statutory or otherwise.

Proprietary Notice

This document and the technical data herein disclosed, are proprietary to Tabor Electronics, and shall not, without express written permission of Tabor Electronics, be used, in whole or in part to solicit quotations from a competitive source or used for manufacture by anyone other than Tabor Electronics. The information herein has been developed at private expense and may only be used for operation and maintenance reference purposes or for purposes of engineering evaluation and incorporation into technical specifications and other documents, which specify procurement of products from Tabor Electronics.

Document Revision History

Revision	Date	Description	Authors
1.4	2-Jul-2023	• 13.4.1 Programming Example 3 – Updated. • 13.5 Using the Digitizer to Capture Baseband and RF Signals – New.	Joan Mercade
1.3	24-May-2023	• Release supporting WDS 1.7.050 or higher, SCPI Rev. 1.132, FPGA version 236 or higher. • Minor editing of text. • Table 1-1 SCPI Common Mode Commands – New. • 3.5 :XINStrument:MODE(?) – Changed to query only. • :XINStrument:SYNChronize:ROLE(?) – New. • 3.7 :XINStrument:SYNChronize:FOLLowers <number_of_follower_instruments>(?) – New. • 3.12 :XINStrument:SYNChronize:STATe{OFF|ON|0|1}(?) – Removed. • 4 Run Mode Commands – Minor corrections. • Figure 4-1 Standard Trigger – New. • 4.3 :TRIGger:SOURce:ENABle{NONE|TRG1|TRG2|TRG3| TRG4|TRG5|TRG6|INTernal |CPU|FBTRg|HWControl}(?) – Added TRG3 -TRG6. • 4.4 :TRIGger:SOURce:DISable{NONE|TRG1|TRG2|TRG3| TRG4|TRG5|TRG6|INTernal |CPU|FBTRg|HWControl}(?) – Added TRG3 -TRG6. • 4.5 :TRIGger[:ACTIVE]:SELect{TRG1|TRG2|TRG3|TRG4| TRG5|TRG6|INTernal}(?) – Added TRG3 -TRG6. • 4.7 :TRIGger:CPU:MODE{LOCAL|GLOBAL}(?) – New. • Figure 4-2 Gate Trigger – New. • 4.9 :TRIGger:LEVel<level>(?) – Updated description. • Figure 4-3 Trigger Width – New. • 4.15 :TRIGger:MODE{EVENtually|IMMediate}(?) – Updated description. • Figure 4-4 Trigger Holdoff – New. • 5.4 [:SOURce]:INTerpolation{ NONE|X2|X4|X8}(?) – Updated. • 5.10 [:SOURce]:FREQuency[:RASTer]{<sclk>|MINimum| MAXimum}(?) – Updated. • 6 Marker Output Commands – Added text “Marker data is transferred …”. • 7.9 :TASK:COMPoser[:DEFine]:ENABle{NONE|TRG1|TRG2|TRG3|TRG4|TRG5|TRG6|INTernal|CPU|FBTRg| ANY}(?) – Added TRG3 -TRG6. • 7.10 :TASK:COMPoser[:DEFine]:ABORt{ NONE|TRG1|TRG2|TRG3|TRG4|TRG5|TRG6| INTernal|CPU|FBTRg|ANY }(?) – Added TRG3 -TRG6. • 7.17 :TASK:COMPoser[:DEFine]:DTRigger{OFF|ON|0|1}(?) – Updated description. • 7.22 :TASK:DATA [<offset>]#<header><binary_block> – Updated description. • 7.21 :TASK: SYNC – New. • 9 Arbitrary Waveform Commands – • Arbitrary Memory Management – New text about banks. • Short and Fast Segments – New text.	Jakob Apelblat

		• 9.1 :TRACe[:DATA](?) [<offset>]#<header><binary_block> – New text "The optional offset parameter …" • 9.25 :TRACe:FRAG? – Updated description. • 10.8 :DIGitizer:DDC:DECimation{ NONE | X1| X4 | X16}? – Added command parameters. • 10.18:DIGitizer:ACQuire:STATus? — Changed from DIGitizer:ACQuire[:FRAMes]:STATus? to :DIGitizer:ACQuire:STATus?. • 10.37 :DIGitizer:TRIGger:DELay[:EXTernal]<delay_time>(?) – Changed EXTernal to [EXTernal]. • 10.48 :DIGitizer:DATA:FORMat { <U16 | F32 | F64}(?) – New. • 10.50 :DIGitizer:LOOPback:DELay< delay>(?) – New. • 10.51 :DIGitizer:LOOPback: SYNC – New. • 10.52 :DIGitizer:LOOPback:IQRotation< scale>,<phase>(?) – New. • 10.53 :DIGitizer:LOOPback:OVERflow(?) – New. • 11.1 Introduction – Updated. • Table 11-1 Possible Data Storage Configurations – Updated. • 11.2 :DSP:STORe{ DIRect | DSP | FFTOut }(?) – Updated. • 11.3 :DSP:IQDemod:SELect{ DBUG | IQ4 | IQ5 | IQ6 | IQ7 | IQ8 | IQ9 | IQ10 | IQ11 | IQ12 | IQ13 }(?) – Updated. • 11.11 :DSP:FFT:INPut{ IQ1 | IQ2 |DBUG }(?) – Updated. • 11.12 :DSP:MATH:OPERation{ MI1 | MQ1 | MI2 | MQ2 | MI3 | MQ3 |MI4 |MQ4 | MI5 |MQ5 | MI6 |MQ6 | MI7 | MQ7 | MI8|MQ8 | MI9 |MQ9 | MI10 |MQ10 ,<SCALe>,<OFFSet> }(?) – Updated. • 11.16 :DSP:MATH:RAVG { MI1 | MQ1 | MI2 | MQ2 | MI3 | MQ3 |MI4 |MQ4 | MI5 |MQ5 | MI6 |MQ6 | MI7 | MQ7 | MI8|MQ8 | MI9 |MQ9 | MI10 |MQ10 |XC,<N> }(?) – Updated • 11.17 :DSP:DECision[:FEEDback]:MAPping{ <awg channel number>,DEC1 | DEC2| DEC3 | DEC4 | DEC5 | DEC6 | DEC7 | DEC8 | DEC9 | DEC10 | XC } (?) – Updated. • 11.20 :DSP:DECision:IQPath:SELect { DSP1 | DSP2 | DSP3 | DSP4| DSP5| DSP6| DSP7| DSP8| DSP9| DSP10 }(?) – Updated. • 12.13 :SYSTem:INFormation:PARSer:VERSion? – Removed. • 13 Appendix Proteus SCPI MATLAB Script Examples – New examples.	
1.2	15-Dec-2021	• 10.7 :DIGitizer:DDC:MODE{REAL|COMPlex}(?) – New DDC commands sub-group 10.7 to 10.12 • 10.19 :DIGitizer:ACQuire:AVERage:STATe{ OFF | ON |0 |1 }(?) - New • 10.20 :DIGitizer:ACQuire:AVERage:COUNt<# frames to average> – New • 11 Digital Signal Processing Commands – New command group	Jonathan Netzer
1.1	26-May-2021	• Changed from '[<offset>,]#<binary header><binary-block>' to '[<offset>]#<binary header><binary-block>' • Table 1-2 SCPI Syntax and Styles – New. • 7.22 :TASK:DATA [<offset>]#<header><binary_block> – Added binary-block description. • 10.5 :DIGitizer:CHANnel:RANGe{ HIGH|MEDium|LOW}(?) – Changed MAX to HIGH and MIN to LOW. • 10.36 :DIGitizer:TRIGger:HOLDoff< holdoff_time>(?) – Changed range to 0 to 16382 and removed TBD • 13 Appendix Proteus SCPI MATLAB Script – New.	Jakob Apelblat
1.0	15-Mar-2021	• Original external release supporting WDS 1.4.740, SCPI Rev. 1.100, FPGA version 1.116.0(rc) or later.	Jonathan Netzer

			Joan Mercade Jakob Apelblat

Acronyms & Abbreviations

Acronym	Description
µs or us	Microseconds
ADC	Analog to Digital Converter
AM	Amplitude Modulation
ASIC	Application-Specific Integrated Circuit
ATE	Automatic Test Equipment
AWG	Arbitrary Waveform Generator
AWT	Arbitrary Waveform Transceiver
BNC	Bayonet Neill–Concelm (coax connector)
BW	Bandwidth
CW	Carrier Wave
DAC	Digital to Analog Converter
dBc	dB/carrier. The power ratio of a signal to a carrier signal, expressed in decibels
dBm	Decibel-Milliwatts. E.g., 0 dBm equals 1.0 mW.
DDC	Digital Down-Converter
DDS	Direct Digital Synthesis
DHCP	Dynamic Host Configuration Protocol
DSO	Digital Storage Oscilloscope
DUC	Digital Up-Converter
ENoB	Effective Number of Bits
ESD	Electrostatic Discharge
EVM	Error Vector Magnitude
FBTRg	FeedBack Trigger (from the digitizer)
FIR	Finite Impulse Response (filter)
FPGA	Field-Programmable Gate Arrays
GHz	Gigahertz
GPIB	General Purpose Interface Bus
GS/s	Giga Samples per Second
GUI	Graphical User Interface
HDMI	High-Definition Multimedia Interface
HP	Horizontal Pitch (PXIe module horizontal width, 1 HP = 5.08mm)
Hz	Hertz
IF	Intermediate Frequency
I/O	Input / Output
IP	Internet Protocol
IQ	In-phase Quadrature
IVI	Interchangeable Virtual Instrument
JSON	JavaScript Object Notation
kHz	Kilohertz
LCD	Liquid Crystal Display
LO	Local Oscillator
MAC	Media Access Control (address)
MDR	Mini D Ribbon (connector)

Acronym	Description
MHz	Megahertz
ms	Milliseconds
NCO	Numerically Controlled Oscillator
ns	Nanoseconds
PC	Personal Computer
PCAP	Projected Capacitive Touch Panel
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PRBS	Pseudorandom Binary Sequence
PRI	Pulse Repetition Interval
PXI	PCI eXtension for Instrumentation
PXIe	PCI Express eXtension for Instrumentation
QC	Quantum Computing
Qubits	Quantum bits
RADAR	Radio Detection And Ranging
R&D	Research & Development
RF	Radio Frequency
RMS	Root Mean Square
RT-DSO	Real-Time Digital Oscilloscope
s	Seconds
SA	Spectrum Analyzer
SCPI	Standard Commands for Programmable Instruments
SFDR	Spurious Free Dynamic Range
SFP	Small Form-Factor Pluggable
SFP	Software Front Panel
SMA	Subminiature version A connector
SMP	Subminiature Push-on connector
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory
TFT	Thin Film Transistor
T&M	Test and Measurement
TPS	Test Program Sets
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
VCP	Virtual COM Port
Vdc	Volts, Direct Current
VISA	Virtual Instrument Software Architecture
V p-p	Volts, Peak-to-Peak
VSA	Vector Signal Analyzer
VSG	Vector Signal Generator
WDS	Wave Design Studio

Contents

6.1 :MARKer:SELect{1|2|3|4}(?) 94
6.2 :MARKer[:STATe]{OFF|ON|0|1}(?) 94
6.3 :MARKer:DELay:COARse <delay>(?) 95
6.4 :MARKer:DELay:FINE<delay>(?) 95
6.5 :MARKer:VOLTage:LEVel <gain>(?) 96
6.6 :MARKer:VOLTage:PTOP<ptop_level>(?) 96
6.7 :MARKer:VOLTage:OFFSet<offset>(?) 97
6.8 :MARKer:DATA [<offset>]#<header><binary_block>(?) 97
6.9 :MARKer:MEMory <offset_in_bytes>,#<header><marker-data>(?) 98
6.10 :MARKer:FILE[:NAMe]{<#<header><binary_block>} 99
6.11 :MARKer:FILE:OFFSet< start-offset inside the file>(?) 99
6.12 :MARKer:FILE:DESTination < SEGMent | MEMory>(?) 99
6.13 :MARKer:FILE:LOAD [[<offset>,]<size>] 100
6.14 :MARKer:FILE:STORe [[<offset>,]<size>] 100

7 Task Commands 102

7.1 :TASK:COMPoser:LENGth<length>(?) 102
7.2 :TASK:COMPoser:SELect<task_#>(?) 102
7.3 :TASK:COMPoser[:DEFine]:TYPE{SINGle|STARt|END| SEQ}(?) 102
7.4 :TASK:COMPoser[:DEFine]:LOOPs<task_loops>(?) 103
7.5 :TASK:COMPoser[:DEFine]:SEQuence<seq_loops>(?) 104
7.6 :TASK:COMPoser[:DEFine]:SEGMent(?) 104
7.7 :TASK:COMPoser[:DEFine]:IDLE[:TYPE] {DC|FIRSt|CURRent}(?) 104
7.8 :TASK:COMPoser[:DEFine]:IDLE:LEVel {<DC_level>}(?) 105
7.9 :TASK:COMPoser[:DEFine]:ENABle{NONE|TRG1|TRG2|TRG3|TRG4|TRG5|TRG6|INTernal|CPU |FBTRg| ANY}(?) 105
7.10 :TASK:COMPoser[:DEFine]:ABORt{ NONE|TRG1|TRG2|TRG3|TRG4|TRG5|TRG6| INTernal|CPU|FBTRg|ANY }(?) 107
7.11 :TASK:COMPoser[:DEFine]:JUMP{EVENtually| IMMediate}(?) 109
7.12 :TASK:COMPoser[:DEFine]:DESTination{NEXT | FBTRg | TRG | NTSel | SCENario | DSP |DSIG}(?) 110
7.13 :TASK:COMPoser[:DEFine]:NEXT1 <next_task>(?) 110
7.14 :TASK:COMPoser[:DEFine]:NEXT2 <next_task>(?) 111
7.15 :TASK:COMPoser[:DEFine]:DELay<task_delay>(?) 111
7.16 :TASK:COMPoser[:DEFine]:KEEP{OFF|ON|0|1}(?) 112
7.17 :TASK:COMPoser[:DEFine]:DTRigger{OFF|ON|0|1}(?) 112
7.18 :TASK:COMPoser:WRITE<offset in task table rows> 112
7.19 :TASK:COMPoser:READ<offset in task table rows> 113
7.20 :TASK:CURRent? 113
7.21 :TASK: SYNC 113
7.22 :TASK:DATA [<offset>]#<header><binary_block> 113
7.23 :TASK:FILE[:NAME] {#<header><binary_block>} 115
7.24 :TASK:FILE:OFFSet <start-offset> 116
7.25 :TASK:FILE:LOAD[<offset>,<num_of_tasks>] 116
7.26 :TASK:FILE:STORe[<offset>,<num_of_tasks>] 116
7.27 :TASK:ZERO[:PORTion] <offset>,<num_of_tasks> 117
7.28 :TASK:ZERO:ALL 117

8 Scenario Commands 118

8.1 :SCENario:DEFine { <scenario-number>, <task-number>, <loops>}(?) 118
8.2 :SCENario:DATA {[<offset>,]#<header><binary_block>} 118
8.3 :SCENario:FILE[:NAME]{ #<header><binary_block>} 119

8.4 :SCENario:FILE:OFFSet {<offset>} 119
8.5 :SCENario:FILE:LOAD {[<offset>,<num_of_scenarios>]} 120
8.6 :SCENario:FILE:STORe {[<offset>,<num_of_scenarios>]} 120
8.7 :SCENario:ZERO[:SINGLe] <scenario-number> 120
8.8 :SCENario:ZERO:ALL 121

9 Arbitrary Waveform Commands 122

9.1 :TRACe[:DATA](?) [<offset>]#<header><binary_block> 123
9.2 :TRACe:FORMat{ <U16 | U8>}(?) 125
9.3 :TRACe:MEMory(?)< offset_in_wave-points>#<header><wave-data> 126
9.4 :TRACe:SEGMents[:DATA] [<first segment number>,]#<header><binary_block> 126
9.5 :TRACe:SEGMents:FILE[:NAMe] #<header><binary_block> 127
9.6 :TRACe:SEGMents:FILE:OFFSet <offset in bytes>(?) 127
9.7 :TRACe:SEGMents:FILE:LOAD[[<first segment number>,],<number of segments>] 128
9.8 :TRACe:FILE[:NAME]#<header><binary_block> 128
9.9 :TRACe:FILE:OFFSet< offset in bytes>(?) 128
9.10 :TRACe:FILE:DESTination{SEGMent | MEMory}(?) 129
9.11 :TRACe:FILE:LOAD[<offset>,<size in wave-points>] 129
9.12 :TRACe:FILE:STORe[<offset>,<size>] 129
9.13 :TRACe:STReaming:MODE {FILE|DYNamic}(?) 130
9.14 :TRACe:STReaming:STATe{OFF|ON|0|1}(?) 130
9.15 :TRACe:DEFine[:SIMPle] [<seg_number>,]<seg_length>(?) 131
9.16 :TRACe:DEFine:LENGth? 131
9.17 :TRACe:ZERO[:SEGMent] [<segment number>] 131
9.18 :TRACe:ZERO:ALL 132
9.19 :TRACe:DELete[:SEGMent] <seg-number> 132
9.20 :TRACe:DELete[:SEGMent]:ALL 132
9.21 :TRACe:SELect[:SEGMent] <seg_number>(?) 132
9.22 :TRACe:SELect:SOURce{ BUS | EXTernal | ADC | DCT }(?) 133
9.23 :TRACe:SELect:TIMing{ EVENTually|IMMediate}(?) 133
9.24 :TRACe:FREE? 134
9.25 :TRACe:FRAG? 134
9.26 :TRACe:DEFRag 135

10 Digitizer Commands 136

10.1 :DIGitizer:[:SELect]{ DIG1|DIG2}(?) 136
10.2 :DIGitizer:MODE{DUAL|SINGle}(?) 137
10.3 :DIGitizer:CHANnel[:SELect]{ CH1|CH2}(?) 137
10.4 :DIGitizer:CHANnel:STATe{ DISabled|ENABled}(?) 138
10.5 :DIGitizer:CHANnel:RANGe{ HIGH|MEDium|LOW}(?) 138
10.6 :DIGitizer:CHANnel:OFFSet< offset_level >(?) 138
10.7 :DIGitizer:DDC:MODE{REAL|COMPlex}(?) 139
10.8 :DIGitizer:DDC:DECimation{ NONE | X1| X4 | X16}? 140
10.9 :DIGitizer:DDC:BIND{ OFF | ON |0 |1 }(?) 140
10.10 :DIGitizer:DDC:CFRequency<1|2> <carr_freq>(?) 141
10.11 :DIGitizer:DDC:PHASe<1|2> {<phase in degrees>}(?) 141
10.12 :DIGitizer:DDC:CLKSource{ DIG|AWG}(?) 141
10.13 :DIGitizer:ACQuire[:FRAMes]:DEFine<num_of_frames><frame_length> (?) 142
10.14 :DIGitizer:ACQuire[:FRAMes]:FREE 143
10.15 :DIGitizer:ACQuire[:FRAMes]:CAPTure[:SELECT]<1st frame>,<num-frames> (?) 143
10.16 :DIGitizer:ACQuire:[FRAMES]:CAPTure:ALL 144
10.17 :DIGitizer:ACQuire[:FRAMes]:MARKer{OFF|ON|0|1}(?) 144
10.18 :DIGitizer:ACQuire:STATus? 144

11.12 :DSP:MATH:OPERation{ MI1 | MQ1 | MI2 | MQ2 | MI3 | MQ3 |MI4 |MQ4 | MI5 |MQ5 | MI6 |MQ6 | MI7 | MQ7 | MI8|MQ8 | MI9 |MQ9 | MI10 |MQ10 ,<SCALe>,<OFFSet> }(?) 172
11.13 :DSP:MATH:OPERation:CLIP(?) 173
11.14 :DSP:MATH:XCORrelation:LENGth<N>(?) 174
11.15 :DSP:MATH:XCORrelation:SIGNal{ <MI1 | MQ1 | MI2 | MQ2 | MI3 | MQ3 |MI4 |MQ4 | MI5 |MQ5 | MI6 |MQ6 | MI7 | MQ7 | MI8|MQ8 | MI9 |MQ9 | MI10 |MQ10 }(?) 174
11.16 :DSP:MATH:RAVG { MI1 | MQ1 | MI2 | MQ2 | MI3 | MQ3 |MI4 |MQ4 | MI5 |MQ5 | MI6 |MQ6 | MI7 | MQ7 | MI8|MQ8 | MI9 |MQ9 | MI10 |MQ10 |XC,<N> }(?) 175
11.17 :DSP:DECision[:FEEDback]:MAPping{ <awg channel number>,DEC1 | DEC2| DEC3 | DEC4 | DEC5 | DEC6 | DEC7 | DEC8 | DEC9 | DEC10 | XC } (?) 176
11.18 :DSP:DECision[:FEEDback]:CONDition{<awg-channel number>, S1 | S2 |S3 | S4 | S5 | S6 | S7 | S8, <segment number>}(?) 177
11.19 :DSP:DECision:FRAMe<the frame size for the calculation>(?) 178
11.20 :DSP:DECision:IQPath:SELect { DSP1 | DSP2 | DSP3 | DSP4| DSP5| DSP6| DSP7| DSP8| DSP9| DSP10 }(?) 178
11.21 :DSP:DECision:IQPath:OUTPut{ THR | SVM }(?) 179
11.22 :DSP:DECision:IQPath:THReshold:LEVel { <N> }(?) 179
11.23 :DSP:DECision:IQPath:THReshold:INPut { I | Q }(?) 179
11.24 :DSP:DECision:IQPath:LINE{ 1 | 2 | 3, <slope>, <y-intercept>}(?) 180
11.25 :DSP:DECision:IQPath:CLIP(?) 180
11.26 :DSP:DECision:XCORrelation: THReshold { <N> }(?) 180
11.27 :DSP:DECision:XCORrelation:CLIP(?) 181

12 System Commands 182

12.1 :SYSTem:LOG[:VERBose] {0|1|2|3|4|5|6}(?) 182
12.2 :SYSTem:ERRor? 182
12.2.1 Error list 182
12.3 :SYSTem:INFormation:CALibration? 184
12.4 :SYSTem:INFormation:MODel? 184
12.5 :SYSTem:INFormation:SERial? 184
12.6 :SYSTem:INFormation:HARDware? 184
12.7 :SYSTem:INFormation:FPGA:VERsion? 185
12.8 :SYSTem:INFormation:FPGA:DATE? 185
12.9 :SYSTem:INFormation:FIRMware:VERsion? 185
12.10 SYSTem:INFormation:FIRMware:DATE? 185
12.11 :SYSTem:INFormation:DAC? 186
12.12 :SYSTem:INFormation:SLOT? 186
12.13 SYSTem:INFormation:SCPI[:VERSion]? 186
12.14 :SYSTem[:MEASure]:TEMPerature? 186
12.15 :SYSTem[:MEASure]:HTPeak? 187
12.16 :SYSTem[:MEASure]:LTPeak? 187
12.17 :SYSTem[:MEASure]:VINTernal? 187
12.18 :SYSTem[:MEASure]:VAUXiliary? 187
12.19 :SYSTem:FILE:CATalog? 188
12.20 :SYSTem:FILE[:NAMe]{< #<header><binary-block>} 188
12.21 :SYSTem:FILE:SIZE? 189
12.22 :SYSTEM:FILE:DATA[<offset>,]#<header><binary_block>(?) 189
12.23 :SYSTEM:FILE:DELete 189

13 Appendix Proteus SCPI MATLAB Script Examples 190

13.1 Introduction 190
13.2 Opening a Session with Proteus 190
13.2.1 Programming Example 1 191

Figures

Tables

1 Introduction

This manual lists and describes the set of SCPI-compatible (Standard Commands for Programmable Instruments) remote commands used to operate the Tabor Proteus series arbitrary waveform generator/transceiver. Refer to the Proteus User Manual for a description of the functionality of the device. The complete listing of all commands used for programming the Proteus is given in chapter 2 SCPI Commands List Summary, page 25.

1.1 Introduction to SCPI

Commands to program the instrument over the GPIB are defined by the SCPI 1993.0 standard. The SCPI standard classifies a common language protocol. It goes one step further than IEEE-STD-488.2 and defines a standard set of commands to control every programmable aspect of the instrument. It also defines the format of command parameters and the format of values returned by the instrument.

SCPI is an ASCII-based instrument command language designed for test and measurement instruments. SCPI commands are based on a hierarchical structure, known as a tree system. In this system, associated commands are grouped together under a common node or root, consequently forming subsystems.

Part of the :INITiate subsystem is shown below to illustrate the tree system:

```
:INITiate
:CONTinuous
:STATe ON|OFF
```

INITiate is the root keyword of the command; CONTinuous is a second level keyword. State is third level keyword. A colon (:) separates a command keyword from a lower level keyword.

1.2 Command Format

The format used to show commands in this manual is shown below.

```
FREQuency {<frequency>|MINimum|MAXimum}
```

The command syntax shows most commands (and some parameters) as a mixture of upper and lowercase letters. The uppercase letters indicate the abbreviated spelling for the command. For shorter program lines, send the abbreviated form. For better program readability, use the long form.

For example, in the above syntax statement, FREQ and FREQUENCY are both acceptable forms. Use upper or lowercase letters. Therefore, FREQ, FREQUENCY, freq, and Freq are all acceptable. Other forms such as FRE and FREQUEN will generate an error.

The above syntax statement shows the frequency parameter enclosed in curly brackets ({}). The brackets are not sent with the command string. A value for the frequency parameter (such as "FREQ 1e+9”) must be specified.

Some parameters are enclosed in square brackets ([ ]). The brackets indicate that the parameter is optional and can be omitted. The brackets are not sent with the command string.

1.3 Command Separator

A colon (:) is used to separate a command keyword from a lower level keyword as shown below:

```
:SOUR:FUNC:MODE ARB
```

A semicolon (;) is used to separate commands within the same subsystem, and can also minimize typing. For example, sending the following command string:

```
:INST:CHAN 1;:OUTP ON
```

is the same as sending the following two commands:

```
:INST:CHAN 1
:OUTP ON
```

Use the colon and semicolon to link commands from different subsystems. For example, in the following command string, an error is generated if both the colon and the semicolon are not used.

```
:FREQ 1e9;:OUTP ON
```

1.4 MIN and MAX Parameters

Substitute MINimum or MAXimum in place of a parameter for some commands. For example, consider the following command:

```
FREQuency {<frequency>|MINimum|MAXimum}
```

Instead of selecting a specific frequency, substitute MIN to set the frequency to its minimum value or MAX to set the frequency to its maximum value.

1.5 Querying Parameter Setting

Query the current value of most parameters by adding a question mark (?) to the command. For example, the following command sets the output function to square:

```
SOUR:FUNC:SHAP SQR
```

Query the output function by executing:

```
SOUR:FUNC:SHAP?
```

1.6 Query Response Format

The response to a query depends on the format of the command. In general, a response to a query contains current values or settings of the generator. Commands that set values can be queried for their current value. Commands that set modes of operation can be queried for their current mode. IEEE-STD-488.2 common queries generate responses, which are common to all IEEE-STD-488.2 compatible instruments.

1.7 SCPI Command Terminator

A command string sent to the generator must terminate with a <new line> character. Command string termination always resets the current SCPI command path to the root level.

1.8 IEEE-STD-488.2 Common Commands and Queries

Since most instruments and devices in an ATE system use similar commands that perform similar functions, the IEEE-STD-488.2 document has specified a common set of commands and queries

that all compatible devices must use. This avoids situations where devices from various manufacturers use different sets of commands to enable functions and report status.

The IEEE-STD-488.2 treats common commands and queries as device dependent commands. For example, *TRG is sent over the bus to trigger the instrument. Some common commands and queries are optional, but most of them are mandatory.

The following is a complete listing of all common-commands and queries, which are used by the Proteus series.

Table 1-1 SCPI Common Mode Commands

Keyword	Notes
*CLS	Clear the error-list (and therefore also the corresponding bit in the STB).
*IDN?	The Identification query outputs an identifying string. The response will show the following information: <company name>, <model name>, <serial number>, <FPGA version>
*OPC?	Returns the ASCII character "1" to the output buffer after all the previous commands have been executed. The command is used for synchronization between a controller and the instrument using the MAV bit in the Status Byte or a read of the Output Queue. Reading the response to the *OPC? query has the advantage of removing the complication of dealing with service requests and multiple polls to the instrument. However, both the system bus and the controller handshake are in a temporary hold-off state while the controller is waiting to read the *OPC? query response.
*OPT?	The options query returns a comma-separated list of all of the instrument options currently installed on the signal generator, such as the number of channels and memory size.
*RST	Resets the instrument to its default state.
*TRG	Same as 4.14:TRIGger:IMMediate, page 71.
*TST? For future use.	The Self-Test query initiates the internal self-test and returns one of the following results: 0 – All tests passed. 1 – One or more tests failed.

1.9 SCPI Parameter Type

The SCPI language defines four different parameter types to be used in program messages and response messages: numeric, discrete, Boolean, and binary block.

1.9.1 Numeric Parameters

Commands that require numeric parameters will accept all commonly used decimal representations of numbers including optional signs, decimal points, and scientific notation. Special values for numeric parameters like MINimum and MAXimum are also accepted.

Engineering units using numeric parameters (e.g., MHz or kHz) can also be sent. If only specific numeric values are accepted, the function generator will ignore values which are not allowed and

will generate an error message. The following command is an example of a command that uses a numeric parameter:

```
VOLT:AMPL <amplitude>
```

1.9.2 Discrete Parameters

Discrete parameters are used to program settings that have a limited number of values (i.e., FIXed, USER and SEQuence). They have short and long form command keywords. Upper and lowercase letters can be mixed. Query responses always return the short form in all uppercase letters. The following command uses discrete parameters:

```
SOUR:FUNC:MODE {ARBitrary | TASK }
```

1.9.3 Boolean Parameters

Boolean parameters represent a single binary condition that is either true or false. The generator accepts "OFF" or "0" for a false condition. The generator accepts "ON" or "1" for a true condition. The instrument always returns "0" or "1" when a Boolean setting is queried. The following command uses a Boolean parameter:

```
OUTP:STAT { OFF | ON }
```

The same command can also be written as follows:

```
OUTP:STAT {0 | 1 }
```

1.9.4 Binary Block Parameters

Binary block parameters are used for transferring data blocks to the generator, for example, waveforms, segment table, sequence table etc. The binary block parameter format is

#<header><binary data>

Where the header, holds the data size, followed by the data itself. For example, the following command uses the binary block parameter **#42048<binary data>** to transfer a 1024 points waveform to the generator

```
TRAC:DATA#42048<binary_block>
```

Information on commands using binary blocks is given later in this manual, e.g., refer to 6.8 :MARKer:DATA [<offset>]#<header><binary_block>, page 97.

1.10 Queries for Commands with Numeric Parameters

Commands that require numeric parameters accept queries that return the accepted legal values of the parameter, as well as the default value. The accepted queries are RANGe?, MIN?, MAX? and DEFault?. The command syntax is the command followed by the required query, e.g.,

```
:FREQ RANGe?
```

1.10.1 RANGe?

The RANGe? query will return the legal range of the accepted values of the numeric parameter used in the command. The response format is **min value, max value, default value.**

Example

Query: `VOLT RANG?`

Response: `0.001,1.3,0.5`

1.10.2 MIN?

The MIN? query will return the legal minimum accepted value of the numeric parameter used in the command. The response format is **`min value.`**

Example

Query: **`VOLT MIN?`**

Response: `0.001`

1.10.3 MAX?

The MAX? query will return the legal maximum accepted value of the numeric parameter used in the command. The response format is **`max value.`**

Example

Query: **`VOLT MAX?`**

Response: `1.3`

1.10.4 DEFault?

The DEF? query will return the default value of the numeric parameter used in the command. The response format is **`default value.`**

Example

Query: **`VOLT DEF?`**

Response: `0.5`

> **Note**
> As these RANGe?, MIN?, MAX? and DEFault? queries are standard for all commands with numeric parameters they are only described above and listed only in the SCPI commands list summary tables in 2 SCPI Commands List Summary, page 25.

1.11 SCPI Syntax and Styles

Where possible, the syntax and styles used in this manual follow those defined by the SCPI consortium. The commands on the following pages are broken into three columns: the Keyword, the Parameter Form, and Default.

The Keyword column provides the name of the command. The actual command consists of one or more keywords, since SCPI commands are based on a hierarchical structure, also known as the tree system. Square brackets ([ ]) are used to enclose a keyword that is optional when programming the command. Therefore, the Proteus series instrument will process the command to have the same affect whether the optional node is omitted by the programmer, or not. Letter case in tables is used to differentiate between the accepted short form (upper case) and the long form (upper and lower case).

The Parameter Form column indicates the number and order of a parameter in a command and their legal value. Parameter types are distinguished by enclosing the type in angle brackets (< >). If parameter form is enclosed by square brackets ([ ]) these are then optional (pay attention to be sure that optional parameters are consistent with the intention of the associated

keywords). The vertical bar (|) can be read as "or" and is used to separate alternative parameter options.

Table 1-2 SCPI Syntax and Styles

Convention	Description	Example
{}	Braces indicate that parameters may be used in the command once, or several times, or not at all.	:LIST:POWer <val>{,<val>} a single power listing: LIST:POWer 5 a series of power listings: LIST:POWer 5,10,15,20
<>	Angle brackets indicate that their contents are not to be used literally in the command. They represent the required parameters.	:FREQuency:STARt <val><unit> In this command, the words <val> and <unit> should be replaced by the actual frequency and unit. :FREQuency:STARt 2.5GHZ
[]	Square brackets indicate that the enclosed keywords or parameters are optional when composing the command. The commands will be executed even if they are omitted.	:FREQuency[:CW]? SOURce and CW are optional items.
|	A vertical stroke between keywords or parameters indicates alterative choices. For parameters, the effect of the command varies depending on the choice.	:AM:MOD DEEP|NORMal DEEP or NORMal are the choices.

1.12 Proteus SCPI Commands

The table below lists all of the Proteus series SCPI commands. The commands are arranged in logical groups that provide similar functionality, and make it easier to understand the various commands.

The commands are divided into 8 different groups, each describing a different type of operation.

- Channel and group control commands
- Run mode commands
- Analog output commands
- Marker output commands
- Task commands
- Scenario commands
- Arbitrary waveform commands
- Digitizer commands
- System commands

Detailed descriptions of each of the various commands are given in the following chapters. Note that the table lists the commands of the entire Proteus series and commands may vary depending on your model and installed options.

2 SCPI Commands List Summary

2.1 Instrument Commands

Refer to section 3 Instrument Commands, page 54 for details.

Table 2-1 Instrument Commands

Keyword	Parameter Form	Default	Notes
:INSTrument			
:ACTive			
[:SELect]	1 ….	1	Select the addressed instrument (1 is the master instrument).
:CHANnel			
[:SELect]	1 |…| 12	1	Select the programmable channel (of the selected instrument).
:OFFSet	0 to 1024	0	Set the delay between channels in units of samples.
:RANGe?			Query only. It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:COUPle			The skew and offset between DAC1 and DAC2.
SKEW	-3e-9 to 3e-9	0	Skew (in seconds) between DAC 1 and DAC 2 of the module of the selected channel.
:RANGe?			Query only. It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:XINStrument			Synchronizing multiple Proteus module instruments.

Keyword	Parameter Form	Default	Notes
:MODE?	SING | MASTer | SLAVe	0	The mode of the instrument. It can be single, master or slave. Query only. The mode is set by the system at power up.
:CHAIN			Use this sub group command to define a chain of instruments comprised of a parent and children. Only the parent receives the SCPI commands. To access the child instruments use the INST:ACT:SEL command.
:SYNChronize			Multi-Instrument sync
:ROLE?	SINGle|LEADer| FOLLower	SINGle	The query will return the synchronization mode of the active instrument (query only). The role is set by the system at power up and can be changed by the :FOLLowers command. **SINGle** – The instrument is not part of a chain. **LEADer** – The instrument is the leader of the chain. **FOLLower** – The instrument is a follower in the chain.
:FOLLowers	<number_of_follower_instruments>	0	Make the n consecutive instruments synchronization followers of this instrument. If n=0 and the instrument is a leader-instrument then the chain is disassembled.
:OFFSet **Future Release**	0 to n (n = waveform length)		Multi-instrument offset.
:RANGe?			Query only. It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:SKEW **Future Release**	-5e-9 to 5e-9	0	Multi-instrument skew.
:RANGe?			Query only. It returns the minimum legal value, the maximum legal value, and the default value separated by commas.

Keyword	Parameter Form	Default	Notes
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.

2.2 Run Mode Commands

Refer to section 4 Run Mode Commands, page 59 for details.

Table 2-2 Run Mode Commands

Keyword	Parameter Form	Default	Notes
:INITiate			
:CONTinuous			
[:STATe]	OFF | ON | 0 | 1	1	Defines the continuous run mode of the instrument.
:TRIGger			
:COUPle			
[:STATe]	OFF | ON | 0 | 1	0	When all modules in the instrument are set to ON it will receive the trigger from the MASTER.
:SOURce			
:ENABle	NONE|TRG1|TRG2|TRG3 |TRG4|TRG5|TRG6| INTernal|CPU|FBTRg| HWControl	NONE	The source of the enabling signal of the selected channel.
:DISable	NONE|TRG1|TRG2|TRG3 |TRG4|TRG5|TRG6| INTernal|CPU|FBTRg| HWControl	NONE	The source of the abort signal of the selected channel.
[:ACTive]			
:SELect	TRG1|TRG2|TRG3|TRG4| TRG5|TRG6|INTernal	TRG1	Select the trigger for programming.
:STATe	OFF | ON | 0 | 1	0	Enable / disable the selected external trigger (channel dependent).
:CPU			
:MODE	LOCAL | GLOBAL	GLOBAL	CPU trigger mode. **LOCAL** – The active channel receives the CPU trigger. **GLOBAL** – All channels receive the same CPU trigger simultaneously.
:GATE			

Keyword	Parameter Form	Default	Notes
[:STATe]	OFF | ON | 0 | 1	0	Enable/disable gated mode of the selected external trigger (channel dependent). (Internal trigger has no gate mode, so its gate mode is only OFF.)
:LEVel?	-5 to 5, 0.1V resolution	0.0	The threshold voltage level of the selected external-trigger (shared by all channels of the same module).
:RANGe?			Query only. It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:COUNt	0 - 1M	1	Defines the number of times the current segment will be played for a given trigger signal.
:RANGe?			Query only, It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:WIDTh	0, from 10e-9 to 2s, 2ns resolution	0	The pulse-detect width of the selected trigger of the selected channel. It is relevant only for an external trigger. Zero means edge.
:RANGe?			Query only. It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.

Keyword	Parameter Form	Default	Notes
:SLOPe	POSitive | NEGative	POS	The valid slope for the selected external trigger (channel dependent).
:TIMer	200e-9 to 2.0s	1.50E-05	The period of the internal trigger in seconds (channel dependent).
:RANGe?			Query only. It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:IMMediate			Same as *TRG. Triggers the generator from the remote interface.
:MODE	EVENtually | IMMediate	EVEN	Use this command to define or query the trigger mode.
:LTJ			Proteus with low trigger jitter option.
[:STATe]	OFF | ON | 0 | 1	1	Activate low jitter option.
:IDLE			
[:TYPE}	DC | FIRSt | CURRent	DC	Set the type of the idle waveform that will be played when waiting for the trigger (channel dependent).
:LEVel	0 to maximum DAC level	Mid DAC level	The DAC level of IDLE DC.
:RANGe?			Query only. It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:PULSe			
[:STATe] **Future Release**	OFF | ON | 0 | 1	OFF	Turn on the pulse counter for the external trigger.
:COUNt? **Future Release**	<count>	0	Query returns the number of counted pulses.

Keyword	Parameter Form	Default	Notes
:RESet **Future Release**			Reset the pulse counter.
:DELay	external-trigger: 0 to at least 6.55µs in sysclk resolution internal-trigger: only 0	0	Set the delay of the selected external trigger of the selected channel. Under the hood it is rounded to sysclk ticks (between 0 and 2047, where 0 means no delay).
:HOLDoff **Future Release**	external-trigger: from 0 to TBD. Internal-trigger: only 0 (no holdoff)	100ns external trigger	Set the holdoff of the selected external trigger of the selected channel. Incoming trigger will be ignored during the holdoff period.

2.3 Analog Output Control Commands

Refer to section 5 Analog Output Control Commands, page 77 for details.

Table 2-3 Analog Output Control Commands

Keyword	Parameter Form	Default	Notes
:OUTPut			
[:STATe]	OFF | ON | 0 | 1	0	Output control
[:SOURce]			
:MODE	DIRect|NCO|DUC	DIR	Generation mode. **DIRect** – Direct arbitrary waveforms (NCO AND DUC OFF) **NCO** – NCO generated signals only (NCO ON, DUC OFF) **DUC** – IQ modulations (NCO ON, DUC ON).
:PTRepeat	X1|X2|X4|X8	X1	Set the point repeat factor. The Point Repeat factor enables the user to configure the unit so that each sample point that is sent to the FPGA is repeated by the point repeat factor. This essentially enables the user to lower the SCLK below the minimum 1GS/s limit. For example, if point repeat is set to x4, each sample is sent 4 times to the DAC, and thus if the SCLK is 1GS/s the output appears as if the SCLK is 250MS/s. Note that this can be used only with segments that are normal (not fast).
:INTerpolation	NONE|X2|X4|X8|	NONE	Set the interpolation factor. Relevant only for models with DUC option. It is shared by all channels in the same module, and in case of synchronized master slaves chain, all slaves have

Keyword	Parameter Form	Default	Notes
			the same interpolation mode of the master (like sampling clock rate).
:NCO			
:MODE	SINGle|DUAL	SING	Set the NCO mode. In dual mode, the user can control two NCOs (1 or 2).
:CFRequency<1|2>	0 Hz to sclk	4e+08	Set the carrier frequency for the selected NCO <1|2> of the selected channel.
:PHASe<1|2>	<phase in degrees>	0	Set the phase of the selected NCO <1|2> (in degrees) of the selected channel.
:SIXDb<1|2>	OFF|ON|0|1	0	Enable/disable 6dB gain of the selected NCO<1|2> of the selected channel.
:IQModulation	NONE|HALF|ONE|TWO	NONE	Set the IQ modulation type. It is shared by all channels in the module and by all modules in a synchronized master-slaves chain. The IQ modulation type are classified by the number of IQ pairs per channel: **HALF** – 'I' in channel 1 and 'Q' in channel 2. **ONE** – 1 IQ-Pair, organized in pairs of 'I' sample followed by 'Q' sample. **TWO** – 2 IQ pairs organized in 4-tuples of the form (I1,q1,I2,q2)
:FREQuency			
[:RASTer]	1e9 to 9e9 | MINimum | MAXimum	1.0e+09	Sampling rate (samples per second).
:RANGe?			Query only. It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:SOURce	INTernal | EXTernal	INT	Selects the SCLK source.
:OUTPut			
[:STATe]	OFF | ON | 0 | 1	0	The state of the output clock.
:FUNCtion			

Keyword	Parameter Form	Default	Notes
:MODE			
[:TYPE]	ARBitrary | TASK	ARB	**Arbitrary** – Plays the selected segment (:FUNCtion:SEGment). **TASK** – Plays the active scenario.
:SEGMent	1 to 64k	1	The number of the segment selected for playback in the selected channel in case of Arbitrary mode. The first 128 segment are "Fast-Segments".
:RANGe?			Query only. It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:TASK	1 to 64k	1	The number of the first task for playback in the selected channel in case of Task-Mode.
:RANGe?			Query only. It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:ROSCillator			
:SOURce	INTernal | EXTernal	INT	Source for the 10 MHz reference signal.
:FREQuency	10M | 100M	100M	When the reference oscillator is set to external, select the frequency of the reference signal that will be entered in the REF IN connector, 10MHz or 100MHz.
:VOLTage			
[:AMPLitude]	1e-3 to 1.2 | MINimum | MAXimum	0.5	Output amplitude
:RANGe?			Query only. It returns the minimum legal value, the maximum legal

			value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:OFFSet	-0.5 to 0.5 | MINimum | MAXimum	0	Output offset.
:RANGe?			Query only, It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.

2.4 Marker Output Commands

Refer to section 6 Marker Output Commands, page 93 for details.

Table 2-4 Marker Output Commands

Keyword	Parameter Form	Default	Notes
:MARKer			
:SELect	1 | 2 | 3 | 4	1	Select the programmable marker of the selected channel.
[:STATe]	OFF | ON | 0| 1	0	Marker activation.
:DELay			
:COARse	16-bit DAC mode: -255 – 254 8-bit DAC mode: -1024 - 1016	0	Marker coarse delay from the output of the corresponding channel (in wave points).
:RANGe?			Query only, It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.

:FINE	-600ps to +600ps	0	Marker fine delay from output (in seconds).
:RANGe?			Query only, It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:VOLTage			
:LEVel	0 to 32		The marker gain level in dB.
:RANGe?			Query only, It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:PTOP	0.05 Vpp to 1.2 Vpp	0.5	The marker peak-to-peak voltage. There are 32 available voltage levels corresponding to 32 steps of 1 dB down from ~1.2V. The best one (according to the requested voltage value) is selected.
:RANGe?			Query only. It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:OFFSet	-0.5V to 0.5V	0	Marker offset level in volt.
:RANGe?			Query only. It returns the minimum legal value, the maximum legal value, and the default value separated by commas.

:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:DATA	[<offset>]#<header> <binary_block >		Write or read markers data to (or from) the specified offset in the selected segment.
:MEMory	<offset>#<header> <marker-data>		Write or read markers data to (or from) the specified offset of the arbitrary-memory space. Direct download of marker data to the arbitrary memory without any segment attributes.
:FILE			
[:NAME]	#<header><binary_block>		File path name passed as binary data.
:OFFSet	<start-offset inside the file in bytes>	0	The start offset inside the file in bytes.
:DESTination	SEGMent | MEMory	SEGM	The destination to load/store the file data: **SEGMent** – The selected segment. **MEMory** – The arbitrary-memory space.
:LOAD	[[<offset>], <size>]		Load marker data from file to the memory (no query). If the offset and size are not specified then the whole DESTination is written.
:STORe	[[<offset>,] <size>]		Store markers-data from the memory to file (no query). If the offset and size are not specified then the whole DESTination is written.

2.5 Task Commands

Refer to section 7 Task Commands, page 102 for details.

Table 2-5 Task Commands

Keyword	Parameter Form	Default	Notes
:TASK			
:COMPoser			
:LENGth	0 to 64k	0	Allocate an array of task table rows for the task table composer.
:SELect	1 to 64k	1	Select the task to define.
:RANGe?			Query only, It returns the minimum legal value, the maximum legal

Keyword	Parameter Form	Default	Notes
			value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:DEFine			
:TYPE	SINGle | STARt | END | SEQ	SINGle	Task type: **Single** – Not part of task-sequence. **Start** – The start of task-sequence. **END** – The end of task-sequence **SEQ** – Inside a task-sequence (neither first nor last).
:LOOPs	0 to 1M	1	Number of loops for the task
:RANGe?			Query only, It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:SEQuence	0 to 1M	1	Number of loops for the sequence.
:RANGe?			Query only, It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:SEGMent	<segment-number> 1 - 64k	1	The segment that is associated with the task.
:IDLE			
[:TYPE]	DC | FIRSt | CURRent	DC	The behavior in the idle state.
:LEVel?	0 to max DAC level (255 / 65535)	Mid DAC	The DAC level used for the DC idle waveform.

Keyword	Parameter Form	Default	Notes
:ENABle	NONE|TRG1|TRG2|TRG3| TRG4|TRG5|TRG6| INTernal|CPU|FBTRg|ANY	NONE	Enabling signal for the task.
:ABORt	NONE|TRG1|TRG2|TRG3 | TRG4|TRG5|TRG6| INTernal|CPU|FBTRg|ANY	NONE	The abort signal for the task.
:JUMP	EVENtually | IMMediate	EVEN	Jump mode.
:DESTination	NEXT|FBTRg|TRG|NTSel|SCENario|DSP|DSIG	NEXT	Next task destination: **NEXT** – The task defined by the command :NEXT1. **FBTRg** – Select the next task by the feedback trigger value. **TRG** – NEXT1 upon Trigger1, Next2 upon Trigger2. **NTSel** – The next task in the table. **SCENario** – The beginning of next scenario. **DSP** - Destination is NEXT1 current segment to be generated is according to decision block condition in DSP. **DSIG** – NEXT1 if digitizer-signal = 1, NEXT2 if digitizer-signal = 0.
:NEXT1	<task-number> 1 to 64k	1	Next task for TRG1 input (zero means end).
:NEXT2	<task-number> 1 to 64k	0	Next task for TRG2 input (zero means end).
:DELay	0 to 65,536	0	Delay in clocks before executing next task.
:RANGe?			Query only, It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:KEEP	OFF | ON | 0 | 1	0	Wait for the trigger when looping.
:DTRigger	OFF | ON | 0 | 1	0	Generate the digitizer trigger.
:WRITe	<offset in task-table rows>	0	Write the composer's array to the task-table of the selected channel at the specified offset (no query).

Keyword	Parameter Form	Default	Notes
:READ	<offset in task-table rows>	0	Read the composer's array from the task-table of the selected channel at the specified offset (no query).
:CURRent?			Query only. Returns the current task number.
:SYNC			Issue this command to synchronize the task tables of all channels. This command needs to be issued every time before generation is started.
:DATA	[<offset>]#<header><binary_block>	0	Write or read a block of rows to (or from) the specified offset in the task table of the selected channel.
:FILE			Read task from file
[:NAME]	#<header><binary_block>		File path-name passed as a binary-block.
:OFFSet	<start-offset>	0	The start offset inside the file in bytes.
:LOAD	[<offset in task-table rows>,<number of task-table rows>]		Load row-data from the file to the task-table of the selected channel (no query). If the offset and number of tasks are not specified, then the whole task-table is written. (If the file is too small then the rest of the task-table rows are zeroed).
:STORe	[<offset in task-table rows>,<number of task-table rows>]		Store rows-data from the task table of the selected channel in the file (no query). If the offset and number-of rows are not specified then the whole task-table is stored.
:ZERO			
[:PORTion]	<offset in task-table rows><number of task-table rows>		Reset portion of the task-table of the selected channel (no query).
:ALL			Reset the whole task-table of the selected channel (no query)

2.6 Scenario Commands

Note
Scenario commands are planned for a future release.

Refer to section 8 Scenario Commands, page 118 for details.

Table 2-6 Scenario Commands

Keyword	Parameter Form	Default	Notes
:SCENario			

Keyword	Parameter Form	Default	Notes
:DEFine	<scenario-number>, <task-number>, <loops>		Define the specified entry in the scenario-table of the selected channel.
:DATA	[<offset>,]#<header><binary_block >		Write data to the specified offset in the scenario table of the selected channel.
:FILE			
[:NAME]	#<header><binary_block>		File path-name passed as block of binary data.
:OFFSet	<offset in the file>	0	The start offset inside the file in bytes.
:LOAD	[<offset>, <num_of_scenarios>]		Load the scenario table of the selected channel from the file (in binary format)
:STORe	[<offset>, <num_of_scenarios>]		Store the scenario table of the selected channel in the file (in binary format).
:ZERO			
[:SINGle]	<scenario-number>		Reset the data of a single row in the scenario table of the selected channel (no query).
:ALL			Reset the data of all rows in the scenario-table of the selected channel (no query).

2.7 Arbitrary Waveform Commands

Refer to section 9 Arbitrary Waveform Commands, page 122 for details.

Table 2-7 Arbitrary Waveform Commands

Keyword	Parameter Form	Default	Notes
:TRACe			
[:DATA]	[<offset>]#<header> <binary_block>		Write or read waveform data to the selected segment starting at the specified offset.
:FORMat	U16 | U8	U16 U8 for P9082x	Set the format of the user waveform data. **U16** – Unsigned 16 bit. **U8** – Unsigned 8 bit.
:MEMory	<offset_in_wave-points>,#<header> <wave-data>		Write waveform data to the arbitrary-memory space starting from the specified offset. The query format is: :TRAC:MEMory? [<offset in wave-points>,]<size in wave-points>.
:SEGMents			

Keyword	Parameter Form	Default	Notes
[:DATA]	[<first segment number>,] #<header><binary_block>		Delete all segments (of the selected channel) and define N consecutive new segments (no query). The N segment-lengths, expressed in bytes of wave-data, are specified by the binary-block which consists of N uint64 values (8N bytes). The new segments are allocated, one after the other, from the beginning of the arbitrary-memory space.
:FILE			
[:NAME]	#<header> <binary_block>		File path-name passed as a block of binary-data.
:OFFSet	<offset in bytes>	0	Set the start offset inside the file in bytes.
:LOAD	[[<first segment number>,]<number of segments>]	1, -1 (until last segment)	This command will load the segment table data from the file defined by the **:TRACe:SEGM:FILE:NAME** command to the Proteus unit memory. If the first segment is not specified, then the default segment is 1.
:FILE			
[:NAMe]	#<header><binary_block>		File path-name passed as a block of binary-data.
:OFFSet	<offset in bytes>	0	Set the start offset inside the file in bytes.
:DESTination	SEGMent | MEMory	SEGM	The destination to load/store the file data: **SEGMent** – The selected (programmable) segment. **MEMory** – The arbitrary-memory space.
:LOAD	[<offset in the segment in wave-points>, <size in wave-points>]		Load waveform data from file to the memory of the programmable segment (no query). If the offset and the number of wave-points are not specified, then the whole segment is written.
:STORe	[<offset in the segment in wave-points>, <size in wave-points>]		Store waveform-data from the memory of the programmable segment in file (no query). If the offset and the number of wave-points are not specified, then the whole segment is read.
:STReaming			Bypass the on-board waveform memory and stream the waveform

Keyword	Parameter Form	Default	Notes
			data straight from the controlling PC.
:MODE	FILE | DYNamic	DYNamic	Define how the data is transferred to the Proteus. **FILE** – The waveform data from a file is transferred to the Proteus. **DYNamic** – The waveform data is generated continuously and transferred to the Proteus.
:STATe	OFF | ON | 0 | 1		Enable or disable the streaming mode. Only for units with installed streaming option (STM).
:DEFine			
[:SIMPle]	[<segment_number>,] <segment_length>		Define simple segment (no query). The segment length is expressed in wave-points. The segment-number is optional (for backward compatibility). If the segment-number is not specified then the selected (programmable) segment is defined.
:LENGth?			Query the length in wave-points of the selected segment.
:ZERO			
[:SEGMent]	[<segment number>]		Reset the markers and waveform data of a single segment (no query). The segment-number is optional. If it is not given then the current segment is zeroed.
:ALL			Reset all the arbitrary-memory space of the selected channel's DDR (no query).
:DELete			
[:SEGMent]	<seg-number>		Delete one segment
:ALL			Delete all segments of the programmable channel's DDR.
:SELect			
[:SEGMent]	<segment number>		Select the programmable segment. Do not confuse it with the selected-segment for playback (:FUNCtion:SEGMent).
:SOURce	BUS | EXTernal | ADC | DCT	BUS	Source for segment selection (for playback). **BUS** – By software Request (:FUNCtion:SEGMent). **EXT** – By segment-select connector

Keyword	Parameter Form	Default	Notes
			(if this option is supported). **ADC** – By ADC trigger (if this option is supported). **DCT** – By daisy chain trigger. Future Release.
:TIMing **Future Release**	EVENtually | IMMediate	EVEN	Transition mode.
:FREE?			Query the available waveform memory
:FRAG?			Query the fragmentation level (between 0 and 1) of the of the selected channel's memory-space.
:DEFRag			Defragment the arbitrary-memory space of the selected channel (no query)

2.8 Digitizer Group Commands

Refer to section 10 Digitizer Commands, page 136.

Table 2-8 Digitizer Group Commands

Keyword	Parameter Form	Default	Notes
:DIGitizer			
[:SELect]	{DIG1 | DIG2}	DIG1	Select digitizer.
:MODE	{DUAL | SINGle}	DUAL	Select single channel use or dual channel use.
:CHANnel			
[:SELect]	{CH1 | CH2}	CH1	Select target channel for SCPI commands setup.
:STATe	{ DISabled ENABled | }	DIS	Enable/disable the channel for acquisition.
:RANGe	{ HIGH | MEDium | LOW}	HIGH	Set the voltage range.
:OFFSet	<offset_level>	0	Set offset level.
:RANGe?			Query only, It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:DDC			

Keyword	Parameter Form	Default	Notes
:MODE	{REAL|COMPlex}	REAL	Set whether data path is REAL or COMPlex. Complex Mode is available only in Dual channel mode, and forces decimation X16 (the sampling rate is sclk/16).
:DECimation?	{NONE | X1 | X4 | X16}	X16	Set the decimation factor when working in complex mode. In real mode decimation is X1. In complex mode user can set X4 or X16. For loopback mode the decimation factor must be set to X4. When querying the decimation factor, the possible responses are X1|X4|X16.
:BIND	OFF| ON| 0| 1	0	If the state is ON the channel 1 is the source of DDC1 and DDC2. Note: Only for :MODE:SINGle and sampling rate <= 2.7GS/s Otherwise, channel1 is the source of DDC1 and channel2 is the source of DDC2. Note: Only for :MODE:DUAL.
:CFRequency<N>	0Hz to MAX DIG SCLK	1e9	N = 1 or 2. Carrier frequency for the selected DDC <1|2>
:PHASE<N>	<phase in degrees>	0	N = 1 or 2. The phase of the selected DDC <1|2> (in degrees)
:CLKSource	DIG | AWG	DIG	When the state is AWG the NCO of the DUC and NCO of the DDC are synchronized to the same source. Note: This limits the DAC and ADC clocks to work within certain ranges.
:ACQuire			
[:FRAMes]			
:DEFine	<num_of_frames>, <frame_length>	14800	Reserve memory-space and define the frame layout for the digitizer channel with state enabled.
:FREE			No Query. Frees the memory allocated to the DDR.
:CAPTure			
[:SELect]	<1st frame>,<num-frames>		Define which frames (from the reserved frames) that will be captured once the capturing starts.
:ALL			Capture all frames.
:MARKer	OFF| ON| 0| 1	0	If the marker-mode is enabled then the LSB of the captured data is a marker that holds the state of the capturing trigger signal.

Keyword	Parameter Form	Default	Notes
:STATus?			Get the status of the acquisition. The format of the answer is <frame-done>,<all-frames-done>,<pulse-counter-busy>,<frames-count>.
:AVERage			
:STATe	OFF| ON| 0| 1	OFF	Enable averaging of the captured frame. Note that this limits the frame size to 10224 samples. Each frame that is defined is averaged the number of times as set in the AVER:COUN command.
:COUNt	<# frames to average can range from 2 to 16M>	2	Number of consecutive acquisitions to average.
:ZERO			
:[SELect]	<1st frame>,<num frames>,<fill value>		Set the designated entries in the acquisition memory to the "fill value. No Query. num-frames = -1, means to the last frame. Fill-value is 12bits.
:ALL	<fill value between 0 and 4095>		Fill the memory of all frames with the specified 12-bits value (no query).
:FREQuency			
[:RASTer]	{<sclk> | MAXimum | MINimum}	2.00e+9	Set digitizer SCLK.
:RANGe?			Query only, It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:SOURce	{INTernal | EXTernal}	INT	Set the digitizer clock source.
:INITiate			
[:STATe]	{OFF| ON| 0| 1}	OFF	Start or stop the acquisition.
:TRIGger			
[:IMMediate]			Force a trigger event for the digitizer when the trigger source is set to CPU.
:SOURce	{CPU|EXT CH1|CH2|TASK1 |TASK2|TASK3|TASK4| MR1|MF1|MR2|MF2	EXT	The source of the trigger that initiates the capturing of the selected digitizer.

Keyword	Parameter Form	Default	Notes
			CPU – By SCPI Command :DIG:TRIG:IMM **EXTernal** –-The external-trigger of the digitizer. **CH1/CH2** – Self trigger from channel 1 or channel 2 of the digitizer. **TASk[n]** – Trigger created by task from the n^{th} channel of the AWG (of the same module). **MR[1/2]** – Raise of the marker-bit on channel 1/2 of the digitizer. **MF[1/2]** – Fall of the marker-bit on channel 1/2 of the digitizer.
:LEVel<1|2>	from -5V to +5V	0	Set the voltage threshold level<1|2> of the external trigger of the digitizer.
:RANGe?			Query only, It returns the minimum legal value, the maximum legal value, and the default value separated by commas.
:MINimum?			Query only. It returns the minimum legal value.
:MAXimum?			Query only. It returns the maximum legal value.
:DEFault?			Query only. It returns the default value.
:SELF			
[:LEVel]	Low Range: -125mV to +125mV Medium Range: -200mV to + 200mV High Range: -250mV to +250 mV	0	Set the threshold voltage level for the self-trigger for the selected channel of the digitizer.
:TYPE	{ EDGE | GATE | WEDGe | WGATe | CUSTom }	EDGE	Set the trigger type: **EDGE** - Sets LEV1 as the trigger threshold. Slope setting will set the Pos and negative edge values. **GATE** – Sets LEV1 as the trigger threshold. Slope setting will set whether gate starts when crossing above (POS) or below the level (NEG). **WEDGe** – Window Edge. Defines a window for edge trigger can be combined with width. **WGATe** – Window Gate. Defines a window for the gate trigger. **CUSTom** – Future Release

Keyword	Parameter Form	Default	Notes
:CONDition	{GREater | SHORter}	GREater	Specifies the criteria to select time-related trigger events.
:SLOPe	{POS | NEG }	POS	Set which trigger edge to trigger on when type is set to edge.
:WINDow			
:STARt	<thrshold-level index (1/2)>, POSitive | NEGativ		Window start edge: <level-index (1/2)>, <slope (pos/neg)>
:STOP	<thrshold-level index (1/2)>, POSitive | NEGativ		Window stop edge: <level-index (1/2)>, <slop (pos/neg)>
:WIDTh	<trigger_event_width>		Set the valid trigger width when trigger type width is selected.
:HOLDoff **Future Release**	<holdoff_time>	0	Set the time the trigger input is ignored after the previous trigger event
:DELay			
[:EXTernal]	<delay_time>	0	Set the time delay of the external-trigger of the digitizer. The resolution is the digitizer's sysclock ticks which is - 10 sampling-clock ticks in case of dual-channels mode - 20 sampling-clock ticks in case of single-channel mode.
:AWG			
:TDELay	<task-trigger delay> 0 to 10s		Set the time-delay of the task-trigger from the selected AWG channel to the digitizer.
:PRETrigger	<pre-trigger length in samples>	0	Set the position of the trigger inside the frame, or in other words, how many samples that arrive before the trigger that starts the frame, should be saved in the frame. Zero means no pre-trigger.
:DATA			
:TYPE	FRAMes | HEADers | BOTH		If reading frames with headers, then each frame is followed by its header.
:SELect	ALL | FRAMes | CHUNk	ALL	Set what should be read: **ALL** – All frames. **FRAMes** – One or more frames. **CHUNk** – Chunk of data from specified frame (the :TYPE is ignored in this case).
:FRAMes	<1st-frame>,<num-frames>	1,1	Which frames to transfer.

Keyword	Parameter Form	Default	Notes
:CHUNk?	<frame-no>,<offset in samples>,<read size in samples>	1,0,-1	Acquire a data-chunk from the specified frame of the selected channel.
:READ?			Acquire the selected frames from the selected channel
:SIZE?			The size in bytes of the data that was selected for read.
:FNAMe	#<header><file-path as binary data>		Set the file-path for the :STORe command
:STORe	<offset>		Store the specified data from the captured memory of the selected channel of the digitizer in the specified offset of the predefined file (no query).
:FORMat	U16 | F32 | F64	U16	The format of the data that will be sent to the control PC. **U16** – Each 12-bits sample is contained in an uint16 (LSB is marker if marker-mode is enabled). **F32** – Each sample is converted to 32-bit floating-point value - marker is lost. Future Release. **F64** – Each sample is converted to 64-bit floating-point value - marker is lost. Future Release.
:YINCr? **Future Release**			
:YOFFset? **Future Release**			
:XINCr? **Future Release**			
:XOFFset? **Future Release**			
:LOOPback			
[:STATe]	{OFF|ON}	OFF	Enable disable loopback state for the active channel.
:DELay	<delay>	0	Set the time delay between the digitizer channel and the corresponding loopback generator channel. Values are from 8 to 2047, where each value corresponds to 16*DAC_SCLK_Period
:SYNC			Use this command to initiate a sync trigger to synchronize all loopback channels.

Keyword	Parameter Form	Default	Notes
:IQRotation	<scale>,<phase>	1,0	Use this command to set the scale and phase of the IQ rotation added to the output signal of the active generator channel. Scale can have values between 0 to 3, and phase can have values between 0 to 360.
:OVERflow?			Query if the configured operation resulted in an overflow and as a result the signal was clipped. Response is 1 if signal was clipped or 0 if not clipped.
:PULSe			
[:DEFine]	{<INTernal | EXTernal>, <FIXed | GATed>, <window_width>} 12.5ns to 15s	EXT FIX 0.1s	Define pulse counter trigger source: **INTernal** – Internal trigger-source **EXTernal –** External trigger-source Define pulse-counter trigger parameters: **FIXed** – Fixed window for the counter. **GATed –** Window type **window-width** – Time width of the counter in seconds.
:COUNt?			Response: <counter 1>,<counter 2>

2.9 DSP Commands

Refer to section 11 Digital Signal Processing Commands, page 163 for details.

Table 2-9 DSP Group Commands

Keyword	Parameter Form	Default	Notes
:DSP			
:STORe	DIRect | DSP | FFTOut	DIR	Select the data that will be stored . Note that the available options are dependent on the DDC:MODE selected in the digitizer command section.
:IQDemod			
:SELect	DBUG | IQ4 | IQ5 | IQ6 | IQ7 | IQ8 | IQ9 | IQ10 | IQ11 | IQ12 | IQ13	DBUG	Set which IQ pair to configure. DSP1 corresponds to IQ4 up to DSP10 which corresponds to IQ13.
:KERnel			Range 10240 samples, each sample is 12 bit signed FIX 12_11 (11 bit for fractional), real and

			imaginary take values between -1 to 1.
:COEFficient	<sample-number>,<real>, <imaginary>		Write the real and imaginary parts of the specified sample in the kernel of the selected IQ pair.
:DATA	#<binary-header><binary block>		Write or read the kernel data, 10240 samples, 12 bit for real and 12 bit for imaginary in 4 bytes.
:FIR			Refers to the complex and Real FIR blocks data path.
:SELect	I1|Q1|I2|Q2|DBUGI|DBUGQ	I1	Select which FIR block to configure. When mode is Complex the complex data path IQ blocks are operational (I1, I2, Q1, Q2) and in REAL mode the DBUG IQ blocks are operational (DBUGI, DBUGQ)
:BYPass	OFF| ON| 0| 1	ON	Set whether to bypass the FIR block in complex mode.
:LENGth?			Query the number of taps.
:COEFficient	<tap-number>,<the value of the specified tap>		Set the tap coefficient value by sending the index value of the desired coefficient and its new value. Value is specified between -1 to 1 (FIX 24_23). The FIR has 51 taps. Query returns the value of the coefficient specified by the index.
:DATA	#<header><binary_block>]		Write or read the taps of FIR as binary data, each tap is FIX 24_23
:FFT			Set whether data is passed through FFT or not. When FFT is ON it is saved to the DDR, frame size in digitizer must be set to 2400 samples. FFT size is constant at 1024.
:INPut	IQ1 | IQ2 | DBUG	IQ1	Select the input of the FFT. In case of REAL mode it is DBUG. In case of DIGitizer COMPlex mode it is IQ1,

			which is sourced at the ADC channel 1 or IQ2, which is sourced at ADC channel 2. When :DDC:BIND is set to ON then both IQ pairs are sourced from channel 1.
:MATH			
:OPERation	MI1 | MQ1 | MI2 | MQ2 | MI3 | MQ3 |MI4 |MQ4 | MI5 |MQ5 | MI6 |MQ6 | MI7 | MQ7 | MI8|MQ8 | MI9 |MQ9 | MI10 |MQ10 ,<SCALe>,<OFFSet>	MI1,1,0	Set the scale and offset of the specified affine transformation in the math block for the selected parameter. Scale value can be between -64 to 63, offset value between -8192 to 8191. MIX or MQX corresponds to the appropriate DSPIX and DSPQX.
:CLIP?			Query if the configured operation resulted in an overflow and as a result the signal was clipped. Response is 1 if the signal was clipped or 0 if not clipped.
:XCORrelation			
:LENGth	<N>	1024	Set the length of the cross correlation in samples, length can range from 1 to 1024 samples.
:SIGNal	MI1 | MQ1 | MI2 | MQ2 | MI3 | MQ3 |MI4 |MQ4 | MI5 |MQ5 | MI6 |MQ6 | MI7 | MQ7 | MI8|MQ8 | MI9 |MQ9 | MI10 |MQ10	MI1	Set the two signals on which to perform the cross correlation.
:RAVG	MI1 | MQ1 | MI2 | MQ2 | MI3 | MQ3 |MI4 |MQ4 | MI5 |MQ5 | MI6 |MQ6 | MI7 | MQ7 | MI8|MQ8 | MI9 |MQ9 | MI10 |MQ10 | XC,<N>		Set the window size (2^n where n=0,1…15) for the rolling average calculation.
:DECision			When using decision blocks, 1. Frame size in digitizer must be larger than decision frame size. 2. When in REAL mode since kernel is limited to 1024 samples (after decimation) then digitizer

			frame size is limited to 1024 samples. 3. When in complex mode there is no limit on digitizer frame size however since decision frame size is limited to 1024 samples only the first 1024 samples of the digitizer will be used for decision.
[:FEEDback]			
:MAPping	<awg channel number>,DEC1 | DEC2| DEC3 | DEC4 | DEC5 | DEC6 | DEC7 | DEC8 | DEC9 | DEC10 | XC		Set the decision block that affect the selected channel of the generator. Channel number can take the value of up to the number of generator channels.
:CONDition	<awg-channel-number>, S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8,<segment number>		Associate state SN of the decision block with the specified segment of the specified AWG channel
:FRAMe	<the frame-size for calculation>	1024	Set the frame size for the calculation, range from 2 to 1024 samples.
:IQPath			
:SELect	DSP1 | DSP2 | DSP3 | DSP4| DSP5| DSP6| DSP7| DSP8| DSP9| DSP10	DSP1	Set which input path to configure.
:OUTPut	THReshold|SVM	THR	Select the output of the IQpath.
:THReshold			
:LEVel	<N>	O	Set the threshold level of the decision block (between -2^23 and 2^23 - 1)
:INPut	I|Q	I	Select the input for the threshold decision.
:LINE	1 | 2 | 3, <slope>, <y intercept>		Set the slope and y intercept of the line equation (1, 2, 3) of the selected IQ Path. Slope value from -256 to 255, y-intercept value from -128 to 127.
:CLIP?			Query if the configured operation resulted in an overflow and as a result the signal was clipped.

			Response is 1 if signal was clipped or 0 if not clipped.
:XCORrellation			
:THReshold	<N>		Set the threshold value of the cross correlation (signed 24 bit).
:CLIP?			Query if the configured operation resulted in an overflow and as a result signal was clipped. Response is 1 if signal was clipped or 0 if not clipped.

2.10 System Commands

Refer to section 12 System Commands, page 182 for details.

Table 2-10 System Commands

Keyword	Parameter Form	Default	Notes
:SYSTem			
:LOG			
[:VERBose]	from 0 to 6	4	The logger verbose level (0: minimal, 6: maximal)
:ERRor?			Query for programming errors.
:INFormation			
:CALibration?			Query calibration date.
:MODel?			Query the model-name
:SERial?			Query the serial-number
:HARDware?			Query hardware version.
:FPGA			
:VERSion?			Query the FPGA FW version.
:DATE?			Query the FPGA FW build date
:SVN?			Query the FPGA SVN.
:FIRMware			
:VERSion?			Query the control PC DLL version.
:DATE?			Query the control PC DLL build date.
:SVN?			Query the control PC SVN software version.
:DAC?			Query the DAC mode. Returns M0 for 16-bit width and M1 for 8-bit width.

:SLOT?			Query the slot-number of the first slot the instrument occupies in the chassis.
:SCPI			Query the SCPI version.
[:VERSion]?			Query the version of the set of SCPI commands.
:REGisters			Query the FPGA register values in HTML format.
[:MEASure]			
:TEMPerature?			Query the temperature (°C).
:HTPeak?			Query the highest temperature recorded (°C).
:LTPeak?			Query the lowest temperature recorded (°C).
:VINTernal?			Query the internal Vcc (V).
:VAUXiliary?			Query the auxiliary Vcc (V).
:FILE			
:CATalog?			Query the file catalog.
[:NAME]	#<header><binary-block>		System file path-name passed as a block of binary data.
:SIZE?			Query the file size in bytes.
:DATA	[<offset in bytes>,] #<header><binary block>	0	Download binary-data to the specified offset in the file.
:DELete			Delete the file (no query).

2.11 SCPI Error List

Refer to 12.2.1 Error list for a list of SCPI errors that are issued by the SCPI parser.

3 Instrument Commands

Commands that are listed in the table below, control parameters for the group and synchronize two or more channels, as well as their relative offset and skew parameters. Factory defaults after the *RST command that resets the generator to its default state are shown in the Default column. The parameter range and low and high limits are listed where applicable.

Note: The command :INST CH1 / CH2 / ..CHx. The default parameter is CH1, which means that commands that are sent to the Proteus affect channel 1 settings only. Select the *INST CHx* parameter if you want to program the channel x parameters.

3.1 :INSTrument:ACTive[:SELect]{1…}(?)

Description

This command will set the active Proteus Module for future programming command sequences. Subsequent commands affect the selected Proteus Module only.

Parameters

Range	Type	Default	Description
1 …until number of modules in the chassis	Discrete	1	Select the addressed instrument (1 is the master instrument).

Response

The Proteus unit will return 1… depending on the present active module setting.

Example

Command **:INST:ACT 1**

Query **:INST:ACT?**

3.2 :INSTrument:CHANnel[:SELect]{1|2|..12}(?)

Description

This command will set the active channel (for a given module) or device (for standalone devices) for future programming command sequences. Subsequent commands affect the selected channel only.

Parameters

Range	Type	Default	Description
1 to 12	Discrete	1	Sets the active channel for programming.

Response

The Proteus unit will return 1 to 12 depending on the present active module setting.

Example

Command **:INST:CHAN 1**

Query **:INST:CHAN?**

3.3 :INSTrument:CHANnel:OFFSet{1|2|..1024}(?)

Description

Use this command to set the delay between channels in units of samples.

Parameters

Range	Type	Default	Description
0 to 1024	Discrete	0	Set the coarse offset in samples between channels.

Response

The Proteus unit will return 0 to 1024 depending on the present active module setting.

Example

Command `:INST:CHAN:OFFS 1`

Query `:INST:CHAN:OFFS?`

3.4 :INSTrument:COUPle:SKEW<ch_skew>(?)

Description

This command sets or queries the skew between Part1 (DAC1) and Part2 (DAC2) within the Proteus unit.

Parameters

Name	Range	Type	Default	Description
<ch_skew>	-2.5e-9 to 2.5e-9	Integer	0	Defines a time delay between DAC1 and DAC 2 of the Proteus module with 5ps resolution

Response

The Proteus unit will return the present value of the offset setting in units of waveform points.

Example

Command `:INST:COUP:SKEW 1.5e-9`

Query `:INST:COUP:SKEW?`

3.5 :XINStrument:MODE(?)

Description

The mode of the instrument. It can be single, master or slave. Query only.

Response

The Proteus will return SING, MAST or SLAV depending on the present mode setting.

Example

Query `:XINS:MODE?`

3.6 :XINStrument:SYNChronize:ROLE(?)

The query will return the synchronization mode of the active instrument (query only). The role is set by the system at power up and can be changed by the :FOLLowers command.

Response

The query will return the synchronization mode of the active instrument (query only).

- SINGle - Instrument is not part of a chain.
- LEADer - Instrument is the leader of the chain.
- FOLLower - Instrument is a follower in the chain.

Example

Query `:XINS:SYNC:ROLE?`

3.7 :XINStrument:SYNChronize:FOLLowers <number_of_follower_instruments>(?)

Description

Make the n consecutive instruments synchronization followers of this instrument. If n=0 and the instrument is a leader instrument, then the chain is disassembled.

Note: How to synchronize n multiple instruments

1. All instruments must have the same model ID.
2. Connect the REF out of the leader instrument to the REF in of the following instrument and so on, thus creating a daisy chain.
3. Verify that the WDS “Tabor Instrumentation Service” is running on the controlling PC.
4. To initialize the instrumentation synchronization send :XINST:SYNC:FOLL n command to the leader instrument.
5. Only the leader instrument’s external trigger TRG1 is sent to all the followers. All the other instrument triggers are not shared.
6. To stop the synchronization send :XINST:SYNC:FOLL 0 to the leader instrument.

Parameters

Name	Type	Default	Description
<number_of_follower_instruments>	Integer	0	Make the n consecutive instruments synchronization followers of this instrument.

Response

The Proteus will return number of follower instruments depending on the present type setting.

Example

Command `:XINS:SYNC:FOLL 3`

Query `:XINS:SYNC:FOLL?`

3.8 :XINStrument:SYNChronize:OFFSet < inst_offset>(?)

Future Release

Description

When couple state is ON, this command sets or queries the offset between the start phase of the selected slave instrument in reference to the master instrument.

Parameters

Name	Range	Type	Default	Description
<inst_offset>	0 to number of waveform points	Numeric(int)	0	Defines a coarse phase offset between multiple instruments. When offset is applied to one instrument it is always in reference to the other instrument. For example, offsetting the slave instrument by 1024 points and then offsetting master instrument by 2048 points will cause slave waveform to lag the master waveform by 1024 points. Offset can be programmed in increments of 8 sample clock periods.

Response

The Proteus will return the present value of the coarse offset setting in units of waveform points (SCLK periods).

Example

Command `:XINS:SYNC:OFFS 8`

Query `:XINS:SYNC:OFFS?`

3.9 :XINStrument:SYNChronize:SKEW< inst_skew>(?)

Future Release

Description

When couple state is ON, this command sets or queries the skew between the start phase of the slave instrument in reference to the master instrument.

Parameters

Range	Range	Type	Default	Description
<inst_skew>	-5e-9 to 5e-9	Numeric	0	Defines a phase offset between two instruments. When offset is applied to one instrument it is always in reference to the other instrument.

Response

The Proteus will return the present value of the skew setting in units of seconds.

Example

Command `:XINS:SYNC:SKEW 1e-09`

Query `:XINS:SYNC:SKEW?`

4 Run Mode Commands

The Run Mode Commands group is used to synchronize device actions with external or internal events. The Proteus can operate in two basic modes: self-armed and armed.

Self-armed mode is the default option where waveforms are generated at the output connector, immediately after the output function has been selected.

In armed mode, the Proteus requires an enable command or an external analog event to cause the output to start generating waveforms and when already armed, a remote abort command will cease the generation of the signal and the output will return to a known idle state. This mode is very useful to control how and when the waveform will start and stop for systems that require precise control of waveform timing respect real-world events.

Other commands in this group control the basic run modes of the waveform generator. The available run modes are:

- **CONTINUOUS** – Waveforms are generated continuously at the output connector and triggered and gated.
- **CONDITIONAL** – Waveforms are generated on conditional events, no matter if they are generated internally from a built-in trigger generator or applied externally to the trigger and event inputs.

Also use the commands in this group to control the sensitivity, the polarity and other conditions of which external signals will affect the trigger and event inputs.

A built-in counter is available to control a precise number of cycles for applications requiring a burst of waveforms that follows a trigger event.

Additional information on the run mode options and how the generator behaves under the various run mode options is given in the following sections. Factory defaults after *RST are shown in the default column. Parameter low and high limits are given where applicable. Use the commands in below to set up the Proteus run mode and for setting up the input conditions for the various trigger inputs.

4.1 :INITiate:CONTinuous[:STATe]{OFF|ON|0|1}(?)

Description

This command defines the continuous run mode of the instrument. This command does not activate the trigger sources, which must be set up and activated using additional commands.

The figure below depicts a standard trigger with a minimum instrument delay (additional user defined delay can be added). The output with Loops = 6 (the segment is played six times) will cause Trigger Event #3 to be ignored.

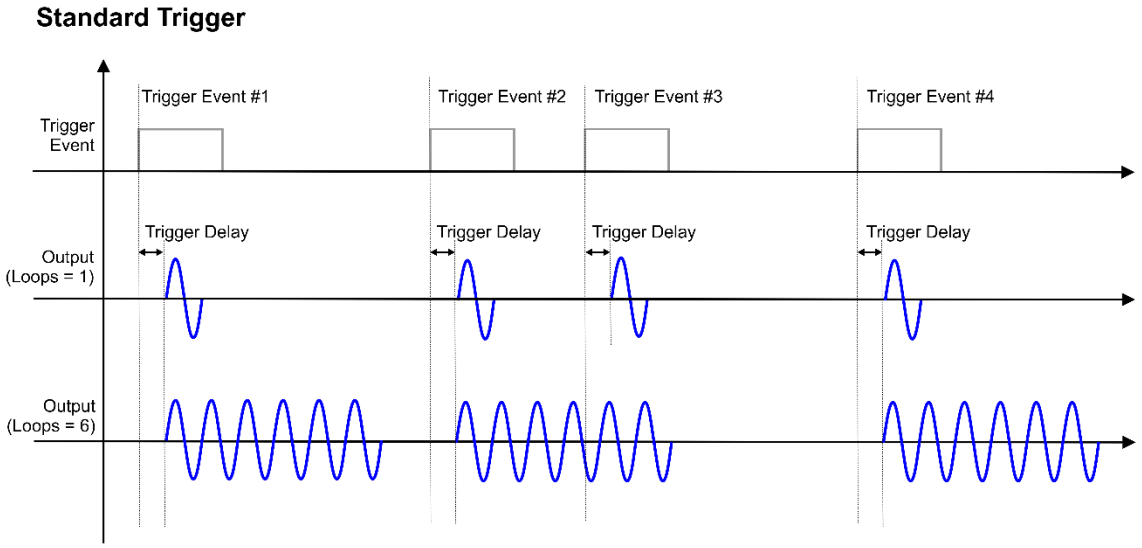

Figure 4-1 Standard Trigger

Parameters

Range	Type	Default	Description
0-1	Discrete	1	**0** – Disables the continuous operation and forces the triggered run mode. Trigger signal is applied to the trigger input only and output waveforms will be generated only when the trigger signal is valid and true. The slope and level of the trigger input are programmable. **1** – Selects the continuous run mode.

Response

The Proteus unit will return 1 or 0 depending on the current run mode setting.

Example

Command `:INIT:CONT OFF`

Query `:INIT:CONT?`

4.2 :TRIGger:COUPle[:STATe]{OFF|ON|0|1}(?)

Description

This command defines the trigger coupling between synchronized modules. When set to ON all synchronized modules will receive the trigger from the master module trigger 1. Use this when you would like all units to receive the trigger from a common trigger input. For example, in a Desktop or Benchtop multi-channel unit when the trigger couple is set to ON all channels will receive the trigger from TRIG1.

Parameters

Range	Type	Default	Description
0-1	Discrete	0	**0** – Trigger coupling between synchronized modules is off. **1** – Activates trigger coupling for all the synchronized modules.

Response

The Proteus unit will return 1 or 0 depending on the current run mode setting.

Example

Command `:TRIG:COUP OFF`

Query `:TRIG:COUP?`

4.3 :TRIGger:SOURce:ENABle{NONE|TRG1|TRG2|TRG3|TRG4|TRG5|TRG6|INTernal|CPU|FBTRg|HWControl}(?)

Description

Use this command to set or query the source of the trigger enable signal. The trigger inputs in the front panel are associated to a specific output channel depending on the Proteus model. Refer to the table “Effected Channels” below.

Parameters

Name	Type	Default	Description
<NONE>	Discrete	NONE	No source of the enable signal.
<TRG1>	Discrete		Trigger input 1
<TRG2>	Discrete		Trigger input 2
<TRG3>	Discrete		Trigger input 3
<TRG4>	Discrete		Trigger input 4
<TRG5>	Discrete		Trigger input 5
<TRG6>	Discrete		Trigger input 6
<INTernal>	Discrete		Internal trigger
<CPU>	Discrete		Bus
<FBTgR>	Discrete		Feedback trigger. Relevant for AWT digitizer option.
<HWControl>	Discrete		Dynamic jump connector.

Effected Channels

Trigger	Model	Effected Channels
TRG1	All	All channels
TRG2	All	All channels
TRG3	P1288D	CH5 -CH8

	P12812D	
	P2588D	
	P25812D	
	P1288B	
	P12812B	
	P2588B	
	P25812B	
	P9084D	CH3 - CH4
	P9086D	
	P9084B	
	P9086B	
TRG4	P1288D	CH5 -CH8
	P12812D	
	P2588D	
	P25812D	
	P1288B	
	P12812B	
	P2588B	
	P25812B	
	P9084D	CH3 - CH6
	P9086D	
	P9084B	
	P9086B	
TRG5	P12812D	CH9-CH12
	P25812D	
	P12812B	
	P25812B	
	P9086D	CH5 -CH6
	P9086B	
TRG6	P12812D	CH9-CH12
	P25812D	
	P12812B	
	P25812B	
	P9086D	CH5 -CH6
	P9086B	

Response

The Proteus will return NONE, CPU, HW, TRG1, TRG2, TRG3, TRG4, TRG5, TRG6, FBTR or INT depending on the present source selection.

Example

Command `:TRIG:SOUR:ENAB CPU`

Query `:TRIG:SOUR:ENAB?`

4.4 :TRIGger:SOURce:DISable{NONE|TRG1|TRG2|TRG3 | TRG4|TRG5|TRG6|INTernal |CPU|FBTRg|HWControl}(?)

Description

Use this command to set or query the source of the trigger disable (abort) signal.

Parameters

Name	Type	Default	Description
<NONE>	Discrete	NONE	No source of the disable signal.
<TRG1>	Discrete		Trigger input 1
<TRG2>	Discrete		Trigger input 2
<TRG3>	Discrete		Trigger input 3
<TRG4>	Discrete		Trigger input 4
<TRG5>	Discrete		Trigger input 5
<TRG6>	Discrete		Trigger input 6
<INTernal>	Discrete		Internal trigger
<CPU>	Discrete		Bus
<FBTgR>	Discrete		Feedback trigger. Relevant for AWT digitizer option.
<HWControl>	Discrete		Dynamic jump connector

Effected Channels

Trigger	Model	Effected Channels
TRG1	All	All channels
TRG2	All	All channels
TRG3	P1288D	CH5 -CH8
	P12812D	
	P2588D	
	P25812D	
	P1288B	
	P12812B	

	P2588B	
	P25812B	
	P9084D	CH3 - CH4
	P9086D	
	P9084B	
	P9086B	
TRG4	P1288D	CH5 -CH8
	P12812D	
	P2588D	
	P25812D	
	P1288B	
	P12812B	
	P2588B	
	P25812B	
	P9084D	CH3 - CH6
	P9086D	
	P9084B	
	P9086B	
TRG5	P12812D	CH9-CH12
	P25812D	
	P12812B	
	P25812B	
	P9086D	CH5 -CH6
	P9086B	
TRG6	P12812D	CH9-CH12
	P25812D	
	P12812B	
	P25812B	
	P9086D	CH5 -CH6
	P9086B	

Response

The Proteus will return NONE, CPU, HW, TRG1, TRG2, TRG3, TRG4, TRG5, TRG6, FBTR or INT depending on the present source selection.

Example

Command `:TRIG:SOUR:DIS CPU`

Query `:TRIG:SOUR:DIS?`

4.5 :TRIGger[:ACTIVE]:SELect{TRG1|TRG2|TRG3|TRG4|TRG5|TRG6|INTernal}(?)

Description

Select the trigger source as the target for the next related SCPI commands setup.

Parameters

Name	Type	Default	Description
TRG1	Discrete	TRG1	Trigger input 1
TRG2	Discrete		Trigger input 2
<TRG3>	Discrete		Trigger input 3
<TRG4>	Discrete		Trigger input 4
<TRG5>	Discrete		Trigger input 5
<TRG6>	Discrete		Trigger input 6
INTernal	Discrete		Internal trigger

Effected Channels

Trigger	Model	Effected Channels
TRG1	All	All channels
TRG2	All	All channels
TRG3	P1288D	CH5 -CH8
	P12812D	
	P2588D	
	P25812D	
	P1288B	
	P12812B	
	P2588B	
	P25812B	
	P9084D	CH3 - CH4
	P9086D	
	P9084B	
	P9086B	
TRG4	P1288D	CH5 -CH8
	P12812D	

	P2588D	
	P25812D	
	P1288B	
	P12812B	
	P2588B	
	P25812B	
	P9084D	CH3 - CH6
	P9086D	
	P9084B	
	P9086B	
TRG5	P12812D	CH9-CH12
	P25812D	
	P12812B	
	P25812B	
	P9086D	CH5 -CH6
	P9086B	
TRG6	P12812D	CH9-CH12
	P25812D	
	P12812B	
	P25812B	
	P9086D	CH5 -CH6
	P9086B	

Response

The Proteus will return TRG1, TRG2, TRG3, TRG4, TRG5, TRG6, or INT depending on the present source selection.

Example

Command `:TRIG:ACTIVE:SEL TRG1`

Query `:TRIG:ACTIVE:SEL?`

4.6 :TRIGger[:ACTIVE]:STATe{OFF|ON|0|1}(?)

Description

Enable / disable the selected external trigger (as designated by the **:TRG:SEL** command)). Enabling the trigger source is mandatory as just selecting a given trigger source will not activate the selected source.

Parameters

Range	Type	Default	Description
0-1	Discrete	0	Enable or disable the selected trigger.

Response

The Proteus will return 1 if the selected trigger is ON, or 0 if the trigger is OFF.

Example

Command `:TRIG:STAT ON`

Query `:TRIG:STAT?`

4.7 :TRIGger:CPU:MODE{LOCAL|GLOBAL}(?)

Description

When using the CPU trigger the user can select between LOCAL mode where only the active channel receives the CPU trigger or GLOBAL where all channels receive the same CPU trigger simultaneously.

Parameters

Range	Type	Default	Description
LOCAL	Discrete	GLOBAL	The active channel receives the CPU trigger.
GLOBAL	Discrete		All channels receive the same CPU trigger simultaneously.

Response

The Proteus will return the CPU mode, LOCAL or GLOBAL.

Example

Command `:TRIG:CPU:MODE LOCAL`

Query `:TRIG:CPU:MODE?`

4.8 :TRIGger:GATE[:STATe]{OFF|ON|1|0}(?)

Description

Enable or disable Gated-Mode of the selected external trigger (channel dependent). (Internal trigger has no gate mode, so its gate mode is only OFF.)

The figure below depicts that the gating signal will for “Jump Eventually” initiate the playing of the whole segment, while for “Jump Immediate” only play the segment that fits in the Gating Signal time length.

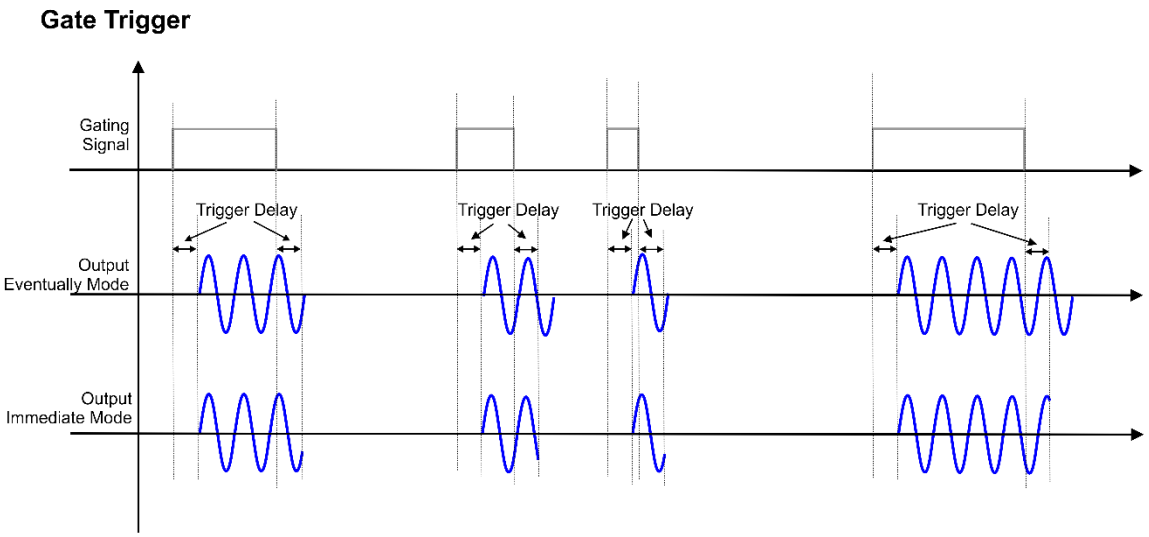

Figure 4-2 Gate Trigger

Parameters

Range	Type	Default	Description
0-1	Discrete	0	**0** – Disable the gated run mode. **1** – Enable the gated run mode. The gated run mode should only be selected if continuous run mode is off otherwise it has no effect on the current run mode. Gating signal is applied to the trigger input only and output waveforms will be generated only when the gate signal is valid and true. The slope and level of the gating entry are programmable.

Response

The Proteus will return 1 if the gated mode is enabled and 0 if the gated mode is disabled.

Example

Command `:TRIG:GATE ON`

Query `:TRIG:GATE?`

4.9 :TRIGger:LEVel<level>(?)

Description

Use this command to set or query the trigger level setting for a given trigger input selected through the :TRIG:SEL command and for the channel defined by the :INST command. This command is effective only when the Proteus unit is programmed to operate in triggered run mode (:INIT:CONT 0). The external Trigger Source must be activated using the :TRIG:STAT ON command to actually produce a trigger event for the associated channel/s.

Parameters

Name	Range	Type	Default	Description
<level>	-5 to 5, 0.1V resolution	Numeric(float)	0.0	Programs the trigger level in volt.

Response

The Proteus will return the present trigger level value.

Example

Command `:TRIG:LEV 1.2`

Query `:TRIG:LEV?`

4.10 :TRIGger:COUNt<cycles>(?)

Description

Use this command to set or query the cycles counter setting for a given trigger input selected through the :TRIG:SEL command and for the channel defined by the :INST command. This command is effective only when the Proteus unit is programmed to operate in triggered run mode (INIT:CONT 0). The command defines the number of times the current segment will be played for a given trigger signal.

Parameters

Name	Range	Type	Default	Description
<cycles>	0 – 1M	Integer	1	Programs the burst count. Following a valid trigger signal, the Proteus generates a pre-programmed number of waveform cycles and then resumes an idle state. The counted burst can be initiated using one of the following: • Front panel Man Trigger push button • Remote command such as *TRG • A transition at any of the trigger input connectors.

Response

Returns the present cycle count value.

Example

Command `:TRIG:COUN 1000`

Query `:TRIG:COUN?`

4.11 :TRIGger:WIDTh<width>(?)

Description

Use this command to set or query the trigger width value for a given trigger input selected through the :TRIG:SEL command and a given channel selected by the :INST command. Trigger signal having a pulse width below the programmed settings will not trigger the unit. Width is measured according to the threshold level set up by the :TRIG:LEV command and it refers to the pulse duration over the threshold level.

The figure below depicts that for “Ext. Trigger #3” the trigger is shorter than the “Min. Width” and it will not trigger in the “Output Trigger Width > 0”.

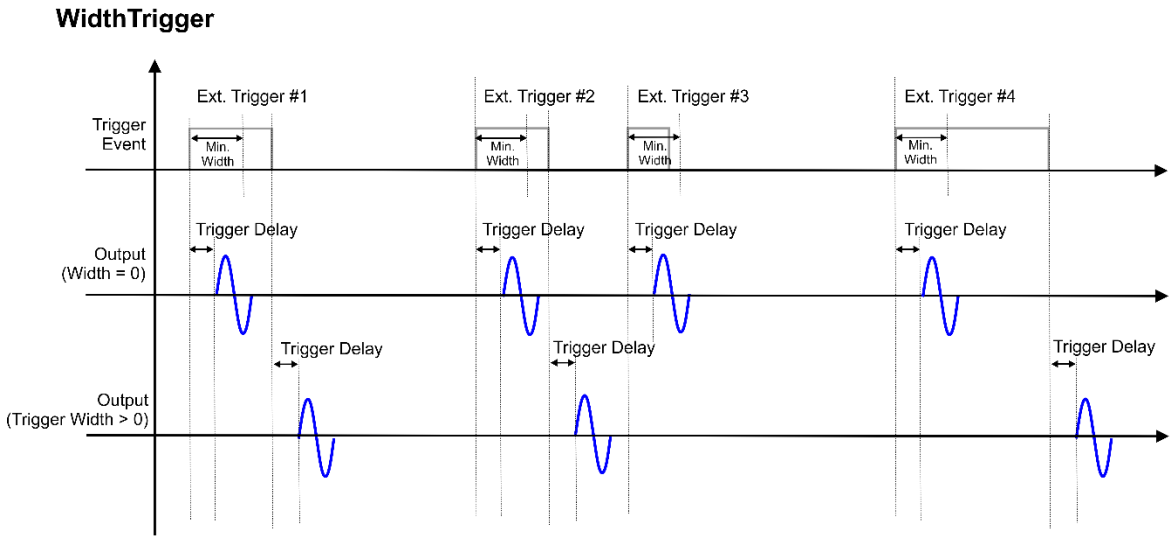

Figure 4-3 Trigger Width

Parameters

Name	Range	Type	Default	Description
<width>	0, from 10e-9 to 2s, 2ns resolution	Numeric	0	The pulse-detect width of the external trigger, zero means edge (channel dependent).

Response

The Proteus will return the present value in units of seconds.

Example

Command `:TRIG:WIDT 1e-03`

Query `:TRIG:WIDT?`

4.12 :TRIGger:SLOPe {POSitive|NEGative}(?)

Description

Use this command to define or query the valid slope for the Proteus trigger input selected through the :TRIG:SEL command and a given channel selected by the :INST command. You can choose between positive (up) and negative (down) independently for each trigger input.

Parameters

Name	Type	Default	Description
<POSITIVE>	Discrete	POS	Selects the positive (up) slope for trigger.
<NEGATIVE>	Discrete		Selects the negative (down) slope for trigger.

Response

The Proteus will return the current selection for the valid trigger slope for a given trigger input.

Example

Command `:TRIG:SLOP NEG`

Query `:TRIG:SLOP?`

4.13 :TRIGger:TIMer<time>(?)

Description

Use this command to set or query the period of the internal timed trigger generator. This value is associated with the internal trigger run mode only and has no effect on other trigger modes. The internal trigger generator is a free-running oscillator, asynchronous with the frequency of the output waveform. The timer intervals are measured from waveform start to waveform start.

Parameters

Name	Range	Type	Default	Description
<time>	200e-9 to 2.0	Numeric	1.5e-5	Programs the internal timed trigger generator period in units of seconds.

Response

The Proteus will return the present internal timed trigger period value in units of seconds.

Example

Command `:TRIG:TIM 100e-6`

Query `:TRIG:TIM?`

4.14 :TRIGger:IMMediate

Description

Use this command to trigger the Proteus unit from a remote computer. You may also use the IEEE-STD-488.2 *TRG common command that will have the same effect. This command will affect the Proteus only after you program the instrument to operate in triggered run mode and select the trigger source BUS option. Note that commands that start with TRIG affect the conditions for the trigger input only.

Example

Command `:TRIG:IMM`

4.15 :TRIGger:MODE{EVENtually|IMMediate}(?)

Description

Use this command to define or query the DISABLE (ABORT) trigger mode. In the EVENtually mode, the trigger aborts the generation of the selected segment as soon as the current loop is completed. In the IMMediate mode, the generation of the selected segment is aborted as soon as possible without waiting for the end of the current loop.

Parameters

Name	Type	Default	Description
<EVENtually>	Discrete	EVEN	Selects the Eventually disable trigger mode.

<IMMediately>	Discrete		Selects the Immediately disable trigger mode.

Response

The Proteus will return EVEN or IMM, depending on the current disable trigger mode setting.

Example

Command `:TRIG:MODE IMM`

Query `:TRIG:MODE?`

4.16 :TRIGger:LTJ[:STATe]{OFF|ON|0|1}(?)

Description

Use this command to set or query the status of the low-jitter trigger functionality for a given trigger input selected through the :TRIG:SEL command and a given channel selected by the :INST command.

Parameters

Name	Type	Default	Description
0-1	Discrete	1	**0** – Low-Jitter Trigger deactivated. **1** – Activates the Low-Jitter Trigger functionality.

Response

The Proteus will return 0 or 1 depending on the current low-jitter trigger functionality status.

Example

Command `:TRIG:LTJ ON`

Query `:TRIG:LTJ?`

4.17 :TRIGger:IDLE:[TYPE]{ DC | FIRSt | CURRent }(?)

Description

Use this command to define or query the trigger mode. In normal mode, the first trigger activates the output and consecutive triggers are ignored for the duration of the output waveform. In override mode, the first trigger activates the output and consecutive triggers restart the output waveform, regardless of if the current waveform has been completed or not.

Parameters

Name	Type	Default	Description
<DC>	Discrete	DC	Selects a DC level.
<FIRSt>	Discrete		Selects the first sample of the waveform.
< CURRent >	Discrete		Keeps playing the current segment.

Response

The Proteus will return FIRS, DC or CURR depending on the current idle mode setting.

Example

Command **:TRIG:IDLE FIRS**

Query **:TRIG:IDLE?**

4.18 :TRIGger:IDLE:LEVel<level>(?)

Description

Use this command to set or query the DC level for the idle state when the mode has been set to DC with the :TRIG:IDLE DC command.

Parameters

Name	Range	Type	Default	Description
<level>	0 to 65,535/256	Numeric	32,768/128	Programs the DC level during the idle state in DAC quantization levels.

Response

The Proteus will return the present DC level set for the idle state.

Example

Command **:TRIG:IDLE:LEV 12000**

Query **:TRIG:IDLE:LEV?**

4.19 :TRIGger:PULSe[:STATe]{ OFF|ON|0|1}(?)

Future Release

Description

Use this command to set or query the status of the pulse trigger for a given trigger input selected through the :TRIGger:ACTive:SELect command and a given channel selected by the :INST:CHAN command.

Parameters

Range	Type	Default	Description
0-1	Discrete	0	Turns the pulse trigger on and off.

Response

The Proteus will return 0 or 1 depending on the present pulse trigger state.

Example

Command **:TRIG:PULS ON**

Query **:TRIG:PULS?**

4.20 :TRIGger:PULSe:COUNt< count>(?)

Future Release

Description

Use this command to query the number of pulses for pulse trigger mode when it has been set activated with the :TRIGger:PULSe[:STATe] on command.

Parameters

Name	Range	Type	Default	Description
<count>	1 to 4,284,967,295	Numeric	0	Counts the number of trigger pulses received. Maximum rate is half of the sample rate.

Response

The Proteus will return the present number of pulses for pulse trigger.

Example

Query `:TRIG:PULS:COUN?`

4.21 :TRIGger:PULSe:COUNt:RESet

Future Release

Description

Use this command to re-start the pulse counter associated to the pulse trigger when the trigger mode when it has been set activated with the :TRIG:PULS on command.

Example

Command `:TRIG:PULS:COUN:RES`

4.22 :TRIGger:DELay<delay>(?)

Description

Use this command to set or query the period of time between a valid trigger event and the action triggered by it.

Parameters

Name	Range	Type	Default	Description
<delay>	external-trigger: 0 to at least 6.55µs. Resolution: DAC mode M0: 8SCLKs DAC mode M1: 32SCLKs internal-trigger: only 0	Numeric	0	Programs the internal delay timer to delay the action triggered by a valid external trigger event.

Response

The Proteus will return the present internal trigger delay value in units of seconds.

Example

Command `:TRIG:DEL 10e-06`

Query `:TRIG:DEL?`

4.23 :TRIGger:HOLDoff< holdoff>(?)

Future Release

Description

Set the holdoff time for the selected external-trigger of the selected channel. Incoming trigger will be ignored during the holdoff period.

The figure below depicts that “Ext. Trigger #3” will be ignored and the segment will not be played when “Output Trigger Holdoff > 0” because the time distance to “Ext. Trigger #2” is shorter than the “Holdoff Time”.

Trigger Holdoff

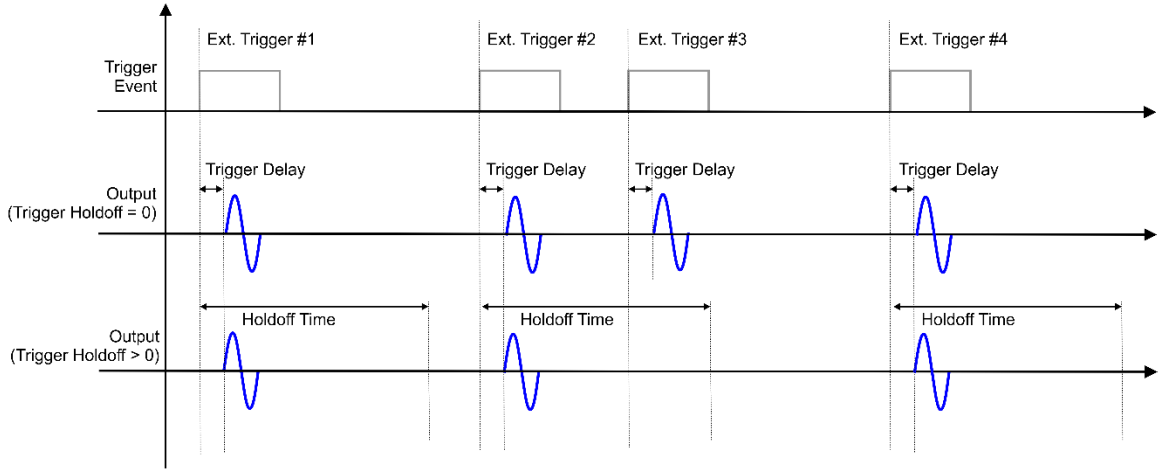

Figure 4-4 Trigger Holdoff

Parameters

Name	Range	Type	Default	Description
< holdoff >	external-trigger: from 0 to TBD. Internal-trigger: only 0 (no holdoff	Numeric	100ns external trigger	Set the holdoff of the selected external trigger of the selected channel.

Response

The Proteus will return the present external trigger holdoff value in units of seconds.

Example

Command `:TRIG:HOLD 10e-6`

Query `:TRIG:HOLD?`

5 Analog Output Control Commands

The Analog Output Control Commands group is used for programming the characteristics of the output waveform. Notice that there are two main subsystems that control output functions: OUTPut and SOURce. The output subsystem commands control parameters that are related directly to the output terminals (main and sync outputs) and the source subsystem commands program parameters that control waveform shape, frequency and level.

The Proteus has several output modules to choose from when ordering the instrument. There is an option of a DC-coupled, single-ended or differential output with variable amplitude, and a optional direct DAC differential output, AC-coupled. The various modules differ in how the outputs behave in time and frequency domains. The DC module offers 4.5 GHz bandwidth with up to 1.2 Vpp amplitude while the AC-coupled, direct-DAC offers a 9 GHz analog bandwidth with a 550 mVpp amplitude.

Other commands in this group control the SYNC parameters: type, position and width. Also use the commands in this group to control the shape of the output waveform, its frequency, its output level (or power) and the source of the clock reference.

Factory defaults after *RST are shown in the default column. Parameter low and high limits are given where applicable. Use the following commands to set up the Proteus output waveforms, and their associated characteristics.

5.1 :OUTPut[:STATe]{ OFF | ON | 0 | 1 }(?)

Description

This command will set or query the output state of the channel specified by the previous :INSTrument:CHANnel:SELect command. Note that for safety, the outputs always default to off, even if the last instrument setting before power down was on. Also note that the offsetting leaves the output connector connected to the amplifier path but no signal is being generated while in the off state.

Parameters

Range	Type	Default	Description
0-1	Discrete	0	Sets the output to on or off.

Response

The Proteus will return 1 if the output is on, or 0 if the output is off.

Example

Command **`:OUTP ON`**

Query **`:OUTP?`**

5.2 [:SOURce]:MODE{ DIRect |NCO | DUC }(?)

Description

This command will set or query the arbitrary generation mode of the channel specified by the previous :INSTrument:CHANnel:SELect command. There three modes: Direct, NCO, and DUC. The direct mode uses one sample per sampling period and samples are applied directly to the DAC.

The NCO mode internally generated a sine wave with the frequency and phase set by the :SOUR:CFR and :SOUR:CPH respectively. The IQ mode uses two samples (I, or in-Phase, and Q, or quadrature) per sampling period in order to feed the associated quadrature modulator in the DUC (Digital Up-Converter) for each channel. The quadrature modulator uses two NCOs (Numerically Controlled Oscillator) whose operating frequency may be set using the :SOUR:CFR command.

Parameters

Name	Type	Default	Description
DIRect	Discrete	DIR	Selects direct mode for waveform generation.
NCO	Discrete		Selects the NCO mode for the internal sinewave waveform generation.
DUC	Discrete		Selects the IQ (quadrature modulation) model in the internal DUC for waveform generation.

Response

The Proteus will return DIR if the generation mode is direct, DUC if the generation mode is IQ, and NCO if the generation mode is NCO.

Example

Command **`:MODE NCO`**

Query **`:MODE?`**

5.3 [:SOURce]:PTRepeat{ X1 | X2 | X4 | X8 }(?)

Description

Set the point repeat factor for the channel previously selected using the :INSTrument:CHANnel:SELect command. The Point Repeat factor enables the user to configure the unit so that each sample point that is sent to the FPGA is repeated by the point repeat factor. This essentially enables the user to lower the SCLK below the minimum 1GS/s limit. e.g. if point repeat is set to x4, each sample is sent 4 times to the DAC, and thus if the SCLK is 1GS/s the output appears as if the SCLK is 250MS/s. Note that this can be used only with segments that are normal (not fast).

Parameters

Name	Type	Default	Description
X1	Discrete	X1	Point repeat factor x1.
X2	Discrete		Point repeat factor x2.
X4	Discrete		Point repeat factor x4.
X8	Discrete		Point repeat factor x8.

Response

The Proteus will return X1, X2, X4, or X8 depending on the active point repeat factor.

Example

Command **`:PTR X8`**

Query **`:PTR?`**

5.4 [:SOURce]:INTerpolation{ NONE|X2|X4|X8}(?)

Description

This command will set or query the interpolation factor of the channel specified by the previous :INSTrument:CHANnel:SELect command. The AWG will interpolate in real-time 2, 4, or 8 samples for each actual sample stored in the waveform memory. Final sampling rate will be multiplied by the same factor respect the sampling rate of the samples in the waveform memory. Sampling rate for the stored data is the one set for the DAC divided by the interpolation factor (2, 4, or 8).

Parameters

Name	Type	Default	Description
NONE	Discrete	NONE	No interpolation.
X2	Discrete		Interpolation factor x2.
X4	Discrete		Interpolation factor x4.
X8	Discrete		Interpolation factor x8.

Response

The Proteus will return NONE, X2, X4, or X8 depending on the active interpolation factor.

Example

Command `:INT X8`

Query `:INT?`

5.5 [:SOURce]:NCO:MODE{ SINGle|DUAL}(?)

Description

Set the NCO mode. In dual mode, the user can control two NCOs (1 or 2) per channel.

Parameters

Name	Type	Default	Description
SINGle	Discrete	SING	Set the NCO mode to single mode.
DUAL	Discrete		Set the NCO mode to double mode.

Response

The Proteus will return SING or DUAL depending on the NCO mode.

Example

Command `:NCO:MODE DUAL`

Query `:NCO:MODE?`

5.6 [:SOURce]:NCO:CFRequency<1|2> <carr_freq>(?)

Description

Use this command to set or query the carrier frequency for the selected NCO <1|2> of the selected channel. It will be effective when the waveform generation mode is set to DIR, NCO, or DUC, refer to 5.2 [:SOURce]:MODE{ DIRect |NCO | DUC }(?), page 77.

Parameters

Name	Range	Type	Default	Description
< carr_freq >	0 Hz to sclk	Numeric	4e+08	Will set the carrier frequency in Hz, while generation is carried out in the IQ or NCO modes.

Response

The Proteus unit will return the present carrier frequency value. The returned value will be in standard scientific format (for example: 1 GHz would be returned as 1e9 – positive numbers are unsigned).

Example

Command `:NCO:CFR1 5.0e9`

Query `:NCO:CFR1?`

5.7 [:SOURce]:NCO:PHASe<1|2> {<phase in degrees>}(?)

Description

Use this command to set or query the phase (in degrees) for the selected NCO <1|2> of the selected channel.

Parameters

Name	Range	Type	Default	Description
<phase in degrees>	0 to 360	Numeric	0	Will set the NCO phase (in degrees) of the selected channel.

Response

The Proteus unit will return the NCO phase.

Example

Command `:NCO:PHAS1 5`

Query `:NCO:PHAS1?`

5.8 [:SOURce]:NCO:SIXDb<1|2>{ OFF|ON|0|1}(?)

Description

This command will set or query the 6dB gain for the selected NCO<1|2>.

Parameters

Name	Type	Default	Description
0-1	Discrete	0	Sets the 6dB gain on and off.

Response

The Proteus will return 1 if the 6dB gain is on, or 0 if the 6dB gain is off.

Example

Command	`:NCO:SIXD1 ON`
Query	`:NCO:SIXD1?`

5.9 [:SOURce]:IQModulation {NONE|HALF|ONE|TWO}(?)

Description

Set the IQ modulation type. It is shared by all channels in the module and by all modules in a synchronized master-slaves chain. The IQ modulation type are classified by the number of IQ pairs per channel.

IQ modulation requires one or more complex waveform, s(n) = I(n) + j x Q(n), to work. The different modes express the complex waveforms in a different way according to the internal processing performed over them.

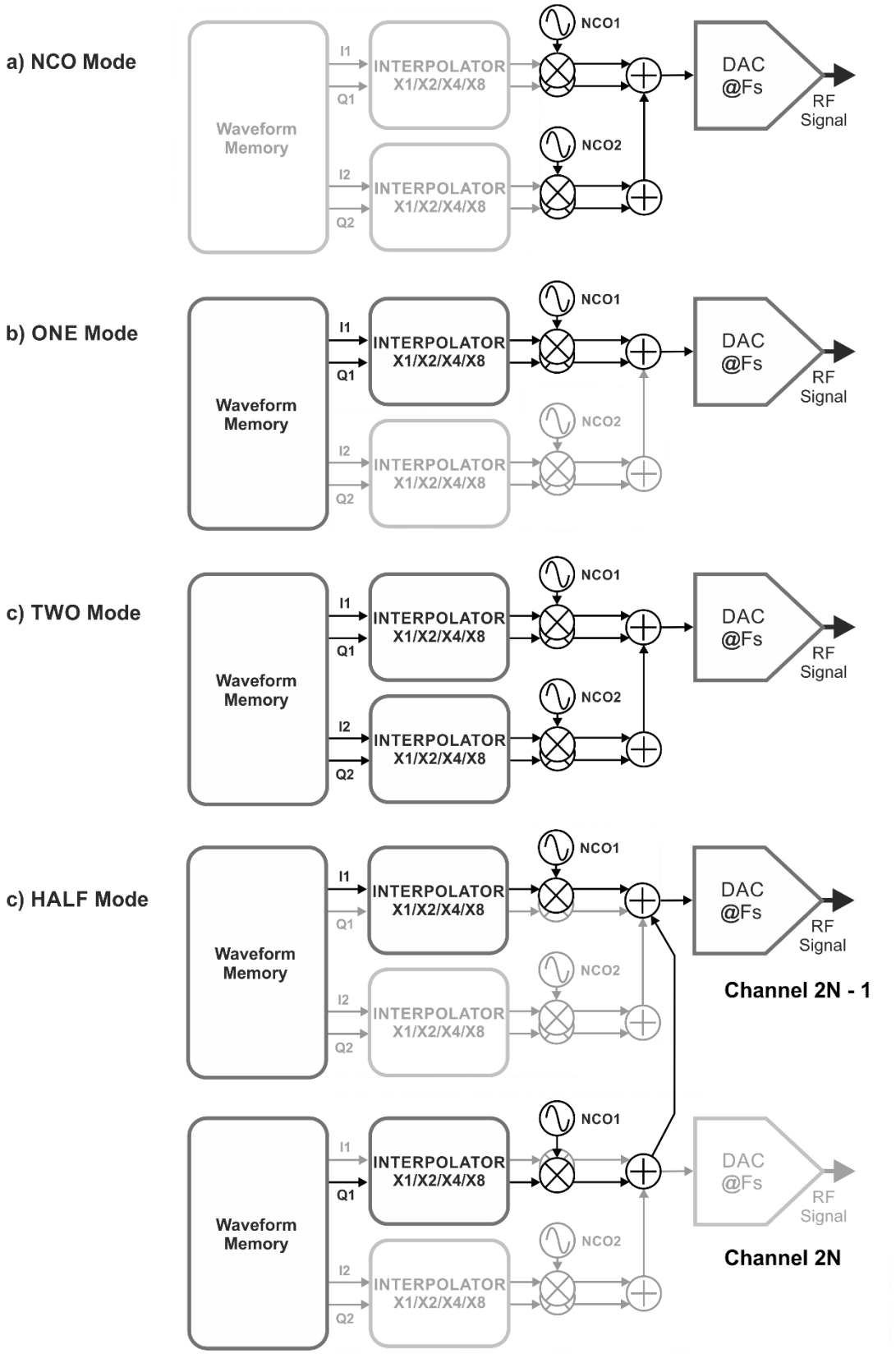

Figure 5-1 IQ Modulation Modes Block Diagram

The HALF mode works using two channels to process the I and Q waveform separately. However, the second channel output samples are routed internally to an adder combining them in a single sample stream fed to the DAC associated to the first channel. As a result, the number of active output channels is reduced by half. The advantage of this mode is doubling the sampling rate of the input waveform, thus increasing modulation bandwidth in the same proportion. Channels associated to the same waveform memory are the ones to be combined in the HALF mode, so the first, active channel, associated to the I waveform, will be always the odd numbered channel #n, while the second channel, associated to the Q waveform, will be channel # n+1, and the DAC associated to this channel will not be generating any signal while in this mode. I and Q waveforms must be downloaded separately to the designated segments, segments must be assigned to each one of the participating channels, the NCO for each channel must be set to the same frequency and phase separately as any regular waveform. Interpolation factors can be selected depending on the sampling clock settings. The sampling rate for the baseband waveforms (before interpolation), Fs/Interpolation, equal to the complex waveform sample rate, must be always lower or equal than 2.5GSa/s.

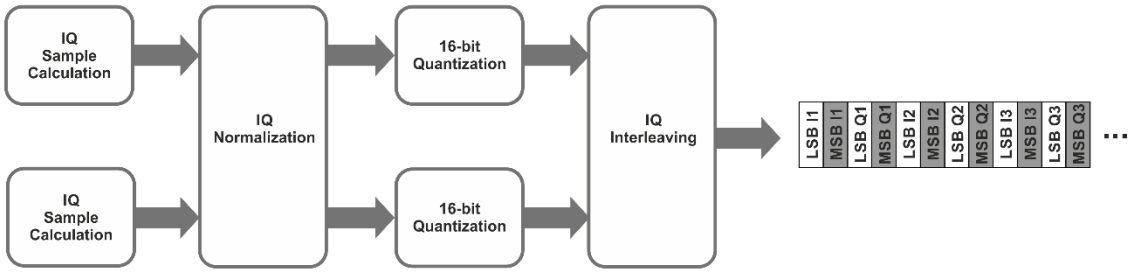

Figure 5-2 One-mode IQ Modulation Data Formatting

The ONE mode feeds one IQ pair to each channel independently, so all channels can be active simultaneously. In this case, as the complex IQ waveform is fed to the same channel, waveforms will consist in a series of interleaved I, Q samples stored in the same waveform segment. Segment length must be set to twice the number of complex samples, then. Interpolation factor can be selected according to the sampling clock setting. The maximum sampling rate for the IQ baseband waveforms COMBINED is 2.5 GSa/s, so the maximum sampling rate for the complex waveform is 1.25 GSa/s.

The TWO mode uses both IQ modulators (and associated NCOs) for each channel. Although both IQ pairs must be calculated to have the same number of samples and the same sample rate, they can use any valid Carrier Frequency and the waveforms can be independently defined. As both IQ pairs will share the same channel, samples for both complex waveforms will be stored as double-interleaved sequences in the waveform memory. This means that the segment length must be four times the number of complex samples for one of the IQ streams. The combined maximum sample transfer rate from the segment must be equal or lower than 2.5 GSa/s, so 1.25GSa/s for each IQ pair. This means than each complex sample rate must be sampled at 612.5 MSa/s or less. Data must be formatted in a particular way for the TWO mode. Although the final goal is interleaving the I and Q samples for both pairs, the data must be properly formatted for the waveform to work properly in the TWO mode. This is a step-by-step description of the waveform data formatting process in the TWO mode:

- Calculate, normalize and quantize to 16-bit unsigned integers the I1, Q1, I2, and Q2 waveforms.
- Split the unsigned 16-bit integers in two bytes (unsigned 8-bit integers), MSB and LSB.
- Take the MSBs and LSBs of each sample in each waveform and interleave them in the sequence I1M, Q1M, Q2M, I2M, I1L, Q1L, Q2L, I2L.
- Download the formatted waveform data to the target segment.

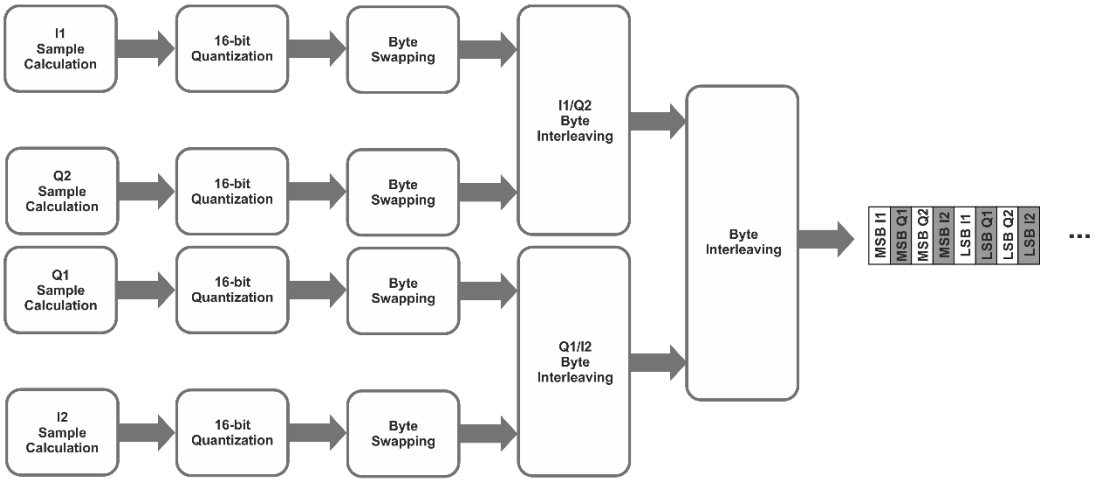

Figure 5-3 Two-mode IQ Modulation Data Formatting

Waveforms to be used in the IQM (DUC) mode must be carefully normalized to avoid DAC clipping. While I and Q waveform may use the full 16-bit range without clipping, the combined IQ modulated waveform can go beyond the DAC range as seen here:

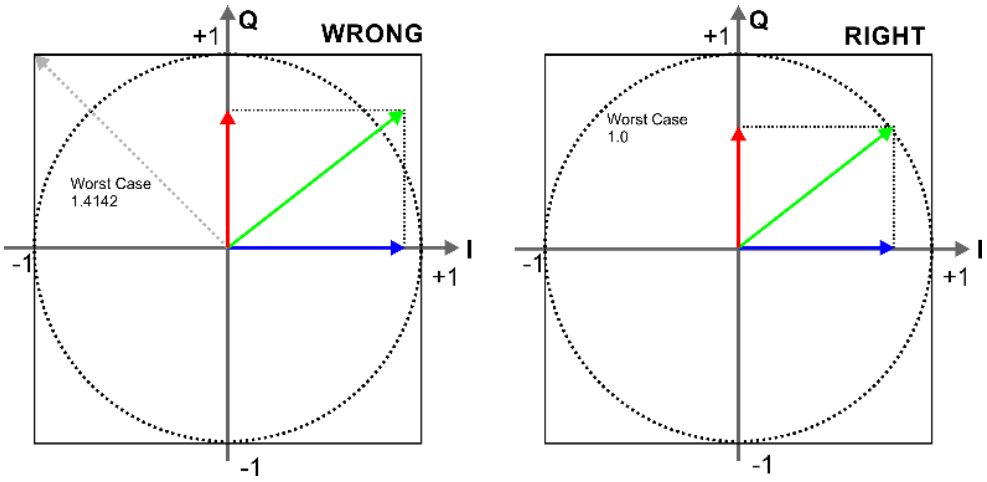

Figure 5-4 DAC Clipping in IQM Mode

The solution to this issue is normalizing according to the module of the combined I and Q signal (M = sqrt(I^2 + Q^2)). Both the I and Q waveforms must be normalized using the same factor so its module never goes beyond the DAC range. I and Q waveforms must be normalized using the same value to keep the quadrature valance in the modulation although this may result in a non-

optimal individual normalization for the baseband IQ waveforms. In the TWO mode, two IQ pairs are combined with carriers at arbitrary frequencies. Both signals are combined by an adder before being converted to analog signals. Again, it is important that the combination of both signals do not go beyond the DAC range as it is shown here:

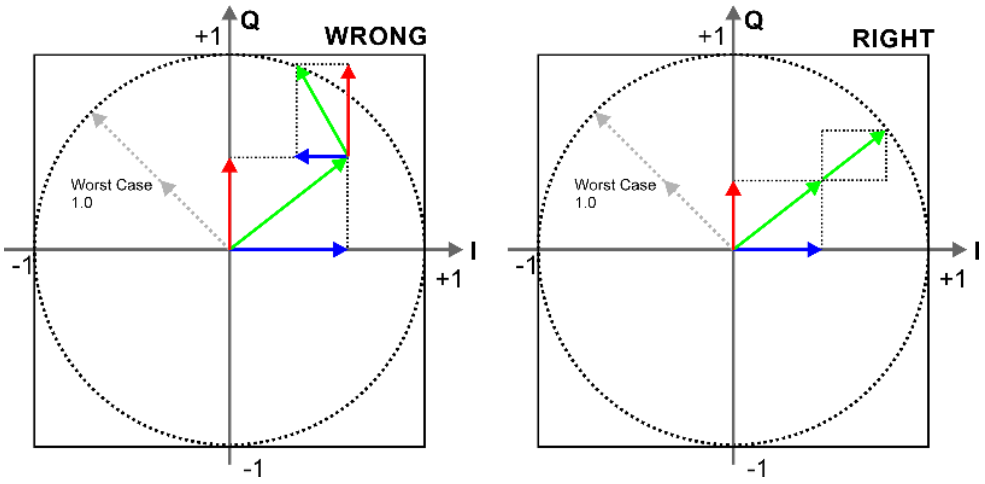

Figure 5-5 DAC Clipping in IQM TWO Mode

In the TWO mode, the best way to handle normalization is by using the addition of the module for both IQ pairs and then normalize all four components together so the worst case combination does not go out of the DAC range.

Quantization must be also performed carefully as it may result in baseband waveforms with some DC component that will show up as a residual carrier in the final IQ modulated waveform. First, the normalized signal must keep the original DC level, what can result in asymmetrical negative and positive peaks. Second the right range for the quantization must go from code 1 (and not zero) to code 2^N-1, where N is the resolution of the samples (i.e. 16 for Proteus). If the full range is used, and small but noticeable residual carrier will show up in the output signal as the number of valid levels in the positive and negative ranges around the mid-level value (1000x for the 16-bit samples in Proteus) are not equal in size as it can be seen here for a 4-bit quantizer:

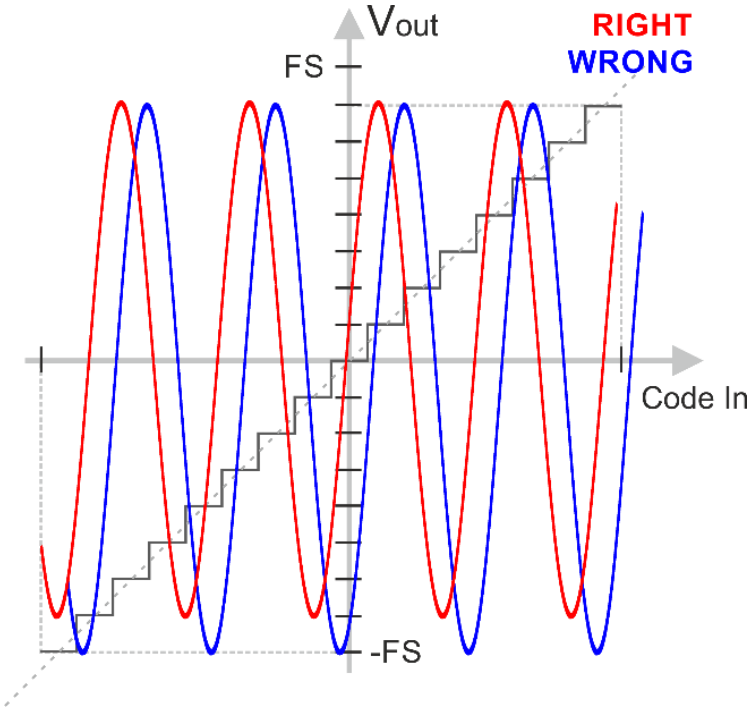

Figure 5-6 Quantization of an IQ Waveform

Parameters

Name	Type	Default	Description
HALF	Discrete	NONE	'I' in channel 1 and 'Q' in channel 2.
ONE	Discrete		1 IQ-Pair, organized in pairs of 'I' sample followed by 'Q' sample.
TWO	Discrete		2 IQ pairs organized in 4-tuples of the form (I1,q1,I2,q2).

Response

The Proteus will return the IQ modulation type.

Example

Command `:IQM ONE`

Query `:IQM?`

5.10 [:SOURce]:FREQuency[:RASTer]{<sclk>|MINimum|MAXimum}(?)

Description

Use this command to set or query the sample clock frequency for the DAC in units of samples per second (Sa/s). The actual waveform sample rate will be determined by the combination of the DAC's sample rate and the corresponding repetition factor (set by the [:SOURce]:PTRepeat command) or the interpolation factor (set by the [:SOURce]:INTerpolation command).

Parameters

Name	Range	Type	Default	Description
< sclk >	1e9 to 1.25 (P128x), 2.5 (P258x), or 9e9 (p908x and P948x	Numeric	1e+09	Will set the sample clock frequency of the arbitrary and sequenced waveform in units of Sa/s. The sample clock command can be programmed with resolutions up to 10 decimal places.
<MINimum>		Discrete		Will set sample clock to 1e09 Sa/s.
<MAXimum>		Discrete		Will set sample clock to maximum rate.

Response

The Proteus unit will return the present sample clock frequency value. The returned value will be in non-scientific format with 3 decimal digits (for example: 1 GHz would be returned as 1000000000.000 – positive numbers are unsigned).

Example

Command `:FREQ 5.0e+09`

Query `:FREQ?`

5.11 [:SOURce]:FREQuency:SOURce{ INTernal|EXTernal}(?)

Description

Use this command to select or query the source of the sample clock generator for all channels in a Proteus unit. This command affects all of the waveforms, as the internal clock is removed, and external clock is applied. Make sure that a valid clock is applied to the external clock input before you change the option to external, because the generator cannot generate waveforms without a valid source of sample clock generator. Note that the internal sample clock generator is unique for each 4-channel group however, when an external clock source is selected, the same source is applied to all channels.

Parameters

Name	Type	Default	Description
INTernal	Discrete	INT	Selects the internal clock generator as the main clock source.
EXTernal	Discrete		Activates the external sample clock input. A valid signal must be applied from the external signal to the Proteus for the generator to continue generating waveforms. Observe the input level and limitations before connecting an external signal to the external sample clock input.

Response

The Proteus will return INT, or EXT depending on the current sample clock source setting.

Example

Command `:FREQ:SOUR EXT`

Query `:FREQ:SOUR?`

5.12 [:SOURce]:FREQuency:OUTPut[:STATe]{OFF|ON|0|1} (?)

Description

This command will set or query the output state of the SCLK clock for all channels.

Parameters

Range	Type	Default	Description
0-1	Discrete	0	Sets the output on and off.

Response

The Proteus will return 1 if the output is on, or 0 if the output is off.

Example

Command `:FREQ:OUTP ON`

Query `:FREQ:OUTP?`

5.13 [:SOURce]:FUNCtion:MODE[:TYPE] {ARBitrary|TASK}(?)

Description

Use this command to set or query the type of waveform that will be available at the output connector.

Parameters

Name	Type	Default	Description
<ARBitrary>	Discrete	ARB	Selects the arbitrary waveform shapes. Arbitrary waveforms must be loaded to the Proteus memory before they can be replayed. Refer to 9 Arbitrary Waveform Commands, page 122.
<TASK>	Discrete		Selects one of the sequences defined in the task lists

Response

The Proteus will return ARB or TASK depending on the present output function mode setting.

Example

Command `:FUNC:MODE TASK`

Query `:FUNC:MODE?`

5.14 [:SOURce]:FUNCtion:MODE:SEGMent <segment_number>(?)

Description

Use this command in case of Arbitrary mode to set or query the active segment to be played back for the user generation mode for the channel selected with the :INST:ACT and :INST:CHAN commands. The first 128 segment are "Fast-Segments".

Parameters

Name	Range	Type	Default	Description
<segment_number>	1-64k	Integer	1	Will set the segment number to be played back. If segment # is greater than the last available segment, then the last segment will be played back. Argument 0 or no argument, currently selected segment will be played back.

Response

The Proteus unit will return the designated segment to be played back.

Example

Command `:FUNC:MODE:SEGM 123`

Query `:FUNC:MODE:SEGM?`

5.15 [:SOURce]:FUNCtion:MODE:TASK< task_number>(?)

Description

Use this command in case of Task-Mode to set or query the initial task to be played back for the task generation mode for the channel selected with the :INST:ACT and :INST:CHAN commands.

Parameters

Name	Range	Type	Default	Description
<task_number>	1-64k	Integer	1	Will set the task number to be played back. If task # does not point to a start sequence or single tasks, then the first previously available start task will be designated for play back.

Response

The Proteus unit will return the designated first task number to be played back.

Example

Command `:FUNC:MODE:TASK 3`

Query `:FUNC:MODE:TASK?`

5.16 [:SOURce]:ROSCillator:SOURce{ INTernal|EXTernal}(?)

Description

Use this command to set or query the source of the 10 MHz reference clock. This source defines the accuracy and stability of the clock generator. The internal reference has an accuracy and stability of 1 ppm; applications requiring higher accuracy or stability can use an external reference clock and use an improved 10 MHz or, alternatively, 100MHz signal

Parameters

Name	Type	Default	Description
INTernal	Discrete	INT	Selects an internal source. The internal source is a TCXO (Temperature Compensated Crystal Oscillator) device that has 1ppm accuracy and stability over the operating temperature range.
EXTernal	Discrete		Reroutes the 10 MHz source to the external reference input. An external reference must be connected to the Proteus for it to continue with its normal operation.

Response

The Proteus will return INT, or EXT depending on the present 10 MHz clock reference source setting.

Example

Command `:ROSC:SOUR EXT`

Query `:ROSC:SOUR?`

5.17 [:SOURce]: ROSCillator:FREQuency{ 10M|100M}(?)

Description

Use this command to set or query the frequency range that will be applied to the reference oscillator input. The frequency value must be close to the value of the external frequency because it sets up the PLLs for the reference oscillator to accept and lock on the correct external frequency value.

Parameters

Name	Type	Default	Description
10M	Discrete	100M	Sets the frequency of the external reference to 10 MHz.
100M	Discrete		Sets the frequency of the external reference to 100 MHz.

Response

The Proteus will return 10M, or 100M depending on the present external clock reference source setting.

Example

Command `:ROSC:FREQ 100M`

Query `:ROSC:FREQ?`

5.18 [:SOURce]:VOLTage[:AMPLitude] {<amplitude>|MINimum|MAXimum}(?)

Description

Use this command to set or query the voltage amplitude of the waveform for the currently selected channel. The Proteus displays a calibrated value when on load impedance of 50 Ω offset and amplitude settings are independent providing that the “offset + amplitude/2” value does not exceed the specified voltage window. This command does not apply to the AC output module.

Parameters

Name	Range	Type	Default	Description
< amplitude >	1e-3 to 1.2	Numeric	0.5	Will set the amplitude of the output waveform in units of volts. The display shows the correct amplitude level only when the output cable is terminated into 50Ω. The range varies depending on the output module installed.
MINimum	Discrete			Will set the amplitude to the lowest possible level.
MAXimum	Discrete			Will set the amplitude to the highest possible level.

Response

The Proteus will return the present DAC amplitude value. The returned value will be in standard scientific format (for example: 100 mV would be returned as 100e-3, positive numbers are unsigned).

Example

Command `:VOLT 0.8`

Query `:VOLT?`

5.19 :SOURce]:VOLTage:OFFSet{<offset>|MINimum|MAXimum}(?)

Description

Use this command to set or query the DC offset of the output waveform for the currently selected channel. The Proteus unit displays a calibrated value with a load impedance of 50 Ω. Offset and amplitude settings are independent providing that the |offset + amplitude| value does not exceed the specified amplitude window. This command does not apply to the DIRECT output option as it is AC-coupled.

Parameters

Name	Range	Type	Default	Description
< offset >	-0.5 to 0.5	Numeric	0	Will set the offset of the output waveform in units of volts. The display shows the correct offset level only when the output cable is terminated into 50Ω.
<MINimum>	Discrete			Will set the offset to the lowest possible level.
<MAXimum>	Discrete			Will set the offset to the highest possible level.

Response

The Proteus unit will return the present dc offset value. The returned value will be in standard scientific format (for example: 100 mV would be returned as 100e-3 – positive numbers are unsigned).

Example

Command `:VOLT:OFFS 0.5`

Query `:VOLT:OFFS?`

6 Marker Output Commands

The Marker Output Control Commands group is used for programming the characteristics of the marker outputs. Each Proteus module has up to 8 single-ended marker outputs. These outputs are located in the front panel. Each marker can be programmed to have unique properties such as: delay, position, width and level. Depending on the Proteus version and the total number of channels and marker outputs, there may be a total of up to two (2) or four (4) markers per channel. Markers are downloaded to the target generator unit independently of waveform data.

Specific physical marker outputs are identified by a number ranging for the currently active channel 1 to 4 for two AWG channel configurations or 1 to 2 for four channel configurations. Marker transitions cannot be defined for all the samples. The 1.25GSa/s and 2.5GSa/s Proteus models can change the marker state every two samples, while the 9GSa/s units can do it every eight samples. This means that the maximum number of marker states that can be defined for a given segment is WL/2 or WL/8 respectively, where WL (waveform length) is the size of the segment in samples.

Factory defaults after *RST are shown in the default column. Parameter low and high limits are given where applicable. Use the commands in Table 2-4 Marker Output Commands to set up the Proteus marker outputs and their associated parameters.

Marker data is transferred to/from the Proteus waveform memory in an efficient way so all the time-aligned states for the markers associated to the selected channel are transferred in a single byte. For the 1.25GS/s and 2.5GS/s Proteus models (and also for the P948X models when used at sampling rates equal to or lower than 2.5GS/s) two consecutive states per byte are transferred, one using the 4 LSBs and the next using the 4 MSBs. For the 9GS/s models, one state per byte is transferred. This means that the maximum size (in bytes) of the transfer is waveform_length / 4 for the 1.25/2.5GS/s models, and waveform_length / 8 for the 9GS/s models.

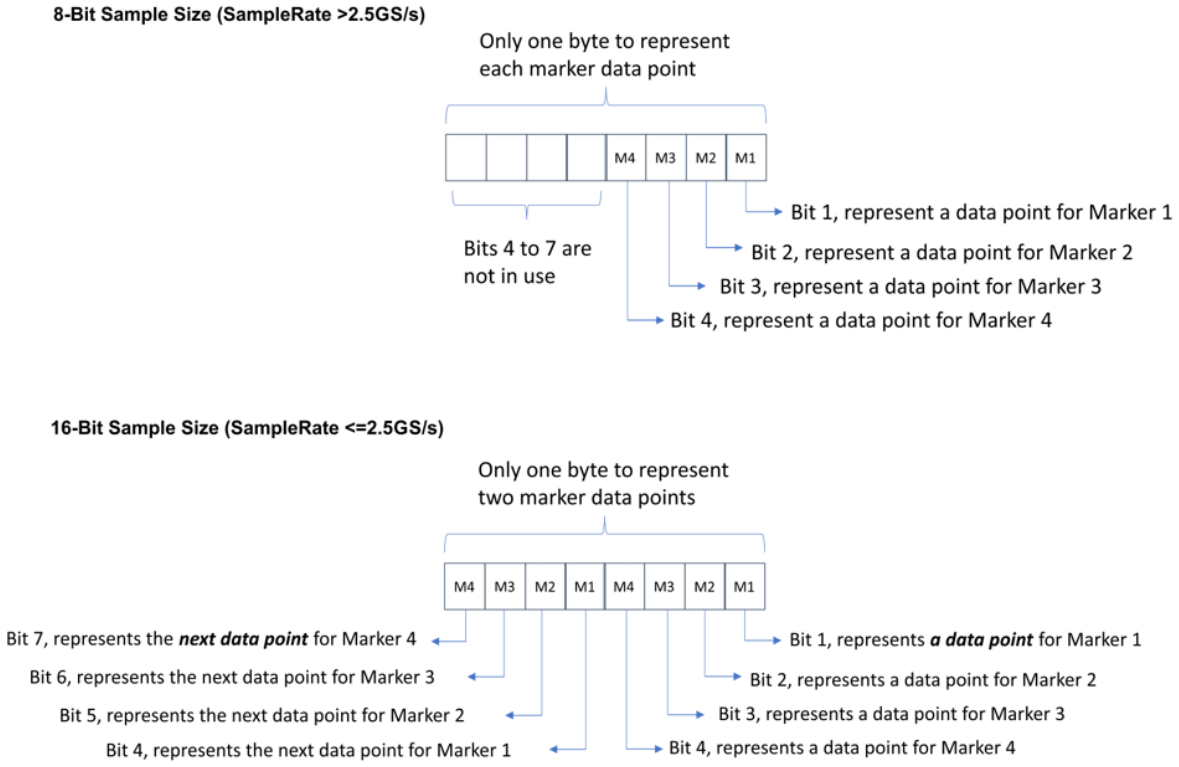

Figure 6-1 Marker Format

6.1 :MARKer:SELect{1|2|3|4}(?)

Description

This command will select a given marker of the currently selected channel for programming. Markers (numbered 1 to 4 in the Proteus unit front panel) are associated to specific arbitrary waveform generation channels. Depending on the Proteus version and the total number of channels and marker outputs, there may be a total of up to two (2) or four (4) markers per channel.

Parameters

Range	Type	Default	Description
1 to 4	Discrete	1	Select a specific marker output

Response

The Proteus will return the selected marker.

Example

Command `:MARK:SEL 3`

Query `:MARK:SEL?`

6.2 :MARKer[:STATe]{OFF|ON|0|1}(?)

Description

This command will set or query the state of the marker outputs for the current active marker. Markers (numbered 1 to 4 on the Proteus unit front panel) are associated to specific arbitrary

waveform generation channel. Depending on the Proteus version and the total number of channels and marker outputs, there may be a total of up to two (2) or four (4) markers per channel. Note that for safety, the outputs always default to off, even if the last instrument setting before power down was on. The on/off setting affects both markers simultaneously on each channel.

Parameters

Range	Type	Default	Description
0-1	Discrete	0	Sets the marker output on and off.

Response

The Proteus will return 1 if the marker output is ON, or 0 if the marker output is OFF.

Example

Command **:MARK ON**

Query **:MARK?**

6.3 :MARKer:DELay:COARse <delay>(?)

Description

Use this command to set or query the coarse delay of the marker output. The delay is measured from the sync output in units of samples. The marker has an initial delay of 0 sample clock periods, not including initial skew.

Parameters

Name	Range	Type	Default	Description
< delay >	16-bit DAC mode (M0): -255 – +254 8-bit DAC mode (M1): -1024 - +1016	Numeric	0	Will set marker coarse delay value in units of samples. Each channel has two/four separate markers that can be programmed to have unique delays and amplitude levels. The resolution is 2 points in case of 8-bit DAC mode, and 8 points in case of 16-bit DAC mode.

Response

The Proteus will return the present coarse marker delay value. The returned value will be in unsigned integer format.

Example

Command **:MARK:DEL:COAR 232**

Query **:MARK:DEL:COAR?**

6.4 :MARKer:DELay:FINE<delay>(?)

Description

Use this command to set or query the delay of the marker output. The delay is measured from the sync output in units of seconds. The marker has an initial delay of 0 seconds (after factory calibration).

Parameters

Name	Range	Type	Default	Description
< delay >	-0.6e-9 to 0.6e09	Numeric	0	Will set marker delay value in units of seconds. Each channel has two/four separate markers that can be programmed to have unique delays and amplitude levels.

Response

The Proteus will return the present marker delay value. The returned value will be in standard scientific format (for example: 1 ns would be returned as 1e-9 – positive numbers are unsigned).

Example

Command `:MARK:DEL:FINE 1.0e-9`

Query `:MARK:DEL:FINE?`

6.5 :MARKer:VOLTage:LEVel <gain>(?)

Description

Use this command to set or query the marker gain. The level is defined in dB.

Parameters

Name	Range	Type	Default	Description
< gain >	0.0 to 32.0	Numeric	0.0	Will program the marker gain in dB. Each marker can be programmed to a different gain setting.

Response

The Proteus will return the present marker gain in dB. The returned value will be in standard scientific format (for example: 1.0 dB would be returned as 1e0 – positive numbers are unsigned).

Example

Command `:MARK:VOLT:LEV 1.0`

Query `:MARK:VOLT:LEV?`

6.6 :MARKer:VOLTage:PTOP<ptop_level>(?)

Description

Use this command to set or query the peak-to-peak level of the marker output. The level is defined in unit of volt.

Parameters

Name	Range	Type	Default	Description
< ptop_level >	0.05 to 1.2	Numeric	0	Will program the marker peak-to-peak level in units of volts. Each marker can be programmed to a different peak-to-peak level setting. The peak-to-peak level is calibrated when the cable is terminated into a 50Ω load.

Response

The Proteus will return the present marker peak-to-peak level value. The returned value will be in standard scientific format (for example: 1.0 V would be returned as 1e0 – positive numbers are unsigned).

Example

Command `:MARK:VOLT:PTOP 1.0`

Query `:MARK:VOLT:PTOP?`

6.7 :MARKer:VOLTage:OFFSet<offset>(?)

Description

Use this command to set or query the offset level of the marker output. The offset level is defined in units of volts.

Parameters

Name	Range	Type	Default	Description
< offset >	-0.5 to +0.5	Numeric	0	Will program the offset level in units of volts. Each marker can be programmed to a different low- and high-level setting. The offset level is calibrated when the cable is terminated into a 50 Ω load.

Response

The Proteus will return the present marker offset value. The returned value will be in standard scientific format (for example: 0.1 V would be returned as 100e-3 – positive numbers are unsigned).

Example

Command `:MARK:VOLT:OFFS 0.5`

Query `:MARK:VOLT:OFFS?`

6.8 :MARKer:DATA [<offset>]#<header><binary_block>(?)

Description

This command will download marker data to the Proteus unit sequence memory for the active segment and channel. Marker data is loaded to the Proteus unit using high-speed binary data transfer. High-speed binary data transfer allows any number of 8-bit bytes to be transmitted in a message. Refer to the Proteus user manual chapter Markers for a detailed description.

The next command will download to the generator a block of marker data of 1,024 entries:

:MARK:DATA #41024<binary_block>

This command causes the transfer of 1,024 bytes of data (1,024 marker states) into the active memory segment. The <header> is interpreted this way:

- The ASCII "#" ($23) designates the start of the binary data block.
- "4" designates the number of digits that follow representing the binary data block size in bytes.

- "1,024" is the number of bytes to follow.
- <binary_block> Represents task-related data.

Response

The Proteus will return the #<header><binary-block>\n.

Parameters

Name	Range	Type	Default	Description
< offset >	0, <max_states>	Integer	0	Will set the initial marker state number for the download of marker data.
< header >		Integer		Contains information on the size of the binary block that follows.
< binary_block >		Binary		Block of binary data that contains task-related data, as explained above.

Example

Command **:MARK:DATA #42048<binary_block>**

Query **:MARK:DATA? 64,#42048<binary_block>**

6.9 :MARKer:MEMory <offset_in_bytes>,#<header><marker-data>(?)

Description

Direct download to the arbitrary memory without any segment attributes. This command (or query) is the same as :MARK[:DATA] [<offset-in-bytes-of-wave-data>] #<binary-header><binary-data> except that the offset, in case of :MARKer:MEMory, is from the beginning of the memory-space rather than the beginning of memory of the selected segment.

Parameters

Name	Range	Type	Default	Description
<offset_in_bytes>	0, <max_states>	Integer	0	Will set the initial marker state number for the download of marker data of the selected segment.
<header>		Integer		Contains information on the size of the binary block that follows.
<marker-data>		Binary		Block of binary data that contains task-related data, as explained above.

Response

The Proteus will return the #<header><binary-block>\n.

Note: A comma is required to separate between the offset and the size.

Example

Command **:MARK:MEM 16#42048<binary_block>**

Query **:MARK:MEM? 16,#42048<binary_block>**

6.10 :MARKer:FILE[:NAMe]{<#<header><binary_block>}

Description

This command will set-up the marker information as the :MAR:DATA command does but reading the contents from a file stored in the target standalone Proteus. The file name is defined as an IEEE-488.2 binary block with the name codified in 8-bit unsigned integers (bytes) with the ASCII codes containing the full path to the source file.

Parameters

Name	Type	Description
<header>	Discrete	The first digit in ASCII following the '#' character is the number of digits to follow. The following digits specify the length of the target file full path name in ASCII.
<binary_block>	String	Full path name for the file in ASCII coded as an unsigned short integer array.

Example

Command `:MARK:FILE #210marker.dat`

6.11 :MARKer:FILE:OFFSet< start-offset inside the file>(?)

Description

This command will set the start offset in the file in bytes for the load or store command.

Parameters

Name	Range	Type	Default	Description
< start-offset inside the file>		Numeric (int(	0	Bytes

Response

The Proteus will return the start offset in bytes.

Note: A comma is required to separate between the offset and the size.

Example

Command `:MARK:FILE:OFFS 256`

Query `:MARK:FILE:OFFS?`

6.12 :MARKer:FILE:DESTination < SEGMent | MEMory>(?)

Description

This command specifies the destination to load/store the file data.

Parameters

Name	Range	Type	Default	Description
SEGMent		Discrete	SEGM	The selected segment.

Name	Range	Type	Default	Description
MEMory		Discrete		The arbitrary-memory space.

Response

The Proteus will return the selected file destination

Example

Command `:MARK:FILE:DEST SEGM`

Query `:MARK:FILE:DEST?`

6.13 :MARKer:FILE:LOAD [[<offset>,]<size>]

Description

Load block of markers-data from the binary-file specified in :MARKer:FILE:NAME to the hardware-memory specified in :MARKer:FILE:DESTination. The starting offset in the file is specified in :MARKer:FILE:OFFSet, while the block-size and the write-offset in the destination hardware memory are specified by the (optional) arguments of the command. If the <offset> argument is missing then zero <offset> is assumed. If both the <offset> argument and the <size> argument are missing, then all data from the start-offset in the file to the end of the file is written.

Parameters

Name	Range	Type	Default	Description
< offset >		Numeric(int64)	0	Will set the start offset in the selected memory space in bytes of markers-data.
< size>		Numeric(int64)		The size of the binary block in bytes of markers-data.

Example

Command `:MARK:FILE:LOAD 64,42048`

6.14 :MARKer:FILE:STORe [[<offset>,]<size>]

Description

Store block of markers-data from the hardware memory specified in :MARKer:FILE:DESTination to the binary-file specified in :MARKer:FILE:NAME. The block-size and the write-offset in the hardware memory are specified by the (optional) arguments of the command. If the <offset> argument is missing then zero <offset> is assumed. If both the <offset> argument and the <size> argument are missing, then all data from the start-offset in the segment to the end of the segment is written.

Parameters

Name	Range	Type	Default	Description
< offset>		Numeric(int64)	0	Will set the start offset in the selected memory space in bytes of markers-data
< size>		Numeric(int64)		The size of the binary block in bytes of marker data.

Example

Command :**MARK:FILE:STOR 64,42048**

7 Task Commands

The task commands allows for the creation, edition and removal of a table of tasks. The task table can contain up to 64K entries. Each entry holds all the data for a given task. Tasks point always to a given segment # and may be part of a sequence of tasks.

7.1 :TASK:COMPoser:LENGth<length>(?)

Description

Use this command to define the length of the task table.

Parameters

Name	Range	Type	Default	Description
< length >	0 to 64 k	Numeric(int)	1	Allocate array of task table-rows for the task-table composer

Response

The Proteus unit will return the size of the task table.

Example

Command `:TASK:COMP:LENG 100`

Query `:TASK:COMP:LENG?`

7.2 :TASK:COMPoser:SELect<task_#>(?)

Description

Use this command to select or query the task number to be defined.

Parameters

Name	Range	Type	Default	Description
< task_#>	1 to 64k	Numeric(int)	1	Define the task # to be set up.

Response

The Proteus unit will return currently selected task number.

Example

Command `:TASK:COMP:SEL 10`

Query `:TASK:COMP:SEL?`

7.3 :TASK:COMPoser[:DEFine]:TYPE{SINGle|STARt|END|SEQ}(?)

Description

Use this command to define the task type the current entry in the task table. It is possible to define different sequences within the task table. A sequence is comprised of a start task, optional

intermediate tasks, and an end task. Once a number of tasks are defined as a sequence it is possible to program the number of times the sequence will be repeated. The task # must be selected before using this command or query through the :TASK:SEL command.

Parameters

Name	Type	Default	Description
<SINGle>	Discrete	SING	Single step task.
<STARt>	Discrete		Initial task in a sequence.
<END>	Discrete		Final task in a sequence.
<SEQ>	Discrete		Intermediate task in a sequence.

Response

The Proteus unit will return SING, STAR, END, or SEQ depending on the type of the currently selected task.

Example

Command **:TASK:COMP:TYPE STAR**

Query **:TASK:COMP:TYPE?**

7.4 :TASK:COMPoser[:DEFine]:LOOPs<task_loops>(?)

Description

Use this command to define the number of loops for the current entry in the task table. The task # must be selected before using this command or query through the :TASK:SEL command.

A task table is made up of several tasks (lines in the task table). There can be up to 64K tasks, or lines in the task table. Each task defines which segment is generated. The task loops parameter defines how many times the current task is repeated. The KEEP parameter (refer to :TASK:COMPoser[:DEFine]:KEEP{OFF|ON|0|1}(?)) is with respect to the trigger. So if the number of Task loops is N, and the enabling signal is TRIG1. There are 2 options of how this task plays out when a trigger is initiated to TRIG1:

1. KEEP=0 - A single trigger is received, and the task is played N times.
2. KEEP=1 – A trigger is received, and the task is played once. After N triggers the task is completed and it proceeds to the next task (line) in the task table.

Parameters

Name	Range	Type	Default	Description
<task_loops>	0 - 1M	Numeric (int)	1	Number of loops for the task. 0 = Infinite

Response

The Proteus unit will return the number of loops for the current task.

Example

Command **:TASK:COMP:LOOP 1234**

Query **:TASK:COMP:LOOP?**

7.5 :TASK:COMPoser[:DEFine]:SEQuence<seq_loops>(?)

Description

Use this command to define the number of loops for the current sequence. The task # for the START task must be selected before using this command or query through the :TASK:SEL command.

Parameters

Name	Range	Type	Default	Description
<seq_loops>	0 - 1M	Numeric (int)	1	Number of loops for the sequence. 0 = Infinite.

Response

The Proteus unit will return the number of loops for the current sequence.

Example

Command `:TASK:COMP:SEQ 123`

Query `:TASK:COMP:SEQ?`

7.6 :TASK:COMPoser[:DEFine]:SEGMent(?)

Description

Use this command to define the segment attached to the current entry in the task table. The task # must be selected before using this command or query through the :TASK:SEL command. The same segment may be used by any number of tasks.

Parameters

Name	Range	Type	Default	Description
	1 - 64k	Numeric (int)	1	Segment # associated to this task.

Response

The Proteus unit will return the segment number for the current task.

Example

Command `:TASK:COMP:SEGM 62345`

Query `:TASK:COMP:SEGM?`

7.7 :TASK:COMPoser[:DEFine]:IDLE[:TYPE] {DC|FIRSt|CURRent}(?)

Description

Use this command to define the behavior of the current task while in the idle state. The task must be selected before using this command or query through the :TASK:SEL command.

Parameters

Name	Type	Default	Description
<DC>	Discrete	DC	DC level

Name	Type	Default	Description
<FIRST>	Discrete		First level in the segment associated to the task.
<CURRENT>	Discrete		Continuous loop of the current segment.

Response

The Proteus will return FIRS, DC or CURRENT depending on the current idle mode setting

Example

Command `:TASK:COMP:IDLE FIRS`

Query `:TASK:COMP:IDLE?`

7.8 :TASK:COMPoser[:DEFine]:IDLE:LEVel {<DC_level>}(?)

Description

Use this command to define the DC level while in the idle state for the current task when the idle type has been set to DC. The task must be selected before using this command or query through the :TASK:SEL command.

Parameters

Name	Range	Type	Default	Description
<dc_level>	0 to 255 or 65,535	Numeric	128/32,768	Programs the DC level during the idle state in quantization levels. 0-255 for the 8-bit DAC (M1) mode and 0-65535 for the 16-bit DAC (M0) mode.

Response

The Proteus unit will return the level for the idle state for the current task.

Example

Command `:TASK:COMP:IDLE:LEV 24567`

Query `:TASK:COMP:IDLE:LEV?`

7.9 :TASK:COMPoser[:DEFine]:ENABle{NONE|TRG1|TRG2|TRG3|TRG4|TRG5|TRG6|INTernal|CPU|FBTRg|ANY}(?)

Description

Use this command to define enabling signal for the current entry in the task table. The task # must be selected before using this command or query through the :TASK:SEL.

Parameters

Name	Type	Default	Description
<NONE>	Discrete	NONE	No enabling signal required.
<TRG1>	Discrete		Trigger input 1

Name	Type	Default	Description
<TRG2>	Discrete		Trigger input 2
<TRG3>	Discrete		Trigger input 3
<TRG4>	Discrete		Trigger input 4
<TRG5>	Discrete		Trigger input 5
<TRG6>	Discrete		Trigger input 6
<INTernal>	Discrete		Enabling signal generated internally.
<CPU>	Discrete		Enabling signal through SCPI command.
<FBTRg>	Discrete		Enabling signal generated by the digitizer block
<ANY>	Discrete		Any of the above.

Effected Channels

Trigger	Model	Effected Channels
TRG1	All	All channels
TRG2	All	All channels
TRG3	P1288D	CH5 -CH8
	P12812D	
	P2588D	
	P25812D	
	P1288B	
	P12812B	
	P2588B	
	P25812B	
	P9084D	CH3 - CH4
	P9086D	
	P9084B	
	P9086B	
TRG4	P1288D	CH5 -CH8
	P12812D	
	P2588D	
	P25812D	
	P1288B	
	P12812B	
	P2588B	
	P25812B	
	P9084D	CH3 - CH6
	P9086D	

	P9084B	
	P9086B	
TRG5	P12812D	CH9-CH12
	P25812D	
	P12812B	
	P25812B	
	P9086D	CH5 -CH6
	P9086B	
TRG6	P12812D	CH9-CH12
	P25812D	
	P12812B	
	P25812B	
	P9086D	CH5 -CH6
	P9086B	

Response

The Proteus unit will return the enabling signal source currently selected for the current task.

Example

Command `:TASK:COMP:ENAB FBTR`

Query `:TASK:COMP:ENAB?`

7.10 :TASK:COMPoser[:DEFine]:ABORt{ NONE|TRG1|TRG2|TRG3|TRG4|TRG5|TRG6| INTernal|CPU|FBTRg|ANY }(?)

Description

Use this command to define the abort signal for the current entry in the task table. The task # must be selected before using this command or query through the :TASK:SEL command.

Parameters

Name	Type	Default	Description
<NONE>	Discrete	NONE	No abort signal required.
<TRG1>	Discrete		Trigger input 1
<TRG2>	Discrete		Trigger input 2
<TRG3>	Discrete		Trigger input 3
<TRG4>	Discrete		Trigger input 4
<TRG5>	Discrete		Trigger input 5
<TRG6>	Discrete		Trigger input 6
<INTernal>	Discrete		Abort signal generated internally.

Name	Type	Default	Description
<CPU>	Discrete		Abort signal through SCPI command.
<FBTRg>	Discrete		Abort signal generated by the digitizer block.
<ANY>	Discrete		Any of the above.

Effected Channels

Trigger	Model	Effected Channels
TRG1	All	All channels
TRG2	All	All channels
TRG3	P1288D	CH5 -CH8
	P12812D	
	P2588D	
	P25812D	
	P1288B	
	P12812B	
	P2588B	
	P25812B	
	P9084D	CH3 - CH4
	P9086D	
	P9084B	
	P9086B	
TRG4	P1288D	CH5 -CH8
	P12812D	
	P2588D	
	P25812D	
	P1288B	
	P12812B	
	P2588B	
	P25812B	
	P9084D	CH3 - CH6
	P9086D	
	P9084B	
	P9086B	
TRG5	P12812D	CH9-CH12
	P25812D	
	P12812B	

	P25812B	
	P9086D	CH5 -CH6
	P9086B	
TRG6	P12812D	CH9-CH12
	P25812D	
	P12812B	
	P25812B	
	P9086D	CH5 -CH6
	P9086B	

Response

The Proteus unit will return the enabling signal source currently selected for the current task.

Example

Command `:TASK:COMP:ABOR FBTR`

Query `:TASK:COMP:ABOR?`

7.11 :TASK:COMPoser[:DEFine]:JUMP{EVENtually|IMMediate}(?)

Description

Use this command to define the way to jump to a different task for the currently selected task when a valid ABORT event occurs. The task # must be selected before using this command or query through the :TASK:SEL command.

Parameters

Name	Type	Default	Description
< EVENtually >	Discrete		Sets the effective jump to happen at the end of the current loop.
< IMMediate >	Discrete	IMM	Results in jumping without waiting for the end of the loop.

Response

The Proteus unit will return the jumping mode for the current task.

Example

Command `:TASK:COMP:JUMP IMM`

Query `:TASK:COMP:JUMP?`

7.12 :TASK:COMPoser[:DEFine]:DESTination{NEXT | FBTRg | TRG | NTSel | SCENario | DSP |DSIG}(?)

Description

Use this command to define the next task to be generated after the currently selected task. The task # must be selected before using this command or query through the :TASK:SEL command.

Parameters

Name	Type	Default	Description
<NEXT>	Discrete	NEXT	Points to the next task to be generated according to the :NEXT1 command setting
<FBTRg>	Discrete		Points to the next task to be generated according to the digitizer setting.
<TRG>	Discrete		A conditional jump. Points to the next task to be generated according to the trigger inputs. Valid signal at trigger 1 points to the next task as set in :NEXT1 setting while a valid signal at trigger 2 points to next task as set in :NEXT2 setting.
<NTSel>	Discrete		The next task in the table
<SCENario>	Discrete		The beginning of next scenario
<DSP>	Discrete		Destination is NEXT1 current segment to be generated is according to decision block condition in DSP.
<DSIG>	Discrete		NEXT1 if digitizer-signal = 1, NEXT2 if digitizer-signal = 0.

Response

The Proteus unit will return the next task number mode for the current task.

Example

Command `:TASK:COMP:DEST TRG`

Query `:TASK:COMP:DEST?`

7.13 :TASK:COMPoser[:DEFine]:NEXT1 <next_task>(?)

Description

Use this command to define the next task to be generated after the currently selected task. The task # must be selected before using this command or query through the :TASK:SEL command.

Parameters

Name	Range	Type	Default	Description
<next_task>	1 - 64k	Numeric (int)	1	Next task # to be generated after the current task is executed. “0” means end.

Response

The Proteus unit will return the next task to be generated after the current task is executed.

Example

Command **:TASK:COMP:NEXT1 1456**

Query **:TASK:COMP:NEXT1?**

7.14 :TASK:COMPoser[:DEFine]:NEXT2 <next_task>(?)

Description

When setting a conditional jump use this command to define the next task to be generated after the currently selected task when the Trigger 2 input or digitizer-signal=0 are the source for jumping. The task # must be selected before using this command or query through the :TASK:SEL command.

Parameters

Name	Range	Type	Default	Description
<next_task>	1 - 64k	Numeric (int)	1	Next task # to be generated when trigger input 2 or digitizer-signal=0 are the source for conditional jump. “0” means end.

Response

The Proteus unit will return the next task number mode for the current task when trigger 2 or digitizer-signal=0 are the source for jumping.

Example

Command **:TASK:COMP:NEXT2 1456**

Query **:TASK:COMP:NEXT2?**

7.15 :TASK:COMPoser[:DEFine]:DELay<task_delay>(?)

Description

Use this command to define the delay in clocks before executing the next task. The task # must be selected before using this command or query through the :TASK:SEL command.

Parameters

Name	Range	Type	Default	Description
< task_delay>	0-65536	Numeric (int)	0	Delay in SCLK cycles.

Response

The Proteus unit will return the delay for the next task in SCLK cycles.

Example

Command **:TASK:COMP:DEL 1000**

Query **:TASK:COMP:DEL?**

7.16 :TASK:COMPoser[:DEFine]:KEEP{OFF|ON|0|1}(?)

Description

Use this command to define the behavior of loops for this task with respect to the trigger. The task # must be selected before using this command or query through the :TASK:SEL command.

Parameters

Range	Type	Default	Description
0-1	Numeric (int)	0	**0** – The task loops are executed consecutively when enabled. **1** – The task is executed once per valid enabling signal until all task loops have been executed.

Response

The Proteus will return 1 if the keep state is set to ON, or 0 if the keep state is set to OFF.

Example

Command `:TASK:COMP:KEEP ON`

Query `:TASK:COMP:KEEP?`

7.17 :TASK:COMPoser[:DEFine]:DTRigger{OFF|ON|0|1}(?)

Description

Use this command to generate the digitizer trigger. In order to this event being effective, the digitizer trigger source must be set with the :DIG:TRIG:SOUR TASKx command, where x is the channel number of the AWG where the relevant task list is being generated (1-4).

Parameters

Range	Type	Default	Description
0-1	Numeric (int)	0	When set to 1 the digitizer trigger is enabled.

Response

The Proteus will return 1 if the digitizer trigger is set to ON, or 0 if the digitizer trigger is set to OFF.

Example

Command `:TASK:COMP:DTR ON`

Query `:TASK:COMP:DTR?`

7.18 :TASK:COMPoser:WRITE<offset in task table rows>

Description

Write the composer's array to the task-table of the selected channel at the specified offset (no query). Issue this command once all the Task table parameters have been defined.

Parameters

Name	Range	Type	Default	Description
< offset >	1 to 64k	Numeric(int)	1	The offset in the task table.

Example

Command `:TASK:COMP:WRIT 100`

7.19 :TASK:COMPoser:READ<offset in task table rows>

Description

Read the composer's array from the task-table of the selected channel at the specified offset (no query). See :TASK:DATA command for data format definitions.

Parameters

Name	Range	Type	Default	Description
< offset >	0 to 64 k	Numeric(int)	1	The offset in the task table.

Example

Command `:TASK:COMP:READ 100`

7.20 :TASK:CURRent?

Description

Query only. Returns the current task number.

Response

The Proteus unit will return the maximum value for this control value in ASCII format.

Example

Query `:TASK:CURR?`

7.21 :TASK: SYNC

Description

No query. Issue this command to synchronize the task tables of all channels. This command needs to be issued every time before generation is started.

Example

Command :TASK: SYNC

7.22 :TASK:DATA [<offset>]#<header><binary_block>

Description

Write data to the specified offset in the task-table of the selected channel. Binary transfers are a much faster way to define tasks lists, especially when they are long. Binary data is defined as an array of structs (see format below) of fixed length, with one element representing each individual task in the list. This format is also used when reading data from the task list using the

:TASK:COMP:READ command and when transferring task lists from/to files using the :TASK:FILE:LOAD and :TASK:FILE:STOR commands.

As an example, the :TASK:DATA #41024<binary_block> command will cause the transfer of 1,024 bytes of data (1,024 marker states) into the active memory segment. The <header> is interpreted this way:

- The ASCII "#" ($23) designates the start of the binary data block.
- "4" designates the number of digits that follow representing the binary data block size in bytes.
- "1,024" is the number of bytes to follow.
- <binary_block> Represents task-related data.

Parameters

Name	Range	Type	Default	Description
< offset >	0 64k	Numeric(int)	0	Write the data to specified offset in the task table.
< header >		Numeric(int)		Contains information on the size of the binary block that follows.
< binary_block >		Binary		Block of binary data that contains task-related data, as explained above.
		Type	**Bytes**	
		UINT32	0 - 3	The segment number.
		UINT32	4 - 7	The next task for trigger 1 (zero for end)
		UINT32	8 - 11	The next task for trigger 2 (zero for end).
		UINT32	12 - 15	The task loop count.
		UINT32	16 - 19	The sequence loop count.
		UINT16	20 - 21	The delay in clocks before executing the next task.
		UINT16	22 - 23	The DAC value of the idle task DC waveform.
		UINT8	24	The behavior during idle-time. 0 – DC 1 – First point 2 – Current segment
		UINT8	25	The enabling signal. 0 – None 1 – ExternTrig1 2 – ExternTrig2 3 – InternTrig 4 – CPU 5 – FeedbackTrig 6 – HW-Ctrl
		UINT8	26	The aborting signal. 0 – None 1 – ExterTrig1

			2 – ExternTrig2 3 – InternTrig 4 – CPU 5 – FeedbackTrig 6 – Any
	UINT8	27	How to decide where to jump. 0 – Next1 1 – By FBTrig-value 2 – ExtTrig[1/2]->Next[1/2] 3 – NextTaskSel 4 – Next scenario
	UINT8	28	Task abort jump type. 0 – Eventually 1 – Immediate
	UINT8	29	The task state. 0 – Single 1 – First of sequence 2 – Last of sequence 3 – Inside sequence
	UINT8	30	Task loop trigger enable, waiting for trigger on looping. 1 – Enable 0 – Disable
	UINT8	31	Generate an ADC trigger at the beginning of the current task. 1 – Enable 0 – Disable

Example

Command `:TASK:DATA #42048<binary_block>`

7.23 :TASK:FILE[:NAME] {#<header><binary_block>}

Description

This command will identify the file path storing the task table information for further transfers to/from the task table. The file path is passed as a binary-block.

Parameters

Name	Type	Description
<header>	Discrete	The first digit in ASCII is the number of digits to follow. The following digits specify the length of the target file full path name in ASCII.
<binary_block>	String	Full path name for the file in ASCII coded as an unsigned short integer array.

Example

Command **:TASK:FILE #210task_1.dat**

7.24 :TASK:FILE:OFFSet <start-offset>

This command will set the start offset inside the file in bytes.

Parameters

Name	Type	Description
<start-offset>	Integer	The start offset inside the file in bytes.

Example

Command **:TASK:FILE:OFFS 64**

7.25 :TASK:FILE:LOAD[<offset>,<num_of_tasks>]

Description

This command will load task data from the file defined by the :TASK:FILE:NAME command to the Proteus desktop unit task table memory. If the offset and number of tasks are not specified, then the whole task-table is written. If the file is too small then the rest of the task-table rows are zeroed. See :TASK:DATA command for data format definitions.

Parameters

Name	Type	Description
<offset>	Integer	Offset in task table rows
< num_of_tasks >	Integer	Number of task table rows

Example

Command **:TASK:FILE:LOAD 12,128**

7.26 :TASK:FILE:STORe[<offset>,<num_of_tasks>]

Description

This command will save task data from the Proteus unit to the file defined by the :TASK:FILE:NAME command. The command, when no parameters are specified, saves all the entries in the task table in the file (no query). See :TASK:DATA command for data format definitions.

Parameters

Name	Type	Description
<offset>	Integer	Offset in task table rows
< num_of_tasks >	Integer	Number of task table rows

Example

Command **:TASK:FILE:STOR 1,1280**

7.27 :TASK:ZERO[:PORTion] <offset>,<num_of_tasks>

Description

This command will set the designated entries in the task to an “all zeros” content. Issue this command e.g., when you have a task table of 8 tasks and want to write a task table of 4 tasks.

Parameters

Name	Type	Description
<offset>	Integer	Offset in task table rows
<num_of_tasks>	Integer	Number of task table rows

Example

Command `:TASK:ZERO 100,250`

7.28 :TASK:ZERO:ALL

Description

This command will set the all the entries in the task to an “all zeros” content.

Example

Command `:TASK:ZERO:ALL`

8 Scenario Commands

Note
Scenario commands are planned for a future release.

The scenario commands allows for the creation, edition and removal of scenario tables. The scenario table can contain up to 1000 entries. Each entry holds a pointer to a task number in the task table.

8.1 :SCENario:DEFine { <scenario-number>, <task-number>, <loops>}(?)

Description

Use this command to define the specified entry in the scenario-table of the selected channel.

In principal it is a table consisting of Task numbers. So, if a task is a “playlist of songs” then a scenario is a “playlist of playlists”.

Parameters

Name	Range	Type	Default	Description
<scenario-number>	1 – 1000	Numeric (int)		Define the scenario number to be set up.
<task-number>	1 – 64k	Numeric (int)		Define the task to be executed.
<loops>	1 – 1M	Numeric (int)		Set the number of times that segment is to be repeated.

Response

The Proteus unit will return currently selected scenario number, task number and loops.

Example

Command `:SCEN:DEF 10, 150, 2000`

Query `:SCEN:DEF?`

8.2 :SCENario:DATA {[<offset>,]#<header><binary_block>}

Description

This command will download scenario data to the Proteus unit sequence memory. Scenario data is loaded to the Proteus unit using high-speed binary data transfer. High-speed binary data transfer allows any number of 8-bit bytes to be transmitted in a message. This command is particularly useful for sending large quantities of data. As an example, the next command will download to the generator a block of scenario related data of 512 entries:

:SCEN:DATA #3512<binary_block>

This command causes the transfer of 512 bytes of data (256 waveform points) into the active scenario. The <header> is interpreted this way:

- The ASCII "#" (0x23) designates the start of the binary data block.
- "4" designates the number of digits that follow representing the binary data block size in bytes.
- "512" is the number of bytes to follow.
- <binary_block> Represents task-related data.

Parameters

Name	Type	Description
[<offset>]	Integer	Offset in scenario table rows.
< header >	Discrete	Contains information on the size of the binary block that follows.
< binary_block >	Binary	Block of binary data that contains scenario-related data, as explained above.

Example

Command **:SCEN:DATA #3512<binary_block>**

8.3 :SCENario:FILE[:NAME]{ #<header><binary_block>}

Description

This command will identify the file path storing the scenario table information for further transfers to/from the scenario table. The file path is passed as a binary block.

Parameters

Name	Type	Description
< header >	Discrete	The first digit in ASCII is the number of digits to follow. The following digits specify the length of the target file full path name in ASCII.
< binary_block >	Binary	Full path name for the file in ASCII coded as an unsigned short integer array.

Example

Command **:SCEN:FILE #238"C:\\scenario_data\scenario_file_03.bin"**

8.4 :SCENario:FILE:OFFSet {<offset>}

Description

This command will set the start offset inside the file in bytes.

Parameters

Name	Type	Description
<offset>	Integer	Set the start offset inside the file in bytes.

Example

Command **:SCEN:FILE:OFFS 12**

8.5 :SCENario:FILE:LOAD {[<offset>,<num_of_scenarios>]}

Description

This command will load the task data from the file defined by the :SCEN:FILE:NAME command to the Proteus unit task table memory. If the offset and number of tasks are not specified, then the whole task-table is written. If the file is too small then the rest of the task-table rows are zeroed.

Parameters

Name	Type	Description
<offset>	Integer	Offset in scenario table rows
<num_of_scenarios>	Integer	Number of scenario table rows

Example

Command `:SCEN:FILE:LOAD 8,64`

8.6 :SCENario:FILE:STORe {[<offset>,<num_of_scenarios>]}

Description

This command will save scenario data from the Proteus unit to the file defined by the :SCEN:FILE:NAME command. The command, when no parameters are specified, saves all the entries in the task table in the file (no query).

Parameters

Name	Type	Description
<offset>	Integer	Offset in scenario table rows
< num_of_scenarios >	Integer	Number of scenario table rows

Example

Command `:SCEN:FILE:STOR`

8.7 :SCENario:ZERO[:SINGLe] <scenario-number>

Description

Reset the data of a single row in the scenario table of the selected channel (no query).

Parameters

Name	Type	Description
< scenario-number >	Integer	Scenario number in the file to be processed.

Example

Command `:SCEN:ZERO 100`

8.8 :SCENario:ZERO:ALL

Description

This command will set the all the entries in the scenario table to an “all zeros” content.

Example

Command **:SCEN:ZERO:ALL**

9 Arbitrary Waveform Commands

This group is used to manage the arbitrary waveforms and their respective parameters. This will allow you to create segments and download waveforms. Using these commands, you can also define segment size and delete some or all unwanted waveforms from your memory.

Generating Arbitrary Waveforms

Arbitrary waveforms are generated from digital data points, which are stored in a dedicated waveform memory. Each data point has a vertical resolution of 16/8 bits depending on the device DAC mode (65,536/256 levels), i.e., each sample is placed on the vertical axis with a resolution of 1/65,536 or 1/256. The Proteus unit supports two DAC modes. In the first mode, all 16 vertical resolutions bits are used. In the second mode, only 8 bits (the MSBs) are fed to the Digital-to-Analog Converter. Proteus memory capacity can be, depending on the model and memory options, of 1 GS, 2 GS, 4 GS, 8 GS, and 16 GS.

Each horizontal point has a unique address; the first being 00000 and the last depending on the memory option. In cases where smaller waveform lengths are required, the waveform memory can be divided into smaller segments.

When the instrument is programmed to output arbitrary waveforms, the clock samples the data points (one at a time) from address 0 to the last address. The rate at which each sample is played back is defined by the sample clock rate parameter.

Arbitrary waveforms must first be loaded into the instrument's memory. Correct memory management is required for best utilization of the arbitrary memory. An explanation of how to manage the arbitrary waveform memory is given in the following paragraphs.

Arbitrary Memory Management

The arbitrary memory in comprised of a finite length of words. The maximum size arbitrary waveform that can be loaded into memory depends on the option that is installed in your instrument.

Waveforms are created using small sections of the arbitrary memory. The memory can be partitioned into smaller segments (up to 64K) identified by a number and different waveforms can be loaded into each segment, each having a unique length. Minimum segment size is 1024 (2.5 GSa/s) or 2048 points (9 GSa/s) and can be increased by increments of 32 points (2.5 GSa/s) or 64 points (9GSa/s). Information on how to partition the memory, define segment length and download waveform data to the Proteus unit is given in the following paragraphs. The arbitrary waveform commands are listed in Table x-y. Factory defaults after *RST are shown in the Default column. Parameter range and low and high limits are listed, where applicable.

Arbitrary Waveform Memory is arranged in one single bank for P1282, P2582 and P9482 and in two equal size banks for P1284, P2584, P9494, and P9082. Bank #1 is shared by CH1 and CH2 in all the Proteus models except for the P9082, where it is attached to CH1. Bank #2 is shared by CH3 and CH4 in all the 4-channel Proteus models except for the P9082, where it is attached to CH2. Segments defined when some channel is selected will be also available to the other channel sharing the same memory bank. If both channels must use the same waveform, there is no need to define two segments containing the same waveform and both can be associated to the same segment number and they can access waveform data simultaneously and asynchronously without any limitation.

Short and Fast Segments

There is an especial type of segment (“short segments” and “fast segments”) that are stored totally (short) or partially (fast) in the FPGA internal memory which are designed for fast switching and sequencing. These segments can be much shorter than the ones stored in the regular waveform memory. For the 2.5 GSa/s (or less) units, the minimum length of fast segments is 32 points while for the 9.0 GSa/s units, minimum length is 128 points. Short Segments and Fast segments do not need to be specified differently as the first 128 segment numbers are reserved for them. This means that users must take care of assigning shorter segments and those requiring very fast switching and jumping to those specific segment numbers.

9.1 :TRACe[:DATA](?) [<offset>]#<header><binary_block>

Description

This command will download waveform data starting from the specified offset to the Proteus waveform memory. Waveform data is loaded to the Proteus using high-speed binary data transfer. High-speed binary data transfer allows any number of 8-bit bytes to be transmitted in a message. This command is particularly useful for sending large quantities of data. As an example, the next command will download to the generator an arbitrary block of data of 1,024 points

TRACe #42048<binary_block>

This command causes the transfer of 2,048 bytes of data (i.e. 1,024 waveform 16-bit samples) into the active memory segment. The data (header plus binary block) is formatted according to the IEEE-488.2 standard for Definite Length Arbitrary Block. The <header> is interpreted this way:

- The ASCII "#" (0x23) designates the start of the binary data block.
- "4" designates the number of digits in ASCII format that follow representing the binary data block size in bytes.
- "2048" is the number of bytes to follow.
- <binary_block> Represents waveform data

The query form of the command may specify the offset and the length of the expected data (both in bytes) as arguments separated by commas. Offset must be specified if expected data is specified. When no offset is specified, returned waveform data will start with the first sample in the target segment. When no length of expected data is specified, the remaining contents of the segment will be transferred.

The optional offset parameter makes feasible updating a section of a segment or the transfer of waveform binary data in chunks that are more manageable by the application software in the control computer memory. It also helps to break one of the limitations of the IEEE-488.2 Definite Length Arbitrary Block format. As the number of digits to specify the binary block size is one (1-9), the maximum number of bytes that can be transferred through a single :TRACE command would be 999,999,999. This number is much shorter than the maximum waveform size. When the intended size of the waveform to be transferred is larger than this, multiple transfers with the right offset parameters can be used to overcome the format primary limitation.

The generator accepts waveform samples as either 8-bit, sent as one-byte word, or 16-bit unsigned integers, which are sent in two-byte words. Therefore, the total number of bytes may be equal to or twice the number of data points in the waveform. For example, 20,000 bytes are required to download a waveform with 10,000 points in the 16-bit format, and 10,000 bytes in

the 8-bit format. The IEEE-STD-488.2 definition of Definite Length Arbitrary Block Data format is demonstrated in Figure x-y.

For the waveform data made of 16-bit words, however, programmers may choose to prepare the data in two bytes and arrange to download these two bytes in a sequence (low byte followed by high byte). There are some facts you should be aware of before you start preparing the data:

1. Waveform data points have 16-bit unsigned integer values (0x0000 to 0xFFFF) or 8-bit unsigned integer values (0x00 to 0xFF).
2. For the 16-bit sample format, data point range is 0 to 65,535 decimal for the Proteus. 0x0000 corresponds to -Amplitude/2 +offset and 0xFFFF corresponds to Amplitude/2 +offset. Point 32,768 corresponds to offset setting.
3. For the 8-bit sample format, data point range is 0 to 255 decimal for the Proteus. 0x00 corresponds to -Amplitude/2 +offset and 0xFF corresponds to Amplitude/2 +offset. Point 128 corresponds to offset setting.

Complex waveform data (I/Q) to be used while the generator works in the DUC (Digital Up-Converter) mode, is handled in a very similar way. However, I and Q waveforms must be sometimes stored in the same segment in the waveform memory with some specific formats depending on the DUC mode. See chapter 5.9 [:SOURce]:IQModulation {NONE|HALF|ONE|TWO}(?), page 81.

Marker data is transferred trough a different command (see :MARKer:DATA). Data format can be selected by the user independently of the DAC working mode in the target Proteus unit through the :TRAC:FORM command. However, the transferred data may be modified by the Proteus unit receiving the data depending on the DAC working mode. This is the behavior of the Proteus unit depending on the specific model and DAC working mode:

- P128X, P258X, and P948X (Sclk < = 2.5GS/s): Waveform data is always stored in the internal waveform memory as 16-bit unsigned integers. Downloading 8-bit integers will cause the Proteus unit to fill up the MSB with all ‘0’.
- P9082, P948X (Sclk >2.5GS/s), 1X interpolation factor: Waveform data is always stored as 8-bit unsigned integers in the internal waveform memory. Downloading 16-bit waveform data will cause the Proteus unit to round the MSB to the nearest integer depending on the contents of the LSB, that will be discarded.
- P9082, P948X (Sclk >5GS/s), 2X interpolation factor: Waveform data is always stored as 8-bit unsigned integers in the internal waveform memory. Downloading 16-bit waveform data will cause the Proteus unit to round the MSB to the nearest integer depending on the contents of the LSB, that will be discarded.
- P948X (Sclk <= 9GS/s), 4X interpolation factor: Waveform data is always stored as 16-bit unsigned integers in the internal waveform memory. Downloading 8-bit integers will cause the Proteus unit to fill up the MSB with all ‘0’.
- P9082, all the other DAC modes: Waveform data is always stored in the internal waveform memory as 16-bit unsigned integers. Downloading 8-bit integers will cause the Proteus unit to fill up the MSB with all ‘0’.

- All models, IQ Mode: Waveform data is always stored as 16-bit sample pairs (I&Q). Waveform information, in this mode, is composed by a series of interleaved I/Q samples (I1, Q1, I2, Q2,…In, Qn) applied simultaneously to the DUC. The way the IQ waveforms are downloaded is set through the :TRACe:FORMat command. I/Q Waveforms can be downloaded together in an interleaved fashion, or independently, as separate I and Q downloads depending on the IQ mode. See chapter 5.9 [:SOURce]:IQModulation {NONE|HALF|ONE|TWO}(?), page 81.

Parameters

Name	Range	Type	Default	Description
[<offset>]	0, <seg_len>	Numeric(int)	0	Expressed in bytes. The position of the first sample to be transferred. Use this for editing the waveform and / or multiple transfer downloads. The offset is in bytes of wave data. Use an offset that is a multiple of 64 for performance reasons.
< header >		Numeric(int)		Contains information on the size of the binary block that contains waveform coordinates.
< binary_block >		Binary		Block of binary data that contains waveform data points (vertical coordinates), as explained above. Use a data size that is a multiple of 64 for performance reasons.

Response

Will upload (read) waveform data starting from the specified offset from the Proteus waveform memory. Note to use a comma after the offset.

Example

Command `:TRAC 64#42048<binary_block>`

Query `:TRAC? 64, 2048`

9.2 :TRACe:FORMat{ <U16 | U8>}(?)

Description

Set the resolution of the user waveform data that is to be transferred to the Proteus to unsigned 16/8-bit. This command does not modify the internal sample size (see :TRACe:DATA command), just the size of the downloaded data.

Parameters

Name	Type	Default	Description
< U16 >	Discrete	U16	Unsigned 16 bit.
< U8>	Discrete	U8 for P908x	Unsigned 8 bit.

Response

The Proteus unit will return the data format.

Example

Command `:TRAC:FORM U16`

Query **:TRAC:FORM?**

9.3 :TRACe:MEMory(?)< offset_in_wave-points>#<header><wave-data>

Description

Write waveform data to the arbitrary-memory space starting from the specified offset.

The query format is:

:TRAC:MEMory? [<offset in wave-points>,]<size in wave-points>.

This command (or query) is the same as

:TRACe[:DATA] [<offset-in-bytes-of-wave-data>] #<binary-header><binary-data>

except that the offset, in case of **:TRACe:MEMory**, is from the beginning of the memory-space rather than the beginning of memory of the selected segment. This command ca be used to write or read multiple segments at once.

Parameters

Name	Range	Type	Default	Description
<offset_in_wave-points >	Up to total waveform memory.	Integer (int)		The position of the first sample to be transferred. Use this for transferring large amount of data in multiple transfer downloads. The offset is in bytes of wave-data.
<#header><wave-data>				Refer to 9.1 :TRACe[:DATA](?) [<offset>]#<header><binary_block>, page 123.

Response

Returns the waveform data from specified location.

Example

Command **:TRAC:MEM 64#42048<binary_block>**

Query **:TRAC:MEM? 64,1024**

9.4 :TRACe:SEGMents[:DATA] [<first segment number>,]#<header><binary_block>

Description

Delete the previous definition, if any, of all the designated N segments of the selected channel (no sample information is actually deleted) and define N consecutive new segments (no query). The N segment-lengths, expressed in bytes of wave-data, are specified by the binary-block which consists of N uint64 values (8N bytes). The new segments are allocated, one after the other, from the beginning of the arbitrary-memory space.

Parameters

Name	Range	Type	Default	Description
[<first segment number>]		Numeric(int)	1	Defines the first segment to start from.

Name	Range	Type	Default	Description
< header >		Numeric(int)		Contains information on the size of the binary block that contains segment lengths.
<binary_block>	1 – 64k	Numeric(int)		The binary data consists of N uint64 values that define the lengths in wave-points of N segments

Example

Command `:TRAC:SEGM #42048<binary_block>`

9.5 :TRACe:SEGMents:FILE[:NAMe] #<header><binary_block>

Description

This command will identify the file path storing the waveform data information for further transfers to/from the unit's memory. The file path is passed as a binary-block.

Parameters

Name	Type	Description
<header>	<discrete>	The first digit in ASCII following the '#' character is the number of digits to follow. The following digits specify the length of the target file full path name in ASCII.
<binary_block>	String	Full path name for the file in ASCII coded as an unsigned short integer array.

Example

Command `:TRAC:SEGM:FILE #220my_segment_table.seg`

9.6 :TRACe:SEGMents:FILE:OFFSet <offset in bytes>(?)

Description

Set the start offset inside the file in bytes.

Parameters

Name	Range	Type	Default	Description
< offset in bytes >		Integer (int)	0	Set the start offset inside the file in bytes.

Response

Returns the start offset inside the file in bytes.

Example

Command `:TRAC:SEGM:FILE:OFFS 28`

Query `:TRAC:SEGM:FILE:OFFS?`

9.7 :TRACe:SEGMents:FILE:LOAD[[<first segment number>,],<number of segments>]

Description

This command will load the segment table data from the file defined by the **:TRACe:SEGM:FILE:NAME** command to the Proteus unit memory. If the first segment is not specified, then the default segment is 1.

Parameters

Name	Type	Default	Description
[<first segment number>]	Integer	1	Defines the first segment to start from. Optional argument, default is 1.
<number of segments>	Integer	-1 (until last segment)	The binary data consists of N uint64 values that define the lengths in wave-points of N segments

Example

Command **:TRAC:SEGM:FILE:LOAD 12,128**

9.8 :TRACe:FILE[:NAME]#<header><binary_block>

Description

This command will identify the file path storing the waveform data information for further transfers to/from the unit's memory. The file path is passed as a binary-block

Parameters

Name	Type	Description
<header>	<discrete>	The first digit in ASCII following the '#' character is the number of digits to follow. The following digits specify the length of the target file full path name in ASCII.
<binary_block>	String	Full path name for the file in ASCII coded as an unsigned short integer array.

Example

Command **:TRAC:FILE #210wave_1.wav**

9.9 :TRACe:FILE:OFFSet< offset in bytes>(?)

Description

Set the start offset inside the file in bytes.

Parameters

Name	Range	Type	Default	Description
< offset in bytes >	Up to total waveform memory.	Integer (int)	0	Set the start offset inside the file in bytes.

Response

Returns the start offset inside the file in bytes.

Example

Command `:TRAC:FILE:OFFS 28`

Query `:TRAC:FILE:OFFS?`

9.10 :TRACe:FILE:DESTination{SEGMent | MEMory}(?)

Description

Use this command to set the destination to load/store the file data.

Parameters

Name	Type	Default	Description
< SEGMent >	Discrete	SEGM	The selected (programmable) segment.
< MEMory >	Discrete		The arbitrary memory space.

Response

The Proteus unit will return the destination to load/store the file data.

Example

Command `:TRACe:FILE:DEST MEM`

Query `:TRACe:FILE:DEST?`

9.11 :TRACe:FILE:LOAD[<offset>,<size in wave-points>]

Description

This command will load the waveform data from the file defined by the `:TRACe:FILE:NAME` command to the Proteus unit memory. If the offset and the number of wave-points are not specified, then the whole segment is written. No query.

Parameters

Name	Type	Description
<offset>	Integer	Offset in the segment in wave-points.
<size>	Integer	Size in wave-points.

Example

Command `:TRAC:FILE:LOAD 12,128`

9.12 :TRACe:FILE:STORe[<offset>,<size>]

Description

This command will save waveform data from the Proteus unit to the file defined by the **`:TRAC:FILE:NAME`** command. If the offset and the number of wave-points are not specified, then the whole segment is read. No query.

Parameters

Name	Type	Description
<offset>	Integer	Offset in the segment in wave-points.
<size>	Integer	Size in wave-points.

Example

Command **:TRACe:FILE:STOR 1,1280**

9.13 :TRACe:STReaming:MODE {FILE|DYNamic}(?)

Description

This command will set or query the target type of streaming mode.

Parameters

Name	Type	Default	Description
< FILE >	Discrete	Dynamic	The waveform data is streamed from a file.
<DYNamic>	Discrete		The waveform data is generated continuously by an API and streamed to the Proteus.

Response

The Proteus will return FILE or DYNamic.

Example

Command **:TRAC:STR:MODE FILE**

Query **:TRAC:STR:MODE?**

9.14 :TRACe:STReaming:STATe{OFF|ON|0|1}(?)

Description

This command will set or query the state of the streaming functionality. Only for units with installed streaming option (STM).

Parameters

Name	Type	Default	Description
OFF	Discrete	0	Disable streaming.
ON	Discrete		Enable streaming.

Response

The Proteus will return 1 if streaming is on, or 0 if streaming is off.

Example

Command **:TRAC:STR ON**

Query **:TRAC:STR?**

9.15 :TRACe:DEFine[:SIMPle] [<seg_number>,]<seg_length>(?)

Description

Use this command to specify a new segment including its length.

Parameters

Name	Range	Type	Default	Description
<seg_number>	1-64k	Numeric (int)	1	Segment number to be defined. The segment-number is optional (for backward compatibility).
<seg_length>	See device specification	Numeric (int)		Length in samples. Length will be rounded to the nearest valid length lower or equal to <seg_length> if greater than the minimum valid segment size.

Response

The Proteus unit will return the segment number currently selected and its size separated by commas.

Example

Command `:TRAC:DEF 54, 1024`

Query `:TRAC:DEF?`

9.16 :TRACe:DEFine:LENGth?

Description

Query only. Returns the length in wave-points of the selected (programmable) segment.

Response

The Proteus unit will return the length in wave-points of the selected (programmable) segment in ASCII format.

Example

Query `:TRAC:DEF:LENG?`

9.17 :TRACe:ZERO[:SEGMent] [<segment number>]

Description

Zero the markers and waveform data of specified single segment (no query). The segment-number is optional. If it is not given then the current segment is zeroed.

Parameters

Name	Type	Description
< segment-number >	Integer	Segment number in the device waveform memory to be processed.

Example

Command `:TRAC:ZERO 100`

9.18 :TRACe:ZERO:ALL

Description

This command will zero all the arbitrary-memory space of the selected channel's DDR (no query).

Example

Command `:TRAC:ZERO:ALL`

9.19 :TRACe:DELete[:SEGMent] <seg-number>

Description

This command will delete the predefined segment from the working memory.

Parameters

Name	Range	Type	Description
< seg-number>	1 to 64k	Numeric(int)	Select the segment number to be deleted.

Example

Command `:TRAC:DEL 1`

9.20 :TRACe:DELete[:SEGMent]:ALL

Description

Delete all segments of the programmable channel's DDR.

Example

Command `:TRAC:DEL:ALL`

9.21 :TRACe:SELect[:SEGMent] <seg_number>(?)

Description

Use this command to specify the segment to be selected. Do not confuse it with the selected-segment for playback (:FUNCtion:MODE:SEGMent).

Parameters

Name	Range	Type	Default	Description
<seg_number>	1-64k	Numeric (int)	1	Segment number to be defined.

Response

The Proteus unit will return the segment number currently selected and its size separated by commas.

Example

Command `:TRAC:SEL 54`

Query **:TRAC:SEL?**

9.22 :TRACe:SELect:SOURce{ BUS | EXTernal | ADC | DCT }(?)

Description

Use this command to set or query the source of the segment select command. This defines from where the select command is expected to be received, causing a waveform segment change. Using the BUS option, waveforms can be selected using remote commands only. The EXT option transfers the control to a connector in the front panel that allows dynamic selection of the active waveform segment. Using the external waveform control, one can dynamically select a waveform from a preprogrammed list of waveforms. Using the ADC option, waveforms can be selected by the digitizer trigger. The transition characteristics from waveform segment to another is programmed using the TRAC:SEL:TIM command.

Parameters

Name	Type	Default	Description
BUS	Discrete	BUS	Defines that waveform segments will be switched only when a remote command has been received.
EXTernal	Discrete		Defines that the segment control is transferred to sequence control connector. The connector has 8 bits of parallel control lines that can switch between up to 256 segments.
ADC	Discrete		Source for segment selection (for playback) is by the ADC trigger (if this option is supported).
DCT Future Release	Discrete		Source for segment selection (for playback) is by the daisy-chain-trigger.

Response

The Proteus unit will return BUS, EXT, ADC or DCT depending on the present segment jump setting.

Example

Command **:TRAC:SEL:SOUR DCT**

Query **:TRAC:SEL:SOUR?**

9.23 :TRACe:SELect:TIMing{ EVENTually|IMMediate}(?)

Future Release

Description

Use this command to set or query the timing characteristics of the trace select command. This defines how the generator transitions from waveform to waveform. Use the eventually option to let the waveform complete before it jumps to the next waveform. Applications that require an unconditional jump can use the immediate option, where the generation of the current waveform is aborted and the new waveform is started immediately thereafter. This command affects the

segment transition timing, regardless of if the segment control is from remote or from the rear panel connector.

Parameters

Name	Type	Default	Description
<EVENtually>	Discrete	EVEN	Defines that when a new waveform segment is selected, the transition to the new waveform will occur only when the current waveform has reached its end point.
<IMMediately>	Discrete		Defines that when a new waveform segment is selected, the current waveform will be aborted and the transition to the new waveform will occur immediately, without waiting for the current waveform to reach its end points.

Response

The Proteus will return EVEN or IMM, depending on the current segment jump timing setting.

Example

Command `:TRAC:SEL:TIM IMM`

Query `:TRAC:SEL:TIM?`

9.24 :TRACe:FREE?

Description

Query only. Query the available waveform memory in the DDR including biggest fragment. Ask for the current behavior of this command as it should give the biggest section of contiguous data so users can define a new segment of equal or shorter length than that. Additionally, it could return a second number with the total free memory.

Response

The Proteus unit will return the available waveform memory.

Example

Query `:TRAC:FREE?`

9.25 :TRACe:FRAG?

Description

Query only. Query the fragmentation level of the of the selected channel's memory-space. Fragmentation can occur after some existing segments are deleted. Waveform and marker data for a give segment is always stored in contiguous sections of the memory. This means that unused sections of the memory can only be reused by a new segment if its length is equal or shorter than the largest unused section.

Response

The Proteus unit will return the fragmentation level (between 0 and 1).

Example

Query `:TRAC:FRAG?`

9.26 :TRACe:DEFRag

Description

This command will defragment the arbitrary-memory space of the selected channel DDR.

Example

Query `:TRAC:DEFR`

10 Digitizer Commands

This group is used to control the multiple-channel digitizer in the Proteus AWT (Arbitrary Waveform Transceiver) models and handle the waveforms captured by it and their respective parameters. Using these commands, you can define acquisition size and upload some or all waveforms from the acquisition memory.

Acquiring Waveforms

Acquired waveforms are digital, quantized versions of the analog waveforms at the input(s) of the digitizer, which are stored in a dedicated waveform memory. Quantization of the incoming signal is made in two dimensions, time and amplitude. Each one of the elements in the acquisition memory is a sample taken at a constant speed known as sampling rate or frequency. The amplitude for each sample point is quantized as a finite digital word. The size of the digital word is known as the vertical resolution and it is expressed in bits (so there are 2^N quantization levels, where N is the vertical resolution in bits). The Proteus' digitizer has a vertical resolution of 12 bits (4,096 levels), i.e., each sample is placed on the vertical axis with a precision of 1/4096. The Proteus digitizer supports two acquisition modes. In the first mode, all the acquisition channels are available to acquire independent waveforms up to 2.7GSa/s. In the second mode, half of the channels can be used to capture waveforms up to 5.4GSa/s.

The acquisition memory is shared with the AWG section of Proteus so the available sizes for both sections are interlocked. Acquisition memory can be made of a single waveform or by a multiplicity of them grouped as an array of waveforms. Each element of the array is called a frame.

Acquisitions are defined in terms of sampling rate, record length (number of samples to be captured for each trigger event), and position (the location of the closest sample to the trigger event). Multiple frame acquisitions (or Multi-Frame) require the definition of the number of frames to be captured.

Acquisitions can be performed in the Real mode, where samples are stored in the acquisition memory directly after digitization, and in the Complex model, where samples go through a real-time Digital Down-Converter (DDC) so a region in the frequency domain around a central or carrier frequency is down-converted to a complex (I/Q) waveform after complex mixing, filtering, and decimation.

Captured waveforms can be read at any moment by the control software and acquisition does not need to be stopped. However, typically acquisitions are read synchronously when acquisitions are completed so the Proteus unit should be queried about the status of the current acquisition to avoid missing or mixing waveform information.

10.1 :DIGitizer:[:SELect]{ DIG1|DIG2}(?)

Description

This command will set the active Digitizer in the Proteus for future programming command sequences. Subsequent commands affect the selected Proteus Digitizer only. This command is not required for PXI modules.

Parameters

Name	Type	Default	Description
DIG1	Discrete	DIG1	Sets the active device to Digitizer #1

Name	Type	Default	Description
DIG2	Discrete		Sets the active device to Digitizer #2

Response

The Proteus unit will return 0 (DIG1) or 1 (DIG2) depending on the present active digitizer setting.

Example

Command `:DIG DIG2`

Query `:DIG?`

10.2 :DIGitizer:MODE{DUAL|SINGle}(?)

Description

This command will set the operation mode for the active digitizer in a Proteus device. Dual channel mode allows for the simultaneous acquisition of two waveforms at sampling rates up to 2.7GSa/s. The single-channel mode supports one channel (CH1)at twice the sampling rate (5.4GSa/s).

Parameters

Name	Type	Default	Description
DUAL	Discrete	DUAL	Sets the mode to dual channel.
SINGle	Discrete		Sets the mode to single channel.

Response

The Proteus unit will return DUAL or SING depending on the operation mode for the current active digitizer.

Example

Command `:DIG:MODE SING`

Query `:DIG:MODE?`

10.3 :DIGitizer:CHANnel[:SELect]{ CH1|CH2}(?)

Description

This command will set the active channel for the currently selected digitizer in a Proteus device for future programming command sequences. Subsequent commands affect the selected digitizer channel only. Selecting CH2 while in the single channel operation mode will have no effect.

Parameters

Name	Type	Default	Description
CH1	Discrete	CH1	Sets the active channel to Channel #1
CH2	Discrete		Sets the active channel to Channel #2

Response

The Proteus unit will return 1 (CH1) or 2 (CH2) depending on the present active channel setting.

Example

Command `:DIG:CHAN CH2`

Query **:DIG:CHAN?**

10.4 :DIGitizer:CHANnel:STATe{ DISabled|ENABled}(?)

Description

This command will enable acquisition for the active channel in the currently selected digitizer in a Proteus device. Non enabled channels will not capture any waveform.

Parameters

Name	Type	Default	Description
DISabled	Discrete	DISabled	Active channel will be disabled
ENABled	Discrete		Active channel will be enabled

Response

The Proteus unit will return DIS or ENAB depending on the present active channel setting.

Example

Command **:DIG:CHAN:STAT ENAB**

Query **:DIG:CHAN:STAT?**

10.5 :DIGitizer:CHANnel:RANGe{ HIGH|MEDium|LOW}(?)

Description

This command will set the voltage range for the active channel for the currently selected Digitizer in a Proteus device. There are three different sensitivity settings: HIGH (500mVpp), Medium(400mVpp), and LOW (250mVpp).

Parameters

Name	Type	Default	Description
HIGH	Discrete	HIGH	Sets the active channel to the high range
MEDium	Discrete		Sets the active channel to the medium range
LOW	Discrete		Sets the active channel to the low range

Response

The Proteus unit will return HIGH, MED or LOW depending on the present active channel setting.

Example

Command **:DIG:CHAN:RANG MED**

Query **:DIG:CHAN:RANG?**

10.6 :DIGitizer:CHANnel:OFFSet< offset_level >(?)

Description

Use this command to set or query the DC offset level for the active channel for the currently selected digitizer in a Proteus device. The DC offset level is defined in units of volts.

Parameters

Name	Range	Type	Default	Description
< offset_level >	-2.0 to +2.0	Numeric	0	Set the offset level in units of volts. Each input can be set to a different low- and high-level setting. The offset level is calibrated when the output impedance of the signal source is 50 Ω.

Response

The Proteus will return the present channel offset value. The returned value will be in standard engineering format. For example: 0.1 V would be returned as 100e-3. Positive numbers are unsigned.

Example

Command `:DIG:CHAN:OFFS 0.5`

Query `:DIG:CHAN:OFFS?`

10.7 :DIGitizer:DDC:MODE{REAL|COMPlex}(?)

Description

This command will set the path for the data acquired by the digitizer channel. There are two different data paths: REAL and COMPlex. The real path is the default setting of the digitizer. This setting acquires the signal and stores directly it in the DDR.

When set to complex mode the signal is demodulated, filtered, decimated so it is down-converted. This is done using the integrated digital downconverter (DDC) in the ADC chip. There are two DDC entities, one for each channel of the digitizer. The complex mode is available only in dual channel mode or in single channel mode for sampling rates <= 2.7GS/s, and forces decimation X16. For future reference the DDC of each channel will be labelled according to the channel number, DDC1 and DDC2. The DDC mode is common to both channels. In Single mode, both DDCs can be applied concurrently to Channel 1,

Parameters

Name	Type	Default	Description
REAL	Discrete	REAL	Set data path to REAL
COMPlex	Discrete		Set data path to Complex and enable demodulation and digital down conversion

Response

The Proteus unit will return REAL or COMP depending on the present setting.

Example

Command `:DIG:DDC:MODE REAL`

Query `:DIG:DDC:MODE?`

10.8 :DIGitizer:DDC:DECimation{ NONE | X1| X4 | X16}?

Description

Set the decimation factor when working in complex mode. In real mode decimation is X1. In complex mode user can set X4 or X16. X4 is mandatory for loopback mode and it cannot be set out of this mode. When querying the decimation factor, the possible responses are X1|X4|X16.

Parameters

Range	Type	Default	Description
NONE	Discrete	X16	No decimation factor, X1. (Real Mode)
X1	Discrete		No decimation factor, X1. (Real Mode)
X4	Discrete		Set decimation factor to X4. (Complex Mode and Loopback Mode)
X16	Discrete		Set decimation factor to X16. (Complex Mode and not Loopback Mode)

Response

The Proteus unit will return the decimation factor applied to the signal acquired.

Example

Command `:DIG:DDC:DEC X16`

Query `:DIG:DDC:DEC?`

10.9 :DIGitizer:DDC:BIND{ OFF | ON |0 |1 }(?)

Description

Use this command to enable or disable binding of the digitizer channels. It is only effective in the Digitizer's Single mode. When the state of the BIND is ON then only channel 1 is active and is the source for DDC1 and DDC2. Sampling Rate must be <= 2.7GS/s.

When the state is OFF channel 1 is the source of DDC1 and channel 2 is the source of DDC2. Note: Only for :MODE:DUAL.

Parameters

Range	Type	Default	Description
0-1	Discrete	0	When set to 0, channel binding is OFF. When set to 1, channel binding is ON.

Response

The Proteus will return 1 if the channel binding state is set to ON, or 0 if set to OFF.

Example

Command `:DIG:DDC:BIND ON`

Query `:DIG:DDC:BIND?`

10.10 :DIGitizer:DDC:CFRequency<1|2> <carr_freq>(?)

Description

Use this command to set or query the carrier frequency for the selected DDC <1|2> . Where CFR1 sets DDC1 and CFR2 sets DDC2.

Parameters

Name	Range	Type	Default	Description
< carr_freq >	0 Hz to max ADC sclk	Numeric	1e9	Will set the carrier frequency of the selected DDC.

Response

The Proteus unit will return the present carrier frequency value. The returned value will be in standard engineering format (for example: 1 GHz would be returned as 1e9, positive numbers are unsigned).

Example

Command `:DIG:DDC:CFR1 1.0e9`

Query `:DIG:DDC:CFR1?`

10.11 :DIGitizer:DDC:PHASe<1|2> {<phase in degrees>}(?)

Description

Use this command to set or query the NCO phase (in degrees) of the selected DDC.

Parameters

Name	Range	Type	Default	Description
<phase in degrees>	0 to 360	Numeric	0	Will set the NCO phase (in degrees) of the selected DDC.

Response

The Proteus unit will return the NCO phase.

Example

Command `:DIG:DDC:PHAS1 45`

Query `:DIG:DDC:PHAS1?`

10.12 :DIGitizer:DDC:CLKSource{ DIG|AWG}(?)

Description

This command will set the operation mode for the clock source of the digitizer sampling clock. When set to AWG the clock source is from the AWG part of the instrument and results in the synchronization of frequency and phase of the NCO in the generator and the NCO of the digitizer so coherent operation between the generator and the digitizer can be obtained. This setting links the SCLK of the generator and digitizer within certain ranges as can be seen in the table below. Sampling rate for the Generator must be exactly 1X, 2X, 4X, or 8X the sample rate for the Digitizer to implement coherent operation of the DUC and the DDC.

Table 10-1 Digitizer and Generator Sampling Clock Ranges for Synchronized Operation

Digitizer Sampling Clock (MSa/s)	Generator Sampling Clock (MSa/s)			
1000-1125	1000-1125	2000-2250	4000-4500	8000-9000
1125-1600	1125-1600	2250-3200	4500-6400	
1600-2000	1600-2000	3200-4000	6400-8000	
2000-2250	2000-2250	4000-4500	8000-9000	
2250-2700	2250-2700	4500-5400		

Parameters

Name	Type	Default	Description
DIG	Discrete	DIG	Sets the CLK source to be the digitizer
AWG	Discrete		Sets the CLK source to be the generator

Response

The Proteus unit will return DIG or AWG depending on the operation mode for the current active digitizer.

Example

Command `:DIG:DDC:CLKS AWG`

Query `:DIG:DDC:CLKS?`

10.13 :DIGitizer:ACQuire[:FRAMes]:DEFine<num_of_frames><frame_length> (?)

Description

Use this command to define memory-space and frame length for the digitizer channels with state enabled. If the digitizer mode is DUAL, then the frame-length should be a multiple of 48 samples, and size of the frame-header is 48 samples (it is not part of the frame-length, but it is added to the memory size). If the digitizer mode is SINGle, then the frame-length should be a multiple of 96 samples, and the size of the frame-header is 96.

In case of dual-mode, the memory for channel1 of the digitizer is in DDR1 and for channel 2 in DDR2. In case of single-mode the memory for the single channel is divided half-by-half between DDR1 and DDR2. Number of frames and frame length are common to all digitizer channels in a given module.

> **Note**
> The DDR1 is used by AWG channel 1 for P908X models and by AWG channels 1 and 2 if for all other models. DDR2 is used by AWG channel 2 for P908X models and by AWG channels 3 and 4 for all other models.

The user should balance between the memory reserved for the digitizer, and the memory for the segments of the AWG (48 samples of the digitizer are equivalent to 64 bytes of AWG waveform data).

Parameters

Name	Range	Type	Default	Description
<num_of_frames>	1-wave form memory	Numeric (int)	1	Number of frames (segments) to capture.
<frame_length>	nx48/nx96	Numeric (int)	4800	Length of each frame for a single acquisition. length-granularity: 48 in case of dual-channel mode, 96 in case of single-channel mode. n= any integer number >= 100

Response

The Proteus unit will return the frame length and number of frames currently set for the current channel.

Example

Command `:DIG:ACQ:DEF 1,48000`

Query `:DIG:ACQ:DEF?`

10.14 :DIGitizer:ACQuire[:FRAMes]:FREE

Description

This command will free the memory allocated on the DDR.

Example

Command `:DIGitizer:ACQuire:FREE`

10.15 :DIGitizer:ACQuire[:FRAMes]:CAPTure[:SELECT]<1st frame>,<num-frames> (?)

Description

Use this command to capture a number of frames (waveforms) starting from specified first frame and up to first frame plus number of frames. This command allows to capture a limited range of frames under software control.

Parameters

Name	Range	Type	Default	Description
<1st frame>	1 - Number of frames	Numeric (int)	1	Start the capture from specified frame.
<num-frames>	1 -Number of frames	Numeric (int)	-1	Number of frames (segments) to capture. Number of frames -1 means to the last frame.

Response

The Proteus unit will return the specified first frame and the number of frames currently set for the selected channel.

Example

Command `:DIG:ACQ:CAPT 10,1004`

Query `:DIG:ACQ:CAPT?`

10.16 :DIGitizer:ACQuire:[FRAMES]:CAPTure:ALL

Description

Use this command to capture all frames from frame number 1 to the last frame.

Example

Command `:DIG:ACQ:CAPT:ALL`

10.17 :DIGitizer:ACQuire[:FRAMes]:MARKer{OFF|ON|0|1}(?)

Description

If the marker-mode is enabled, then the digitizer is set to the external trigger input. In this mode the LSB of the captured data is a marker that holds the state of the capturing trigger signal. When the marker bit rises for the first time after the pre-trigger section of the frame, it marks the sample at which the trigger signal was received.

Parameters

Range	Type	Default	Description
0-1	Discrete	0	Sets the marker outputs on and off.

Response

The Proteus will return 1 if the marker output is ON, or 0 if the marker output is OFF.

Example

Command `:DIG:ACQ:MARK ON`

Query `:DIG:ACQ:MARK?`

10.18 :DIGitizer:ACQuire:STATus?

Description

Query only. Get the status of the acquisition.

Parameters in the Response

Range	Type	Description
<frame-done>	Discrete	0 – No frame acquired 1 – At least one frame acquired
<all-frames-done>	Discrete	0 – No all frames acquired 1 – All frames acquired
<pulse-counter-busy>	Discrete	0 – Not busy 1 – Busy

Range	Type	Description
<frames-count>	Integer	Number of frames acquired so far.

Response

The Proteus will return the status of the acquisition. The format of the answer is <frame-done>,<all-frames-done>,<pulse-counter-busy>,<frames-count>

Example

Query **:DIG:ACQ:STAT?**

10.19 :DIGitizer:ACQuire:AVERage:STATe{ OFF | ON |0 |1 }(?)

Description

The average function of the digitizer enables the user to perform time domain averaging of up to 16M captures. The result is stored in single frame in the DDR and consists of the sum of the number of captures as defined by the :DIG:ACQ:AVER:COUN command. The length of the frame is as defined by the user and is limited to a maximum of 10244 samples where each sample is up to 28bits long.

Use this command to enable or disable averaging of the captured frames. When set to 0 the averaging is OFF and when set to 1 the averaging is ON.

Note that the average function only works in certain clock ranges of the digitizer depending on the DDC Mode. In addition, the sampling clock of the generator part is derived from the digitizer sampling clock according to a specific mathematical formula. The sampling clock ranges, and formula are summarized in table 10.2 below.

Table 10-2 Average Mode Digitizer and Sampling Clock Settings

	DDC Complex Mode	DDC Real Mode
Valid digitizer clock range	2500MSa/s – 2700MSa/s	5000MSa/s – 5400MSa/s
Corresponding generator sampling clock	$DIG\_SCLK \times 3.2$ =>8000MSa/s – 8640MSa/a	$DIG\_SCLK \times 1.6$ =>8000MSa/s – 8640MSa/a

For example, when setting the digitizer sampling clock (SCLK) to 5100MSa/s in real mode, the generator sampling clock must be set to 5100x1.6 = 8160MSa/s.

Parameters

Range	Type	Default	Description
0-1	Discrete	0	**0** – Averaging is OFF. **1** – Averaging is ON.

Response

The Proteus will return 1 if the averaging is set to ON, or 0 if set to OFF.

Example

Command **:DIG:ACQ:AVER:STAT ON**

Query **:DIG:ACQ:AVER:STAT?**

10.20 :DIGitizer:ACQuire:AVERage:COUNt<# frames to average>

Description

Set the count of how many frames to average. The minimum number of frames to average is 2, and the maximum is 16M.

Parameters

Name	Range	Type	Default	Description
<# frames to average>	2 to 16M	Numeric	1000	How many frames to average.

Response

The Proteus will return the present count value.

Example

Command `:DIG:ACQ:AVER:COUN 100`

Query `:DIG:ACQ:AVER:COUN?`

10.21 :DIGitizer:ACQuire:ZERO[:SELect]<1st frame>,<num frames>,<fill value>

Description

This command will set the value of the data of the selected frames in the acquisition memory to the “fill value”. This command enables to actively wipe the data in the frames to a defined value.

Parameters

Name	Type	Description
<1st frame>	Integer	First frame in the Proteus acquisition memory to be set to fill value.
<num frames>	Integer	Number of frames in the Proteus acquisition memory to be set to a defined value. **-1** – Means to the last frame.
<fill value>		Fill-value is 12bits. Delete frame data.

Example

Command `:DIG:ACQ:ZERO 1,15,2047`

10.22 :DIGitizer:ACQuire:ZERO:ALL <fill value>

Description

This command will set the value of the data of all frames in the acquisition memory to the “fill value”. This command enables to wipe the data in all frames to a defined value

Parameters

Name	Type	Description
<fill value>	Integer	Fill-value is 12bits. Deletes the original frame data.

Example

Command `:DIG:ACQ:ZERO:ALL 0`

10.23 :DIGitizer:FREQuency[:RASTer]{<sclk>|MAXimum|MINimum}(?)

Description

Use this command to set or query the sample clock frequency of the digitizer section in units of samples per second (Sa/s).

Parameters

Name	Range	Type	Default	Description
< sclk >	1.6e9 to 5.4e9 (single mode) 800e6 to 2.7e9 (dual mode)	Numeric	2.00E+09	Will set the sample clock frequency of the digitizer in units of Sa/s. The sample clock command can be programmed with resolution up to 5 digits.
<MINimum>		Discrete		Will set sample clock to min SCLK. 1.6e09 Sa/s for single channel mode and 800e06 Sa/s for dual channel mode
<MAXimum>		Discrete		Will set sample clock to max SCLK. 5.4e09 Sa/s for single channel mode and 2.7e09 Sa/s for dual channel mode

Response

The Proteus unit will return the present sample clock frequency value for the digitizer. The returned value will be in standard scientific format with up to 6 digits (for example: 1 GHz would be returned as 1e9. Positive numbers are unsigned).

Example

Command `:DIG:FREQ 5.0e9;`

Query `:DIG:FREQ?`

10.24 :DIGitizer:FREQuency:SOURce{INTernal|EXTernal}(?)

Description

Use this command to select or query the source of the sample clock generator of the digitizer. The clock source can be set to external or internal. Make sure that a valid clock is applied to the external clock input before you change the option to external. Note that the sample clock generator is applied to both channels.

Parameters

Range	Type	Default	Description
INTernal	Discrete	INT	Selects the internal clock generator as the main clock source.
EXTernal	Discrete		Activates the external sample clock input. A valid signal must be applied the SCLK IN connector on the digitizer board. Observe the input level and limitations before connecting an external signal to the external sample clock input.

Response

The Proteus will return INT or EXT, depending on the current sample clock source.

Example

Command `:DIG:FREQ:SOUR EXT`

Query `:DIG:FREQ:SOUR?`

10.25 :DIGitizert:INITiate[:STATE]{OFF|ON|0|1}(?)

Description

Use this command to start or stop acquiring waveforms with the digitizer. Use the :DIG:ACQ:STAT to find out about the status of the acquisition after issuing this command.

Parameters

Range	Type	Default	Description
0-1	Discrete	0	**0** – Waveform acquisition will stop acquisition for the current digitizer. **1** – Waveform acquisition will start acquisition for the current digitizer.

Response

The Proteus will return 1 if the acquisition state is set to ON, or 0 if set to OFF.

Example

Command `:DIG:INIT ON`

Query `:DIG:INIT?`

10.26 :DIGitizer:TRIGger[:IMMediate]

Description

Use this command to force a trigger event for the digitizer when trigger source is set to CPU.

Example

Command `:DIG:TRIG`

10.27 :DIGitizer:TRIGger:SOURce{ CPU|EXT|CH1|CH2|TASK1|TASK2|TASK3|TASK4|MR1|MF1|MR2|MF2}(?)

Description

Use this command to set or query the source of the trigger that initiates the capturing of the selected digitizer channel.

Parameters

Name	Type	Default	Description
CPU	Discrete	EXT	By SCPI Command :DIG:TRIG:IMM.
EXTernal	Discrete		The external-trigger of the digitizer connected to the TRIG IN connector of the digitizer slot.
CH1|CH2	Discrete		Self-trigger from channel 1 or channel 2 of the digitizer.
TASk[n]	Discrete		Trigger created by task from the nth channel of the AWG (of the same module).
MR[1|2]	Discrete		Raise of the marker-bit on channel 1/2 of the digitizer. The marker is raised when a trigger signal is detected in the TRIG IN
MF[1/2]	Discrete		Fall of the marker-bit on channel 1/2 of the digitizer.

Response

The Proteus will return CPU| EXT| CH1| CH2| TASK1| TASK2| TASK3| TASK4|MR1|MF1|MR2|MF2 depending on the current source for acquisition triggers.

Example

Command `:DIG:TRIG:SOUR CPU`

Query `:DIG:TRIG:SOUR?`

10.28 :DIGitizer:TRIGger:LEVel<1|2>{<trigger_level>}(?)

Description

Use this command to set the voltage threshold Level<1|2> of the external-trigger of the digitizer. When using EDGE or GATE type trigger Level1 is set as the trigger threshold. For a window type trigger WEDGE and WGATE both trigger levels are set as explained in the :DIGitizer:TRIGger:TYPE{ EDGE | GATE | WEDGe | WGATe | CUSTom }(?) command.

Parameters

Name	Range	Type	Default	Description
<1|2>	-5.0V to +5.0V	Numeric	0V	Programs the trigger level <1|2> in volts.

Response

The Proteus will return the present trigger level set for the current digitizer.

Example

Command `:DIG:TRIG:LEV2 -0.2`

Query `:DIG:TRIG:LEV2?`

10.29 :DIGitizer:TRIGger:SELF[:LEVel]<trigger_level>(?)

Description

Use this command to set the threshold voltage level for the self-trigger for the selected channel of the digitizer.

Parameters

Name	Range	Type	Default	Description
<trigger_level>	Low Range: -125mV to +125mV Medium Range: -200mV to +200mV. High Range: -250mV to +250 mV	Numeric	0V	Programs the self-trigger level in volts.

Response

The Proteus will return the present self-trigger level set for the current digitizer.

Example

Command `:DIG:TRIG:SELF -0.2`

Query `:DIG:TRIG:SELF?`

10.30 :DIGitizer:TRIGger:TYPE{ EDGE | GATE | WEDGe | WGATe | CUSTom }(?)

Description

Use this command to set or query the type of trigger that will be derived from the external trigger of the digitizer.

Parameters

Name	Type	Default	Description
EDGE	Discrete	EDGE	Sets LEV1 as the trigger threshold. Slope setting will set the positive and negative edge values.
GATE			Sets LEV1 as the trigger threshold. Slope setting will set whether gate starts when crossing above (POS) or below the level (NEG).
WEDGe			Window Edge. Defines a window where once both the window start and stop conditions are met a trigger is initiated and capture begins. Can be combined with

Name	Type	Default	Description
			width, refer to :DIGitizer:TRIGger:WIDTh<trigger_event_width>(?)
WGATe			Window Gate. Defines a window where once the window start condition is met, gate trigger rises and capture begins. The capture ends once the window stop condition is met and the gate trigger falls.
CUSTom			Future Release.

Response

The Proteus unit will return EDGE | GATE | WEDGe | WGATe | CUSTom depending on the trigger type set for the active digitizer.

Example

Command `:DIG:TRIG:TYPE GATE`

Query `:DIG:TRIG:TYPE?`

10.31 :DIGitizer:TRIGger:CONDition{ GREater|SHORter}(?)

Description

This command will setup the valid timing condition, greater (or equal) or shorter, for time-related trigger events.

Parameters

Name	Type	Default	Description
GREater	Discrete	GREater	Time is greater or equal form valid trigger
SHORter			Time is shorter for valid trigger

Response

The Proteus unit will return GREa or SHOR depending on the present active channel setting.

Example

Command `:DIG:TRIG:COND SHOR`

Query `:DIG:TRIG:COND?`

10.32 :DIGitizer:TRIGger:SLOPe{ POS | NEG }(?)

Description

This command will set which trigger edge to trigger on when :TYPE is set to EDGE.

Parameters

Name	Type	Default	Description
POS	Discrete	POS	Sets the positive or rising edge as the valid trigger condition.
NEG	Discrete		Sets the negative or falling edge as the valid trigger condition.

Response

The Proteus unit will return POS, or NEG depending on the present active trigger slope setting.

Example

Command `:DIG:TRIG:SLOP POS`

Query `:DIG:TRIG:SLOP?`

10.33 :DIGitizer:TRIGger:WINDow:STARt { <threshold-level index (1/2)>, POSitive | NEGative }(?)

Description

This command will set the window start edge. The window start is defined by a combination of trigger level, 1 or 2, and trigger polarity, POS or NEG. Therefore, there are 4 possible combinations for defining the window start:

1. Level 2, POS
2. Level 2, NEG
3. Level 1, NEG
4. Level 1, POS

This is valid for the external trigger source only.

Parameters

Name	Type	Default	Description
<threshold-level index (1/2)>	Discrete		Set trigger LEVel 1 or 2 as the threshold level for the starting edge.
POSitive	Discrete		Set the polarity of the trigger signal. When positive trigger signal must cross from low to high.
NEGativ	Discrete		Set the polarity of the trigger signal. When negative trigger signal must cross from high to low.

Response

The Proteus unit will return trigger level 1, or 2 and POS, or NEG depending on the present window start edge trigger setting.

Example

Command `:DIG:TRIG:WIND:STAR 1,POS`

Query `:DIG:TRIG:WIND:STAR?`

10.34 :DIGitizer:TRIGger:WINDow:STOP { <thrshold-level index (1/2)>, POSitive | NEGative }(?)

Description

This command will set the window stop edge. The window stop is defined by a combination of trigger level, 1 or 2, and trigger polarity, POS or NEG. Therefore, there are 4 possible combinations for defining the window stop:

1. Level 2, POS
2. Level 2, NEG
3. Level 1, NEG
4. Level 1, POS

This is valid for the external trigger source only.

Parameters

Name	Type	Default	Description
<threshold-level index (1/2)>	Discrete		Set trigger level 1 or 2 as the threshold level for the stop edge.
POSitive	Discrete		Set the polarity of the trigger signal. When positive, trigger signal must cross the trigger level from low to high.
NEGativ	Discrete		Set the polarity of the trigger signal. When negative, trigger signal must cross the trigger level from high to low.

Response

The Proteus unit will return POS, or NEG depending on the present window stop edge trigger start setting.

Example

Command `:DIG:TRIG:WIND:STOP 1,POS`

Query `:DIG:TRIG:WIND:STOP?`

10.35 :DIGitizer:TRIGger:WIDTh<trigger_event_width>(?)

Description

Use this command to set or query the valid trigger width when trigger condition width is enabled. This setting is valid external trigger source and WEDGE trigger type only. Use the :DIG:TRIG:COND command to define this width as the maximum (SHORter) or minimum (GREater).

Parameters

Name	Range	Type	Default	Description
<trigger_event_width>	0, 0 to 15	Numeric	1e-06	Time reference for time-qualified trigger types. Resolution in 1/ADC_SCLK.

Response

The Proteus will return the present trigger width set for the current digitizer.

Example

Command `:DIG:TRIG:WIDT 100e-09`

Query `:DIG:TRIG:WIDT?`

10.36 :DIGitizer:TRIGger:HOLDoff< holdoff_time>(?)

Future Release

Description

Set the holdoff of the external trigger for the selected channel. Incoming trigger will be ignored during the holdoff period.

Parameters

Name	Range	Type	Default	Description
< holdoff >	external-trigger: from 0 to 16382 (14-bit ADC). Internal-trigger: only 0 (no holdoff	Numeric	0	Set the holdoff of the selected external-trigger of the selected channel.

Response

The Proteus will return the present external trigger holdoff value in units of seconds.

Example

Command `:DIG:TRIG:HOLD 10e-06`

Query `:DIG:TRIG:HOLD?`

10.37 :DIGitizer:TRIGger:DELay[:EXTernal]<delay_time>(?)

Description

Use this command to set the time delay of the external trigger of the digitizer. The resolution is in units of the digitizer's sampling clock:

- 10 sample clocks in case of dual-channels mode.
- 20 sample clocks in case of single-channel mode.

Parameters

Name	Range	Type	Default	Description
<delay_time>	0 to 6.0677e-5	Numeric	0	Set the time delay of the externa trigger of the digitizer.

Response

The Proteus will return the present delay time value in units of second.

Example

Command `:DIG:TRIG:DEL 1e-06`

Query `:DIG:TRIG:DEL:EXT?`

10.38 :DIGitizer:TRIGger:AWG:TDELay<task-trigger delay>(?)

Description

Set the time-delay of the task-trigger from the active AWG channel to the digitizer. For Task-trigger, delay can be set for each TASK list, not for each digitizer channel, so the :INST:CHAN command must be sent before using this command so the target task is selected. The currently selected Digitizer channel is not relevant for this command.

Parameters

Name	Range	Type	Default	Description
<task-trigger delay>	0 to 10s	Numeric		Set the time-delay of the task-trigger from the selected AWG channel to the digitizer.

Response

The Proteus will return the task-trigger delay.

Example

Command `:DIG:TRIG:AWG:TDEL 5.0`

Query `:DIG:TRIG:AWG:TDEL?`

10.39 :DIGitizer:PRETrigger< pre-trigger length in samples>(?)

Description

Set the position of the trigger inside the frame, or in other words, how many samples that arrive before the trigger that starts the frame, should be saved in the frame. In case of dual-channel mode, the length must be a multiple of 48 samples. In case of single-channel mode the length must be a multiply of 96 samples. Zero means no pre-trigger.

Note

The trigger will be ignored if not all the pre-trigger samples have been acquired.

Parameters

Name	Range	Type	Default	Description
<pre-trigger length in samples>	0 to size of frame	Integer	0	Set the position of the trigger inside the frame.

Response

The Proteus will return the pre-trigger length in samples.

Example

Command `:DIG:PRET 96`

Query `:DIG:PRET?`

10.40 :DIGitizer:DATA:TYPE< FRAMes | HEADers | BOTH >(?)

Description

This command sets the type of data that will be read from the digitizer memory. It is possible to select between reading only the frame data, only the frame header or both. If reading frames with headers, then each frame is followed by its header.

The “HEADER” field is divided to sub-fields of 32bit each. The content of the fields is:

Field 1 [31:0] – trigger location in the frame. Used for PRE-trigger feature.

Field 2 [63:32] – gate mode last address

Field 3 [75:64] – min ADC level in the current frame (12 bits)

Field 4 [87:76] – max ADC level in the current frame (12 bits)

Field 5 [151:88] – time stamp (64 bits)

Parameters

Name	Type	Default	Description
FRAMes	Discrete		Reads the frame data only
HEADers	Discrete		Reads the frame header only
BOTH	Discrete		Reads frame data followed by the header

Response

The Proteus will return the data type selected.

Example

Command `:DIG:DATA:TYPE FRAM`

Query `:DIG:DATA:TYPE?`

10.41 :DIGitizer:DATA:SELect < ALL | FRAMes | CHUNk>(?)

Description

Use this command to set what should be read from the digitizer captured data.

Parameters

Name	Type	Default	Description
ALL	Discrete		All frames should be read.
FRAMes	Discrete		One or more frames should be read.
CHUNk	Discrete		Chunk of data from specified frame (the :TYPE is ignored in this case) should be read.

Response

The Proteus will return the selected read type.

Example

Command `:DIG:DATA:SEL FRAM`

Query `:DIG:DATA:SEL?`

10.42 :DIGitizer:DATA:FRAMes <1st-frame>,<num-frames>(?)

Description

Use this command to select which of the captured frames to read when the :DIG:DATA:TYPE FRAM has been issued.

Parameters

Name	Range	Type	Default	Description
<1st-frame>	1 to num-frames	Integer	1	Set the first frame to transfer.
<num-frames>	1 to num-frames, -1 means to the last frame.	Integer	1	Set the number of frames to transfer.

Response

The Proteus will return the settings of the 1st-frame and the num-frames.

Example

Command `:DIG:DATA:FRAM 4,16`

Query `:DIG:DATA:FRAM?`

10.43 :DIGitizer:DATA:CHUNk <frame-no>,<offset in samples>,<read size in samples>(?)

Description

Use this command to a acquire data-chunk from the specified frame of the selected channel when the :DIG:DATA:TYPE CHUNK has been issued.

Parameters

Name	Range	Type	Default	Description
<frame-no>	1 to num-frames	Integer	1	Set the number of the frame to acquire.
< offset in samples >	0 to frame length	Integer	0	Set the offset in samples of the frame to acquire.
<read size in samples>	0 to chunk size	Integer	48	Set the number of samples to read.

Response

The Proteus will return the parameter settings:

#<binary-data header><binary-data block>

Example

Command `:DIG:DATA:CHUN 4,16,512`

Query `:DIG:DATA:CHUN?`

10.44 :DIGitizer:DATA:READ(?)

Description

Use this command to acquire the selected frames from the selected digitizer channel.

Response

The Proteus will return #<binary-data header><binary-data block>.

Name	Type	Description
<binary_data_header>	Discrete	The first digit in ASCII following the '#' character is the number of digits to follow. The following digits specify the length of the waveform data to be read in bytes.
< binary data_block >	Bytes	Sample values made of 16-bit unsigned int in Real mode, 32-bit unsigned int in the Complex mode, and 48-bit unsigned int for the Average mode.

Example

Query `:DIG:DATA:READ?`

10.45 :DIGitizer:DATA:SIZE(?)

Description

Query the size in bytes of the data that was selected for read.

Response

The Proteus will return the number of bytes of the data selected for read.

Example

Query `:DIG:DATA:size?`

10.46 :DIGitizer:DATA:FNAMe #<header><file-path as binary data>

Description

Use this command to set the file-path for the :STORe command.

Parameters

Name	Type	Description
<header>	Discrete	The first digit in ASCII following the '#' character is the number of digits to follow. The following digits specify the length of the target file full path name in ASCII.
< file-path as binary data >	String	Full path name for the file in ASCII coded as an unsigned short integer array.

Example

Command `:DIG:DATA:FNAM #212captured.dat`

10.47 :DIGitizer:DATA:STORe <offset>

Description

Use this command to store the specified data from the captured memory of the selected channel of the digitizer in the specified offset of the predefined file (no query).

Parameters

Name	Type	Description
<offset>	Integer	Offset in bytes inside the file.

Example

Command `:DIG:DATA:STOR 512`

10.48 :DIGitizer:DATA:FORMat { <U16 | F32 | F64}(?)

Description

Set the format of the data that will be sent to the user.

Parameters

Name	Type	Default	Description
< U16 >	Discrete	U16	Each 12-bits sample is contained in an uint16 (LSB is marker if marker-mode is enabled).
< F32>	Discrete		Each sample is converted to 32-bit floating-point value - marker is lost.
< F64>	Discrete		Each sample is converted to 64-bit floating-point value - marker is lost.

Response

The Proteus unit will return the data format.

Example

Command `:DIG:DATA:FORM F64`

Query `:DIG:DATA:FORM?`

10.49 :DIGitizer:LOOPback[:STATe]{ OFF|ON|0|1}(?)

Description

Enable/disable loopback state for the active channel.

Parameters

Name	Type	Default	Description
OFF	Discrete	OFF	Disable loopback state for the active channel.
ON	Discrete		Enable loopback state for the active channel.

Response

The Proteus will return the selected state ON or OFF.

Example

Command `:DIG:LOOP OFF`

Query `:DIG:LOOP?`

10.50 :DIGitizer:LOOPback:DELay< delay>(?)

Description

Set the time delay between the digitizer channel and the corresponding loopback generator channel. Values are from 8 to 2047, where each unit corresponds to a 16*DAC_SCLK_Period delay.

Parameters

Name	Type	Default	Description
<delay>	Integer	8	Set the time delay between the digitizer channel and the corresponding loopback generator channel.

Response

The Proteus will return the selected parameters.

Example

Command `:DIG:LOOP:DEL 512`

Query `:DIG:LOOP:DEL?`

10.51 :DIGitizer:LOOPback: SYNC

Description

No query. Use this command to initiate a sync trigger to synchronize all loopback channels.

Example

Command `:DIG:LOOP:SYNC`

10.52 :DIGitizer:LOOPback:IQRotation< scale>,<phase>(?)

Description

Use this command to set the scale and phase of the IQ rotation added to the output signal of the active generator channel. Scale can have values between 0 to 3, and phase can have values between 0 to 360.

Parameters

Name	Type	Default	Description
<scale>	Integer	1	Set the scale of the IQ rotation added to the output signal of the active generator channel.
<phase	Integer	0	Set the phase of the IQ rotation added to the output signal of the active generator channel.

Response

The Proteus will return the selected parameters separated by a comma.

Example

Command `:DIG:LOOP:IQR 2,180`

Query `:DIG:LOOP:IQR?`

10.53 :DIGitizer:LOOPback:OVERflow(?)

Description

Query if the configured operation resulted in an overflow and as a result the signal after processing was clipped. Response is 1 if signal was clipped or 0 if not clipped.

Response

The Proteus will return 1 if signal was clipped or 0 if not clipped.

Example

Query `:DIG:LOOP:OVER?`

10.54 :DIGitizer:PULSe[:DEFine]{<INTernal | EXTernal>,<FIXed | GATed>,<window_width(?)

Description

Use this command to set the pulse-counter parameters of the digitizer.

Parameters

Name	Type	Default	Description
INTernal|EXTernal	Discrete	INT	Define pulse-counter trigger-source parameters.
FIXed | GATed	Discrete	FIX	Define pulse-counter window-type parameters.
window_width	Numeric	0.1	Define pulse-counter window-width in seconds parameter. Range from 12.5ns to 15s.

Response

The Proteus will return the selected parameters.

Example

Command `:DIG:PULS INT,GAT,4.5`

Query `:DIG:PULS?`

10.55 :DIGitizer:PULSe:COUNt?

Description

Query only. It reads the current contents of counters attached to each channel.

Response

The Proteus unit will return the contents of <counter 1>,<counter 2>.

Example

Query `:DIG:PULS:COUN?`

11 Digital Signal Processing Commands

11.1 Introduction DSP

This group is used to control the signal processing path of the data acquired by the digitizer and handle the decision-based waveform generation. Using these commands, you can define which processing blocks to use, what data to store, and which waveform to generate based on the result from various calculations on the acquired signal.

The Proteus FPGA has real time digital signal processing capabilities so the user can reduce the data transfer to the control PC and perform the data processing at a much higher rate on the FPGA. This can significantly reduce the application run time and feedback control latency.

These commands are relevant only for those Proteus instruments equipped with the AWT and the FPGA PROG option.

Configuring the signal processing chain

The DSP commands give the user the ability to enable, disable and configure various blocks along the signal processing chain. When approaching the task of configuring the blocks it is necessary to understand the data flow and the dependencies between the blocks. In the figure below is a block diagram illustrating the various blocks and their connectivity. The following paragraphs will describe the overall approach that needs to be taken when configuring the DSP blocks.

The first selection that must be made is whether to route the signal through the complex path or real path.

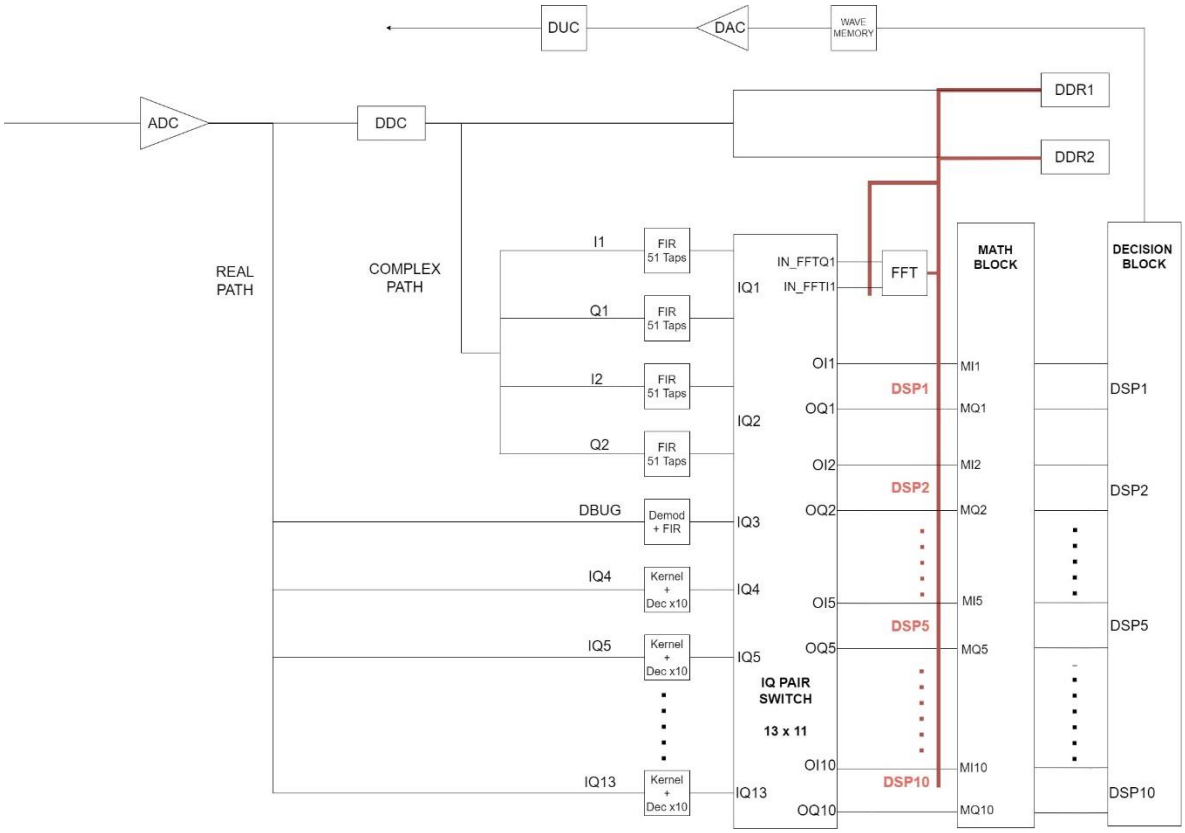

Figure 11-1 Proteus Digital Signal Processing Block Diagram

Complex Path

When a signal is received at the input of the digitizer it can either be routed to the COMPLEX PATH or the REAL PATH. In the complex path, after being sampled, the signal is down converted and demodulated using the digital down converter (DDC) integrated in the Analog to Digital Converter. There are two DDC entities in the ADC chip, one for each of the two channels, DDC1 for channel 1 and DDC2 for channels 2, or when working in DDC bind mode both are configured to work with channel 1 thus enabling to de-multiplex two frequencies on one signal.

> **Note**
> The complex path is only available in the dual channel mode of the digitizer.

Each DDC outputs an I and Q signal (I1, Q1, I2, Q2) that can then be filtered using the FIR block. The FIR block consists of a 51 tap FIR filter that can be configured for each signal.

Real Path

When selecting the real path, the DDC is bypassed and the sampled data is routed to the DSP IQ demodulation block. The IQ demodulation block offers 10 different demodulation entities so it can handle up to 10 frequencies multiplexed on one signal. For each entity, user defines a kernel for the I and Q pair. For an example on how to create the Kernel you can refer to the teproteus_functions.py file on the Tabor Github page (https://github.com/Tabor-Electronics) or contact Tabor online support. After the kernel the signal is decimated by 10. In addition, there is a block (DBUG) that implements a sine and cosine Kernel with a 51 tap configurable filter that can be used for debugging purposes.

Data Storage

Once the user has selected the data path, complex or real, it is necessary to configure which data should be stored on the Proteus memory. The Proteus on-board memory consists of 2xDDR4 SODIMMs. User can select to store different data at different points along the data path according to the table below.

Table 11-1 Possible Data Storage Configurations

		Complex Mode		Real Mode	
		DDR1	**DDR2**	**DDR1**	**DDR2**
Store	**Direct**	Stores digitizer channel 1 sampled IQ data after DDC	Stores digitizer channel 2 sampled IQ data after DDC	Stores digitizer channel 1 sampled data	Stores digitizer channel 2 sampled data
	DSP	N/A	N/A	Stores DSP1-5 IQ data after the demodulation and decimation block	Stores DSP6-10 and debug IQ data after the demodulation and decimation block
	FFT	Stores the output (result) of the FFT performed on the input selected in the :FFT:IN command .	Stores DDC1 IQ (I1,Q1) data and DDC2 IQ (I2,Q2) data after the configurable FIR filter in the FIR block. These are then routed as DSP1 and DSP6.	Stores the output (result) of the FFT performed on the input selected in the :FFT:IN command . In this mode, DBUG	Stores DSP6-10 and debug IQ data after the demodulation and decimation block

FFT and MATH Operations

After selecting what data to store, the user can configure if any mathematical operations are to be done on the sampled data. There are three modules, the first is for scaling and shifting, the second is for cross correlation and third is for averaging. The idea is that every I and Q input can be scaled and shifted via a constant provided by the user. In addition, cross correlation can be done after the scaling and shifting. Finally, a rolling average can be performed on any of the parameters including the cross correlation.

Decision

Finally, the user must configure how a decision is made and the outcome. In the decision block there are 10 decision modules DEC1 to DEC 10 which correspond to DSP1 to DSP10. Thus, when in complex mode only DEC1 and DEC6 are relevant and in real mode all ten are relevant.

The decision stage is used to both compress the dataflow (e.g. converting a time-series of numbers into a single 0 or 1 for readout output) and to generate the required trigger signals for feedback. Thus, decision can be made based on a threshold logic or by using a state classifier or state vector machine (SVM).

The threshold module simply compares a given input stream with a value specified by the user to output a 0 or 1 if the value is less or greater than that supplied. The resulting 1-bit number can be

used to conditionally jump between two tasks later in the feedback loop. Similarly, in the SVM module the user programs the coefficients and the SVM outputs a 3 bit number with the value of the discrete state that corresponds to the given input bitstream. The user can define which segment to generate for each of the 8 possible states the SVM can output. The decision module can be seen in the figure below.

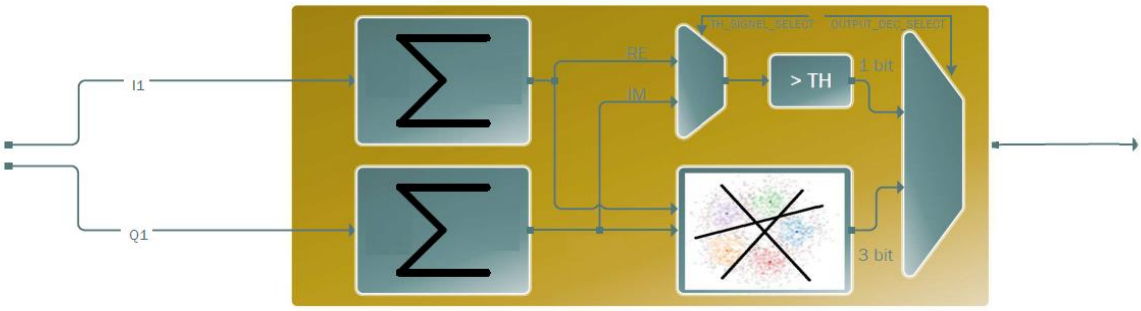

Figure 11-2 Decision Block Module

Notes when using decision blocks

- Frame size in digitizer must be larger than decision frame size.
- When in REAL mode since kernel is limited to 1024 samples (after decimation) then digitizer frame size is limited to 1024 samples.
- When in complex mode there is no limit on digitizer frame size however since decision frame size is limited to 1024 samples only the first 1024 samples of the digitizer will be used for decision.

11.2 :DSP:STORe{ DIRect | DSP | FFTOut }(?)

Description

Use this command to set or query what data will be stored in a DDR. Note that the valid options are dependent on the DDC:MODE selected in the digitizer command section. See chapter 10.7 :DIGitizer:DDC:MODE{REAL|COMPlex}(?). Refer to Table 11-1 Possible Data Storage Configurations for details.

Complex Mode

Refer to 10.7 :DIGitizer:DDC:MODE{REAL|COMPlex}(?).

Parameters

Name	Type	Default	Description
Direct	Discrete	DIR	Stores the raw input data after the ADC down conversion and demodulation. Data is saved as IQ data before filtering. IQ pair sourced in channel 1 is stored on DDR 1 and IQ pair sourced in channel 2 is stored in DDR2.
DSP			Same as FFTOut.
FFTOut			Stores the IQ data after the FIR filter. DDR 1 stores the output of the FFT performed on the input selected by the :FFT:IN command. DDR2 stores two IQ pairs, IQ1 sourced from channel 1 and IQ2 sourced from channel 2. When :DDC:BIND is set to ON then both IQ pairs are sourced from channel 1. These are then routed as DSP1 and DSP6. Pay attention that in Complex mode when selecting DSP, the frame size as defined in the ADC must be at least 4104 samples.

Real Mode

Refer to 10.7 :DIGitizer:DDC:MODE{REAL|COMPlex}(?).

Pay attention that in real mode maximum frame size as defined in the ADC should not exceed 12384 samples, this contains 12 IQ pair entities, 10DSP, Debug and FFT.

Parameters

Name	Type	Default	Description
Direct	Discrete	DIR1	This stores the raw input data before the FPGA down conversion and demodulation. Data is saved as raw sample data.
DSP			This stores the IQ data after the FIR filter. In this mode there are ten IQ pairs, DSP1-10. DSP1-5 are sourced from ADC channel 1 and DSP 6-10 are sourced from channel 2. When DDC bind is set to ON, DSP1-10 are sourced from channel 1. DDR1 stores time domain samples of DSP1-5. DDR2 stores time domain samples of DSP 6-10 as well as samples of the DBUG channel. The DBUG path performs a traditional IQ demodulation which includes an extra 61 system clocks delay relative to the DSPn path.
FFTOut			In this mode DDR2 stores time domain samples of DSP 6-10 as well as samples of the DEBUG channel. DDR1 stores the output (result) of the FFT performed on the output of the DBUG path. The time domain input signal to the FFT is always the DBUG path. When FFT is selected the frame size as defined in the ADC must be exactly 12384 samples. When FFTOut is selected the frame size as defined in the ADC must be exactly 12384 samples.

Response

The Proteus unit will return DIR, DSP, or FFTO depending on the current parameter selected to be stored.

Example

Command `:DSP:STOR DIR`

Query `:DSP:STOR?`

11.3 :DSP:IQDemod:SELect{ DBUG | IQ4 | IQ5 | IQ6 | IQ7 | IQ8 | IQ9 | IQ10 | IQ11 | IQ12 | IQ13 }(?)

Description

When in REAL mode this command will select which IQ pair demodulation block to configure. DSP1 corresponds to IQ4 up to DSP10 which corresponds to IQ13.

Parameters

Name	Type	Default	Description
DBUG	Discrete	DBUG	Selects the DBUG block
IQ4	Discrete		Selects the IQ4 block
IQ5	Discrete		Selects the IQ5 block
IQ6	Discrete		Selects the IQ6 block
IQ7	Discrete		Selects the IQ7 block

Name	Type	Default	Description
IQ8	Discrete		Selects the IQ8 block
IQ9	Discrete		Selects the IQ9 block
IQ10	Discrete		Selects the IQ10 block
IQ11	Discrete		Selects the IQ11 block
IQ12	Discrete		Selects the IQ12 block
IQ13	Discrete		Selects the IQ13 block

Response

The Proteus unit will return DBUG, IQ4,…IQ13 depending on which block was selected.

Example

Command `:DSP:IQD:SEL IQ4`

Query `:DSP:IQD:SEL?`

11.4 :DSP:IQDemod:KERnel:COEFficient <sample number>,<real>,<imaginary>(?)

Description

Use this command to write the real and imaginary parts of the specified sample in the kernel of the selected IQ pair. Range 10240 samples, each sample is 12 bit signed FIX 12_11 (11 bit for fractional), real and imaginary take values between -1 to 1.

Parameters

Name	Range	Type	Default	Description
<sample number>	1 to 10240	Integer		The specified sample to edit
< real >	-1 to 1	Float		Value of the real number
<imaginary>	-1 to 1	Float		Value of the imaginary number

Response

The Proteus will return the real and imaginary values of the specified sample number.

Example

Command `:DSP:IQD:KER:COEF 5,0.5,-0.5`

Query `:DSP:IQD:KER:COEF? 5`

11.5 :DSP:IQDemod:KERnel:DATA#<header><binary_block>(?)

Description

Use this command to write or read the I and Q kernel data of the selected IQ pair. The kernel data must consist of 10240 samples, where each sample holds an I and Q value.

Parameters

Name	Type	Description
<header>	Integer	Contains information on the size of the binary block that contains kernel data.
< binary block >	binary	Block of binary data that contains kernel data, as explained above. Each sample is 4 bytes, 12 bit for I and 12 bit for Q. data arrangement is 12 bit signed FIX 12_11,

Example

Command `:DSP:IQD:KER:DATA #3408`

Query `:DSP:IQD:KER:DATA?`

11.6 :DSP:FIR:SELect{ I1 | Q1 |I2 | Q2 |DBUGI | DBUGQ}(?)

Description

This command will select which FIR block to configure. When mode is Complex the complex data path IQ blocks are operational (I1, I2, Q1, Q2) and in REAL mode the DBUG IQ blocks are operational (DBUGI, DBUGQ).

Parameters

Name	Type	Default	Description
I1	Discrete	I1	Selects the I1 FIR
Q1	Discrete		Selects the Q1 FIR
I2	Discrete		Selects the I2 FIR
Q2	Discrete		Selects the Q2 FIR
DBUG1	Discrete		Selects the DBUGI FIR
DBUG2	Discrete		Selects the DBUGQ FIR

Response

The Proteus unit will return I1, Q1, I2, Q2, DBUG1 or DBUG2 according to the selected FIR block

Example

Command `:DSP:FIR:SEL Q2`

Query `:DSP:FIR:SEL?`

11.7 :DSP:FIR:BYPass{OFF|ON|0|1}(?)

Description

Use this command in complex mode to set whether to bypass the FIR block.

Parameters

Range	Type	Default	Description
0-1	Discrete	0	**0** – FIR bypass is off. **1** – FIR bypass is ON.

Response

The Proteus will return 1 if the FIR block is bypassed, or 0 if it is not.

Example

Command `:DSP:FIR:BYP ON`

Query `:DSP:FIR:BYP?`

11.8 :DSP:FIR:LENGth(?)

Description

Query only. Queries the FIR (Finite Impulse Response) length (number of taps).

Response

The Proteus unit will return the length of the FIR applied to the signal acquired.

Example

Query `:DSP:FIR:LENG?`

11.9 :DSP:FIR: COEFficient <tap number>,<the value of the specified tap>(?)

Description

Use this command to set the tap coefficient value by sending the index value of the desired coefficient and its new value. Value is specified between -1 to 1 (FIX 24_23). The FIR has 51 taps. Query returns the value of the coefficient specified by the index.

Parameters

Name	Range	Type	Default	Description
<tap number>	1 to 51	Integer		Set the index value of the coefficient to edit.
< value of specified tap >	-1 to 1	Float		Set the tap value of the selected tap.

Response

The Proteus will return the value of the specified tap.

Example

Command `:DSP:FIR:COEF 1,0.5`

Query `:DSP:FIR:COEF? 1`

11.10 :DSP:FIR:DATA#<header><binary_block>(?)

Description

Use this command to write or read the taps of the FIR as binary data, the FIR has 51 taps each tap is FIX 24_23.

Parameters

Name	Type	Description
<header>	Integer	Contains information on the size of the binary block that contains taps data.
< binary block >	binary	Block of binary data that contains taps data, as explained above. Each tap value is specified between -1 to 1 (FIX 24_23)

Example

Command `:DSP:FIR:DATA #3408`

`:DSP:FIR:DATA? 5`

11.11 :DSP:FFT:INPut{ IQ1 | IQ2 |DBUG }(?)

Description

This command will select the input of the FFT. In case of real mode it is DBUG ; In case of complex mode it is IQ1 which is sourced at the ADC channel 1 or IQ2 which is sourced at ADC channel 2. When :DDC:BIND is set to ON then both IQ pairs are sourced from channel 1.

When FFT is used the frame size as defined in the ADC must be exactly 12384 samples and the FFT length is set to 1024.

Parameters

Name	Type	Default	Description
IQ1	Discrete	IQ1	Select the input of the FFT to be IQ1.
IQ2	Discrete		Select the input of the FFT to be IQ2.
DBUG	Discrete		Select the input of the FFT to be DBUG

Response

The Proteus unit will return IQ1, IQ1or DBUG depending on the selected FFT input.

Example

Command `:DSP:FFT:INP IQ1`

Query `:DSP:FFT:INP?`

11.12 :DSP:MATH:OPERation{ MI1 | MQ1 | MI2 | MQ2 | MI3 | MQ3 |MI4 |MQ4 | MI5 |MQ5 | MI6 |MQ6 | MI7 | MQ7 | MI8|MQ8 | MI9 |MQ9 | MI10 |MQ10 ,<SCALe>,<OFFSet> }(?)

Description

Set the scale and offset of the specified affine transformation in the math-block for the selected parameter. Scale value can be between -64 to 63, offset value between -8192 to 8191. MIX or MQX corresponds to the appropriate DSPIX and DSPQX. Please note that when in complex mode

only MI1, MQ1 and MI3, MQ3 are available. MIX or MQX corresponds to the appropriate DSPIX and DSPQX.

Parameters

Name	Range	Type	Default	Description
MI1		Discrete	MI1	Set MI1 as the parameter for the math operation
MQ1		Discrete		Set MQ1 as the parameter for the math operation
MI2		Discrete		Set MI2 as the parameter for the math operation
MQ2		Discrete		Set MQ2 as the parameter for the math operation
…				
MI10		Discrete		Set MI10 as the parameter for the math operation
MQ10		Discrete		Set MQ10 as the parameter for the math operation
<scale>	-64 to 63	Integer	1	Set the scaling factor that will be applied to the selected parameter
<offset>	-8192 to 8191	Integer	0	Set the offset that will be applied to the selected parameter

Response

The Proteus unit will return the selected parameter, the scale factor and offset value.

Example

Command `:DSP:MATH:OPER MQ2,5,1`

Query `:DSP:MATH:OPER?`

11.13 :DSP:MATH:OPERation:CLIP(?)

Description

Query only. Query if the configured operation resulted in an overflow and as a result signal was clipped.

Response

The Proteus unit will return 1 if the signal was clipped and 0 if it was not clipped.

Example

Query `:DSP:MATH:OPER:CLIP?`

11.14 :DSP:MATH:XCORrelation:LENGth<N>(?)

Description

Use this command to set the length of the cross correlation in samples. The length can range from 1 to 1024 samples.

Parameters

Name	Range	Type	Default	Description
<N>	1 to 1024	integer	1024	Set the length of the cross correlation in samples.

Response

The Proteus will return the length of the cross correlation in samples.

Example

Command `:DSP:MATH:XCOR:LENG 512`

Query `:DSP:MATH:XCOR:LENG?`

11.15 :DSP:MATH:XCORrelation:SIGNal{ <MI1 | MQ1 | MI2 | MQ2 | MI3 | MQ3 |MI4 |MQ4 | MI5 |MQ5 | MI6 |MQ6 | MI7 | MQ7 | MI8|MQ8 | MI9 |MQ9 | MI10 |MQ10 }(?)

Description

This command will select the two signals on which to perform the cross correlation.

Note that when in complex mode only MI1, MQ1 and MI3, MQ3 are available.

Parameters

Name	Type	Default	Description
MI1 | MQ1 | MI2 | MQ2 | MI3 | MQ3 |MI4 |MQ4 | MI5 |MQ5 | MI6 |MQ6 | MI7 | MQ7 | MI8|MQ8 | MI9 |MQ9 | MI10 |MQ10	Discrete	MI1	Select the first signal for cross correlation
MI1 | MQ1 | MI2 | MQ2 | MI3 | MQ3 |MI4 |MQ4 | MI5 |MQ5 | MI6 |MQ6 | MI7 | MQ7 | MI8|MQ8 | MI9 |MQ9 | MI10 |MQ10	Discrete	MQ1	Select the second signal for cross correlation

Response

The Proteus unit will return the two signals for which the cross correlation is calculated as specified by the command.

Example

Command `:DSP:MATH:XCOR:SIGN MI2,MI3`

Query `:DSP:MATH:XCOR:SIGN?`

11.16 :DSP:MATH:RAVG { MI1 | MQ1 | MI2 | MQ2 | MI3 | MQ3 |MI4 |MQ4 | MI5 |MQ5 | MI6 |MQ6 | MI7 | MQ7 | MI8|MQ8 | MI9 |MQ9 | MI10 |MQ10 |XC,<N> }(?)

Description

This command will select the signal on which to perform the rolling average on as well as the window size of the rolling average. Set the window size of the rolling average in values of 2^n where n=0,1…15. E.g. MI1 means n=1.

Parameters

Name	Type	Default	Description
MI1	Discrete	0	Select the MI1 signal to perform the rolling average on.
MQ1	Discrete		Select the MQ1 signal to perform the rolling average on.
…			
MI10	Discrete		Select the MI10 signal to perform the rolling average on.
MQ10	Discrete		Select the MQ10 signal to perform the rolling average on.
XC	Discrete		Select the XC (cross correlation) calculation to perform the rolling average on.
<N>	Integer		Set the window size (number of samples) of the rolling average in values of 2^n where n=0,1…15.

Response

The Proteus unit will return the signal selected for the rolling average and the window size.

Example

Command `:DSP:MATH:RAVG MQ2,8`

Query `:DSP:MATH:RAVG?`

11.17 :DSP:DECision[:FEEDback]:MAPping{ <awg channel number>,DEC1 | DEC2| DEC3 | DEC4 | DEC5 | DEC6 | DEC7 | DEC8 | DEC9 | DEC10 | XC } (?)

Description

This command will select which decision block affects the specified channel. Each decision block holds the result of the corresponding FPGA DSP block, so that DEC1 to DEC10 correspond to DSP 1 to 10. Note that when in complex mode only DEC1, DEC6 and XC are relevant. To be used in conjunction with the task destination parameter TASK:COMPoser[:DEFine]:DESTination DSIG.

When using decision blocks:

1. Frame size in digitizer must be larger than decision frame size.
2. When in REAL mode, since kernel is limited to 1024 samples (after decimation), then the digitizer frame size is limited to 1024 samples.
3. When in complex mode there is no limit on digitizer frame size. However, since decision frame size is limited to 1024 samples only the first 1024 samples of the digitizer will be used for decision.

Parameters

Name	Range	Type	Default	Description
<awg channel number>	1 to number of generator channels	Integer	I1	Set the AWG channel that will be affected by the decision.
DEC1		Discrete		Set the decision result that the AWG channel will be waiting for to be DEC1.
…				
DEC10		Discrete		Set the decision result that the AWG channel will be waiting for to be DEC10.
XC		Discrete		Set the decision result that the AWG channel will be waiting for to be XC (cross correlation).

Response

The Proteus unit will return the decision result that the specified AWG channel will be waiting for.

Example

Command `:DSP:DEC:MAP 2,DEC2`

Query `:DSP:DEC:MAP? 2`

11.18 :DSP:DECision[:FEEDback]:CONDition{<awg-channel number>, S1 | S2 |S3 | S4 | S5 | S6 | S7 | S8, <segment number>}(?)

Description

This command will associate the state Sx of the decision block with the specified segment of the specified AWG channel. This is used in conjunction with the task destination parameter DSP. Whenever the Task destination is set to be DSP, the segment generated by that task is the one associated with the relevant state as set by this command. For example DSP:DEC:COND 1,S1,3 means that when the destination in channel 1 task is DSP, then segment 3 will be generated if the result of the decision is S1. Note that a segment number must be defined for all states.

Parameters

Name	Range	Type	Default	Description
<awg-channel number>	1 to number of generator channels	Integer		Set the AWG channel that will be affected by the decision.
S1		Discrete		Select S1 as the state of the decision block to which the segment number will be associated to.
S2		Discrete		Select S2 as the state of the decision block to which the segment number will be associated to.
S3		Discrete		Select S3 as the state of the decision block to which the segment number will be associated to.
S4		Discrete		Select S4 as the state of the decision block to which the segment number will be associated to.
S5		Discrete		Select S5 as the state of the decision block to which the segment number will be associated to.
S6		Discrete		Select S6 as the state of the decision block to which the segment number will be associated to.
S7		Discrete		Select S7 as the state of the decision block to which the segment number will be associated to.
S8		Discrete		Select S8 as the state of the decision block to which the segment number will be associated to.
<segment number>	1 to last defined segment	Integer		Select the segment number that will be associated with the selected channel and state.

Response

The Proteus unit will return the segment number associated with the specified AWG channel and state.

Example

Command `:DSP:DEC:COND 2,S2,3`

Query `:DSP:DEC:COND? 2,S2`

11.19 :DSP:DECision:FRAMe<the frame size for the calculation>(?)

Description

Use this command to set the frame size of the calculation, range from 2 to 1024 samples.

Parameters

Name	Range	Type	Default	Description
< the frame size for the calculation >	2 to 1024	Integer	1024	Set the frame size for the calculation, range from 2 to 1024 samples.

Response

The Proteus will return the length of the frame size for the calculation in samples

Example

Command `:DSP:DEC:FRAM 512`

Query `:DSP:DEC:FRAM?`

11.20 :DSP:DECision:IQPath:SELect { DSP1 | DSP2 | DSP3 | DSP4| DSP5| DSP6| DSP7| DSP8| DSP9| DSP10 }(?)

Description

This command will set which input path to configure. Note that when in complex mode only DSP1 and DSP3 are relevant.

Parameters

Name	Type	Default	Description
DSP1	Discrete	DSP1	Set DSP1 as the input path to configure.
…			
DSP10	Discrete		Set DSP10 as the input path to configure.

Response

The Proteus unit will return the selected DSP.

Example

Command `:DSP:DEC:IQP:SEL DSP2`

Query `:DSP:DEC:IQP:SEL?`

11.21 :DSP:DECision:IQPath:OUTPut{ THR | SVM }(?)

Description

This command will set the output of the selected IQpath. The result of the output is described using 3 bits. When set to threshold the output is 0 or 1 and when set to SVM the output it is 0 to 7.

Parameters

Name	Type	Default	Description
THR	Discrete	THR	Set the threshold as the output of the selected IQpath.
SVM	Discrete		Set the SVM as the output of the selected IQpath.

Response

The Proteus unit will return the selected output.

Example

Command `:DSP:DEC:IQP:OUTP THR`

Query `:DSP:DEC:IQP:OUTP?`

11.22 :DSP:DECision:IQPath:THReshold:LEVel { <N> }(?)

Description

Use this command to set the threshold level of the decision block (between -2^23 and 2^23 -1)

Parameters

Name	Range	Type	Default	Description
<N>	-2^23 and 2^23 -1	Integer	0	Set the threshold level of the decision block

Response

The Proteus will return the threshold level

Example

Command `:DSP:DEC:IQP:THR:LEV 512`

Query `:DSP:DEC:IQP:THR:LEV?`

11.23 :DSP:DECision:IQPath:THReshold:INPut { I | Q }(?)

Description

This command will select the input for the threshold decision.

Parameters

Name	Type	Default	Description
I	Discrete	I	Set I as the input for the threshold decision
Q	Discrete		Set Q as the input for the threshold decision

Response

The Proteus unit will return the selected input.

Example

Command `:DSP:DEC:IQP:THR:INP I`

Query `:DSP:DEC:IQP:THR:INP?`

11.24 :DSP:DECision:IQPath:LINE{ 1 | 2 | 3, <slope>, <y-intercept>}(?)

Description

This command will set the slope and y intercept of the line equation (1, 2, 3) of the selected IQ Path. Slope value from -256 to 255, y-intercept value from -128 to 127

Parameters

Name	Type	Default	Description
1	Discrete	1	Line equation 1.
2	Discrete		Line equation 2.
3	Discrete		Line equation 3.
<slope>	integer	1	Slope
<y-intercept>	Integer	0	Y-intercept

Response

The Proteus unit will return the selected input.

Example

Command `:DSP:DEC:IQP:LINE 1,0.7,3`

Query `:DSP:DEC:IQP:LINE? 1`

11.25 :DSP:DECision:IQPath:CLIP(?)

Description

Query only. Query if the configured operation resulted in an overflow and as a result signal was clipped.

Response

The Proteus unit will return 1 if the signal was clipped and 0 if it was not clipped.

Example

Query `:DSP:DEC:IQP:CLIP?`

11.26 :DSP:DECision:XCORrelation: THReshold { <N> }(?)

Description

Use this command to set the threshold value of the cross correlation.

Parameters

Name	Range	Type	Default	Description
<N>	-2^23 and 2^23 -1	integer	0	Set the threshold level of the cross correlation

Response

The Proteus will return the parameter settings the length of the cross correlation in samples

Example

Command `:DSP:DEC:XCOR:THR 512`

Query `:DSP:DEC:XCOR:THR?`

11.27 :DSP:DECision:XCORrelation:CLIP(?)

Description

Query only. Query if the configured operation resulted in an overflow and as a result signal was clipped.

Response

The Proteus unit will return 1 if the signal was clipped and 0 if it was not clipped.

Example

Query `:DSP:DEC:XCOR:CLIP?`

12 System Commands

The system-related commands are not related directly to waveform generation but are an important part of operating the Proteus unit. Use these commands to reset the instrument and query its system settings.

12.1 :SYSTem:LOG[:VERBose] {0|1|2|3|4|5|6}(?)

Description

This command will set the logger verbose level (0: minimal, 6: maximal)

Parameters

Range	Type	Default	Description
0 to 6	Discrete	4	Select the logger verbose level. 0 – (Undefined) 1 – Critical 2 – Error 3 – Warning 4 – Info 5 – Debug1 6 – Debug2

Response

The Proteus will return the logger verbose level 0 to 6.

Example

Command `:SYST:LOG 3`

Query `:SYST:LOG?`

12.2 :SYSTem:ERRor?

Description

Query only. This query will interrogate the Proteus unit for programming errors. It pops the last error from the error-queue (returns "<err-code>,<description>"). When the error-queue is empty, the Proteus unit will return just the termination character (an empty string).

Response

The Proteus unit will return the error code. Error messages are listed below.

Example

Query `:SYST:ERR?`

12.2.1 Error list

SCPI Errors

201, unspecified error

202, missing parameter in scpi

203, invalid suffix in scpi
204, data out of range in scpi
205, invalid data type in scpi
206, illegal parameter in scpi
207, invalid separator in scpi
208, syntax error in scpi
209, illegal/unknown scpi

Operational Errors

210, not implemented
211, assertion failed
212, validation failure
213, operation was aborted
214, operation was rejected
215, operation was canceled
216, unfinished operation
217, write buffer overflow
218, data not applicable
219, invalid data size
220, null pointer
221, illegal argument
222, already exists
223, settings conflict
224, invalid handle
225, allocation error
226, DeFrag needed
227, timeout elapsed
228, busy
229, exception occurred

File-System Errors

230, bad path
231, file does not exist
232, open file error
233, bad file size
234, file read error
235, file write error
236, file save error
237, file delete error
238, file compress error
239, file extract error

240, make directory error

241, remove directory error

12.3 :SYSTem:INFormation:CALibration?

Description

Query only. Query the calibration date.

Response

The Proteus unit will return the last calibration date in the format "yyyy-mm-dd hh:MM", e.g., 2020-12-20 12:30.

Example

Query `:SYST:INF:CAL?`

12.4 :SYSTem:INFormation:MODel?

Description

Query only. This query will interrogate the instrument for its model ID.

Response

The generator will return its model number, e.g., P25812B.

Example

Query `:SYST:INF:MOD?`

12.5 :SYSTem:INFormation:SERial?

Description

Query only. This query will interrogate the instrument for its serial number.

Response

The generator will return its serial number in a format similar to the following: 2xxxxx.

Example

Query `:SYST:INF:SER?`

12.6 :SYSTem:INFormation:HARDware?

Description

Query only. This query will interrogate the instrument for its hardware revision level. The hardware revision includes PCB revision, FPGA revision and FPLD revision. It is programmed to a secure location in the flash memory and cannot be modified by the user.

Response

The generator will return its hardware revisions in the form "<AB-<AM>-DB>-<DM>-<ADC>" where:

AB – Analog Base version.
AM – Analog Mezzanine version.
DB – Digital Base version.
DM – Digital Mezzanine version.
ADC – ADC board version.

Example

Query `:SYST:INF:HARD?`

12.7 :SYSTem:INFormation:FPGA:VERsion?

Description

Query only. Query the FPGA FW version.

Response

The Proteus unit will return the FPGA FW version x.yyy.z.

Example

Query `:SYST:INF:FPGA:VER?`

12.8 :SYSTem:INFormation:FPGA:DATE?

Description

Query only. Query the FPGA FW build date.

Response

The Proteus unit will return the FPGA FW build date.

Example

Query `:SYST:INF:FPGA:DATE?`

12.9 :SYSTem:INFormation:FIRMware:VERsion?

Description

Query only. Query the control PC DLL version.

Response

The Proteus unit will return the control PC DLL version.

Example

Query `:SYST:INF:FIRM:VERS?`

12.10 SYSTem:INFormation:FIRMware:DATE?

Description

Query only. Query the installed Proteus DLL build date.

Response

The Proteus unit will return the Proteus DLL build date.

Example

Query `:SYST:INF:FIRM:DATE?`

12.11 :SYSTem:INFormation:DAC?

Description

Query Only. Query the DAC mode. Returns M0 for 16-bit width and M1 for 8-bit width.

Response

The Proteus unit will return the DAC mode, M0 or M1.

Example

Query `:SYST:INF:DAC?`

12.12 :SYSTem:INFormation:SLOT?

Description

Query the PXIe slot-number that the second slot of the instrument occupies in the chassis. (The instrument may occupy between 2 and 3 slots.)

Response

The Proteus unit will return the slot-number of the second slot the instrument occupies in the chassis.

Example

Query `:SYST:INF:SLOT?`

12.13 SYSTem:INFormation:SCPI[:VERSion]?

Description

Query only. This query will interrogate the set of SCPI command version.

Response

The Proteus unit will return the set of SCPI command version.

Example

Query `:SYST:INF:SCPI?`

12.14 :SYSTem[:MEASure]:TEMPerature?

Description

Query the internal temperature of the instrument.

Response

The Proteus unit will return the internal temperature (°C).

Example

Query `:SYST:TEMP?`

12.15 :SYSTem[:MEASure]:HTPeak?

Description

Query the highest recorded temperature of the instrument.

Response

The Proteus unit will return the highest recorded temperature of the instrument (°C).

Example

Query `:SYST:HTP?`

12.16 :SYSTem[:MEASure]:LTPeak?

Description

Query the lowest recorded temperature of the instrument.

Response

The Proteus unit will return the lowest recorded temperature of the instrument (°C).

Example

Query `:SYST:LTP?`

12.17 :SYSTem[:MEASure]:VINTernal?

Description

Query the internal Vcc of the instrument. Use this for troubleshooting the instrument.

Response

The Proteus unit will return the internal Vcc of the instrument (V).

Example

Query `:SYST:VINT?`

12.18 :SYSTem[:MEASure]:VAUXiliary?

Description

Query the auxiliary Vcc of the instrument.

Response

The Proteus unit will return the auxiliary Vcc of the instrument (V).

Example

Query `:SYST:VAUX?`

12.19 :SYSTem:FILE:CATalog?

Description

Query the file catalog in the folder "C:\Users\<user-name>\Documents\ProteusFiles". This is a list of the system files, such as waveform data or setup files.

There are several SCPI commands that receive filename as argument:

- :MARKer:FILE:[:NAME]
- :TASK:FILE[:NAME]
- :SCEN:FILE[:NAME]
- :TRACe:SEGMents:FILE[:NAME]
- :TRACe:FILE[:NAME]
- :DIG:DATA:FNAMe
- :SYST:FILE[:NAME]

All of them can receive either an absolute full-path, relative-path or just file-name.

If the file-name is not an absolute full-path, then it is assumed that it is inside the folder "C:\Users\<user-name>\Documents\ProteusFiles".

Response

The Proteus unit will return the file catalog. The response to :SYSTem:FILE:CATalog? is a comma-separated list of the file names inside that folder.

Example

Query **:SYST:FILE:CAT?**

12.20 :SYSTem:FILE[:NAMe]{< #<header><binary-block>}

Description

Specifies the system file. This is used to transfer any binary file from your PC to the instrument. The file name is defined as an IEEE-488.2 binary block with the name codified in 8-bit unsigned integers (bytes) with the ASCII codes containing the full path to the source file.

Parameters

Name	Type	Description
<header>	<discrete>	The first digit in ASCII following the '#' character is the number of digits to follow. The following digits specify the length of the target file full path name in ASCII.
<binary_block>	<string>	Full path name for the file in ASCII coded as an unsigned short integer array.

Example

Command **:SYST:FILE #264<binary_block with 64 bytes with ASCII info>**

Example **:SYST:FILE #210myfile.dat**

12.21 :SYSTem:FILE:SIZE?

Description

Query the file size in bytes.

Response

The Proteus unit will return the file size in bytes.

Example

Query `:SYST:FILE:SIZE?`

12.22 :SYSTEM:FILE:DATA[<offset>,]#<header><binary_block>(?)

Description

This command will download binary-data to the specified offset in the Proteus system file.

Parameters

Name	Range	Type	Default	Description
[< offset >]		Integer	0	The byte offset in the system file.
#< header >		Integer		Contains information on the size of the binary block that follows.
< binary_block >		Binary		Block of binary data.

Example

Command `:SYSTEM:FILE:DATA #42048<binary_block>`

Query `:SYSTEM:FILE:DATA?`

12.23 :SYSTEM:FILE:DELete

Description

This command will delete the Proteus system file.

Example

Command `:SYSTEM:FILE:DEL`

13 Appendix Proteus SCPI MATLAB Script Examples

13.1 Introduction

The following MATLAB scripts are a series of examples to show how different operations can be performed by the Proteus series of Arbitrary Waveform Transceivers using SCPI commands. The scripts show the right sequence of commands leading to several results and also the way different binary data involved in each functionality is properly calculated, formatted, and transferred. The code in this section is found in the **Proteus_SCPI_MATLAB_Script_Examples_Ver.x.y.zip** file at the downloads page of the Tabor Electronics web site (https://www.taborelec.com/Downloads) and it can be used and modified according to the GNU rules.

Each script includes a series of supporting functions. These functions have been created to be reusable so they can be used as building blocks for any application. However, most of the communications and basic control of the Proteus unit is handled by a generic library (TEProteusInst.m) that it is also available in the example programs download package. Both the library and the example scripts (and the associated functions) are properly documented through the extensive usage of comments. All the examples are self-contained so external data (as waveforms) are not required.

The current examples are designed to use the VISA (Virtual Instrument Software Architecture) API to communicate with the target Proteus devices. They are also written to support all the Proteus models and operating modes. However, a particular Proteus may not support a specific function.

13.2 Opening a Session with Proteus

13.2.1 Programming Example 1 shows the way to open a session with any Proteus unit regardless of the interface being used for communication. The “LAN” option implements communication through a VISA standard socket-based TCP-IP device. The “DLL” directly uses the functions in the Proteus driver DLL to communicate through the PCIe bus within the PXIe bus. The “LAN” option can be implemented in any internal (embedded) or external controller through any TCP-IP compatible connection (internal host, external computer through Ethernet, USB, WiFi, Bluetooth, etc). The “DLL” option can be used with any internal (embedded) computer, or an external computer connected to a PXIe chassis using some PXIe bus extender including those using Thunderbolt interfaces. The “DLL” API offers a much faster transfer rate to/from the instrument, especially when large blocks of binary data must be transferred.

The examples shown here use a MATLAB function library, TEProteusInst.m, developed by Tabor to simplify and speed up communications to/from Proteus. An important feature of this library is that the same functions can be used for the same purpose regardless of the API being used, DLL or LAN. In other words, the same code will work using both APIs without any modification other than defining the right interface and identifier. For the “LAN” API; the IP address must be defined. For the DLL API, it is the slot number that must be declared.

13.2.1 Programming Example 1 identifies the model (including the serial number), extracts all the installed options, and give some basic parameters that must be known to calculate and download waveforms.

Among many others, these useful functions are included in the example:

- **ConnecToProteus**: Opening a session with a Proteus unit supporting all kind of control computers, interfaces, and APIs.
- ***getOptions***: Getting the Proteus options and returning them in an easy-to-handle format.

13.2.1 Programming Example 1

```
% BASIC EXAMPLE FOR CONNECTION TO PROTEUS USING VISA OR PXI
%============================================================
% VISA Communications from MATLAB requires the Instrument Control
Toolbox

clear;
close all;
clear variables;
clear global;
clc;

% Define IP Address for Target Proteus device descriptor
% VISA "Socket-Based" TCP-IP Device. Socket# = 5025
ipAddr = '127.0.0.1'; % '127.0.0.1' = Local Host, your IP address here
pxiSlot = 0; % Set 0 to select slot from attached modules

% Instrument setup
cType =          "LAN";  %"LAN" = VISA or "DLL" = PXI

if cType == "LAN"
    connPar =       ipAddr;
else
    connPar =       pxiSlot; % Your stot # here, o for manual
selection
end

paranoia_level = 2; % 0, 1 or 2

[inst, admin, idnstr, slotNumber] = ConnecToProteus(cType, connPar,
paranoia_level);
fprintf('Connected to: %s, slot: %d\n', idnstr, slotNumber);

% Get options using the standard IEEE-488.2 Command
optstr = getOptions(inst);

for i=1:length(optstr)
    fprintf('\nOption #%d Installed: %s', i, char(optstr(i)));
end

% Get Number of Channels
numchan = getNumOfChannels(idnstr);
fprintf('\n\nNumber of Channels = %d\n', numchan);

% Get min sampling rate
minsr = getMinSamplingRate(inst);
fprintf('\nMinimum Sample Rate = %d samples/sec\n', minsr);

% Get max sampling rate
```

```
maxsr = getMaxSamplingRate(inst);
fprintf('\nMaximum Sample Rate = %d samples/sec\n', maxsr);

% Get granularity
granul = getGranularity(idnstr, optstr, 16);
fprintf('\nGranularity = %d samples\n', granul);

% Get Sample Resolution
dacres = getDacResolution(inst);
fprintf('\nSample Resolution = %d bits\n', dacres);

% Disconnect, close VISA handle and destroy handle
if cType == "LAN"
    inst.Disconnect();
else
    admin.CloseInstrument(inst.InstrId);
    admin.Close();
end

clear inst;
clear admin;
close all;

function [  inst,...
            admin,...
            modelName,...
            sId] = ConnecToProteus( cType, ...
                                    connStr, ...
                                    paranoia_level)

% Connection to target Proteus
% cType specifies API. "LAN" for VISA, "DLL" for PXI
% connStr is the slot # as an integer(0 for manual selection) or IP
adress
% as an string
% Paranoia Level add additional checks for each transfer. 0 = no
checks.
% 1 = send OPC?, 2 = send SYST:ERROR?

% It returns
% inst: handler for the selected instrument
% admin: administrative handler
% modelName: string with model name for selected instrument (i.e.
"P9484")
% sId: slot number for selected instrument

    pid = feature('getpid');
    fprintf(1,'\nProcess ID %d\n',pid);

    dll_path = 'C:\\Windows\\System32\\TEPAdmin.dll';
    admin = 0;

    if cType == "LAN"
        try
```

```
            connStr = strcat('TCPIP::',connStr,'::5025::SOCKET');
            inst = TEProteusInst(connStr, paranoia_level);

            res = inst.Connect();
            assert (res == true);
            modelName = identifyModel(inst);
        catch ME
            rethrow(ME)
        end
    else
        asm = NET.addAssembly(dll_path);

        import TaborElec.Proteus.CLI.*
        import TaborElec.Proteus.CLI.Admin.*
        import System.*

        admin = CProteusAdmin(@OnLoggerEvent);
        rc = admin.Open();
        assert(rc == 0);

        try
            slotIds = admin.GetSlotIds();
            numSlots = length(size(slotIds));
            assert(numSlots > 0);

            % If there are multiple slots, let the user select one ..
            sId = slotIds(1);
            if numSlots > 1
                fprintf('\n%d slots were found\n', numSlots);
                for n = 1:numSlots
                    sId = slotIds(n);
                    slotInfo = admin.GetSlotInfo(sId);
                    if ~slotInfo.IsSlotInUse
                        modelName = slotInfo.ModelName;
                        if slotInfo.IsDummySlot && connStr == 0
                            fprintf(' * Slot Number:%d Model %s [Dummy
Slot].\n', sId, modelName);
                        elseif connStr == 0
                            fprintf(' * Slot Number:%d Model %s.\n',
sId, modelName);
                        end
                    end
                end
                pause(0.1);
                if connStr == 0
                    choice = input('Enter SlotId ');
                    fprintf('\n');
                else
                    choice = connStr;
                end
                sId = uint32(choice);
                slotInfo = admin.GetSlotInfo(sId);
                modelName = slotInfo.ModelName;
                modelName = strtrim(netStrToStr(modelName));
```

```
            end

            % Connect to the selected instrument ..
            should_reset = true;
            inst = admin.OpenInstrument(sId, should_reset);
            instId = inst.InstrId;

        catch ME
            admin.Close();
            rethrow(ME)
        end
    end
end

function model = identifyModel(inst)
    idnStr = inst.SendScpi('*IDN?');
    idnStr = strtrim(netStrToStr(idnStr.RespStr));
    idnStr = split(idnStr, ',');

    if length(idnStr) > 2
        model = idnStr(2);
    else
        model ='P9484M';
    end

%     model = idnStr;
end

function options = getOptions(inst)
    optStr = inst.SendScpi('*OPT?');
    optStr = strtrim(netStrToStr(optStr.RespStr));
    options = split(optStr, ',');
end

function minSr = getMinSamplingRate(inst)
    minSr = inst.SendScpi(':FREQ:RAST MIN?');
    minSr = strtrim(netStrToStr(minSr.RespStr));
    minSr = str2double(minSr);
end

function maxSr = getMaxSamplingRate(inst)
    maxSr = inst.SendScpi(':FREQ:RAST MAX?');
    maxSr = strtrim(netStrToStr(maxSr.RespStr));
    maxSr = str2double(maxSr);
end

function granularity = getGranularity(model, options, dacMode)

    flagLowGranularity = false;

    for i = 1:length(options)
        if contains(options(i), 'G1') || contains(options(i), 'G2')
            flagLowGranularity = true;
        end
```

```
    end

    granularity = 32;

    if contains(model, 'P258')
        granularity = 32;
        if flagLowGranularity
            granularity = 16;
        end
    elseif contains(model, 'P128')
        granularity = 32;
        if flagLowGranularity
            granularity = 16;
        end
    elseif contains(model, 'P948')
        if dacMode == 16
            granularity = 32;
            if flagLowGranularity
                granularity = 16;
            end
        else
            granularity = 64;
            if flagLowGranularity
                granularity = 32;
            end
        end
    elseif contains(model, 'P908')
        granularity = 64;
        if flagLowGranularity
            granularity = 32;
        end
    end
end

function dacRes = getDacResolution(inst)

    dacRes = inst.SendScpi(':TRAC:FORM?');
    dacRes = strtrim(netStrToStr(dacRes.RespStr));

    if contains(dacRes, 'U8')
        dacRes = 8;
    else
        dacRes = 16;
    end
end

function numOfChannels = getNumOfChannels(model)

    numOfChannels = 4;

    if contains(model, 'P9082')
        numOfChannels = 2;
    elseif contains(model, 'P9482')
        numOfChannels = 2;
```

```
    elseif contains(model, 'P1282')
        numOfChannels = 2;
    elseif contains(model, 'P2582')
        numOfChannels = 2;
    elseif contains(model, 'P9086')
        numOfChannels = 6;
    elseif contains(model, 'P9486')
        numOfChannels = 6;
    elseif contains(model, 'P1286')
        numOfChannels = 6;
    elseif contains(model, 'P2586')
        numOfChannels = 6;
    elseif contains(model, 'P9488')
        numOfChannels = 8;
    elseif contains(model, 'P1288')
        numOfChannels = 8;
    elseif contains(model, 'P2588')
        numOfChannels = 8;
    elseif contains(model, 'P94812')
        numOfChannels = 12;
    elseif contains(model, 'P12812')
        numOfChannels = 12;
    elseif contains(model, 'P25812')
        numOfChannels = 12;
    end
end
function [str] = netStrToStr(netStr)
    try
        str = convertCharsToStrings(char(netStr));
    catch
        str = '';
    end
end
```

13.3 Generating a Waveform in Multiple Channels

13.3.1 Programming Example 2 shows the way to calculate, format, and download waveforms to the target Proteus unit. The example adapts itself to the specific model so the number of channels, either 2 or 4, is defined. The script generates 4 different periodic waveforms by integration of the basic square waveform, so a square, a triangular, a cosine, and a sine waveform are generated as seen in this scope acquisition:

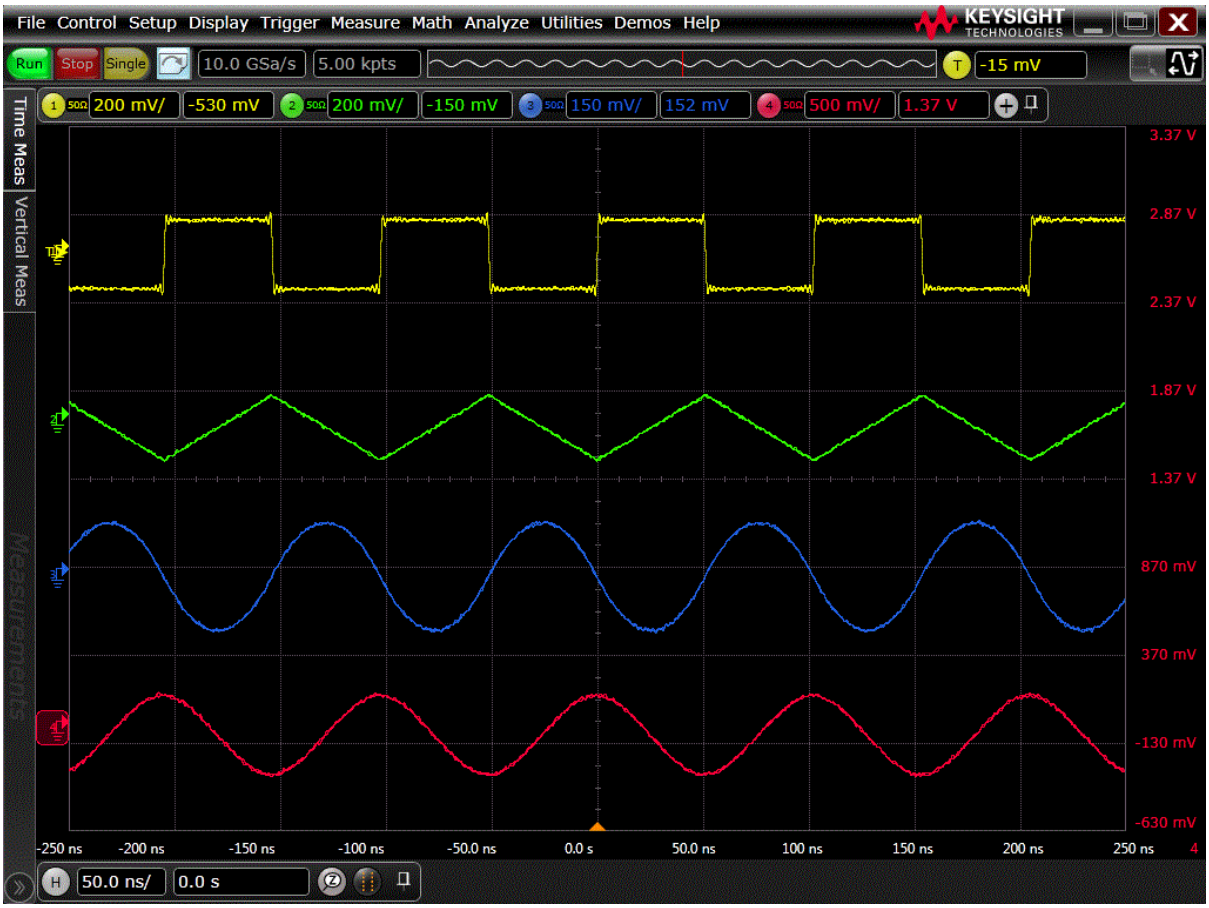

Figure 13-1 Waveforms Generated by Channel 1 – 4, Square, Triangular, Cosine, Sine

13.3.1 Programming Example 2 also shows how to generate, format, and download marker data. For each channel two markers are defined. Marker 1 carries a sync pulse aligned with the beginning of the waveform segment, while Marker 2 carries a random bit sequence. The duration of marker 1 is proportional to the channel number and the random data is different for each marker 2:

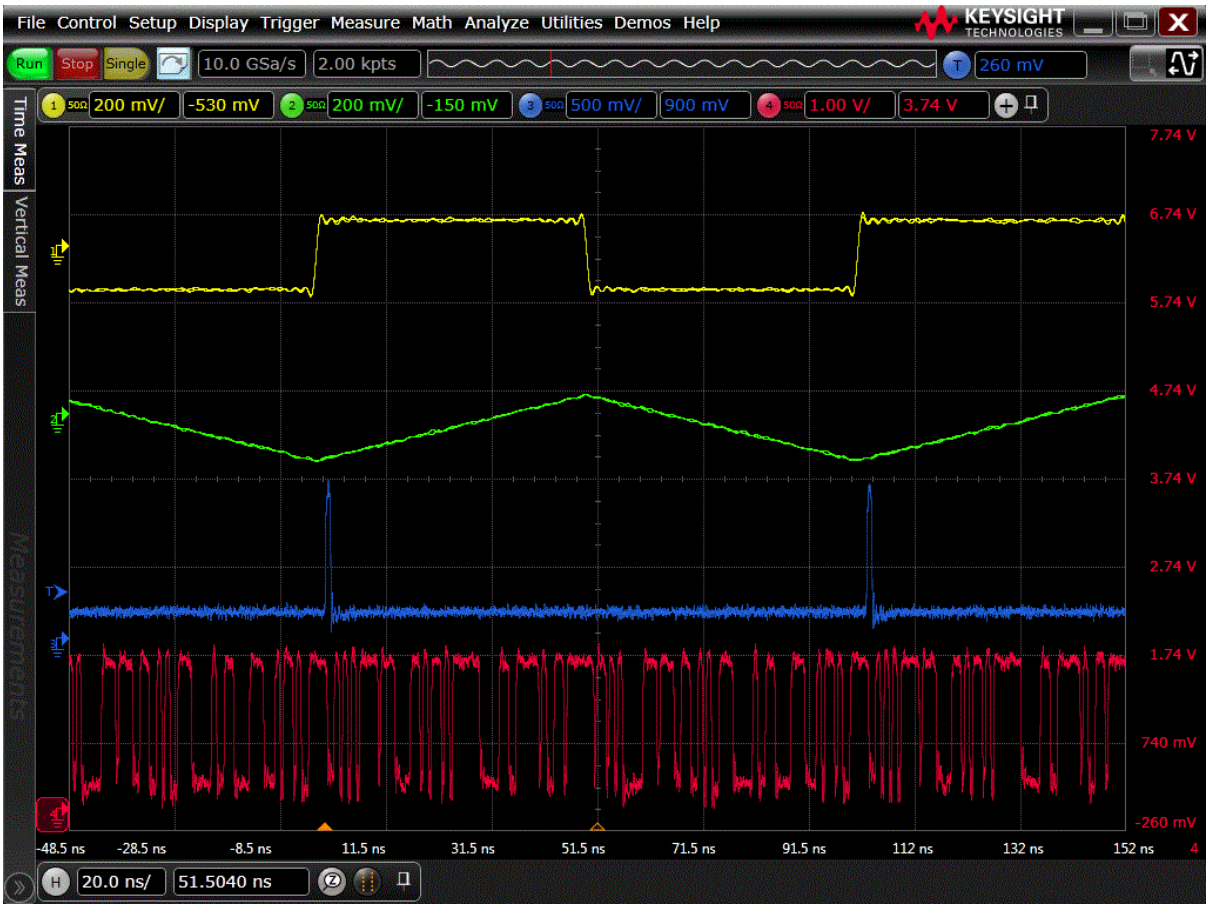

Figure 13-2 Channel 1, 2 with Channel 1 Marker Data (Blue, Red)

In the figure above marker 1 and marker 2 from channel 1 are shown at the bottom of the oscilloscope display.

In the figure below random data for maker 2 from ch1 and marker 2 from channel 2 are shown:

Figure 13-3 Channel 1 with Marker Data (Blue), Channel 2 with Marker Data (Red)

13.3.1 Programming Example 2 direct generation for all proteus models in direct and interpolated modes.

The example includes useful functions for different purposes:

- ***SendWfmToProteus***: Waveform download.
- ***myQuantization***: Proper normalized waveform (-1.0/+1.0 range) quantization for all the DAC modes.
- ***SendMkrToProteus***: Marker download
- ***FormatMkr2 and FormatMkr4***: Marker data formatting for 2 and 4 markers per channel.

13.3.1 Programming Example 2

```
% EXAMPLE FOR DIRECT MODE
%=====================================================
% This example calculates up to 4 different signals and download them
into
% each available channel in the target Proteus device.
%
% The basic waveform is an square waveform using the full DAC range
and it
```

```
% is downloaded to channel #1. For each channel, the waveform is
calculated
% by integration of the previous waveform in a similar way to some
analog
% signal generators, where the triangular wave is obtained by
integration
% of an square wave, and the sinusoidal waveform is obtained by
integration
% of the triangular wave. Channel #4, when available, will generate a
% "cosine" wave obatined by integration of the sinewave assigned to
channel
% #3.
% Markers for each channel are also calculated and downloaded. Marker
1 is
% a sync pulse. Its duration (in states) is equal to the channel
number.
% Marker 2 is just geenrating a random stream of bits.

clear;
close all;
clear variables;
clear global;
clc;

fprintf(1, 'INITIALIZING SETTINGS\n');

pid = feature('getpid');
fprintf(1,'\nProcess ID %d\n',pid);

% BASIC EXAMPLE FOR CONNECTION TO PROTEUS USING VISA OR PXI
%============================================================
% VISA Communications from MATLAB requires the Instrument Control
Toolbox

clear;
close all;
clear variables;
clear global;
clc;

% Define IP Address for Target Proteus device descriptor
% VISA "Socket-Based" TCP-IP Device. Socket# = 5025
ipAddr = '127.0.0.1'; %'127.0.0.1'= Local Host; % your IP here
pxiSlot = 0;

% Instrument setup
cType =         "LAN";  %"LAN" = VISA or "DLL" = PXI

if cType == "LAN"
    connPar =       ipAddr;
else
    connPar =       pxiSlot; % Your slot # here, o for manual
selection
end
```

```
paranoia_level = 2; % 0, 1 or 2
% Open Session and load libraries
[inst, admin, model, slotNumber] = ConnecToProteus(cType, connPar,
paranoia_level);

% Report model
fprintf('Connected to: %s, slot: %d\n', model, slotNumber);

% Reset AWG
inst.SendScpi('*CLS;*RST');

% Get options using the standard IEEE-488.2 Command
optstr = getOptions(inst);

samplingRate = 9000E6;
interpol     = 4;
dacMode = 16;

if (samplingRate / interpol) > 2.5E9
    dacMode = 8;
    interpol = 1;
end

if (samplingRate / interpol) < 250E6
    interpol = 1;
end

% Get granularity
granul = getGranularity(model, optstr, dacMode);
fprintf('\nGranularity = %d samples\n', granul);

% Get Active Channels and Segment #
[chanList, segmList] = GetChannels(model, samplingRate / interpol);
numOfChannels = length(chanList);

fprintf(1, 'Calculating WAVEFORMS\n');

minCycles = 1;
period = 1E-7;

% SETTING AWG
fprintf(1, 'SETTING AWG\n');

% Set sampling rate for AWG to maximum.
if interpol > 1
    inst.SendScpi(':FREQ:RAST 2.5E9');
    inst.SendScpi([':INT X ' num2str(interpol)]);
end
inst.SendScpi([':FREQ:RAST ' num2str(samplingRate)]);

wfmVolt = 0.5;
wfmOff = 0.0;
```

```
mkrVolt = 1.0;
mkrOff = 0.5;

for channel = 1:numOfChannels
    % Calculate basic square wave
    if mod(channel, 4) == 1
        myWfm = getSquareWfm(   samplingRate / interpol,...
                                minCycles,...
                                period,...
                                granul);
    end

    mkrDiv = 2;
    if dacMode == 8
        mkrDiv = 8;
    end

    myMkr1 = uint8(zeros(1, length(myWfm) / mkrDiv));
    myMkr1(1:channel) = uint8(1); % Sync marker duration depends on
the channel
    myMkr2 = rand(1, length(myMkr1)); % Random data is different for
each channel
    myMkr2 = uint8(myMkr2 > 0.5);
    myMkr = FormatMkr2(dacMode, myMkr1, myMkr2);
%     myMkr2 = uint8(myMkr2 > 0.5);
%     myMkr = myMkr + 2 * myMkr2;
%
%     if dacMode == 16
%         myMkr = myMkr(1:2:length(myMkr)) + 16 *
myMkr(2:2:length(myMkr));
%     end

    %Select Channel
    inst.SendScpi(sprintf(':INST:CHAN %d', chanList(channel)));
    % DAC Mode set to 'DIRECT" (Default)
    inst.SendScpi(':SOUR:MODE DIRECT');

    % Segment # processing
    % All Proteus models except the P908X share the same waveform
memory
    % bank among channel N+1 and N+2, N=0..NumOfChannels/2. This means
that
    % the same segment number cannot be used for this pair of
channels. In
    % this case the designated segment is used for the odd numbered
    % channels and the next segment is assigned to the even numbered
channel
    % of the same pair. All segments can be deleted just once for each
pair
    % of channels.
    if segmList(channel) == 1
        % All segments deleted for current waveform memory bank
        inst.SendScpi(':TRAC:DEL:ALL');
    end
```

```
    % Waveform Downloading
    % ********************

    fprintf(1, 'DOWNLOADING WAVEFORM FOR CH%d\n', chanList(channel));

    SendWfmToProteus(       inst,...
                            samplingRate,...
                            chanList(channel),...
                            segmList(channel),...
                            myWfm,...
                            dacMode,...
                            false);

    result = SendMkrToProteus(inst, myMkr);

    fprintf(1, 'WAVEFORM DOWNLOADED!\n');
    % Select segment for generation
    fprintf(1, 'SETTING AWG OUTPUT\n');

    inst.SendScpi(sprintf(':SOUR:FUNC:MODE:SEGM %d',
segmList(channel)));

    % Output voltage and offset
    inst.SendScpi([':SOUR:VOLT ' num2str(wfmVolt)]);
    inst.SendScpi([':SOUR:VOLT:OFFS ' num2str(wfmOff)]);
    % Activate outpurt and start generation
    inst.SendScpi(':OUTP ON');

    inst.SendScpi(':MARK:SEL 1');          %Marker1
    inst.SendScpi([':MARK:VOLT:PTOP ' num2str(mkrVolt)]);      %Vpp
    inst.SendScpi([':MARK:VOLT:OFFS ' num2str(mkrOff)]);      %DC
Offset
    inst.SendScpi(':MARK ON');
    inst.SendScpi(':MARK:SEL 2');          %Marker1
    inst.SendScpi([':MARK:VOLT:PTOP ' num2str(mkrVolt)]);      %Vpp
    inst.SendScpi([':MARK:VOLT:OFFS ' num2str(mkrOff)]);      %DC
Offset
    inst.SendScpi(':MARK ON');

    % The new waveform is calculated for the next channel
    if mod(channel, 4) < 3
        % Integration
        myWfm = cumsum(myWfm);
        % DC removal
        myWfm = myWfm - mean(myWfm);
        % Normalization to the -1.0/+1.0 range
        myWfm = myWfm / max(abs(myWfm));
    end
end

% It is recommended to disconnect from instrument at the end
if cType == "LAN"
    inst.Disconnect();
```

```
else
    admin.CloseInstrument(inst.InstrId);
    admin.Close();
end

function sqrWfm = getSquareWfm( samplingRate,...
                                numCycles,...
                                period,...
                                granularity)

    wfmLength = round(numCycles * period *samplingRate);
    wfmLength = round(wfmLength / granularity) * granularity;

    period = wfmLength / numCycles;
    sqrWfm = 0:(wfmLength - 1);
    sqrWfm = square(sqrWfm * 2 * pi / period);

end

function result = SendWfmToProteus(     inst,...
                                        samplingRate,...
                                        channel,...
                                        segment,...
                                        myWfm,...
                                        dacRes,...
                                        initialize)

    if dacRes == 16
            inst.SendScpi(':TRAC:FORM U16');
    else
            inst.SendScpi(':TRAC:FORM U8');
    end

    %Select Channel
    if initialize
        inst.SendScpi(':TRAC:DEL:ALL');
        inst.SendScpi([':FREQ:RAST ' num2str(samplingRate)]);
    end

    inst.SendScpi(sprintf(':INST:CHAN %d', channel));
    inst.SendScpi(sprintf(':TRAC:DEF %d, %d', segment,
length(myWfm)));
    % select segmen as the the programmable segment
    inst.SendScpi(sprintf(':TRAC:SEL %d', segment));

    % format Wfm
    myWfm = myQuantization(myWfm, dacRes, 0);

    % Download the binary data to segment
    prefix = ':TRAC:DATA 0,';

    if (dacRes==16)
        myWfm = uint16(myWfm);
        myWfm = typecast(myWfm, 'uint8');
```

```
    else
        myWfm = uint8(myWfm);
    end

    res = inst.WriteBinaryData(prefix, myWfm);
    assert(res.ErrCode == 0);

    if initialize
        inst.SendScpi(sprintf(':SOUR:FUNC:MODE:SEGM %d', segment))
        % Output voltage set to MAX
        inst.SendScpi(':SOUR:VOLT MAX');
        % Activate outpurt and start generation
        inst.SendScpi(':OUTP ON');
    end

    result = length(myWfm);
end

function result = SendMkrToProteus( inst,myMkr)
    % Download the binary data to segment
    prefix = ':MARK:DATA 0,';
    inst.WriteBinaryData(prefix, myMkr);
    %instHandle.SendBinaryData(prefix, myMkr, 'uint8');
    result = length(myMkr);
end

function mkrData = FormatMkr2(dac_Mode, mkr1, mkr2)
    % Mkr1 goes to bit 0 and Mkr2 goes to bit 1 in a 4-bit Nibble
    mkrData = mkr1 + 2 * mkr2;
    % For DAC Mode 8, just one Nibble per Byte is sent
    % For DAC Mode 16, two consecutive nibbles are multiplexed in one
byte
    if dac_Mode == 16
        mkrData = mkrData(1:2:length(mkrData)) + ...
            16 * mkrData(2:2:length(mkrData));
    end
end

function mkrData = FormatMkr4(dac_Mode, mkr1, mkr2, mkr3, mkr4)
    % Mkr1 goes to bit 0 and Mkr2 goes to bit 1 in a 4-bit Nibble
    mkrData = mkr1 + 2 * mkr2 + 4 * mkr3 + 8 * mkr4;
    % For DAC Mode 8, just one Nibble per Byte is sent
    % For DAC Mode 16, two consecutive nibbles are multiplexed in one
byte
    if dac_Mode == 16
        mkrData = mkrData(1:2:length(mkrData)) + ...
            16 * mkrData(2:2:length(mkrData));
    end
end

function [  inst,...
            admin,...
            modelName,...
            sId] = ConnecToProteus( cType, ...
```

```
                                    connStr, ...
                                    paranoia_level)

% Connection to target Proteus
% cType specifies API. "LAN" for VISA, "DLL" for PXI
% connStr is the slot # as an integer(0 for manual selection) or IP
adress
% as an string
% Paranoia Level add additional checks for each transfer. 0 = no
checks.
% 1 = send OPC?, 2 = send SYST:ERROR?

% It returns
% inst: handler for the selected instrument
% admin: administrative handler
% modelName: string with model name for selected instrument (i.e.
"P9484")
% sId: slot number for selected instrument

    pid = feature('getpid');
    fprintf(1,'\nProcess ID %d\n',pid);

    dll_path = 'C:\\Windows\\System32\\TEPAdmin.dll';
    admin = 0;
    sId = 0;
    if cType == "LAN"
        try
            connStr = strcat('TCPIP::',connStr,'::5025::SOCKET');
            inst = TEProteusInst(connStr, paranoia_level);

            res = inst.Connect();
            assert (res == true);
            modelName = identifyModel(inst);
        catch ME
            rethrow(ME)
        end
    else
        asm = NET.addAssembly(dll_path);

        import TaborElec.Proteus.CLI.*
        import TaborElec.Proteus.CLI.Admin.*
        import System.*

        admin = CProteusAdmin(@OnLoggerEvent);
        rc = admin.Open();
        assert(rc == 0);

        try
            slotIds = admin.GetSlotIds();
            numSlots = length(size(slotIds));
            assert(numSlots > 0);

            % If there are multiple slots, let the user select one ..
            sId = slotIds(1);
```

```
            if numSlots > 1
                fprintf('\n%d slots were found\n', numSlots);
                for n = 1:numSlots
                    sId = slotIds(n);
                    slotInfo = admin.GetSlotInfo(sId);
                    if ~slotInfo.IsSlotInUse
                        modelName = slotInfo.ModelName;
                        if slotInfo.IsDummySlot && connStr == 0
                            fprintf(' * Slot Number:%d Model %s [Dummy
Slot].\n', sId, modelName);
                        elseif connStr == 0
                            fprintf(' * Slot Number:%d Model %s.\n',
sId, modelName);
                        end
                    end
                end
                pause(0.1);
                if connStr == 0
                    choice = input('Enter SlotId ');
                    fprintf('\n');
                else
                    choice = connStr;
                end
                sId = uint32(choice);
                slotInfo = admin.GetSlotInfo(sId);
                modelName = slotInfo.ModelName;
                modelName = strtrim(netStrToStr(modelName));
            end

            % Connect to the selected instrument ..
            should_reset = true;
            inst = admin.OpenInstrument(sId, should_reset);
            instId = inst.InstrId;

        catch ME
            admin.Close();
            rethrow(ME)
        end
    end
end

function model = identifyModel(inst)
    idnStr = inst.SendScpi('*IDN?');
    idnStr = strtrim(netStrToStr(idnStr.RespStr));
    idnStr = split(idnStr, ',');

    if length(idnStr) > 1
        model = idnStr(2);
    else
        model ='';
    end
end

function options = getOptions(inst)
```

```
    optStr = inst.SendScpi('*OPT?');
    optStr = strtrim(netStrToStr(optStr.RespStr));
    options = split(optStr, ',');
end

function granularity = getGranularity(model, options, dacMode)

    flagLowGranularity = false;

    for i = 1:length(options)
        if contains(options(i), 'G1') || contains(options(i), 'G2')
            flagLowGranularity = true;
        end
    end

    flagLowGranularity = false; % TEMPORARY

    granularity = 32;

    if contains(model, 'P258')
        granularity = 32;
        if flagLowGranularity
            granularity = 16;
        end
    elseif contains(model, 'P128')
        granularity = 32;
        if flagLowGranularity
            granularity = 16;
        end
    elseif contains(model, 'P948')
        if dacMode == 16
            granularity = 32;
            if flagLowGranularity
                granularity = 16;
            end
        else
            granularity = 64;
            if flagLowGranularity
                granularity = 32;
            end
        end
    elseif contains(model, 'P908')
        granularity = 64;
        if flagLowGranularity
            granularity = 32;
        end
    end
end

function [chanList, segmList] = GetChannels(model, sampleRate)

    if contains(model, 'P9484') || contains(model, 'P2584') ||
contains(model, 'P1284')
        if sampleRate <= 2.5E9
```

```
            chanList = [1 2 3 4];
            segmList = [1 2 1 2];
        else
            chanList = [1 3];
            segmList = [1 1];
        end

    elseif contains(model, 'P9482') || contains(model, 'P2582') ||
contains(model, 'P1282')
        if sampleRate <= 2.5E9
            chanList = [1 2];
            segmList = [1 2];
        else
            chanList = [1];
            segmList = [1];
        end

    elseif contains(model, 'P9488') || contains(model, 'P2588') ||
contains(model, 'P1288')
        if sampleRate <= 2.5E9
            chanList = [1 2 3 4 5 6 7 8];
            segmList = [1 2 1 2 1 2 1 2];
        else
            chanList = [1 3 5 7];
            segmList = [1 1 1 1];
        end

    elseif contains(model, 'P94812') || contains(model, 'P25812') ||
contains(model, 'P12812')
        if sampleRate <= 2.5E9
            chanList = [1 2 3 4 5 6 7 8 9 10 11 12];
            segmList = [1 2 1 2 1 2 1 2 1 2 1 2];
        else
            chanList = [1 3 5 7 9 11];
            segmList = [1 1 1 1 1 1];
        end

    elseif contains(model, 'P9082')
        chanList = [1 2];
        segmList = [1 1];

    elseif contains(model, 'P9084')
        chanList = [1 2 3 4];
        segmList = [1 1 1 1];

    elseif contains(model, 'P9086')
        chanList = [1 2 3 4 5 6];
        segmList = [1 1 1 1 1 1];
    end
end

function retval = myQuantization (myArray, dacRes, minLevel)

    maxLevel = 2 ^ dacRes - 1;
```

```
    numOfLevels = maxLevel - minLevel + 1;

    retval = round((numOfLevels .* (myArray + 1) - 1) ./ 2);
    retval = retval + minLevel;

    retval(retval > maxLevel) = maxLevel;
    retval(retval < minLevel) = minLevel;

end

function [str] = netStrToStr(netStr)
    try
        str = convertCharsToStrings(char(netStr));
    catch
        str = '';
    end
end
```

13.4 Using the DUC Mode to Generate RF Signals

13.4.1 Programming Example 3 shows the way to calculate, format, and download baseband IQ signals and generate RF signals using the DUC in the different supported modes; ONE, TWO, or HALF. It also shows how to generate non-RF signals in the DUC mode for applications as symbol clock, reference, or auxiliary (i.e., envelope) signals. The example supports the generation or an arbitrary number of single sideband (SSB) tones, including equally spaced multitone signals, QPSK/QAM with arbitrary number of symbols and symbol rate, and 802.11ax (160MHz Modulation BW) and 802.11ad (1.8GHz Modulation Bandwidth). The 802.11ax, and 802.11ad waveforms require the WLAN Toolbox for MATLAB. This example can handle baseband signals at any sampling rate and a re-sampling function can transform them to the baseband sampling rate defined by the DAC sampling rate and interpolation factor.

The figure below depicts a 50MBaud QPSK signal generated by this script at 1GHz carrier frequency.

Figure 13-4 Modulation Analysis of a 50MBaud QPSK Signal with a 1GHz Carrier Frequency

The signal below is generated at 9GS/s, 8x interpolation factor, in IQ Mode 1. In a different channel, a symbol clock signal is generated. Here the I baseband signal and the clock signal are shown in an oscilloscope as depicted in the figure below.

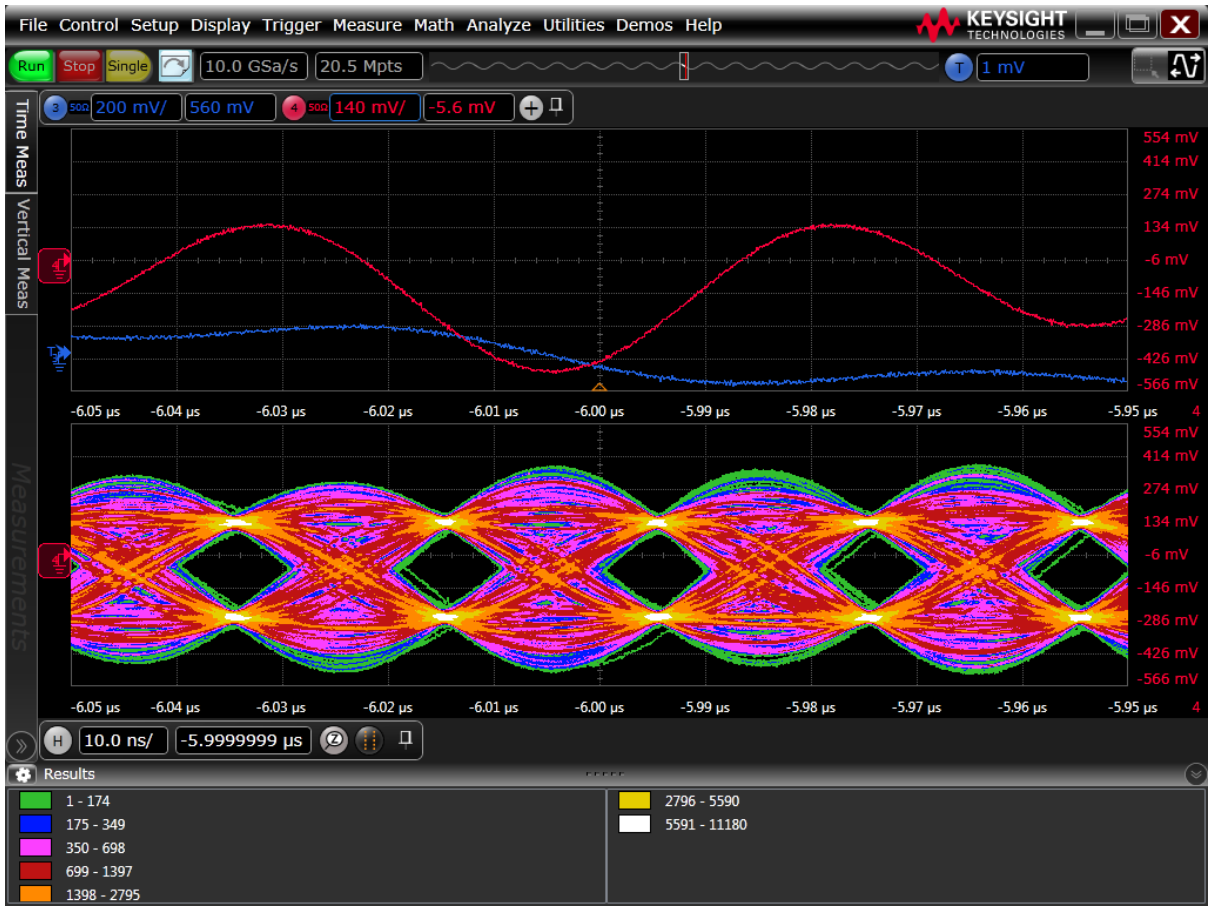

Figure 13-5 Top Window Shows the Baseband I&Q Waveforms, Bottom Window Shows the Eye Diagram for the I Waveform Using a Clock Signal Generated by a Another Channel

When selecting the “envelope” option for the baseband output, the RF signal and the corresponding envelope signal (i.e., to be fed to an “envelope tracking” amplifier” can be seen in an oscilloscope, see figure below.

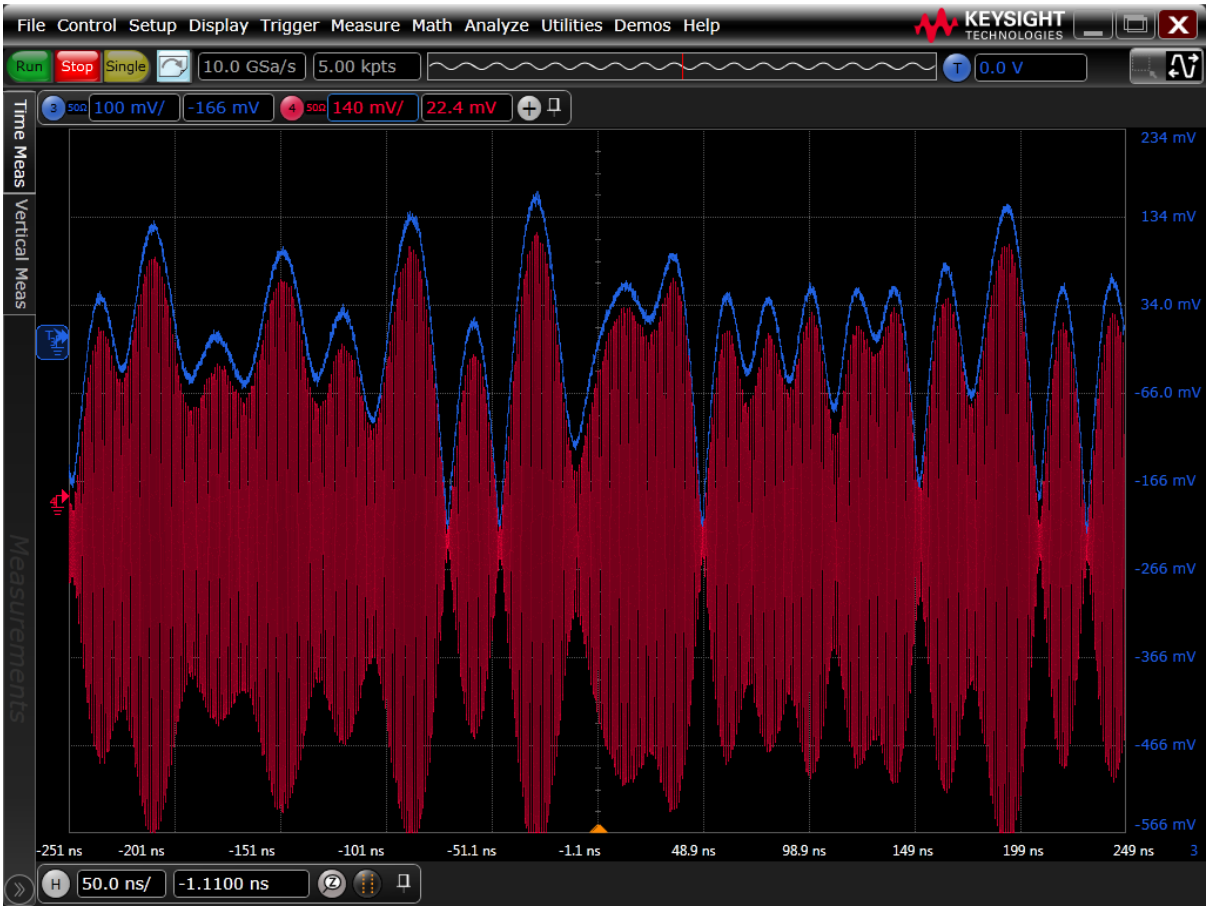

Figure 13-6 Red Waveform Shows a Digitally Modulated RF Signal While the Blue Waveform Shows the Envelope Signal

The figure below shows a 1800MBaud QPSK signal that is generated at 2GHz carrier frequency in the HALF mode.

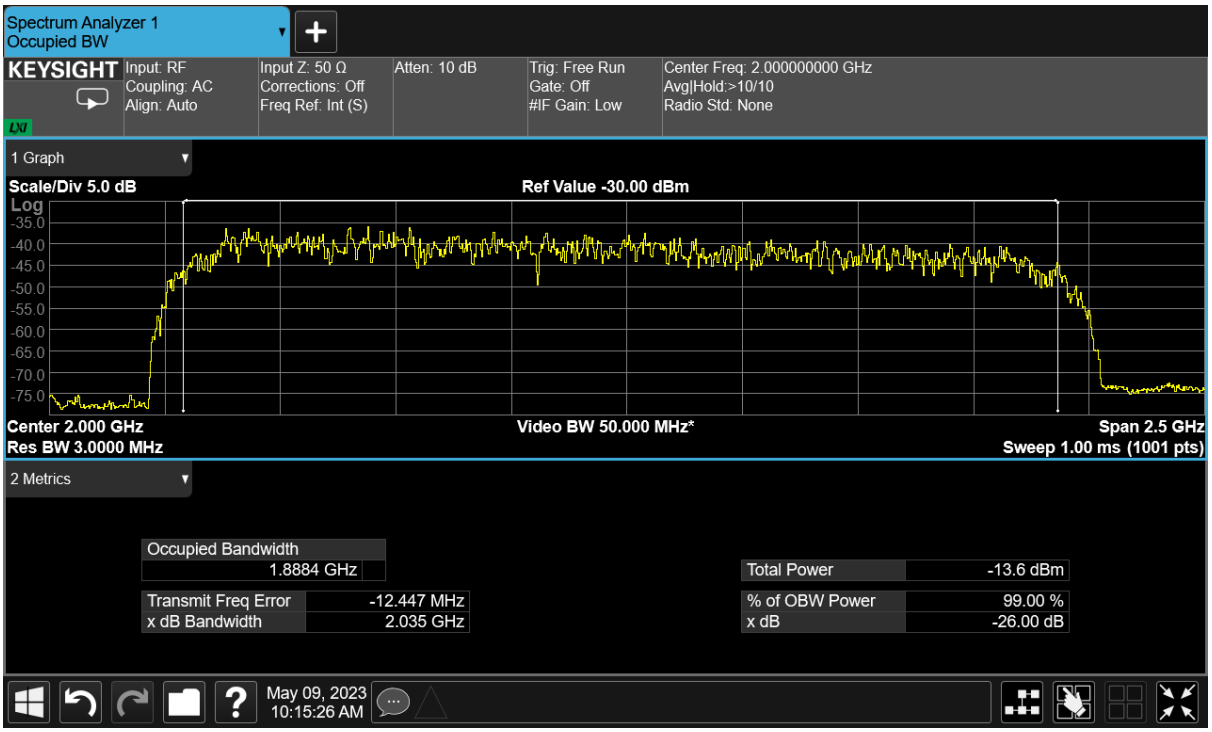

Figure 13-7 1800MBaud QPSK signal is Generated at 2GHz Carrier Frequency in the HALF Mode

Mode TWO can generate two different (and independent) modulated carriers. Here a 50MBaud QPSK signal is generated at 500MHz, and a 100MBaud QAM16 signal is generated at 2GHz using both DUC blocks in the same channels, see figure below.

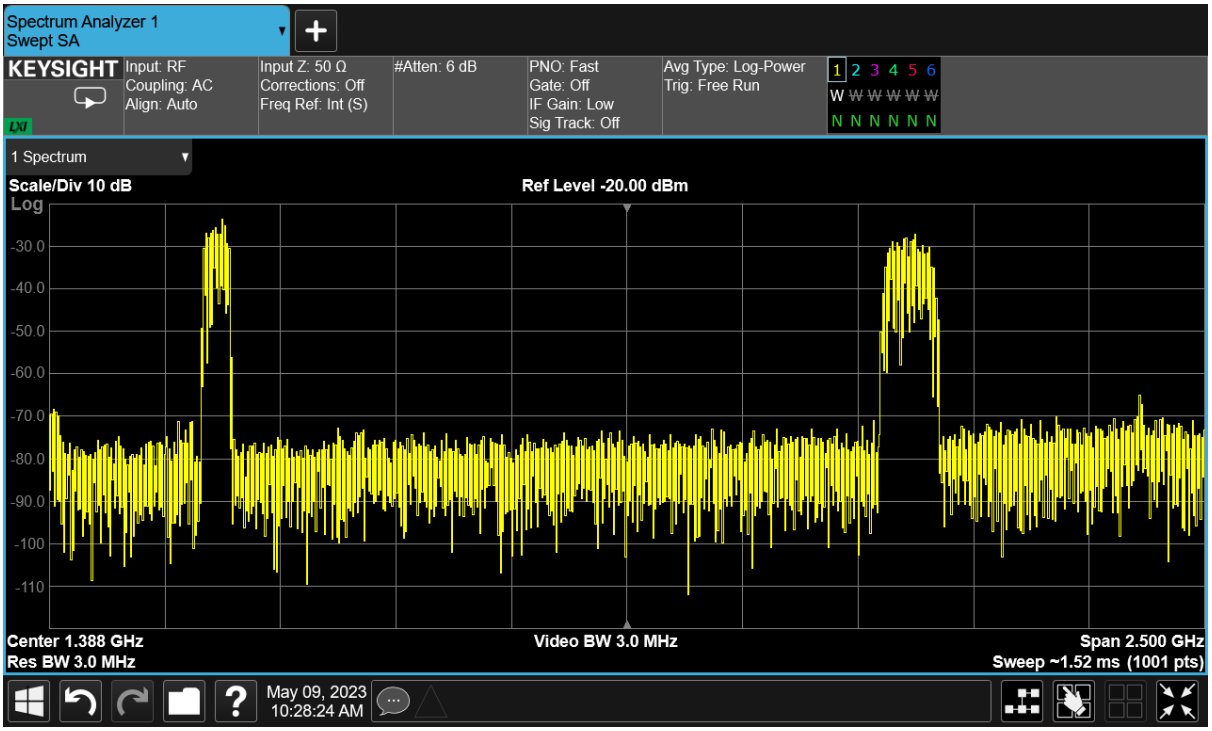

Figure 13-8 50MBaud QPSK signal is Generated at 500MHz, and a 100MBaud QAM16 signal is Generated at 2GHz Using Both DUC blocks in the Same Channels

The figure below depicts the 500MHz signal analysed with a VSA.

Figure 13-9 500MHz QPSK Signal Analysed with a VSA

The figure below depicts the 2GHz signal analysed with an VSA.

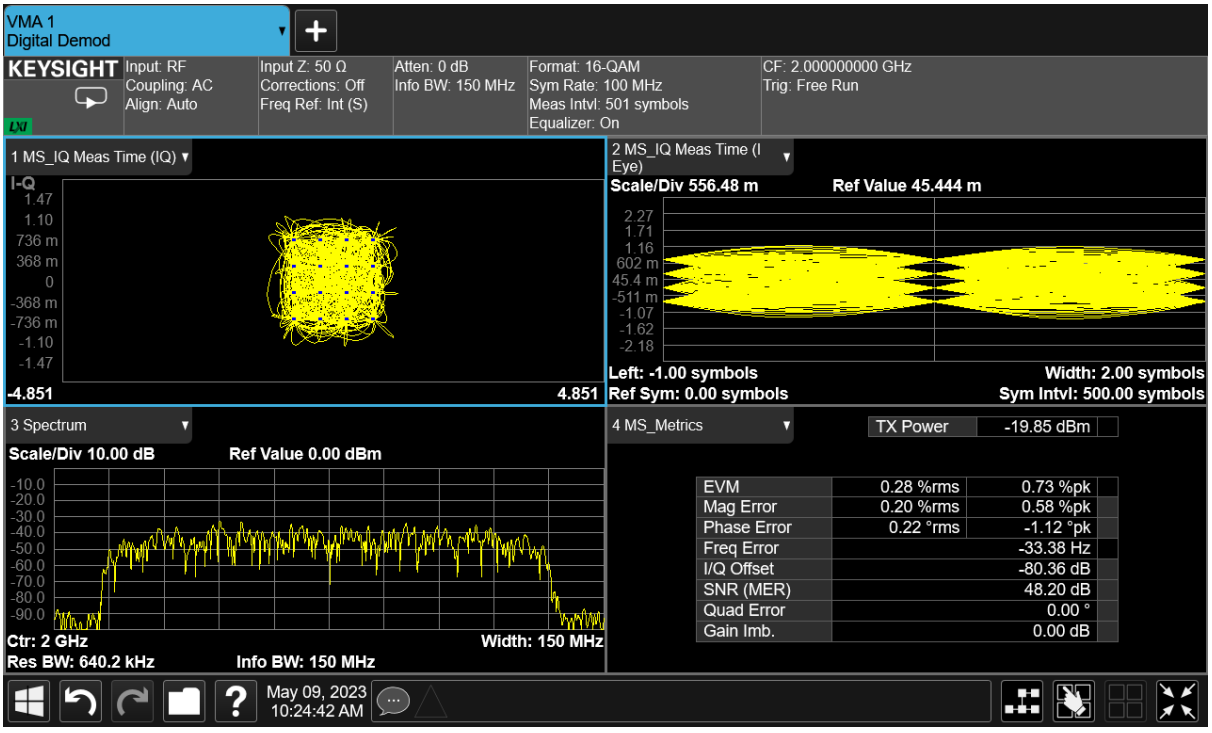

Figure 13-10 16QAM 2GHz Signal Analysed with an VSA

13.4.1 Programming Example 3 includes useful functions as listed below.

- ***SendIqmHalfWfm***: Setup Proteus for the IQ HALF mode and format and download I and Q waveforms.
- ***SendIqmOneWfm***: Setup Proteus for the IQ ONE mode and format and download I and Q waveforms.
- ***SendIqmTwoWfm***: Setup Proteus for the IQ TWO mode and format and download I and Q waveforms.
- ***NormalIq*** and ***NormalIq2***: Normalization for IQ baseband signals so the full DAC range is used without any chance to experience any signal clipping.
- ***myResampling***: Resampling of real or complex waveforms from one sample rate to another (higher or lower than the original) in a near optimal mode.
- ***Get_Multi_Tone***: Calculating arbitrary frequency multi-tone waveforms.
- ***Get_Qam***: Calculating QPSK, QAM16/32/64/128/256/1024 waveforms at any symbol rate with any number of symbols carrying random data. It supports raised cosine and sqrt(raised cosine) baseband filtering.
- ***Get_Qam_Clock***: Calculating symbol clock / N waveforms (N >=2) for symbol clock generation.
- ***Interleave***: I/Q waveform interleaving as required by mode ONE
- ***formatWfm2***: Multi-level interleaving for mode IQ mode TWO

13.4.1 Programming Example 3

```
% Baseband DUC example
% This is an example of how to generate signals and RF modulated
signals
% simultaneously with the Proteus AWT. A complex modulated  RF signal
is
% generated by one channel and the corresponding envelope signal is
% generated by another channel.

clear;
close all;
clear variables;
clear global;
clc;

% Define IP Address for Target Proteus device descriptor
% VISA "Socket-Based" TCP-IP Device. Socket# = 5025
ipAddr = '127.0.0.1'; %'127.0.0.1'= Local Host; % your IP here
pxiSlot = 0;

% Instrument setup
cType               = "LAN";  %"LAN" = VISA or "DLL" = PXI

if cType == "LAN"
    connPar         = ipAddr;
else
    connPar         = pxiSlot; % Your slot # here, o for manual
selection
end
```

```
paranoia_level = 0; % 0, 1 or 2
% Open Session and load libraries
[inst, admin, model, slotNumber] = ConnecToProteus(cType, connPar,
paranoia_level);

% Report model
fprintf('Connected to: %s, slot: %d\n', model(1), slotNumber(1));

% Reset AWG
inst.SendScpi('*CLS;*RST');

% Get options using the standard IEEE-488.2 Command
optstr = getOptions(inst);

% AWG Settings
duc_iq_mode                             = 1; % 0 = HALF, 1 = ONE, 2 =
TWO, 3 = NCO
sample_rate_dac                         = 9E9;
rf_channel                              = 1;
rf_segment                              = 1;
baseband_channel                        = 3;
baseband_segment                        = 3;
% Type of signal for test
% 1 = 802.11ax, 2 = 802.11ad, 3 = Multi-Tone, 4 = QAM
signal_type                             = 1;
carrier_freq                            = 2.412E9;
carrier_freq_2                          = 2.0E9; % For IQ Mode 2
baseband_mode                           = 2; % 1 = envelope, 2 = clock
(QAM)
% Envelope Tracking Settings
minimum_pwr                             = -20.0; % dB vs. peak power
smoothing_factor                        = 1000;

% Clock processing only makes sense for QAM
if signal_type ~= 4 && baseband_mode == 2
    baseband_mode = 1;
end

fprintf(1, 'BASEBAND WAVEFORM CALCULATION\n');

% Baseband waveform parameter definition
switch signal_type
    case 1
        interpolation_factor            = 8;
        actual_granularity              = 16;
        oversampling                    = 2;
        smoothing_factor                = 0.001;
        if baseband_mode == 2
            baseband_mode = 1;
        end

        [wfm_in, sample_rate_bb_in]     = Get_Wlan_ax(oversampling);
        wfm_in_2                        = wfm_in;
```

```
    case 2
        interpolation_factor            = 4;
        actual_granularity              = 32;
        smoothing_factor                = 0.005;
        if baseband_mode == 2
            baseband_mode = 1;
        end

        [wfm_in, sample_rate_bb_in]     = Get_Wlan_ad;
        wfm_in_2                        = wfm_in;
    case 3
        interpolation_factor            = 8;
        actual_granularity              = 16;
        num_of_tones                    = 40;
        offset_tone                     = 15;
        spacing                         = 1E6;
        oversampling                    = 1.1;
        smoothing_factor                = 1000; %0.05


        [wfm_in, sample_rate_bb_in]     = Get_Multi_Tone(
num_of_tones, ...

offset_tone,...
                                                        spacing,
...

oversampling);
        wfm_in_2                        = wfm_in;
    case 4
        interpolation_factor            = 8;
        actual_granularity              = 32;
        % modType           Modulation
        % 1                 QPSK
        % 2                 QAM16
        % 3                 QAM32
        % 4                 QAM64
        % 5                 QAM128
        % 6                 QAM256
        % 7                 QAM512
        % 8                 QAM1024
        modulation_type                 = 1; % QPSK
        num_of_symbols                  = 2^11;
        symbol_rate                     = 100E6; %50E6
        filter_type                     = 'sqrt'; % 'normal' or 'sqrt'
        roll_off                        = 0.15;
        oversampling                    = 6;
        smoothing_factor                = 0.001;

        [wfm_in, sample_rate_bb_in]     = Get_Qam(  modulation_type,
...
                                                    num_of_symbols,
...
                                                    symbol_rate, ...
```

```
                                                    filter_type,...
                                                    roll_off, ...
                                                    oversampling);
        % Second baseband waveform for IQ Mode 2. It must be
consistent in
        % sampling rate and time window with first waveform
        modulation_type_2               = 2; % 16QAM
        num_of_symbols_2                = 2^12; % Twice the symbols
        symbol_rate_2                   = 100E6; %Twice the baud rate
        filter_type_2                   = 'sqrt'; % 'normal' or 'sqrt'
        roll_off_2                      = 0.25;
        oversampling_2                  = 3; % Half the oversampling
        smoothing_factor                = 0.001;

        [wfm_in_2, sample_rate_bb_in_2] = Get_Qam(  modulation_type_2,
...
                                                    num_of_symbols_2,
...
                                                    symbol_rate_2, ...
                                                    filter_type_2,...
                                                    roll_off_2, ...
                                                    oversampling_2);
        % For QAM and clock baseband signal, clock waveform must be
calculated
        if  baseband_mode == 2
            baseband_wfm = Get_Qam_Clock(   num_of_symbols, ...
                                            roll_off, ...
                                            oversampling,...
                                            4);
        end

end
% Resampling must be carrier out for the DUC baseband sampling rate
sample_rate_bb_out = sample_rate_dac / interpolation_factor;
wfm_length_in = length(wfm_in);

%Calculation of lenght of the interpolated waveform
wfm_length_out = floor(wfm_length_in * sample_rate_bb_out /...
    (sample_rate_bb_in * actual_granularity)) * actual_granularity;

fprintf(1, 'BASEBAND WAVEFORM RESAMPLING\n');

%%%%%%%%%%%%%%%%%%%%%%       RESAMPLING      %%%%%%%%%%%%%%%%%%%%%%%%%
wfm_out = myResampling(wfm_in, wfm_length_out, true, 60);
wfm_out_2 = myResampling(wfm_in_2, wfm_length_out, true, 60);

if signal_type == 4 && baseband_mode == 2
    % Clock waveform resampling
    baseband_wfm = myResampling(baseband_wfm, wfm_length_out, true,
60);
else
    % Get envelope tracking waveform from RF waveform
    [baseband_wfm, ref_envelope] = Get_Envelope(    wfm_out, ...
```

```
                                                        smoothing_factor,
...
                                                        minimum_pwr);
end

% Sample rate must be corected to compensate for the timing error
% introduced by the granularity requirements
actual_dac_sample_rate = wfm_length_out * interpolation_factor *...
    sample_rate_bb_in / wfm_length_in;

% Graph calculated waveforms in a proper way
fprintf(1, 'BASEBAND WAVEFORM GRAPHS\n');
if baseband_mode == 1
    % Show RF waveform in graph #1
    % And raw envelope and smoothed envelope in Graph #2
    DrawEnvelope(   wfm_out, ...
                    baseband_wfm, ...
                    ref_envelope, ...
                    sample_rate_bb_out);
else
    % Show unfiltered IQ and eye diagram in the top
    % and filtered IQ and eye diagram in the bottom
    DrawEyeDiagram( 3,...
                    1000, ...
                    actual_dac_sample_rate / interpolation_factor, ...
                    symbol_rate, ...
                    roll_off, ...
                    wfm_out, ...
                    baseband_wfm);
end

%%%%%%%%%%%%%%%%%%%%%%%%% DOWNLOAD RF WAVEFORM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf(1, 'RF WAVEFORM DOWNLOAD AND ACTIVATION\n');
% All previous waveforms will be deleted from waveform memory
inst.SendScpi(':TRAC:DEL:ALL');
% Format and download RF Signal
switch duc_iq_mode
    case 0
        result = SendIqmHalfWfm(inst,...
                                actual_dac_sample_rate,...
                                interpolation_factor,...
                                rf_channel,...
                                rf_segment,...
                                carrier_freq,...
                                0.0,...
                                true,...
                                wfm_out,...
                                16);
    case 1
        result = SendIqmOneWfm( inst,...
                                actual_dac_sample_rate,...
                                interpolation_factor,...
                                rf_channel,...
```

```
                                rf_segment,...
                                carrier_freq,...
                                0.0,...
                                true,...
                                wfm_out,...
                                16);
        result = SendIqmOneWfm( inst,...
                                actual_dac_sample_rate,...
                                interpolation_factor,...
                                rf_channel + 1,...
                                rf_segment + 1,...
                                carrier_freq,...
                                -90.0,...
                                true,...
                                wfm_out,...
                                16);
    case 2
        result = SendIqmTwoWfm( inst,...
                                actual_dac_sample_rate,...
                                interpolation_factor,...
                                rf_channel,...
                                rf_segment,...
                                carrier_freq,...
                                carrier_freq_2,...
                                0.0,...
                                0.0,...
                                true,...
                                wfm_out,...
                                wfm_out_2,...
                                16);

    case 3
       SetNco(  inst,...
                sample_rate_dac,...
                rf_channel,...
                carrier_freq,...
                0.0,...
                true);
%        for fr = 1E6:1E6:4500E6
%            inst.SendScpi(sprintf(':NCO:CFR1 %f', fr));
%        end


end

%%%%%%%%%%%%%%%%%%%%%%%% DOWNLOAD BB WAVEFORM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Format and download Baseband (Envelope or Clock) Signal
fprintf(1, 'BASEBAND WAVEFORM DOWNLOAD AND ACTIVATION\n');
switch duc_iq_mode
    case 0
        result = SendIqmHalfWfm(inst,...
                                actual_dac_sample_rate,...
                                interpolation_factor,...
```

```
                                baseband_channel,...
                                baseband_segment,...
                                0.0,...
                                0.0,...
                                true,...
                                baseband_wfm,...
                                16);

    case 1
        result = SendIqmOneWfm( inst,...
                                actual_dac_sample_rate,...
                                interpolation_factor,...
                                baseband_channel,...
                                baseband_segment,...
                                0.0,...
                                0.0,...
                                true,...
                                baseband_wfm,...
                                16);


end

% It is recommended to disconnect from instrument at the end
if cType == "LAN"
    inst.Disconnect();
else
    admin.CloseInstrument(inst.InstrId);
    admin.Close();
end

function result = SendIqmOneWfm(    inst,...
                                    samplingRate,...
                                    interpol,...
                                    channel,...
                                    segment,...
                                    cfr,...
                                    phase,...
                                    apply6db,...
                                    myWfm,...
                                    dacRes)

    % format Wfm and normalize waveform
    %myWfm = MyProteusInterpolation(myWfm, interpol, true);
    myWfm = NormalIq(myWfm);
    myWfm = Interleave(real(myWfm), imag(myWfm));
    myWfm = myQuantization(myWfm, dacRes, 1);

    % Select Channel
    inst.SendScpi(sprintf(':INST:CHAN %d', channel));

    inst.SendScpi([':FREQ:RAST ' num2str(2.5E9)]);
    % Interpolation factor for I/Q waveforms
    switch interpol
```

```
        case 2
            inst.SendScpi(':SOUR:INT X2');

        case 4
            inst.SendScpi(':SOUR:INT X4');

        case 8
            inst.SendScpi(':SOUR:INT X8');
    end

    % DAC Mode set to 'DUC' and IQ Modulation mode set to 'ONE'
    inst.SendScpi(':MODE DUC');
    inst.SendScpi(':IQM ONE');


    inst.SendScpi([':FREQ:RAST ' num2str(samplingRate)]);

    fprintf(1, sprintf('DOWNLOADING WAVEFORM: %d samples\n',
length(myWfm)));
    result = SendWfmToProteus(  inst,...
                                samplingRate,...
                                channel,...
                                segment,...
                                myWfm,...
                                dacRes,...
                                false);

    fprintf(1, 'WAVEFORM DOWNLOADED!\n');
    clear myWfm;

    % Select segment for generation
    fprintf(1, 'SETTING AWG OUTPUT\n');
    inst.SendScpi(sprintf(':FUNC:MODE:SEGM %d', segment));
    % Output volatge set to MAX
    inst.SendScpi(':SOUR:VOLT MAX');

    % NCO set-up
    % 6dB IQ Modulation gain applied
    if apply6db
        inst.SendScpi(':NCO:SIXD1 ON');
    else
        inst.SendScpi(':NCO:SIXD1 OFF');
    end
    % NCO frequency and phase setting
    inst.SendScpi(sprintf(':NCO:CFR1 %d', cfr));
    inst.SendScpi(sprintf(':NCO:PHAS1 %d', phase));

    % Activate outpurt and start generation
    inst.SendScpi(':OUTP ON');

    fprintf(1, 'SETTING SAMPLING CLOCK\n');
    % Set sampling rate for AWG as defined in the preamble.
    inst.SendScpi([':FREQ:RAST ' num2str(samplingRate)]);
end
```

```
function result = SendIqmHalfWfm(   inst,...
                                    samplingRate,...
                                    interpol,...
                                    channel,...
                                    segment,...
                                    cfr,...
                                    phase,...
                                    apply6db,...
                                    myWfm,...
                                    dacRes)

    myWfm = NormalIq(myWfm);
    myWfmI = real(myWfm);
    myWfmI = myQuantization(myWfmI, dacRes, 1);
    myWfmQ = imag(myWfm);
    myWfmQ = myQuantization(myWfmQ, dacRes, 1);

    % Channel I is 2N - 1 and Channel Q is 2N
    % If channel is even, then base channle number is corrected
    if mod(channel, 2) == 0
        channel = channel - 1;
    end

    % Set temporary sampling rate for AWG.
    inst.SendScpi([':SOUR:FREQ:RAST ' num2str(2.5E9)]);

    res = inst.SendScpi('*OPC?');

    % The Half mode requires setting two channels
    inst.SendScpi(sprintf(':INST:CHAN %d', channel));

    inst.SendScpi(':MODE DUC');
    inst.SendScpi(':IQM HALF');

    % Interpolation factor for I/Q waveforms
    switch interpol
        case 2
            inst.SendScpi(':INT X2');

        case 4
            inst.SendScpi(':SOUR:INT X4');

        case 8
            inst.SendScpi(':SOUR:INT X8');
    end

    inst.SendScpi(sprintf(':INST:CHAN %d', channel + 1));

    inst.SendScpi(':SOUR:MODE DUC');
    inst.SendScpi(':SOUR:IQM HALF');

    % Interpolation factor for I/Q waveforms
    switch interpol
```

```
    case 2
        inst.SendScpi(':SOUR:INT X2');

    case 4
        inst.SendScpi(':SOUR:INT X4');

    case 8
        inst.SendScpi(':SOUR:INT X8');
end

inst.SendScpi([':SOUR:FREQ:RAST ' num2str(samplingRate)]);
% DAC Mode set to 'DUC' and IQ Modulation mode set to 'ONE';

% Waveform Downloading
% ********************
fprintf(1, 'DOWNLOADING WAVEFORM I\n');
result = SendWfmToProteus(  inst,...
                            samplingRate,...
                            channel,...
                            segment,...
                            myWfmI,...
                            dacRes,...
                            false);

fprintf(1, 'DOWNLOADING WAVEFORM Q\n');
result = SendWfmToProteus(  inst,...
                            samplingRate,...
                            channel + 1,...
                            segment + 1,...
                            myWfmQ,...
                            dacRes,...
                            false);

fprintf(1, 'WAVEFORMS DOWNLOADED!\n');
clear myWfm;

% Select segment for generation
fprintf(1, 'SETTING AWG OUTPUT\n');
% Q Channel
inst.SendScpi(sprintf(':INST:CHAN %d', channel + 1));
inst.SendScpi(sprintf(':FUNC:MODE:SEGM %d', segment + 1));
% NCO frequency and phase setting
inst.SendScpi(sprintf(':SOUR:NCO:CFR1 %d', cfr));
inst.SendScpi(sprintf(':SOUR:NCO:PHAS1 %d', phase));
if apply6db
    inst.SendScpi(':SOUR:NCO:SIXD1 ON');
else
    inst.SendScpi(':SOUR:NCO:SIXD1 OFF');
end

% Output volatge set to MAX
inst.SendScpi(':SOUR:VOLT 0.5');
% Activate outpurt and start generation
inst.SendScpi(':OUTP ON');
```

```
    % I Channel is set up in the end as this is the physical active
output
    % I Channel
    inst.SendScpi(sprintf(':INST:CHAN %d', channel));
    inst.SendScpi(sprintf(':FUNC:MODE:SEGM %d', segment));
    % NCO frequency and phase setting
    inst.SendScpi(sprintf(':SOUR:NCO:CFR1 %d', cfr));
    inst.SendScpi(sprintf(':SOUR:NCO:PHAS1 %d', phase));
    if apply6db
        inst.SendScpi(':SOUR:NCO:SIXD1 ON');
    else
        inst.SendScpi(':SOUR:NCO:SIXD1 OFF');
    end

    % Output volatge set to MAX
    inst.SendScpi(':SOUR:VOLT 0.5');
    % Activate outpurt and start generation
    inst.SendScpi(':OUTP ON');

    fprintf(1, 'SETTING SAMPLING CLOCK\n');
    % Set sampling rate for AWG as defined in the preamble.
    inst.SendScpi([':FREQ:RAST ' num2str(samplingRate)]);
end

function result = SendIqmTwoWfm(    inst,...
                                    samplingRate,...
                                    interpol,...
                                    channel,...
                                    segment,...
                                    cfr1,...
                                    cfr2,...
                                    phase1,...
                                    phase2,...
                                    apply6db,...
                                    myWfm1,...
                                    myWfm2,...
                                    dacRes)

    [myWfm1,  myWfm2] = NormalIq2(myWfm1, myWfm2);

    myWfm = formatWfm2(myWfm1, myWfm2);

    % Select Channel
    inst.SendScpi(sprintf(':INST:CHAN %d', channel));


    inst.SendScpi([':FREQ:RAST ' num2str(2.5E9)]);
    % Interpolation factor for I/Q waveforms
    switch interpol
        case 2
            inst.SendScpi(':SOUR:INT X2');

        case 4
```

```
            inst.SendScpi(':SOUR:INT X4');

        case 8
            inst.SendScpi(':SOUR:INT X8');
    end

    % DAC Mode set to 'DUC' and IQ Modulation mode set to 'ONE'
    % DAC Mode set to 'DUC' and IQ Modulation mode set to 'TWO'
    inst.SendScpi(':MODE DUC');
    inst.SendScpi(':IQM TWO');


    inst.SendScpi([':FREQ:RAST ' num2str(samplingRate)]);

    fprintf(1, sprintf('DOWNLOADING WAVEFORM: %d samples\n',
length(myWfm)));
    result = SendWfmToProteus(  inst,...
                                samplingRate,...
                                channel,...
                                segment,...
                                myWfm,...
                                dacRes,...
                                false);

    fprintf(1, 'WAVEFORM DOWNLOADED!\n');
    clear myWfm;

    % Select segment for generation
    fprintf(1, 'SETTING AWG OUTPUT\n');
    inst.SendScpi(sprintf(':FUNC:MODE:SEGM %d', segment));
    % Output volatge set to MAX
    inst.SendScpi(':SOUR:VOLT 0.5');

    % NCO set-up
    % 6dB IQ Modulation gain applied
    if apply6db
        inst.SendScpi(':NCO:SIXD1 ON');
        inst.SendScpi(':NCO:SIXD2 ON');
    else
        inst.SendScpi(':NCO:SIXD1 OFF');
        inst.SendScpi(':NCO:SIXD2 OFF');
    end
    % NCO frequency and phase setting
    inst.SendScpi(sprintf(':NCO:CFR1 %d', cfr1));
    inst.SendScpi(sprintf(':NCO:CFR2 %d', cfr2));
    inst.SendScpi(sprintf(':NCO:PHAS1 %d', phase1));
    inst.SendScpi(sprintf(':NCO:PHAS2 %d', phase2));

    % Activate outpurt and start generation
    inst.SendScpi(':OUTP ON');

    fprintf(1, 'SETTING SAMPLING CLOCK\n');
    % Set sampling rate for AWG as defined in the preamble.
    inst.SendScpi([':FREQ:RAST ' num2str(samplingRate)]);
```

```
end

function SetNco(    inst,...
                    samplingRate,...
                    channel,...
                    cfr,...
                    phase,...
                    apply6db)

    % Select Channel
    inst.SendScpi(sprintf(':INST:CHAN %d', channel));
    fprintf(1, 'SETTING SAMPLING CLOCK\n');
    inst.SendScpi([':FREQ:RAST ' num2str(samplingRate)]);
    % DAC Mode set to 'NCO'
    inst.SendScpi(':MODE NCO');
    % 'NCO' Settings
    inst.SendScpi(sprintf(':NCO:CFR1 %d', cfr));
    inst.SendScpi(sprintf(':NCO:PHAS1 %d', phase));
    if apply6db
        inst.SendScpi(':NCO:SIXD1 ON');
    else
        inst.SendScpi(':NCO:SIXD1 OFF');
    end

    % Output volatge set to MAX
    inst.SendScpi(':SOUR:VOLT 0.5');
    % Activate outpurt and start generation
    inst.SendScpi(':OUTP ON');

    %fprintf(1, 'SETTING SAMPLING CLOCK\n');
    % Set sampling rate for AWG as defined in the preamble.
    %inst.SendScpi([':FREQ:RAST ' num2str(samplingRate)]);
end

function result = SendWfmToProteus( inst,...
                                    samplingRate,...
                                    channel,...
                                    segment,...
                                    myWfm,...
                                    dacRes,...
                                    initialize)

    if dacRes == 16
            inst.SendScpi(':TRAC:FORM U16');
    else
            inst.SendScpi(':TRAC:FORM U8');
    end

    %Select Channel
    if initialize
        inst.SendScpi(':TRAC:DEL:ALL');
        inst.SendScpi([':FREQ:RAST ' num2str(samplingRate)]);
    end
```

```
    inst.SendScpi(sprintf(':INST:CHAN %d', channel));
    inst.SendScpi(sprintf(':TRAC:DEF %d, %d', segment,
length(myWfm)));
    % select segmen as the the programmable segment
    inst.SendScpi(sprintf(':TRAC:SEL %d', segment));

    % format Wfm
%     myWfm = myQuantization(myWfm, dacRes, 1);

    % Download the binary data to segment
    prefix = ':TRAC:DATA 0,';

    if (dacRes==16)
        myWfm = uint16(myWfm);
        myWfm = typecast(myWfm, 'uint8');
    else
        myWfm = uint8(myWfm);
    end
    tic;
    %res = inst.WriteBinaryData(':TRAC:DATA ', myWfm);
    res = inst.WriteBinaryData(prefix, myWfm);

    assert(res.ErrCode == 0);

%     if dacRes == 16
%         inst.SendBinaryData(prefix, myWfm, 'uint16');
%     else
%         inst.SendBinaryData(prefix, myWfm, 'uint8');
%     end

    if initialize
        inst.SendScpi(sprintf(':SOUR:FUNC:MODE:SEGM %d', segment))
        % Output voltage set to MAX
        inst.SendScpi(':SOUR:VOLT 0.5');
        % Activate outpurt and start generation
        inst.SendScpi(':OUTP ON');
    end

    result = length(myWfm);
end

function resampling_filter = GetResamplingFilter(
num_of_convolution_samples, ...

resolution_of_filter, ...
                                                    bw_fraction)
    % Creation of sinc lookup table
    % The NumOfConvolutionSamples paramters controls the quality of
the
    % resampling filter in terms of roll-off and attenuation at the
stop
    % band. The more, the better quality, the longer calculation time.
    % ResFilter sets the number of values per sample time to be
included in
```

```
    % the look-up table. The more, the better quality, the longer
    % calculation time.
    % bwFrac reduces de BW of the filter to avoid aliasing problems
caused
    % by the roll-off of the resampling filter.
    % A resampling filter object is created with the lookup table for
it
    % (just one side as it is symmetrical) and all the associated
    % parameters.
    resampling_filter.num_of_samples = num_of_convolution_samples;
    resampling_filter.resolution = resolution_of_filter;
    resampling_filter.bw_fraction = bw_fraction;
    sinc_length = floor(num_of_convolution_samples *
resolution_of_filter / bw_fraction);
    resampling_filter.filter = 0:(sinc_length);
    resampling_filter.filter = resampling_filter.filter /
resolution_of_filter;
    resampling_filter.filter = resampling_filter.filter * bw_fraction;
    % Basic filter shape is ideal low pass filter (sinc)
    resampling_filter.filter = sinc(resampling_filter.filter);
    % Flattop window is applied to improve flatness and stop band
rejection
    windowed_filter = flattopwin(2 * sinc_length);
    windowed_filter = windowed_filter(sinc_length:end);
    resampling_filter.filter = resampling_filter.filter .*
windowed_filter';
end

function output_wfm = myResampling (    input_wfm, ...
                                        output_wfm_length, ...
                                        is_circular, ...
                                        quality, ...
                                        resampling_filter)
% This funtion resamples the input waveform (inWfm) to generate a new
% waveform with a new length (outWl). New length can be longer
(upsampling)
% or shorter (downsampling) than the original one. The new waveform
can be
% selfconsistent for loop generation (isCirc == true) or not for singe
shot
% generation.

    input_wfm_length = length(input_wfm);
    % Sampling rate ratio (>1.0, upsampling)
    sampling_ratio = double(output_wfm_length) /
double(input_wfm_length);
    % If resampling filter exists it is not calculated so time is
saved
    % when calling the resampling function more than once
    if ~exist('resampling_filter', 'var') ||
isempty(resampling_filter)
        % Default parameters for resampling filter
        filter_resolution = 50000; %50000
        bw_fraction = 1.0; %0.98;
```

```
        if sampling_ratio < 1.0
            bw_fraction = 0.98;
        end
        resampling_filter = GetResamplingFilter(    quality, ...
                                                    filter_resolution,
...
                                                    bw_fraction);
    end
    % The parameters of the resampling filter are part of the
associated
    % object
    convolution_length = resampling_filter.num_of_samples;
    filter_resolution = resampling_filter.resolution;
    resampling_filter_length = length(resampling_filter.filter);
    bw_fraction = resampling_filter.bw_fraction;
    % For undersampling filter, the amplitude of the resampling filter
must
    % be corrected by the relative BW
    if sampling_ratio < 1.0
        % The distance for samples in the input (measured in samples
of the
        % output) must be corrected for undersampling as well in order
to
        % preserve SFDR
        convolution_length = floor(convolution_length /
(sampling_ratio * bw_fraction));
        resampling_filter.filter = resampling_filter.filter *
(sampling_ratio * bw_fraction);
    else
        convolution_length = floor(resampling_filter.num_of_samples /
bw_fraction);
    end

    % Output waveform is initialized to "all zeros"
    output_wfm = zeros(1, output_wfm_length);
    % Convolution loop for each output sample
    if sampling_ratio >= 1.0
        mult_factor1 = bw_fraction * filter_resolution;
    else
        mult_factor1 = bw_fraction * filter_resolution *
sampling_ratio;
    end

    for i = 0:(output_wfm_length - 1)
        % Index for the central sample to process in the input wfm
        central_sample = i / sampling_ratio;
        central_sample_int = round(central_sample);
        % Contribution for all the participating samples form the
input is
        % accumulated on the current output sample
        for j = (central_sample_int - convolution_length):...
                (central_sample_int + convolution_length)
            % Actual fractional distance to the input sample
            time_distance = abs(central_sample - j);
```

```
            % Distance is converted to a relative integer index to the
            % resampling filter (lookup table)
            time_distance = round(mult_factor1 * time_distance);

            % If convolution is circular the initial samples are used
at
            % the end and the end samples are used at the beginning.
            input_wfm_index = j;
            if is_circular
                input_wfm_index = mod(input_wfm_index,
input_wfm_length);
            end
            % If the pointer to the resampling filter is within teh
limits
            % of the lookup table, the contribution of the input
sample is
            % added to the current output sample
            if time_distance < resampling_filter_length && ...
                    input_wfm_index >=0 && ...
                    input_wfm_index < input_wfm_length
                output_wfm(i + 1) = output_wfm(i + 1) +...
                    input_wfm(input_wfm_index + 1) * ...
                    resampling_filter.filter(time_distance + 1);
            end
        end
    end
end

function output_wfm = LimitBW ( input_wfm, ...
                                bw_fraction)
    num_of_peak_samples = round(1.0 / bw_fraction);
    output_wfm = input_wfm;
    for k = 0:(length(input_wfm) - 1)
        ref_sample = k + 1;
        for j = (k - num_of_peak_samples):(k + num_of_peak_samples)
            current_sample = int32(mod(j, length(input_wfm)) + 1);
            if input_wfm(current_sample) > output_wfm(ref_sample)
                output_wfm(ref_sample) = input_wfm(current_sample);
            end
        end
    end
end

function [waveform, Fs] = Get_Wlan_ad()
% Generated by MATLAB(R) 9.14 (R2023a) and WLAN Toolbox 3.6 (R2023a).
% Generated on: 19-Apr-2023 18:52:47

    %% Generating 802.11ad waveform
    % 802.11ad configuration
    dmgCfg = wlanDMGConfig('MCS', '16', ...
        'TrainingLength', 0, ...
        'TonePairingType', 'Static', ...
        'PSDULength', 1000, ...
```

```
        'AggregatedMPDU', false, ...
        'LastRSSI', 0, ...
        'Turnaround', false);

    num_of_packets = 1;
    idle_time = 2E-6;
    % input bit source:
    in = randi([0, 1], 1000, 1);

    % Generation
    waveform = wlanWaveformGenerator(in, dmgCfg, ...
        'NumPackets', num_of_packets, ...
        'IdleTime', idle_time, ...
        'WindowTransitionTime', 6.0606e-09, ...
        'ScramblerInitialization', 2);

    Fs = wlanSampleRate(dmgCfg); % Specify the sample rate of the
waveform in Hz
end

function [waveform, Fs] = Get_Wlan_ax(oversampling)
    % 802.11ax configuration
    heSUCfg = wlanHESUConfig('ChannelBandwidth', 'CBW160', ...
        'NumTransmitAntennas', 1, ...
        'NumSpaceTimeStreams', 1, ...
        'SpatialMapping', 'Direct', ...
        'PreHESpatialMapping', false, ...
        'MCS', 5, ...
        'DCM', false, ...
        'ChannelCoding', 'LDPC', ...
        'APEPLength', 100, ...
        'GuardInterval', 3.2, ...
        'HELTFType', 4, ...
        'UplinkIndication', false, ...
        'BSSColor', 0, ...
        'SpatialReuse', 0, ...
        'TXOPDuration', 127, ...
        'HighDoppler', false, ...
        'NominalPacketPadding', 0);

    % input bit source:
    in = randi([0, 1], 10000, 1);

    num_of_packets = 1;
    idle_time = 20E-6;

    % Generation
    waveform = wlanWaveformGenerator(in, heSUCfg, ...
        'NumPackets', num_of_packets, ...
        'IdleTime', idle_time, ...
        'OversamplingFactor', oversampling, ...
        'ScramblerInitialization', 93, ...
        'WindowTransitionTime', 1e-07);
```

```
    Fs = oversampling * wlanSampleRate(heSUCfg, 'OversamplingFactor',
1);
end


function [waveform, Fs] = Get_Multi_Tone(   num_of_tones, ...
                                            offset_tone,...
                                            spacing, ...
                                            oversampling)
    % Compute maximum frequency component in the signal
    max_freq = (num_of_tones - 1) * spacing / 2.0;
    max_freq = max_freq + spacing * offset_tone;
    % Sample rate for calcualtion will be twice the maximum freq x
    % oversampling factor
    Fs = oversampling * 2.0 * max_freq;
    % Tone frequency calculation
    tone_freq = 0:(num_of_tones - 1);
    tone_freq = tone_freq - (num_of_tones - 1.0) / 2.0;
    tone_freq = spacing * tone_freq;
    tone_freq = tone_freq + spacing * offset_tone;

    % Time window will be the minimum one: 1 / spacing
    % It must be double when the number of tones is even for
symmetrical
    % spectrum around carrier frequency.
    if mod(num_of_tones,2) == 1
        time_window = 1.0 / spacing;
    else
        time_window = 2.0 / spacing;
    end

    % Waveform length must be an integer
    wfm_length = round(Fs * time_window);
    % Fs must be recalculated after rounding wavweform length
    Fs = wfm_length / time_window;
    % Time values for samples
    x_data = 0 :(wfm_length -1);
    x_data = x_data / Fs;
    % Phase distribution for PAPR reduction is selected. Newman = 2.
    tones_phase = PhaseDistribution(2, num_of_tones);
    % Waveform data is initialized to zero
    waveform = zeros(1, wfm_length);
    % The contribution of each tone is added to the waveform
    for k = 1 : num_of_tones
        waveform = waveform + ...
            exp(1i * (x_data * 2 * pi * tone_freq(k) +
tones_phase(k)));
    end
end

function phase_table = PhaseDistribution(dist_type, number_of_tones)
    switch dist_type
        case 1
            % Random
```

```
            phase_table = 2.0 * pi .* (rand(1, number_of_tones) -
0.5);

        case 2
            % Newman (near-optimal for equal amplitude tones)
            phase_table = 1:number_of_tones;
            phase_table = wrapToPi(-(pi / number_of_tones) .* ...
                ( 1.0 - phase_table .* phase_table));

        case 3
            % Rudin (near optimal for equal amplitude tones when
number of
            % tones = 2^N)
            num_of_steps = int16(round(log(number_of_tones) /
log(2)));

            if 2^num_of_steps < number_of_tones
                num_of_steps = num_of_steps + 1;
            end

            num_of_steps = num_of_steps - 1;
            phase_table(1:2) = 1;
            % Rudin sequence construction
            for n=1:num_of_steps
                m = int16(length(phase_table) / 2);
                phase_table = [phase_table, phase_table(1 : m),...
                    -phase_table(m + 1 : 2 * m)];
            end
            % Conversion to radians
            phase_table = -0.5 * pi .* (phase_table(1 :
number_of_tones) - 1);
    end
end

function [symbol] = getIqMap(data, bPerS)

    if bPerS == 5 % QAM32 mapping
        lev = 6;
        data = data + 1;
        data(data > 4) = data(data > 4) + 1;
        data(data > 29) = data(data > 29) + 1;

    elseif bPerS == 7 % QAM128 mapping
        lev = 12;
        data = data + 2;
        data(data > 9) = data(data > 9) + 4;
        data(data > 21) = data(data > 21) + 2;
        data(data > 119) = data(data > 119) + 2;
        data(data > 129) = data(data > 129) + 4;

     elseif bPerS == 9 % QAM512 mapping
        lev = 24;
        data = data + 4;
        data(data > 19) = data(data > 19) + 8;
```

```
        data(data > 43) = data(data > 43) + 8;
        data(data > 67) = data(data > 67) + 8;
        data(data > 91) = data(data > 91) + 4;
        data(data > 479) = data(data > 479) + 4;
        data(data > 499) = data(data > 499) + 8;
        data(data > 523) = data(data > 523) + 8;
        data(data > 547) = data(data > 547) + 8;
    else
        lev = 2 ^ (bPerS / 2); % QPSK, QAM16, QAM64, QAM256, QAM1024
    end

    symbI = floor(data / lev);
    symbQ = mod(data, lev);
    lev = lev / 2 - 0.5;
    symbI = (symbI - lev) / lev;
    symbQ = (symbQ - lev) / lev;
    symbol = symbI + 1i * symbQ;
end

function dataOut = getRnData(nOfS, bPerS)
    maxVal = 2 ^ bPerS;
    dataOut = maxVal * rand(1, nOfS);
    dataOut = floor(dataOut);
    dataOut(dataOut >= maxVal) = maxVal - 1;
end

function out_vector = ZeroPadding(in_vector, oversampling)
    out_vector = zeros(1, oversampling * length(in_vector));
    out_vector(1:oversampling:length(out_vector)) = in_vector;
end

function [waveform, Fs] = Get_Qam(  modulation_type, ...
                                    num_of_symbols, ...
                                    symbol_rate, ...
                                    filter_type,...
                                    roll_off, ...
                                    oversampling)
    % modType           Modulation
    % 1                 QPSK
    % 2                 QAM16
    % 3                 QAM32
    % 4                 QAM64
    % 5                 QAM128
    % 6                 QAM256
    % 7                 QAM512
    % 8                 QAM1024

    bits_per_symbol = [2, 4, 5, 6, 7, 8, 9, 10];
    bits_per_symbol = bits_per_symbol(modulation_type);
    oversampling = round(oversampling);

    % Create IQ for QPSK/QAM
```

```
    % Get symbols in the range 1..2^bps and Map to IQ as Complex
Symbol
    data = getRnData(num_of_symbols, bits_per_symbol);
    [waveform] = getIqMap(data, bits_per_symbol);

    % Adapt I/Q sample rate to the Oversampling parameter
    waveform = ZeroPadding(waveform, oversampling);

    % Calculate baseband shaping filter
    % accuracy is the length of-1 the shaping filter
    accuracy = 512;
    %filter_type = 'sqrt'; % 'normal' or 'sqrt'
    baseband_filter = rcosdesign(   roll_off, ...
                                    accuracy, ...
                                    oversampling, ...
                                    filter_type);

    % Apply filter through circular convolution and calculate Fs
    waveform = cconv(waveform, baseband_filter, length(waveform));
    Fs = symbol_rate * oversampling;
end

function [envelope_wfm, ref_envelope] = Get_Envelope(wfm_out,
smoothing_factor, minimum_pwr)
% ENVELOPE CALCULATION
    % Envelope wfm made from the module of the IQ complex wfm
    envelope_wfm = abs(wfm_out);
    % LPF
    envelope_wfm = LimitBW( envelope_wfm, smoothing_factor);
    envelope_wfm = movmean(envelope_wfm, 10);
    %envelope_wfm = LimitBW( envelope_wfm, bw_factor);
    % Minimum level processing
    minimum_pwr = max(envelope_wfm) * 10^(minimum_pwr / 20.0);
    envelope_wfm(envelope_wfm < minimum_pwr) = minimum_pwr;
    % Normalization so 0 will be mapped to the lowest DAC value and
max is
    % mapped to +1.0. wfm_out is always positive
    if max(envelope_wfm) > 0.0
        envelope_wfm = 2.0 * (envelope_wfm / max(envelope_wfm) - 0.5);
    else
        envelope_wfm = envelope_wfm + 1.0;
    end

    ref_envelope = abs(wfm_out);
    if max(ref_envelope) > 0.0
        ref_envelope = 2.0 * (ref_envelope / max(ref_envelope) - 0.5);
    else
        ref_envelope = ref_envelope + 1.0;
    end

end

function waveform = Get_Qam_Clock(  num_of_symbols, ...
                                    roll_off, ...
```

```
                                    oversampling, ...
                                    div_factor)

    oversampling = round(oversampling);
    div_factor = round(div_factor);

    % Create IQ for QPSK/QAM

    % Get symbols in the range 1..2^bps and Map to IQ as Complex
Symbol
    waveform = zeros(1, num_of_symbols);
    for k = 0:(div_factor - 1)
        waveform((k + 1):(2 * div_factor):length(waveform)) = 1.0;
    end
    for k = div_factor:(2 * div_factor - 1)
        waveform((k + 1):(2 * div_factor):length(waveform)) = -1.0;
    end

    % Adapt I/Q sample rate to the Oversampling parameter
    waveform = ZeroPadding(waveform, oversampling);

    % Calculate baseband shaping filter
    % accuracy is the length of-1 the shaping filter
    accuracy = 512;
    filter_type = 'sqrt'; % 'normal' or 'sqrt'
    baseband_filter = rcosdesign(   roll_off, ...
                                    accuracy, ...
                                    oversampling, ...
                                    filter_type);

    % Apply filter through circular convolution and calculate Fs
    waveform = cconv(waveform, baseband_filter, length(waveform));
end

function DrawEnvelope(  wfm_out, ...
                        envelope_wfm, ...
                        ref_envelope, ...
                        sample_rate_bb_out)

    % Two plots
    tiledlayout(1,2);

    x0=100;
    y0=100;
    width=1000;
    height=800;
    set(gcf,'position',[x0,y0,width,height]);

    wfm_length_out = length(wfm_out);

    nexttile;
    x_data = 0 : (wfm_length_out - 1);
    x_data = x_data / sample_rate_bb_out;
    plot(x_data, real(wfm_out));
```

```
    hold;
    plot(x_data, imag(wfm_out));
    title(strcat('IQ Waveform:', num2str(wfm_length_out),' samples
@',...
        num2str(sample_rate_bb_out / 1E6), 'MS/s'));
    xlabel('Seconds');

    nexttile;
    plot(x_data, ref_envelope);
    hold;
    plot(x_data, envelope_wfm);
    ylim([-1.0 1.1]);
    title(strcat('Envelope Waveform:', num2str(wfm_length_out),'
samples @',...
        num2str(sample_rate_bb_out / 1E6), 'MS/s'));
    xlabel('Seconds');
end

function DrawEyeDiagram(    eye_width, ...
                            max_symbol_shown, ...
                            sample_rate, ...
                            symbol_rate, ...
                            roll_off, ...
                            wfm_in, ...
                            clock_wfm)

    % For better graph accuracy, samples per symbol > = 100
    interpol_factor = ceil(sample_rate / symbol_rate);
    if interpol_factor < 100
        interpol_factor = ceil(100 / interpol_factor);
        new_wfm_length = interpol_factor * length(wfm_in);
        wfm_in = myResampling(wfm_in, new_wfm_length, true, 60);
        clock_wfm = myResampling(clock_wfm, new_wfm_length, true, 60);
        sample_rate = interpol_factor * sample_rate;
    end
    % Graph data definition
    size_window_in_samples = ceil(eye_width / symbol_rate *
sample_rate);
    size_window_in_samples = ceil(size_window_in_samples / 2);
    symbol_shift = round(0.5 / symbol_rate * sample_rate);
    % Zero crossing for clock signal
    zero_crossings = zeros(1, max_symbol_shown);
    previous_state = clock_wfm(1);

    filter_type = 'sqrt'; % 'normal' or 'sqrt'
    baseband_filter = rcosdesign(roll_off, 60, ...
        round(sample_rate / symbol_rate) , filter_type);
    baseband_filter = baseband_filter / sum(baseband_filter);

    % Zero crossing processing
    counter = 1;

    for k = 2:length(clock_wfm)
        if clock_wfm(k) >= 0.0 && previous_state <= 0.0 ||...
```

```
            clock_wfm(k) <= 0.0 && previous_state >= 0.0
            zero_crossings(counter) = k - 1;
            previous_state = clock_wfm(k);
            if counter > max_symbol_shown
                break;
            else
                counter = counter + 1;
            end
        end
    end

    % Four plots
    tiledlayout(2,2);

    x0 = 100;
    y0 = 100;
    width = 1000;
    height = 800;
    set(gcf,'position',[x0,y0,width,height]);

    nexttile;
    plot(wfm_in);
    hold;
    const_diagram = wfm_in(zero_crossings(2:counter - 2) +
symbol_shift);
    scatter(real(const_diagram), imag(const_diagram), 20, [1, 1, 0],
'filled');

    max_ampl = max([max(abs(real(wfm_in))), max(abs(imag(wfm_in)))]);

    xlim([-max_ampl max_ampl]);
    ylim([-max_ampl max_ampl]);
    title('Constellation Unfiltered');

    nexttile;

    base_x_data = -size_window_in_samples:1:size_window_in_samples;
    base_x_data = base_x_data / sample_rate;
    hold_flag = true;

    for k = 1:counter
        if (zero_crossings(k) - size_window_in_samples) >= 1 &&...
                (zero_crossings(k) + size_window_in_samples) <=
length(wfm_in)
            plot(   base_x_data, ...
                    real(wfm_in(zero_crossings(k) -
size_window_in_samples + symbol_shift:...
                                zero_crossings(k) +
size_window_in_samples + symbol_shift)));
            if hold_flag
                hold on;
                hold_flag = false;
            end
        end
```

```
    end

    title('Eye Diagram Unfiltered');
    xlabel('Symbol Period');

    nexttile;
    % Apply filter through circular convolution
    clock_wfm = cconv(clock_wfm, baseband_filter, length(clock_wfm));
    % Apply filter through circular convolution and calculate Fs
    wfm_in = cconv(wfm_in, baseband_filter, length(wfm_in));

    counter = 1;

    for k = 2:length(clock_wfm)
        if clock_wfm(k) >= 0.0 && previous_state < 0.0 ||...
                clock_wfm(k) < 0.0 && previous_state >= 0.0
            zero_crossings(counter) = k;
            previous_state = clock_wfm(k);
            if counter > max_symbol_shown
                break;
            else
                counter = counter + 1;
            end
        end
    end

    plot(wfm_in);
    const_diagram = wfm_in(zero_crossings(2:counter - 2) +
symbol_shift);
    hold;
    scatter(real(const_diagram), imag(const_diagram), 20, [1, 1, 0],
'filled');

    max_ampl = max([max(abs(real(wfm_in))), max(abs(imag(wfm_in)))]);

    xlim([-max_ampl max_ampl]);
    ylim([-max_ampl max_ampl]);

    title('Constellation Filtered');

    nexttile;

    hold_flag = true;

    for k = 1:counter
        if (zero_crossings(k) - size_window_in_samples) >= 1 &&...
                (zero_crossings(k) + size_window_in_samples) <=
length(wfm_in)
            plot(base_x_data, real(wfm_in(zero_crossings(k) -
size_window_in_samples + symbol_shift:...
                zero_crossings(k) + size_window_in_samples +
symbol_shift)));
            if hold_flag
                hold on;
```

```
                hold_flag = false;
            end
        end
    end

    title('Eye Diagram Filtered');
    xlabel('Symbol Period');
end

function [  inst,...
            admin,...
            modelName,...
            sId] = ConnecToProteus( cType, ...
                                    connStr, ...
                                    paranoia_level)

% Connection to target Proteus
% cType specifies API. "LAN" for VISA, "DLL" for PXI
% connStr is the slot # as an integer(0 for manual selection) or IP
adress
% as an string
% Paranoia Level add additional checks for each transfer. 0 = no
checks.
% 1 = send OPC?, 2 = send SYST:ERROR?

% It returns
% inst: handler for the selected instrument
% admin: administrative handler
% modelName: string with model name for selected instrument (i.e.
"P9484")
% sId: slot number for selected instrument

    pid = feature('getpid');
    fprintf(1,'\nProcess ID %d\n',pid);

    dll_path = 'C:\\Windows\\System32\\TEPAdmin.dll';
    admin = 0;
    sId = 0;
    if cType == "LAN"
        try
            connStr = strcat('TCPIP::',connStr,'::5025::SOCKET');
            inst = TEProteusInst(connStr, paranoia_level);

            res = inst.Connect();
            assert (res == true);
            modelName = identifyModel(inst);
        catch ME
            rethrow(ME)
        end
    else
        asm = NET.addAssembly(dll_path);

        import TaborElec.Proteus.CLI.*
        import TaborElec.Proteus.CLI.Admin.*
```

```
        import System.*

        admin = CProteusAdmin(@OnLoggerEvent);
        rc = admin.Open();
        assert(rc == 0);

        try
            slotIds = admin.GetSlotIds();
            numSlots = length(size(slotIds));
            assert(numSlots > 0);

            % If there are multiple slots, let the user select one ..
            sId = slotIds(1);
            if numSlots > 1
                fprintf('\n%d slots were found\n', numSlots);
                for n = 1:numSlots
                    sId = slotIds(n);
                    slotInfo = admin.GetSlotInfo(sId);
                    if ~slotInfo.IsSlotInUse
                        modelName = slotInfo.ModelName;
                        if slotInfo.IsDummySlot && connStr == 0
                            fprintf(' * Slot Number:%d Model %s [Dummy
Slot].\n', sId, modelName);
                        elseif connStr == 0
                            fprintf(' * Slot Number:%d Model %s.\n',
sId, modelName);
                        end
                    end
                end
                pause(0.1);
                if connStr == 0
                    choice = input('Enter SlotId ');
                    fprintf('\n');
                else
                    choice = connStr;
                end
                sId = uint32(choice);
                slotInfo = admin.GetSlotInfo(sId);
                modelName = slotInfo.ModelName;
                modelName = strtrim(netStrToStr(modelName));
            end

            % Connect to the selected instrument ..
            should_reset = true;
            inst = admin.OpenInstrument(sId, should_reset);
            instId = inst.InstrId;

        catch ME
            admin.Close();
            rethrow(ME)
        end
    end
end
```

```
function model = identifyModel(inst)
    idnStr = inst.SendScpi('*IDN?');
    idnStr = strtrim(netStrToStr(idnStr.RespStr));
    idnStr = split(idnStr, ',');

    if length(idnStr) > 1
        model = idnStr(2);
    else
        model ='';
    end
end

function options = getOptions(inst)
    optStr = inst.SendScpi('*OPT?');
    optStr = strtrim(netStrToStr(optStr.RespStr));
    options = split(optStr, ',');
end

function [str] = netStrToStr(netStr)
    try
        str = convertCharsToStrings(char(netStr));
    catch
        str = '';
    end
end

function retval = myQuantization (myArray, dacRes, minLevel)

    maxLevel = 2 ^ dacRes - 1;
    numOfLevels = maxLevel - minLevel + 1;

    retval = round((numOfLevels .* (myArray + 1) - 1) ./ 2);
    retval = retval + minLevel;

    retval(retval > maxLevel) = maxLevel;
    retval(retval < minLevel) = minLevel;

end

function outWfm = Interleave2(wfmI, wfmQ)

    wfmLength = length(wfmI);
    if length(wfmQ) < wfmLength
        wfmLength =  length(wfmQ);
    end

    %wfmLength = 2 * wfmLength;
    outWfm = uint8(zeros(1, 2 * wfmLength));

    outWfm(1:2:(2 * wfmLength - 1)) = wfmI;
    outWfm(2:2:(2 * wfmLength)) = wfmQ;
end

function outWfm = formatWfm2(inWfm1, inWfm2)
```

```
%formatWfm2 This function formats data for two I/Q streams to be
dwnloaded
%to a single segment in Proteus to be generated in the IQM Mode 'TWO'
%   All waveforms must be properly normalized to the -1.0/+1.0 range.
%   All waveforms must have the same length

    % Formatting requires to go through the following steps:
    %   1) quantize samples to 16-bit unsigned integers
    %   2) swap the LSB and MSB as MSBs will be sent first for this
mode
    %   3) convert the uint16 array to an uint8 array of twice the
size
    % Final wfm is MSB, LSB, MSB, LSB,...
    inWfmI1 = typecast(swapbytes(uint16(myQuantization(real(inWfm1),
16, 1))),'uint8');
    inWfmQ1 = typecast(swapbytes(uint16(myQuantization(imag(inWfm1),
16, 1))),'uint8');
    inWfmI2 = typecast(swapbytes(uint16(myQuantization(real(inWfm2),
16, 1))),'uint8');
    inWfmQ2 = typecast(swapbytes(uint16(myQuantization(imag(inWfm2),
16, 2))),'uint8');
    % Sequence MSBI1, MSBQ1, MSBQ2, MSBI2, LSBI1, LSBQ1, LSBQ2, LSBI2
    % This is done in three interleaving steps
    outWfmI = Interleave2(inWfmI1, inWfmQ2);
    outWfm = Interleave2(inWfmQ1, inWfmI2);
    outWfm = Interleave2(outWfmI, outWfm);

    % Format as 16 bit integers as this is how waveforms are
transferred
    outWfm = uint16(outWfm(1:2:length(outWfm))) + ...
            256 * uint16(outWfm(2:2:length(outWfm)));
end

function shifted_vector = ShiftVector(input_wfm, shifts)

    vector_l = length(input_wfm);
    shifts = shifts - 1;
    shifted_vector = input_wfm(mod((1:vector_l) + shifts, vector_l) +
1);
end

function zeroed_vector = InsertZeros(input_vector, isEven)

    if isEven
        zeroed_vector = zeros(1, 2 * length(input_vector));
    else
        zeroed_vector = zeros(1, 2 * length(input_vector) - 1);
    end

    zeroed_vector(1:2:length(zeroed_vector)) = input_vector;
end

function [interpol_filter, max_response] =
GetProteusInterpolFilter(interpolation_factor)
```

```
    basic_2x_filter_taps = [6, 0, -19, 0, 47, 0, -100, 0, 192, 0, -
342, 0,...
    572, 0, -914, 0, 1409, 0, -2119, 0, 3152, 0, -4729, 0, 7420, 0,...
    -13334, 0, 41527, 65536, 41527, 0, -13334, 0, 7420, 0, -4729,
0,...
    3152, 0, -2119, 0, 1409, 0, -914, 0, 572, 0, -342, 0, 192, 0, -
100,...
    0, 47, 0, -19, 0, 6];

    switch interpolation_factor

        case 2
            interpol_filter = basic_2x_filter_taps;

        case 4
            interpol_filter = InsertZeros(basic_2x_filter_taps,
false);
            interpol_filter = conv(interpol_filter,
basic_2x_filter_taps);

        case 8
            interpol_filter = InsertZeros(basic_2x_filter_taps,
false);
            interpol_filter = conv(interpol_filter,
basic_2x_filter_taps);
            interpol_filter = InsertZeros(interpol_filter, false);
            interpol_filter = conv(interpol_filter,
basic_2x_filter_taps);
    end

    % Filter normalization for 0dB gain at 0Hz
    interpol_filter = interpol_filter/sum(interpol_filter);
    interpol_filter = interpolation_factor * interpol_filter;

    % Worst case maximum output abs(amplitude) for
    max_response = 0.0;

    for k = 0:(interpolation_factor - 1)
        current_max_response =
sum(abs(interpol_filter((k+1):interpolation_factor:length(interpol_fil
ter))));
        if current_max_response > max_response
            max_response = current_max_response;
        end
    end
end

function output_wfm = MyProteusInterpolation(input_wfm,
interpol_factor, apply_norm)
      % Function used in traditional resampling
      % Expansion by zero-padding
      output_wfm = zeros(1, interpol_factor * length(input_wfm));
      output_wfm(1:interpol_factor:end) = input_wfm;
      % "Ideal" Interpolation filter
```

```
      [interpol_filter, max_response] =
GetProteusInterpolFilter(interpol_factor);
      shifts = floor(length(interpol_filter) / 2);

      %convolution
      output_wfm = cconv(output_wfm, interpol_filter,
length(output_wfm));
      output_wfm = ShiftVector(output_wfm, shifts);
      if apply_norm
          output_wfm = input_wfm /  max(abs(output_wfm));
      end
end

function outWfm = NormalIq(wfm)
    maxPwr = max(abs(wfm));
    outWfm = wfm / maxPwr;
end

function [outWfm1,  outWfm2] = NormalIq2(wfm1, wfm2)
    maxPwr = max(abs(wfm1) + abs(wfm2));
    outWfm1 = wfm1 / maxPwr;
    outWfm2 = wfm2 / maxPwr;
end

function outWfm = Interleave(wfmI, wfmQ)

    wfmLength = length(wfmI);
    outWfm = zeros(1, 2 * wfmLength);

    outWfm(1:2:(2 * wfmLength - 1)) = wfmI;
    outWfm(2:2:(2 * wfmLength)) = wfmQ;
end
```

13.5 Using the Digitizer to Capture Baseband and RF Signals

13.5.1 Programming Example 4 shows the following:

1. The way to use the Task List and sequencing in the generation side.
2. The usage of the ADC with or without using DDC
3. The way to synchronize the DUC and the DDC for coherent demodulation. A pulsed RF signal is used as a convenient signal to show the previous items.

This example generates a simple radar RF pulse using the AWG, captures it using the digitizer, and analyzes the captured waveforms. Pulse parameters such as pulse width, pulse repetition interval (PRI), and carrier frequency can be arbitrarily defined. The RF pulse is generated using the DUC by any channel of the AWG section of Proteus using IQ mode ONE (refer to 13.4.1 Programming Example 3). The modulating signal consists of a square pulse in the I component, while the Q component is set to all zeros. This application example takes the pulse parameters and the AWG requirements in terms of sample rate, interpolation factor, minimum segment length and granularity to minimize waveform memory by defining the signal through the sequencing of several simpler segments. It also automates the creation of the task list resulting in the defined pulse using the lowest number of segments with the smallest possible size. The sequence also generates an internal task trigger to trigger the acquisition of multiple frames by the digitizer. This trigger signal is aligned with the beginning of the transmission of the pulse (see the figure below).

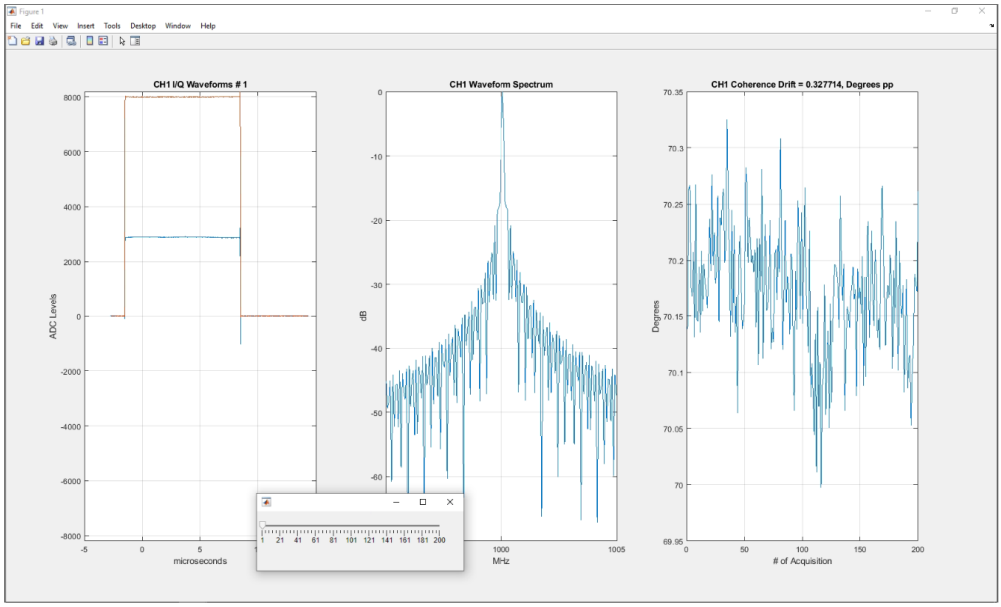

Figure 13-11 Radar pulse analysis of one of the acquired frames. The graph in the left shows the demodulated (by the DDC) I and Q signals for the selected frame. The one in the center shows the FFT of the complex demodulated signal. The graph in the right shows the evolution in time of the DUC to the DDC phase for all frames. The peak-to-peak excursion is shown in the title of the graph. The MATLAB slider control at the bottom allows for the frame selection. This acquisition has been made while the NCOs in the DDC and the DUC work in the coherent mode.

The digitizer-related functionality captures any number of frames. Each frame is automatically set-up to capture the pulse section of the signal. Each frame is triggered by the internal task trigger generated by the task list being executed by the AWG section. The acquisition of the RF signal can be made in the direct mode so the complete modulated signal will be stored in the memory (Figure 13-13), or using the DDC (Digital Down Converter), so a demodulated pulse is stored as a complex IQ signal (Figure 13-12, Figure 13-13).

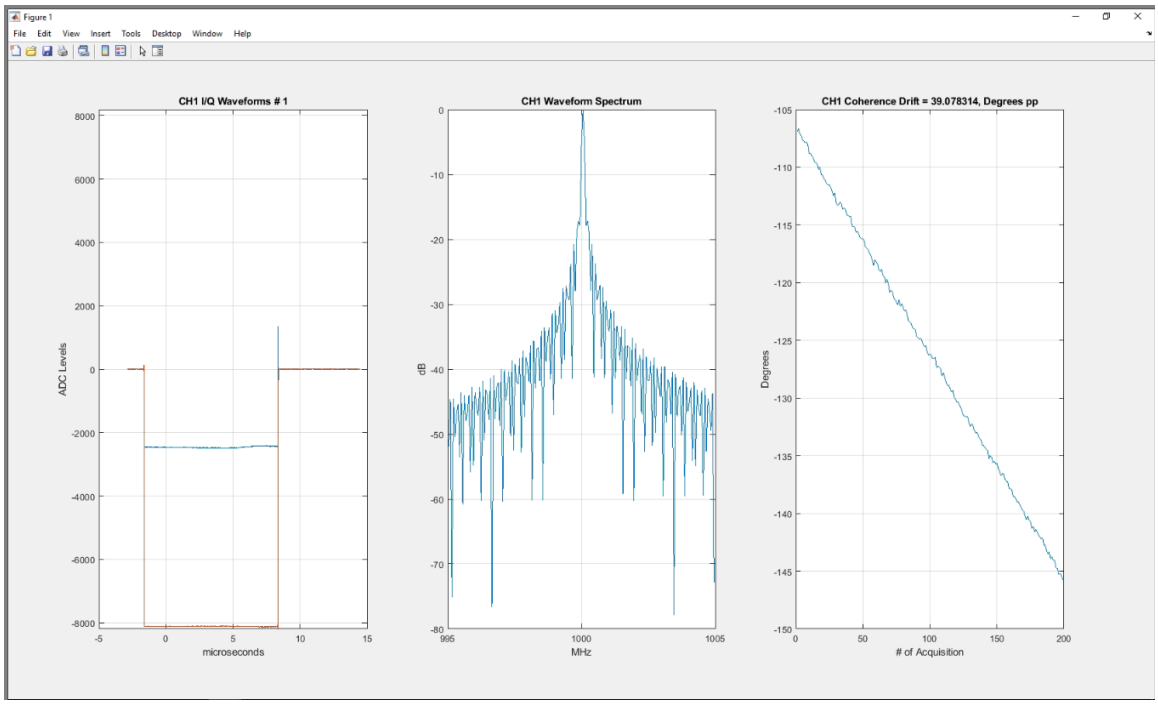

Figure 13-12 Radar pulse analysis when the DUC and DDC NCOs do not work in the coherent mode. The graph in the right shows the linear evolution of the phase caused by the tiny frequency difference between the NCOs in the transmitter and the receiver. The way the pulse is split between the I and Q components (shown in the left), will change significantly depending on the selected frame.

The NCO frequencies of the DUC and DDC are set to the same frequency, so modulation and demodulation is applied to the same frequency. These NCOs can work independently, so they will not run coherently (Figure 13-12), or in the synchronous mode (Figure 13-11), when TX and RX work coherently so the phase between the pulse being generated and the one being demodulated is constant and deterministic. The application shows the received frames (with one pulse each), its spectrum, and if coherence analysis is activated, the TX to RX pulse phase for all the captured frames. Coherence analysis is only possible when the DDC is enabled. The frame shown in the graph can be selected through an MATLAB slider control giving access to any of the individual frames.

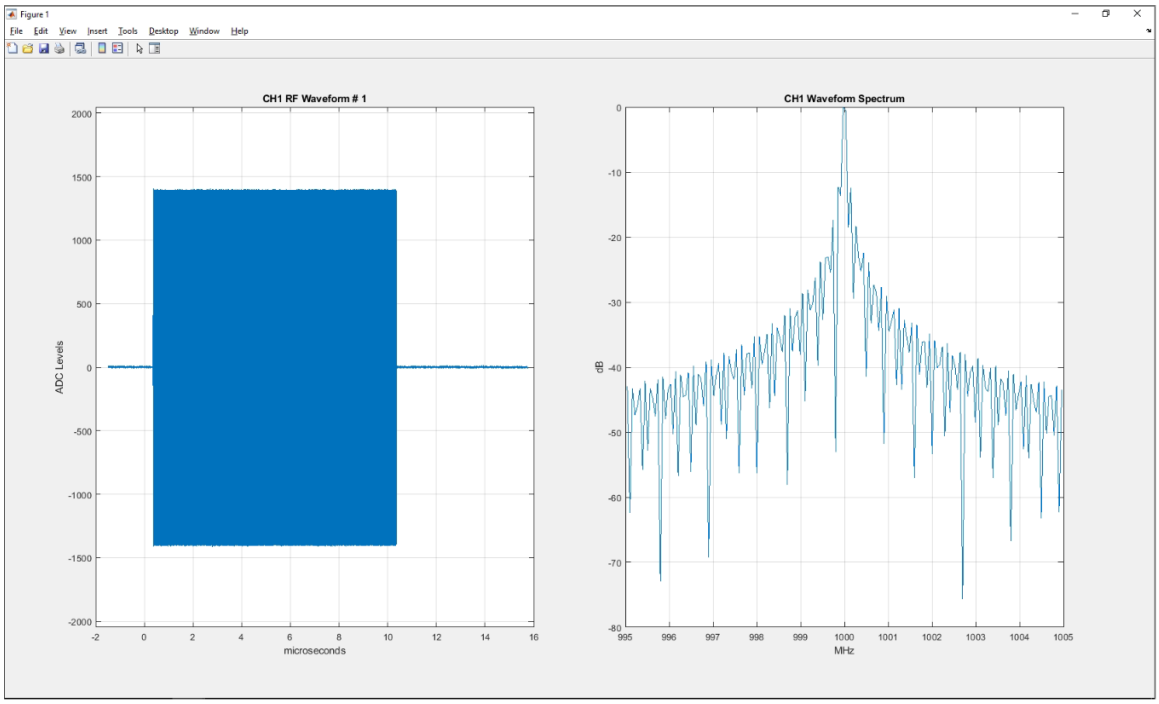

Figure 13-13 Radar pulse analysis of one of the acquired frames when the digitizer works in the direct (non-DDC) mode. The graph in the left shows the captured pulse including the carrier information at full sampled rate (without decimation). The one in the right shows the spectrum of the waveform by performing an FFT on the real data containing the modulated RF signal. The coherence analysis does not make any sense when the DDC is not used so the corresponding graph is not shown.

13.5.1 Programming Example 4 includes useful functions as listed below.

- ***SendIqmOneSeq***: Setup Proteus for the IQ ONE mode and format and download I and Q waveforms for multiple segments at once. It also sets up the corresponding task list as defined by the user.
- ***GetDigitizerData***: It sets up any number of ADC channels for direct or DDC based acquisitions and returns the captured waveforms as a bi-dimensional array with N rows and M columns, where N is the number of captured frames and M the number of samples in a frame. Samples are retuned as real numbers for direct acquisition and as complex numbers for DDC-based acquisitions.
- ***TaskListSetup***: This function creates and configures a complete task list after the contents of a simple two-dimensional integer array coding all the basic fields for each task in the task list.
- ***GetPulseWfms***: It automatically defines a modulating pulse and minimizes waveform memory usage by splitting it in minimum length segments and defining the task list (in a format compatible with the ***TaskListSetup*** function) that implements the right timing.

13.5.1 Programming Example 4

```
% Using the Digitizer to Capture Baseband and RF Signals
% Pulse Radar Demo with DUC and DDC
% It generates a pulsed RF signal using the DUC and optimizes waveform
% memory usage using sequencing. It captures multiple frames of the RF
```

```
% waveform directly or using the DDC (Digital Down Converter). DUC/DDC
% can be synchronized and TX to RX phase drift can be shown in either
case.
% It shows all the acquisitions (acquisition # cbe selected usiong an
slide
% control).
clear;
close all;
clear variables;
clear global;
clc;

% Define IP Address for Target Proteus device descriptor
% VISA "Socket-Based" TCP-IP Device. Socket# = 5025
ipAddr = '127.0.0.1'; %'127.0.0.1'= Local Host; % your IP here
pxiSlot = 0;

% Instrument setup
cType               = "LAN";  %"LAN" = VISA or "DLL" = PXI

if cType == "LAN"
    connPar         = ipAddr;
else
    connPar         = pxiSlot; % Your slot # here, o for manual
selection
end

paranoia_level = 0; % 0, 1 or 2
% Open Session and load libraries
[inst, admin, model, slotNumber] = ConnecToProteus(cType, connPar,
paranoia_level);

% Report model
fprintf('Connected to: %s, slot: %d\n', model(1), slotNumber(1));

% DDC and DUC activation
% ======================
use_ddc             = false;     % true = DDC on, false = DDC off
sync_nco            = true;     % true = Coherent acq
get_coherence       = true;     % true = get coherence (only when DDC
on)
%=======================

use_dtr_trigger     = true;     % Digitizer will be triggered by AWG

%AWG Settings
awg_channel         = 1;
segment             = 1;
awg_sampling_rate   = 9E+09;    % Sampling rate for the target AWG
dac_res             = 16;       % Default DAC resolution
awg_granularity     = 32;       % Waveform length granularity
min_segment_length  = 64;       % Minimum Segment Size

awg_out_lvl         = 0.4;      % Amplitude level for the AWG
```

```
awg_interpol        = 8;        % Interolation factor for DUC
apply6db            = true;     % true = twice the amplitude
% Digitizer Settings
digChannel          = 1;
adcRange            = 3;
adc_granularity     = 48;
adc_resolution      = 12;
adc_sampling_rate   = 2.7E+09;  % Overwritten if sync_nco = true
num_of_acquisitions = 1;        % Just one set of frames
num_of_frames       = 200;      % Number of frames per acquisition
pre_trigger         = 10;       % Size of the pretrigger in % of frame
size

if use_ddc
    adc_interpol    = 16;       % This is the decimation factor for
DDC
else
    adc_interpol    = 1;        % This is the decimation factor for
Direct
    get_coherence   = false;    % If no DDC, coherence cannot be
analyzed
    sync_nco        = false;
end

fprintf('RADAR DEMO STARTS\n');
% RF Pulse Parameters
pulse_width         = 10.0E-6;  % Pulse Width
pulse_rep_frequency = 5.0E+03;  % PRI
carrier_freq        = 1.0E+09;  % Carrier Frequency
num_of_pulses       = 1;        % Num of pulses in the sequence
% Delay for each digitizer channel
delay               = 0.0E-06;  % ADC Trigger Delay set to zero

% The Carrier Frequency for the ADC must be an integer submultiple
(x1, x2,
% x4, x8) of the AWG sampling rate. Four (4) is selected in this case.
if sync_nco
    adc_sampling_rate = awg_sampling_rate / 4;
end



% Frame length calculation for ADC according to pulse width amd
pretrigger
% setting.
frame_length = adc_sampling_rate * pulse_width * 1.5;
pre_trigger = frame_length * pre_trigger / 100.0;
pre_trigger = pre_trigger / adc_interpol;
pre_trigger = round (pre_trigger /adc_granularity) * adc_granularity;
frame_length = frame_length / adc_interpol;
frame_length = frame_length + 1.5 * pre_trigger;
frame_length = ceil(frame_length / adc_granularity) * adc_granularity;

fprintf('Reset instrument ..\n');
```

```
inst.SendScpi('*CLS; *RST');

% Pulse RF Signal Calculation and Download

% The final waveform will be generated by a sequence specified in a
% task list as follows:
%
%   Task #      segment             num of repetitions
% ____________________________________________________

%   TASK1       pulse_section       0 or 1
%   TASK2       pulse_section       repeat_pulse_section - 1
%   TASK3       trans_section       0 or 1
%   TASK4       off_section         repeat_off_section
%
% Depending on the characteristics of the pulsed waveform any of the
% Tasks listed above may or not exist. If repeat_pulse_section = 1,
% then TASK3 will not exists. If repeat_pulse_section = 0, neither
task
% nor task 2 will exist and Task 3 will become the first task in the
% list carrying the full pulse. this only happens when duration of the
% pulse is lower than the minimum segment length. Task 3 may not exist
% if the overall length of the waveform is a multiple of the minimum
% segment length and the length of the pulse is also a multiple of it.
% Task 4 may bot exist in case Task1, 2 and 3 already implements the
% complete waveform.

[myWfm,...
map_wfm,...
task_list] = GetPulseWfms(  awg_sampling_rate, ...
                            awg_interpol, ...
                            awg_granularity, ...
                            min_segment_length, ...
                            pulse_width, ...
                            pulse_rep_frequency);
% Download all segments, build task list, set up DDC, and activate
task list
result = SendIqmOneSeq(     inst,...
                            awg_sampling_rate,...
                            awg_interpol,...
                            awg_channel,...
                            awg_out_lvl,...
                            segment,...
                            carrier_freq,...
                            0.0,...
                            apply6db,...
                            myWfm,...
                            map_wfm,...
                            task_list);

% ------------------------------------
% Operate the ADC with direct functions
% ------------------------------------
```

```
%DIGITIZER SECTION

% Coherent operation of NCOs in DUC and DDC activation if required
if sync_nco
    inst.SendScpi(':DIG:DDC:CLKS AWG');
end
% Get all frames from digitizer
fprintf('Acquired Waveforms Upload to Computer Starts\n');

acqWfm = GetDigitizerData(  inst,...
                            cType,...
                            true,...
                            digChannel,...
                            adcRange,...
                            use_ddc,...
                            use_dtr_trigger,...
                            awg_channel,...
                            0,...
                            0.025,...
                            adc_sampling_rate,...
                            carrier_freq,...
                            frame_length,...
                            num_of_frames,...
                            pre_trigger);

fprintf('Acquired Waveforms Upload to Computer Ends\n');

% Coherence processing
% One phase value is obtained for each frame so evolution can be
observed
phase1 = zeros(1, num_of_frames);

for i = 1:num_of_frames
    %for each acquisition, complex IQ waveforms are extracted
    samples1 = acqWfm(i,:);
    % Relative amplitude of the I and Q pulses are obtained by
integration
    I1 = sum(real(samples1));
    Q1 = sum(imag(samples1));
    % 4 quadrant arctangent is applied to the areas of teh I and Q
pulses
    phase1(i) = atan2(Q1,I1);
end
phase1 = unwrap(phase1);
phase1 = 180.0 * phase1 / pi;

itemToShow = num_of_frames;

if use_ddc
    actual_pre_trigger = 2 * pre_trigger;
else
    actual_pre_trigger = pre_trigger;
end
```

```
% Span for spectrum graph
span = 100.0 / pulse_width;
% Show data for first frame
ShowResults(    use_ddc,...
                acqWfm,...
                get_coherence,...
                phase1,...
                1,...
                actual_pre_trigger,...
                carrier_freq,...
                span,...
                adc_interpol,...
                adc_sampling_rate);
% ------------------------------------------------------------------------
-
% End of the example
% ------------------------------------------------------------------------
-

fprintf('\nRADAR DEMO COMPLETED\n');

% GUI to access all the information about all frames is generated
fig = uifigure('Position',[100 100 350 100]);
% Frame is selected using an slider control
sld = uislider(fig,...
    'Position',[10 75 300 3],...
    'Limits', [1 itemToShow],...
    'ValueChangedFcn',@(sld,event) ShowResults( use_ddc,...
                                                acqWfm,...
                                                get_coherence,...
                                                phase1,...
                                                sld.Value,...
                                                actual_pre_trigger,...
                                                carrier_freq,...
                                                span,...
                                                adc_interpol,...
                                                adc_sampling_rate));


% Close the session
% It is recommended to disconnect from instrument at the end
if cType == "LAN"
    inst.Disconnect();
else
    admin.CloseInstrument(inst.instId);
    admin.Close();
end

% ************************************************************************
% *                              FUNCTIONS                               *
% ************************************************************************

function ShowResults(   useDdc,...
                        acqWfm,...
```

```
                    showCoh,...
                    phase1,...
                    itemToShow,...
                    preTrig,...
                    cFreq,...
                    span,...
                    ddcFactor,...
                    samplingRateDig)

itemToShow = round(itemToShow);
numOfFrames = length(acqWfm(:,1));

if itemToShow < 1
    itemToShow = 1;
end

if itemToShow > numOfFrames
    itemToShow = numOfFrames;
end

if useDdc
else
    cFreq = abs(GetNcoFreq(cFreq, samplingRateDig, false));
end

samples1 = acqWfm(itemToShow,:);

if useDdc
else
    samples1 = samples1(1:ddcFactor:length(samples1));
end

if showCoh
    tiledlayout(1,3);
else
    tiledlayout(1,2);
end

% Top plot
ax1 = nexttile;
xData = 0:(length(samples1) - 1);
xData = xData - preTrig;
xData = xData / samplingRateDig;

if useDdc
    xData = xData * ddcFactor;
end

xData = xData * 1e+06;
if useDdc
    plot(ax1, xData, real(samples1), xData, imag(samples1));
else
    dc_level = mean(samples1);
    plot(ax1, xData, samples1 - dc_level);
```

```
    end

    if useDdc
        title(ax1,sprintf('CH1 I/Q Waveforms # %d', itemToShow));
    else
        title(ax1,sprintf('CH1 RF Waveform # %d', itemToShow));
    end

    xlabel('microseconds')
    ylabel('ADC Levels')
    if useDdc
        ylim([-8192 8192]); % 15 bits
    else
        ylim([-2048 2048]); % 12Bits
    end
    grid(ax1,'on')

    % Mid plot
    ax3 = nexttile;
    pSpec = abs(fft(samples1));
    if useDdc
        pSpec = circshift(pSpec,round(length(pSpec) /2));
        startF = - span/2;
        stopF = startF + span;

        xData = 0:(length(samples1) - 1);
        xData = xData * samplingRateDig / (length(samples1) *
ddcFactor);
        xData = xData - xData(round(length(pSpec) /2));

        xData1 = xData(find(xData >= startF & xData <= stopF));
        pSpec = pSpec(find(xData >= startF & xData <= stopF));
        pSpec = 20 * log10(pSpec / max(pSpec));

        xData1 = xData1 / 1e+06 + cFreq / 1e+06;

    else
        startF = cFreq - span/2;
        stopF = startF + span;

        xData = 0:(length(samples1) - 1);
        xData = xData * samplingRateDig / length(samples1);

        xData1 = xData(find(xData >= startF & xData <= stopF));
        pSpec = pSpec(find(xData >= startF & xData <= stopF));
        pSpec = 20 * log10(pSpec / max(pSpec));

        xData1 = xData1 / 1e+06;
    end

    plot(ax3, xData1, pSpec);

    title(ax3,'CH1 Waveform Spectrum')
```

```
    xlabel('MHz')
    ylabel('dB')
    grid(ax3,'on')

    if showCoh
        % Phase drift and coherence plot
        ax5 = nexttile;

        plot(ax5, phase1);

        title(ax5,sprintf('CH1 Coherence Drift = %f, Degrees pp',
max(phase1) - min(phase1)));


        xlabel('# of Acquisition')
        ylabel('Degrees')
        grid(ax5,'on')
    end
end

function outWfm = ComplexToInterleaved(wfmIq)
    wfmLength = length(wfmIq);
    outWfm = zeros(1, 2 * wfmLength);

    outWfm(1:2:(2 * wfmLength - 1)) = real(wfmIq);
    outWfm(2:2:(2 * wfmLength)) = imag(wfmIq);
end

function outWfm = InterleavedToComplex(wfmIq)
    wfmLength = length(wfmIq);

    outWfm = double(wfmIq(1:2:wfmLength)) + 1.0i *
double(wfmIq(2:2:wfmLength));
end

function ncoFreq = GetNcoFreq(carrierFreq ,samplingRate,
remove_second_nyquist)
% This function maps any frequency to its image in the first or second
% Nyquist Zone. Second Nyquist Zone can be excluded when
% remove_second_nyquist is set to 'true'
    ncoFreq = abs(carrierFreq);
    ncoFreq = mod(ncoFreq, samplingRate);

    if remove_second_nyquist
        if ncoFreq > samplingRate / 2
            ncoFreq = samplingRate - ncoFreq;
        end
    end
end

function wfmData = GetDigitizerData(inst,...
                                    cType,...
                                    dualMode,...
                                    adChan,...
```

```
                                        range,...
                                        useDdc,...
                                        useDtrTrig,...
                                        dtrChan,...
                                        dtrDelay,...
                                        trgLevel,...
                                        samplingRateDig,...
                                        cFreq,...
                                        frameLen,...
                                        numberOfFrames,...
                                        preTrig)
    % ----------
    % ADC Config
    % It supports up to two channels with different setups
    % ----------

    % Set the ADC mode and set the channel mapping
    if dualMode
        inst.SendScpi(':DIG:MODE DUAL');
        if length(adChan) > 1
            adChan = adChan(1:2);
            if adChan(1) == adChan(2)
                adChan = adChan(1);
            end
        end
    else
        inst.SendScpi(':DIG:MODE SINGLE');
        % Only Channel 1 in Single mode
        adChan = 1;
    end

    adChan(adChan > 2) = 2;
    adChan(adChan < 1) = 1;

    numOfChannels = length(adChan);

    % Free Acquistion Memory and Set sampling rate
    inst.SendScpi(':DIG:ACQ:FREE');
    inst.SendScpi(sprintf(':DIG:FREQ %g', samplingRateDig));
    % DDC activation
    if useDdc
        inst.SendScpi(':DIG:DDC:MODE COMP');
        for i = 1:numOfChannels
            % NCO frequency mapped to the first and second NZ
            ddcFreq = GetNcoFreq(cFreq(mod(i, length(cFreq)) + 1),
samplingRateDig, false);
            inst.SendScpi([sprintf(':DIG:DDC:CFR%d ', adChan(i))
num2str(abs(ddcFreq))]);
        end
    end
    % Calculate actual frame length depending on the DDC mode
    actualFrameLen = frameLen;
    if useDdc
        actualFrameLen = 2 * actualFrameLen;
```

```
    end

    % ADC Range
    % 1:LOW, 2:MED, 3:HIGH
    range(range > 2) = 3;
    range(range < 2) = 1;

    for chan = 1:numOfChannels
        % Select digitizer channel:
        inst.SendScpi(sprintf(':DIG:CHAN %d', adChan(chan)));
        % Set the voltage-range of the selected channel
        switch range(mod(chan - 1, length(range)) + 1)
            case 1
                inst.SendScpi(':DIG:CHAN:RANG LOW');
            case 2
                inst.SendScpi(':DIG:CHAN:RANG MED');
            case 3
                inst.SendScpi(':DIG:CHAN:RANG HIGH');
        end
        %Enable acquisition in the selected channel
        inst.SendScpi(':DIG:CHAN:STATE ENAB');

        % Setup frames layout. Common to both ADC channels.
        inst.SendScpi(sprintf(':DIG:ACQ:DEF %d, %d',...
            numberOfFrames, actualFrameLen));

        % Set channel 1 of the digitizer as its trigger source
        % If DTR trigger, it is directed to the designated AWG channel
        if useDtrTrig(mod(chan - 1, length(useDtrTrig)) + 1)
            % DTR trigger must be assigned after selecting the target
AWG
            % channel as it is a property of the AWG channel and the
ADC
            % channel
            inst.SendScpi(sprintf(':INST:CHAN %d',...
                dtrChan(mod(chan - 1, length(dtrChan)) + 1)));
            inst.SendScpi(sprintf(':DIG:TRIG:SOURCE TASK%d',...
                dtrChan(mod(chan - 1, length(dtrChan)) + 1)));
            % Set DTR trigger Dealy
            inst.SendScpi(sprintf(':DIG:TRIG:AWG:TDEL %f',...
                dtrDelay));
        else
            % Level trigger set tup
            inst.SendScpi(sprintf(':DIG:TRIG:SOUR CH%d', adChan(i)));
            inst.SendScpi(sprintf(':DIG:TRIG:SELF %f',...
                trgLevel(mod(chan - 1, length(trgLevel)) + 1)));
%0.025
        end
        % Pretrigger for DDC must be set to double as acquisions are
made
        % by IQ pair of samples
        if useDdc
            actualPreTrig = 2 * preTrig;
        else
```

```
            actualPreTrig = preTrig;
        end

        inst.SendScpi(sprintf(':DIG:PRET %d', actualPreTrig));

        % Select which frames are filled with captured data
        %(all frames in this example)
        inst.SendScpi(':DIG:ACQ:FRAM:CAPT:ALL');

        % Delete all wfm memory
        inst.SendScpi(':DIG:ACQ:ZERO:ALL');
        % Get ADC wfm format. For informative purposes
        resp = inst.SendScpi(':DIG:DATA:FORM?');
        resp = strtrim(netStrToStr(resp.RespStr));
    end

    % Stop the digitizer
    inst.SendScpi(':DIG:INIT OFF');
    % And start for a new acquisition
    wfmData = zeros(numOfChannels, frameLen);
    inst.SendScpi(':DIG:INIT ON');
    % Read Acquired Wfm Data
    for i=1:numOfChannels
        % Select channel
        inst.SendScpi(sprintf(':DIG:CHAN %d', adChan(i)));
        % Get acquisition status CSV string from Proteus for selected
        % channel
        for n = 1:25000
            resp = inst.SendScpi(':DIG:ACQ:FRAM:STAT?');
            resp = strtrim(netStrToStr(resp.RespStr));
            resp = strtrim(resp);
            items = split(resp, ',');
            items = str2double(items);
            % If item 2 in the CSV string is '1', then all frames have
been
            % captured
            if length(items) >= 3 && items(2) == 1
                break
            end
            % This is just to give some information when trigger times
out
            if mod(n, 10) == 0
                fprintf('%d. %s Time:\n', fix(n / 10), resp);
            end
            pause(0.1);
        end

        % Define what we want to read
        % (frames data, frame-header, or both).
        % In this example we read the frames-data
        inst.SendScpi(':DIG:DATA:TYPE FRAM');
        inst.SendScpi(':DIG:DATA:SEL ALL');

        % Read binary block
```

```
        % Get the size in bytes of the acquisition
        resp = inst.SendScpi(':DIG:DATA:SIZE?');
        resp = strtrim(netStrToStr(resp.RespStr));
        num_bytes = str2double(resp);
        % upload time will be shown so transfer rate can be compared
        fprintf('ADC Upload Time for %d bytes:\n', num_bytes);

        if cType == "LAN"
            if useDdc
                tic;
                % DDC data is formatted as 15-bit in a 32-bit unsigned
                % integer
                samples = inst.ReadBinaryData(':DIG:DATA:READ?',
'uint32');
                toc;
                samples = int32(samples) - 16384; % Set zero level
                % Convert to complex I + jQ samples
                samples = InterleavedToComplex(samples);
                % Invert spectrum if ddcFreq < 0.0
                if ddcFreq < 0.0
                    samples = conj(samples);
                end
            else
                tic;
                % Direct ADC data is formated as 12-bit samples in 16-
bit
                % unsigned integers
                samples = inst.ReadBinaryData(':DIG:DATA:READ?',
'uint16');
                toc;
                samples = int16(samples) - 2048; % Set zero level
            end
        else
            % For the PXI library, downloads can only handlw 8 or 16-
bit
            % unsigned.
            if useDdc
                % For DDC, because read format is UINT16 we divide
byte
                % number by 2
                wavlen = floor(num_bytes / 2);
                % allocate NET array
                netArray = NET.createArray('System.UInt16', wavlen);
                % read the captured frame
                tic;
                res = inst.ReadMultipleAdcFrames(i - 1, 1,
numberOfFrames, netArray);
                toc;

                assert(res == 0);
                % Each 32 sample is now 2 contiguous 16-bit samples
                samples = uint16(netArray);
                % As the first 16-bit samples in the pair is "all
zeros"
```

```
                % they can be discarded by taking one very two bytes
                samples = samples(1:2:length(samples));

                % cast to matlab vector
                samples = int16(samples) - 16384; % Set zero level
                % Convert to complex I + jQ samples
                samples = InterleavedToComplex(samples);
                % Invert spectrum if ddcFreq < 0.0
                if ddcFreq < 0.0
                    samples = conj(samples);
                end
                % deallocate the NET array
                delete(netArray);
            else
                wavlen = floor(num_bytes / 2);

                % allocate NET array
                netArray = NET.createArray('System.UInt16', wavlen);
                % read the captured frame
                tic
                res = inst.ReadMultipleAdcFrames(0, 1, numberOfFrames,
netArray);
                toc
                assert(res == 0);

                samples = uint16(netArray);
                % cast to matlab vector
                samples = int16(samples) - 2048;

                % deallocate the NET array
                delete(netArray);
            end
        end
        % Ouput data is formatted as a two dimensions array with A x F
        % rows (A = number of acquisitions, F = number of Frames) and
        % FrameLem columns
        for j=1:numberOfFrames
            wfmData((i - 1) * numberOfFrames + j,:) = samples(((j-1) *
frameLen + 1):(j * frameLen));
        end
    end
    % Sttop digitizer after all acquisitions and frames for all the
    % channels have been captured
    inst.SendScpi(':DIG:INIT OFF');
end

function [  inst,...
            admin,...
            modelName,...
            sId] = ConnecToProteus( cType, ...
                                    connStr, ...
                                    paranoia_level)

% Connection to target Proteus
```

```
% cType specifies API. "LAN" for VISA, "DLL" for PXI
% connStr is the slot # as an integer(0 for manual selection) or IP
adress
% as an string
% Paranoia Level add additional checks for each transfer. 0 = no
checks.
% 1 = send OPC?, 2 = send SYST:ERROR?

% It returns
% inst: handler for the selected instrument
% admin: administrative handler
% modelName: string with model name for selected instrument (i.e.
"P9484")
% sId: slot number for selected instrument

    pid = feature('getpid');
    fprintf(1,'\nProcess ID %d\n',pid);

    dll_path = 'C:\\Windows\\System32\\TEPAdmin.dll';
    admin = 0;
    sId = 0;
    if cType == "LAN"
        try
            connStr = strcat('TCPIP::',connStr,'::5025::SOCKET');
            inst = TEProteusInst(connStr, paranoia_level);

            res = inst.Connect();
            assert (res == true);
            modelName = identifyModel(inst);
        catch ME
            rethrow(ME)
        end
    else
        asm = NET.addAssembly(dll_path);

        import TaborElec.Proteus.CLI.*
        import TaborElec.Proteus.CLI.Admin.*
        import System.*

        admin = CProteusAdmin(@OnLoggerEvent);
        rc = admin.Open();
        assert(rc == 0);

        try
            slotIds = admin.GetSlotIds();
            numSlots = length(size(slotIds));
            assert(numSlots > 0);

            % If there are multiple slots, let the user select one ..
            sId = slotIds(1);
            if numSlots > 1
                fprintf('\n%d slots were found\n', numSlots);
                for n = 1:numSlots
                    sId = slotIds(n);
```

```
                    slotInfo = admin.GetSlotInfo(sId);
                    if ~slotInfo.IsSlotInUse
                        modelName = slotInfo.ModelName;
                        if slotInfo.IsDummySlot && connStr == 0
                            fprintf(' * Slot Number:%d Model %s [Dummy
Slot].\n', sId, modelName);
                        elseif connStr == 0
                            fprintf(' * Slot Number:%d Model %s.\n',
sId, modelName);
                        end
                    end
                end
                pause(0.1);
                if connStr == 0
                    choice = input('Enter SlotId ');
                    fprintf('\n');
                else
                    choice = connStr;
                end
                sId = uint32(choice);
                slotInfo = admin.GetSlotInfo(sId);
                modelName = slotInfo.ModelName;
                modelName = strtrim(netStrToStr(modelName));
            end

            % Connect to the selected instrument ..
            should_reset = true;
            inst = admin.OpenInstrument(sId, should_reset);
            instId = inst.InstrId;

        catch ME
            admin.Close();
            rethrow(ME)
        end
    end
end

function result = SendWfmToProteus( inst,...
                                    samplingRate,...
                                    channel,...
                                    segment,...
                                    myWfm,...
                                    dacRes,...
                                    initialize)

    if dacRes == 16
            inst.SendScpi(':TRAC:FORM U16');
    else
            inst.SendScpi(':TRAC:FORM U8');
    end

    %Select Channel
    if initialize
        inst.SendScpi(':TRAC:DEL:ALL');
```

```
        inst.SendScpi([':FREQ:RAST ' num2str(samplingRate)]);
    end

    inst.SendScpi(sprintf(':INST:CHAN %d', channel));
    inst.SendScpi(sprintf(':TRAC:DEF %d, %d', segment,
length(myWfm)));
    % select segmen as the the programmable segment
    inst.SendScpi(sprintf(':TRAC:SEL %d', segment));

    % format Wfm
%     myWfm = myQuantization(myWfm, dacRes, 1);

    % Download the binary data to segment
    prefix = ':TRAC:DATA 0,';

    if (dacRes==16)
        myWfm = uint16(myWfm);
        myWfm = typecast(myWfm, 'uint8');
        dataLength = 2 * length(myWfm);
    else
        myWfm = uint8(myWfm);
        dataLength = length(myWfm);
    end

    fprintf('AWG Download Time for %d bytes:\n', dataLength);
    tic;
    res = inst.WriteBinaryData(prefix, myWfm);
    toc;
    assert(res.ErrCode == 0);

    if initialize
        inst.SendScpi(sprintf(':SOUR:FUNC:MODE:SEGM %d', segment))
        % Output voltage set to MAX
        inst.SendScpi(':SOUR:VOLT MAX');
        % Activate outpurt and start generation
        inst.SendScpi(':OUTP ON');
    end

    result = length(myWfm);
end

function TaskListSetup( inst,...
                        channel,...
                        segment,...
                        taskList)

% This function defines a complete Task List based in the definitons
% defined by the taskList integer uint32 array. The array is NxM,
where N
% is the number of entries in the task list being defined and M is the
% number of fields specifying the parameters of each tasks. This is
the
% defintion of the fields:
%
```

```
%   Number          Description
% _____________________________________________
%
%   1               pointer_to_segment
%   2               task_type: 0: Single, 1: StartSeq, 2: InSeq, 3:
EndSeq
%   3               num_of_loops_task
%   4               num_of_loops_seq
%   5               next_task
%   6               dtr_trig_flag: 0: dtr trigger off, 1: dtr trigger
on

    num_of_tasks = size(taskList, 1);
    % Select Channel
    inst.SendScpi(sprintf(':INST:CHAN %d', channel));
    % The Task Composer is configured to handle a certain number of
task
    % entries
    inst.SendScpi(sprintf(':TASK:COMP:LENG %d', num_of_tasks));

    % Then, each task is defined
    for task_number = 1:num_of_tasks
        % Task to be defined is selected
        inst.SendScpi(sprintf(':TASK:COMP:SEL %d', task_number));
        % The type of task is defined.
        switch taskList(task_number, 2)
            case 0
                inst.SendScpi(':TASK:COMP:TYPE SING');
            case 1
                inst.SendScpi(':TASK:COMP:TYPE STAR');
            case 2
                inst.SendScpi(':TASK:COMP:TYPE SEQ');
            case 3
                inst.SendScpi(':TASK:COMP:TYPE END');
        end

        % The action to take after completing the task is defined.
NEXT is the
        % default so sending this command is not mandatory
        inst.SendScpi(':TASK:COMP:DEST NEXT');
        % Assigns segment for task in the sequence
        inst.SendScpi(sprintf(':TASK:COMP:SEGM %d',
taskList(task_number, 1) + segment - 1));
        % Assigns task to generate next in the sequence
        inst.SendScpi(sprintf(':TASK:COMP:NEXT1 %d',
taskList(task_number, 5)));

        % Set the Trigger for Digitizer
        switch taskList(task_number, 6)
            case 0
                inst.SendScpi(':TASK:COMP:DTR OFF');
            otherwise
                inst.SendScpi(':TASK:COMP:DTR ON');
        end
```

```
        % Num of loops for the current sequence
        if taskList(task_number, 2) == 1
            inst.SendScpi(sprintf(':TASK:COMP:SEQ %d',
taskList(task_number, 4)));
        end

        %  Num of loops for the current task
        inst.SendScpi(sprintf(':TASK:COMP:LOOP %d',
taskList(task_number, 3)));
    end

    % The task table created with the Composer is written to the
actual task
    % table of teh selected channel
    inst.SendScpi(':TASK:COMP:WRIT');
    fprintf(1, 'SEQUENCE CREATED!\n');
    % Select Task Mode for generation
    % Start in task #1 (#1 is the default)
    inst.SendScpi(':FUNC:MODE TASK');
end

function result = SendIqmOneSeq(    inst,...
                                    samplingRate,...
                                    interpol,...
                                    channel,...
                                    awg_out_lvl,...
                                    segment,...
                                    cfr,...
                                    phase,...
                                    apply6db,...
                                    myWfm,...
                                    segMap,...
                                    taskList)
% This function handles the dwonload of multiple segments and also the
task
% list to generate them with the task sequencer. Waveform data are
passed as
% concatenated waveforms in a single vector. segMap is a vector with
the
% size for each segment so they can be extracted from myWfm. Assigned
% segment is always in the order they are defined in the myWfm array
% starting with the segment variable. taskList is a two dimensional
array.
% See the TaskListSetup functon for more information

    % format Wfm
    dacRes = 16;
    myWfm = NormalIq(myWfm);
    % myWfm is originally a complex vector
    myWfm = ComplexToInterleaved(myWfm);
    myWfm = myQuantization(myWfm, dacRes, 1);

    % Select Channel
```

```
    inst.SendScpi(sprintf(':INST:CHAN %d', channel));
    % Setup Sclk for initial DUC settings
    inst.SendScpi([':FREQ:RAST ' num2str(2.5E9)]);
    % Interpolation factor for I/Q waveforms
    switch interpol
        case 2
            inst.SendScpi(':SOUR:INT X2');

        case 4
            inst.SendScpi(':SOUR:INT X4');

        case 8
            inst.SendScpi(':SOUR:INT X8');
    end

    % DAC Mode set to 'DUC' and IQ Modulation mode set to 'ONE'
    inst.SendScpi(':MODE DUC');
    inst.SendScpi(':IQM ONE');
    % Set the definitive Sclk
    inst.SendScpi([':FREQ:RAST ' num2str(samplingRate)]);
    fprintf(1, sprintf('DOWNLOADING %d WAVEFORMS: %d samples\n',...
        length(segMap) , length(myWfm)));
    segMap = 2 * segMap;
    pointer_wfm = 1;
    for k = 1:length(segMap)
        % For each segment, the right waveform data is selected and
        % downloaded
        result = SendWfmToProteus(  inst,...
                                    samplingRate,...
                                    channel,...
                                    k + segment - 1,...
                                    myWfm(pointer_wfm:(pointer_wfm +
segMap(k) - 1)),...
                                    dacRes,...
                                    false);
        % pointer for next segment data is updated
        pointer_wfm = pointer_wfm + segMap(k);
    end

    fprintf(1, 'WAVEFORMS DOWNLOADED!\n');

    % task List setup
    TaskListSetup(  inst,...
                    channel,...
                    segment,...
                    taskList);

    % Select segment for generation
    fprintf(1, 'SETTING AWG OUTPUT\n');
    % Output voltage
    inst.SendScpi(sprintf(':SOUR:VOLT %d', awg_out_lvl));
    %inst.SendScpi(':SOUR:VOLT 0.4');

    % NCO set-up
```

```
    % 6dB IQ Modulation gain applied
    if apply6db
        inst.SendScpi(':NCO:SIXD1 ON');
    else
        inst.SendScpi(':NCO:SIXD1 OFF');
    end
    % NCO frequency and phase setting
    inst.SendScpi(sprintf(':NCO:CFR1 %d', cfr));
    inst.SendScpi(sprintf(':NCO:PHAS1 %d', phase));

    % Activate output and start generation
    inst.SendScpi(':OUTP ON');
end

function outWfm = NormalIq(wfm)
% Normalization to peak modulus
    maxPwr = max(abs(wfm));
    outWfm = wfm / maxPwr;
end

function retval = myQuantization (myArray, dacRes, minLevel)

    maxLevel = 2 ^ dacRes - 1;
    numOfLevels = maxLevel - minLevel + 1;

    retval = round((numOfLevels .* (myArray + 1) - 1) ./ 2);
    retval = retval + minLevel;

    retval(retval > maxLevel) = maxLevel;
    retval(retval < minLevel) = minLevel;
end

function [  myWfm,...
            map_wfm,...
            task_list] = GetPulseWfms(  awg_sampling_rate, ...
                                        awg_interpol, ...
                                        awg_granularity, ...
                                        min_segment_length, ...
                                        pulse_width, ...
                                        pulse_rep_frequency)
% This function calculates a baseband pulse and creates an optimal
% segmentation of the waveform and the corresponding task list to
generate
% the overall pulsed waveform. These requires one, two or three
segments.
% There may be a ON segment, an OFF segment, and a Transition segment.

    % Baseband waveform sample rate
    actual_sampling_rate = awg_sampling_rate / awg_interpol;
    % Actual Granularity and Minimum Segement Length for complex wfms
    awg_granularity = awg_granularity / 2;
    min_segment_length = min_segment_length / 2;
    % Duration of one complete cycle
    total_duration = 1.0 / pulse_rep_frequency;
```

```
    % Wfm length in samples rounde to the actual granularity
    awg_wfm_length = total_duration * actual_sampling_rate;
    awg_wfm_length = awg_granularity * floor(awg_wfm_length /
awg_granularity);

    % I/Q complex pulse
    % =================
    % Pulse length is samples
    pulse_length = round(pulse_width * actual_sampling_rate);
    % Number of segments with minimum segment length
    repeat_pulse_section = floor(pulse_length / min_segment_length);
    % Pulse segment is "all 1s"
    pulse_section = complex(ones(1, min_segment_length));

    % I/Q interval between pulses
    % ===========================
    % Total length for the all zeros section in samples
    off_section_length = awg_wfm_length - pulse_length;
    % Number of segments with minimum segment length
    repeat_off_section = floor(off_section_length /
min_segment_length);
    % Pulse segment is "all 1s"
    off_section = complex(zeros(1, min_segment_length));

    % I/Q transition segment
    % ======================
    % Total length for the trabsition section in samples
    trans_section_length = awg_wfm_length -...
        min_segment_length * (repeat_pulse_section +
repeat_off_section);
    % Number of ones in the transition section
    ones_in_trans_section = pulse_length -...
            repeat_pulse_section * min_segment_length;

    % Transition segment calculation
    if trans_section_length ~= 0
        trans_section = complex(zeros(1, trans_section_length));
        if ones_in_trans_section ~= 0
            trans_section(1:ones_in_trans_section) = complex(1.0);
        end
        % Transition length must be corrected in case its length is
shorter
        % than the minimum segment length by adding either a pulse or
an
        % off segment so the number of reps must be edited
        if trans_section_length < min_segment_length
            if repeat_off_section > 0
                trans_section = [trans_section, off_section];
                repeat_off_section = repeat_off_section - 1;
            elseif repeat_pulse_section > 0
                trans_section = [pulse_section, trans_section];
                repeat_pulse_section = repeat_pulse_section - 1;
            end
        end
```

```
    end
    % Meaning of fields in task array
    %   1               pointer_to_segment
    %   2               task_type: 0: Single, 1: StartSeq, 2: InSeq,
3: EndSeq
    %   3               num_of_loops_task
    %   4               num_of_loops_seq
    %   5               next_task
    %   6               dtr_trig_flag: 0: dtr trigger off, 1: dtr
trigger on

    % The number of waveforms may be one, two, or three
    % The number of tasks may be one, two, three, o four
    num_of_wfms = 0;
    myWfm = complex(double.empty);
    map_wfm = uint32.empty;
    num_of_tasks = 0;

    if repeat_pulse_section > 0
        num_of_wfms = num_of_wfms + 1;
        myWfm = [myWfm, pulse_section];
        map_wfm = [map_wfm, min_segment_length];
        % If there must be more than one repetition of the pulse
segment,
        % there must be an additional task so the DTR trigger happens
only
        % once in a waveform cycle.
        if repeat_pulse_section == 1
            num_of_tasks = num_of_tasks + 1;
        else
            num_of_tasks = num_of_tasks + 2;
        end
    end
    if length(trans_section) >= 1
        num_of_wfms = num_of_wfms + 1;
        myWfm = [myWfm, trans_section];
        map_wfm = [map_wfm, trans_section_length];
        num_of_tasks = num_of_tasks + 1;
    end
    if repeat_off_section > 0
        num_of_wfms = num_of_wfms + 1;
        myWfm = [myWfm, off_section];
        map_wfm = [map_wfm, min_segment_length];
        num_of_tasks = num_of_tasks + 1;
    end
    % Task list two dimensional array is created
    task_list = uint32(zeros(num_of_tasks, 6));
    % And then populated for proper sequencing, pulse timing, PRI, and
    % continuous geneation
    task_pointer = 1;
    segment_pointer = 1;

    if repeat_pulse_section > 0
        % Pulse segment is always generated once in the first task
```

```
        task_list(task_pointer, 1) = segment_pointer;
            task_list(task_pointer, 2) = 0;
            task_list(task_pointer, 3) = 1;
            task_list(task_pointer, 5) = task_pointer + 1;
            task_list(task_pointer, 6) = 1;
            task_pointer = task_pointer + 1;
        % If number of repetitions is larger than one, the second task
must
        % be added with the number of repetitions - 1 as task #1 takes
care
        % of the first repetition
        if repeat_pulse_section > 1
            task_list(task_pointer, 1) = segment_pointer;
            task_list(task_pointer, 2) = 0;
            task_list(task_pointer, 3) = repeat_pulse_section - 1;
            task_list(task_pointer, 5) = task_pointer + 1;
            task_list(task_pointer, 6) = 0;
            task_pointer = task_pointer + 1;
        end
        segment_pointer = segment_pointer + 1;
    end

    if length(trans_section) >= 1
        task_list(task_pointer, 1) = segment_pointer;
        task_list(task_pointer, 2) = 0;
        task_list(task_pointer, 3) = 1;
        task_list(task_pointer, 5) = task_pointer + 1;
        % Transition segments may be repeated zero or one time. When
there
        % is no pulse segment, the transition segment will be
generated
        % first and it has to generate the DTR trigger
        if repeat_pulse_section == 0
            task_list(task_pointer, 6) = 1;
        else
            task_list(task_pointer, 6) = 0;
        end
        task_pointer = task_pointer + 1;
        segment_pointer = segment_pointer + 1;
    end

    if repeat_off_section > 0
        % The off segment may be repeated zero or more times
        task_list(task_pointer, 1) = segment_pointer;
        task_list(task_pointer, 2) = 0;
        task_list(task_pointer, 3) = repeat_off_section;
        task_list(task_pointer, 5) = 1;
        task_list(task_pointer, 6) = 0;
    end
    % The last task always point yo task # 1
    task_list(task_pointer, 5) = 1;

end
```

```
function model = identifyModel(inst)
    idnStr = inst.SendScpi('*IDN?');
    idnStr = strtrim(netStrToStr(idnStr.RespStr));
    idnStr = split(idnStr, ',');

    if length(idnStr) > 1
        model = idnStr(2);
    else
        model ='';
    end
end

function [str] = netStrToStr(netStr)
    try
        str = convertCharsToStrings(char(netStr));
    catch
        str = '';
    end
end
```