Contents

HP E1445A AFG Module User’s Manual

Warranty ................................................. 13
WARNINGS ............................................. 14
Safety Symbols ......................................... 14
Declaration of Conformity ............................. 15
Reader Comment Sheet ............................... 17

Chapter 1. Getting Started ............................. 19
Chapter Contents ....................................... 19
Preparation for Use ..................................... 19
  VXIbus Factory Settings ............................ 20
  The AFG Logical Address .......................... 21
  Addressing the AFG (External Controller and PC) .... 22
  Setting the AFG Servant Area .................... 23
  The AFG Bus Request Level ...................... 24
  AFG Installation in a Mainframe ................. 25
Instrument Language (SCPI) ......................... 26
  SCPI Programming .................................. 26
  Command Coupling .................................. 27
Program Languages .................................... 29
  HP BASIC Language Programs .................. 29
  Visual BASIC Language Programs Using HP SICL ... 32
  Visual C/C++ Language Programs Using HP SICL .... 40
Introductory Programs ................................. 46
  AFG Self-Test ....................................... 46
  HP BASIC Program Example (SLFTST) ............ 47
  Resetting and Clearing the AFG .................. 47
  HP BASIC Program Example (RSTCLS) .......... 47
  Querying the Power-On/Reset Configuration ...... 48
  HP BASIC Program Example (LRN) ............... 48
  Checking for Errors ............................... 49
  HP BASIC Program Example (ERRORCHK) ...... 49
  Generating Sine Waves ............................. 51
  HP BASIC Program Example (RSTSINE) ........... 51

Chapter 2. Generating Standard Waveforms .......... 53
Chapter Contents ....................................... 53
Standard Waveforms Flowchart ...................... 54
Generating DC Voltages .............................. 56
  HP BASIC Program Example (DCVOLTS) ........... 56
Generating Sine Waves ............................... 58
  HP BASIC Program Example (SINEWAVE) .......... 59
Generating Square Waves ............................ 61
  HP BASIC Program Example (SQUWAVE) .......... 62
Generating Triangle/Ramp Waves .................... 65
<table>
<thead>
<tr>
<th>Chapter 3. Generating Arbitrary Waveforms</th>
<th>83</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter Contents</td>
<td>83</td>
</tr>
<tr>
<td>Arbitrary Waveforms Flowchart</td>
<td>84</td>
</tr>
<tr>
<td>How the AFG Generates Arbitrary Waveforms</td>
<td>86</td>
</tr>
<tr>
<td>Generating a Simple Arbitrary Waveform</td>
<td>88</td>
</tr>
<tr>
<td>HP BASIC Program Example (ARBWAVE)</td>
<td>91</td>
</tr>
<tr>
<td>Executing Several Waveform Segments</td>
<td>93</td>
</tr>
<tr>
<td>HP BASIC Program Example (MULSEG)</td>
<td>96</td>
</tr>
<tr>
<td>Using Different Frequency Generators</td>
<td>99</td>
</tr>
<tr>
<td>HP BASIC Program Example (AFGGEN1)</td>
<td>102</td>
</tr>
<tr>
<td>HP BASIC Program Example (AFGGEN2)</td>
<td>104</td>
</tr>
<tr>
<td>Sample Programs</td>
<td>104</td>
</tr>
<tr>
<td>Generating a Sin(X)/X Waveform</td>
<td>105</td>
</tr>
<tr>
<td>HP BASIC Program Example (SIN_X)</td>
<td>105</td>
</tr>
<tr>
<td>Generating a Damped Sine Wave</td>
<td>107</td>
</tr>
<tr>
<td>HP BASIC Program Example (SIN_D)</td>
<td>107</td>
</tr>
<tr>
<td>Generating an Exponential Charge/Discharge Waveform</td>
<td>108</td>
</tr>
<tr>
<td>HP BASIC Program Example (CHARGE)</td>
<td>108</td>
</tr>
<tr>
<td>Generating a Sine Wave with Spikes</td>
<td>109</td>
</tr>
<tr>
<td>HP BASIC Program Example (SPIKES)</td>
<td>109</td>
</tr>
<tr>
<td>Generating a 1/2 Rectified Sine Wave</td>
<td>111</td>
</tr>
<tr>
<td>HP BASIC Program Example (SIN_R)</td>
<td>111</td>
</tr>
<tr>
<td>Generating Noise</td>
<td>112</td>
</tr>
<tr>
<td>HP BASIC Program Example (NOISE)</td>
<td>112</td>
</tr>
<tr>
<td>Arbitrary Waveform Program Comments</td>
<td>113</td>
</tr>
<tr>
<td>Determining the Amount of Segment and Sequence Memory</td>
<td>113</td>
</tr>
<tr>
<td>How to Free Segment and Sequence Memory</td>
<td>113</td>
</tr>
<tr>
<td>Amplitude Effects on Voltage Lists</td>
<td>113</td>
</tr>
<tr>
<td>Using DAC Codes to Send Segment Data</td>
<td>114</td>
</tr>
<tr>
<td>Sending Segment Sequences</td>
<td>114</td>
</tr>
<tr>
<td>Reference Oscillator Sources</td>
<td>115</td>
</tr>
</tbody>
</table>
Chapter 4. Sweeping and Frequency-Shift Keying

Sweeping and Frequency Lists
- Sweeping Using Start and Stop Frequencies
  - HP BASIC Program Example (SMPLSWP1)
- Specifying a Frequency List
  - HP BASIC Program Example (LIST1)
- Sweeping Using Start and Span Frequencies
  - HP BASIC Program Example (SMPLSWP2)
- Frequency Lists Using Definite and Indefinite Length Arbitrary Blocks
  - HP BASIC Program Example (LISTDEF)
- Logarithmic Sweeping
  - HP BASIC Program Example (LOG_SWP)
- Sweep Points Versus Time
  - HP BASIC Program Example (SWP_PVST)
- Frequency Lists Versus Time
  - HP BASIC Program Example (LIST_TME)
- Sweeping Arbitrary Waveforms
  - HP BASIC Program Example (SWP_ARB)
- AC Output Leveling
  - HP BASIC Program Example (SWP_LEVL)
- Frequency-Shift Keying
  - FSK Using the "FSK" Control Source
    - HP BASIC Program Example (FSK1)
  - FSK Using the TTLTrg<n> Control Source
    - HP BASIC Program Example (FSK2)
  - FSK Using an Arbitrary Waveform
    - HP BASIC Program Example (FSK_ARB)
- Sweeping and FSK Program Comments
  - Reference Oscillator Sources
  - Sample Sources
  - AFG Frequency Modes
  - Frequency Range: Sweeping and Sampling
  - Frequency Range: Frequency Lists and FSK
  - Sweep Count and Frequency List Repetition Count
  - Arbitrary Block Data
  - Frequency Points
  - Sweep Spacing
  - Sweep Direction
### Chapter 5. Arming and Triggering

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter Contents</td>
<td>163</td>
</tr>
<tr>
<td>The ARM-TRIG Configuration</td>
<td>164</td>
</tr>
<tr>
<td>The ARM-TRIG States</td>
<td>164</td>
</tr>
<tr>
<td>Initiating Waveforms</td>
<td>165</td>
</tr>
<tr>
<td>Arming the AFG</td>
<td>165</td>
</tr>
<tr>
<td>Arming Commands</td>
<td>165</td>
</tr>
<tr>
<td>Setting Arming Sources</td>
<td>166</td>
</tr>
<tr>
<td>HP BASIC Program Example (EXT_ARM)</td>
<td>167</td>
</tr>
<tr>
<td>Setting the Arm and Waveform Cycle Count</td>
<td>169</td>
</tr>
<tr>
<td>HP BASIC Program Example (BURST)</td>
<td>170</td>
</tr>
<tr>
<td>Triggering the AFG</td>
<td>172</td>
</tr>
<tr>
<td>Triggering Commands</td>
<td>172</td>
</tr>
<tr>
<td>Using the Divide-by-N Frequency Generator</td>
<td>174</td>
</tr>
<tr>
<td>HP BASIC Program Example (DIV_N)</td>
<td>174</td>
</tr>
<tr>
<td>Lock-Stepping Multiple AFGs</td>
<td>176</td>
</tr>
<tr>
<td>HP BASIC Program Example (LOCKSTEP)</td>
<td>177</td>
</tr>
<tr>
<td>Using Stop Triggers</td>
<td>180</td>
</tr>
<tr>
<td>HP BASIC Program Example (STOPTRIG)</td>
<td>181</td>
</tr>
<tr>
<td>Gating Trigger Signals</td>
<td>183</td>
</tr>
<tr>
<td>HP BASIC Program Example (GATE)</td>
<td>184</td>
</tr>
<tr>
<td>Arming and Triggering Frequency Sweeps and Lists</td>
<td>186</td>
</tr>
<tr>
<td>Frequency Sweeps Using Triggers</td>
<td>186</td>
</tr>
<tr>
<td>HP BASIC Program Example (SWP_TRIG)</td>
<td>188</td>
</tr>
<tr>
<td>Arming and Triggering a Frequency Sweep</td>
<td>190</td>
</tr>
<tr>
<td>HP BASIC Program Example (SWP_STEP)</td>
<td>191</td>
</tr>
<tr>
<td>Arming and Triggering a Frequency List</td>
<td>193</td>
</tr>
<tr>
<td>HP BASIC Program Example (LIST_STP)</td>
<td>194</td>
</tr>
<tr>
<td>Aborting Waveforms</td>
<td>196</td>
</tr>
<tr>
<td>Using ABORT, Stop Triggers, or Gating</td>
<td>196</td>
</tr>
<tr>
<td>Arming and Triggering Program Comments</td>
<td>197</td>
</tr>
<tr>
<td>Reference Oscillator Sources</td>
<td>197</td>
</tr>
<tr>
<td>AFG Frequency Synthesis Modes</td>
<td>197</td>
</tr>
<tr>
<td>AFG Frequency Modes</td>
<td>198</td>
</tr>
<tr>
<td>AFG Arming Sources</td>
<td>199</td>
</tr>
<tr>
<td>AFG Arm Count</td>
<td>199</td>
</tr>
<tr>
<td>Waveform Repetition Count</td>
<td>199</td>
</tr>
<tr>
<td>Stop Trigger Sources</td>
<td>199</td>
</tr>
<tr>
<td>External Stop Trigger Slope</td>
<td>200</td>
</tr>
<tr>
<td>AFG Gating Sources</td>
<td>200</td>
</tr>
<tr>
<td>AFG Gate Polarity</td>
<td>200</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Chapter 8. Command Reference</td>
<td>281</td>
</tr>
<tr>
<td>Chapter Contents</td>
<td>281</td>
</tr>
<tr>
<td>Command Types</td>
<td>284</td>
</tr>
<tr>
<td>Common Command Format</td>
<td>284</td>
</tr>
<tr>
<td>SCPI Command Format</td>
<td>284</td>
</tr>
<tr>
<td>Command Separator</td>
<td>285</td>
</tr>
<tr>
<td>Abbreviated Commands</td>
<td>285</td>
</tr>
<tr>
<td>Implied (Optional) Commands</td>
<td>285</td>
</tr>
<tr>
<td>Variable Command Syntax</td>
<td>285</td>
</tr>
<tr>
<td>SCPI Command Parameters</td>
<td>286</td>
</tr>
<tr>
<td>Parameter Types, Explanations, and Examples</td>
<td>286</td>
</tr>
<tr>
<td>Optional Parameters</td>
<td>287</td>
</tr>
<tr>
<td>Querying Parameter Settings</td>
<td>287</td>
</tr>
<tr>
<td>SCPI Command Execution</td>
<td>288</td>
</tr>
<tr>
<td>Command Coupling</td>
<td>288</td>
</tr>
<tr>
<td>Linking Commands</td>
<td>288</td>
</tr>
<tr>
<td>SCPI Command Reference</td>
<td>289</td>
</tr>
<tr>
<td>ABORt</td>
<td>290</td>
</tr>
<tr>
<td>ARM</td>
<td>291</td>
</tr>
<tr>
<td>[:STARt][:LAYer[1]][:COUNt]</td>
<td>291</td>
</tr>
<tr>
<td>[:STARt][LAYer2][:COUNt]</td>
<td>292</td>
</tr>
<tr>
<td>[:STARt][LAYer2][:IMMediate]</td>
<td>293</td>
</tr>
<tr>
<td>[:STARt][LAYer2][:SLOPe]</td>
<td>293</td>
</tr>
<tr>
<td>[:STARt][LAYer2][:SOURrce]</td>
<td>294</td>
</tr>
<tr>
<td>:SWEep[:COUNt]</td>
<td>295</td>
</tr>
<tr>
<td>:SWEep[:IMMediate]</td>
<td>295</td>
</tr>
<tr>
<td>:SWEep[:LINK]</td>
<td>296</td>
</tr>
<tr>
<td>:SWEep[:SOURrce]</td>
<td>297</td>
</tr>
<tr>
<td>CALibration</td>
<td>298</td>
</tr>
<tr>
<td>:COUNt?</td>
<td>298</td>
</tr>
</tbody>
</table>
:DATA:AC2 ................................................................. 299
:DATA[:DC] ............................................................... 300
[:DC]:BEGIN ............................................................ 300
[:DC]:POINT? ............................................................ 301
:SECure:CODE ........................................................... 302
:SECure[:STATe] .......................................................... 303
:STATe ................................................................. 304
:STATe:AC ............................................................... 304
:STATe:DC ............................................................... 305
INITiate ................................................................. 306
[:IMMediate] ............................................................. 306
OUTPut[1] ............................................................... 308
:FILTer[:LPASs]:FREQuency ........................................ 308
:FILTer[:LPASs][:STATe] ........................................... 309
:IMPedance ............................................................. 309
:LOAD ................................................................. 310
:LOAD:AUTO ............................................................. 311
[:STATe] ................................................................. 311
[SOURce:] ............................................................. 312
[SOURce]:ARBitrary .................................................... 313
:DAC:FORMat ............................................................ 313
:DAC:SOURce ........................................................... 315
:DOWNLOAD ............................................................. 316
:DOWNLOAD:COMPLETE ............................................... 318
[SOURce]:FREQuency[1] ............................................. 319
:CENTer ................................................................. 321
[:CW]:FIXed .............................................................. 322
:FSKey ................................................................. 323
:FSKey:SOURce .......................................................... 324
:MODE ................................................................. 325
:RANGE ................................................................. 326
:SPAN ................................................................. 327
:STARt ................................................................. 328
:STOP ................................................................. 329
[SOURce]:FREQuency2 ................................................ 330
[:CW]:FIXed .............................................................. 331
[SOURce]:FUNCTION ................................................... 332
[:SHAPE] ............................................................... 332
:USER ................................................................. 333
[SOURce]:LIST[1] ...................................................... 334
:FORMat[:DATA] ......................................................... 335
[:SEGMent]:ADDRes? ................................................... 336
[:SEGMent]:CATalog? ................................................... 336
[:SEGMent]:COMBined .................................................. 337
[:SEGMent]:COMBined:POINts? ....................................... 338
[:SEGMent]:DEFINE ..................................................... 339
[:SEGMent]:DELeTe:ALL .................................................. 340
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWEep:COUNt</td>
<td>372</td>
</tr>
<tr>
<td>:DIREction</td>
<td>373</td>
</tr>
<tr>
<td>:POINTs</td>
<td>374</td>
</tr>
<tr>
<td>:SPACing</td>
<td>375</td>
</tr>
<tr>
<td>:TIME</td>
<td>376</td>
</tr>
<tr>
<td>[SOURCE:]VOLTage</td>
<td>377</td>
</tr>
<tr>
<td>[:LEVEL][:IMMediate][:AMPLitude]</td>
<td>377</td>
</tr>
<tr>
<td>[:LEVEL][:IMMediate][:AMPLitude]:UNIT[:VOLTage]</td>
<td>379</td>
</tr>
<tr>
<td>[:LEVEL][:IMMediate]:OFFSET</td>
<td>380</td>
</tr>
<tr>
<td>STATUS</td>
<td>381</td>
</tr>
<tr>
<td>:OPC:INITiate</td>
<td>382</td>
</tr>
<tr>
<td>:OPERation:CONDition?</td>
<td>383</td>
</tr>
<tr>
<td>:OPERation:ENABLE</td>
<td>383</td>
</tr>
<tr>
<td>:OPERation[:EVENt]?</td>
<td>384</td>
</tr>
<tr>
<td>:OPERation:NTRansition</td>
<td>384</td>
</tr>
<tr>
<td>:OPERation:PTRANSition</td>
<td>385</td>
</tr>
<tr>
<td>:PRESet</td>
<td>385</td>
</tr>
<tr>
<td>:QUESTionable:CONDition?</td>
<td>386</td>
</tr>
<tr>
<td>:QUESTionable:ENABLE</td>
<td>386</td>
</tr>
<tr>
<td>:QUESTionable[:EVENt]?</td>
<td>387</td>
</tr>
<tr>
<td>:QUESTionable:NTRANSition</td>
<td>387</td>
</tr>
<tr>
<td>:QUESTionable:PTRANSition</td>
<td>388</td>
</tr>
<tr>
<td>SYSTem</td>
<td>389</td>
</tr>
<tr>
<td>:ERRor?</td>
<td>389</td>
</tr>
<tr>
<td>:VERSiOn?</td>
<td>390</td>
</tr>
<tr>
<td>TRIGger</td>
<td>391</td>
</tr>
<tr>
<td>[:START]:COUNt</td>
<td>392</td>
</tr>
<tr>
<td>[:START]:GATE:POLarity</td>
<td>393</td>
</tr>
<tr>
<td>[:START]:GATE:SOURce</td>
<td>393</td>
</tr>
<tr>
<td>[:START]:GATE:STATE</td>
<td>394</td>
</tr>
<tr>
<td>[:START][:IMMediate]</td>
<td>395</td>
</tr>
<tr>
<td>[:START]:SLOPe</td>
<td>395</td>
</tr>
<tr>
<td>[:START]:SOURce</td>
<td>396</td>
</tr>
<tr>
<td>[:STOP][:IMMediate]</td>
<td>397</td>
</tr>
<tr>
<td>[:STOP]:SLOPe</td>
<td>398</td>
</tr>
<tr>
<td>[:STOP]:SOURce</td>
<td>398</td>
</tr>
<tr>
<td>SWEep[:IMMediate]</td>
<td>399</td>
</tr>
<tr>
<td>SWEep:LINK</td>
<td>400</td>
</tr>
<tr>
<td>SWEep:SOURce</td>
<td>401</td>
</tr>
<tr>
<td>SWEep:TIME</td>
<td>402</td>
</tr>
<tr>
<td>VINstrument</td>
<td>403</td>
</tr>
<tr>
<td>[:CONFiGure]:LBUS[:MODE]</td>
<td>403</td>
</tr>
<tr>
<td>[:CONFiGure]:LBUS[:MODE]:AUTO</td>
<td>404</td>
</tr>
<tr>
<td>[:CONFiGure]:TEST:CONFiGure</td>
<td>405</td>
</tr>
<tr>
<td>[:CONFiGure]:TEST:DATA?</td>
<td>406</td>
</tr>
<tr>
<td>[:CONFiGure]:VME[:MODE]</td>
<td>406</td>
</tr>
</tbody>
</table>
Appendix B. Useful Tables

Appendix A. HP E1445A Specifications

Appendix C. Register-Based Programming
Selecting the Waveform Sequence .................................................. 498
The Waveform Sequence Registers ............................................... 498
Sequence Selection Program ......................................................... 500
HP BASIC Program Example (WAVE_SEL) ..................................... 500
Loading the DAC from the VXIbus ............................................... 506
HP BASIC Program Example (VXISRCE) ...................................... 506
Index ......................................................................................... 509
Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Institute of Standards and Technology (formerly National Bureau of Standards), to the extent allowed by that organization’s calibration facility, and to the calibration facilities of other International Standards Organization members.

Warranty

This Hewlett-Packard product is warranted against defects in materials and workmanship for a period of three years from date of shipment. Duration and conditions of warranty for this product may be superseded when the product is integrated into (becomes a part of) other HP products. During the warranty period, Hewlett-Packard Company will, at its option, either repair or replace products which prove to be defective.

For warranty service or repair, this product must be returned to a service facility designated by Hewlett-Packard (HP). Buyer shall pre-pay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country.

HP warrants that its software and firmware designated by HP for use with a product will execute its programming instructions when properly installed on that product. HP does not warrant that the operation of the product, or software, or firmware will be uninterrupted or error free.

Limitation Of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied products or interfacing, unauthorized modification or misuse, operation outside of the environmental specifications for the product, or improper site preparation or maintenance.

The design and implementation of any circuit on this product is the sole responsibility of the Buyer. HP does not warrant the Buyer’s circuitry or malfunctions of HP products that result from the Buyer’s circuitry. In addition, HP does not warrant any damage that occurs as a result of the Buyer’s circuitry or any defects that result from Buyer-supplied products.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED. HP SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Exclusive Remedies

THE REMEDIES PROVIDED HEREIN ARE BUYER’S SOLE AND EXCLUSIVE REMEDIES. HP SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER BASED ON CONTRACT, TORT, OR ANY OTHER LEGAL THEORY.

Notice

The information contained in this document is subject to change without notice. HEWLETT-PACKARD (HP) MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. HP shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material. This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of Hewlett-Packard Company. HP assumes no responsibility for the use or reliability of its software on equipment that is not furnished by HP.

U.S. Government Restricted Rights

The Software and Documentations have been developed entirely at private expense. They are delivered and licensed as "commercial computer software" as defined in DFARS 252.227-7013 (October 1988), DFARS 252.211.7015 (May 1991) or DFARS 252.227-7014 (June 1995), as a "commercial item" as defined in FAR 2.101(a), or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) (or any equivalent agency regulation or contract clause), whichever is applicable. You have only those rights provided for such Software and Documentation by the applicable FAR or DFARS clause or the HP standard software agreement for the product involved.
WARNINGS

The following general safety precautions must be observed during all phases of operation, service, and repair of this product. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the product. Hewlett-Packard Company assumes no liability for the customer’s failure to comply with these requirements.

Ground the equipment: For Safety Class 1 equipment (equipment having a protective earth terminal), an uninterruptible safety earth ground must be provided from the mains power source to the product input wiring terminals or supplied power cable.

DO NOT operate the product in an explosive atmosphere or in the presence of flammable gases or fumes. For continued protection against fire, replace the line fuse(s) only with fuse(s) of the same voltage and current rating and type. DO NOT use repaired fuses or short-circuited fuse holders.

Keep away from live circuits: Operating personnel must not remove equipment covers or shields. Procedures involving the removal of covers or shields are for use by service-trained personnel only. Under certain conditions, dangerous voltages may exist even with the equipment switched off. To avoid dangerous electrical shock, DO NOT perform procedures involving cover or shield removal unless you are qualified to do so.

DO NOT operate damaged equipment: Whenever it is possible that the safety protection features built into this product have been impaired, either through physical damage, excessive moisture, or any other reason, REMOVE POWER and do not use the product until safe operation can be verified by service-trained personnel. If necessary, return the product to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

DO NOT service or adjust alone: Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

DO NOT substitute parts or modify equipment: Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification to the product. Return the product to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.
# Declaration of Conformity

according to ISO/IEC Guide 22 and EN 45014

| Manufacturer’s Name: | Hewlett-Packard Company  
| Loveland Manufacturing Center |
| Manufacturer’s Address: | 815 14th Street S.W.  
| Loveland, Colorado 80537 |

declares, that the product:

| Product Name: | Arbitrary Function Generator |
| Model Number: | HP E1445A |
| Product Options: | All |

conforms to the following Product Specifications:

**Safety:**

- CSA C22.2 #1010.1 (1992)
- UL 3111

**EMC:**

- CISPR 11:1990/EN55011 (1991): Group 1 Class A
- IEC 801-2:1991/EN50082-1 (1992): 4 kVCD, 8 kVAD
- IEC 801-3:1984/EN50082-1 (1992): 3 V/m
  0.5 kV Signal Lines

**Supplementary Information:** The product herewith complies with the requirements of the Low Voltage Directive 73/23/EEC and the EMC Directive 89/336/EEC (inclusive 93/68/EEC) and carries the “CE” marking accordingly.

Tested in a typical HP C-Size VXI Mainframe.

July 29, 1996  
Jim White, QA Manager

European contact: Your local Hewlett-Packard Sales and Service Office or Hewlett-Packard GmbH,  
Department HQ-TRE, Herrenberger Straße 130, D-71034 Böblingen, Germany (FAX +49-7031-14-3143).
You can help us improve our manuals by sharing your comments and suggestions. In appreciation of your time, we will enter you in a quarterly drawing for a Hewlett-Packard Palmtop Personal Computer (U.S. government employees cannot participate in the drawing).

Please list the system controller, operating system, programming language, and plug-in modules you are using.

Please pencil-in one circle for each statement below:

- The documentation is well organized.
- Instructions are easy to understand.
- The documentation is clearly written.
- Examples are clear and useful.
- Illustrations are clear and helpful.
- The documentation meets my overall expectations.

Please write any comments or suggestions below--be specific.
Chapter 1
Getting Started

Chapter Contents
This chapter shows you how to configure, install, and begin using the HP E1445A Arbitrary Function Generator (AFG). The main sections of this chapter include:

• Preparation for Use ....................... Page 19
  – VXIbus Factory Settings .................. Page 20
  – The AFG Logical Address ................. Page 21
  – Addressing the AFG ...................... Page 22
  – Setting the AFG Servant Area .......... Page 23
  – The AFG Bus Request Level .............. Page 24
  – AFG Installation in a Mainframe ........ Page 25

• Instrument Language (SCPI) ............. Page 26
  – SCPI Programming ...................... Page 26
  – Command Coupling ..................... Page 27

• Program Languages ....................... Page 29
  – HP BASIC Language Programs .......... Page 29
  – Visual BASIC Language Programs ...... Page 32
  – Visual C/C++ Language Programs ...... Page 40

• Introductory Programs .................... Page 46
  – AFG Self-Test ......................... Page 46
  – Resetting and Clearing the AFG ....... Page 47
  – Querying the Power-On/Reset Configuration Page 48
  – Checking for Errors .................... Page 49
  – Generating Sine Waves .................. Page 51

Preparation for Use

This section contains the HP E1445A AFG VXIbus information required to configure the device and install it in the HP 75000 Series C mainframe.

Note
The following VXIbus configuration information pertains to the HP E1445A Arbitrary Function Generator. For more (VXIbus) system configuration information, refer to the C-Size VXIbus Systems Configuration Guide.
VXIbus Factory Settings

The HP E1445A AFG (shown in Figure 1-1) is configured at the factory as shown in Table 1-1.

![Device Information]

Device Information

- **Device Type**: message-based
- **C-Size**: (1 Slot)
- **Connectors**: P1 and P2
- **Addressing Mode**: A18/A24
- **A24 Size**: 4096 bytes
- **Dynamically Configurable**
- **Non-Interrupted/Non-Interrupt Handler**
- **VXIbus Revision Compliance**: 1.3
- **SCPI Revision**: 1.0
- See side of AFG for power/cooling requirements.

![Figure 1-1. The HP E1445A Arbitrary Function Generator]

Table 1-1. HP E1445A VXIbus System Factory Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical Address</td>
<td>80</td>
</tr>
<tr>
<td>Servant Area</td>
<td>0</td>
</tr>
<tr>
<td>Bus Request Level</td>
<td>3</td>
</tr>
</tbody>
</table>
Appendix A has the complete list of HP E1445A VXIbus system specifications.

### The AFG Logical Address

The HP E1445A AFG logical address is used:

- to place the AFG in the servant area of a commander (HP E1406A Command Module or an embedded controller, for example);
- with the AFG servant area switch to set the AFG servant area;
- and to address the AFG (see “Addressing the AFG” on page 22 and “Using an Embedded Controller” on page 23.)

### Assigning the AFG to a Commander

In a VXIbus system, every device must be in the servant area of a commander (with the exception of the top-level commander).

Note the following when assigning the HP E1445A AFG to a commander:

- A commander’s servant area is defined as:
  Servant area = (logical address + 1) through (logical address + servant area switch setting)

- The HP E1445A AFG is a message-based device. If an embedded controller and an HP E1406A Command Module are part of your VXIbus system, put the AFG in the servant area of the controller. This enables you to program the AFG at higher speeds across the VXIbus backplane, rather than over the Hewlett-Packard Interface Bus (HP-IB*) via the command module.

- If your system uses an external controller and the HP E1406A Command Module, put the AFG in the servant area of the command module. This enables the module to function as the HP-IB interface to the arbitrary function generator.

The HP E1406A Command Module has a factory set logical address of 0 and a servant area switch setting of 255. Using the factory settings, it is not necessary to change the logical address of the AFG (80) to place it in the servant area of the command module.

The HP E1445A AFG logical address switch is shown in Figure 1-2.

---

* HP-IB is Hewlett-Packard’s implementation of IEEE Std. 488.1-1978
Addressing the AFG
(External Controller and PC)

Devices in the C-size mainframe and in the servant area of the HP E1406A Command Module are located by an HP-IB address. The HP-IB address is a combination of the controller’s interface select code, the command module’s primary HP-IB address, and the device’s secondary HP-IB address. An address in this form in a HP BASIC statement appears as:

```
OUTPUT 70910:"SOUR:ROSC:SOUR INT1;:TRIG:STAR:SOUR INT1"
```

**Interface Select Code (7):** This code is determined by the address of the HP-IB interface card in the controller. In most Hewlett-Packard controllers, this card has a factory set address of 7, including the HP 82340/82341 HP-IB Interface Card (this card was used with an HP Vectra PC to create the Visual BASIC and Visual C/C++ example programs).

**Primary HP-IB Address (09):** This is the address of the HP-IB port on the command module. Valid addresses are 0 to 30. The module has a factory set address of 9.

**Secondary HP-IB Address (10):** This address is derived from the logical address of the device (AFG) by dividing the logical address by 8. Thus, for the HP E1445A AFG factory set logical address of 80, the secondary address is 10.
Using an Embedded Controller

As a message-based device, the HP E1445A can easily be programmed across the VXIbus backplane from an embedded controller. The select code of the VXI interface board in embedded controllers is typically 16. Since no secondary HP-IB address is required when programming over the backplane, the logical address of the HP E1445A is combined with the VXI interface select code.

For example, to send commands to the AFG with logical address 80, the OUTPUT statement in an HP BASIC program appears as:

```plaintext
OUTPUT 1680;"SOUR:ROSC:SOUR INT1;:TRIG:STAR:SOUR INT1"
```
(for device logical addresses from 01 to 99)

```plaintext
or
```

```plaintext
OUTPUT 160xxx;"SOUR:ROSC:SOUR INT1;:TRIG:STAR:SOUR INT1"
```
(for device logical addresses from 100 to 255)

Setting the AFG Servant Area

The HP E1445A servant area is set when the HP E1446A Summing Amplifier/DAC is used with the Arbitrary Function Generator. Note the following when setting the AFG servant area:

- The HP E1445A servant area need only be set when the HP E1446A Summing Amplifier/DAC is used with the AFG (factory setting = 0).
- The HP E1446A must be in the AFG servant area in order for the AFG to control the Summing Amplifier/DAC.
- The HP E1445A servant area is defined as:
  Servant area = (logical address + 1) through 
  (logical address + servant area switch setting).
- The HP E1446A Summing Amplifier/DAC should be the only device in the AFG servant area. Other devices in the servant area would be inaccessible to other commanders (HP E1406A Command Module, for example).

The HP E1445A AFG servant area switch is shown in Figure 1-2.
The AFG Bus Request Level

The bus request level is a priority at which the HP E1445A can request the use of the Data Transfer Bus.

Bus Request Level Guidelines

- There are four bus request lines (BG0–BG3) from which one is selected (Figure 1-3). Bus request line 3 has the highest priority, bus request line 0 has the lowest priority.

- It is not necessary to change the bus request level setting (BG3) on the AFG. (More information on the Data Transfer Bus can be found in the C-Size VXIbus Systems Configuration Guide.)

![Figure 1-3. Setting the AFG Bus Request Level](image-url)
AFG Installation in a Mainframe

The HP E1445A may be installed in any slot (except slot 0) in a C-size VXIbus mainframe. If an HP E1446A Summing Amplifier/DAC is part of your system, the amplifier should be installed in a slot next to the HP E1445A.

To install in a mainframe:

1. Set the extraction levers out. Slide the module into any slot (except slot 0) until the backplane connectors touch.
2. Seat the module by moving the levers toward each other.
3. Tighten the top and bottom screws to secure the module in the mainframe.

Note

For compliance with European EMI standards, order the Backplane Connector Shield Kit HP Part Number E1400-80920.

Removing a Module

To remove a module from a mainframe:

1. Loosen the top and bottom screws securing the module in the mainframe.
2. Move the extraction levers away from each other. As the levers are moved, the module will detach from the backplane connectors.
3. Slide the module out.
The extraction levers will not seat and unseat the backplane connectors on older HP VXIbus mainframes and non-HP mainframes. You must manually seat the connectors by pushing the module into the mainframe until the front panel is flush with the front of the mainframe. The extraction levers may be used to guide or remove the module.

**Instrument Language (SCPI)**

The HP E1445A AFG uses the Standard Commands for Programmable Instruments (called SCPI) as the instrument control language. The programs shown in this manual are written in HP BASIC which uses the SCPI commands for instrument controls. These programs, and also programs in other languages, are contained on the CD that came with the instrument (see “Program Languages” on page 29 for more information).

**SCPI Programming**

SCPI (Standard Commands for Programmable Instruments) is an ASCII-based instrument command language designed for test and measurement instruments. The message-based AFG has an on-board microprocessor which interprets the ASCII command strings and returns ASCII formatted results.

**SCPI Command Structure**

The HP E1445A SCPI command set is found in Chapter 8. SCPI commands are based on a hierarchical structure, also known as a tree system. In this system, associated commands are grouped together under a common node or root, thus, forming subtrees or subsystems. An example is the HP E1445's ARM subsystem shown below.

```
ARM
   [:START|:SEQuence[1]]
   [:LAYer[1]]
       :COUNt <number>
       :LAYer2
           :COUNt <number>
           [:IMMediate] [no query]
           :SLOPe <edge>
           :SOURce <source>

   :SWEep|:SEQuence3
       :COUNt <number>
       [:IMMediate] [no query]
       :LINK <link>
       :SOURce <source>
```

ARM is the root keyword of the command, [:START|:SEQuence[1]] and [:SWEep|:SEQuence3 are second-level keywords, [:LAYer[1]] and [:LAYer2
are third-level keywords, and so on. A colon (:) always separates a command keyword from a lower-level keyword as shown below.

```
ARM:LAY2:SOUR EXT
```

A semicolon (;) is used to separate two commands within the same subsystem, and can also save typing. For example, sending this command message:

```
ARM:LAY2:SOUR EXT;SLOP POS;COUN 10
```

Is the same as sending these three commands:

```
ARM:LAY2:SOUR EXT
ARM:LAY2:SLOP POS
ARM:LAY2:COUN 10
```

**Manual Format**

The typical format of commands listed in the command reference and throughout this manual is:

```
[SOURce:]FREQuency[1]:MODE <mode>
```

Command headers enclosed in square brackets ([]) are optional. Upper-case letters in the header are required, lower-case letters can be omitted.

---

**Note**

The brackets are not part of the command and are not sent to the instrument.

---

**Command Coupling**

Many of the AFG SCPI commands are value coupled. This means that the value set by one command may affect the valid limits for the values of other commands. This can result in "Settings Conflict" errors when the program executes. To prevent these errors, the AFG commands must be executed in "Coupling Groups". The coupling groups are frequency and voltage. Some commands (like [SOURce:]FUNCTION[:SHAPe]) are associated with both groups. These commands are a bridge, linking (coupling) the two groups. Commands not in a coupling group must precede or follow commands in the coupling groups. Executing uncoupled commands in a coupling group breaks the coupling and can cause a "Settings Conflict" error.

The coupling groups and associated commands can be found in Table B-2 in Appendix B.
Executing Coupled Commands

Command coupling determines the AFG programming sequence. The high-level sequence, based on the coupling groups, is shown in Figure 1-5.

Coupled commands must be contiguous and executed in the same program statement. This is done by placing the commands in the same program line, or by suppressing the end-of-line terminator until the last (coupled) command has been sent.

To send multiple commands in a single line or in a single statement, the commands are linked with a semicolon (;) and a colon (:). This is illustrated in the following lines:

```
SOUR:ROSC:SOUR INT2;:TRIG:STAR:SOUR INT2
```

```
SOUR:ROSC:SOUR INT2;
:TRIG:STAR:SOUR INT2
```

Both techniques are used in the programs found throughout this manual.

Note that the semicolon (;) and colon (;) link commands within different subsystems. Only a semicolon (;) is required to link commands within the same subsystem (see “SCPI Command Structure” on page 26).

---

**Note**

See page 31 for information on suppressing the end-of-line terminator.
Program Languages

The program language shown in this manual is HP BASIC. This language was selected since it easily shows how to program the AFG.

However, the same programs (except where noted) are also supplied in Visual C/C++ and Visual BASIC using the HP Standard Instrument Control Library (SICL). The programs using SICL are Windows® programs. All programs are supplied on the CD that came with this manual (see next section).

Example Program CD

To determine the location of the different programs and the required libraries, read the “README” files. The different directories are:

- VBPROG for Visual BASIC programs
- VCPROG for Visual C/C++ programs

HP BASIC Language Programs

The following information identifies the system on which the HP BASIC programs were written and shows how the programs are structured.

System Configuration

Except where noted, the example programs in HP BASIC were developed on the following system:

- **Controller:** HP 9000 Series 300
- **Mainframe:** HP 75000 Series C
- **Slot 0/Resource Manager:** HP E1406A Command Module
- **HP E1445A Logical Address:** 80
- **Instrument Language:** SCPI
Typical HP BASIC Example Program

The structure of an example program in HP BASIC is shown below. This program enables output leveling by sweeping.

```
1 !RE-STORE"SWP.LEVL"
2 !This program enables output leveling over the 0 Hz to 10 MHz sweep.
3 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !
50 !Set up error checking
60 ON INTR 7 CALL Errmsg
70 ENABLE INTR 7:2
80 OUTPUT @Afg;"CLS"
90 OUTPUT @Afg;"SRE 32"
100 OUTPUT @Afg;"ESE 60"
110 !
120 !Call the subprograms
130 CALL Rst
140 CALL Swp_levl
150 !
160 WAIT .1 !allow interrupt to be serviced
170 OFF INTR 7
180 END
190 !
200 SUB Swp_levl
210 Swp_levl: !Subprogram which sets output leveling for sweeping from
220 !0 TO 10 MHz
230 COM @Afg
240 OUTPUT @Afg;"SOUR:FREQ1:MODE SWE;"
250 OUTPUT @Afg;"SOUR:FREQ1:STAR 0;"
260 OUTPUT @Afg;"SOUR:FREQ1:STOP 10E6;"
270 OUTPUT @Afg;"SOUR:SWE:COUN INF;"
280 OUTPUT @Afg;"SOUR:FUNC:SHAP SIN;"
290 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5 V"
300 OUTPUT @Afg;"OUTP1:FILT:LPAS:FREQ 10 MHZ"
310 OUTPUT @Afg;"OUTP1:FILT:LPAS:STAT ON"
320 OUTPUT @Afg;"INIT:IMM"
330 SUBEND
340 !
350 SUB Rst
360 Rst: !Subprogram which resets the E1445.
370 COM @Afg
380 OUTPUT @Afg;"RST;"OPC?"
390 ENTER @Afg;Complete
400 SUBEND
410 !
420 SUB Errmsg
430 Errmsg: !Subprogram which displays E1445 programming errors
440 COM @Afg
```

Continued on Next Page
Turning Off (Suppressing) the End-Of-Line Terminator

As mentioned earlier, coupled commands must be contiguous and executed in the same program statement. By suppressing the end-of-line (EOL) terminator (Line Feed) on a command line, coupled commands can be sent on separate lines, yet as a single program statement.

In HP BASIC programs, the EOL terminator is suppressed by placing a semicolon (;) following the quotation mark (") which closes the command string. In the previous program, the commands in lines 240–270 are in the frequency coupling group, line 280 is in the frequency/voltage coupling group, and line 290 is in the voltage coupling group. The semicolons following the command strings in lines 240 through 280 suppress the EOL terminator; therefore lines 240–290 are sent as a single statement. Since the command in line 290 is not coupled to the commands in lines 300–320, the EOL terminator is not suppressed on line 290.
These example programs are written in the Visual BASIC language for the HP 82340/82341 HP-IB Interface Cards using the HP Standard Instrument Control Library (SICL).

The following identifies the system on which the programs are written, shows how to compile the programs, and gives a typical example program.

**System Configuration**

The Visual BASIC programs were developed on the following system:

- **Controller:** HP Vectra PC
- **HP-IB Interface Card:** HP 82341 HP-IB Interface with HP SICL
- **Required Program:** See “What’s Needed to Run the Programs” below
- **Mainframe:** HP 75000 Series C
- **Slot 0/Resource Manager:** HP E1406A Command Module
- **HP E1445A Logical Address:** 80
- **Instrument Language:** SCPI

**What’s Needed to Run the Programs**

You need to include the “SICL.BAS” program file in your Visual BASIC project to run the example programs. To add the file, be sure you have opened a project you will be using. Then, Press Ctrl-D and enter the path and file name or select the “Add File” menu item under the “File” menu and select the SICL.BAS file (see the Visual BASIC documentation for more information).

**How to Run a Program**

You can run the example program in the Visual BASIC environment or compile it to make an executable file. Use the appropriate menu in the environment to compile the program (see the Visual BASIC documentation for more information). Note that the program can only operate under Windows®.
Typical Visual BASIC Example Program Using HP SICL

The following is an example program written in Visual BASIC using the HP Standard Instrument Control Library. The program:

- sends commands to the AFG to generate an arbitrary waveform;
- receives data from the AFG;
- shows how to send coupled commands;
- and performs error checking of the AFG.

Only program codes are given here. Refer to the actual program on the CD to see the data that generates the form, buttons, etc.

' ARBWAVE.FRM - This program generates a 100 points ramp. The data to generate
' the ramp is transferred to the AFG as comma separated voltages

' Instrument HP-IB address
Const ShowAddr = "hpib7,9,10"
Dim Addr As Integer

Dim ChkName As String

Sub CheckError (SubName As String)
' Check for any errors

Dim Actual As Long
Dim RdErr As String * 256
Dim Work As String
Dim ErrNum As Integer
Dim TempName As String

TempName = ChkName
ChkName = "CheckError"

' Read error message
Call iwrite(Addr, ByVal "SYSTem:ERRor?" + Chr$(10), 14, 1, Actual)
Call iread(Addr, ByVal RdErr, 256, 0, Actual)

' If error was detected
ErrNum = Val(RdErr)

If ErrNum <> 0 Then

' Store message only into Work string
Work = Mid$(RdErr, 1, Actual - 1)
Work = Work + " - in Sub " + SubName

' Enable and clear error list box
ShowErr.Enabled = True
ShowErr.Visible = True
ShowErr.Clear

Continued on Next Page
Action.Text = "The program generated the following error(s):"

' Show error message
ShowErr.AddItem Work

' Loop until error message is 0
Do

' Read error message
Call iwrite(Addr, ByVal "SYSTem:ERRor?" + Chr$(10), 14, 1, Actual)
Call iread(Addr, ByVal RdErr, 256, 0, Actual)

' Store message only into Work string
Work = Mid$(RdErr, 1, Actual - 1)

' Get error number
ErrNum = Val(Work)

' If error, show error message
If ErrNum <> 0 Then
    Work = Work + " - in Sub " + SubName
    ShowErr.AddItem Work
End If
Loop Until (ErrNum = 0)

' Close communication with instrument
Call iclose(Addr)

' Clean up sicl
Call siclcleanup

' Press to exit
DispErr = "The program detected errors in sub/function: " + SubName + Chr$(10)
DispErr = DispErr + Chr$(10) + ShowErr + "Press " + Chr$(34) + "OK" + Chr$(34) + " to exit!"
MsgBox DispErr, 64, "sw_vbs: CheckError"

End

End If

ChkName = TempName

End Sub

Sub CmdExe (Cmd() As String)
' This sub sends SCPI commands
Dim Cnt As Integer
Dim Actual As Long

Continued on Next Page
Cnt = 1

While Len(Cmd(Cnt))

    ' Send SCPI command
    Call iwrite(Addr, ByVal Cmd(Cnt) + Chr$(10), Len(Cmd(Cnt)) + 1, 1, Actual)

    Cnt = Cnt + 1
Wend

End Sub

Sub ExitProg_Click ()

    ' End program
    End
End Sub

End Sub

Sub Form_Load ()

    ' Disable showing exit program button and lists
    ExitProg.Visible = False
    ShowQuery.Visible = False

    ' Show Action
    Action.Enabled = False
    Action.Visible = True

    ' Enable form
    Arbwave.Visible = True

    ' Call program to execute instrument
    Call Main

    ' Enable showing exit program button and make it the focus
    ExitProg.Visible = True
    ExitProg.SetFocus

End Sub

Sub GenSeg ()

    ' Setup AFG to generate an arbitrary waveform

    Static SetCommands(1 To 10) As String
    Static OutCommands(1 To 10) As String
    Dim SegCommand As String

    Continued on Next Page
' Use the "SetCommands" array to setup the AFG
SetCommands(1) = "SOUR:LIST1:SSEQ:DEL:ALL"  ' Clear sequence memory
SetCommands(2) = "SOUR:LIST1:SEGM:DEL:ALL"  ' Clear segment memory

SetCommands(3) = "SOUR:ROSC:SOUR INT1;"  ' Select the Ref. Oscillator
SetCommands(3) = SetCommands(3) + ":TRIG:STAR:SOUR INT1;"  ' Select the sample source"
SetCommands(3) = SetCommands(3) + ":SOUR:FREQ1:FIX 100E3;"  ' Set the sample frequency
SetCommands(3) = SetCommands(3) + ":SOUR:FUNC:SHAP USER;"  ' Command to select the user function
SetCommands(3) = SetCommands(3) + ":SOUR:VOLT:LEV:IMM:AMPL 5.1V"  ' Set the amplitude

SetCommands(4) = "SOUR:LIST1:SEGM:SEL ramp"  ' Define the "ramp" segment name
SetCommands(5) = "SOUR:LIST1:SEGM:DEF 100"  ' Define the segment size

' Use the "OutCommands" array to generate output
OutCommands(1) = "SOUR:LIST1:SSEQ:SEL ramp_out"  ' Define the sequence name as "ramp_out"
OutCommands(2) = "SOUR:LIST1:SSEQ:DEF 1"  ' Define the sequence size
OutCommands(3) = "SOUR:LIST1:SSEQ:SEQ ramp"  ' Set the segment execution order
OutCommands(4) = "SOUR:FUNC:USER ramp_out"  ' Define the user name
OutCommands(5) = "INIT:IMM"  ' Start waveform generation

' Use "SegCommand" to store segments
SegCommand = "SOUR:LIST1:SEGM:VOLT "  ' Command to send volts data

' Setup the AFG
Call CmdExe(SetCommands())

' Call sub to check for AFG errors
Call CheckError("GenSeg (SetCommands)")

' Generating and storing segments into string
For I = 0 To 99
If I = 99 Then
   SegCommand = SegCommand + Str$(I * .0505)
Else
   SegCommand = SegCommand + Str$(I * .0505) + ","
End If
Next I

' Send command with segment data
Call iwrite(Addr, ByVal SegCommand + Chr$(10), Len(SegCommand) + 1, 1, Actual)

' Call sub to check for AFG errors
Call CheckError("GenSeg (SegCommand)")

' Setup the AFG for output
Call CmdExe(OutCommands())

Continued on Next Page
' Call sub to check for AFG errors
Call CheckError("GenSeg (OutCommands)")

End Sub

Sub Main ()
    ' Main sub

    Dim Actual As Long

    ' Set error routine
    On Error GoTo AfgErr

    ChkName = "Main"

    ' Open communication path
    Addr = iopen(ShowAddr)

    ' Set timeout for 10 Sec
    Call itimeout(Addr, 10000)

    ' Reset and clear the module
    Action.Text = "Resetting the AFG; please wait"
    ChkName = "RstClr"
    Call RstClr

    ' Generate segment list and output sequence
    Action.Text = "Generate Segments"
    ChkName = "GenSeg"
    Call GenSeg

    ' Query segment and segment/sequence memory
    Action.Text = "Getting Memory Data"
    ChkName = "RunQuery"
    Call RunQuery

    Action.Text = "DONE!"

    ' Close communication with instrument
    Call iclose(Addr)

    ' Clean up sic1
    Call siclcleanup

    Exit Sub

    ' In case of timeout
    AfgErr:

    Continued on Next Page
Call TimeOut

End Sub

Sub RstClr ()

Dim RdMsg As String * 10
Dim Actual As Long
Dim Length As Integer

Length = 10
' Executes the commands that resets the module and clears its error register
Call iwrite(Addr, ByVal "*RST;*OPC?", 11, 1, Actual)
Call iread(Addr, ByVal RdMsg, Length, 0, Actual)

Length = 10
Call iwrite(Addr, ByVal "*CLS;*OPC?", 11, 1, Actual)
Call iread(Addr, ByVal RdMsg, Length, 0, Actual)

End Sub

Sub RunQuery ()

Dim GetMem As String
Dim RdMsg As String * 100
Dim Actual As Long

ShowQuery.Visible = True
ShowQuery.Enabled = True

' Query segment memory
GetMem = "SOUR:LIST1:SEGM:FREE?"
Call iwrite(Addr, ByVal GetMem + Chr$(10), Len(GetMem) + 1, 1, Actual)
Call iread(Addr, ByVal RdMsg, 100, 0, Actual)
ShowQuery.AddItem "Segment Memory Available/Used: " + Mid$(RdMsg, 1, Actual - 1)

' Query sequence memory
GetMem = "SOUR:LIST1:SSEQ:FREE?"
Call iwrite(Addr, ByVal GetMem + Chr$(10), Len(GetMem) + 1, 1, Actual)
Call iread(Addr, ByVal RdMsg, 100, 0, Actual)
ShowQuery.AddItem "Sequence Memory Available/Used: " + Mid$(RdMsg, 1, Actual - 1)

End Sub

Sub TimeOut ()
' Shows timeout message and exits program
Continued on Next Page
Dim ShowTimeMsg As String
Dim ErrMsg As String

' Set error routine
On Error Resume Next

' Get error message
ErrMsg = igeterrstr(igeterrno())

ShowTimeMsg = "The program generated error message " + Chr$(34) + ErrMsg + Chr$(34) + Chr$(10)
ShowTimeMsg = ShowTimeMsg + "in Sub/Function: " + ChkName + Chr$(10) + Chr$(10)
ShowTimeMsg = ShowTimeMsg + "Press " + Chr$(34) + "OK" + Chr$(34) + " to exit"
MsgBox ShowTimeMsg, 64, "Verif: TimeOut"

' Close communication with instrument
Call iclose(Addr)

' Clean up sicl
Call siclcleanup

' End program
End

End Sub
These example programs are written in the Visual C/C++ language for the HP 82340/82341 HP-IB Interface Cards using the HP Standard Instrument Control Library (SICL).

The following identifies the system on which the programs are written, shows how to compile the programs, and gives a typical example program.

**System Configuration**

The Visual C/C++ programs were developed on the following system:

- **Controller:** HP Vectra PC
- **HP-IB Interface Card:** HP 82341 HP-IB Interface with HP SICL
- **Required Libraries:** See “What’s Needed to Compile the Programs” below
- **Mainframe:** HP 75000 Series C
- **Slot 0/Resource Manager:** HP E1406A Command Module
- **HP E1445A Logical Address:** 80
- **Instrument Language:** SCPI

**What’s Needed to Compile the Programs**

You need the following libraries and header files. These are supplied with HP SICL.

- msapp16.lib - for Microsoft® Visual C and C++
- bcapp16.lib - Borland C and C++
- sicl16.lib
- sicl.h

---

**Note**

The programs must be compiled in the Large Memory Model

---

**How to Run a Program**

To run a program, first compile and link the program to make an executable file using the Large memory model. You can compile from the command line or the Windows™ interface. The two methods are:

**From the Command Line**

Make sure the program to be compiled and the appropriate libraries are in a project file. Do this in the C/C++ environment. Then do the following:

- For Borland compilers, type:

  ```
  MAKE <project_name>.MAK and press Enter
  ```

  ```
• For Microsoft® compilers used in Windows, type:

NMAKE <project_name>.MAK and press Enter

From the Windows Interface

Select the C/C++ Windows environment and make sure the program to be compiled and the appropriate libraries are in a project file. Then do the following:

• For Borland compilers, select:
  
  Project | Open Project to open the project, then
  Compile | Build All to compile the program

• For Microsoft compilers used in Windows, type:

  Project | Open to open the project, then
  Project | Re-build All to compile the program

Typical Visual C/C++
Example Program
Using HP SICL

Following is an example program written in Visual C/C++ using the HP Standard Instrument Control Library. The program:

– sends commands to the AFG to generate an arbitrary waveform;
– receives data from the AFG;
– shows how to send coupled commands;
– and performs error checking of the AFG.

// ARBWAVE.C - This program generates a 100 points ramp. The data to generate
// the ramp is transferred to the AFG as voltages

// Include the following header files
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>             // Use "alloc.h" for Turbo C(c) or C++(c)
#include <sicl.h>               // Included with SICL

#define DEV_ADDR "hpib7,9,10"   // Assign the HP-IB address

// Functions
void gen_seg(void);
void cmd_exe(char *commands[], int length, char *func_tion);
void rst_clr(void);
void send_data(char *commands, float *Wave_seg, int num_size);
void run_query(void);
void check_error(char *func_tion);
void time_out(char *func_name);

Continued on Next Page
void main(void)                 // Run the program
{
    
#if defined(__BORLANDC__) && !defined(__WIN_32)
_InitEasyWin();         // Required for Borland EasyWin program
#endif

    // Enable communication path to the module
    addr=iopen(DEV_ADDR);
    if (!addr)
    { printf("Unable to communicate with the module\n"); printf("Close the window or press Alt-F4 to exit\n"); exit(1); }

    // Set HP-IB Timeout to 10 seconds
    if (itimeout(addr, 10000))
        time_out("main - send timeout command");

    rst_clr();              // Reset the AFG
    gen_seg();              // Generate segment list and output sequence
    run_query();            // Query segment and segment/sequence memory

    // Close communication
    iclose(addr);
    // Release SICL resource allocation; not needed for Windows NT
    _siclcleanup();
    printf("Close the window or press Alt-F4 to exit");
    exit(0);
}

void gen_seg(void)
{
    char *set_commands[] =       // Use "set_commands" to setup the AFG
    { "SOUR:LIST1:SSEQ:DEL:ALL", // Clear sequence memory
        "SOUR:LIST1:SEGM:DEL:ALL", // Clear segment memory
        "SOUR:ROSC:SOUR INT1;"      // Select the Ref. Oscillator
        ":TRIG:STAR:SOUR INT1;"     // Select the sample source
        ":SOUR:FREQ1:FIX 100E3;"    // Set the sample frequency
        ":SOUR:FUNC:SHAP USER;"    // Command to select the user function

    };

Continued on Next Page
"::SOUR::VOLT::LEV::IMM:AMPL 5.1V",    // Set the amplitude
"SOUR::LIST1:SEGM:SEL ramp",        // Define the "ramp" segment name
"SOUR::LIST1:SEGM:DEF 100"           // Define the segment size
},

*seg_commands =                    // Use "seg_commands" to store segments
"SOUR::LIST1:SEGM::VOLT",         // Command to send volts data

*out_commands[] =                   // Use the "out_commands" array to generate output
{                                     
  "SOUR::LIST1:SSEQ:SEL ramp_out",   // Define the sequence name as "ramp_out"
  "SOUR::LIST1:SSEQ:DEF 1",          // Define the sequence size
  "SOUR::LIST1:SSEQ:SEQ ramp",       // Set the segment execution order
  "SOUR::FUNC::USER ramp_out",       // Define the user name
  "INIT::IMM"                        // Start waveform generation
};

float *Wave_seg;
int    loop,
seg_size = 100;                     // Set the segment size to 100 points
char   send_str[50];

// Allocate sufficient memory for storing the segments into computer memory
Wave_seg = (float*) malloc (seg_size * sizeof (float));

// Setup the AFG
cmd_exe(set_commands, sizeof(set_commands) / sizeof(char*), "gen_seg (set_commands)");

// Call routine to check for AFG errors
check_error("gen_seg (set_commands)");

// Calculate the segments
for (loop = 0; loop < seg_size; loop++)
  Wave_seg[loop] = (loop * .0505);

// Setup for iprintf function
sprintf(send_str, "%s %%%,f\n", seg_commands, seg_size);

// Call function to execute the final command with segment data
if(!iprintf(addr, send_str, Wave_seg))
  time_out("gen_seg (seg_commands)");

// Call routine to check for AFG errors
check_error("gen_seg (seg_commands)");

// Setup the AFG for output
cmd_exe(out_commands, sizeof(out_commands) / sizeof(char*), "gen_seg (out_commands)");
// Call routine to check for AFG errors
check_error("gen_seg (out_commands)");

// Free the allocated memory
free (Wave_seg);

void cmd_exe(char *commands[], int length, char *func_tion)
{
    int loop;

    for (loop = 0; loop < length; loop++)
        if(!iprintf(addr, "%s\n", commands[loop]))
            time_out(func_tion);

}

void run_query(void)
{
    char mem_size[21];

    // Query segment memory
    if(!ipromptf(addr, "SOUR:LIST1:SEGM:FREE?\n", "%t", mem_size))
        time_out("run_query - seg memory");

    printf("\nSegment Memory Available/Used: %s", mem_size);

    // Query sequence memory
    if(!ipromptf(addr, "SOUR:LIST1:SSEQ:FREE?\n", "%t", mem_size))
        time_out("run_query - seq memory");

    printf("\nSequence Memory Available/Used: %s", mem_size);

}

void rst_clr(void)
{
    int into;

    // Executes the commands that resets the AFG and clears its error register
    if(!ipromptf(addr, "*RST;*OPC?\n", "%i", &into))
        time_out("rst_clr - send *RST command");

    if(!ipromptf(addr, "*CLS;*OPC?\n", "%i", &into))
        time_out("rst_clr - send *CLS command");

}

void check_error(char *func_tion)
{
    char into[257];

    // Call routine to check for AFG errors
    check_error("gen_seg (out_commands)");

    // Free the allocated memory
    free (Wave_seg);

}
ipromptf(addr, "SYSTem:ERRor?\n", "%t", into); // Query error register

if (atoi(into)) // Determine if error is present
{
    // If errors present, print and exit
    printf("\n\nThe program detected the following error(s):\n\n");
    while (atoi(into))
    {
        printf("%s \t- in function %s\n", into, func_tion);
        ipromptf(addr, "SYSTem:ERRor?\n", "%t", into); // Query error register
    }
    // Close communication
    iclose(addr);

    // Release SICL resource allocation; not needed for Windows NT
    _siclcleanup();

    printf("\n\nClose the window or press Alt-F4 to exit");
    exit(1);
}

//********************************************************************************
void time_out(char *func_name)
{
    printf("\n\nThe program timed out in function: %s", func_name);
    // Close communication
    iclose(addr);

    // Release SICL resource allocation; not needed for Windows NT
    _siclcleanup();

    printf("\n\nClose the window or press Alt-F4 to exit");
    exit(1);
}
**Introductory Programs**

The introductory programs in this section include:

- AFG Self-Test
- Resetting the AFG and clearing its status registers
- Querying the AFG power-on/reset settings
- Checking for Errors
- Generating a sine wave with a single command

**AFG Self-Test**

The AFG self-test is executed with the command:

```
*TST?
```

The AFG parameters tested include:

- internal interrupt lines
- waveform select RAM
- segment sequence RAM
- waveform segment RAM
- DDS/NCO operation
- sine wave generation
- arbitrary waveform generation
- marker generation
- waveform cycle and arm counters
- sweep timer
- frequency-shift keying
- stop trigger
- DC analog parameters (amplitude, offset, attenuators, filters, calibration DACs)

Upon completion of the test, one of the self-test codes listed in Table 1-2 is returned.

**Table 1-2. HP E1445A Self-Test Codes**

<table>
<thead>
<tr>
<th>Self-Test Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Test passed</td>
</tr>
<tr>
<td>1</td>
<td>Test failed. An error message describes the failure.</td>
</tr>
</tbody>
</table>
HP BASIC Program Example (SLFTST)

1 !RE-STORE "SLFTST"
10 !Send the self-test command, enter and display the result.
20 DIM Message$[256]
30 OUTPUT 70910;"*TST?"
40 ENTER 70910;Rslt
50 IF Rslt <>0 THEN
60 REPEAT
70 OUTPUT 70910;"SYST:ERR?"
80 ENTER 70910;Code,Message$
90 PRINT Code,Message$
100 UNTIL Code=0
110 END IF
120 PRINT Rslt
130 END

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, SLFTST.FRM, is in directory “VBPROG” and the Visual C/C++ example program, SLFTST.C, is in directory “VCPROG” on the CD that came with your HP E1445A.

Resetting and Clearing the AFG

The commands used to reset and clear the AFG are:

*RST
*CLS

Resetting the AFG sets it to its power-on configuration and clearing the AFG clears its Status Registers. Status Register programming is covered in Chapter 9.

HP BASIC Program Example (RSTCLS)

1 !RE-STORE "RSTCLS"
10 !Assign an I/O path between the computer and AFG.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !Call the subprogram
50 CALL Rst_cls
60 END
70 !
80 SUB Rst_cls
90 Rst_cls: !subprogram which resets and clears the AFG.
100 COM @Afg
110 OUTPUT @Afg;"*RST;*CLS;*OPC?" !reset and clear the AFG
120 ENTER @Afg;Complete
130 SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, RSTCLS.FRM, is in directory “VBPROG” and the Visual C/C++ example program, RSTCLS.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Querying the Power-On/Reset Configuration

After resetting the HP E1445A or cycling power, the AFG parameters are set to their power-on values. These values are listed in Table B-5 in Appendix B.

The command which queries each AFG parameter setting is:

*LRN?

HP BASIC Program Example (LRN)

```
1 !RE-STORE "LRN"
10 !Assign an I/O path between the computer and AFG.
20 ASSIGN @Afg to 70910
30 !Call the subprogram
40 Lrn_conf(@Afg)
50 END
60 !
70 SUB Lrn_conf(@Afg)
80 Lrn_conf: !subprogram which queries the AFG configuration
90 DIM Lrn$[5000]
100 OUTPUT @Afg;"*LRN?"
110 ENTER @Afg;Lrn$
120 Lrn$=Lrn$&";"
130 REPEAT
140 I=POS(Lrn$,";"
150 PRINT Lrn$[1:I-1]
160 Lrn$=Lrn$[I+1]
170 UNTIL Lrn$=""
180 SUBEND
```

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, LRN.FRM, is in directory “VBPROG” and the Visual C/C++ example program, LRN.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Checking for Errors

The following HP BASIC program shows the lines and subprogram which are added to the HP BASIC programs to check for errors. Line 140 clears the AFG Standard Event Status Register. Lines 150 and 160 unmask the appropriate bits in the AFGs Status Byte Register and Standard Event Status Register.

When an error occurs, the subprogram "Errmsg" reads the AFG error queue and displays the code and message. Note that line 310 is used as an "end of statement" should a syntax error occur among coupled commands. Otherwise, line 320 would serve as the end of statement and the ABORT command would be ignored by the AFG parser.

Note

An alternative HP BASIC error checking program can be found in the C-Size VXIbus Systems Configuration Guide. Error checking routines for Visual C/C++ language and Visual BASIC programs are found in programs ARBWAVE.C and ARBWAVE.FRM, listed previously in this chapter.

HP BASIC Program Example (ERRORCHK)

```hp-bas
1 !RE-STORE"ERRORCHK"
2 !This program represents the method used to check for programming errors in HP BASIC programs.
3 !
4 !Assign I/O path between the computer and E1445A.
10 ASSIGN @Afg TO 70910
20 COM @Afg
30 !Define branch to be taken when an E1445A error occurs.
40 !Enable HP-IB interface to generate an interrupt when an error occurs.
50 ON INTR 7 CALL Errmsg
60 !Clear all bits in the standard event status register, unmask the standard event status group summary bit in the E1445A status byte
70 OUTPUT @Afg;"*CLS"
80 OUTPUT @Afg;"*SRE 32"
90 !Clear all bits in the standard event status register, unmask the query error, device dependent error, execution error, and command error bits
100 OUTPUT @Afg;"*ESE 60"
110 !Clear all bits in the standard event status register, unmask the query error, device dependent error, execution error, and command error bits
120 !Clear all bits in the standard event status register, unmask the query error, device dependent error, execution error, and command error bits
130 !Clear all bits in the standard event status register, unmask the query error, device dependent error, execution error, and command error bits
140 !Clear all bits in the standard event status register, unmask the query error, device dependent error, execution error, and command error bits
150 !Clear all bits in the standard event status register, unmask the query error, device dependent error, execution error, and command error bits
160 !Clear all bits in the standard event status register, unmask the query error, device dependent error, execution error, and command error bits
170 !Clear all bits in the standard event status register, unmask the query error, device dependent error, execution error, and command error bits
180 !Clear all bits in the standard event status register, unmask the query error, device dependent error, execution error, and command error bits
190 !Clear all bits in the standard event status register, unmask the query error, device dependent error, execution error, and command error bits
200 WAIT .1 !Wait error branch to occur before turning intr off
210 OFF INTR 7
220 END
```

Continued on Next Page
230 !
240 SUB Errmsg
250 Errmsg: !Subprogram which displays E1445 programming errors
260 COM @Afg
270 DIM Message$[256]
280 !Read AFG status byte register and clear service request bit
290 B=SPOLL(@Afg)
300 !End of statement if error occurs among coupled commands
310 OUTPUT @Afg;"
320 OUTPUT @Afg;"ABORT" !abort output waveform
330 REPEAT
340 OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
350 ENTER @Afg;Code,Message$
360 PRINT Code,Message$
370 UNTIL Code=0
380 STOP
390 SUBEND
Generating Sine Waves

From the power-on/reset configuration you can output a 0.16187 Vp, 10 kHz sine wave by setting the AFG to the wait-for-arm state with the \texttt{INITiate:IMMediate} command. This is done with the \texttt{RSTSINE} program.

![Sine Wave Graph]

\textbf{HP BASIC Program Example (RSTSINE)}

\begin{verbatim}
1 !RE-STORE"RSTSINE"
2 !This program outputs a sine wave based on the reset conditions
3 !of the AFG.
4 !
5 !Assign an I/O path between the computer and AFG.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !
50 !Reset the AFG
60 CALL Rst
70 OUTPUT @Afg;"INIT:IMM" !output sine wave using reset conditions
80 END
90 !
100 SUB Rst
110 Rst: !subprogram which resets the AFG.
120 COM @Afg
130 OUTPUT @Afg;"RST:""OPC?" !reset the AFG
140 ENTER @Afg;Complete
150 SUBEND
\end{verbatim}

\textbf{Visual BASIC and Visual C/C++ Program Versions}

The Visual BASIC example program, RSTSINE.FRM, is in directory “VBPROG” and the Visual C/C++ example program, RSTSINE.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Chapter 2
Generating Standard Waveforms

Chapter Contents

This chapter shows how to generate standard waveforms (sinusoid, square, triangle, and ramp) using the HP E1445A 13-Bit Arbitrary Function Generator (called the "AFG").

The following sections show how to generate standard waveforms, how to setup the AFG for different output loads, how to select the output amplitude units (for example, V, Vpeak, etc.), and how to set the waveform amplitude and offset. The sections are as follows:

- Standard Waveforms Flowchart . . . . . . . . . . . . . . . . . . . . . . Page 54
- Generating DC Voltages . . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 56
- Generating Sine Waves . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 58
- Generating Square Waves . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 61
- Generating Triangle/Ramp Waves . . . . . . . . . . . . . . . . . . . . Page 65
- Selecting the Output Loads . . . . . . . . . . . . . . . . . . . . . . . . . . Page 69
- Selecting the Amplitude Levels and Output Units . . . . . . . Page 72
- Using Phase Modulation . . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 75
- Standard Waveform Program Comments . . . . . . . . . . . . . . . Page 78
  - Sinusoid Function Requirements . . . . . . . . . . . . . . . . . . . . Page 78
  - Reference Oscillator Sources . . . . . . . . . . . . . . . . . . . . . . Page 78
  - Sample Sources . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 78
  - DDS Frequency Generator Ranges . . . . . . . . . . . . . . . . . . Page 79
  - Number of Points versus Frequency . . . . . . . . . . . . . . . . . Page 79
  - Output Load Comments . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 79
  - Output Units Comments . . . . . . . . . . . . . . . . . . . . . . . . . . Page 80
  - Selecting the Deviation Units for Phase Modulation . . . . . Page 80
  - Using MINimum and MAXimum Parameters . . . . . . . . Page 81

Note
For information on how the AFG electronically generates the Standard Waveforms, refer to Chapter 10 of this manual.
The flowchart in Figure 2-1 shows the sequence used to generate standard waveforms. The reset (power-on) values of each command are also noted on the flowchart. The programs in this chapter begin with a reset (the IEEE 488.2 RST command) which places the AFG into its power-on state. Thus, the programs do not execute all of the commands on the flowchart. Remove the flowchart from the binder for easy accessibility. Refer to the flowchart while doing the examples in this chapter, if desired.
Figure 2-1. Commands for Generating Standard Waveforms

(continued from previous page)
Generating DC Voltages

The DCVOLTS program outputs a +5 Vdc voltage. The commands are:

1. **Reset the AFG**
   
   *RST
   
   This command aborts any waveform output and selects the sinusoid function, output impedance, and output load to 50 Ω.

2. **Select the Function**
   
   [SOURce:]FUNCTION[:SHAPE] DC
   
   This command selects the DC function.

3. **Set the Amplitude**
   
   [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>
   
   This command specifies the amplitude. Refer to the section called “Selecting the Amplitude Levels and Output Units” on page 72 for more information.

**HP BASIC Program Example (DCVOLTS)**

```
1 !RE-STORE"DCVOLTS"
2 !This program outputs a +5V DC voltage.
3  !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40  !
50 !Set up error checking
60 ON INTR 7 CALL Errmsg
70 ENABLE INTR 7,2
80 OUTPUT @Afg;"CLS"
90 OUTPUT @Afg;"SRE 32"
100 OUTPUT @Afg;"ESE 60"
110  !
120 !Call the subprograms
130 CALL Rst
140 CALL Dc_volts
150  !
160 WAIT .1 !allow interrupt to be serviced
170 OFF INTR 7
180 END
190  !
200 SUB Dc_volts
210 Dc_volts: !subprogram which outputs a dc voltage
220  COM @Afg
230 OUTPUT @Afg;"SOUR:FUNC:SHAPE DC;" ; !function
240 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5V" ; !amplitude

Continued on Next Page
```
SUBEND
!
SUB Rst
Rst: !Subprogram which resets the E1445.
COM @Afg
OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
ENTER @Afg;Complete
SUBEND
!
SUB Ermsg
Ermsg: !Subprogram which displays E1445 programming errors
COM @Afg
DIM Message$[256]
!Read AFG status byte register and clear service request bit
B=SPOLL(@Afg)
!End of statement if error occurs among coupled commands
OUTPUT @Afg;"
OUTPUT @Afg;"ABORT" !abort output waveform
REPEAT
OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
ENTER @Afg;Code,Message$
PRINT Code,Message$
UNTIL Code=0
STOP
SUBEND

Visual BASIC and
Visual C/C++ Program
Versions

The Visual BASIC example program, DCVOLTS.FRM, is in directory
“VBPROG” and the Visual C example program, DCVOLTS.C, is in
directory “VCPROG” on the CD that came with your HP E1445A.
Generating Sine Waves

The SINEWAVE program outputs a Sine Wave at 1 kHz and 5 V output level. The commands are:

1. **Reset the AFG**
   *RST
   This command aborts any waveform output and selects the sinusoid function, output impedance, and output load to 50 Ω.

2. **Set the Waveform Frequency**
   [SOURce:]FREQuency[1][:FIXed] <frequency>
   This command specifies the waveform frequency. You must use the direct synthesis frequency generator for the sinusoid function. Refer to Table B-3 in Appendix B for the frequency limits.

3. **Select the Function**
   [SOURce:]FUNCtion[:SHAPe] SINusoid
   This command selects the sinusoid function. (Although *RST automatically selects this function, it is selected here for good programming practice.)

4. **Set the Amplitude**
   [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>
   This command specifies the amplitude. Refer to the section called “Selecting the Amplitude Levels and Output Units” on page 72 for more information.

5. **Initiate the Waveform**
   INITiate[:IMMediate]
   This command generates an immediate output with the arm source set to IMMediate. Refer to Chapter 5 for triggering information.
HP BASIC Program Example (SINEWAVE)

1 !RE-STORE"SINEWAVE"
2 !The following program generates a 1 kHz, 5 Vp sine wave.
3 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !
50 !Set up error checking
60 ON INTR 7 CALL Errmsg
70 ENABLE INTR 7;2
80 OUTPUT @Afg;"*CLS"
90 OUTPUT @Afg;"*SRE 32"
100 OUTPUT @Afg;"*ESE 60"
110 !
120 !Call the subprograms
130 CALL Rst
140 CALL Sine_wave
150 !
160 WAIT .1 !allow interrupt to be serviced
170 OFF INTR 7
180 END
190 !
200 SUB Sine_wave
210 Sine_wave: !Subprogram which outputs a sine wave
220 COM @Afg
230 OUTPUT @Afg;"SOUR:FREQ1:FIX 1E3;"; !frequency
240 OUTPUT @Afg;"SOUR:FUNC:SHAP SIN;"; !function
250 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5V" !amplitude
260 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
270 SUBEND
280 !
290 SUB Rst
300 Rst: !Subprogram which resets the E1445.
310 COM @Afg
320 OUTPUT @Afg;"RST;"OPC?" !reset the AFG
330 ENTER @Afg;Complete
340 SUBEND
350 !
360 SUB Errmsg
370 Errmsg: !Subprogram which displays E1445 programming errors
380 COM @Afg
390 DIM Message$[256]
400 !Read AFG status byte register and clear service request bit
410 B=SPOLL(@Afg)
420 !End of statement if error occurs among coupled commands
430 OUTPUT @Afg;""
440 OUTPUT @Afg;"ABORT" !abort output waveform
450 REPEAT

Continued on Next Page
460 OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
470 ENTER @Afg;Code,Message$
480 PRINT Code,Message$
490 UNTIL Code=0
500 STOP
510 SUBEND

Visual BASIC and Visual C/C++ Program Versions
The Visual BASIC example program, SINEWAVE.FRM, is in directory “VBPROG” and the Visual C example program, SINEWAVE.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Generating Square Waves

The SQUWAVE program outputs a square wave at 1 Mhz, 4 V output level, and +1 V offset. The commands are:

1. **Reset the AFG**
   *RST*
   This command aborts any waveform output and selects the 42.9 MHz reference oscillator source, DDS sample source (that is, trigger start source), sinusoid function, arm start immediate, 0 V offset, and a 50 Ω output impedance and output load.

2. **Select the Reference Oscillator**
   [SOURce:]ROSCillator:SOURce INTernal[1]
   This command selects the reference oscillator source (see “Reference Oscillator Sources” on page 78). (Although *RST selects 42.9 MHz reference oscillator, it is selected here for good programming practice.)

3. **Select the Sample Source**
   TRIGger:STARt:SOURce INTernal[1]
   This command selects the sample source (that is, trigger start source). (Although *RST selects trigger start source that selects the DDS frequency generator, it is selected here for good programming practice.) The SQUare function can use any of the trigger start sources (see “Sample Sources” on page 78).

4. **Set the Frequency Range**
   [SOURce:]FREQuency[1]:RANGe <range>
   This command specifies the square wave upper frequency limit (see “DDS Frequency Generator Ranges” on page 79). (Since *RST automatically sets the range to the lower range, it is executed in this program for good programming practice.)
5. **Set the Frequency**
   
   `[SOURce:]FREQuency[1][:FIXed] <frequency>`
   
   This command specifies the frequency. Refer to Table B-3 in 
   Appendix B for the frequency limits.

6. **Select the Function**
   
   `[SOURce:]FUNCTION[:SHApe] SQUare`
   
   This command selects the square wave function.

7. **Select the Square Wave Polarity**
   
   `[SOURce:]RAMP:POLarity INVerted`
   
   This command selects the square wave polarity. For NORMal, the 
   initial voltage goes positive. For INVerted, the initial voltage goes 
   negative.

8. **Set the Amplitude**
   
   `[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>`
   
   This command specifies the amplitude. Refer to the section called 
   “Selecting the Amplitude Levels and Output Units” on page 72 for 
   more information.

9. **Set the Offset**
   
   `[SOURce:]VOLTage[:LEVel][:IMMediate]OFFSet <offset>`
   
   This command specifies the offset. Refer to the section called 
   “Selecting the Amplitude Levels and Output Units” on page 72 for 
   more information.

10. **Initiate the Waveform**
   
    `INITiate[:IMMediate]`
   
    This command generates an immediate output with the arm source 
    set to IMMediate. Refer to Chapter 5 for triggering information.
HP BASIC Program Example (SQUWAVE)

1 !RE-STORE"SQUWAVE"
2 !This program outputs a 1 MHz, 4V square wave with a 1V DC offset.
3 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !
50 !Set up error checking
60 ON INTR 7 CALL Errmsg
70 ENABLE INTR 7;2
80 OUTPUT @Afg;"CLS"
90 OUTPUT @Afg;"SRE 32"
100 OUTPUT @Afg;"ESE 60"
110 !
120 !Call the subprograms
130 CALL Rst
140 CALL Squ_wave
150 !
160 WAIT .1 !allow interrupt to be serviced
170 OFF INTR
180 END
190 !
200 SUB Squ_wave
210 Squ_wave: !Subprogram which outputs a square wave
220 COM @Afg
230 OUTPUT @Afg;"SOUR:ROSC:SOUR INT1;"; !reference oscillator
240 OUTPUT @Afg;":TRIG:STAR:SOUR INT1;"; !trigger source
250 OUTPUT @Afg;":SOUR:FREQ:RANG 0;"; !frequency range
260 OUTPUT @Afg;":SOUR:FREQ:FIX 1E6;"; !frequency
270 OUTPUT @Afg;":SOUR:FUNC:SHAP SQU;"; !function
280 OUTPUT @Afg;":SOUR:RAMP:POL INV;"; !polarity (more negative)
290 OUTPUT @Afg;":SOUR:VOLT:LEV:IMM:AMPL 4V;"; !amplitude
300 OUTPUT @Afg;":SOUR:VOLT:LEV:IMM:OFFS 1V" !dc offset
310 OUTPUT @Afg;":INIT:IMM" !wait-for-arm state
320 SUBEND
330 !
340 SUB Rst
350 Rst: !Subprogram which resets the E1445.
360 COM @Afg
370 OUTPUT @Afg;"RST;*OPC?" !reset the AFG
380 ENTER @Afg;Complete
390 SUBEND
400 !
410 SUB Errmsg
420 Errmsg: !Subprogram which displays E1445 programming errors
430 COM @Afg
440 DIM Message$[256]
450 !Read AFG status byte register and clear service request bit

Continued on Next Page
460     B=SPOLL(@Afg)
470     !End of statement if error occurs among coupled commands
480     OUTPUT @Afg;""
490     OUTPUT @Afg;"ABORT"   !abort output waveform
500     REPEAT
510     OUTPUT @Afg;"SYST:ERR?"    !read AFG error queue
520     ENTER @Afg;Code,Message$
530     PRINT   Code,Message$
540     UNTIL Code=0
550     STOP
560     SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, SQUWAVE.FRM, is in directory “VBPROG” and the Visual C example program, SQUWAVE.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Generating Triangle/Ramp Waves

The TRIWAVE program outputs a triangle wave at 10 kHz, 4 V output level, and +1 V offset. The commands are:

1. **Reset the AFG**
   
   *RST
   
   This command aborts any waveform output and selects the 42.9 MHz reference oscillator source, DDS sample source (that is, trigger start source), sinusoid function, arm start immediate, 0 V offset, and a 50 Ω output impedance and output load.

2. **Select the Reference Oscillator**

   [SOURce:]ROSCillator:SOURce INTernal[1]

   This command selects the reference oscillator source (see “Reference Oscillator Sources” on page 78). (Although *RST selects 42.9 MHz reference oscillator, it is selected here for good programming practice.)

3. **Select the Sample Source**

   TRIGger:STARt:SOURce INTernal[1]

   This command selects the sample source (that is, trigger start source). (Although *RST selects trigger start source that selects the DDS frequency generator, it is selected here for good programming practice.) The TRIangle/RAMP functions can use any of the trigger start sources (see “Sample Sources” on page 78).

4. **Set the Frequency Range**

   [SOURce:]FREQuency[1]:RANGe <range>

   This command specifies the triangle/ramp wave upper frequency limit (see “DDS Frequency Generator Ranges” on page 79). (Since *RST automatically sets the range to the lower range, it is executed in this program for good programming practice.)
5. **Set the Frequency**  
   [SOURce:]FREQuency[1]:FIXed <frequency>  
   This command specifies the frequency. Refer to Table B-3 in Appendix B for the frequency limits.

6. **Select the Function**  
   [SOURce:]FUNCTION[:SHAPE] TRIangle  
   This command selects the TRIangle function. For the RAMP function, use the RAMP parameter instead of the TRIangle parameter.

7. **Set the Number of Ramp Points**  
   [SOURce:]RAMP:POINts <number>  
   This command specifies the number of ramp points. The more points give better resolution but lower frequency response.

8. **Select the Triangle Wave Polarity**  
   [SOURce:]RAMP:POLarity INVerted  
   This command selects the polarity of the TRIangle/RAMP wave. Use NORMal for the initial voltage to go positive. Use INVerted for the initial voltage to go negative.

9. **Set the Amplitude**  
   [SOURce:]VOLTage[:LEV]e[1]:IMMediate[:AMPLitude] <amplitude>  
   This command specifies the amplitude. Refer to the section called “Selecting the Amplitude Levels and Output Units” on page 72 for more information.

10. **Set the Offset**  
    [SOURce:]VOLTage[:LEV]e[1]:IMMediate:OFFSet <offset>  
    This command specifies the offset. Refer to the section called “Selecting the Amplitude Levels and Output Units” on page 72 for more information.

11. **Initiate the Waveform**  
    INITiate[:IMMediate]  
    This command generates an immediate output with the arm source set to IMMEDIATE. Refer to Chapter 5 for triggering information.
HP BASIC Program Example (TRIWave)

1 !RE-STORE"TRIWAVE"
2 !This program outputs a 200 point, 10 kHz, 4V triangle wave
3 !with a 1V DC offset.
4 !
5 !Assign I/O path between the computer and E1445A.
6 ASSIGN @Afg TO 70910
7 COM @Afg
8 !
9 !Set up error checking
10 ON INTR 7 CALL Errmsg
11 ENABLE INTR 7;2
12 OUTPUT @Afg:"CLS"
13 OUTPUT @Afg:"SRE 32"
14 OUTPUT @Afg:"ESE 60"
15 !
16 !Call the subprograms
17 CALL Rst
18 CALL Tri_wave
19 !
20 WAIT .1 !allow interrupt to be serviced
21 OFF INTR
22 END
23 !
24 SUB Tri_wave
25 !Subprogram which outputs a triangle wave
26 COM @Afg
27 OUTPUT @Afg;"SOUR:ROSC:SOUR INT1;"; !reference oscillator
28 OUTPUT @Afg;"TRIG:STAR:SOUR INT1;"; !trigger source
29 OUTPUT @Afg;"SOUR:FREQ1:RANG 0;"; !frequency range
30 OUTPUT @Afg;"SOUR:FREQ1:FIX 10E3;"; !frequency
31 OUTPUT @Afg;"SOUR:FUNC:SHAP TRI;"; !function
32 OUTPUT @Afg;"SOUR:RAMP:POIN 200;"; !waveform points
33 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 4V;"; !amplitude
34 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:OFFS 1V" !dc offset
35 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
36 SUBEND
37 !
38 SUB Rst
39 !Subprogram which resets the E1445.
40 COM @Afg
41 OUTPUT @Afg;"RST;OPC?" !reset the AFG
42 ENTER @Afg:Complete
43 SUBEND
44 !
45 SUB Errmsg
46 !Subprogram which displays E1445 programming errors
47 COM @Afg
48 DIM Message$[256]
49 !
50 !
51 !Continued on Next Page

Chapter 2 Generating Standard Waveforms 67
450 !Read AFG status byte register and clear service request bit
460 B=SPOLL(@Afg)
470 !End of statement if error occurs among coupled commands
480 OUTPUT @Afg;:"
490 OUTPUT @Afg;:"ABORT"	! abort output waveform
500 REPEAT
510 OUTPUT @Afg;:"SYST:ERR?"	! read AFG error queue
520 ENTER @Afg;Code,Message$
530 PRINT Code,Message$
540 UNTIL Code=0
550 STOP
560 SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, TRIWAVE.FRM, is in directory “VBPROG” and the Visual C example program, TRIWAVE.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Selecting the Output Loads

The OUTPLOAD program sets the AFG’s output impedance to the output load value of 75 Ω. The commands are:

1. **Setup the AFG**
   *RST*
   Use the *RST command to setup the AFG. You can also use the commands listed in the previous sections of this chapter (“Generating Sine Waves” on page 58) to setup the AFG.

2. **Set the Amplitude**
   \[\text{SOURce:]VOLTage[:LEVel[:IMMediate][:AMPLitude]} <amplitude>\]
   This command specifies the amplitude. Refer to the section called “Selecting the Amplitude Levels and Output Units” on page 72 for more information.

3. **Select the Auto Load On, Off, or Once**
   \[\text{OUTPut[1]:LOAD:AUTO} <mode>\]
   With this command, the assumed load applied to the AFG’s "Output 50/75 Ω" terminals tracks the AFG output impedance. The modes are:
   - **ON** – load value tracks output impedance
   - **OFF** – load value does not track output impedance
   - **ONCE** – load value tracks output impedance once and then goes to OFF

4. **Select the Output Impedance**
   \[\text{OUTPut[1]:IMPedance} <impedance>\]
   This command selects the AFG output impedance. The AFG output impedance can be either 50 Ω or 75 Ω.
5. **Select the Output Load Value**

OUTPut[1]:LOAD <load>

This command selects the load value expected at the "Output 50/75 Ω" terminals. The values are:

- **50** – for 50 Ω loads; must be same as output impedance.
- **75** – for 75 Ω loads; must be same as output impedance.
- **9.9E+37 or INFinity** – for open circuit; output value is twice the normal matched load output value.

6. **Initiate the Waveform**

INITiate[:IIMMediate]

This command generates an immediate output with the arm source set to IMMEDIATE. Refer to Chapter 5 for triggering information.

**HP BASIC Program Example (OUTPLOAD)**

```
1 !RE-STORE"OUTPLOAD"
2 !This program sets the AFG's output impedance and output load
3 !to 75 ohms.
4 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910.
30 COM @Afg
40 !
50 !Set up error checking
60 ON INTR 7 CALL Errmsg
70 ENABLE INTR 7;2
80 OUTPUT @Afg;"CLS"
90 OUTPUT @Afg;"SRE 32"
100 OUTPUT @Afg;"ESE 60"
110 !
120 !Call the subprograms
130 CALL Rst
140 CALL Out_load
150 !
160 WAIT .1 !allow interrupt to be serviced
170 OFF INTR 7
180 END
190 !
200 SUB Out_load
210 Out_load: !Subprogram which sets the output load
220 COM @Afg
230 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5V;"; !amplitude
240 OUTPUT @Afg;"OUTP:LOAD:AUTO OFF;"; !decouple load from impedance
250 OUTPUT @Afg;"OUTP:IMP 75;"; !output impedance
260 OUTPUT @Afg;"OUTP:LOAD 75" !output load
270 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
280 SUBEND
290 !
```

*Continued on Next Page*
300 SUB Rst
310 Rst: !Subprogram which resets the E1445.
320 COM @Afg
330 OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
340 ENTER @Afg;Complete
350 SUBEND
360 !
370 SUB Errmsg
380 Errmsg: !Subprogram which displays E1445 programming errors
390 COM @Afg
400 DIM Message$[256]
410 !Read AFG status byte register and clear service request bit
420 B=SPOLL(@Afg)
430 !End of statement if error occurs among coupled commands
440 OUTPUT @Afg;"
450 OUTPUT @Afg;"ABORT" !abort output waveform
460 REPEAT
470 OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
480 ENTER @Afg;Code,Message$
490 PRINT Code,Message$
500 UNTIL Code=0
510 STOP
520 SUBEND

**Visual BASIC and Visual C/C++ Program Versions**

The Visual BASIC example program, OUTPLOAD.FRM, is in directory “VBPROG” and the Visual C example program, OUTPLOAD.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Selecting the Amplitude Levels and Output Units

The OUTPUNIT program shows how to set the output amplitude using the VPP (volts peak-to-peak) output unit. The commands are:

1. **Reset the AFG**
   
   *RST

   This command aborts any waveform output and selects the 42.9 MHz reference oscillator source, DDS sample source (that is, trigger start source), sinusoid function, arm start immediate, 0 V offset, and a 50 Ω output impedance and output load.

2. **Select the Output Units**

   `[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude]:UNIT[:VOLTage] <units>`

   This command selects the following output units:
   
   - **V** = Volts
   - **VPK** = Volts peak
   - **VPP** = Volts peak-to-peak
   - **VRMS** = Volts rms
   - **W** = Watts
   - **DBM** | **DBMW** = dB referenced to 1 milliwatt

   These units are assumed only if no other units are specified in the `[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>` command. The output units are only valid for amplitude and not offsets (volts is assumed for offsets).

3. **Set the Amplitude and the Offset**

   `[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>`
   
   `[SOURce:]VOLTage[:LEVel][:IMMediate]:OFFSet <offset>`

   These commands specify the amplitude and offset. Refer to Table B-4 in Appendix B for the amplitude limits. The maximum value of the combined amplitude and offset voltages must remain within the 6.025 V limit.

4. **Initiate the Waveform**

   `INITiate[:IMMediate]`

   This command generates an immediate output with the arm source set to IMMediate. Refer to Chapter 5 for triggering information.
HP BASIC Program Example (OUTPUNIT)

1  !RE-STORE"OUTPUNIT"
2  !This programs sets the output amplitude units to volts peak-to-peak
3  !
10  !Assign I/O path between the computer and E1445A.
20  ASSIGN @Afg TO 70910
30  COM @Afg
40  !
50  !Set up error checking
60  ON INTR 7 CALL Errmsg
70  ENABLE INTR 7;2
80  OUTPUT @Afg;"*CLS"
90  OUTPUT @Afg;"*SRE 32"
100 OUTPUT @Afg;"*ESE 60"
110  !
120  !Call the subprograms
130  CALL Rst
140  CALL Out_unit
150  !
160  WAIT .1 !allow interrupt to be serviced
170  OFF INTR 7
180  END
190  !
200  SUB Out_unit
210  Out_unit:!Subprogram which sets the amplitude units
220  COM @Afg
230  OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL:UNIT:VOLT VPP" !amplitude units
240  OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 8;"; !amplitude
250  OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:OFFS 1" !offset
260  OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
270  SUBEND
280  !
290  SUB Rst
300  Rst: !Subprogram which resets the E1445.
310  COM @Afg
320  OUTPUT @Afg;"RST;"OPC?" !reset the AFG
330  ENTER @Afg;Complete
340  SUBEND
350  !
360  SUB Errmsg
370  Errmsg: !Subprogram which displays E1445 programming errors
380  COM @Afg
390  DIM Message$[256]
400  !Read AFG status byte register and clear service request bit
410  B=SPOLL(@Afg)
420  !End of statement if error occurs among coupled commands
430  OUTPUT @Afg;""
440  OUTPUT @Afg;"ABORT" !abort output waveform
450  REPEAT

Continued on Next Page
460  OUTPUT @Afg;“SYST:ERR?”  lread AFG error queue
470  ENTER @Afg;Code,Message$
480  PRINT  Code,Message$
490  UNTIL Code=0
500  STOP
510  SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, OUTPUNIT.FRM, is in directory “VBPROG” and the Visual C example program, OUTPUNIT.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Using Phase Modulation

Phase modulation allows you to change the output deviation or phase of a sinusoid wave while it is output. This only works in the sinusoid function. The PHS_MOD program shows how change the deviation from 0° to 180°. The commands are:

1. **Reset the AFG**
   
   `*RST`

   This command aborts any waveform output and selects the 42.9 MHz reference oscillator source, DDS sample source (that is, trigger start source), sinusoid function, arm start immediate, 0 V offset, and a 50 Ω output impedance and output load.

2. **Set the Waveform Frequency**

   `[SOURce:]FREQuency[1]:FIXed <frequency>`

   This command specifies the waveform frequency. You must use the direct synthesis frequency generator for the sinusoid function. Refer to Table B-3 in Appendix B for the frequency limits.

3. **Select the Phase Modulation Source**

   `[SOURce:]PM:SOURce <source>`

   This command sets the phase modulation source. The command determines which source to use for a phase change. The available sources are:
   - **INTernal** – `[SOURce:]PM:DEViation` sets the deviation angle (power-on value)
   - **DPORT** – The front panel “Dig Port” connector
   - **LBUS** – The VXI Local Bus
   - **VXI** – The VXI Backplane

4. **Enable Phase Modulation**

   `[SOURce:]PM:STATe <mode>`

   This command turns phase modulation on or off. A "1" (one) or "ON" turns it on, and a "0" (zero) or "OFF" turns it off.
5. **Select the Function**
[SOURce:]FUNCTION[:SHAPE] SINusoid
This command selects the sinusoid function. (Although *RST automatically selects this function, it is selected here for good programming practice.)

6. **Set the Amplitude**
[SOURce:]VOLTage[:LEVEL][:IMMediate][:AMPLitude] <amplitude>
This command specifies the amplitude. Refer to the section called “Setting the Amplitude Levels and Output Units” on page 72 for more information.

7. **Set the Phase Modulation Deviation**
[SOURce:]PM:DEViation <phase>
This command sets the deviation angle. The angle can either be in radians or degrees. The values can be from $-\pi$ to $+\pi$ or $-180^\circ$ to $+180^\circ$. (See “Selecting the Deviation Units for Phase Modulation” on page 80 to select the different units).

8. **Initiate the Waveform**
INITiate[:IMMediate]
This command generates an immediate output with the arm source set to IMMediate. Refer to Chapter 5 for triggering information.

**HP BASIC Program Example (PHS_MOD)**

```hpbasic
1 !RE-STORE"PHS.Mod"
2 !The following program shifts the phase of the output sine wave
3 !from 0 degrees to 180 degrees.
4 !Assign I/O path between the computer and E1445A.
10 ASSIGN @Afg TO 70910
20 COM @Afg
40 !
50 !Set up error checking
60 ON INTR 7 CALL Ermsg
70 ENABLE INTR 7:2
80 OUTPUT @Afg:"CLS"
90 OUTPUT @Afg:"SRE 32"
100 OUTPUT @Afg:"ESE 60"
110 !
120 !Call the subprograms
130 CALL Rst
140 CALL Phase_mod
150 !
160 WAIT .1 !allow interrupt to be serviced
170 OFF INTR 7
180 END
```

Continued on Next Page
Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, PHS_MOD.FRM, is in directory “VBPROG” and the Visual C example program, PHS_MOD.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Standard Waveform Program Comments

The following comments give additional details on the program examples in this chapter.

Sinusoid Function Requirements

The sinusoid requires that the sample source (see “Sample Sources” below) is set to INTernal[1] (that is, TRIGger:STARt:SOURce INTernal[1]). This selects the DDS frequency generator. No other sample source can generate a sinewave.

Reference Oscillator Sources

- The SINusoid, SQUare, TRIangle, and RAMP functions can use any of the reference oscillator sources. The sources selected by [SOURce:]ROSCillator:SOURce are:
  - INTernal[1] – 42.94967296 MHz (power-on value)
  - INTernal2 – 40 MHz
  - CLK10 – 10 MHz (the VXIbus CLK line)
  - EXTernal – User provided value (the front panel “Ref/Smpl In” BNC)
  - ECLTrg0 or 1 – User provided value (the VXIbus ECL trigger lines)

- If using either the EXTernal or ECLTrg0 or 1 reference oscillator sources, enter the source frequency to the AFG using [SOURce:]ROSCillator:FREQuency:EXTernal <frequency>.

- For best frequency linearity, use the 42.9 MHz (that is, INTernal[1]) reference oscillator source with the DDS (frequency1) frequency generator. This combination provides .01 Hz resolution. For higher frequency values, use the 40 MHz (that is, INTernal2) reference oscillator source with the Divide-by-N (frequency2) frequency generator. Use the other sources for custom frequency values. However, any reference oscillator sources can be used with any frequency generator.

Sample Sources

- The SINUsoid function only operates with the INTernal[1] sample source.

- The SQUare, TRIangle, and RAMP functions operate with any of the sample sources selected by the TRIGger:STARt:SOURce command. The functions can use either the DDS (frequency1) frequency generator or the Divide-by-N (frequency2) frequency generator for waveform generation. However, the DDS frequency generator gives better frequency resolution and should be used for the these functions. The Divide-by-N frequency generator should be used for arbitrary waveform generation where high frequency values are needed. The different sample sources are:
  - INTernal[1] (power-on value; selects the DDS frequency generator)
  - INTernal2 (selects the Divide-by-N frequency generator)
  - BUS (the HP-IB GET or *TRG commands)
  - EXTernal (the front panel “Ref/Smpl In” BNC)
ECLTrg0 or 1 (the VXIbus ECL trigger lines)  
HOLD (suspends sample generation)  
TTLTrg0 through 7 (the VXIbus TTL trigger lines)

**DDS Frequency Generator Ranges**

The [SOURce:]FREQuency[1]:RANGe command selects frequency doubling of the DDS frequency generator for the SQUare, TRIangle, and RAMP functions. This command is only used with DDS (frequency1) frequency generator.

The lower frequency setting (that is, normal setting) for the SQUare function is determined by:

- Reference Oscillator frequency / 16

The high frequency setting for the SQUare function is determined by:

- Reference Oscillator frequency / 8

The lower frequency setting (that is, normal setting) for the TRIangle and RAMP functions is determined by:

- Reference Oscillator frequency / 4 / number of points

The doubled frequency setting for the TRIangle and RAMP functions is determined by:

- Reference Oscillator frequency / 2 / number of points

The doubled setting worsens the frequency resolution by a factor of two and introduces some sample rate jitter.

**Number of Points versus Frequency**

The number of points ([SOURce:]RAMP:POINts) determine the maximum frequency of the TRIangle and RAMP functions. The more points results in a lower maximum frequency, but with a better waveform shape. The fewer points results in a higher maximum frequency, but with lower resolution.

**Output Load Comments**

- For correct output amplitude values, the load applied to the AFG "Output 50/75 Ω" output terminals must be the same value as the selected AFG output impedance value.

- To output to an open circuit, execute OUTPut[1]:LOAD INFinity or 9.9E+37 (this sets the auto load value off). The HP E1445A then outputs the correct amplitude and offset for an open circuit. The amplitude and offset range are doubled while resolution worsens by a factor of 2.
Output Units
Comments

- The selected unit type can be overridden by sending a unit suffix with the amplitude command. For example, if the selected unit is VPP, sending:

  \[ \text{[SOURce:]} \text{VOLTage[:LEVel][:IMMediate][:AMPLitude]} \ 5V \]

changes the unit type to volts (that is, V) for that command. However, the default unit type remains in effect for subsequent amplitude commands that are sent without the unit suffix.

- The \( V \) (volts) suffix and VPK (volts peak) suffix generate the same amplitude values for all time varying waveforms like SINusoid, SQUare, TRIangle, and RAMPS.

- The default unit type only applies for amplitudes and not for offsets. The unit for offsets is always specified in \( V \) for volts. For example, executing:

  \[ \text{[SOURce:]} \text{VOLTage:OFFSet} \ .1VPP \]

causes an error. To prevent the error, execute either:

  \[ \text{[SOURce:]} \text{VOLTage:OFFSet} \ 0.1 \]

  or

  \[ \text{[SOURce:]} \text{VOLTage:OFFSet} \ 0.1V \]

- The W, DBM, and DBMW unit types references the amplitude levels to the 50 \( \Omega \) or 75 \( \Omega \) output load values (set by the OUTPut[1]:LOAD command). Thus, the W, DBM, and DBMW values are meaningless and not available when selecting an open circuit load.

Selecting the Deviation Units for Phase Modulation

Use either degrees or radians to change the phase in the phase modulation function. There are two ways to select the units, either send the unit type with the deviation command, like:

\[ \text{[SOURce:]} \text{PM:DEViation} \ -90\text{DEG} \]

  or select the unit type with the unit command like:

\[ \text{[SOURce:]} \text{PM:UNIT:ANGLe DEG} \ or \ [\text{SOURce:}] \text{PM:DEViation} \ -90 \]
Using MINimum and MAXimum Parameters

You can execute many commands (like `[SOURce:]FREQuency[1][:CW]:FIxED]`) using the MINimum or MAXimum parameters instead of a number value. However, when using the parameters, the commands are immediately executed when received. This happens even if the commands are coupled to other commands in a coupling group. This is different than sending the commands with number values, where the commands are executed after a new coupling group is sent.

Thus, if a group of coupled commands are sent where the MINimum and MAXimum parameters conflict with the current AFG setting, the AFG generates an error. This happens even though the commands that follow may set the AFG to a state that does not conflict with the MINimum and MAXimum parameters.

For best results, use values in the commands and do not use the MINimum and MAXimum parameters.
Chapter 3
Generating Arbitrary Waveforms

Chapter Contents

This chapter shows how to generate arbitrary waveforms using the HP E1445A 13-Bit Arbitrary Function Generator (called the “AFG”).

The following sections show how to generate arbitrary waveforms. Also included are example programs that generate various arbitrary waveforms. The sections are as follows:

- Arbitrary Waveforms Flowchart . . . . . . . . . . . . . . . . . . . . . . Page 84
- How the AFG Generates Arbitrary Waveforms . . . . . . . . . . Page 86
- Generating a Simple Arbitrary Waveform . . . . . . . . . . . . . . Page 88
- Executing Several Waveform Segments . . . . . . . . . . . . . . . . Page 93
- Using Different Frequency Generators . . . . . . . . . . . . . . . . Page 99
- Sample Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 104
  - Generating a Sin(X)/X Waveform . . . . . . . . . . . . . . . . . . . Page 105
  - Generating a Damped Sine Wave . . . . . . . . . . . . . . . . . . . . Page 107
  - Generating an Exponential Change/Discharge Waveform . . . . . . . . Page 108
  - Generating a Sine Wave with Spikes . . . . . . . . . . . . . . . . . Page 109
  - Generating a \( \frac{1}{2} \) Rectified Sine Wave . . . . . . . . . . Page 111
  - Generating Noise . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 112
- Arbitrary Waveform Program Comments . . . . . . . . . . . . . . Page 113
  - Determining the Amount of Segment and Sequence Memory . . . . . . . Page 113
  - How to Free Segment and Sequence Memory . . . . . . . . . Page 113
  - Amplitude Effects on Voltage Lists . . . . . . . . . . . . . . . . . Page 113
  - Using DAC Codes to Send Segment Data . . . . . . . . . . Page 114
  - Sending Segment Sequences . . . . . . . . . . . . . . . . . . . . . . Page 114
  - Reference Oscillator Sources . . . . . . . . . . . . . . . . . . . . . Page 115
  - Sample Sources . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 115
  - Frequency1 Generator Range . . . . . . . . . . . . . . . . . . . . . Page 116
  - Returning the Waveform Segment Names . . . . . . . . . . . Page 116
  - Determining the Waveform Segment Size . . . . . . . . . . Page 116
  - Returning the Segment Sequence List Names . . . . . . . . Page 116
  - Returning the Repetition Count List Length . . . . . . . . . Page 116
Arbitrary Waveforms Flowchart

The flowchart in Figure 3-1 shows the commands and the command execution order to generate arbitrary waveforms. The reset (power-on) values of each command are also noted on the flowchart. Note that the IEEE 488.2 *RST command places the AFG into its power-on state. Thus, it may be unnecessary to execute all of the commands on the flowchart. Remove the flowchart from the binder for easy accessibility. Refer to the flowchart while doing the examples in this chapter, if desired.

Figure 3-1. Commands for Generating Arbitrary Waveforms
(continued on next page)
Figure 3-1. Commands for Generating Arbitrary Waveforms
(continued from previous page)
How the AFG Generates Arbitrary Waveforms

Refer to Figure 3-2. An arbitrary waveform consists of two parts, a waveform segment (or all points on a waveform) and a segment sequence. The segments are the actual voltage points of the waveform. The segment sequence determines the order in which one or more waveform segments are output.

To output a waveform, the waveform segment must be stored into the AFG’s segment memory. To do this, you must assign a unique name (use [SOURce:]LIST[1]:SEGMENT:SELECT <name>) for each waveform segment to be stored into memory. This allows you to select one of many waveform segments, which may exist in memory, to be output. Legal names must start with an alphabetic character, but can contain alphabetic, numeric, and underscore (“_”) characters. The names can have a maximum length of 12 characters. The AFG generates an error for duplicate names.

Besides the name, the AFG must also know the size (use [SOURce:]LIST[1]:SEGMENT:DEFINE <length>) of the waveform segment (that is, the number of points). The assigned segment size must be equal to or larger than the actual size of the waveform segment. The AFG generates an error if the waveform segment is larger than the size sent.

The segment values can be either sent as voltage values (use [SOURce:]LIST[1]:SEGMENT:VOLTage <voltage_list>) or DAC (digital-to-analog converter) codes (use [SOURce:]LIST[1]:SEGMENT:VOLTage:DAC <voltage_list>). If sent as voltage values, the AFG converts them to DAC codes before storing them in memory.
The segment sequence determines the order in which the waveform segments in memory are to be output, which order is assigned by the user (use [SOURce:]LIST[1]:SSEQquence:SEQquence <segment_list>).

Each segment sequence must be stored into the AFG’s sequence memory. To do this, you must assign a unique name (use [SOURce:]LIST[1]:SSEQquence:SELect <name>) for each segment sequence to be stored into memory. This allows you to select one of many segment sequences, which may exist in memory, to be output. Legal names must start with an alphabetic character, but can contain alphabetic, numeric, and underscore (“_”) characters. The names can have a maximum length of 12 characters. The AFG generates an error for duplicate names.

The waveform segment names in a segment sequence can either be sent as names or as an address value (see Chapter 7 for more information).

To output a waveform, the AFG sets the DAC to the voltage value of each waveform segment in the segment sequence. The sample frequency determines the rate at which the DAC is set to the different voltage values. Depending on the sample source selected (by TRIGger[:STAr]:SOURce <source>), the sample rate is set by the DDS (frequency1) frequency generator ([SOURce]:FREQuency[1]), Divide-by-N (frequency2) frequency generator ([SOURce]:FREQuency2), or the samples rates of the external sample sources.

The sample rate and the number of points in the waveform segment determine the waveform repetition frequency. The repetition frequency is the sample rate / number of points.
Generating a Simple Arbitrary Waveform

The ARBWAVE program shows how to generate an arbitrary waveform with a single waveform segment. The example generates a 100 point ramp. The AFG stores the waveform segment into segment memory as voltage data points. The commands are:

1. **Reset the AFG**
   
   `*RST`
   
   The `*RST` command aborts any waveform output and sets the AFG to a defined state.

2. **Clear the AFG Memory of All Sequence Data**
   
   `[SOURce:]LIST[1]:SSEQUence:DELete:ALL`
   
   This command clears all segment sequence data stored in the sequence memory (see “How to Free Segment and Sequence Memory” on page 113 for more information).

3. **Clear the AFG Memory of All Segment Data**
   
   `[SOURce:]LIST[1]:SEGment:DELete:ALL`
   
   This command clears all segment data stored in the segment memory (see “How to Free Segment and Sequence Memory” on page 113 for more information).

4. **Select the Reference Oscillator**
   
   `[SOURce:]ROSCillator:SOURce <source>`
   
   This command selects the reference oscillator source (see “Reference Oscillator Sources” on page 115).
5. **Set the Segment Sample Rate**
   
   `[SOURce:]FREQuency[1][:CW | :FIXed] <frequency>`
   
   This command sets the rate at which the points in a waveform segment are output by the AFG. The waveform frequency is determined by:
   
   \[
   \text{(sample frequency) / (number of points)}
   \]
   
   Refer to Table B-3 in Appendix B for the frequency limits.

6. **Select the Arbitrary Waveform Function**
   
   `[SOURce:]FUNCTION[:SHAPE] USER`
   
   This command selects the arbitrary waveform function. Couple the command to the previous frequency command.

7. **Set the Maximum Output Amplitude**
   
   `[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>`
   
   This command specifies the maximum output amplitude. The amplitude must be equal to, or greater than the maximum voltage value of the waveform segment. Refer to Table B-4 in Appendix B for the amplitude limits.

8. **Name the Waveform Segment**
   
   `[SOURce:]LIST[1][:SEGment]:SELect <name>`
   
   This command names the waveform segment. Each waveform segment to be stored into memory must have a unique name. Legal names must start with an alphabetic character, but can contain alphabetic, numeric, and underscore (“_”) characters. The names can have a maximum length of 12 characters.

9. **Set the Waveform Segment Size**
   
   `[SOURce:]LIST[1][:SEGment]:DEFine <length>`
   
   This command defines the size of the number of voltages or points in the selected waveform segment. The size must be equal or greater than the number of points in the waveform segment (minimum value is 8 points). The command reserves enough memory needed for the waveform segment.

10. **Store the Waveform Segment as Voltages**
    
    `[SOURce:]LIST[1][:SEGment]:VOLTage <voltage_list>`
    
    This command stores the points of the waveform segment into the AFG’s segment memory. These points are sent to the AFG as volts which are the output voltage points that constitutes the waveform segment.
11. **Name the Segment Sequence**
   
   ```
   [SOURce:]LIST[1]:SSEQUence:SELect <name>
   ```
   
   This command names the segment sequence. Each sequence stored into memory must have a unique name. Legal names must start with an alphabetic character, but can contain alphabetic, numeric, and underscore (“_”) characters. The names can have a maximum length of 12 characters. The names **MUST** be different from any waveform segment names stored in memory.

12. **Set the Segment Sequence Length**
   
   ```
   [SOURce:]LIST[1]:SSEQUence:DEFine <length>
   ```
   
   This command defines the length of the selected segment sequence. The length must be equal to, or greater than the number of waveform segments in the sequence (next step).

13. **Define the Segment Sequence Order**
   
   ```
   [SOURce:]LIST[1]:SSEQUence:SEQuence <segment_list>
   ```
   
   This command determines the order in which the waveform segments are to be executed. Each waveform segment name must be separated by a comma (for example, A,B,C). (see “Executing Several Waveform Segments” on page 93 for more information.)

14. **Select the User Name**
   
   ```
   [SOURce:]FUNCTION:USER <name>
   ```
   
   This command selects the segment sequences to be output. Make the `<name>` in this command the same name as the stored segment sequence to be output.

15. **Initiate the Waveform**
   
   ```
   INITiate[:IMMediate]
   ```
   
   This command generates an immediate output with the arm source set to IMMediate. Refer to Chapter 5 for triggering information.

16. **Query the Segment Memory (Optional)**
   
   ```
   [SOURce:]LIST[1]:SSEQuence:FREE?
   ```
   
   This command returns the amount of segment memory remaining (first number) in the AFG and the amount of memory used (second number).

17. **Query the Segment Sequence Memory (Optional)**
   
   ```
   [SOURce:]LIST[1]:SSEQuence:FREE?
   ```
   
   This command returns the amount of segment memory remaining in the AFG (the first number) and the amount of memory used (the second number).
HP BASIC Program Example (ARBWAVE)

1 !RE-STORE"ARBWAVE"
2 !This program demonstrates the procedure for developing and
3 !outputting an arbitrary waveform.
4 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg,Seg_mem$[256],Seq_mem$[256]
40 !
50 !Set up error checking
60 ON INTR 7 CALL Errmsg
70 ENABLE INTR 7;2
80 OUTPUT @Afg;"*CLS"
90 OUTPUT @Afg;"*SRE 32"
100 OUTPUT @Afg;"*ESE 60"
110 !
120 !Call the subprograms
130 CALL Rst
140 CALL Wf_del
150 !
160 OUTPUT @Afg;"SOUR:FREQ1:FIX 100E3;"; !frequency
170 OUTPUT @Afg;"SOUR:FUNC:SHAP USER;">; !function
180 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5.1V" !amplitude
190 !
200 CALL Ramp_wave
210 !
220 OUTPUT @Afg;"SOUR:FUNC:USER RAMP_OUT" !waveform sequence
230 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
240 !
250 PRINT "Segment memory points available, used: ";Seg_mem$
260 PRINT 
270 PRINT "Sequence memory points available, used: ";Seq_mem$
280 !
290 WAIT .1 !allow interrupt to be serviced
300 OFF INTR 7
310 END
320 !
330 SUB Ramp_wave
340 Ramp_wave: !Subprogram which defines a ramp waveform and output
350 !sequence.
360 COM @Afg,Seq_mem$,Seq_mem$
370 DIM Waveform(1:100) !Calculate waveform points
380 FOR l=1 TO 100
390 Waveform(l)=*0.0505
400 NEXT l
410 !
420 OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL RAMP" !segment name
430 OUTPUT @Afg;" SOUR:LIST1:SEGM:DEF 100" !segment size
440 OUTPUT @Afg;" SOUR:LIST1:SEGM:VOLT ";Waveform(*) !waveform points

Continued on Next Page
450  OUTPUT @Afg;" SOUR:LIST1:SEGM:FREE?"
460  ENTER @Afg:Seg_mem$
470  !
480  OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEL RAMP_OUT" !sequence name
490  OUTPUT @Afg;" SOUR:LIST1:SSEQ:DEF 1" !sequence size
500  OUTPUT @Afg;" SOUR:LIST1:SSEQ:SEQ RAMP" !segment order
510  OUTPUT @Afg;" SOUR:LIST1:SSEQ:FREE?"
520  ENTER @Afg:Seq_mem$
530  SUBEND
540  !
550  SUB Rst
560  Rst:  !Subprogram which resets the E1445.
570  COM @Afg,Seg_mem$,Seq_mem$
580  OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
590  ENTER @Afg:Complete
600  SUBEND
610  !
620  SUB Wf_del
630  Wf_del:  !Subprogram which deletes all sequences and segments.
640  COM @Afg,Seg_mem$,Seq_mem$
650  OUTPUT @Afg;"FUNC:USER NONE" !select no sequences
660  OUTPUT @Afg;"LIST:SSEQ:DEL:ALL" !Clear sequence memory
670  OUTPUT @Afg;"LIST:SEGM:DEL:ALL" !Clear segment memory
680  SUBEND
690  !
700  SUB Errmsg
710  Errmsg:  !Subprogram which displays E1445 programming errors
720  COM @Afg,Seg_mem$,Seq_mem$
730  DIM Message$[256]
740  !Read AFG status byte register and clear service request bit
750  B=SPOLL(@Afg)
760  !End of statement if error occurs among coupled commands
770  OUTPUT @Afg;"
780  OUTPUT @Afg;"ABORT" !abort output waveform
790  REPEAT
800  OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
810  ENTER @Afg;Code,Message$
820  PRINT Code,Message$
830  UNTIL Code=0
840  STOP
850  SUBEND

Visual BASIC and Visual C/C++ Program Versions
The Visual BASIC example program, ARBWAVE.FRM, is in directory “VBPROG” and the Visual C example program, ARBWAVE.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
The MULSEG program shows how to generate an arbitrary waveform using two different waveform segments. One waveform segment generates a 1 kHz, 5 V sine wave repeated twice. The other one generates a 1 kHz, 5 V peak triangle repeated once. The commands are:

1. **Reset the AFG**
   
   *RST
   
   The *RST command aborts waveform output and sets the AFG to a defined state.

2. **Clear the AFG Memory of All Sequence Data**
   
   [SOURce:LIST[1]:SSEQuence:DELete:ALL]
   
   This command clears all segment sequence data stored in the sequence memory.

3. **Clear the AFG Memory of All Segment Data**
   
   [SOURce:LIST[1]:DELete:ALL]
   
   This command clears all segment data stored in the segment memory.

4. **Set the Sample Rate**
   
   [SOURce:FREQuency[1]:CW | :FIXed] <frequency>
   
   This command sets the rate at which the points are output by the AFG. The frequency is:

   \[
   \text{frequency} = \frac{\text{sample frequency}}{\text{number of points}}
   \]

   Refer to Table B-3 in Appendix B for the frequency limits.
5. **Select the Arbitrary Waveform Function**
   
   ```
   [SOURce:]FUNCTION:[SHApe] USER
   ```
   This command selects the arbitrary waveform function. Couple the command to the previous frequency command.

6. **Set the Maximum Output Amplitude**
   
   ```
   [SOURce:]VOLTage[:LEVEL][:IMMediate][:AMPLitude] <amplitude>
   ```
   This command specifies the maximum output amplitude. The amplitude must be equal or greater than the maximum voltage value of the waveform segment. Refer to Table B-4 in Appendix B for the amplitude limits.

7. **Name the First Waveform Segment**
   
   ```
   [SOURce:]LIST[1]:SEGment:SELection <name>
   ```
   This command names the first waveform segment.

8. **Set the First Waveform Segment Size**
   
   ```
   [SOURce:]LIST[1]:SEGment:DEFine <length>
   ```
   This command defines the size of the selected waveform segment.

9. **Store the First Waveform Segment as Voltages**
   
   ```
   [SOURce:]LIST[1]:SEGment:VOLTage <voltage_list>
   ```
   This command stores the first waveform segment into the AFG’s segment memory.

10. **Name the Second Waveform Segment**
    
    ```
    [SOURce:]LIST[1]:SEGment:SELection <name>
    ```
    This command names the second waveform segment.

11. **Set the Second Waveform Segment Size**
    
    ```
    [SOURce:]LIST[1]:SEGment:DEFine <length>
    ```
    This command defines the size of the selected waveform segment.

12. **Store the Second Waveform Segment as Voltages**
    
    ```
    [SOURce:]LIST[1]:SEGment:VOLTage <voltage_list>
    ```
    This command stores the second waveform segment into the AFG’s segment memory.

13. **Name the Segment Sequence**
    
    ```
    [SOURce:]LIST[1]:SSEQuence:SELection <name>
    ```
    This command names the segment sequence. The name must be different from any segment names stored in memory.

14. **Set the Segment Sequence Length**
    
    ```
    [SOURce:]LIST[1]:SSEQuence:DEFine <length>
    ```
    This command defines the length of the selected segment sequence. The length must be equal or greater than the number of the waveform segments stored in memory.
15. **Define the Segment Sequence Order**
   
   [SOURce:]LIST[1]:SSEquence:SEQUence <segment_list>
   
   This command determines the order in which the waveform segments are to be executed. The names of each waveform segment to be output must be separated by a comma (for example, A,B,C). See “Sending Segment Sequences” on page 114 for more information.

16. **Define the Waveform Segments Repetition Count**
   
   [SOURce:]LIST[1]:SSEquence:DWELl:COUNT <repetition_list>
   
   This command sets how many times each waveform segment is to be executed. See “Sending Segment Sequences” on page 114 for more information.

17. **Select the User Name**
   
   [SOURce:]FUNCTION:USER <name>
   
   This command sets the AFG to output the selected segment sequence. The <name> in this command the same name as the stored segment sequence to be executed.

18. **Initiate the Waveform**
   
   INITiate[:IMMediate]
   
   This command generates an immediate output with the arm source set to IMMEDIATE. Refer to Chapter 5 for triggering information.
HP BASIC Program Example (MULSEG)

1 !RE-STORE"MULSEG"
2 !This program outputs an arbitrary waveform that is comprised of
3 !two waveform segments.
4 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !
50 !Set up error checking
60 ON INTR 7 CALL Ermsg
70 ENABLE INTR 7;2
80 OUTPUT @Afg;"CLS"
90 OUTPUT @Afg;"SRE 32"
100 OUTPUT @Afg;"ESE 60"
110 !
120 !Call the subprograms which reset the AFG and clear segment
130 !and sequence memory.
140 CALL Rst
150 CALL Wf_del
160 !Set the signal frequency, the function, and the amplitude.
170 OUTPUT @Afg;"SOUR:FREQ1:FIX 2.048E6;"
180 OUTPUT @Afg;"SOUR:FUNC:SHAP USER;"
190 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5.1V"
200 !Call the subprograms which define the triangle wave, sine wave,
210 !and output waveform sequence.
220 CALL Tri_wave
230 CALL Sine_wave
240 CALL Seq_def
250 !Select the output sequence and start the waveform.
260 OUTPUT @Afg;"SOUR:FUNC:USER WAVE_OUT"
270 OUTPUT @Afg;"INIT:IMM"
280 !
290 WAIT .1 !allow interrupt to be serviced
300 OFF INTR 7
310 END
320 !
330 SUB Tri_wave
340 Tri_wave: !Subprogram which defines a triangle waveform and stores
350 !it in a segment
360 COM @Afg
370 DIM Waveform(1:2048) !Calculate waveform points
380 FOR l=1 TO 2048
390 IF l<1024 THEN
400 Waveform(l)=l*.0048828
410 ELSE
420 Waveform(l)=(2048-l)*.0048828
430 END IF
440 NEXT I

Continued on Next Page
SUB Sine_wave
SUB Seq_def
SUB Rst
SUB Wf_del
SUB Errmsg

Chapter 3 Generating Arbitrary Waveforms 97
950 !End of statement if error occurs among coupled commands
960 OUTPUT @Afg:"
970 OUTPUT @Afg:"ABORT" !abort output waveform
980 REPEAT
990 OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
1000 ENTER @Afg;Code,Message$
1010 PRINT Code,Message$
1020 UNTIL Code=0
1030 STOP
1040 SUBEND

Visual BASIC and Visual C/C++ Program Versions
The Visual BASIC example program, MULSEG.FRM, is in directory “VBPROG” and the Visual C example program, MULSEG.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Using Different Frequency Generators

The AFG can use either the DDS (Frequency1) Frequency Generator or the Divide-by-N (Frequency2) Frequency Generator to generate arbitrary waveforms. The DDS generator gives lower frequency response with better resolution. The Divide-by-N generator gives higher frequency response with less resolution. For best operating practice, use the 42.9 MHz reference oscillator source (INTernal[1]) for the DDS generator; use the 40 MHz reference oscillator source (INTernal2) for the Divide-by-N generator. (See “Arbitrary Waveform Program Comments” on page 113 for more information.)

The AFGGEN1 program shows how to use the AFG’s DDS generator (selected at power-on) to generate waveforms. Use this generator for better frequency resolution and to perform frequency sweeping, frequency shifting, and so forth (see Chapter 4). See program AFGGEN2 for a frequency2 generator example. This program generates a 100 point ramp at 100 kHz. The commands are:

1. **Reset the AFG**
   
   *RST
   
   The *RST command aborts waveform output and sets the AFG to a defined state.

2. **Clear the AFG Memory of All Sequence Data**
   
   [SOURce:]LIST[1]:SSEQUence:DELeate:ALL
   
   This command clears all segment sequence data stored in the sequence memory.

3. **Clear the AFG Memory of All Segment Data**
   
   [SOURce:]LIST[1]:DELeate:ALL
   
   This command clears all segment data stored in the segment memory.
4. **Select the Reference Oscillator**
   
   `[SOURce:]ROSCillator:SOURce INTernal[1]`
   
   This command selects the 42.9 MHz (Internal1) reference oscillator source to be used with the DDS frequency generator (see “Reference Oscillator Sources” on page 115). (Although “RST selects the 42.9 MHz reference oscillator, it is selected here for good programming practice.)

   If you wish to use the Divide-by-N frequency generator, use:
   
   `[SOURce:]ROSCillator:SOURce INTernal2`

5. **Select the Sample Source**
   
   `TRIGger:STARt:SOURce INTernal[1]`
   
   This command selects the sample source for the DDS generator (that is, trigger start source INTernal[1]). (Although “RST selects this trigger start source, it is selected here for good programming practice.) The USER (that is, arbitrary waveform) function can use any of the trigger start sources (see “Sample Sources” on page 115).

   If you wish to use the Divide-by-N generator, use:
   
   `TRIGger:STARt:SOURce INTernal2`

6. **Set the Sample Frequency Range**
   
   `[SOURce:]FREQuency[1]:RANGe <range>`
   
   This command specifies the upper sample frequency limit (see “Frequency1 Generator Range” on page 116) for the DDS generator. Do not send this command if using the Divide-by-N generator.

7. **Set the Segment Sample Rate**
   
   `[SOURce:]FREQuency[1][:CW | :FIXed] <frequency>`
   
   This command sets the rate at which the points in a waveform segment are output by the AFG. The frequency is:

   \[
   \text{(sample frequency) / (number of points)}
   \]

   Refer to Table B-3 in Appendix B for the frequency limits.

8. **Select the Arbitrary Waveform Function**
   
   `[SOURce:]FUNCTION[:SHAPe] USER`
   
   This command selects the arbitrary waveform function. Couple the command to the previous frequency command.

9. **Set the Maximum Output Amplitude**
   
   `[SOURce:]VOLTage[:LEVel][:IMMediate]:AMPLitude <amplitude>`
   
   This command specifies the maximum output amplitude. The amplitude must be equal or greater than the maximum voltage value of the waveform segment. Refer to Table B-4 in Appendix B for the amplitude limits.
10. **Name the Waveform Segment**  
   \[\text{[SOURce:]LIST[1]:SEGMen}t\}:SELect \ <name>\]  
   This command names the waveform segment.

11. **Set the Waveform Segment Size**  
    \[\text{[SOURce:]LIST[1]:SEGMen}t\}:DEFine \ <length>\]  
    This command defines the size of the selected waveform segment.

12. **Store the Waveform Segment as Volts**  
    \[\text{[SOURce:]LIST[1]:SEGMen}t\}:VOLTage \ <voltage_list>\]  
    This command stores the segments into the AFG’s segment memory.

13. **Name the Segment Sequence**  
    \[\text{[SOURce:]LIST[1]:SEQuence:SELe}ct \ <name>\]  
    This command names the segment sequence.

14. **Set the Segment Sequence Length**  
    \[\text{[SOURce:]LIST[1]:SEQuence:DEF}ine \ <length>\]  
    This command defines the length of the selected segment sequence.

15. **Define the Segment Sequence Order**  
    \[\text{[SOURce:]LIST[1]:SEQuence:SEQu}ence \ <segment_list>\]  
    This command determines the order in which the waveform segments are to be executed.

16. **Select the User Name**  
    \[\text{[SOURce:]FUNCTION:USER} \ <name>\]  
    This command sets the AFG to output the selected segment sequence.

17. **Initiate the Waveform**  
    \[\text{INITiate[:IMMediate]}\]  
    This command generates an immediate output with the arm source set to IMMediate. Refer to Chapter 5 for triggering information.
HP BASIC Program Example (AFGGEN1)

1 !RE-STORE"AFGGEN1"
2 !This program outputs a ramp arbitrary waveform using the
3 !AFG's frequency1 generator.
4
5 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40
50 !Set up error checking
60 ON INTR 7 CALL Errmsg
70 ENABLE INTR 7;2
80 OUTPUT @Afg;"CLS"
90 OUTPUT @Afg;"SRE 32"
100 OUTPUT @Afg;"ESE 60"
110
120 !Call the subprograms which reset the AFG and which clear
130 !segment and sequence memory.
140 CALL Rst
150 CALL Wf_del
160 !Set waveform parameters
170 OUTPUT @Afg;"SOUR:ROSC:SOUR INT1;"
180 OUTPUT @Afg;"TRIG:STAR:SOUR INT1;"
190 OUTPUT @Afg;"SOUR:FREQ1:RANG 10E6;"
200 OUTPUT @Afg;"SOUR:FREQ1:FIX 10E6;"
210 OUTPUT @Afg;"SOUR:FUNC:SHAP USER;"
220 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5.1V" !set amplitude to 5.1V
230 !Call subprogram which defines waveform segment and sequence
240 CALL Ramp_wave
250 !Select output sequence and initiate waveform
260 OUTPUT @Afg;"SOUR:FUNC:USER RAMP_OUT"
270 OUTPUT @Afg;"INIT:IMM"
280
290 WAIT .1 !allow interrupt to be serviced
300 OFF INTR 7
310 END
320
330 SUB Ramp_wave
340 Ramp_wave: !Subprogram which defines a ramp waveform
350 COM @Afg
360 DIM Waveform(1:100) !Calculate waveform points
370 FOR I=1 TO 100
380 Waveform(I)=I*.0505
390 NEXT I
400
410 OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL RAMP" !Define segment name
420 OUTPUT @Afg;"SOUR:LIST1:SEGM:DEF 100" !Define segment size
430 OUTPUT @Afg;"SOUR:LIST1:SEGM:VOLT";Waveform(*) !load waveform points
440

Continued on Next Page
SUB Rst
Rst: !Subprogram which resets the E1445.

SUB Wf_del
Wf_del: !Subprogram which deletes all sequences and segments.

SUB Errmsg
Errmsg: !Subprogram which displays E1445 programming errors

Visual BASIC and Visual C/C++ Program Versions
The Visual BASIC example program, AFGGEN1.FRM, is in directory “VBPROG” and the Visual C example program, AFGGEN1.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
**HP BASIC Program Example (AFGGGEN2)**

This program is similar to the AFGGEN1 program on page 102 except it selects different reference oscillator and sample sources. The differences are as follows:

1 !RE-STORE"AFGGGEN2"
2 !This program outputs a ramp arbitrary waveform using the
3 !AFG's frequency2 generator.

160 !Set waveform parameters
170 OUTPUT @Afg;"SOUR:ROSC:SOUR INT2;";
180 OUTPUT @Afg;"TRIG:STAR:SOUR INT2;";
190 OUTPUT @Afg;"SOUR:FREQ2:RANG 40E6;";
200 OUTPUT @Afg;"SOUR:FREQ2:FIX 40E6;";

**Visual BASIC and Visual C/C++ Program Versions**

The Visual BASIC example program, AFGGEN2.FRM, is in directory “VBPROG” and the Visual C example program, AFGGEN2.C, is in directory “VCPROG” on the CD that came with your HP E1445A.

**Sample Programs**

The programs in this section generate various arbitrary waveforms. All programs output waveforms at a 1 kHz repetition frequency and 5 V amplitude. These programs do not delete any waveform segments and segment sequences stored in memory. Thus, once a program is executed, it generates Error +1100,"Illegal segment name" and Error +1110,"Illegal sequence name", if executed again.

Due to the similarity of all the programs, only the first program is completely presented here. Only the differences are shown by the other programs.
Generating a Sin(X)/X Waveform

The SIN_X program generates a Sin(X)/X waveform using 4096 segments or points.

HP BASIC Program Example (SIN_X)

1 !RE-STORE “SIN_X”
2 !This program generates the arbitrary waveform Sin(x)/x.
3 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !
50 !Set up error checking
60 ON INTR 7 CALL Errmsg
70 ENABLE INTR 7;2
80 OUTPUT @Afg;”*CLS”
90 OUTPUT @Afg;”*SRE 32”
100 OUTPUT @Afg;”*ESE 60”
110 !
120 !Call the subprogram which resets and clears the AFG.
130 CALL Rst
140 !Set the signal frequency, function, and amplitude.
150 OUTPUT @Afg;”SOUR:FREQ1:FIX 4.096E6;”
160 OUTPUT @Afg;”SOUR:FUNC:SHAP USER;”
170 OUTPUT @Afg;”SOUR:VOLT:LEV:IMM:AMPL 1.1V”
180 !Call the subprogram which defines the Sin(x)/x waveform and
190 !output sequence.
200 CALL Sinx_def
210 !Select the output sequence and start the waveform.
220 OUTPUT @Afg;”SOUR:FUNC:USER SIN_X_OUT”
230 OUTPUT @Afg;”INIT:IMM”
240 !
250 WAIT .1 !allow interrupt to be serviced
260 OFF INTR 7
270 END

Continued on Next Page
SUB Sinx_def
Sinx_def: !Define Sin(x)/x waveform and output sequence.

FOR I=-2047 TO 2048
    IF I=0 THEN I=1.E-38
    Waveform(I+2048)=((SIN(2*PI*.53125*I/256))/(.53125*I/256)*.159154943092)
NEXT I
!
OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL SIN_X" !select segment to be defined
OUTPUT @Afg;"SOUR:LIST1:SEGM:DEF 4096" !reserve memory for segment
OUTPUT @Afg;"SOUR:LIST1:SEGM:VOLT";Waveform(*) !load waveform points
!
OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEL SIN_X_OUT" !select sequence to be defined
OUTPUT @Afg;"SOUR:LIST1:SSEQ:DEF 1" !specify # segments in sequence
OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEQ SIN_X" !set segment order in sequence
SUBEND

SUB Rst
Rst: !Subprogram which resets the E1445.

OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
ENTER @Afg;Complete

SUBEND

SUB Errmsg
Errmsg: !Subprogram which displays E1445 programming errors

DIM Message$[256]
B=SPOLL(@Afg)
!End of statement if error occurs among coupled commands
OUTPUT @Afg;""
OUTPUT @Afg;"ABORT" !abort output waveform
REPEAT
    OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
    ENTER @Afg;Code,Message$
    PRINT Code,Message$
    UNTIL Code=0
STOP
SUBEND

Visual BASIC and Visual C/C++ Program Versions
The Visual BASIC example program, SIN_X.FRM, is in directory “VBPROG” and the Visual C example program, SIN_X.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Generating a Damped Sine Wave

The SIN_D program generates a Damped sine wave using 4096 segments or points.

---

HP BASIC Program Example (SIN_D)

This program is similar to the “SIN_X” BASIC program on page 105, with the following differences:

```
1 !RE-STORE "SIN_D"
2 !This program outputs a damped sine wave arbitrary waveform.

180 !Call the subprogram which defines a damped sine wave and
190 !the output sequence.
200 CALL Sind_def
210 !Select the output sequence and start the waveform.
220 OUTPUT @Afg;"SOUR:FUNC:USER SIN_D_OUT"
230 OUTPUT @Afg;"INIT:IMM"

300 SUB Sind_def
310 Sind_def: !Compute waveform (damped sine wave) and define segment.
320 COM @Afg
330 DIM Waveform(1:4096)
340 A=4/4096
350 FOR T=1 TO 4096
360 Waveform(T)=EXP(-A*T)*SIN(W*T)
370 NEXT T
380 OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL SIN_D" !select segment to be defined
390 OUTPUT @Afg;"SOUR:LIST1:SEGM:DEF 4096" !set segment size
400 OUTPUT @Afg;"SOUR:LIST1:SEGM:VOLT";Waveform(*) !load waveform points
410 !
420 OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEL SIN_D_OUT" !Define sequence name
430 OUTPUT @Afg;"SOUR:LIST1:SSEQ:DEF 1" !Define sequence size
440 OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEQ SIN_D" !Set segment execution order
450 SUBEND
```
The Visual BASIC example program, SIN_D.FRM, is in directory “VBPROG” and the Visual C example program, SIN_D.C, is in directory “VCProg” on the CD that came with your HP E1445A.

Generating an Exponential Charge/Discharge Waveform

The CHARGE program generates an Exponential Charge/Discharge waveform with 4096 segments or points.

HP BASIC Program Example (CHARGE)

This program is similar to the “SIN_X” BASIC program on page 105, with the following differences:

1  !RE-STORE "CHARGE"
2  !This program generates an exponential charge/discharge waveform
3  !as an arbitrary waveform.

180  !Call the subprogram which defines the exponential charge/
190  !discharge waveform and output sequence.
200  CALL Charge_def
210  !Select the output sequence and start the waveform.
220  OUTPUT @Afg;"SOUR:FUNC:USER CHARGE_OUT"
230  OUTPUT @Afg;"INIT:IMM"

280  SUB Charge_def
290  Charge_def: !Compute waveform (exponential) and define segment and
300  !sequence.
310  COM @Afg
320  DIM Waveform(1:4096)
330  Rc=400
340  FOR T=1 TO 4096
350  IF T>=0 AND T<2047 THEN
360      Waveform(T)=1*(1-EXP(-T/Rc))
370  END IF

Continued on Next Page
Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, CHARGE.FRM, is in directory “VBPROG” and the Visual C example program, CHARGE.C, is in directory “VCPROG” on the CD that came with your HP E1445A.

Generating a Sine Wave with Spikes

The SPIKES program generates a sine wave with spikes using 4096 segments or points.

HP BASIC Program Example (SPIKES)

This program is similar to the “SIN_X” BASIC program on page 105, with the following differences:

1  !RE-STORE “SPIKES”
2  !This program generates a spiked sine wave as an arbitrary waveform.
180  !Call the subprogram which defines a sine wave with a spike and
190  !the output sequence.
200  CALL Spike_def
210  !Select the output sequence and start the waveform.
220  OUTPUT @Afg;“SOUR:FUNC:USER SPIKES_OUT”
230  OUTPUT @Afg;“INIT:IMM”
290  SUB Spike_def

Continued on Next Page
300  Spike_def:  !Compute waveform (sine wave with spike) and define segment.
310    COM @Afg
320    DIM Waveform(1:4096)
330    FOR I=1 TO 4096
340        Waveform(I)=SIN(2*PI*(I/4096))
350    NEXT I
360    Width=50
370    FOR J=1 TO Width/2
380        Waveform(J+1024)=Waveform(J+1024)+J*.04
390    NEXT J
400    FOR J=1 TO Width/2
410        Waveform(J+1024+Width/2)=Waveform(J+1024+Width/2)+1-(J*.04)
420    NEXT J
430    OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL SPIKES"  !select segment to be defined
440    OUTPUT @Afg;"SOUR:LIST1:SEGM:DEF 4096"  !reserve memory for segment
450    OUTPUT @Afg;"SOUR:LIST1:SEGM:VOLT";Waveform(*)  !load waveform points
460     !
470    OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEL SPIKES_OUT"  !Define sequence name
480    OUTPUT @Afg;"SOUR:LIST1:SSEQ:DEF 1"  !Define sequence size
490    OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEQ SPIKES"  !Define segment execution order
500  SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, SPIKES.FRM, is in directory “VBPROG” and the Visual C example program, SPIKES.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Generating a \( \frac{1}{2} \) Rectified Sine Wave

The SIN_R program generates a Rectified sine wave using 4096 segments or points.

---

HP BASIC Program Example (SIN_R)

This program is similar to the “SIN_X” BASIC program on page 105, with the following differences:

1. !RE-STORE “SIN_R”
2. !This program outputs a rectified sine wave as an arbitrary waveform.

180 !Call the subprogram which defines a rectified sine wave and
190 !the output sequence.
200 CALL Sinr_def
210 !Select the output sequence and start the waveform.
220 OUTPUT @Afg;“SOUR:FUNC:USER SIN_R_OUT”
230 OUTPUT @Afg;“INIT:IMM”

280 SUB Sinr_def
290 Sinr_def: !Compute waveform (rectified sine wave) and define segment.
300 COM @Afg
310 DIM Waveform(1:4096)
320 FOR I=1 TO 4096
330 Waveform(I)=SIN(2*PI*(I/4096))
340 NEXT I
350 FOR I=2048 TO 4096
360 Waveform(I)=0
370 NEXT I
380 OUTPUT @Afg;“SOUR:LIST1:SEGM:SEL SIN_R” !Define segment name
390 OUTPUT @Afg;“SOUR:LIST1:SEGM:DEF 4096” !Define segment size
400 OUTPUT @Afg;“SOUR:LIST1:SEGM:VOLT”;Waveform(*) !load waveform points
410 !
420 OUTPUT @Afg;“SOUR:LIST1:SSEQ:SEL SIN_R_OUT” !Define sequence name
430 OUTPUT @Afg;“SOUR:LIST1:SSEQ:DEF 1” !Define sequence size
440 OUTPUT @Afg;“SOUR:LIST1:SSEQ:SEQ SIN_R” !Set segment execution order
450 SUBEND
Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, SIN_R.FRM, is in directory “VBPROG” and the Visual C example program, SIN_R.C, is in directory “VCPROG” on the CD that came with your HP E1445A.

Generating Noise

The NOISE program generates pseudo-noise.

HP BASIC Program Example (NOISE)

This program is similar to the “SIN_X” BASIC program on page 105, with the following differences:

1  !RE-STORE “NOISE”
2  !This program outputs a pseudo random noise waveform as an
3  !arbitrary waveform.

180  !Call the subprogram which defines the noise signal and
190  !output sequence.
200  CALL Noise_def
210  !Select the output sequence and start the waveform.
220  OUTPUT @Afg;“SOUR:FUNC:USER NOISE_OUT”
230  OUTPUT @Afg;“INIT:IMM”

280  SUB Noise_def
290  Noise_def:  !Subprogram which defines the noise signal and output
300     !sequence.
310  COM @Afg
320  DIM Waveform(1:4096)
330  FOR I=1 TO 4096
340     Waveform(I)=2.*RND-1
350  NEXT I
360  OUTPUT @Afg;“LIST:SEG:M:SEL NOISE”       !select segment to be defined
370  OUTPUT @Afg;“ LIST:SEG:M:DEF 4096”        !reserve memory for segment
380  OUTPUT @Afg;“ LIST:SEG:M:VOLT*:Waveform(*)” !load waveform points
390  !
400  OUTPUT @Afg;“SOUR:LIST1:SSEQ:SEL NOISE_OUT”   !define sequence
410  OUTPUT @Afg;“SOUR:LIST1:SSEQ:DEF 1”    !number segments in sequence
420  OUTPUT @Afg;“SOUR:LIST1:SSEQ:SEQ NOISE”   !segment order
430  SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, NOISE.FRM, is in directory “VBPROG” and the Visual C example program, NOISE.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Arbitrary Waveform Program Comments

The following comments give additional details on the program examples in this chapter.

Determining the Amount of Segment and Sequence Memory

• To determine the amount of segment sequence data remaining in the AFG and the amount of data used, execute the [SOURce:]LIST[1]:SSEQUence:FREE? command. The command returns two number values. The first number shows, in points, the amount of segment sequence memory available. The second number shows, in points, the amount of segment sequence memory used.

• To determine the amount of segment data remaining in the AFG and the amount of segment data used, execute the [SOURce:]LIST[1]:SEGMent:FREE? command. The command returns two number values. The first number shows, in points, the amount of segment memory available. The second number shows, in points, the amount of segment memory used.

How to Free Segment and Sequence Memory

• Use [SOURce:]LIST[1]:SSEQUence:DELeTe[:SELected] to delete the currently selected segment sequence data that was last selected by the [SOURce:]LIST[1]:SSEQUence:SELect command.

• [SOURce:]LIST[1]:SSEQUence:DELeTe:ALL deletes all segment sequence data stored in the AFG’s sequence memory. Use the command if there is insufficient segment sequence memory available to store new segment sequences. Note that a segment sequence cannot be deleted if it is currently selected by the [SOURce:]FUNCTION:USER <name> command.

• Use [SOURce:]LIST[1]:SEGMent:DELeTe[:SELected] to delete the currently selected segment data that was last selected by the [SOURce:]LIST[1]:SEGMent:SELect command.

• [SOURce:]LIST[1]:SEGMent:DELeTe:ALL deletes all segment data currently in the segment memory. Use the command if there is insufficient segment memory available to store new segments.

Amplitude Effects on Voltage Lists

If the segment data is sent as voltage values, the AFG changes the data into digital-to-analog converter (DAC) codes. This requires that the voltage value of the segment data MUST NOT exceed the AFG’s current amplitude level (set by [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude]). If it does, the AFG generates an error.
Using DAC Codes to Send Segment Data

Besides sending the points in a waveform segment as voltage data, they can also be sent as signed or unsigned DAC codes data. Since the AFG always stores DAC codes into memory, setting the amplitude levels is not necessary if sending segment data as DAC codes instead of voltages. See Chapter 7 on how to store DAC codes.

Sending Segment Sequences

- \[\text{SOURCE:LIST}1:SEQ:SEQ \text{segment_list}\] selects the sequence in which the waveform segments are to be executed. The waveform segments must be in memory, or the AFG generates an error. Each waveform segment name to be executed must be separated by a comma (“,”). For example, to execute the “sine” and “tri” waveform segments, send the command as:
  \[\text{SOUR:LIST}1:SEQ:SEQ \text{sine,tri}\]

- A waveform segment can be executed more than once in a single segment sequence. There are two different methods. In one method, a waveform segment is placed in the \[\text{SOUR:LIST}1:SEQ:SEQ \text{segment_list}\] command several times. The other method uses an additional command, the \[\text{SOUR:LIST}1:SEQ:DWELL:COUNT \text{repetition_list}\] command. The following examples illustrates the two methods. The examples show how to execute waveform segment “sine” three times and segment list “tri” once.

  **Method 1:**
  \[\text{SOUR:LIST}1:SEQ:SEQ \text{sine,sine,sine,tri}\]

  **Method 2:**
  \[\text{SOUR:LIST}1:SEQ:SEQ \text{sine, tri} \text{SOUR:LIST}1:SEQ:DWELL:COUNT 3,1\]

Method 1 requires more memory since the minimum segment sequence length (\[\text{SOUR:LIST}1:SEQ:DEF \text{length}\]) must be at least the number of waveform segments in the sequence (i.e., sine,sine,sine,tri = a length of 4). Since in Method 2 the waveform segments consist of “sine,tri”, the sequence length is only 2.

Method 1 is required if the marker outputs (set by the \[\text{SOUR:LIST}1:SEQ:MARK\] command) are to be different for the various repetitions (see Chapters 6 and 7 for marker output information).
**Reference Oscillator Sources**

- The USER function can use any of the reference oscillator sources selected by the \([\text{SOURce:}]\text{ROSCillator:SOURce}\) command. The reference oscillator sources are:
  
  - \text{INTernal}^1 – 42.94967296 MHz (power-on value)
  - \text{INTernal}^2 – 40 MHz
  - \text{CLK}^10 – 10 MHz (the VXIbus CLK line)
  - \text{EXTernal} – User provided value (the front panel “Ref/Smpl In” BNC)
  - \text{ECLTrg}^0 \text{ or } 1 – User provided value (the VXIbus ECL trigger lines)

- If using either the \text{EXTernal} or \text{ECLTrg}^0 \text{ or } 1 reference oscillator sources, enter the source frequency to the AFG using \([\text{SOURce:}]\text{ROSCillator:FREQuency:EXTernal}\ <\text{frequency}>\).

- For best frequency linearity, use the 42.9 MHz (i.e., \text{INTernal}^1) reference oscillator source with the DDS (frequency1) frequency generator. This combination provides .01 Hz resolution. For higher frequency values, use the 40 MHz (i.e., \text{INTernal}^2) reference oscillator source with the Divide-by-N (frequency2) frequency generator. Use the \text{EXTernal} or \text{ECLTrg}^0 \text{ or } \text{ECLTrg}^1 sources for custom frequency values. However, any reference oscillator source can be used with any frequency generator.

**Sample Sources**

- The USER function operates with any of the sample sources selected by the \(\text{TRIGger:STARt:SOURce}\) command. The \text{INTernal}^1 source automatically selects the DDS frequency generator. The \text{INTernal}^2 source selects the Divide_by_N frequency generator. The other sources are not used with any frequency generator. The sample frequency thus depends on the externally generated sample signal. The different sample sources are:
  
  - \text{INTernal}^1 (power-on value; selects the DDS frequency generator)
  - \text{INTernal}^2 (selects the Divide-by-N frequency generator)
  - \text{BUS} (the HP-IB GET or \*TRG commands)
  - \text{EXTernal} (the front panel “Ref/Smpl In” BNC)
  - \text{ECLTrg}^0 \text{ or } \text{ECLTrg}^1 (the VXIbus ECL trigger lines)
  - \text{HOLD} (suspends sample generation)
  - \text{TTLTrg}^0 \text{ through } 7 \text{ (the VXIbus TTL trigger lines)}
The [SOURce:]FREQuency[1]:RANGe command allows for higher sample frequency operations of the USER function. This command is only used with frequency1 generator. If set to 0 (MINimum), the normal setting, the maximum sample frequency is the Reference Oscillator frequency / 4.

If set to MAXimum, the maximum sample frequency is the Reference Oscillator frequency / 2.

The MAXimum setting worsens the frequency resolution by a factor of two and introduces some sample rate jitter.

Returning the Waveform Segment Names

Use [SOURce:]LIST[1]:SEGMent:CATalog? to return the names of the different waveform segments stored in memory. The command returns comma-separated strings that contain the names of the segment lists.

Determining the Waveform Segment Size

Use [SOURce:]LIST[1]:SEGMent:VOLTage:POINts? to determine the size, in number of waveform segments or points, of the currently selected waveform segment.

Returning the Segment Sequence List Names

Use [SOURce:]LIST[1]:SSEQuence:CATalog? to return the names of the different segment sequence lists stored in memory. The command returns comma-separated strings that contain the names of the segment sequence lists.

Returning the Repetition Count List Length

Use [SOURce:]LIST[1]:SSEQuence:DWELl:COUNt:POINts? to determine the length of the currently selected segment sequence’s repetition count list.
Chapter 4

Chapter Contents

This chapter covers the sweeping, frequency list, and frequency-shift keying (FSK) features of the HP E1445A 13-Bit Arbitrary Function Generator (called the “AFG”). The chapter is organized as follows:

- FSK Programming Flowchart .................................. Page 118
  - FSK Command Reference .................................. Page 120

- Sweeping and Frequency Lists ................................. Page 120
  - Sweeping Using Start and Stop Frequencies .......... Page 121
  - Specifying a Frequency List ............................... Page 124
  - Sweeping Using Start and Span Frequencies ......... Page 127
  - Frequency Lists Using Definite and Indefinite Length
    Arbitrary Blocks ........................................ Page 130
  - Logarithmic Sweeping ..................................... Page 133
  - Sweep Points Versus Time ................................ Page 135
  - Frequency Lists Versus Time ............................. Page 138
  - Sweeping Arbitrary Waveforms ............................ Page 141
  - AC Output Leveling ....................................... Page 144

- Frequency-Shift Keying ...................................... Page 147
  - FSK Using the “FSK” Control Source .................. Page 147
  - FSK Using the TTLTrg<n> Control Source ............ Page 150
  - FSK Using an Arbitrary Waveform ...................... Page 152

- Sweeping and FSK Program Comments ..................... Page 154
  - Reference Oscillator Sources ............................ Page 154
  - Sample Sources ........................................... Page 154
  - AFG Frequency Modes ..................................... Page 155
  - Frequency Range: Sweeping and Sampling .......... Page 155
  - Frequency Range: Frequency Lists and FSK ........ Page 155
  - Sweep Count and Frequency List Repetition Count ........................................ Page 156
  - Arbitrary Block Data ...................................... Page 156
  - Frequency Points .......................................... Page 157
  - Sweep Spacing ............................................. Page 157
  - Sweep Direction ........................................... Page 157
  - Sweep Time ................................................ Page 158
  - Output Frequency and Sample Rate .................... Page 160
  - AC Leveling ............................................... Page 160
  - FSK Control Sources ....................................... Page 161
  - Frequency-Shift Delay ..................................... Page 162
  - Driving the TTLTrg<n> Trigger Lines ................. Page 162
FSK Programming Flowchart

The flowchart in Figure 4-1 shows the commands used to program the AFG for frequency sweeps, frequency lists, and for frequency-shift keying. Shown with each command is its power-on/reset setting. Since each example program begins by resetting the AFG, many of the default settings are used. Thus, the examples, which follow the sequence, do not execute every command shown on the flowchart. Remove the flowchart from the binder for easy accessibility. Refer to the flowchart while doing the examples in this chapter, if desired.

Figure 4-1. Commands for Frequency Sweeps, Frequency Lists, and FSK Keying (continued on next page)
Figure 4-1. Commands for Frequency Sweeps, Frequency Lists, and FSK Keying
(continued from previous page)
FSK Command Reference

Detailed information on the commands introduced in this chapter can be found in Chapter 8, “Command Reference”. The commands in this chapter are shown in their entirety (optional headers included) to help you locate them in the reference.

Sweeping and Frequency Lists

The AFG offers linear frequency sweeping of standard waveforms (that is, sine, square, triangle, ramp) and arbitrary waveforms from 0.0 Hz to 10.73741824 MHz, and logarithmic sweeping of standard and arbitrary waveforms from 0.01 Hz to 10.73741824 MHz.

The AFG can also “frequency hop” — where the AFG outputs a sequence of discrete frequencies from a pre-defined list. Up to 256 frequencies from 0.0 Hz to 10.73741824 MHz can be specified in a single list, and the AFG can sequence through the list at up to 800 frequencies per second.

Sweeps and frequency lists are programmed with the same commands. The command subsystems covered in this section include:

- [SOURce:]ROSCillator
- TRIGger
- [SOURce:]FREQuency[1]
  - Sweep mode and related commands
  - Frequency list mode and related commands
- [SOURce:]SWEep
- ARM:SWEep

The following programs show how to perform sweeps and frequency lists.
Sweeping Using Start and Stop Frequencies

The SMPLSWP1 program specifies a start frequency and a stop frequency and continuously sweeps between 0 and 1 MHz. The program also queries the start frequency, stop frequency, center frequency, and frequency span to show the relationship between them.

Using the flowchart in Figure 4-1 as a guide, the steps of this program are:

1. **Select the 42.9 MHz reference oscillator**
   
   \[\text{SOURce:ROS} \text{Cillator:SOURce } <\text{source}>\]

2. **Select the frequency generator that allows frequency sweeping**
   
   \[\text{TRIGger[:STARt]:SOURce } <\text{source}>\]

3. **Select the frequency sweep mode**
   
   \[\text{SOURce:Frequency[1]:MODE } <\text{mode}>\]

4. **Set the start frequency**
   
   \[\text{SOURce:Frequency[1]:STARt } <\text{start_freq}>\]

5. **Set the stop frequency**
   
   \[\text{SOURce:Frequency[1]:STOP } <\text{stop_freq}>\]

6. **Set the number of sweeps**
   
   \[\text{SOURce:SWEep:COUNT } <\text{number}>\]

7. **Set the output function**
   
   \[\text{SOURce:FUNCTION[:SHAPE] } <\text{shape}>\]

8. **Set the signal amplitude**
   
   \[\text{SOURce:VOLTage[:LEVEL][:IMMediate][:AMPLitude] } <\text{amplitude}>\]

9. **Place the AFG in the wait-for-arm state**
   
   \[\text{INITi}ate[:IMMediate] \]
HP BASIC Program Example (SMPLSWP1)

1 !RE-STORE"SMPLSWP1"
2 !This program specifies start and stop frequencies to sweep
3  a sine wave from 0 to 1 MHz.
4 !
5 !Assign I/O path between the computer and E1445A.
6 ASSIGN @Afg TO 70910
7 COM @Afg
8 !
9 !Set up error checking
10 ON INTR 7 CALL Ermsg
11 ENABLE INTR 7;2
12 OUTPUT @Afg;"*CLS"
13 OUTPUT @Afg;"*SRE 32"
14 OUTPUT @Afg;"*ESE 60"
15 !
16 !Call the subprograms
17 CALL Rst
18 CALL Sweep1
19 CALL Query
20 !
21 WAIT .1 !allow interrupt to be serviced
22 OFF INTR 7
23 END
24 !
25 SUB Sweep1
26 Sweep1: !Subprogram which outputs a swept sine wave from 0 Hz to 1 MHz
27 COM @Afg
28 OUTPUT @Afg;"SOUR:ROSC:SOUR INT1;"; !reference oscillator
29 OUTPUT @Afg;":TRIG:STAR:SOUR INT1;"; !frequency1 generator (DDS)
30 OUTPUT @Afg;":SOUR:FREQ1:MODE SWE;"; !sweep mode
31 OUTPUT @Afg;":SOUR:FREQ1:STAR 0;"; !start frequency
32 OUTPUT @Afg;":SOUR:FREQ1:STOP 1E6;"; !stop frequency
33 OUTPUT @Afg;":SOUR:SWE:COUN INF;"; !sweep count
34 OUTPUT @Afg;":SOUR:FUNC:SHAP SIN;"; !function
35 OUTPUT @Afg;":SOUR:VOLT:LEV:IMM:AMPL 5 V"!amplitude
36 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
37 SUBEND
38 !
39 SUB Rst
40 Rst: !Subprogram which resets the E1445.
41 COM @Afg
42 OUTPUT @Afg;"*RST:"OPC?" !reset the AFG
43 ENTER @Afg;Complete
44 SUBEND
45 !
46 SUB Query
47 Query: !Subprogram which queries sweep parameters
48 COM @Afg

Continued on Next Page
The start, stop, center, and span values returned are:

\[
\begin{align*}
\text{START} &= +0.000000000\times10^0 \\
\text{STOP} &= +1.000000000\times10^6 \\
\text{CENTER} &= +5.000000000\times10^5 \\
\text{SPAN} &= +1.000000000\times10^6
\end{align*}
\]

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, SMPLSWP1.FRM, is in directory “VBPROG” and the Visual C example program, SMPLSWP1.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Specifying a Frequency List

The LIST1 program shows the basic steps involved in setting up and “hopping” through a frequency list. The program also shows how to query the number of frequencies in the list.

Using the flowchart in Figure 4-1 as a guide, the steps of this program are:

1. Select the 42.9 MHz reference oscillator
   [SOURce:]ROSCillator:SOURce <source>

2. Select the frequency generator which allows frequency lists (hopping)
   TRIGger[:STARt]:SOURce <source>

3. Select the frequency list mode
   [SOURce:]FREQuency[1]:MODE <mode>

4. Specify the frequency list
   [SOURce:]LIST2:FREQuency <freq_list>

5. Set the output function
   [SOURce:]FUNCtion[:SHAPE] <shape>

6. Set the signal amplitude
   [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>

7. Place the AFG in the wait-for-arm state
   INITiate[:IMMediate]
HP BASIC Program Example (LIST1)

1 !RE-STORE "LIST1"
2 !The following program outputs the frequencies 1 kHz, 10 kHz, 100 kHz, and 1 MHz in a (default) period of 1 second.
3 !
4 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !
50 !Set up error checking
60 ON INTR 7 CALL Ermsg
70 ENABLE INTR 7;2
80 OUTPUT @Afg;"*CLS"
90 OUTPUT @Afg;"*SRE 32"
100 OUTPUT @Afg;"*ESE 60"
110 !
120 !Call the subprograms
130 CALL Rst
140 CALL List1
150 CALL List_length
160 !
170 WAIT .1 !allow interrupt to be serviced
180 OFF INTR 7
190 END
200 !
210 SUB List1
220 List1: !Subprogram which outputs a frequency list
230 COM @Afg
240 OUTPUT @Afg;"SOUR:ROSC:SOUR INT1;"; !reference oscillator
250 OUTPUT @Afg;"TRIG:STAR:SOUR INT1;"; !frequency1 generator
260 OUTPUT @Afg;"SOUR:FREQ1:MODE LIST;"; !list mode
270 OUTPUT @Afg;"SOUR:LIST2:FREQ 1E3,10E3,100E3,1E6;"; !freq list
280 OUTPUT @Afg;"SOUR:FUNC:SHAP SIN;"; !function
290 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5 V" !amplitude
300 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
310 SUBEND
320 !
330 SUB Rst
340 Rst: !Subprogram which resets the E1445.
350 COM @Afg
360 OUTPUT @Afg;"RST;*OPC?" !reset the AFG
370 ENTER @Afg;Complete
380 SUBEND
390 !
400 SUB List_length
410 List_length: !Subprogram which queries frequency list length
420 COM @Afg
430 OUTPUT @Afg;"SOUR:LIST2:FREQ:POIN?"

Continued on Next Page
Upon completion, the program displays:

"Number of frequencies in list: +4"

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, LIST1.FRM, is in directory “VBPROG” and the Visual C example program, LIST1.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
The SMPLSWP2 program specifies a start frequency and a frequency span to continuously sweep from 1 kHz to 21 kHz. The program also queries the start frequency, stop frequency, center frequency, and frequency span to show the relationship between them.

Using the flowchart in Figure 4-1 as a guide, the steps of this program are:

1. **Set the frequency sweep mode**
   
   [SOURce:]FREQuency[1]:MODE <mode>

2. **Set the start frequency**
   
   [SOURce:]FREQuency[1]:STARt <start_freq>

3. **Set the frequency span**
   
   [SOURce:]FREQuency[1]:SPAN <freq_span>

4. **Set the number of sweeps**
   
   [SOURce:]SWEep:COUNt <number>

5. **Set the output function**
   
   [SOURce:]FUNCtion[:SHAPe] <shape>

6. **Set the signal amplitude**
   
   [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>

7. **Place the AFG in the wait-for-arm state**
   
   INITiate[:IMMediate]
HP BASIC Program Example (SMPLSWP2)

1 !RE-STORE"SMPLSWP2"
2 !This program continuously sweeps from 1 kHz to 21 kHz and specifies
3 !a start frequency and a frequency span.
4 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !
50 !Set up error checking
60 ON INTR 7 CALL Ermsg
70 ENABLE INTR 7:2
80 OUTPUT @Afg;"*CLS"
90 OUTPUT @Afg;"*SRE 32"
100 OUTPUT @Afg;"*ESE 60"
110 !
120 !Call the subprograms
130 CALL Rst
140 CALL Sweep2
150 CALL Query
160 !
170 WAIT .1 !allow interrupt to be serviced
180 OFF INTR 7
190 END
200 !
210 SUB Sweep2
220 Sweep2: !Subprogram which outputs a swept sine wave from 1 kHz to
230 !21 kHz.
240 !
250 OUTPUT @Afg;"SOUR:FREQ1:MODE SWE;"; !sweep mode
260 OUTPUT @Afg;"SOUR:FREQ1:STAR 1E3;"; !start frequency
270 OUTPUT @Afg;"SOUR:FREQ1:SPAN 20E3;"; !frequency span
280 OUTPUT @Afg;"SOUR:SWE:COUN INF;"; !sweep count
290 OUTPUT @Afg;"SOUR:FUNC:SHAP SIN;"; !function
300 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5 V" !amplitude
310 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
320 SUBEND
330 !
340 SUB Rst
350 Rst: !Subprogram which resets the E1445.
360 COM @Afg
370 OUTPUT @Afg;"*RST;OPC?" !reset the AFG
380 ENTER @Afg:Complete
390 SUBEND
400 !
410 SUB Query
420 Query: !Subprogram which queries sweep parameters
430 COM @Afg
440 OUTPUT @Afg;"SOUR:FREQ1:CENT?"

Continued on Next Page
The start, stop, center, and span values returned are:

\[
\begin{align*}
\text{START} & = +1.000000000 \times 10^3 \\
\text{CENTER} & = +1.100000000 \times 10^4 \\
\text{STOP} & = +2.100000000 \times 10^4 \\
\text{SPAN} & = +2.000000000 \times 10^4
\end{align*}
\]

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, SMPLSWP2.FRM, is in directory “VBPROG” and the Visual C example program, SMPLSWP2.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
**Frequency Lists Using Definite and Indefinite Length Arbitrary Blocks**

When specifying a large frequency list (up to 256 frequencies), the ease in which the list is specified and the speed at which data is loaded into the AFG is enhanced by using definite or indefinite length arbitrary blocks. The data in an arbitrary block is in IEEE-754 64-bit floating point format.

The LISTDEF program sends a definite length arbitrary block of 100 frequencies to the AFG. Once the frequencies are received, the AFG steps through the list at one frequency per second.

At the end of the listing are program modifications for sending the data in an indefinite length arbitrary block.

Using the flowchart in Figure 4-1 as a guide, the steps of this program are:

1. **Select the reference oscillator**
   
   [SOURce:]ROSCillator:SOURce <source>

2. **Select the frequency generator which allows frequency lists (hopping)**
   
   TRIGger[:STARt]:SOURce <source>

3. **Select the frequency list mode**
   
   [SOURce:]FREQuency[1]:MODE <mode>

4. **Download the frequency list**
   
   [SOURce:]LIST2:FREQuency <freq_list>

5. **Set the duration of the list**
   
   [SOURce:]SWEep:TIME <number>

6. **Set the output function**
   
   [SOURce:]FUNCtion[:SHAPe] <shape>

7. **Set the signal amplitude**
   
   [SOURce:]VOLTage[:LEVEL][:IMMediate][:AMPLitude] <amplitude>

8. **Place the AFG in the wait-for-arm state**

   INITiate[:IMMediate]
HP BASIC Program Example (LISTDEF)

1 !RE-STORE"LISTDEF"
2 !This program sends a definite length arbitrary block of frequencies
3 !to the AFG. Once the AFG receives the frequencies, it steps through
4 !the list at a rate of one frequency per second.
5 !
6 !Assign I/O paths between the computer and E1445A. One path sends
7 !data in ASCII format to the AFG, the other path sends frequency
8 !list data to the AFG in binary format.
9
10 COM @Afg,@Afg1
11 ASSIGN @Afg TO 70910 !path for ASCII data
12 ASSIGN @Afg1 TO 70910;FORMAT OFF !path for binary data
13 !
14 !Set up error checking
15 ON INTR 7 CALL Errmsg
16 ENABLE INTR 7;2
17 OUTPUT @Afg;"*CLS"
18 OUTPUT @Afg;"*SRE 32"
19 OUTPUT @Afg;"*ESE 60"
20 !
21 !Call the subprograms
22 CALL Rst
23 CALL List1
24 !
25 WAIT .1 !allow interrupt to be serviced
26 OFF INTR 7
27 END
28 !
29 SUB List1
30 !Subprogram which downloads a list of 100 frequencies
31 !(1 kHz to 100 kHz) in a definite length arbitrary block.
32 COM @Afg,@Afg1
33 DIM Freqlist(1:100)
34 FOR I=1 TO 100
35 Freqlist(I)=1000.*I
36 NEXT I
37 !
38 OUTPUT @Afg;"SOUR:ROSC:SOUR INT1;"; !reference oscillator
39 OUTPUT @Afg;"::TRIG:STAR:SOUR INT1;"; !frequency1 generator
40 OUTPUT @Afg;"::SOUR:FREQ1:MODE LIST;"; !frequency list mode
41 OUTPUT @Afg USING ",,K";::SOUR:LIST2:FREQ #3800"download freqs
42 OUTPUT @Afg1;Freqlist(*) !^ 800 bytes - 3 digits
43 OUTPUT @Afg !CR LF
44 OUTPUT @Afg;"SOUR:SWE:TIME 100;"; !time (seconds) through list
45 OUTPUT @Afg;"::SOUR:FUNC:SHAP SIN;"; !function
46 OUTPUT @Afg;"::SOUR:VOLT:LEV:IMM:AMPL 5 V" !amplitude
47 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
48 SUBEND
49 !
50 !Continued on Next Page
SUB Rst
Rst: !Subprogram which resets the E1445.
COM @Afg,Afg1
OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
ENTER @Afg;Complete
SUBEND
!
SUB Errmsg
Errmsg: !Subprogram which displays E1445 programming errors
COM @Afg,Afg1
DIM Message$(256)
!Read AFG status byte register and clear service request bit
B=SPOLL(@Afg)
!End of statement if error occurs among coupled commands
OUTPUT @Afg;"
OUTPUT @Afg;"ABORT" !abort output waveform
REPEAT
OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
ENTER @Afg;Code,Message$
PRINT Code,Message$
UNTIL Code=0
STOP
SUBEND

Program Modifications

In order to download the frequency list as an indefinite length arbitrary block, modify lines 350 through 370 as follows:

OUTPUT @Afg USING ",K";":SOUR:LIST2:FREQ #0" !download freqs
OUTPUT @Afg1;Freqlist(*)
OUTPUT @Afg;CHR$(10);END !LF EOI (NL END)

Additional information on definite and indefinite length arbitrary blocks is located under “Arbitrary Block Data” on page 156.

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, LISTDEF.FRM, is in directory “VBPROG” and the Visual C example program, LISTDEF.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Logarithmic Sweeping

The LOG_SWP program shows you how to select logarithmic spacing between the frequencies in a frequency sweep. The program sets up a seven point logarithmic frequency sweep from 1 Hz to 1 MHz. Thus, the swept frequencies are: 1 Hz, 10 Hz, 100 Hz, 1 kHz, 10 kHz, 100 kHz, 1 MHz.

Using the flowchart in Figure 4-1 as a guide, the steps of this program are:

1. Set the frequency sweep mode
   [SOURce:]FREQuency[1]:MODE <mode>

2. Set the start frequency
   [SOURce:]FREQuency[1]:STARt <start_freq>

3. Set the stop frequency
   [SOURce:]FREQuency[1]:STOP <stop_freq>

4. Set the number of points (frequencies) in the frequency sweep
   [SOURce:]SWEep:POINts <number>

5. Set linear or logarithmic spacing
   [SOURce:]SWEep:SPACing <mode>

6. Set the number of sweeps
   [SOURce:]SWEep:COUNt <number>

7. Set the output function
   [SOURce:]FUNCtion[:SHAPE] <shape>

8. Set the signal amplitude
   [SOURce:]VOLTage[:LEVEL][:IMMediate][:AMPLitude] <amplitude>

9. Place the AFG in the wait-for-arm state
   INITiate[:IMMediate]

HP BASIC Program Example (LOG_SWP)

```
1 !RE-STORE"LOG_SWP"
2 !This program logarithmically sweeps from 1 Hz to 1 MHz in seven
3 !points.
4!
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40!
50 !Set up error checking
60 ON INTR 7 CALL Errmsg
70 ENABLE INTR 7;2
80 OUTPUT @Afg;"CLS"
90 OUTPUT @Afg;"SRE 32"
100 OUTPUT @Afg;"ESE 60"
```

Continued on Next Page
!Call the subprograms

CALL Rst
CALL Swp_pvss
!
WAIT .1 !allow interrupt to be serviced
OFF INTR 7
END
!

SUB Swp_pvss
Subprogram which sets a logarithmic sweep

COM @Afg
OUTPUT @Afg;"SOUR:FREQ1:MODE SWE;"; !sweep mode
OUTPUT @Afg;" :SOUR:FREQ1:STAR 1;"; !start frequency
OUTPUT @Afg;" :SOUR:FREQ1:STOP 1E6;"; !stop frequency
OUTPUT @Afg;" :SOUR:SWE:POIN 7;"; !sweep points
OUTPUT @Afg;" :SOUR:SWE:SPAC LOG;"; !logarithmic sweep
OUTPUT @Afg;" :SOUR:SWE:COUN INF;"; !sweep count
OUTPUT @Afg;" :SOUR:FUNC:SHAP SIN;"; !function
OUTPUT @Afg;" :SOUR:VOLT:LEV:IMM:AMPL 5 V" !amplitude
OUTPUT @Afg;"INIT:IMM" !wait-for-arm state

SUBEND
!

SUB Rst
Subprogram which resets the E1445.

COM @Afg
OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
ENTER @Afg;Complete
SUBEND
!

SUB Errmsg
Subprogram which displays E1445 programming errors

DIM Message$[256]

B=SPOLL(@Afg)

!End of statement if error occurs among coupled commands

OUTPUT @Afg;"
OUTPUT @Afg;"ABORT" !abort output waveform

REPEAT
OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
ENTER @Afg;Code,Message$
PRINT Code,Message$
UNTIL Code=0
STOP
SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, LOG_SWP.FRM, is in directory “VBPROG” and the Visual C example program, LOG_SWP.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Sweep Points Versus Time

To demonstrate the relationship between the number of points (frequencies) in a frequency sweep and the time of the sweep, the SWP_PVST program uses 100 frequency points to continuously sweep from 5 kHz to 15 kHz in 0.125 seconds. The program also shows you how to control the direction of a sweep.

Using the flowchart in Figure 4-1 as a guide, the steps of this program are:

1. **Select the frequency sweep mode**
   
   [SOURce:]FREQuency[1]:MODE <mode>

2. **Set the start frequency**
   
   [SOURce:]FREQuency[1]:STARt <start_freq>

3. **Set the stop frequency**
   
   [SOURce:]FREQuency[1]:STOP <stop_freq>

4. **Set the direction (up or down) of the frequency sweep**
   
   [SOURce:]SWEep:DIRection <direction>

5. **Set the number of points (frequencies) in the frequency sweep**
   
   [SOURce:]SWEep:POINts <number>

6. **Set the number of sweeps**
   
   [SOURce:]SWEep:COUNt <number>

7. **Set the frequency advance source**
   
   TRIGger:SWEep:SOURce <source>

8. **Set the duration of the sweep**
   
   [SOURce:]SWEep:TIME <number>

9. **Set the output function**
   
   [SOURce:]FUNCtion[:SHAPe] <shape>

10. **Set the signal amplitude**
    
    [SOURce:]VOLTage[:LEVEL][:IMMediate][:AMPLitude] <amplitude>

11. **Place the AFG in the wait-for-arm state**
    
    INITiate[:IMMediate]
HP BASIC Program Example (SWP_PVST)

1  !RE-STORE"SWP_PVST"
2  !This program sweeps from 5 kHz to 15 kHz in 0.1 seconds to
3  !demonstrate how to set the sweep time. The program also sets
4  !the direction of the sweep.
5  !
10  !Assign I/O path between the computer and E1445A.
20  ASSIGN @Afg TO 70910
30  COM @Afg
40  !
50  !Set up error checking
60  ON INTR 7 CALL Errmsg
70  ENABLE INTR 7;2
80  OUTPUT @Afg;"*CLS"
90  OUTPUT @Afg;"*SRE 32"
100 OUTPUT @Afg;"*ESE 60"
110  !
120  !Call the subprograms
130  CALL Rst
140  CALL Swp_pvst
150  !
160  WAIT .1 !allow interrupt to be serviced
170  OFF INTR 7
180  END
190  !
200  SUB Swp_pvst
210  !Subprogram which sets sweep direction, points, and time
220  COM @Afg
230  OUTPUT @Afg;"SOUR:FREQ1:MODE SWE;"
240  OUTPUT @Afg;"  :SOUR:FREQ1:STAR 5E3;"
250  OUTPUT @Afg;"  :SOUR:FREQ1:STOP 15E3;"
260  OUTPUT @Afg;"  :SOUR:SWEEP:DIR DOWN;"
270  OUTPUT @Afg;"  :SOUR:SWEEP:POIN 100;"
280  OUTPUT @Afg;"  :SOUR:SWE:COUN INF;"
290  OUTPUT @Afg;"  :TRIG:SWE:SOUR TIM;"
300  OUTPUT @Afg;"  :SOUR:SWE:TIME .12375;"
310  OUTPUT @Afg;"  :SOUR:FUNC:SHAP SIN;"
320  OUTPUT @Afg;"  :SOUR:VOLT:LEV:IMM:AMPL 5 V"
330  OUTPUT @Afg;"  :INIT:IMM"
340  SUBEND
350  !
360  SUB Rst
370  !Subprogram which resets the E1445.
380  COM @Afg
390  OUTPUT @Afg;"RST;"OPC?" !reset the AFG
400  ENTER @Afg;Complete
410  SUBEND
420  !
430  SUB Errmsg

Continued on Next Page
440  Errmsg:  !Subprogram which displays E1445 programming errors
450        COM @Afg
460        DIM Message$[256]
470        !Read AFG status byte register and clear service request bit
480        B=SPOLL(@Afg)
490        !End of statement if error occurs among coupled commands
500        OUTPUT @Afg;"
510        OUTPUT @Afg;"ABORT"  !abort output waveform
520        REPEAT
530        OUTPUT @Afg;"SYST:ERR?"  !read AFG error queue
540        ENTER @Afg;Code,Message$
550        PRINT  Code,Message$
560        UNTIL Code=0
570        STOP
560        SUBEND

Visual BASIC and
Visual C/C++ Program
Versions

The Visual BASIC example program, SWP_PVST.FRM, is in directory
“VBPROG” and the Visual C example program, SWP_PVST.C, is in
directory “VCPROG” on the CD that came with your HP E1445A.
To demonstrate the relationship between the number of frequencies in a frequency list and the time to hop through the list, program LIST_TME makes continuous passes through a frequency list where the frequencies are spaced 1 second apart.

Using the flowchart in Figure 4-1 as a guide, the steps of this program are:

1. **Select the frequency list mode**  
   
   `[SOURce:]FREQuency[1]:MODE <mode>`

2. **Specify the frequency list**  
   
   `[SOURce:]LIST2:FREQuency <freq_list>`

3. **Set the list repetition count**  
   
   `[SOURce:]SWEep:COUNt <number>`

4. **Set the frequency advance source**  
   
   `TRIGger:SWEep:SOURce <source>`

5. **Set the frequency hop rate**  
   
   `[SOURce:]SWEep:TIME <number>`

6. **Set the output function**  
   
   `[SOURce:]FUNCTION[:SHApe] <shape>`

7. **Set the number of waveform points (triangle wave)**  
   
   `[SOURce:]RAMP:POINts <number>`

8. **Set the signal amplitude**  
   
   `[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>`

9. **Place the AFG in the wait-for-arm state**  
   
   `INITiate[:IMMediate]`
1 !RE-STORE"LIST_TME"
2 !The following program steps through a frequency list at a rate
3 !such that a new frequency is output every 1 second.
4 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Af to 70910
30 COM @Af
40 !
50 !Set up error checking
60 ON INTR 7 CALL Ermsg
70 ENABLE INTR 7;2
80 OUTPUT @Af;"*CLS"
90 OUTPUT @Af;"*SRE 32"
100 OUTPUT @Af;"*ESE 60"
110 !
120 !Call the subprograms
130 CALL Rst
140 CALL List_time
150 !
160 WAIT .1 !Allow interrupt to be serviced
170 OFF INTR 7
180 END
190 !
200 SUB List_time
210 List_count: !Subprogram which continuously outputs a frequency list
220 !in which the frequencies are 1s apart
230 !
240 COM @Af
250 OUTPUT @Af;"SOUR:FREQ1:MODE LIST;"; !List mode
260 OUTPUT @Af;" :SOUR:LIST2:FREQ 2.5E3,5E3,7.5E3,10E3;"; !Freq list
270 OUTPUT @Af;":TRIG:SWE:SOUR TIM;"; !List advance source
280 OUTPUT @Af;":SOUR:SWE:TIME 3;"; !Time through list
290 OUTPUT @Af;":SOUR:FUNC:SHAPE TRI;"; !Function
300 OUTPUT @Af;":SOUR:RAMP:POIN 1E3;"; !1000 point waveform
310 OUTPUT @Af;":SOUR:VOLT:LEV:IMM:AMPL 5 V" !Amplitude
320 OUTPUT @Af;":INIT:IMM" !Wait-for-arm state
330 SUBEND
340 !
350 SUB Rst
360 Rst: !Subprogram which resets the E1445.
370 COM @Af
380 OUTPUT @Af;"*RST;*OPC?" !Reset the AFG
390 ENTER @Af;Complete
400 SUBEND
410 !
420 SUB Ermsg
430 Ermsg: !Subprogram which displays E1445 programming errors
440 !

Continued on Next Page
450  DIM Message$[256]
460  !Read AFG status byte register and clear service request bit
470  B=SPOOL(@Afg)
480  !End of statement if error occurs among coupled commands
490  OUTPUT @Afg;""
500  OUTPUT @Afg;"ABORT" !abort output waveform
510  REPEAT
520  OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
530  ENTER @Afg;Code,Message$
540  PRINT Code,Message$
550  UNTIL Code=0
560  STOP
570  SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, LIST_TME.FRM, is in directory “VBPROG” and the Visual C example program, LIST_TME.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Sweeping Arbitrary Waveforms

When sweeping arbitrary waveforms, the start and stop frequencies specified are start and stop sample rates. The corresponding output frequency is the sample rate divided by the number of points in the waveform. The following programs demonstrate how to sweep arbitrary waveforms by specifying starting and stopping sample rates.

Sweeping Sin(x)/x and Pseudo-Random Noise Signals

The SWP_ARB program computes a 4096 point, $8\frac{1}{2}$ cycle, phase-continuous Sin(x)/x waveform with a peak amplitude of 1V; and a 4096 point pseudo-random noise signal.

By sweeping the Sin(x)/x signal, the different frequencies of the signal are swept simultaneously. Starting and stopping sample rates are specified such that Sin(x)/x is swept from 1 kHz to 2 kHz.

The noise signal is a “comb” of frequencies separated by the repetition rate of the signal. The pseudo-random signal is repetitive. Sweeping this signal effectively decreases the repetition rate by increasing the length of the signal. The sampling bandwidth is 40 kHz with an effective bandwidth of 20 kHz. The output is swept from 10 Hz to 20 Hz.

HP BASIC Program Example (SWP_ARB)

```basic
1 !RE-STORE"SWP_ARB"
2 !This program sweeps two arbitrary waveforms: sin(x)/x and pseudo
3 !random noise. The 4096 point waveforms are swept from 4.096 MHz to
4 !8.192 MHz which results in an output frequency sweep from 1 kHz to
5 !2 kHz.
6 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !
50 !Set up error checking
60 ON INTR 7 CALL Errmsg
70 ENABLE INTR 7;2
80 OUTPUT @Afg:"*CLS"
90 OUTPUT @Afg:"*SRE 32"
100 OUTPUT @Afg:"*ESE 60"
110 !
120 !Call the subprograms
130 CALL Wvfm_m anage
140 !Abort the current waveform, select and initiate either the sin(x)/x
150 !(S1) or noise waveform (N1) sequence. Comment out (!) the line of
160 !the one you DO NOT want to select.
170 OUTPUT @Afg:"ABORT"
180 OUTPUT @Afg:"SOUR:FUNC:USER S1" !select waveform sequence (Sin(x))
190 OUTPUT @Afg:"SOUR:FUNC:USER N1" !select waveform sequence (Noise)
200 OUTPUT @Afg:"INIT:IMM" !wait-for-arm state
```

Continued on Next Page
210 !
220 WAIT .1 !allow interrupt to be serviced
230 OFF INTR 7
240 END
250 !
260 SUB Wvfm_manage
270 Wvfm_manage: !Subprogram which calls the subprograms which delete
280 !all existing waveforms and define Sin(x)/x and
290 !pseudo random noise.
300 COM @Afg
310 CALL Rst
320 CALL Wf_del
330 CALL Sinx_def !Comment out this line if line 180 is commented
340 CALL Noise_def !Comment out this line if line 190 is commented
350 SUBEND
360 !
370 SUB Sinx_def
380 Sinx_def !Set sweep mode, specify start and stop sample rates for a
390 !1 kHz to 2 kHz sweep, set arbitrary waveform function.
400 !Compute waveform (Sin(x)/x), define waveform segment and
410 !sequence.
420 COM @Afg
430 OUTPUT @Afg;"SOUR:FREQ1:MODE SWE;"; !sweep mode
440 OUTPUT @Afg;" :SOUR:FREQ1:STAR 4.096E6;"; !start sample rate
450 OUTPUT @Afg;" :SOUR:FREQ1:STOP 8.192E6;"; !stop sample rate
460 OUTPUT @Afg;":SOUR:SWE:COUN INF;"; !sweep count
470 OUTPUT @Afg;":SOUR:FUNC:SHAP USER;"; !function (arbitrary)
480 OUTPUT @Afg;" :SOUR:VOLT:LEV:IMM:AMPL 1.1V" !scale arb values
490 !
500 DIM Waveform(1:4096)
510 FOR I=-2047 TO 2048
520 IF I=0 THEN I=1.E-38
530 Waveform(I+2048)=((SIN(2*PI*.53125*I/256))/(.53125*I/256)*.159154943092)
540 NEXT I
550 !
560 OUTPUT @Afg;"LIST:SEGM:SEL SIN_X" !select segment
570 OUTPUT @Afg;" LIST:SEGM:DEF 4096" !reserve memory
580 OUTPUT @Afg;" LIST:SEGM:VOLT";Waveform(*) !load points
590 !
600 OUTPUT @Afg;"LIST:SSEQ:SEL S1" !select sequence
610 OUTPUT @Afg;" LIST:SSEQ:DEF 1" !number of segments
620 OUTPUT @Afg;" LIST:SSEQ:SEQ SIN_X" !segment order in sequence
630 SUBEND
640 !
650 SUB Noise_def
660 Noise_def: !Set sweep mode, specify start and stop sample rates for a
670 !10 Hz to 20 Hz sweep, set arbitrary waveform function.
680 !Compute waveform (Noise), define waveform segment and
690 !sequence.
700 COM @Afg

Continued on Next Page
710 OUTPUT @Afg;"SOUR:FREQ1:MODE SWE;"; !sweep mode
720 OUTPUT @Afg;" :SOUR:FREQ1:STAR 40.96E3;"; !start sample rate
730 OUTPUT @Afg;" :SOUR:FREQ1:STOP 81.92E3;"; !stop sample rate
740 OUTPUT @Afg;" :SOUR:SWE:COUN INF;"; !sweep count
750 OUTPUT @Afg;" :SOUR:FUNC:SHAP USER;"; !function (arbitrary)
760 OUTPUT @Afg;" :SOUR:VOLT:LEV:IMM:AMPL 1.1V" !scale arb values
770!
780 DIM Waveform(1:4096)
790 FOR I=1 TO 4096
800 Waveform(I)=2*RND-1.
810 NEXT I
820 OUTPUT @Afg;"LIST:SEGM:SEL NOISE" !select segment
830 OUTPUT @Afg;" :LIST:SEGM:DEF 4096" !reserve memory
840 OUTPUT @Afg;" :LIST:SEGM:VOLT ",Waveform(*) !load points
850!
860 OUTPUT @Afg;"LIST:SSEQ:SEL N1" !select sequence
870 OUTPUT @Afg;" :LIST:SSEQ:DEF 1" !number of segments
880 OUTPUT @Afg;" :LIST:SSEQ:SEQ NOISE" !segment order in sequence
890 SUBEND
900!
910 SUB Rst
920 Rst: !Subprogram which resets the E1445.
930 COM @Afg
940 OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
950 ENTER @Afg;Complete
960 SUBEND
970!
980 SUB Wf_del
990 Wf_del: !Subprogram which deletes all sequences and segments.
1000 COM @Afg
1010 OUTPUT @Afg;"FUNC:USER NONE" !select no sequences
1020 OUTPUT @Afg;" :LIST:SSEQ:DEL:ALL" !delete all sequences
1020 OUTPUT @Afg;" :LIST:SEGM:DEL:ALL" !delete all waveform segments
1040 SUBEND
1050!
1060 SUB Errmsg
1070 Errmsg: !Subprogram which displays E1445 programming errors
1080 COM @Afg
1090 DIM Message$[256]
1100 !Read AFG status byte register and clear service request bit
1110 B=SPOLL(@Afg)
1120 !End of statement if error occurs among coupled commands
1130 OUTPUT @Afg;""
1140 OUTPUT @Afg;"ABORT" !abort output waveform
1150 REPEAT
1160 OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
1170 ENTER @Afg;Code,Message$
1180 PRINT Code,Message$
1190 UNTIL Code=0
1200 STOP
1210 SUBEND
Program Modifications

To select another waveform, comment out (!) line 180 or 190 depending on the waveform sequence (S1 or N1) you DO NOT want to output. You must also comment out line 330 if line 180 is commented, or line 340 if line 190 is commented.

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, SWP_ARB.FRM, is in directory “VBPROG” and the Visual C example program, SWP_ARB.C, is in directory “VCPROG” on the CD that came with your HP E1445A.

AC Output Leveling

The SWP_LEVL program sets up a sine wave frequency sweep from 0 Hz to 10 MHz and uses the AFG’s 10 MHz filter and AC output leveling to maintain a constant amplitude over the span.

With the flowchart in Figure 4-1 as a guide, the steps of this program are:

1. Select the frequency sweep mode
   [SOURce:]FREQuency[1]:MODE <mode>

2. Set the start frequency
   [SOURce:]FREQuency[1]:STARt <start_freq>

3. Set the stop frequency
   [SOURce:]FREQuency[1]:STOP <stop_freq>

4. Set the number of sweeps
   [SOURce:]SWEep:COUNt <number>

5. Set the output function
   [SOURce:]FUNCtion[:SHAPe] <shape>

6. Set the signal amplitude
   [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>

7. Select the output filter
   OUTPut[1]:FILTer[:LPASs]:FREQuency <frequency>

8. Enable the output filter
   OUTPut[1]:FILTer[:LPASs][:STATe] <mode>

9. Place the AFG in the wait-for-arm state
   INITiate[:IMMediate]
HP BASIC Program Example (SWP_LEVL)

1 !RE-STORE"SWP_LEVL"
2 !This program enables output leveling over the 0 Hz to 10 MHz sweep.
3 !
4 !Assign I/O path between the computer and E1445A.
10 ASSIGN @Afg TO 70910
20 COM @Afg
30 !
50 !Set up error checking
60 ON INTR 7 CALL Errmsg
70 ENABLE INTR 7;2
80 OUTPUT @Afg:""CLS"
90 OUTPUT @Afg:""SRE 32"
100 OUTPUT @Afg:""ESE 60"
110 !
120 !Call the subprograms
130 CALL Rst
140 CALL Swp_levl
150 !
160 WAIT .1 !allow interrupt to be serviced
170 OFF INTR 7
180 END
190 !
200 SUB Swp_levl
210 Swp_levl: !Subprogram which sets output leveling for sweeping from 0 TO 10 MHz
220 !
230 COM @Afg
240 OUTPUT @Afg;"SOUR:FREQ1:MODE SWE;"; !sweep mode
250 OUTPUT @Afg;" :SOUR:FREQ1:STAR 0;"; !start frequency
260 OUTPUT @Afg;" :SOUR:FREQ1:STOP 10E6;"; !stop frequency
270 OUTPUT @Afg;" :SOUR:SWE:COUN INF;"; !sweep count
280 OUTPUT @Afg;";"SOUR:FUNC:SHAP SIN;"; !function
290 OUTPUT @Afg;";"SOUR:VOLT:LEV:IMM:AMPL 5 V"!amplitude
300 OUTPUT @Afg;"OUTP1:FILT:LPAS:FREQ 10 MHZ"!filter cutoff frequency
310 OUTPUT @Afg;"OUTP1:FILT:LPAS:STAT ON"!enable output filter
320 OUTPUT @Afg;"INIT:IMM"!wait-for-arm state
330 SUBEND
340 !
350 SUB Rst
360 Rst: !Subprogram which resets the E1445.
370 COM @Afg
380 OUTPUT @Afg;":RST:"OPC?"!reset the AFG
390 ENTER @Afg;Complete
400 SUBEND
410 !
420 SUB Errmsg
430 Errmsg: !Subprogram which displays E1445 programming errors
440 COM @Afg

Continued on Next Page
450    DIM Message$[256]
460    !Read AFG status byte register and clear service request bit
470    B=SPOLL(@Afg)
480    !End of statement if error occurs among coupled commands
490    OUTPUT @Afg;"
500    OUTPUT @Afg;"ABORT" !abort output waveform
510    REPEAT
520      OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
530      ENTER @Afg,Code,Message$
540      PRINT Code,Message$
550      UNTIL Code=0
560    STOP
570    SUBEND

**Visual BASIC and Visual C/C++ Program Versions**

The Visual BASIC example program, SWP_LEVL.FRM, is in directory “VBPROG” and the Visual C example program, SWP_LEVLC.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Frequency-Shift Keying

Frequency-shift keying (FSK) changes the frequency of the output waveform or sample rate based on the signal level of the frequency-shift keying control source. FSK frequencies can range from 0.0 Hz to 10 MHz.

The command subsystems associated with frequency-shift keying include:

- [SOURce:]ROScillator
- TRIGger
- [SOURce:]FREQuency[1]
  - FSK mode and related commands

FSK Using the “FSK” Control Source

The FSK1 program shows the basic steps involved in setting up and using the frequency-shift keying function of the AFG. A 5 V, 1 MHz square wave control signal is applied to the AFG’s front panel “FSK” port. Output frequencies of 5 MHz and 10 MHz occur as the level of the 1 MHz signal changes.

Note

When the frequency shifts, there is a delay of 20 reference oscillator clock cycles before the frequency is active. This delay occurs with all reference oscillator sources.

Using the flowchart in Figure 4-1 as a guide, the steps of this program are:

1. Select the 42.9 MHz reference oscillator
   [SOURce:]ROScillator:SOURce <source>

2. Select the frequency generator which allows frequency-shift keying
   TRIGger[:STARt]:SOURce <source>

3. Select the frequency-shift keying mode
   [SOURce:]FREQuency[1]:MODE <mode>

4. Select the FSK frequencies
   [SOURce:]FREQuency[1]:FSKey <frequency1>,<frequency2>

5. Select the FSK control source
   [SOURce:]FREQuency[1]:FSKey:SOURce <source>

6. Set the output function
   [SOURce:]FUNCTION[:SHApe] <shape>
7. Set the signal amplitude
[SOURce:VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>

8. Place the AFG in the wait-for-arm state
INITiate[:IMMediate]

HP BASIC Program Example (FSK1)

1 !RE-STORE "FSK1"
2 !This program shifts between 5 MHz and 10 MHz based on a 1 MHz
3 !control signal applied to the “FSK” control source. The
4 !program also queries the FSK frequencies and the FSK control
5 !source.
6 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !
50 !Set up error checking
60 ON INTR 7 CALL Errmsg
70 ENABLE INTR 7,2
80 OUTPUT @Afg;"CLS"
90 OUTPUT @Afg;"SRE 32"
100 OUTPUT @Afg;"ESE 60"
110 !
120 !Call the subprograms
130 CALL Rst
140 CALL Fsk
150 CALL Fsk_info
160 !
170 WAIT .1 !allow interrupt to be serviced
180 OFF INTR 7
190 END
200 !
210 SUB Fsk
220 Fsk: !Subprogram which sets up frequency-shift keying and the front
230 !panel FSK In BNC as the control source.
240 !
250 OUTPUT @Afg;"ROSC:SOUR INT1;"
260 OUTPUT @Afg;"TRIG:STAR:SOUR INT1;"
270 OUTPUT @Afg;"SOUR:FREQ1:MODE FSK;"
280 OUTPUT @Afg;"SOUR:FREQ1:FSK 5E6,10E6;"
290 OUTPUT @Afg;"SOUR:FREQ1:FSK:SOUR EXT;"
300 OUTPUT @Afg;"SOUR:FUNC:SHAP SIN;"
310 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5 V"
320 OUTPUT @Afg;"OUTP1:FILT:LPAS:FREQ 10 MHZ"
330 OUTPUT @Afg;"OUTP1:FILT:LPAS:STAT ON"
340 OUTPUT @Afg;"INIT:IMM"
350 SUBEND
360 !

Continued on Next Page
SUB Rst
Rst: !Subprogram which resets the E1445.

COM @Afg
OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
ENTER @Afg;Complete

SUBEND

Table: Fsk_info

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM @Afg</td>
<td></td>
</tr>
<tr>
<td>DIM Frequencies$[80]</td>
<td></td>
</tr>
</tbody>
</table>
| OUTPUT @Afg;"SOUR:FREQ1:FSK?" | !query FSK frequencies
| ENTER @Afg;Frequencies$ | |
| OUTPUT @Afg;"SOUR:FREQ:FSK:SOUR?" | !query FSK source
| ENTER @Afg;Source$ | |
| PRINT "FSK frequencies are: ";Frequencies$ | |
| PRINT "FSK control source is: ";Source$ | |

SUBEND

Table: Errmsg

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM @Afg</td>
<td></td>
</tr>
<tr>
<td>DIM Message$[256]</td>
<td></td>
</tr>
</tbody>
</table>
| B=SPOLL(@Afg) | !Read AFG status byte register and clear service request bit
| | !End of statement if error occurs among coupled commands
| OUTPUT @Afg;"" | |
| OUTPUT @Afg;"ABORT" | !abort output waveform
| REPEAT | |
| OUTPUT @Afg;"SYST:ERR?" | !read AFG error queue
| ENTER @Afg;Code,Message$ | |
| PRINT Code,Message$ | |
| UNTIL Code=0 | |
| STOP | |

SUBEND

Upon completion, the program displays:

**FSK frequencies are:**
+5.000000000E+006,+1.000000000E+007

**FSK control source is:** EXT

**Visual BASIC and Visual C/C++ Program Versions**
The Visual BASIC example program, FSK1.FRM, is in directory "VBPROG" and the Visual C example program, FSK1.C, is in directory "VCPROG" on the CD that came with your HP E1445A.
FSK Using the TTLTrg<sub>n</sub> Control Source

The FSK2 program sets up frequency-shift keying using a TTLTrg<sub>n</sub> trigger line as the control source. The TTLTrg trigger line is driven by the HP E1406A Command Module.

Using the flowchart in Figure 4-1 as a guide, the steps of this program are:

1. **Select the frequency-shift keying mode**
   
   [SOURce:]FREQuency[1]:MODE <mode>

2. **Select the FSK frequencies**
   
   [SOURce:]FREQuency[1]:FSKey <frequency1>,<frequency2>

3. **Select the FSK control source**
   
   [SOURce:]FREQuency[1]:FSKey:SOURce <source>

4. **Set the output function**
   
   [SOURce:]FUNCTION[:SHAPe] <shape>

5. **Set the signal amplitude**
   
   [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>

6. **Place the AFG in the wait-for-arm state**
   
   INITiate[:IMMediate]

**HP BASIC Program Example (FSK2)**

1 !RE-STORE "FSK2"
2 !This program shifts between 1 MHz and 2 MHz based on a control signal supplied by the HP E1406 Command Module on TTLTRG
3 !trigger line 5.
4 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 ASSIGN @Cmd_mod TO 70900
40 COM @Afg,@Cmd_mod
50 !
60 !Set up error checking
70 ON INTR 7 CALL Errmsg
80 ENABLE INTR 7;2
90 OUTPUT @Afg;"*CLS"
100 OUTPUT @Afg;"*SRE 32"
110 OUTPUT @Afg;"*ESE 60"
120 !
130 !Call the subprograms which reset the AFG, set up frequency-shift keying, and which set up the TTLTrg5 trigger line.
140 CALL Rst
150 CALL Fsk_ttl
160 CALL Setup_ttl5
170 WAIT .1 !allow interrupt to be serviced
190 OFF INTR 7

*Continued on Next Page*
SUB Fsk_ttl

Subprogram which sets up frequency-shift keying and trigger line TTLTRG 5 as the control source.

COM @Afg,@Cmd_mod

OUTPUT @Afg;"SOUR:FREQ1:MODE FSK;";%FSK mode
OUTPUT @Afg;" :SOUR:FREQ1:FSK 1E6,2E6;";%FSK frequencies
OUTPUT @Afg;" :SOUR:FREQ:FSK:SOUR TTLT5;";%FSK source
OUTPUT @Afg;" :SOUR:FUNC:SHAP SIN;";%function
OUTPUT @Afg;" :SOUR:VOLT:LEV:IMM:AMPL 5 V";%amplitude
OUTPUT @Afg;"INIT:IMM";%wait-for-trigger state

SUBEND

SUB Setup_ttl5

Subprogram which sets up trigger line TTLTrg5 to change the AFG frequency-shift keying frequencies.

COM @Afg,@Cmd_mod

OUTPUT @Cmd_mod;"OUTP:TTLT5:STAT ON";%enable line TTLTrg5
OUTPUT @Cmd_mod;"OUTP:TTLT5:SOUR INT";%drive TTLTrg5 internally

Loop which shifts frequency

Disp “Press ‘Continue’ to shift frequency”

PAUSE

Disp ""

FOR I=1 TO 10

IF BIT(I,0) THEN

OUTPUT @Cmd_mod;"OUTP:TTLT5:LEV:IMM 1";%level is electrically low

ELSE

OUTPUT @Cmd_mod;"OUTP:TTLT5:LEV:IMM 0";%level is electrically high

END IF

WAIT 1

NEXT I

SUBEND

SUB Rst

Subprogram which resets the E1445.

COM @Afg

OUTPUT @Afg;"**RST;**OPC?";%reset the AFG

ENTER @Afg;Complete

SUBEND

SUB Errmsg

Subprogram which displays E1445 programming errors

COM @Afg,@Cmd_mod

DIM Message$[256]

!Read AFG status byte register and clear service request bit

B=SPOLL(@Afg)

!End of statement if error occurs among coupled commands

OUTPUT @Afg;""

OUTPUT @Afg;"ABORT";%abort output waveform

Continued on Next Page
Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, FSK2.FRM, is in directory “VBPROG” and the Visual C example program, FSK2.C, is in directory “VCPROG” on the CD that came with your HP E1445A.

FSK Using an Arbitrary Waveform

The FSK_ARB program uses frequency-shift keying with an arbitrary waveform to shift between two sample rates. The control source is a 5 V signal applied to the AFG’s front panel “FSK In” BNC connector.

HP BASIC Program Example (FSK_ARB)

1   !RE-STORE "FSK_ARB"
2   !This program shifts the frequency of an arbitrary waveform
3   !based on a control signal applied to the “FSK In” BNC connector.
4   !
10  !Assign I/O path between the computer and E1445A.
20  ASSIGN @Afg TO 70910
30  COM @Afg
40  !
50  !Set up error checking
60  ON INTR 7 CALL Errmsg
70  ENABLE INTR 7;2
80  OUTPUT @Afg;"*CLS"
90  OUTPUT @Afg;"*SRE 32"
100 OUTPUT @Afg;"*ESE 60"
110  
130  CALL Rst
150  
180  OUTPUT @Afg;"SOUR:FREQ1:MODE FSK;";   !FSK mode
190  OUTPUT @Afg;" :SOUR:FREQ1:FSK 4.096E6,8.192E6;"; !sample frequencies
200 OUTPUT @Afg;" :SOUR:FREQ1:FSK:SOUR_EXT;"; !FSK source
210 OUTPUT @Afg;" :SOUR:FUNC:SHAP USER;"; !function
220 OUTPUT @Afg;" :SOUR:VOLT:LEV:IMM:AMPL 1.1V" !amplitude
250  
Continued on Next Page
270 OUTPUT @Afg;"SOUR:FUNC:USER SIN_X_OUT"
280 OUTPUT @Afg;"INIT:IMM"
290 !
300 WAIT .1 !allow interrupt to be serviced
310 OFF INTR 7
320 END
330 SUB Sinx_def
340 Sinx_def: !Define Sin(x)/x waveform and output sequence.
350 COM @Afg
360 DIM Waveform(1:4096)
370 FOR I=-2047 TO 2048
380 IF I=0 THEN I=1.E-38
390 Waveform(I+2048)=((SIN(2*PI*.53125*I/256))/(.53125*I/256)*.159154943092)
400 NEXT I
410 !
420 OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL SIN_X" !select segment
430 OUTPUT @Afg;"SOUR:LIST1:SEGM:DEF 4096" !reserve memory
440 OUTPUT @Afg;"SOUR:LIST1:SEGM:VOLT";Waveform(*) !load points
450 !
460 OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEL SIN_X_OUT" !select sequence
470 OUTPUT @Afg;"SOUR:LIST1:SSEQ:DEF 1" !number of segments
480 OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEQ SIN_X" !segment order
490 SUBEND
500 !
510 SUB Rst
520 Rst: !Subprogram which resets the E1445.
530 COM @Afg
540 OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
550 ENTER @Afg;Complete
560 SUBEND
570 !
580 SUB Wf_del
590 Wf_del: !Subprogram which deletes all sequences and segments.
600 COM @Afg
610 OUTPUT @Afg;"FUNC:USER NONE" !select no sequences
620 OUTPUT @Afg;"LIST:SSEQ:DEL:ALL" !delete all sequences
630 OUTPUT @Afg;"LIST:SEGM:DEL:ALL" !delete all segments
640 SUBEND
650 !
660 SUB Errmsg
670 Errmsg: !Subprogram which displays E1445 programming errors
680 COM @Afg
690 DIM Message$[256]
700 !Read AFG status byte register and clear service request bit
710 B=SPOLL(@Afg)
720 !End of statement if error occurs among coupled commands
730 OUTPUT @Afg;""
740 OUTPUT @Afg;"ABORT" !abort output waveform
750 REPEAT
760 OUTPUT @Afg;"SYST:ERR?" !read AFG error queue

Continued on Next Page
Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, FSK_ARB.FRM, is in directory “VBPROG” and the Visual C example program, FSK_ARB.C, is in directory “VCPROG” on the CD that came with your HP E1445A.

Sweeping and FSK Program Comments

The following information is associated with sweeping, frequency lists, and frequency-shift keying. Included are details on the operation of these functions, and on the various modes, ranges, etc., used in the programs in this chapter.

Reference Oscillator Sources

There are five reference oscillator sources for the AFG which are selected by the [SOURce:]ROSCillator:SOURce command:

- **CLK10** – The VXIbus CLK10 (10 MHz) line.
- **EXTERNAL** – The AFG’s front panel “Ref/Smpl In” BNC.
- **ECLTrg0** or **1** – The VXIbus ECL trigger lines.
- **INTernal[1]** – The internal 42.94967296 MHz oscillator.
- **INTernal[2]** – The internal 40 MHz oscillator.

The INTernal[1] reference oscillator is recommended for use with the Direct-Digital-Synthesis (DDS) time base ([SOURce:]FREQuency[1] subsystem) for high resolution and frequency range.

INTernal[1] is the default reference oscillator source. Thus, in many programs, the source is not specified.

Sample Sources

Sweeping, frequency lists, and frequency-shift keying are only available using the direct-digital-synthesis (DDS) frequency synthesis method ([SOURce:]FREQuency[1] subsystem). The method by which the output is advanced to the next sample point is selected with the TRIGger[:STARt]:SOURce command. The available sources are:

- **BUS** – The HP-IB Group Execute Trigger (GET command) or the IEEE-488.2 *TRG common command.
- **ECLTrg0** or **ECLTrg1** – The VXIbus ECL trigger lines.
- **EXTERNAL** – The AFG’s front panel “Ref/Smpl In” BNC.
- **HOLD** – Suspends sample generation.
- **INTernal[1]** – The [SOURce:]FREQuency[1] subsystem DDS frequency synthesis.
- **INTernal[2]** – The [SOURce:]FREQuency2 subsystem Divide-by-n frequency synthesis.
• **TTLTrg0 through 7** – The VXIbus TTL trigger lines.

INTernal[1] is the source selected at power-on or following a reset, but is specified in the programs to emphasize that sweeping, frequency lists, and frequency-shift keying are only allowed when INTernal[1] is the source.

### AFG Frequency Modes

There are four frequency “modes” available using the INTernal1 sample source (DDS timebase). The modes selected by the [SOURce:]FREQuency[1]:MODE command are:

- **CW | FIXed** – single frequency mode
- **FSKey** – frequency shift keying mode
- **LIST** – frequency list mode
- **SWEep** – frequency sweep mode

### Frequency Range: Sweeping and Sampling

The frequency range for sweeping or sampling depends on the reference oscillator used. However, for the INTernal1 42.94967296 MHz oscillator, the range for swept sine, square, triangle, and ramp waveforms, and the swept sampling range for arbitrary waveforms is 0 Hz to 10.73741824 MHz.

### Frequency Range: Frequency Lists and FSK

The maximum number of frequencies in a frequency list is 256. The range of frequencies allowed in a frequency list or for frequency-shift keying depends on the output function:

- **Sine Wave and Arbitrary Waveforms**: The minimum frequency is 0 Hz; the maximum frequency is the selected reference oscillator frequency divided by 4.

- **Square Waves**: The minimum frequency is 0 Hz; the maximum frequency is the selected reference oscillator frequency divided by 16.

- **Ramp and Triangle Waveforms**: The minimum frequency is 0 Hz; the maximum frequency is the selected reference oscillator frequency divided by four multiplied by the number of waveform points (Rosc/(4*npts)).

### Frequency Doubling

For all waveforms except sine waves, the sweeping or sampling range can be doubled using the [SOURce:]FREQuency[1]:RANGe command. **Frequency doubling is enabled by specifying a range that is greater than the maximum “undoubled” frequency allowed for a given waveform.** The maximum undoubled frequencies for the various waveforms are given below.

- **Arbitrary Waveforms**: The maximum undoubled frequency is the current reference oscillator frequency divided by 4.

- **Square Waves**: The maximum undoubled frequency is the current reference oscillator frequency divided by 16.
**Triangle and Ramp Waves:** The maximum undoubled frequency is the current reference oscillator frequency divided by 4, further divided by the [SOURce:]RAMP:POINts value.

**Sweep Count and Frequency List Repetition Count**

The sweep count specifies the number of sweeps to occur, or the number of passes through the frequency list before the AFG returns to the idle state from the wait-for-arm state (see Chapter 5).

The sweep count, set with the [SOURce:]SWEep:COUNt command has a range from 1 to 2,147,483,647 or INFinity. The default count is 1. Continuous sweeps, or loops through a frequency list can be stopped with the ABORt command.

**Arbitrary Block Data**

Data sent to the AFG in an arbitrary block is in a binary format. The encoding syntax for downloading frequency list data in this format is shown Figure 4-2.

**In a definite length arbitrary block:**

- "#" indicates the data to be sent is in an arbitrary block
- "non-zero digit" is a single digit number which shows the number of digits contained in "digits". For example, if the "digits" value is 100 or 2000, the "non-zero digit" value is 3 or 4, respectively.
- "8-bit data bytes" is the data (i.e. frequencies) sent to the AFG. Note that there are eight bytes per frequency list frequency.

In the LISTDEF program on page 131, a list of 100 frequencies is downloaded using the definite length block format. In the definite length encoding syntax, "digit" specifies the number of bytes downloaded (800). Since ‘800’ is three characters, “non-zero digit” is ‘3’.

<table>
<thead>
<tr>
<th>Repeated Count</th>
<th>Sweep Count</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>The sweep count specifies the number of sweeps to occur, or the number of passes through the frequency list before the AFG returns to the idle state from the wait-for-arm state (see Chapter 5).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The sweep count, set with the [SOURce:]SWEep:COUNt command has a range from 1 to 2,147,483,647 or INFinity. The default count is 1. Continuous sweeps, or loops through a frequency list can be stopped with the ABORt command.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data sent to the AFG in an arbitrary block is in a binary format. The encoding syntax for downloading frequency list data in this format is shown Figure 4-2.</td>
</tr>
</tbody>
</table>

**Figure 4-2. Arbitrary Block Data Diagram**

- In a definite length arbitrary block:  

  - "#" indicates the data to be sent is in an arbitrary block  
  - "non-zero digit" is a single digit number which shows the number of digits contained in "digits". For example, if the "digits" value is 100 or 2000, the “non-zero digit” value is 3 or 4, respectively. 
  - "8-bit data bytes" is the data (i.e. frequencies) sent to the AFG. Note that there are eight bytes per frequency list frequency. 

In the LISTDEF program on page 131, a list of 100 frequencies is downloaded using the definite length block format. In the definite length encoding syntax, “digit” specifies the number of bytes downloaded (800). Since ‘800’ is three characters, “non-zero digit” is ‘3’.
In an indefinite length arbitrary block:

- “#” indicates the data to be sent is in an arbitrary block.
- “0” indicates that an indefinite length block of data is to be sent.
- “8-bit data bytes” is the data (i.e. frequencies) sent to the AFG. There are eight bytes per frequency list frequency.
- NL ^END means a line feed (LF) is sent with END (EOI) asserted. It indicates to the AFG that the end of the data block has been reached.

Additional information on arbitrary block data can be found in ANSI/IEEE Standard 488.2-1987 IEEE Standard Codes, Formats, Protocols, and Common Commands.

Frequency Points

The number of frequencies generated (points) in a frequency sweep can be from 2 to 1,073,741,824. The default number is 800. The number of points is set with the [SOURce:]SWEep:POINts command and applies to sweeps only.

Sweep Spacing

The spacing between the frequencies (points) in a sweep can be either linear or logarithmic as set by [SOURce:]SWEep:SPACing. Linear sweeps can start at 0 Hz. Logarithmic sweeps can start at the reference oscillator frequency/4,294,967,296/npts. The number of waveform points (npts) for sine waves and arbitrary waveforms is 1, for square waves npts is 4, for ramp and triangle waves npts is the number of ramp points.

Sweep Direction

The direction of the frequency sweep can be up or down. When the direction is up, the sweep begins at the specified start frequency and stops at the specified stop frequency. When the direction is down, the sweep begins at the stop frequency and stops at the start frequency. You must stop (abort) the sweep before changing direction.

For arbitrary waveforms when the direction is up, sampling begins at the start frequency and stops at the stop frequency. When the direction is down, sampling begins at the stop frequency and stops at the start frequency. You must stop (abort) sampling before changing direction.

Frequency lists begin with the first frequency in the list and end with the last frequency. There is no directional control.
Sweep Time

The number of frequencies (points) in a sweep and the number of frequencies in a frequency list change the duration of the sweep or pass through the list. The relationship between the sweep or list time ([SOURce:]SWEep:TIME), the time between frequencies (TRIGger:SWEep:TIMer), and the number of frequencies (points) is shown below:

\[
\text{TIME} = \text{TIMer} \times (\text{points} - 1)
\]

Changing the number of points keeps the value set by the last command sent (either [SOURce:]SWEep:TIME or TRIGger:SWEep:TIMer) and changes the other. Changing TIME or TIMer affects the other.

Sweep Advance Source

The source which advances the sweep or list to the next frequency is set with the TRIGger:SWEep:SOURce command. The available sources are:

- **BUS** – The HP-IB Group Execute Trigger (GET) command or the IEEE-488.2 *TRG* common command.
- **HOLD** – Suspend sweep or frequency list advance triggering. Advance to the next frequency (sweeping or in a list) using TRIGger:SWEep:IMMediate.
- **LINK** – The next valid start arm advances the sweep or list.
- **TIMer** – The [SOURce:]SWEep:TIME and/or TRIGger:SWEep:TIMer commands control the sweep and frequency list advance timing (default source).
- **TTLTrg0 through TTLTrg7** – The VXIbus TTL trigger lines.

More information on the sweep advance source can be found in Chapter 5, “Arming and Triggering”.

Specifying a Sweep Time

The sweep time (set by [SOURce:]SWEep:TIME) is the period from the generation of the first frequency in the sweep or list to the generation of the last frequency (see below).

\[
\begin{array}{cccccc}
  f_1 & f_2 & f_3 & f_4 & \ldots & f_n \\
  t & t & t & t & & t \\
\end{array}
\]

Sweep Time

The duration (t) of each frequency (except the last \( f_n \)) is:

\[\text{specified sweep time} / (\text{frequency points} - 1)\]

For multiple sweeps or repetitions through the list, the duration of the last frequency \( f_n \) is also t. To maintain a constant rate between sweeps or repetitions, the duration of \( f_n \) must be accounted for as follows:

\[\text{Sweep time specified} = \text{Sweep repetition time desired} \times ((\text{points} - 1)/\text{points})\]
Sweep Points Versus Time

In SWP_PVST (Sweep Points Versus Time) on page 136, the program continually sweeps 100 frequency points in 0.125 seconds. To maintain this rate continuously, the time between the last frequency point and the first point is accounted for as follows:

\[ \text{Sweep time specified} = 0.125 \times \left(\frac{99}{100}\right) = 0.125 \times 0.99 = 0.12375 \]

Thus, the actual sweep time specified is 0.12375 seconds.

Frequency Lists Versus Time

In the LIST_TME program (Frequency Lists Versus Time) on page 139, the program outputs a new frequency every 1 second. To maintain this rate continuously, the time between the last frequency in the list and the first frequency is accounted for as follows:

\[ \text{Repetition rate specified} = 4 \times \left(\frac{3}{4}\right) = 4 \times 0.75 = 3 \]

Thus, the actual repetition rate specified is 3 (seconds).

The minimum and maximum sweep times and frequency list repetition rates are based on the number of points and frequencies, and are calculated as follows:

\[ \text{Minimum} = 1.25 \text{ mS} \times (\text{points} - 1) \]
\[ \text{Maximum} = 4.19430375 \text{ S} \times (\text{points} - 1) \]

Again, for continuous sweeping list repetitions, the desired (minimum or maximum) time must be multiplied by the quantity:

\[ (\text{points} - 1)/\text{points} \]

The default sweep time is 1 second for any number of points or frequencies specified.

The function in this program is a 1,000 point triangle wave. The maximum frequency allowed in the frequency list is the reference oscillator frequency divided by the quantity four multiplied by the number of waveform points:

\[ \frac{\text{Rosc}/(4 \times \text{npts})}{= 42.94967296E6 / (4 \times 1000) = 10.7374 \text{ kHz}} \]
Output Frequency and Sample Rate

The output frequency of an arbitrary waveform is defined as:

\[ F_O = \frac{\text{sample rate}}{\text{waveform_points}} \]

For frequency sweeps the sample rate(s) are the start and stop frequencies. For example, with an arbitrary waveform with 4096 amplitude points, a start frequency of 4.096E6 and a stop frequency of 8.192E6 produces a sweep from 1 kHz to 2 kHz.

For frequency-shift keying the sample rates are \textit{frequency1} and \textit{frequency2} specified by the \texttt{[SOURce:]FREQuency[1]:FSKey} command. The output frequencies are the sample rates divided by the number of amplitude points in the arbitrary waveform.

AC Leveling

The AFG has a 250 kHz output filter and a 10 MHz output filter. When the filter has been selected and enabled, AC output leveling maintains the amplitude at a constant level over the frequency sweep or frequency list. AC leveling, which is performed by the \texttt{CALibration:STATe:AC ON} command (reset setting), applies to the sine wave function only.

AC Leveling Amplitude Errors

When AC leveling is enabled during a sweep or frequency list, errors in the output amplitude still occur during a frequency change. In most cases, the errors are negligible. However, in applications where the step-to-step frequency changes are large (10% or greater), or when frequency changes occur near the filter’s cutoff frequency, the error is such that settling times on the order of milliseconds are required for the output to settle to the correct amplitude.

Table 4-1 shows typical (non-warranted) amplitude errors versus settling times when a frequency change occurs. These “worst case” settling times represent frequency changes (freq1 to freq2) of 10% and 1% from the filter’s cutoff frequency.
### Table 4-1. Amplitude Errors Versus Settling Times

<table>
<thead>
<tr>
<th>Error in Volts</th>
<th>Settling Time (ms)</th>
<th>% Error</th>
<th>dB Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.532</td>
<td>0.01</td>
<td>10.6</td>
<td>0.878</td>
</tr>
<tr>
<td>0.5</td>
<td>0.48</td>
<td>10.0</td>
<td>0.828</td>
</tr>
<tr>
<td>0.25</td>
<td>1.16</td>
<td>5.0</td>
<td>0.424</td>
</tr>
<tr>
<td>0.1</td>
<td>2.04</td>
<td>2.0</td>
<td>0.172</td>
</tr>
<tr>
<td>0.05</td>
<td>2.62</td>
<td>1.0</td>
<td>0.086</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error in Volts</th>
<th>Settling Time (ms)</th>
<th>% Error</th>
<th>dB Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.056</td>
<td>0.01</td>
<td>1.12</td>
<td>0.097</td>
</tr>
<tr>
<td>0.05</td>
<td>0.48</td>
<td>1.0</td>
<td>0.086</td>
</tr>
<tr>
<td>0.025</td>
<td>1.10</td>
<td>0.5</td>
<td>0.043</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error in Volts</th>
<th>Settling Time (ms)</th>
<th>% Error</th>
<th>dB Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.256</td>
<td>0.01</td>
<td>5.12</td>
<td>0.433</td>
</tr>
<tr>
<td>0.25</td>
<td>0.39</td>
<td>5.0</td>
<td>0.424</td>
</tr>
<tr>
<td>0.10</td>
<td>1.22</td>
<td>2.0</td>
<td>0.172</td>
</tr>
<tr>
<td>0.05</td>
<td>1.84</td>
<td>1.0</td>
<td>0.086</td>
</tr>
<tr>
<td>0.025</td>
<td>2.60</td>
<td>0.5</td>
<td>0.043</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error in Volts</th>
<th>Settling Time (ms)</th>
<th>% Error</th>
<th>dB Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.025</td>
<td>0.32</td>
<td>0.5</td>
<td>0.043</td>
</tr>
<tr>
<td>0.01</td>
<td>1.20</td>
<td>0.2</td>
<td>0.0174</td>
</tr>
</tbody>
</table>

### FSK Control Sources

The frequency-shift keying control sources are:

**EXTernal** – The HP E1445A AFG’s front panel “Stop Trig/FSK/Gate In” BNC connector (TTL levels).

**TTLTrg<n>** – The VXIbus TTL trigger lines TTLTrg0 through TTLTrg7.

Both the “FSK” BNC connector and the TTLTrg<n> trigger lines use TTL compatible signal levels. A “high” level on the BNC or trigger line selects *frequency1*, a “low” level selects *frequency2*.
Frequency-Shift Delay

Once the control signal to shift the frequency is received, there is a delay of 20 reference oscillator clock cycles before the frequency is active. This delay occurs with all reference oscillator sources.

Driving the TTLTrg<\textsubscript{n}> Trigger Lines

When driving the TTLTrg<\textsubscript{n}> trigger lines with the HP E1406A Command Module, note that the module uses negative-true logic. Thus, writing a logic ‘1’ to a trigger line sets the line to an electrically low level. This would select FSK<\textit{frequency2}. Writing a logic ‘0’ to a trigger line sets the line to an electrically high level which selects <\textit{frequency1}. Refer to your Command Module manual for more information on setting up and enabling the TTLTrg trigger lines.
This chapter shows you how to arm and trigger the HP E1445A AFG in order to start and advance standard and arbitrary waveforms. The sections of this chapter include:

- The ARM-TRIG Configuration ............................................. Page 164
  - The ARM-TRIG States .................................................. Page 164

- Initiating Waveforms ...................................................... Page 165
- Arming the AFG ............................................................... Page 165
  - Arming Commands ....................................................... Page 165
  - Setting Arming Sources ................................................ Page 166
  - Setting the Arm and Waveform Cycle Count ..................... Page 169

- Triggering the AFG ........................................................ Page 172
  - Triggering Commands .................................................. Page 172
  - Using the Divide-by-N Frequency Generator ..................... Page 174
  - Lock-Stepping Multiple AFGs ........................................ Page 176
  - Using Stop Triggers ..................................................... Page 180
  - Gating Trigger Signals ................................................ Page 183

- Arming and Triggering Frequency Sweeps and Lists ................. Page 186
  - Frequency Sweeps Using Triggers .................................. Page 186
  - Arming and Triggering a Frequency Sweep ....................... Page 190
  - Arming and Triggering a Frequency List .......................... Page 193

- Aborting Waveforms ....................................................... Page 196
  - Using ABORT, Stop Triggers, or Gating ............................ Page 196

- Arming and Triggering Program Comments .......................... Page 197
  - Reference Oscillator Sources ....................................... Page 197
  - AFG Frequency Synthesis Modes ................................. Page 197
  - AFG Frequency Modes ................................................ Page 198
  - AFG Arming Sources .................................................. Page 199
  - AFG Arm Count .......................................................... Page 199
  - Waveform Repetition Count ........................................ Page 199
  - Stop Trigger Sources ................................................ Page 199
  - External Stop Trigger Slope ........................................ Page 200
  - AFG Gating Sources .................................................... Page 200
  - AFG Gate Polarity ....................................................... Page 200
  - Enabling the Gate ...................................................... Page 200
  - Frequency Sweep/List Arming ....................................... Page 201
  - Frequency Sweep/List Advance Trigger ............................ Page 201
The ARM-TRIG Configuration

Each standard and arbitrary waveform is a series of discrete amplitude points (digital-to-analog (DAC) codes). The HP E1445A AFG uses an ARM-TRIG triggering configuration to output these points. When initiated, an arm signal enables the AFG to output one amplitude point each time a trigger signal is received. The “arm-trigger” model is shown in Figure 5-1.

The AFG operates within four states: Idle, Wait-for-Arm, Wait-for-Trigger, and Instrument Action (see Figure 5-1).

When power is applied or following a reset or an abort, the AFG is in the Idle state. The AFG is set to the Wait-for-Arm state with the \texttt{INITiate[:IMMediate]} command.

The AFG moves to the Wait-for-Trigger state when an arm from the specified arm source is received. The AFG moves to the Instrument Action state when a trigger is received.

After the Instrument Action (amplitude point is output) occurs, the AFG returns to the Wait-for-Trigger state until the next trigger occurs. When enough triggers have occurred such that the specified waveform cycle (repetition) count has been reached, the AFG returns to the Wait-for-Arm state until the next arm occurs. When the specified arm count has been reached, the AFG returns to the Idle state.
Initiating Waveforms

After the AFG has been configured to output the desired waveform, the AFG is set to the Wait-for-Arm state with the command:

\[\text{INITiate[:IMMediate]}\]

\text{INITiate} is an uncoupled command and is generally the last command executed before a waveform is output:

\begin{verbatim}
SUB Sine_wave
Sine_wave: !Subprogram which outputs a sine wave
    COM @Afg
    OUTPUT @Afg;"SOUR:FREQ1:FIX 1E3;"; !frequency
    OUTPUT @Afg;"SOUR:FUNC:SHAP SIN;"; !function
    OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5 V" !amplitude
    OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
SUBEND
\end{verbatim}

If \text{INITiate[:IMMediate]} is executed when the AFG is not in the Idle state, Error -213, "Init ignored" is generated.

Arming the AFG

In order for the AFG to accept trigger signals which output the amplitude points of the waveform, the AFG must be armed. The information in this section covers the commands and programming sequence used to arm the AFG for fixed frequency waveform generation.

Arming Commands

The commands which arm the AFG allow you to specify the following:

- the arm source
- the slope of an external arm signal
- the number of arms per \text{INITiate[:IMMediate]} command
- the number of waveform cycles per arm

The arming commands include:

\[\text{ARM}\]
\[[:\text{START} | \text{SEQUence}[1]]\]
\[[:\text{LAYer}[1]]\]
\[:\text{COUNT} <number>\]
\[:\text{LAYer2}\]
\[:\text{COUNT} <number>\]
\[[:\text{IMMediate}]\]
\[:\text{SLOPe} <edge>\]
\[:\text{SOURce} <source>\]
The arming commands for continuous waveforms are uncoupled commands. They are executed relative to other AFG commands in the sequence of Figure 5-2.

![Diagram showing the command sequence](image)

**Figure 5-2. AFG Arming Command Sequence**

**Note**

Detailed information on the commands introduced in this chapter can be found in Chapter 8, “Command Reference”. The commands in this chapter are shown in their entirety (optional headers included) to help you locate them in the reference.

### Setting Arming Sources

The EXT_ARM program shows how to select the source which arms the AFG. The program selects the AFG’s “Start Arm In” BNC connector as the arming source. When an arming signal is received, a 10 kHz, 1 Vpp square wave is output.

The steps of this program are:

1. **Select the FIXed frequency mode**
   
   
   ![Command](image)

2. **Set the output frequency**
   
   
   ![Command](image)

3. **Set the output function**
   
   
   ![Command](image)

4. **Set the signal amplitude**
   
   
   ![Command](image)
5. Set the arm source
   ARM[:START]:LAYer2:SOURce <source>

6. Set the trigger edge of the external trigger signal
   ARM[:START]:LAYer2:SLOPe <edge>

7. Place the AFG in the wait-for-arm state
   INITiate[:IMMediate]

HP BASIC Program Example (EXT_ARM)

```
1 !RE-STORE"EXT_ARM"
2 !This program arms the AFG with an external signal applied to the
3 !AFG Start Arm In port. When armed, a 10 kHz, 1 VPP square wave is
4 !output
5  
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40  
50 !Set up error checking
60 ON INTR 7 CALL Errmsg
70 ENABLE INTR 7;2
80 OUTPUT @Afg;"*CLS"
90 OUTPUT @Afg;"*SRE 32"
100 OUTPUT @Afg;"*ESE 60"
110  
120 !Call the subprograms
130 CALL Rst
140 CALL External_arm
150  
160 WAIT .1  !allow interrupt to be serviced
170 OFF INTR 7
180 END
190  
200 SUB External_arm
210 External_arm: !Subprogram which externally arms the AFG and outputs
220  
230     !a square wave.
240     COM @Afg
250     OUTPUT @Afg;"SOUR:FREQ1:MODE FIX;";  !frequency mode
260     OUTPUT @Afg;"SOUR:FREQ1:FIX 3;";  !frequency
270     OUTPUT @Afg;"SOUR:FUNC:SHAP SQU;";  !function
280     OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 1VPP" !amplitude
290     OUTPUT @Afg;"ARM:STAR:LAY2:SOUR EXT" !arm source
300     OUTPUT @Afg;"ARM:STAR:LAY2:SLOP NEG" !signal edge
310     OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
320  
330 SUB Rst
```

Continued on Next Page
340  Rst:  !Subprogram which resets the E1445.
350     COM @Afg
360     OUTPUT @Afg;"*RST;*OPC?"  !reset the AFG
370     ENTER @Afg;Complete
380     SUBEND
390     !
400     SUB Errmsg
410    Errmsg:  !Subprogram which displays E1445 programming errors
420     COM @Afg
430     DIM Message$[256]
440     !Read AFG status byte register and clear service request bit
450     B=SPOLL(@Afg)
460     !End of statement if error occurs among coupled commands
470     OUTPUT @Afg;"
480     OUTPUT @Afg;"ABORT"  !abort output waveform
490     REPEAT
500     OUTPUT @Afg;"SYST:ERR?"  !read AFG error queue
510     ENTER @Afg;Code,Message$
520     PRINT  Code,Message$
530     UNTIL Code=0
540     STOP
550     SUBEND

**Visual BASIC and Visual C/C++ Program Versions**

The Visual BASIC example program, EXT_ARM.FRM, is in directory "VBPROG" and the Visual C example program, EXT_ARM.C, is in directory "VCPROG" on the CD that came with your HP E1445A.
Setting the Arm and Waveform Cycle Count

The BURST program shows you how to set the number of arms the AFG is to receive before returning to the Idle state, and how to set the number of waveform cycles (repetitions) per arm.

The program sets a five cycle burst that occurs each time an external arm is received.

The steps of this program are:

1. **Set the output (burst) frequency**
   
   \[
   \text{[SOURce:]FREQuency[1]:CW | :FIXed]} <\text{frequency}> 
   \]

2. **Set the output function**
   
   \[
   \text{[SOURce:]FUNCtion[:SHAPe]} <\text{shape}> 
   \]

3. **Set the signal amplitude**
   
   \[
   \text{[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude]} <\text{amplitude}> 
   \]

4. **Set the arm source**
   
   \[
   \text{ARM[:START]:LAYer2:SOURce} <\text{source}> 
   \]

5. **Set the slope of the external arm signal**
   
   \[
   \text{ARM[:START]:LAYer2:SLOPe} <\text{edge}> 
   \]

6. **Set the arm count**
   
   \[
   \text{ARM[:START]:LAYer2:COUNt} <\text{number}> 
   \]

7. **Set the number of waveform cycles (burst count)**
   
   \[
   \text{ARM[:START][:LAYer[1]]:COUNt} <\text{number}> 
   \]

8. **Place the AFG in the wait-for-arm state**
   
   \[
   \text{INITiate[:IMMediate]} 
   \]
HP BASIC Program Example (BURST)

1 !RE-STORE"BURST"
2 !This program sets the arm count to infinity, and the cycle count
3 !count to 5. The arm source is set to external and a 1 kHz square
4 !wave is applied to the AFG's "Start Arm In" BNC connector. The
5 !AFG outputs a 5 cycle burst on each positive edge of the external
6 !arm signal.
7 !
8 !Assign I/O path between the computer and E1445A.
9 ASSIGN @Afg TO 70910
10 COM @Afg
11 !
12 !Set up error checking
13 ON INTR 7 CALL Errmsg
14 ENABLE INTR 7:2
15 OUTPUT @Afg;"*CLS"
16 OUTPUT @Afg;"*SRE 32"
17 OUTPUT @Afg;"*ESE 60"
18 !
19 !Call the subprograms
20 CALL Rst
21 CALL Burst_arm
22 !
23 WAIT .1 !allow interrupt to be serviced
24 OFF INTR 7
25 END
26 !
27 SUB Burst_arm
28 Burst_arm: !Subprogram which outputs a 5 cycle burst on each
29 !positive edge of an external arm signal.
30 COM @Afg
31 OUTPUT @Afg;"SOUR:FREQ1:FIX 10E3;"; !burst frequency
32 OUTPUT @Afg;";SOUR:FUNC:SHAP SIN;"; !function
33 OUTPUT @Afg;";SOUR:VOLT:LEV:IMM:AMPL 2.5VPP" !amplitude
34 OUTPUT @Afg;";ARM:STAR:LAY2:SOUR EXT" !arm source
35 OUTPUT @Afg;";ARM:STAR:LAY2:SLOP POS" !arm slope
36 OUTPUT @Afg;";ARM:STAR:LAY2:COUN INF" !arm count
37 OUTPUT @Afg;";ARM:STAR:LAY1:COUN 5" !cycle count
38 OUTPUT @Afg;";INIT:IMM" !wait-for-arm state
39 SUBEND
40 !
41 SUB Rst
42 Rst: !Subprogram which resets the E1445.
43 COM @Afg
44 OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
45 ENTER @Afg;Complete
46 SUBEND
47 !
48 SUB Errmsg

Continued on Next Page
420 Errmsg: !Subprogram which displays E1445 programming errors
430    COM @Afg
440    DIM Message$[256]
450    !Read AFG status byte register and clear service request bit
460    B=SPOLL(@Afg)
470    !End of statement if error occurs among coupled commands
480    OUTPUT @Afg;"
490    OUTPUT @Afg;"ABORT"                   !abort output waveform
500    REPEAT
510    OUTPUT @Afg;"SYST:ERR?"            !read AFG error queue
520    ENTER @Afg;Code,Message$
530    PRINT   Code,Message$
540    UNTIL Code=0
550    STOP
560    SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, BURST.FRM, is in directory “VBPROG” and the Visual C example program, BURST.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Triggering the AFG

Arming the AFG places the device in the Wait-for-Trigger state (Figure 5-1). When a trigger occurs, the AFG digital-to-analog converter (DAC) outputs one waveform amplitude point.

The information in this section covers the commands and programming sequence used to trigger the AFG when outputting fixed frequency waveforms.

Triggering Commands

The commands which trigger the AFG allow you to specify the following:

- the (start) trigger source
- the slope of an external (start) trigger signal
- the stop trigger source
- the slope of an external stop trigger signal
- the sample gating source
- the polarity of an external gating signal
- to enable sample gating

The triggering commands include:

TRIGger
[:STARt|SEQuence[1]]
:COUNT <number>
 :GATE
 :POLarity <polarity>
 :SOURce <source>
 :STATE <state>
 [:IMMediate]
 :SLOPe <edge>
 :SOURce <source>

:STOP|SEQuence2
 [:IMMediate]
 :SLOPe <edge>
 :SOURce <source>

Note

The trigger count (TRIGger[:STARt]:COUNT) is always equal to the number of amplitude points in the current waveform, multiplied by the number of waveform cycles. This value is not programmable (other than 9.91E37), but is included for SCPI compatibility purposes only.
The commands in the TRIGger subsystem are frequency coupled. They are executed relative to other AFG commands in the sequence shown in Figure 5-3.

Figure 5-3. AFG Triggering Command Sequence
Using the Divide-by-N Frequency Generator

The DIV_N program shows how to set the AFG trigger source. The program selects the AFG’s divide-by-N frequency generator ([SOURce]:FREQuency2 subsystem). This generator is recommended for use with the AFG’s 40 MHz reference oscillator (also selected in the program) to produce exact frequencies such as 10 MHz, 20 MHz, etc..

The steps of this program are:

1. **Select the 40 MHz reference oscillator**
   
   [SOURce]:ROSCillator:SOURce <source>

2. **Select the divide-by-N time base**
   
   TRIGger[:STARt]:SOURce <source>

3. **Set the output frequency**
   
   [SOURce]:FREQuency2[:CW | :FIXed] <frequency>

4. **Set the output function**
   
   [SOURce]:FUNCtion[:SHAPe] <shape>

5. **Set the signal amplitude**
   
   [SOURce]:VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>

6. **Place the AFG in the wait-for-arm state**
   
   INITiate[:IMMediate]

**HP BASIC Program Example (DIV_N)**

1 !RE-STORE"DIV_N"
2 !This program selects the 40 MHz reference oscillator and the
3 !SOURce:FREQuency2 subsystem (divide-by-N frequency synthesis) to
4 !generate an “exact” square wave frequency of 10 MHz.
5 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !
50 !Set up error checking
60 ON INTR 7 CALL Errmsg
70 ENABLE INTR 7:2
80 OUTPUT @Afg;"**CLS"
90 OUTPUT @Afg;"**SRE 32"
100 OUTPUT @Afg;"**ESE 60"
110 !
120 !Call the subprograms which reset the AFG and which
130 !generate the square wave.
140 CALL Rst
150 CALL Squ_wave

Continued on Next Page
Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, DIV_N.FRM, is in directory “VBPROG” and the Visual C example program, DIV_N.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
The LOCKSTEP program configures two AFGs such that they share the same trigger source. A “master” AFG is programmed to output its triggers on ECLTrg trigger line 0. The trigger source of a “servant” AFG is set to ECLTrg 0. Thus, both AFGs output waveforms (sin(x)/x) at the same frequency, and changing the frequency of the master changes the frequency of the servant simultaneously.

The programming sequence for lock-stepping multiple AFGs is given below:

1. Set the reference oscillator source for the master and slave as desired or use the default source
   [SOURce:]ROSCillator:SOURce <source>

2. Set the trigger source of the master AFG as desired. Set the trigger source of the slave AFG to the source driven by the master
   TRIGger[:STARt]:SOURce <source>

3. Set the frequency mode
   [SOURce:]FREQuency[1]:MODE <mode>

4. Set the output frequency of the master
   [SOURce:]FREQuency[1]:CW [:FIXed] <frequency>
      or
   [SOURce:]FREQuency2[:CW [:FIXed] <frequency>

5. Set the output function
   [SOURce:]FUNCTION[:SHAPe] <shape>

6. Set the signal amplitude
   [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>
7. **Feed the master AFG trigger to the trigger source of the slave AFG**
   
   \[ \text{[SOURce:]MARKer:ECLTrg<n>:FEED <source>} \]

8. **Enable the routing of the trigger signal**
   
   \[ \text{[SOURce:]MARKer:ECLTrg<n>:STATe <mode>} \]

9. **Place the AFG in the wait-for-arm state**
   
   \[ \text{INITiate[:IMMediate]} \]

**HP BASIC Program Example (LOCKSTEP)**

```
1 !RE-STORE"LOCKSTEP"
2 !This program "lock-steps" two AFGs. The trigger source which
3 !advances the waveform of the master AFG (DDS time base) is shared by
4 !the slave AFG. Thus, changing the frequency of the master AFG
5 !changes the frequency of the slave AFG simultaneously.
6 !
10 !Assign I/O path between the computer and E1445A, dimension an array
20 !variable for the sin(x)/x waveform amplitude points.
30 ASSIGN @Afg_m TO 70910
40 ASSIGN @Afg_s TO 70911
50 COM @Afg_m,@Afg_s,REAL Waveform(1:4096)
60 !
70 !Set up error checking
80 ON INTR 7 CALL Ermsg
90 ENABLE INTR 7:2
100 OUTPUT @Afg_m;"*CLS"
110 OUTPUT @Afg_m;"*SRE 32"
120 OUTPUT @Afg_m;"*ESE 60"
130 !
140 OUTPUT @Afg_s;"*CLS"
150 OUTPUT @Afg_s;"*SRE 32"
160 OUTPUT @Afg_s;"*ESE 60"
170 !
180 !Call the subprograms which reset the AFGs, delete all
190 !existing waveform segments and sequences, and which set up the
200 !AFGs to output arbitrary waveforms.
210 CALL Rst
220 CALL Wt_del
230 CALL Sinx_def
240 CALL Sinx_m
250 CALL Sinx_s
260 !
270 !Select the waveform sequence of the master AFG, place the
280 !master AFG in the Wait-for-arm state.
290 OUTPUT @Afg_m;"SOUR:FUNC:USER_SINX_M"
300 OUTPUT @Afg_m;"INIT:IMM"
310 !
320 WAIT .1 !allow interrupt to be serviced
330 OFF INTR 7
```

*Continued on Next Page*
Sub Sinx_def

Subprogram which computes the sin(x)/x waveform amplitudes used by both AFGs.

COM @Afg_m,@Afg_s,Waveform(*)

FOR I=-2047 TO 2048
  IF I=0 THEN I=1.E-38
  Waveform(I+2048)=((SIN(2*PI*.53125*I/256))/(.53125*I/256)*.159154943092)
NEXT I

Subend

Sub Sinx_m

COM @Afg_m,@Afg_s,Waveform(*)

Set the reference oscillator source, trigger source, frequency mode/frequency, and amplitude for the master AFG waveform. Feed the master trigger source to the slave AFG via ECL trigger line ECLT0.

OUTPUT @Afg_m;"SOUR:ROSC:SOUR INT1;"
OUTPUT @Afg_m;":TRIG:STAR:SOUR INT1;"
OUTPUT @Afg_m;":SOUR:FREQ1:MODE FIXED;"
OUTPUT @Afg_m;":SOUR:FREQ1:FIX 4.096E6;"
OUTPUT @Afg_m;":SOUR:FUNC:SHAP USER;"
OUTPUT @Afg_m;":SOUR:VOLT:LEV:IMM:AMPL 1.1V"
OUTPUT @Afg_m;"SOUR:MARK:ECLT0:FEED "TRIG:STAR"
OUTPUT @Afg_m;"SOUR:MARK:ECLT0:STAT ON"

Define the waveform segment and download the amplitude points. Define the output waveform sequence.

OUTPUT @Afg_m;"SOUR:LIST1:SEGM:SEL SIN_X" !select segment
OUTPUT @Afg_m;"SOUR:LIST1:SEGM:DEF 4096" !reserve memory
OUTPUT @Afg_m;"SOUR:LIST1:SEGM:VOLT";Waveform(*) !load points

Define the output waveform sequence. Select the sequence for output and place the slave AFG in the Wait-for-arm state.

OUTPUT @Afg_m;"SOUR:LIST1:SSEQ:SEL SINX_M" !select sequence
OUTPUT @Afg_m;"SOUR:LIST1:SSEQ:DEF 1" !specify segments
OUTPUT @Afg_m;"SOUR:LIST1:SSEQ:SEQ SIN_X" !segment order

Subend

Sub Sinx_s

COM @Afg_m,@Afg_s,Waveform(*)

Set the trigger source, frequency mode, function, and amplitude for the slave AFG waveform.

OUTPUT @Afg_s;"TRIG:STAR:SOUR ECLT0;"
OUTPUT @Afg_s;":SOUR:FREQ1:MODE FIXED;"
OUTPUT @Afg_s;":SOUR:FUNC:SHAP USER;"
OUTPUT @Afg_s;":SOUR:VOLT:LEV:IMM:AMPL 1.1V"

Define the waveform segment and download the amplitude points. Define the output waveform sequence. Select the sequence for output and place the slave AFG in the Wait-for-arm state.

Continued on Next Page
Continued on Next Page
Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, LOCKSTEP.FRM, is in directory “VBPROG” and the Visual C example program, LOCKSTEP.C, is in directory “VCPROG” on the CD that came with your HP E1445A.

Using Stop Triggers

The STOPTRIG program shows you how to use stop triggers to abort the remaining cycles of a cycle count (ARM[:STARt][:LAYer][1]:COUNt). The program sets up five, 5,000 cycle bursts. Stop triggers are used to abort the burst before all of the 5,000 cycles occurs. An external arm is used to start the bursts. A stop trigger aborts a burst at the end of the current cycle.

The steps of the program are:

1. Set the (start) trigger source
   TRIGger[:STARt]:SOURce <source>

2. Set the stop trigger source
   TRIGger:STOP:SOURce <source>

3. Set the (external) stop trigger slope
   TRIGger:STOP:SLOPe <edge>

4. Set the output frequency
   [SOURce:]FREQuency[1]:CW | :FIXed <frequency>

5. Set the output function
   [SOURce:]FUNCTION[:SHAPe] <shape>

6. Set the signal amplitude
   [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>

7. Set the arm source
   ARM[:STARt]:LAYer2:SOURce <source>

8. Set the (external) arm slope
   ARM[:STARt]:LAYer2:SLOPe <edge>

9. Set the arm count
   ARM[:STARt]:LAYer2:COUNt <number>
10. Set the number of waveform cycles
   ARM[:START][:LAYer[1]]:COUNt <number>

11. Place the AFG in the wait-for-arm state
   INITiate[:IMMediate]

HP BASIC Program Example (STOPTRIG)

```basic
1   !RE-STORE"STOPTRIG"
2   !This program sets the arm count to 5 and the repetition count to
3   5,000. A stop trigger applied to the "Stop Trig" BNC connector
4   aborts the remaining cycles of the current burst. An arm signal
5   applied to the "Start Arm In" BNC re-arms the AFG which then
6   outputs the next burst.
7   !
8   Assign I/O path between the computer and E1445A.
9   ASSIGN @Afg TO 70910
10  COM @Afg
11   !
12  !Set up error checking
13  ON INTR 7 CALL Ermsg
14  ENABLE INTR 7,2
15  OUTPUT @Afg;"CLS"
16  OUTPUT @Afg;"SRE 32"
17  OUTPUT @Afg;"ESE 60"
18  !
19  !Call the subprograms
20  CALL Rst
21  CALL Stop_trig
22  WAIT .1 !allow interrupt to be serviced
23  OFF INTR 7
24  END
25  !
26  SUB Stop_trig
27  Stop_trig: !Subprogram which sets up the AFG to output 5, 5,000
28   cycle bursts. Sets the trigger source to INternal1,
29   the stop trigger source to EXternal, the stop trigger
30   slope to NEGative. The arm source is also set to
31   EXternal.
32  COM @Afg
33  OUTPUT @Afg;"TRIG:STAR:SOUR INT1;"; !trigger source
34  OUTPUT @Afg;" :TRIG:STOP:SOUR EXT;"; !stop trigger source
35  OUTPUT @Afg;" :TRIG:STOP:SLOP NEG;"; !stop trigger slope
36  OUTPUT @Afg;" :SOUR:FREQ1:FIX 100;"; !output frequency
37  OUTPUT @Afg;" :SOUR:FUNC:SHAP SIN;"; !output function
38  OUTPUT @Afg;" :SOUR:VOLT:LEV:IMM:AMPL 5VPP" !amplitude
39  OUTPUT @Afg;"ARM:STAR:LAY2:SOUR EXT" !arm source
```

Continued on Next Page
330 OUTPUT @Afg;" ARM:STAR:LAY2:SLOP POS" !arm slope
340 OUTPUT @Afg;" ARM:STAR:LAY2:COUN 5" !arm count
350 OUTPUT @Afg;" ARM:STAR:LAY1:COUN 5E3" !repetition count
360 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
370 SUBEND
380 !
390 SUB Rst
400 Rst: !Subprogram which resets the E1445.
410 COM @Afg
420 OUTPUT @Afg;" *RST;*OPC?" !reset the AFG
430 ENTER @Afg;Complete
440 SUBEND
450 !
460 SUB Errmsg
470 Errmsg: !Subprogram which displays E1445 programming errors
480 COM @Afg
490 DIM Message$[256]
500 !Read AFG status byte register and clear service request bit
510 B=SPOLL(@Afg)
520 !End of statement if error occurs among coupled commands
530 OUTPUT @Afg;""
540 OUTPUT @Afg;"ABORT" !abort output waveform
550 REPEAT
560 OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
570 ENTER @Afg;Code,Message$
580 PRINT Code,Message$
590 UNTIL Code=0
600 STOP
610 SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, STOPTRIG.FRM, is in directory “VBPROG” and the Visual C example program, STOPTRIG.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Gating Trigger Signals

Gating is the process of suspending the output waveform. When the gate is active, AFG triggering is suspended. The output remains at the last amplitude point triggered. When the gate is inactive, the output resumes with the next amplitude point.

The gating commands are frequency coupled and are executed relative to other AFG commands as shown in Figure 5-3.

The GATE example shows how to use the AFG’s “Gate In” BNC to suspend AFG triggering and thus, generation of the output signal. A high (TTL levels) on the BNC activates the gate.

The steps of program are as follows:

1. Set the reference oscillator source
   [SOURce:]ROSCillator:SOURce <source>

2. Set the (start) trigger source
   TRIGger[:START]:SOURce <source>

3. Set the trigger gating source
   TRIGger[:START]:GATE:SOURce <source>

4. Set the gating signal polarity
   TRIGger[:START]:GATE:POLarity <polarity>

5. Enable trigger gating
   TRIGger[:START]:GATE:STATe <mode>

6. Set the output frequency
   [SOURce:]FREQuency[1][:CW | :FIXed] <frequency>

7. Set the output function
   [SOURce:]FUNCtion[:SHAPE] <shape>

8. Set the number of waveform points
   [SOURce:]RAMP:POINts <number>

9. Set the signal amplitude
   [SOURce:]VOLTage[:LEVEL][:IMMediate][:AMPLitude] <amplitude>

10. Place the AFG in the wait-for-arm state
    INITiate[:IMMediate]
HP BASIC Program Example (GATE)

1 !RE-STORE"GATE"
2 !The following program gates the output of a 40 point triangle
3 !wave whose frequency is 1 MHz. When the signal on the “Gate In” BNC
4 !is high, the gate is active and the output is suspended at the last
5 !amplitude point triggered. When the signal is low, the gate is inactive
6 !and the waveform resumes.
7 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !
50 !Set up error checking
60 ON INTR 7 CALL Ermsg
70 ENABLE INTR 7;2
80 OUTPUT @Afg;"*CLS"
90 OUTPUT @Afg;"*SRE 32"
100 OUTPUT @Afg;"*ESE 60"
110 !
120 !Call the subprograms
130 CALL Rst
140 CALL Tri_wave
150 !
160 WAIT .1 !allow interrupt to be serviced
170 OFF INTR 7
180 END
190 !
200 SUB Tri_wave
210 Tri_wave: !Subprogram which outputs a triangle wave
220 COM @Afg
230 OUTPUT @Afg;"SOUR:ROSC:SOUR INT2;"; !reference oscillator
240 OUTPUT @Afg;":TRIG:STAR:SOUR INT2;"; !frequency2 generator
250 OUTPUT @Afg;":TRIG:STAR:GATE:SOUR EXT;"; !gate source
260 OUTPUT @Afg;":TRIG:STAR:GATE:POL NORM;"; !gate polarity
270 OUTPUT @Afg;":TRIG:STAR:GATE:STAT ON;"; !enable gate
280 OUTPUT @Afg;":SOUR:FREQ2:FIX 1E6;"; !frequency
290 OUTPUT @Afg;":SOUR:FUNC:SHAP TRI;"; !function
300 OUTPUT @Afg;":SOUR:RAMP:POIN 40;"; !waveform points
310 OUTPUT @Afg;":SOUR:VOLT:LEV:IMM:AMPL 5V" !amplitude
320 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
330 SUBEND
340 !
350 SUB Rst
360 Rst: !Subprogram which resets the E1445.
370 COM @Afg
380 OUTPUT @Afg;"*RST;*OPC?")!reset the AFG
390 ENTER @Afg;Complete
400 SUBEND
410 !

Continued on Next Page
420    SUB Errmsg
430   Errmsg:  !Subprogram which displays E1445 programming errors
440   COM @Afg
450   DIM Message$[256]
460   !Read AFG status byte register and clear service request bit
470   B=SPOLL(@Afg)
480   !End of statement if error occurs among coupled commands
490   OUTPUT @Afg;""
500   OUTPUT @Afg;"ABORT"                      !abort output waveform
510   REPEAT
520     OUTPUT @Afg;"SYST:ERR?"                  !read AFG error queue
530     ENTER @Afg;Code,Message$
540     PRINT  Code,Message$
550     UNTIL Code=0
560   STOP
570    SUBEND

Visual BASIC and  The Visual BASIC example program, GATE.FRM, is in directory
Visual C/C++ Program "VBPROG" and the Visual C example program, GATE.C, is in directory
Versions      "VCPROG" on the CD that came with your HP E1445A.
Arming and Triggering Frequency Sweeps and Lists

Frequency sweeps and lists are started and advanced using the arm and trigger signals described in this section. The commands used to set up the arming and triggering of sweeps and lists are:

**ARM**

```
:SWEep|SEQuence3
  :COUNt <number>
  [:IMMediate]
  :LINK <link>
  :SOURce <source>
```

**TRIGger**

```
:SWEep|SEQuence3
  [:IMMediate]
  :LINK <link>
  :SOURce <source>
  :TIMer <period>
```

The frequency sweep and frequency list arming and triggering commands are frequency coupled. Thus, they are executed in the sequence shown in the flowchart in Figure 4-1 on page 118.

### Frequency Sweeps Using Triggers

The AFG can output frequency sweeps each time it is triggered. However, the maximum sweep time and frequency steps depend on the number of waveform repetitions and the average sweep frequency.

To determine the maximum sweep time ([SOURce]:SWEep:TIMe <number>), divide the number of waveform repetitions to be output (i.e., maximum is 66536) by the average frequency. For example:

- STARt frequency: 1 kHz
- STOP frequency: 1 MHz

then

\[
\text{sweep time} = 66536 / ((1000 + 1000000) / 2) = 66536 / 500,500 = 0.1329
\]

To determine the maximum number of frequency steps (or points) ([SOURce]:SWEep:POINts <number>), divide the sweep time by the minimum time between frequency points (i.e., 0.00125 S). For example, using the above calculated time:

\[
0.1329 / 0.00125 = 106 \text{ points}
\]

The SWP TRIG program shows how to output a sweep using a user selected trigger mode.
The steps of this program are:

1. **Set the sweep mode**
   
   `[SOURce:]FREQuency[1]:MODE SWEep

2. **Set the start frequency**
   
   `[SOURce:]FREQuency[1]:STARt <start_freq>

3. **Set the stop frequency**
   
   `[SOURce:]FREQuency[1]:STOP <stop_freq>

4. **Set the number of sweeps**
   
   `[SOURce:]SWEep:COUNT INFinity

5. **Set the number of points in a sweep**
   
   `[SOURce:]SWEep:POINts <number>

6. **Set the sweep time**
   
   `[SOURce:]SWEep:TIME <number>

7. **Select the source to start a sweep**
   
   ARM:SWEep:SOURce LINK

8. **Set the number of waveform repetitions**
   
   ARM[:START]:LAYer[1]:COUNT <number>

9. **Set the number of waveform arm starts**
   
   ARM[:START]:LAYer2:COUNT <number>

10. **Select the source to start waveform output**
    
    ARM[:START]:LAYer2:SOURce <source>

11. **Set the output function**
    
    `[SOURce:]FUNCtion[:SHApe] <shape>

12. **Set the signal amplitude**
    
    `[SOURce:]VOLTage[:LEVEL][:IMMediate][:AMPLitude] <amplitude>

13. **Place the AFG in the wait-for-arm state**
    
    INITiate[:IMMediate]

14. **Trigger the AFG to start a sweep**
    
    use the source selected above in Step 10
HP BASIC Program Example (SWP_TRIG)

1   !RE-STORE"SWP_TRIG"
2   !This program triggers a sweep using the Group Execute
3   !Trigger command. The sweep is from 1 kHz to 1 MHz.
4   !
10   !Assign I/O path between the computer and E1445A.
20   ASSIGN @Afg TO 80910.
30   COM @Afg
40   !
50   !Set up error checking
60   ON INTR 8 CALL Errmsg
70   ENABLE INTR 8;2
80   OUTPUT @Afg;"*CLS"
90   OUTPUT @Afg;"*SRE 32"
100  OUTPUT @Afg;"*ESE 60"
110  !
120   !Call the subprograms
130   CALL Rst
140   CALL Swp_trig
150   !
160   WAIT .1 !allow interrupt to be serviced
170   OFF INTR 8
180   END
190   !
200   SUB Swp_trig
210   Swp_trig: !Subprogram that triggers a sweep
220   COM @Afg
230   OUTPUT @Afg;"SOUR:FREQ1:MODE SWE;"; !Sweep mode
240   OUTPUT @Afg;" :SOUR:FREQ1:STAR 1E3;"; !start frequency
250   OUTPUT @Afg;" :SOUR:FREQ1:STOP 1E6;"; !stop frequency
260   OUTPUT @Afg;" :SOUR:SWE:COUN INF;"; !repetition count
270   OUTPUT @Afg;" :SOUR:SWE:POIN 100;"; !frequency points
280   OUTPUT @Afg;" :SOUR:SWE:TIME .13" !sweep time
290   OUTPUT @Afg;":ARM:SWE:SOUR LINK" !trigger mode
300   OUTPUT @Afg;":ARM:STAR:LAY1:COUN 65536" !waveform repetitions
310   OUTPUT @Afg;":ARM:STAR:LAY2:COUN INF" !waveform starts
320   OUTPUT @Afg;":ARM:STAR:LAY2:SOUR BUS" !trigger source
330   OUTPUT @Afg;":SOUR:FUNC:SHAP SIN;"; !function
340   OUTPUT @Afg;" :SOUR:VOLT:LEV:IMM:AMPL 5 V" !amplitude
350   OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
360   CALL Step
370   SUBEND
380   !
390   SUB Step
400   Step: !Subprogram which starts sweep
410   COM @Afg
420   DISP "Press 'Continue' when ready to start a sweep"
430   PAUSE
440   TRIGGER @Afg !trigger AFG

Continued on Next Page
FOR I=1 TO 10
    DISP "Wait until sweep completes, then press 'Continue' to start a new sweep"
    PAUSE
    TRIGGER @Afg !trigger AFG
NEXT I
DISP ""
SUBEND
!
SUB Rst
Rst: !Subprogram which resets the E1445.
    COM @Afg
    OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
    ENTER @Afg;Complete
SUBEND
!
SUB Errmsg
Errmsg: !Subprogram which displays E1445 programming errors
    COM @Afg
    DIM Message$[256]
    !Read AFG status byte register and clear service request bit
    B=SPOLL(@Afg)
    !End of statement if error occurs among coupled commands
    OUTPUT @Afg;"
    OUTPUT @Afg;"ABORT" !abort output waveform
    REPEAT
    OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
    ENTER @Afg;Code,Message$
    PRINT Code,Message$
    UNTIL Code=0
STOP
SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, SWP_TRIG.FRM, is in directory “VBPROG” and the Visual C example program, SWP_TRIG.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Arming and Triggering a Frequency Sweep

The SWP_STEP program shows you how to control the start and advancing of a frequency sweep. The program sets the sweep arm and sweep trigger sources to HOLD. The AFG is armed, and advances to the next frequency in the sweep using the ARM:SWEep[:IMMediate] and TRIGger:SWEep[:IMMediate] commands respectively.

Using the flowchart in Figure 4-1 on page 118 as a guide, the steps of the program are:

1. **Select the frequency generator that allows frequency sweeping**
   \[\text{TRIGger[:STARt]:SOURce} \enspace <\text{source}>\]

2. **Select the frequency sweep mode**
   \[\text{[SOURce:]FREQuency[1]:MODE} \enspace <\text{mode}>\]

3. **Set the start frequency**
   \[\text{[SOURce:]FREQuency[1]:STARt} \enspace <\text{start_freq}>\]

4. **Set the stop frequency**
   \[\text{[SOURce:]FREQuency[1]:STOP} \enspace <\text{stop_freq}>\]

5. **Set the number of points (frequencies) in the frequency sweep**
   \[\text{[SOURce:]SWEep:POINts} \enspace <\text{number}>\]

6. **Set the source which starts the frequency sweep**
   \[\text{ARM:SWEep:SOURce} \enspace <\text{source}>\]

7. **Set the source which advances the sweep to the next frequency**
   \[\text{TRIGger:SWEep:SOURce} \enspace <\text{source}>\]

8. **Set the output function**
   \[\text{[SOURce:]FUNCtion[:SHAPe]} \enspace <\text{shape}>\]

9. **Set the signal amplitude**
   \[\text{[SOURce:]VOLTage[:LEVEL][:IMMediate][:AMPLitude]} \enspace <\text{amplitude}>\]

10. **Place the AFG in the wait-for-arm state**
    \[\text{INITiate[:IMMediate]}\]
HP BASIC Program Example (SWP_STEP)

1 !RE-STORE "SWP_STEP"
2 !This program sets the AFG arm source and trigger source to
3 !HOLD. The AFG is armed and advanced through the sweep points
4 !using “arm immediate” and “trigger immediate” commands.
5 !
6 !Assign I/O path between the computer and E1445A.
7 ASSIGN @Afg TO 70910
8 COM @Afg, Pts
9 !
10 !Set up error checking
11 ON INTR 7 CALL Errmsg
12 ENABLE INTR 7:2
13 OUTPUT @Afg; "*CLS"
14 OUTPUT @Afg; "*SRE 32"
15 OUTPUT @Afg; "*ESE 60"
16 !
17 !Call the subprograms
18 CALL Rst
19 CALL Swp_step
20 !
21 WAIT .1 !allow interrupt to be serviced
22 OFF INTR 7
23 END
24 !
25 SUB Swp_step
26 !Subprogram which sets up a sweep from 1 kHz to 10 kHz
27 !which is armed and advanced on “IMMediate” command.
28 COM @Afg, Pts
29 OUTPUT @Afg; "TRIG:STAR:SOUR INT1;"
30 OUTPUT @Afg; "SOUR:FREQ1:MODE SWE;"
31 OUTPUT @Afg; "SOUR:FREQ1:STAR 1E3;"
32 OUTPUT @Afg; "SOUR:FREQ1:STOP 10E3;"
33 OUTPUT @Afg; "ARM:SWE:SOUR HOLD;"
34 OUTPUT @Afg; "SOUR:FUNC:SHAP SIN;"
35 OUTPUT @Afg; "VOLT:LEV:IMM:AMPL 5 V"
36 OUTPUT @Afg; "INIT:IMM" !wait-for-arm state
37 CALL Step
38 SUBEND
39 !
40 !
41 Step: !Subprogram which starts and advances sweep
42 COM @Afg, Pts
43 DISP "Press 'Continue' to arm trigger system"
44 PAUSE
45 OUTPUT @Afg; "ARM:SWE:IMM" !start sweep (sweep does not advance)
46 OUTPUT @Afg; "SOUR:SWE:POIN?" !query number of waveform points

Continued on Next Page
440 ENTER @Afg;Pts
450 FOR I=1 TO (Pts-1)
460 DISP "Press 'Continue' to advance to next frequency"
470 PAUSE
480 OUTPUT @Afg;"TRIG:SWE:IMM" !step to next frequency
490 NEXT I
500 DISP ""
510 SUBEND
520 !
530 SUB Rst
540 Rst: !Subprogram which resets the E1445.
550 COM @Afg,Pts
560 OUTPUT @Afg; "*RST;*OPC?" !reset the AFG
570 ENTER @Afg;Complete
580 SUBEND
590 !
600 SUB Errmsg
610 Errmsg: !Subprogram which displays E1445 programming errors
620 COM @Afg,Pts
630 DIM Message$[256]
640 !Read AFG status byte register and clear service request bit
650 B=SPOLL(@Afg)
660 !End of statement if error occurs among coupled commands
670 OUTPUT @Afg;"
680 OUTPUT @Afg;"ABORT" !aborts output waveform
690 REPEAT
700 OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
710 ENTER @Afg;Code,Message$
720 PRINT Code,Message$
730 UNTIL Code=0
740 STOP
750 SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, SWP_STEP.FRM, is in directory “VBPROG” and the Visual C example program, SWP_STEP.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Frequency lists are started and advanced using the same arming and triggering commands used for sweeps. The LIST_STP program sets the arm and list advance sources to BUS. Thus, the AFG is armed and advanced through the frequency list using the HP-IB group execute trigger command TRIGGER 7.

Using the flowchart in Figure 4-1 on page 118 as a guide, the steps of this program are:

1. **Select the frequency generator which allows frequency lists (hopping)**
   
   TRIGger[:START]:SOURce <source>

2. **Select the frequency list mode**
   
   [SOURce]:FREQuency[1]:MODE <mode>

3. **Set the frequency list**
   
   [SOURce]:LIST[2]:FREQuency <freq_list>

4. **Set the source which starts the frequency list**
   
   ARM:SWEep:SOURce <source>

5. **Set the source which advances the list to the next frequency**
   
   TRIGger:SWEep:SOURce <source>

6. **Set the output function**
   
   [SOURce]:FUNCTION[:SHAPe] <shape>

7. **Set the signal amplitude**
   
   [SOURce]:VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>

8. **Place the AFG in the wait-for-arm state**
   
   INITiate[:IMMediate]
HP BASIC Program Example (LIST_STP)

1  !RE-STORE“LIST_STP”
2  !The following program configures the AFG to step through a
3  !frequency list when an HP-IB group execute trigger is received.
4  !
5  !Assign I/O path between the computer and E1445A.
20  ASSIGN @Afg TO 70910
30  COM @Afg,Pts
40  !
50  !Set up error checking
60  ON INTR 7 CALL Errmsg
70  ENABLE INTR 7;:2
80  OUTPUT @Afg;“CLS”
90  OUTPUT @Afg;“SRE 32”
100  OUTPUT @Afg;“ESE 60”
110  !
120  !Call the subprograms
130  CALL Rst
140  CALL List_stp
150  !
160  WAIT .1 !allow interrupt to be serviced
170  OFF INTR 7
180  END
190  !
200  SUB List_stp
210  List_stp: !Subprogram which sets up a frequency list which is armed
220  !and advanced with HP-IB group execute triggers (TRIGGER 7).
230  COM @Afg,Pts
240  OUTPUT @Afg;“TRIG:STAR:SOUR INT1;”;
250  OUTPUT @Afg;“:SOUR:FREQ1:MODE LIST;”;
260  OUTPUT @Afg;“:SOUR:LIST2:FREQ 10E3,20E3,30E3,40E3,50E3;”;
270  OUTPUT @Afg;“:ARM:SWE:SOUR BUS;”;
280  OUTPUT @Afg;“:TRIG:SWE:SOUR BUS;”;
290  OUTPUT @Afg;“:SOUR:FUNC:SHAP SQU;”;
300  OUTPUT @Afg;“:SOUR:VOLT:LEV:IMM:AMPL 1 V”
310  OUTPUT @Afg;“INIT:IMM”
320  WAIT .1 !wait in case of error
330  CALL Step
340  SUBEND
350  !
360  SUB Step
370  Step: !Subprogram which starts and advances frequency list
380  COM @Afg,Pts
390  DISP "Press ‘Continue’ to arm trigger system"
400  PAUSE
410  TRIGGER 7
420  FOR I=1 TO 4
430  DISP “Press ‘Continue’ to advance to next frequency”
440  PAUSE

Continued on Next Page
TRIGGER 7 !advance to next frequency
NEXT I
""
SUBEND

! Subprogram which resets the E1445.
COM @Afg,Pts
OUTPUT @Afg; "*RST;*OPC? !reset the AFG
ENTER @Afg;Complete
SUBEND

! Subprogram which displays E1445 programming errors
COM @Afg,Pts
DIM Message$[256]
!Read AFG status byte register and clear service request bit
B=SPOLL(@Afg)
!End of statement if error occurs among coupled commands
OUTPUT @Afg;"
OUTPUT @Afg;"ABORT" !abort output waveform
REPEAT
OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
ENTER @Afg;Code,Message$
PRINT Code,Message$
UNTIL Code=0
STOP
SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, LIST_STP.FRM, is in directory “VBPROG” and the Visual C example program, LIST_STP.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Abort Waveforms

Abort a waveform places the AFG in the Idle state (Figure 5-4). The waveform is halted and the output remains at the last amplitude point triggered when the abort was executed. The command which aborts a waveform is:

`ABORt`

Using ABORt, Stop Triggers, or Gating

Figure 5-4 compares the effects of aborting a waveform, or using stop triggers or gating to stop or suspend the output.

```
ARM:STAR:LAY2:COUNT 3
ARM:STAR:LAY1:COUNT 3 | (START ARM)
(3 ARMS, 3 CYCLES PER ARM)
```

![Waveform diagrams showing effects of ABORt, Stop Triggers, and Gating]

Figure 5-4. Effects of ABORt, Stop Triggers, and Gating
Arming and Triggering Program Comments

The following information is associated with arming and triggering the AFG. Included are details on the operation of the AFG’s arming and triggering functions, and on the various modes, ranges, etc., used in the programs in this chapter.

Reference Oscillator Sources

There are five reference oscillator sources for the AFG which are selected by the \texttt{[SOURce:]ROSCillator:SOURce} command:

- \texttt{CLK10} – The VXIbus CLK10 (10 MHz) line.
- \texttt{EXTernal} – The AFG’s front panel “Ref/Smpl In” BNC (TTL levels).
- \texttt{ECLTrg0 or 1} – The VXIbus ECL trigger lines.
- \texttt{INTernal[1]} – The internal 42.94967296 MHz oscillator (default source).
- \texttt{INTernal2} – The internal 40 MHz oscillator.

The \texttt{INTernal[1]} reference oscillator is recommended for use with the Direct-Digital-Synthesis (DDS) time base ([SOURce:]FREQuency[1] subsystem) for high resolution and frequency range. The \texttt{INTernal2} reference oscillator is recommended for use with the divide-by-n time base ([SOURce:]FREQuency2 subsystem) to produce exact frequencies such as 10 MHz, 20 MHz, etc.

AFG Frequency Synthesis Modes

When outputting a fixed (continuous) frequency signal, the DDS time base ([SOURce:]FREQuency[1]) or the divide-by-N time base ([SOURce:]FREQuency2) are the most often used. In addition to these time bases, other sources which can be selected with the TRIGger[:STARt]:SOURce command are:

- \texttt{BUS} (the HP-IB Group Execute Trigger (GET command) or the IEEE-488.2 *TRG common command)
- \texttt{ECLTrg0 or ECLTrg1} (the VXIbus ECL trigger lines)
- \texttt{EXTernal} (the AFG’s front panel “Ref/Smpl In” BNC)
- \texttt{HOLD} (suspends sample generation)
- \texttt{INTernal[1]} (the [SOURce:]FREQuency[1] subsystem DDS frequency synthesis)
- \texttt{INTernal2} (the [SOURce:]FREQuency2 subsystem Divide-by-n frequency synthesis)
- \texttt{TTLTrg0 through 7} (the VXIbus TTL trigger lines)

In programs where the time base (trigger source) is not specified, the default DDS time base ([SOURce:]FREQuency[1]) is used.
Direct frequency control (that is, the [SOURce]:FREQuency commands) is only available with the INTernal1 and INTernal2 time base sources. For all other sources, the output frequency depends upon the frequency of the time base source. [SOURce]:FREQuency commands will be accepted with other time base sources, but become effective when the source is changed to INTernal1 or INTernal2.

**Divide-by-N Frequency Synthesis**

Fixed frequency (continuous) waveforms are the only signals allowed by the Divide-by-N frequency synthesis method ([SOURce]:FREQuency2 subsystem). All waveforms except standard function sine waves can be output using Divide-by-N frequency synthesis.

**AFG Frequency Modes**

There are four frequency “modes” available using the DDS time base ([SOURce]:FREQuency[1]). The modes selected by the [SOURce]:FREQuency[1]:MODE command are:

- CW | FIXed – single frequency mode.
- FSKey – frequency shift keying mode
- LIST – frequency list mode
- SWEep – frequency sweep mode

CW or FIXed is the default mode but is specified in many of the programs to emphasize that the arm source specified by ARM[:START]:LAYer2:SOURce <source> is for fixed (continuous) frequency waveforms.

**Frequency Sweeping and Lists**

Frequency sweeping and frequency lists are only available using the DDS time base. When setting the frequency mode, SWEep must be selected for frequency sweeps and LIST must be selected for frequency lists.
AFG Arming Sources

The arming sources set by the `ARM[:STARt]:LAYer2:SOURce <source>` command are:

- **BUS** – The HP-IB Group Execute Trigger (GET) command or the IEEE-488.2 `*TRG` common command.
- **ECLTrg0 and ECLTrg1** – The VXIbus ECL trigger lines.
- **EXTernal** – The HP E1445A’s front panel “Start Arm In” BNC connector (TTL levels).
- **HOLD** – Suspend arming. Use the `ARM:STARt:LAYer2:IMMediate` command to start the waveform.
- **IMMediate** – Immediate arming. An arm is internally generated two to three reference oscillator cycles after the start trigger sequence enters the wait-for-arm state.
- **TTLTrg0 through TTLTrg7** – The VXIbus TTL trigger lines.

AFG Arm Count

The arm count specifies the number of arms the AFG is to receive before it returns to the Idle state. The arm count is set with the `ARM[:STARt]:LAYer2:COUNt` command. The range is 1 through 65535, or `INFinity`. The default value is 1.

Waveform Repetition Count

The waveform repetition (cycle) count specifies the number of cycles per arm. The cycle count is specified with the `ARM[:STARt][:LAYer[1]]:COUNt <number>` command. The range for the cycle count is 1 through 65536, or `INFinity`. The default value is `INFinity`.

Stop Trigger Sources

Stop triggers abort the waveform cycle (repetition) count at the end of the current cycle. The stop trigger sources set with the `TRIGger:STOP:SOURce` command are:

- **BUS** – The HP-IB Group Execute Trigger (GET) command or the IEEE-488.2 `*TRG` common command.
- **EXTernal** – The AFG’s front panel “Stop Trigger/FSK/Gate In” BNC connector (TTL levels).
- **HOLD** – Suspend stop triggering. Use the `TRIGger:STOP:IMMediate` command to terminate a start arm cycle (default source).
- **TTLTrg0 through TTLTrg7** – The VXIbus TTL trigger lines.
External Stop Trigger Slope

An external stop trigger signal is applied to the AFG’s “Stop Trig/FSK/Gate In” BNC connector. The edge of the signal on which the AFG is triggered is set with the TRIGger:STOP:SLOPe command. The edges are:

- **POSitive** – Selects the rising edge of the signal.
- **NEGative** – Selects the falling edge of the signal.

AFG Gating Sources

The source which gates the triggers is specified with the TRIGger[:STARt]:GATE:SOURce command. The available sources are:

- **EXTernal** – The HP E1445A’s front panel “Stop Trig/FSK/Gate In” BNC connector (default source). This BNC is driven by TTL levels.
- **TTLTrg0 through TTLTrg7** – The VXIbus TTL trigger lines.

AFG Gate Polarity

The polarity of the signal which gates the output is specified with the TRIGger[:STARt]:GATE:POLarity command. The polarities which can be selected are:

- **NORMal** – Selects an “active high” gate. When the gate signal is high, the gate is active and the output is suspended at the last amplitude point triggered. When the gate is low (inactive), the output resumes with the next point.
- **INVerted** – Selects an “active low” gate (default polarity). When the gate signal is low, the gate is active and the output is suspended at the last amplitude point triggered. When the gate is high (inactive), the output resumes with the next point.

The gate polarity applies only to the EXTernal gate source (front panel “Gate In” BNC). If you are using a TTLTrg0 through TTLTrg7 trigger line as a gating source, the gate is always “active low”.

Gating and Signal Phase

Gating the triggers suspends the output at the last amplitude point triggered. When the gate is inactive, the waveform resumes with the next amplitude point. Thus, the phase of the signal remains continuous.

Enabling the Gate

Before the AFG triggers can be gated, the gate must be enabled. This is done with the TRIGger[:STARt]:GATE:STATe command. When the mode is ON, gating is enabled. When OFF, gating is disabled.
Frequency Sweep/
List Arming

The source which arms the frequency sweep or list is set with the
ARM:SWEep:SOURce command. The available sources are:

- **BUS** – The HP-IB Group Execute Trigger (GET) command or the
  IEEE-488.2 *TRG common command.
- **HOLD** – Suspend sweep or frequency list arming. Arm using
  ARM:SWEep[:IMMediate].
- **IMMediate** – Immediate sweep or frequency list arming. If the sweep
  or frequency list advance trigger (TRIGger:SWEep:SOURce) is set to
  TIMer, the sweep or list starts when the first start arm is received. If the
  sweep or frequency list advance source is set to any other source, the
  sweep or list starts when INITiate[:IMMediate] is executed.
- **LINK** – The next valid start arm starts the sweep or frequency list.
- **TTLTrg0 through TTLTrg7** – The VXIbus TTL trigger lines.

After the AFG is armed, the first frequency in the sweep or list is output.
Trigger signals output the remaining frequencies.

Frequency Sweep/
List Advance Trigger

The source which advances the sweep or frequency list to the next
frequency is set with the TRIGger:SWEep:SOURce command. The
available sources are:

- **BUS** – The HP-IB Group Execute Trigger (GET) command or the
  IEEE-488.2 *TRG common command.
- **HOLD** – Suspend sweep or frequency list advance triggering.
  Advance to the next frequency using TRIGger:SWEep[:IMMediate].
- **LINK** – The next valid start arm advances the sweep or frequency list.
- **TIMer** – The SOURce:SWEep:TIME and TRIGger:SWEep:TIMer
  commands control the sweep and frequency list advance timing
  (default source).
- **TTLTrg0 through TTLTrg7** – The VXIbus TTL trigger lines.

Placing the AFG in the wait-for-arm state (INITiate[:IMMediate]) puts the first
frequency in the sweep or list at the output. Trigger signals output the
remaining frequencies. Thus, for multiple sweeps or passes through the
frequency list, n-1 triggers are required for the first pass and n triggers are
required for all subsequent passes (n = number of points = number of triggers).

Immediate Arming
and Triggering

When the sweep and frequency list arming and triggering sources are set to
HOLD, the starting frequency is output when the AFG is set to the wait-for-arm
state (INITiate[:IMMediate]). Once the sweep or frequency list arm is received
ARM:SWEep[:IMMediate], the sweep or list can be advanced when a sweep or
list advance trigger (TRIGger:SWEep:IMMediate) is received.
Chapter Contents

This chapter shows how to generate the different signals at the front panel’s “Marker Out” BNC and how to select the ECL trigger lines. Use these signals and trigger lines to synchronize multiple AFGs, generate trigger pulses, etc. The sections are as follows:

- Marker Pulse Enable Flowchart ........................................ Page 204
- Available Marker Sources .................................................. Page 205
- Arbitrary Generated Marker Pulses ................................. Page 206
  - Generating Marker Pulses for Arbitrary Waveforms ........ Page 206
  - Generating Multiple Marker Pulses in Multiple Segment Lists .... Page 207
  - Generating Single Marker Pulses in Single Waveform Segments ... Page 212
- Generating Marker Pulses for Each Waveform Point ........ Page 214
- Operating Multiple AFGs Together ............................. Page 218
- Marker Program Comments ............................................. Page 222
  - Determining the Number of Marker Points of a Waveform Segment ... Page 222
  - Determining the Number of Marker Points of a Segment Sequence ... Page 222
The flowchart in Figure 6-1 shows how to select and output the different marker pulses at the front panel “Marker Out” BNC and the ECL trigger lines. Remove the flowchart from the binder for easy accessibility. Refer to the flowchart while doing the examples in this chapter, if desired.

Figure 6-1. Commands for Marker Pulses
Available Marker Sources

There are six marker sources available for output at the AFG’s front panel “Marker Out” BNC and the ECL Trigger Lines. Use the [SOURce:]MARKer:FEED <source> command to select the “Marker Out” BNC; use [SOURce:]MARKer:ECLTrg<en>:FEED <source> command to select the ECL trigger lines.

The sources for the “Marker Out” BNC, not the ECL trigger lines, can output the marker pulses as either active high (NORMAL) or active low (INVERTed). Use the [SOURce:]MARKer:POLarity <polarity> command to select the polarity.

The different marker sources are as follows:

**ARM[:STARt|:SEQuence[1]][:LAYer[1]]**

For arbitrary waveforms, the marker level changes with the first point on the waveform of the first waveform repetition. The source then outputs a marker pulse at the last waveform point of each repetition. For SINUsoid outputs, the marker is a 50% duty cycle square wave at the waveform frequency.

**ARM[:STARt|:SEQuence[1]]:LAYer2**

The AFG asserts a marker when triggering the first amplitude point after receiving a start arm. The AFG unasserts a marker with the last amplitude point of the last waveform repetition, or following an ABORT.

**[SOURce:]FREQuency[1]:CHANge**

The source outputs a one sample period wide marker pulse after a frequency change occurs. This shows that the steady state frequency was reached.

**[SOURce:]LIST[1]**

The source outputs marker pulses specified by the [SOURce:]LIST[1][:SEGMent]:MARKer and [SOURce:]LIST[1]:SSEQUence:MARKer commands. Increase the pulse size by selecting marker output for consecutive points on the waveform. Can only be used with arbitrary waveforms (see Chapters 3 and 7 on how to generate arbitrary waveforms).

**[SOURce:]PM:DEViation:CHANge**

This source outputs a one sample period wide marker pulse after a phase change occurs. This shows that the new phase was reached.
[SOURce:]ROSCillator

The source outputs the reference oscillator selected by [SOURce:]ROSCillator:SOURce.

TRIGger[:START]::SEQquence[1]

The source outputs a nominal 12 nS marker pulse for each point of a waveform segment.

Arbitrary Generated Marker Pulses

To generate marker pulses for arbitrary waveforms, do the following:

1. Select the “[SOURce:]LIST[1]” source for the front panel “Marker Out” BNC connector using [SOURce:]MARKer:FEED “[SOURce:]LIST[1]”.
2. Select the marker pulse polarity using [SOURce:]MARKer:POLarity <polarity>.
3. Enable the AFG to output a marker list using [SOURce:]MARKer[:STATe] <mode>.
4. Define a marker list in a waveform segment using [SOURce:]LIST[1][:SEGment]:MARKer <marker_list> or [SOURce:]LIST[1][:SEGment]:MARKer:SPOint <point>.
5. Enable the waveform segment in a segment sequence to output the marker list using [SOURce:]LIST[1]:SSEQuence:MARKer <marker_list> or [SOURce:]LIST[1]:SSEQuence:MARKer:SPOint <point>.

Generating Marker Pulses for Arbitrary Waveforms

The following programs show how to generate the marker pulses using two different methods. The two methods are:

1. [SOURce:]LIST[1][:SEGment]:MARKer defines for each point in a waveform segment where a marker is to be output. Likewise, [SOURce:]LIST[1]:SSEQuence:MARKer enables (or disables) marker outputs for each waveform segment in a segment sequence.

2. [SOURce:]LIST[1][:SEGment]:MARKer:SPOint defines a single segment or point in a waveform segment where the marker pulse is to be output. Likewise, [SOURce:]LIST[1]:SSEQuence:MARKer:SPOint enables a marker output for a single waveform segment in a segment sequence.
Generating Multiple Marker Pulses in Multiple Segment Lists

The MARKSEG1 program shows how to generate marker pulses using [SOURce:LIST1]:SEGment:MARKer and [SOURce:LIST1]:SSEQquence:MARKer. The program generates a sine wave and triangle wave. It generates a 10 points wide active low marker pulses starting at the center of the triangle waveform. The program generates a 512 point, 5 V sine wave and 5 V triangle wave.

The commands are:

1. **Reset the AFG**
   
   `*RST`

2. **Clear the AFG Memory of All Sequence and Segment Data**
   
   [SOURce:LIST1]:SEQUence:DELe:ALL
   [SOURce:LIST1]:SEGment:DELe:ALL

3. **Setup the AFG for Output**
   
   [SOURce:FREQuency1][:CW | :FIXed] <frequency>
   [SOURce:FUNCtion][:SHAPe] USER
   [SOURce:VOLTage][:LEVEL][:IMMediate][:AMPLitude] <amplitude>

4. **Select the Marker Source**
   
   [SOURce:MARKer]:FEED “[SOURce:LIST1]”
   
   This command selects the marker source for the front panel’s “Marker Out” connector to output marker pulses generated by arbitrary waveforms. (See “Available Marker Sources” on page 205 for the different sources.)

5. **Select the Marker Polarity**
   
   [SOURce:MARKer]:POLarity <polarity>
   
   NORMal <polarity> selects active high marker pulses; INVerted selects active low marker pulses.
6. **Enable Marker Outputs**
   
   [SOURce:]MARKer[:STATe] ON
   
   This command enables the AFG to output marker pulses. However, before the marker pulses can be output, they must be selected in the waveform segment and the waveform segment must be selected for marker output in the segment sequence. (Although *RST automatically enables the AFG for marker outputs, it is given here for good programming practice.)

7. **Setup the First Waveform Segment**
   
   [SOURce:]LIST[1]:SEGMent:SELect <name>
   [SOURce:]LIST[1]:SEGMent:DEFine <length>

8. **Store the First Waveform Segment as Voltage Data Points**
   
   [SOURce:]LIST[1]:SEGMent:VOLTage <voltage_list>

9. **Setup the Second Waveform Segment**
   
   [SOURce:]LIST[1]:SEGMent:SELect <name>
   [SOURce:]LIST[1]:SEGMent:DEFine <length>

10. **Store the Second Waveform Segment as Voltage Data Points**
    
    [SOURce:]LIST[1]:SEGMent:VOLTage <voltage_list>

11. **Store the Marker List for the Second Waveform Segment**
    
    [SOURce:]LIST[1]:SEGMent:MARKer <marker_list>
    
    This command stores the marker list into memory as a comma (",") separated list. A “1” selects a marker pulse and a “0” does not. (You can also send this list as Definite or Indefinite Length Arbitrary Block Data, as explained in Chapter 7.)

12. **Setup the Segment Sequence**
    
    [SOURce:]LIST[1]:SSEQuence:SELect <name>
    [SOURce:]LIST[1]:SSEQuence:DEFine <length>
    [SOURce:]LIST[1]:SSEQuence:SEQUence <segment_list>

13. **Select the Waveform Segment for Marker Output**
    
    [SOURce:]LIST[1]:SSEQuence:MARKer <marker_list>
    
    This command selects the waveform segment in a segment sequence that is to output the marker pulses. The marker pulses must be selected by
    
    [SOURce:]LIST[1]:SEGMent:MARKer <marker_list> or
    [SOURce:]LIST[1]:SEGMent:MARKer:SPOInt <point>
    
    before they are output.

14. **Generate the Output**
    
    [SOURce:]FUNCTION:USER <name>
    
    INITiate[:IMMediate]
HP BASIC Program Example (MARKSEG1)

1 !RE-STORE"MARKSEG1"
2 !This program computes a sine wave and a triangle wave as arbitrary
3 !waveforms. A corresponding marker list is defined for the triangle
4 !wave. The program sets the output sequence to consist of both
5 !waveforms, and enables marker pulses to be output with selected
6 !triangle waveform amplitude points.
7 !
8 Assign I/O path between the computer and E1445A.
9 ASSIGN @Afg TO 70910
10 COM @Afg
11 !
12 !Set up error checking
13 ON INTR 7 CALL Errmsg
14 ENABLE INTR 7;2
15 OUTPUT @Afg;"*CLS"
16 OUTPUT @Afg;"*SRE 32"
17 OUTPUT @Afg;"*ESE 60"
18 !
19 !Call the subprograms which reset the AFG and delete all existing
20 !waveform segments and sequences.
21 CALL Rst
22 CALL Wf_del
23 !
24 !Set up the AFG
25 OUTPUT @Afg;"SOUR:FREQ1:FIX 512E3;"; !frequency
26 OUTPUT @Afg;"SOUR:FUNC:SHAP USER;"; !function
27 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5.1V" !amplitude
28 OUTPUT @Afg;"SOUR:MARK:FEED "SOUR:LIST1"" !marker source
29 OUTPUT @Afg;"SOUR:MARK:POL INV" !marker polarity
30 OUTPUT @Afg;"SOUR:MARK:STAT ON" !enable marker
31 !
32 CALL Sine_wave
33 CALL Tri_wave
34 CALL Seq_list
35 !
36 !waveform sequence
37 !wait-for-arm state
38 !
39 WAIT .1 !allow interrupt to be serviced
40 OFF INTR 7
41 END
42 !
43 SUB Sine_wave
44 Sine_wave: !Subprogram which computes a sine wave.
45 COM @Afg
46 DIM Waveform(1:512) !Calculate sine wave
47 FOR l=1 TO 512
48 Waveform(l)=5.*(SIN(2.*PI*(l/512.)))
49 !Continued on Next Page
420 NEXT I  
430 !  
440 OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL SINE" !segment name  
450 OUTPUT @Afg;" SOUR:LIST1:SEGM:DEF 512" !segment size  
460 OUTPUT @Afg;"SOUR:LIST1:SEGM:VOLT ";Waveform(*) !amplitude points  
470 SUBEND  
480 !  
490 SUB Tri_wave  
500 Tri_wave: !Subprogram which computes a triangle wave and marker list.  
510 COM @Afg  
520 DIM Waveform(1:512),Marker_list(1:512)  
530 FOR I=1 TO 256 !Calculate triangle wave  
540 Waveform(I)=I*.0195313  
550 NEXT I  
560 FOR I=257 TO 512  
570 Waveform(I)=(512-I)*.0195313  
580 NEXT I  
590 !  
600 FOR I=256 TO 266 !Define marker list  
610 Marker_list(I)=1  
620 NEXT I  
630 !  
640 !Load waveform points and marker list  
650 OUTPUT @Afg;"SOUR:MARK:FEED ""SOUR:LIST1"" !markers at fp BNC  
660 OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL TRI" !segment name  
670 OUTPUT @Afg;" SOUR:LIST1:SEGM:DEF 512" !segment size  
680 OUTPUT @Afg;"SOUR:LIST1:SEGM:VOLT ";Waveform(*) !amplitude points  
690 OUTPUT @Afg;"SOUR:LIST1:SEGM:MARK ";Marker_list(*) !marker list  
700 SUBEND  
710 !  
720 SUB Seq_list  
730 Seq_list: !This subprogram defines the sequence list and enables  
740 !marker signals to be output with the triangle wave  
750 !segment.  
760 COM @Afg  
770 OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEL WAVE_OUT" !sequence name  
780 OUTPUT @Afg;"SOUR:LIST1:SSEQ:DEF 2" !number of segments  
790 OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEQ SINE,TRI" !segments in sequence  
800 OUTPUT @Afg;"SOUR:LIST1:SSEQ:MARK 0,1" !enable marker on segment TRI  
810 SUBEND  
820 !  
830 SUB Rst  
840 Rst: !Subprogram which resets the E1445.  
850 COM @Afg  
860 OUTPUT @Afg;"**RST;"OPC?" !reset the AFG  
870 ENTER @Afg;Complete  
880 SUBEND  
890 !  
900 SUB Wf_del

Continued on Next Page
Wf_del: !Subprogram which deletes all sequences and segments.
COM @Afg
OUTPUT @Afg;"FUNC:USER NONE" !select no sequences
OUTPUT @Afg;"LIST:SSEQ:DEL:ALL" !Clear sequence memory
OUTPUT @Afg;"LIST:SEGM:DEL:ALL" !Clear segment memory
SUBEND
!
SUB Errmsg
Errmsg: !Subprogram which displays E1445 programming errors
COM @Afg, Seg_mem$,Seq_mem$
DIM Message$[256]
!Read AFG status byte register and clear service request bit
B=SPOLL(@Afg)
!End of statement if error occurs among coupled commands
OUTPUT @Afg;"
OUTPUT @Afg;"ABORT" !abort output waveform
REPEAT
OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
ENTER @Afg;Code,Message$
PRINT Code,Message$
UNTIL Code=0
STOP
SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, MARKSEG1.FRM, is in directory “VBPROG” and the Visual C example program, MARKSEG1.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
The MARKSEG2 program shows how to generate marker pulses using [SOURce:]LIST[1]:SEGMen[:MARKer:SPOint] and [SOURce:]LIST[1]:SEQUence:MARKer:SPOint. The program generates a sine wave and triangle wave. It outputs Active high marker pulse at the center of the triangle waveform. The program generates a 512 point, 5 V sine wave and 5 V triangle wave.

The commands are the same ones listed in “Generating Multiple Marker Pulses in Multiple Waveform Segment Lists” on page 207, except they only select single point-wide marker pulses. The exceptions are as follows:

10. **Store the Marker Pulse Location for the Second Waveform Segment**
    
    [SOURce:]LIST[1]:SEGMen[:MARKer:SPOint <point>]
    
    This command selects the segment or point on a waveform where the marker pulse is to be output. For example, to output a marker pulse at point 5 of a 10 point waveform, execute
    
    [SOURce:]LIST[1]:SEGMen[:MARKer:SPOint 5].

12. **Select the Waveform Segment for Marker Output**
    
    [SOURce:]LIST[1]:SEQUence:MARKer:SPOint <point>]
    
    This command selects the waveform segment in a segment sequence that is to output the marker pulses. The marker pulses must be selected by
    
    [SOURce:]LIST[1]:SEGMen[:MARKer:SPOint <point>] or [SOURce:]LIST[1]:SEGMen[:MARKer <marker_list>] before they are output.
HP BASIC Program Example (MARKSEG2)

The MARKSEG2 program is the same as the MARKSEG1 program on page 209 except it selects the marker pulses differently. The differences are as follows:

1 !RE-STORE"MARKSEG2"
2 !This program computes a sine wave and a triangle wave as arbitrary waveforms. A single marker pulse is output with amplitude point 256
3 !of the triangle wave.

170 !Set up the AFG
180 OUTPUT @Afg;"SOUR:FREQ1:FIX 512E3;"; !frequency
190 OUTPUT @Afg;":SOUR:FUNC:SHAP USER;"; !function
200 OUTPUT @Afg;":SOUR:VOLT:LEV:IMM:AMPL 5.1V" !amplitude
210 OUTPUT @Afg;"SOUR:MARK:FEED ""SOUR:LIST1""" !marker source
220 OUTPUT @Afg;"SOUR:MARK:POL NORM" !marker polarity
230 OUTPUT @Afg;"SOUR:MARK:STAT ON" !enable marker

500 Tri_wave: !Subprogram which computes a triangle wave and specifies
510 !a marker pulse to be output with amplitude point 256.
520 COM @Afg
530 DIM Waveform(1:512)
540 FOR I=1 TO 256 !Calculate triangle wave
550 Waveform(I)=I*.0195313
560 NEXT I
570 FOR I=257 TO 512
580 Waveform(I)=(512-I)*.0195313
590 NEXT I
600 !
610 !Load waveform points and specify a single marker pulse
620 OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL TRI" !segment name
630 OUTPUT @Afg;"SOUR:LIST1:SEGM:DEF 512" !segment size
640 OUTPUT @Afg;"SOUR:LIST1:SEGM:VOLT ""Waveform(*)" !amplitude points
650 OUTPUT @Afg;"SOUR:LIST1:SEGM:MARK:SPO 256" !marker on point 256
660 SUBEND
670 !
680 SUB Seq_list
690 Seq_list: !This subprogram defines the sequence list and enables
700 !a marker signal to be output with the triangle wave
710 !segment.
720 COM @Afg
730 OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEL WAVE_OUT" !sequence name
740 OUTPUT @Afg;"SOUR:LIST1:SSEQ:DEF 2" !number of segments
750 OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEQ SINE,TRI" !segments in sequence
760 OUTPUT @Afg;"SOUR:LIST1:SSEQ:MARK:SPO 2" !enable marker on segment TRI
770 SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, MARKSEG2.FRM, is in directory “VBPROG” and the Visual C example program, MARKSEG2.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Generating Marker Pulses for Each Waveform Point

The MARKTRG program shows how to generate and output a 12 nS wide marker pulse at each point of the waveform. The pulses are output at the “Marker Out” BNC. Since the pulses are output each time a segment is output, the pulse rate is the same as the sample rate (you can use this function as another way to lockstep multiple AFGs). The example generates a 10 point, +5 V ramp. Although this example generates an arbitrary waveform, the pulses can be generated in any function and sample source.

The commands are:

1. **Reset the AFG**
   
   `*RST`

2. **Clear the AFG Memory of All Sequence and Segment Data**
   
   `[SOURce:]LIST[1]:SSEQ:DEL:ALL`
   
   `[SOURce:]LIST[1]:SEG:DEL:ALL`

3. **Setup the AFG for Output**
   
   `[SOURce:]FREQ:FIX:CW <frequency>
   
   `[SOURce:]FUNCTION:SHAPE USER`
   
   `[SOURce:]VOLT:LEV:IMMed:AMPL <amplitude>`

4. **Select the Marker Source**
   
   `[SOURce:]MARK:FEED TRIG[STAR]:SEQ[1]`

   This command selects the marker source for the front panel’s “Marker Out” connector to output marker pulses at the sample rate. (See “Available Marker Sources” on page 205 for the different sources.)
5. Select the Marker Polarity
   [SOURce:]MARKer:POLarity <polarity>
   NORMAL <polarity> selects active high marker pulses;
   INVerted <polarity> selects active low marker pulses.

6. Enable Marker Outputs
   [SOURce:]MARK[er]:[STATe] ON
   This commands enables the AFG to output marker pulses. (Although
   *RST automatically enables the AFG for marker outputs, it is given
   here for good programming practice.)

7. Setup the Waveform Segment; Store it as Voltage Data Points
   [SOURce:]LIST[1]:SEGMent:SELect <name>
   [SOURce:]LIST[1]:SEGMent:DEFine <length>
   [SOURce:]LIST[1]:SEGMent:VOLTage <voltage_list>

8. Setup the Sequence and Generate the Output
   [SOURce:]LIST[1]:SSEQUence:SELect <name>
   [SOURce:]LIST[1]:SSEQUence:DEFine <length>
   [SOURce:]LIST[1]:SSEQUence:SEQUence <segment_list>
   [SOURce:]FUNCtion:USER <name>
   INITiate[:IMMediate]

HP BASIC Program Example (MARKTRG)

1 !RE-STORE"MARKTRG"
2 !This program computes a ramp wave as an arbitrary waveform, and
3 !outputs a marker pulse with each waveform amplitude point.
4 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !
50 !Set up error checking
60 ON INTR 7 CALL Errmsg
70 ENABLE INTR 7,2
80 OUTPUT @Afg;"**CLS"
90 OUTPUT @Afg;"**SRE 32"
100 OUTPUT @Afg;"**ESE 60"
110 !
120 !Call the subprograms which reset the AFG and delete all existing
130 !waveform segments and sequences.
140 CALL Rst
150 CALL Wf_del
160 !
170 !Set up the AFG
180 OUTPUT @Afg;"SOUR:FREQ1:FIX 1E6;"; ![frequency]
190 OUTPUT @Afg;".SOUR:FUNC:SHAP USER;"; function
200 OUTPUT @Afg;".SOUR:VOLT:LEV:IMM:AMPL 5.1V" !amplitude
210 OUTPUT @Afg;".SOUR:MARK:FEED ""TRIG:STAR""" !marker source
220 OUTPUT @Afg;".SOUR:MARK:POL NORM" !marker polarity
230 OUTPUT @Afg;".SOUR:MARK:STAT ON" !enable marker
240 !
250 CALL Ramp_wave
260 !
270 OUTPUT @Afg;".SOUR:FUNC:USER RAMP_OUT" !waveform sequence
280 OUTPUT @Afg;".INIT:IMM" !wait-for-arm state
290 !
300 WAIT .1 !allow interrupt to be serviced
310 OFF INTR 7
320 END
330 !
340 SUB Ramp_wave
350 Ramp_wave: !Subprogram which computes a ramp wave and sets the
351 !output sequence.
360 COM @Afg
370 DIM Waveform(1:10) !Calculate ramp wave
380 FOR I=1 TO 10
390 Waveform(I)=I*.5
400 NEXT I
410 !
420 OUTPUT @Afg;".SOUR:LIST1:SEGM:SEL RAMP" !segment name
430 OUTPUT @Afg;".SOUR:LIST1:SEGM:DEF 10" !segment size
440 OUTPUT @Afg;".SOUR:LIST1:SEGM:VOLT ";Waveform(*) !amplitude points
450 !
460 OUTPUT @Afg;".SOUR:LIST1:SSEQ:SEL RAMP_OUT" !sequence name
470 OUTPUT @Afg;".SOUR:LIST1:SSEQ:DEF 1" !number of segments
480 OUTPUT @Afg;".SOUR:LIST1:SSEQ:SEQ RAMP" !segments in sequence
490 SUBEND
500 !
510 SUB Rst
520 Rst: !Subprogram which resets the E1445.
530 COM @Afg
540 OUTPUT @Afg;".*RST;*OPC?" !reset the AFG
550 ENTER @Afg;Complete
560 SUBEND
570 !
580 SUB Wf_del
590 Wf_del: !Subprogram which deletes all sequences and segments.
600 COM @Afg
610 OUTPUT @Afg;".FUNC:USER NONE" !select no sequences
620 OUTPUT @Afg;".LIST:SSEQ:DEL:ALL" !Clear sequence memory
630 OUTPUT @Afg;".LIST:SEGM:DEL:ALL" !Clear segment memory
640 SUBEND
650 !
660 SUB Errmsg

Continued on Next Page
ErrMsg: Subprogram which displays E1445 programming errors

680 COM @Afg
690 DIM Message$(256)
700 !Read AFG status byte register and clear service request bit
710 B=SPOLL(@Afg)
720 !End of statement if error occurs among coupled commands
730 OUTPUT @Afg;"
740 OUTPUT @Afg;"ABORT"
750 !Abort output waveform
760 REPEAT
770 OUTPUT @Afg;"SYST:ERR?"
780 ENTER @Afg;Code,Message$
790 PRINT Code,Message$
800 UNTIL Code=0
810 SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, MARKTRG.FRM, is in directory “VBPROG” and the Visual C example program, MARKTRG.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Operating Multiple AFGs Together

The DRIFT program shows how to operate multiple AFGs together to synchronize their outputs to each other. One AFG (the master AFG) uses its Reference Oscillator Output as the Reference Oscillator source for the second AFG (the servant AFG). Thus, any frequency change caused by drift of the master AFG reference results in the same amount of change in the servant AFG. The master AFG generates a 1 MHz square wave; the servant, a 500 KHz square wave.

The commands are:

1. Reset the Master and Servant AFGs
   \*RST

2. Setup the Master AFG For Output
   [SOURce:]ROSCillator:SOURce INT2
   TRIGger[:STARt]:SOURce INTernal2
   [SOURce:]FREQuency[1]:FIXed <frequency>
   [SOURce:]FUNCTION[:SHAPe] SQUare
   [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>

3. Select the Master AFG’s Marker Source
   [SOURce:]MARKer:ECLTrg<0>:FEED
   “[SOURce:]ROSCillator”
   This command selects the marker source for the ECLTrg0 trigger line to output the Reference Oscillator clock pulses.
4. **Enable the ECLTrg0 Line**
   ```plaintext
   [SOURce:MARKer:ECLTrg0[:STATe] ON
   ``
   This enables the marker output on the ECLTrg0 trigger line.
   (Although *RST automatically enables the AFG for marker outputs, it is given here for good programming practice.)

5. **Select the Master AFG’s ECLTrg1 Feed Trigger Source**
   ```plaintext
   [SOURce:MARKer:ECLTrg1:FEED <source>
   “ARM:STARt:LAYer2”
   ``
   This source outputs a marker pulse when the master’s waveform output starts. The marker is output on the ECLTrg1 trigger line.

6. **Enable the ECLTrg1 Line**
   ```plaintext
   [SOURce:MARKer:ECLTrg1[:STATe] ON
   ``
   This enables the arm output on the ECLTrg1 trigger line.

7. **Select the Servant AFG’s Reference Oscillator Source**
   ```plaintext
   [SOURce:ROSCillator:SOURce ECLTrg<n>
   [SOURce:ROSCillator:FREQuency:EXTernal 40M
   ``
   This command selects the Reference Oscillator Source. To synchronize the servant AFG with the master, select the ECLTrg0 trigger line. (The ECLTRG0 line is a 40 MHz clock.)

8. **Select the Servant AFG’s Sample Source**
   ```plaintext
   TRIGger[:STARt]:SEQuence[1]:SOURce INT2
   ``
   Select the Divide-by-n time base for the sample source.

9. **Setup the Servant AFG For a 5 V Square Wave Output**
    ```plaintext
    [SOURce:FREQuency[1]:FIXed <frequency>
    [SOURce:FUNCtion[:SHAPe] SQUare
    ``

10. **Setup the Servant AFG Arm Source to be the ECLTrg1 Line**
    ```plaintext
    ARM[:STARt]:LAYer2:SOURce ECLTrg1
    ``
    This command tells the servant AFG to start on the arm signal from the master AFG.

11. **Generate the Servant AFG’s Output**
    ```plaintext
    INITiate[:IMMediate]
    ``

12. **Wait for the Servant AFG to Complete its Setup**
    ```plaintext
    STATus:OPC:INITiate OFF;*OPC?
    ``

13. **Generate the Master AFG’s Output**
    ```plaintext
    INITiate[:IMMediate]
HP BASIC Program Example (DRIFT)

!RE-STORE"DRIFT"
!This program sets up two AFG's to output 1 MHz square waves.
!To prevent these signals from drifting and creating a phase difference, the reference oscillator of a "master" AFG is shared by a "servant" AFG. The master's reference oscillator signal is output on VXI backplane trigger line ECLT0.

!Assign I/O paths between the computer and the AFGs.
ASSIGN @Afg_m TO 70910 !master AFG
ASSIGN @Afg_s TO 70911 !servant AFG

COM @Afg_m,@Afg_s

!Set up error checking
CALL Rst
OUTPUT @Afg_m;"*CLS" !master
OUTPUT @Afg_m;"*SRE 32"
OUTPUT @Afg_m;"*ESE 60"
OUTPUT @Afg_m;"*OPC?"
ENTER @Afg_m;Complete

OUTPUT @Afg_s;"*CLS" !servant
OUTPUT @Afg_s;"*SRE 32"
OUTPUT @Afg_s;"*ESE 60"
OUTPUT @Afg_s;"*OPC?"
ENTER @Afg_s;Complete
ON INTR 7 CALL Errmsg
ENABLE INTR 7;2

!Call the subprograms which reset the AFGs output sine waves 180 degrees out of phase.
CALL Square_wave_m
CALL Square_wave_s

!Set master AFG to wait-for-arm state
OUTPUT @Afg_m;"INIT:IMM" !start waveform

!allow interrupt to be serviced
OFF INTR 7
END

!Subprogram which sets up master AFG
COM @Afg_m,@Afg_s
OUTPUT @Afg_m;"SOUR:ROSC:SOUR INT2;"; !reference osc. source
OUTPUT @Afg_m;"TRIG:STAR:SOUR INT2;"; !trigger source
OUTPUT @Afg_m;"SOUR:FREQ2:FIX 1E6;"; !frequency
OUTPUT @Afg_m;"SOUR:FUNC:SHAP SQU;"; !function
OUTPUT @Afg_m;"SOUR:VOLT:LEV:IMM:AMPL 5V" !amplitude
420 OUTPUT @Afg_m;"SOUR:MARK:ECLT0:FEED "SOUR:ROSC"" !feed ref osc
430 OUTPUT @Afg_m;"SOUR:MARK:ECLT0:STAT ON" !enable ECLT0 trig line
440 OUTPUT @Afg_m;"SOUR:MARK:ECLT1:FEED "ARM:STAR:LAY2"!feed arm source
450 OUTPUT @Afg_m;"SOUR:MARK:ECLT1:STAT ON" !enable ECLT1 trig line
460 SUBEND
470 !
480 SUB Square_wave_s
490 Square_wave_s: !Subprogram which sets up servant AFG: square wave
500 in phase with master AFG, reference oscillator source
510 !external.
520 COM @Afg_m,@Afg_s
530 OUTPUT @Afg_s;"SOUR:ROSC:SOUR ECLT0;"; !reference source
540 OUTPUT @Afg_s;":SOUR:ROSC:FREQ:EXT 40E6;"; !reference frequency
550 OUTPUT @Afg_s;":TRIG:STAR:SOUR INT2;"; !trigger source
560 OUTPUT @Afg_s;":SOUR:FREQ2:FIX .5E6;"; !frequency
570 OUTPUT @Afg_s;":SOUR:FUNC:SHAP SQU;"; !function
580 OUTPUT @Afg_s;":SOUR:VOLT:LEV:IMM:AMPL 5V" !amplitude
590 OUTPUT @Afg_s;":ARM:STAR:LAY2:SOUR ECLT1" !arm source
600 !
610 OUTPUT @Afg_s;"INIT:IMM" !wait-for-arm state
620 OUTPUT @Afg_s;":STAT:OPC:INIT OFF;"*OPC?" !allow setup to complete
630 ENTER @Afg_s;Complete
640 SUBEND
650 !
660 SUB Rst
670 Rst: !Subprogram which resets the AFGs.
680 COM @Afg_m,@Afg_s
690 OUTPUT @Afg_m;"RST;"*OPC?" !reset the master AFG
700 ENTER @Afg_m;Complete
710 !
720 OUTPUT @Afg_s;"RST;"*OPC?" !reset the servant AFG
730 ENTER @Afg_s;Complete
740 SUBEND
750 !
760 SUB Errmsg
770 Errmsg: !Subprogram which displays E1445 programming errors
780 COM @Afg_m,@Afg_s
790 DIM Message$[256]
800 !Read AFG status byte register and clear service request bit
810 B=SPOLL(@Afg_m)
820 IF B THEN !master error
830 !End of statement if error occurs among coupled commands
840 OUTPUT @Afg_m;""
850 OUTPUT @Afg_m;"ABORT" !abort output waveform
860 PRINT "Master AFG"
870 PRINT
880 REPEAT
890 OUTPUT @Afg_m;"SYST:ERR?" !read AFG error queue
900 ENTER @Afg_m;Code,Message$

Continued on Next Page
1010 REPEAT
1020 OUTPUT @Afg_s;"SYST:ERR?" !read AFG error queue
1030 ENTER @Afg_s;Code,Message$
1040 PRINT Code,Message$
1050 UNTIL Code=0
1060 STOP
1070 END IF
1080 SUBEND

**Visual BASIC and Visual C/C++ Program Versions**
The Visual BASIC example program, DRIFT.FRM, is in directory "VBPROG" and the Visual C example program, DRIFT.C, is in directory "VCPROG" on the CD that came with your HP E1445A.

**Marker Program Comments**
The following program comments give additional details on the program examples in this chapter.

**Determining the Number of Marker Points of a Waveform Segment**
Use [SOURce:]LIST[1]:MARKer:POI Nts? to determine the length of the marker pulse list selected by [SOURce:]LIST[1]:MARKer. The command returns the marker list size of the currently selected waveform segment.

**Determining the Number of Marker Points of a Segment Sequence**
Use [SOURce:]LIST[1]:SSEQuence:MARKer:POINts? to determine the length of the marker pulse list selected by [SOURce:]LIST[1]:SSEQuence:MARKer. The command returns the marker list size of the currently selected segment sequence.
This chapter explains how to use the HP E1445A Arbitrary Function Generator at faster speeds and other operations.

Chapter 3 shows how to transfer waveform segments and segment sequences to the AFG as voltage values and ASCII data, respectively. This is the slowest method to transfer the lists to the AFG. This chapter shows faster ways to transfer the lists to the AFG. The sections are as follows:

- Data Transfer Methods and Speed Comparisons ........ Page 224
- Using Signed Data to Generate Waveforms .......... Page 225
  - Using the Signed Number Format ........ Page 225
- Using Unsigned Data to Generate Waveforms ....... Page 229
  - Using the Unsigned Number Format .......... Page 229
- Using Definite Length Arbitrary Blocks to Transfer Data .................................. Page 231
  - Definite Length Block Data Format ........ Page 231
  - Data Byte Size ................................ Page 231
- Using Indefinite Length Arbitrary Blocks to Transfer Data .................................. Page 235
  - Indefinite Length Block Data Format ........ Page 235
  - Data Byte Size ................................ Page 235
- Using Combined Signed Data .......................... Page 239
  - Combined Segment List Format ........ Page 239
  - Using the Combined List with the Signed Number Format ................................ Page 240
- Using Combined Unsigned Data ....................... Page 245
  - Using the Combined List with the Unsigned Number Format ................................ Page 245
- Using Combined Waveform Segments and Segment Sequences .................................. Page 250
  - Combined Segment Sequence List Format .... Page 250
- Using the VXIbus Backplane ......................... Page 259
  - Downloading Segment Data ..................... Page 259
  - Downloading Segment Data into Memory .... Page 259
  - Downloading Data Directly into the DAC .... Page 269
Data Transfer Methods and Speed Comparisons

Table 7-1 shows the timing relationship of the different data transfer methods used. The table lists the relative timing in descending order with the slowest method on top.

Table 7-1. Speed Relationships of Data Transfer Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Command</th>
<th>Approximate Time Savings*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment Voltage List</td>
<td>[SOURce:LIST[1]:SEGMent]:VOLTage</td>
<td>0</td>
</tr>
<tr>
<td>Segment DAC Code List</td>
<td>[SOURce:LIST[1]:SEGMent]:VOLTage:DAC</td>
<td>35%</td>
</tr>
<tr>
<td>Segment Combined List</td>
<td>[SOURce:LIST[1]:SEGMent]:COMBined</td>
<td>35%</td>
</tr>
<tr>
<td>Segment DAC Codes as Block Data</td>
<td>[SOURce:LIST[1]:SEGMent]:VOLTage:DAC</td>
<td>88%</td>
</tr>
<tr>
<td>Segment Combined List as Block Data</td>
<td>[SOURce:LIST[1]:SEGMent]:COMBined</td>
<td>94%</td>
</tr>
<tr>
<td>Segment/Sequence Combined List as Block Data</td>
<td>[SOURce:LIST[1]:SEGMent]:COMBined/ [SOURce:LIST[1]:SSEQ uence]:COMBined</td>
<td>94%</td>
</tr>
</tbody>
</table>

*The time saving percentages are referenced to the speed of the Segment Voltage List method
Using Signed Data to Generate Waveforms

Transferring waveform segments as Digital-to-Analog Converter (DAC) Codes to the AFG is faster than transferring a voltage list. This section shows how to transfer the lists as DAC codes using the Signed number format. The DAC codes are transferred to the AFG as a comma (",") separated list.

Note

The AFG can only accept a single number format at a time. Thus, if the AFG currently contains Unsigned data and you wish to send Signed data, you MUST delete the data in memory first before enabling the AFG to receive Signed data.

Using the Signed Number Format

This section shows how to setup the AFG to receive DAC codes in the Signed number format and how to calculate the codes from voltage values.

Transferring DAC Codes in the Signed Number Format

With the AFG set to receive DAC codes in the Signed number format, it receives the codes in 16-bit two’s complement numbers. Use the [SOURce:]ARBitrary:DAC:FORMat SIGNed command to select the format.

Determining DAC Codes in the Signed Number Format

For outputs into matched loads and with the amplitude set to maximum (+5.11875V), the following DAC codes generate the following outputs:

- Code 0 outputs 0 V
- Code -4096 outputs -5.12 V or negative full scale voltage
- Code +4095 outputs +5.11875 V or positive full scale voltage

To calculate DAC codes from voltage values, use the formula:

\[
\text{DAC Code} = \text{voltage value} / .00125
\]

For example, to output -2V:

\[
\text{DAC Code} = -2 / .00125 = -1600
\]

The SIGN_DAT program shows how to store a waveform segment (i.e., points of an arbitrary waveform) into the AFG’s segment memory. The points are stored in the Signed DAC number format. The data is transferred to the AFG as a comma (",") separated list. The example generates a 200 point -5 V to +5 V positive going ramp.
The commands are:

1. **Reset the AFG**  
   \*RST

2. **Clear the AFG Memory of All Sequence and Segment Data**  
   
   [SOURce:LIST[1]:SSEQuence:DELeTe:ALL]  
   [SOURce:LIST[1]:SEGMent:DELeTe:ALL]

3. **Setup the AFG for Output**  
   [SOURce:FREQuency[1]:CW | :FIXed] <frequency>  
   [SOURce:FUNCTION[:SHAPe] USER]  
   [SOURce:VOLTage[:LEVel][:IMMediate][:AMPlitude] <amplitude>]

4. **Select the DAC Data Source**  
   [SOURce:ARBitrary:DAC:SOURce INTernal]  
   This command selects the source that transfers data to the DAC  
   (see “DAC Sources” on page 280). Use INTernal to transfer the data  
   using the [SOURce:LIST[1]] subsystem.

5. **Select the DAC Data Format**  
   [SOURce:ARBitrary:DAC:FORMat SIGNed]  
   This command selects the SIGNed number format.

6. **Setup the Waveform Segment**  
   [SOURce:LIST[1]:SEGMent:SELect <name>]  
   [SOURce:LIST[1]:SEGMent:DEFine <length>]

7. **Store the Waveform Segment as Signed DAC Data**  
   [SOURce:LIST[1]:SEGMent:VOLTage:DAC <voltage_list>]  
   This command stores the waveform segment into segment memory  
   using the Signed number format set by the  

8. **Setup the Segment Sequence and Generate Output**  
   [SOURce:LIST[1]:SSEQuence:SELect <name>]  
   [SOURce:LIST[1]:SSEQuence:DEFine <length>]  
   [SOURce:LIST[1]:SSEQuence:SEQuence <segment_list>]
   [SOURce:FUNCTION:USER <name>]
   [INITiate[:IMMediate]]
The SIGN_DAT program is very similar to the example programs used in Chapter 3. The only difference is that this program generates (in line 360) and transfers (in line 430) segment data as DAC codes in the Signed number format instead of voltage values.

1 !RE-STORE"SIGN_DAT"
2 !This program downloads arbitrary waveform data as signed
3 !(2's complement) DAC codes. The waveform defined is a 200 point,
4 !-5V to +5V ramp wave.
5 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !
50 !Set up error checking
60 ON INTR 7 CALL Errmsg
70 ENABLE INTR 7:2
80 OUTPUT @Afg;"*CLS"
90 OUTPUT @Afg;"*SRE 32"
100 OUTPUT @Afg;"*ESE 60"
110 !
120 !Call the subprograms which reset the AFG and erase all waveform
130 !segments and sequences.
140 CALL Rst
150 CALL Wf_del
160 !
170 OUTPUT @Afg;"SOUR:FREQ1:FIX 200E3;"; !frequency
180 OUTPUT @Afg;"SOUR:FUNC:SHAP USER;"; !function (arbitrary)
190 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5.11875V" !amplitude
200 !
210 CALL Ramp_wave
220 !
230 OUTPUT @Afg;"SOUR:FUNC:USER RAMP_OUT" !waveform sequence
240 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
250 !
260 WAIT .1!allow interrupt to be serviced
270 OFF INTR 7
280 END
290 !
300 SUB Ramp_wave
310 Ramp_wave: !Subprogram which defines a ramp waveform and output
320 !sequence.
330 COM @Afg
340 DIM Waveform(1:200) !Calculate waveform points as dac codes
350 FOR I=-100 TO 99
360 Waveform(I+101)=(I*.050505)/.00125
370 NEXT I
380 !

Continued on Next Page
Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, SIGN_DAT.FRM, is in directory “VBPROG” and the Visual C example program, SIGN_DAT.C, is in directory “VCPROG” on the CD that came with your HP E1445A.

These programs are very similar to the example programs used in Chapter 3. The only difference is that this program transfers the segment data as DAC codes in the Signed number format instead of voltage values.
Using Unsigned Data to Generate Waveforms

Transferring waveform segments as Digital-to-Analog Converter (DAC) Codes to the AFG is faster than transferring a voltage list. This section shows how to transfer the lists as DAC codes using the Unsigned number format. The DAC codes are transferred to the AFG as a comma ("," ) separated list.

Note

The AFG can only accept a single number format at a time. Thus, if the AFG currently contains Unsigned data and you wish to send Signed data, you MUST delete the data in memory first before enabling the AFG to receive Signed data.

Using the Unsigned Number Format

This section shows how to setup the AFG to receive DAC codes in the Unsigned number format and how to generate the codes from voltage values.

Transferring DAC Codes in the Unsigned Number Format

With the AFG set to receive DAC codes in the UNSigned number format, it receives the codes as unsigned or offset binary numbers. Use the [SOURce:]ARBitrary:DAC:FORMat USigned command to select the format.

Determining DAC Codes in the Unsigned Number Format

For outputs into matched loads and with the amplitude set to maximum (+5.11875V), the following DAC codes generate the following outputs:

- Code 0 outputs -5.12 V or negative full scale voltage
- Code +4096 outputs 0 V
- Code +8191 outputs +5.11875 V or positive full scale voltage

To calculate DAC codes from voltage values, use the formula:

$$ DAC \text{ Code} = (\text{voltage value} / .00125) + 4096 $$

For example, to output -2V:

$$ DAC \text{ Code} = (-2 / .00125) + 4096 = -1600 + 4096 = 2496 $$

The UNS_DAT program shows how to store a waveform segment (i.e., points of an arbitrary waveform) into the AFG’s segment memory. The waveform segment is stored in the Unsigned number format. The data is transferred to the AFG as a comma ("," ) separated list. The example generates a 200 point +5 V to -5 V negative going ramp.

The commands are the same ones listed on page 226, except on how to select the Unsigned format and how generate the data. These exceptions are as follows:
5. **Select the DAC Data Format**

   [SOURce:]ARBitrary:DAC:FORMat UNSigned

   This command selects the UNSigned number format.

7. **Store the Waveform Segment as Unsigned DAC Data**

   [SOURce:]LIST[1][:SEGMen]t[:VOLTage]:DAC <voltage_list>

   This command stores the waveform segment into segment memory according to the Unsigned number format set by the
   [SOURce:]ARBitrary:DAC:FORMat UNSigned command.

**HP BASIC Program Example (UNS_DAT)**

Use the same BASIC program as the “SIGN_DAT” program beginning on page 227. The only difference is that this program generates (in line 360) and transfers (in line 400) the segment data as DAC codes in the Unsigned number format instead of the Signed format. The following lines show the differences of the two program examples:

```
1 !RE-STORE"UNS_DAT"

300 SUB Ramp_wave
310 Ramp_wave: !Subprogram which defines a ramp waveform and output
320 !sequence.
330 COM @Afg
340 DIM Waveform(1:200) !Calculate waveform points as dac codes
350 FOR I=-100 TO 99
    360 Waveform(I+101)=((I*.050505)/.00125)+4096
370 NEXT I
380 !
390 OUTPUT @Afg;"SOUR:ARB:DAC:SOUR INT" !dac data source
400 OUTPUT @Afg;"SOUR:ARB:DAC:FORM UNS" !dac data format (unsigned)
410 OUTPUT @Afg;"SOUR:LIST1:SEGMENT:RAMP" !segment name
420 OUTPUT @Afg;"SOUR:LIST1:SEGMENT:DEF 200" !segment size
430 OUTPUT @Afg;"SOUR:LIST1:SEGMENT:VOLT:DAC ";Waveform(*) !waveform pts
440 !
450 OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEL RAMP_OUT" !sequence name
460 OUTPUT @Afg;"SOUR:LIST1:SSEQ:DEF 1" !sequence size
470 OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEQ RAMP" !segment order
480 SUBEND
```

**Visual BASIC and Visual C/C++ Program Versions**

The Visual BASIC example program, UNS_DAT.FRM, is in directory “VBPROG” and the Visual C example program, UNS_DAT.C, is in directory “VCPROG” on the CD that came with your HP E1445A.

These program is very similar to the example programs used in Chapter 3. The only difference is that this program transfers the segment data as DAC codes in the Unsigned number format instead of voltage values.
Using Definite Length Arbitrary Blocks to Transfer Data

The AFG can receive DAC codes as Definite Length Arbitrary Block Data using either the Signed or Unsigned number format. This is a much faster method to transfer data than using a comma (",") separated list that was used in “Using Signed Data to Generate Waveforms” on page 225 and “Using Unsigned Data to Generate Waveforms” on page 229. (The speed is about the same as the method used in “Using Indefinite Length Arbitrary Blocks to Transfer Data” on page 235.)

Definite Length Block Data Format

A typical data block using the definite length format consists of:

where:

- “#” – Shows that the data to be sent is in an arbitrary block format.
- “<non-zero digit>” – is a single digit number that shows the number of digits contained in <digits>; for example, if the <digits> value equals 100 or 2000, the <non-zero digit> value equals 3 or 4, respectively.
- “<digits>” – Shows the number of data bytes to be sent; for example, if 100 data bytes are to be sent, <digits> equals 100 (see “Data Byte Size” below).
- “<8-bit data bytes>” – Is the data (i.e., DAC codes) sent to the AFG.
- A typical example of a data block sending 2000 8-bit data bytes is:
  
  #42000<data bytes>

Data Byte Size

The DAC codes are transferred to the AFG as 16-bit integer values that meet the coding set by the IEEE 488.2 standard. Since IEEE 488.2 requires an 8-bit code, the 16-bit integer must be sent as 2 8-bit values for each 16-bit integer.

For example, to send a waveform segment consisting of 1000 DAC codes (1000 points), the actual number of “<digits>” and “<8-bit data bytes>” equals:

\[ 1000 \times 2 = 2000 \]
HP BASIC Program Example (DACBLOK1)

The DACBLOK1 program shows how to store a waveform segment (i.e., points of an arbitrary waveform) into the AFG’s segment memory. The waveform segment is stored as DAC codes in the Signed number format. This program is the same program as SIGN_DAT beginning on page 227, except the data is transferred to the AFG using the Definite Length Arbitrary Block method. The example generates a 200 point -5 V to +5 V positive going ramp.

To transfer Definite Length Block Data to the AFG requires that the data sent with the [SOURce:]LIST[1]:SEGMent:VOLTage:DAC command must be contiguous. To do this, sent no carriage return (CR) and line feed (LF) before all the data is transferred. The format in line 440 disables the CR and LF. The CR and LF sent in line 460 tells the AFG that the data transfer is complete.

1 !RE-STORE"DACBLOK1"
2 !This program downloads arbitrary waveform data as signed
3 !(2's complement) DAC codes. The data is sent in an IEEE-488.2
4 !definite length block in 16-bit integer format. The waveform is
5 !a 200 point, -5V to +5V ramp wave.
6 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 ASSIGN @Afg1 TO 70910.;FORMAT OFF !path for binary data
40 COM @Afg,@Afg1
50 !
60 !Set up error checking
70 ON INTR 7 CALL Errmsg
80 ENABLE INTR 7;2
90 OUTPUT @Afg;"*CLS"
100 OUTPUT @Afg;"*SRE 32"
110 OUTPUT @Afg;"*ESE 60"
120 !
130 !Call the subprograms which reset the AFG and erase all waveform
140 !segments and sequences.
150 CALL Rst
160 CALL Wf_del
170 !
180 OUTPUT @Afg;"SOUR:FREQ1:FIX 200E3;"; !frequency
190 OUTPUT @Afg;"SOUR:FUNC:SHAPE USER;"; !function
200 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5.11875V" !amplitude
210 !
220 CALL Ramp_wave
230 !
240 OUTPUT @Afg;"SOUR:FUNC:USER RAMP_OUT" !waveform sequence
250 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
260 !
270 WAIT .1 allow interrupt to be serviced
280 OFF INTR 7

Continued on Next Page
SUB Ramp_wave
Ramp_wave: !Subprogram which defines a ramp waveform and output sequence.

COM @Afg1
INTEGER Waveform(1:200) !Calculate waveform points as dac codes
FOR I=-100 TO 99
    Waveform(I+101)=(I*.050505)/.00125
NEXT I

!OUTPUT @Afg;"SOUR:ARB:DAC:SOUR INT" !dac data source
OUTPUT @Afg;"SOUR:ARB:DAC:FORM SIGN" !dac data format (signed)
OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL RAMP" !segment name
OUTPUT @Afg;" SOUR:LIST1:SEGM:DEF 200" !segment size
OUTPUT @Afg USING ",K";"SOUR:LIST1:SEGM:VOLT:DAC #3400"
OUTPUT @Afg1;Waveform(*) !400 bytes: 3 digits (2 bytes/ampl point)
OUTPUT @Afg !CR LF

!OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEL RAMP_OUT" !sequence name
OUTPUT @Afg;" SOUR:LIST1:SSEQ:DEF 1" !sequence size
OUTPUT @Afg;" SOUR:LIST1:SSEQ:SEQ RAMP" !segment order

SUBEND

SUB Rst
Rst: !Subprogram which resets the E1445.
COM @Afg1
OUTPUT @Afg;"**RST;*OPC?" !reset the AFG
ENTER @Afg;Complete

SUBEND

SUB Wf_del
Wf_del: !Subprogram which deletes all sequences and segments.
COM @Afg1
OUTPUT @Afg;"FUNC:USER NONE" !select no sequences
OUTPUT @Afg;"LIST:SSEQ:DEL:ALL" !Clear sequence memory
OUTPUT @Afg;"LIST:SEGM:DEL:ALL" !Clear segment memory

SUBEND

SUB Errmsg
Errmsg: !Subprogram which displays E1445 programming errors
COM @Afg1
DIM Message$[256]
!Read AFG status byte register and clear service request bit
B=SPOLL(@Afg)
!End of statement if error occurs among coupled commands
OUTPUT @Afg;"";
OUTPUT @Afg;"ABORT" !abort output waveform
REPEAT
OUTPUT @Afg;"SYST:ERR?" !read AFG error queue

Continued on Next Page
Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, DACBLOK1.FRM, is in directory “VBPROG” and the Visual C example program, DACBLOK1.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Using Indefinite Length Arbitrary Blocks to Transfer Data

The AFG can receive DAC codes as Indefinite Length Arbitrary Block Data using either the Signed or Unsigned number format. This is a much faster method to transfer data than using a comma ("," ) separated list that was used in “Using Signed Data to Generate Waveforms” on page 225 and “Using Unsigned Data to Generate Waveforms” on page 229. (The speed is about the same as the method used in “Using Definite Length Arbitrary Blocks to Transfer Data” on page 231.)

Indefinite Length Block Data Format

A typical data block using the indefinite length format consists of:

```
# 0 <8-bit data bytes> LF^END
```

where:

- “#” – Shows that the data to be sent is in an arbitrary block format.
- “0” – Shows that the format is an indefinite length arbitrary block format; the “0” number must be sent since a different number shows the definite length arbitrary block format.
- “<8-bit data bytes>” – Is the data (i.e., DAC codes) sent to the AFG.
- “LF^END” – Means line feed (LF) sent with END (EOI) asserted. It indicates to the AFG that the end of data has been reached.

Data Byte Size

The DAC codes are transferred to the AFG as 16-bit integer values that meet the coding set by the IEEE 488.2 standard. Since IEEE 488.2 requires an 8-bit code, the 16-bit integer must be sent as 2 8-bit values for each 16-bit integer.

For example, to send a waveform segment consisting of 1000 DAC codes (1000 points), the actual number of “digits” and “8-bit data bytes” equals:

\[ 1000 \times 2 = 2000 \]
The DACBLOK2 program shows how to store a waveform segment (i.e., points of an arbitrary waveform) into the AFG’s segment memory. The waveform segment is stored as DAC codes in the Unsigned number format. This program is the same program as UNS_DAT beginning on page 230.

The data is transferred to the AFG using the Indefinite Length Arbitrary Block method. The example generates a 200 point +5 V to -5 V negative going ramp.

To transfer Indefinite Length Block Data to the AFG requires that the data sent with the \[\text{SOURce:}\text{LIST[1]}:\text{SEGMen}t:\text{VOLTage:DAC}\] command must be contiguous. To do this, sent no carriage return (CR) and line feed (LF) before all the data is transferred. Also, since EOL is a data terminating string, it must not be sent before the data transfer is complete. The format in line 440 disables the CR, LF, and EOL. The LF character and EOL string sent in line 460 tells the AFG that the data transfer is complete.

```
1 !RE-STORE"DACBLOK2"
2 !This program downloads arbitrary waveform data as unsigned
3 !DAC codes. The data is sent in an IEEE-488.2 indefinite length
4 !block in 16-bit integer format. The waveform is a 200 point,
5 !+5V to -5V ramp wave.
6 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 ASSIGN @Afg1 TO 70910.;FORMAT OFF !path for binary data
40 COM @Afg,@Afg1
50 !
60 !Set up error checking
70 ON INTR 7 CALL Errmsg
80 ENABLE INTR 7;2
90 OUTPUT @Afg;"*CLS"
100 OUTPUT @Afg;"*SRE 32"
110 OUTPUT @Afg;"*ESE 60"
120 !
130 !Call the subprograms which reset the AFG and erase all waveform
140 !segments and sequences.
150 CALL Rst
160 CALL Wf_del
170 !
180 OUTPUT @Afg;"SOUR:FREQ1:FIX 200E3;"; !frequency
190 OUTPUT @Afg;"SOUR:FUNC:SHAP USER;"; !function
200 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5.11875V" !amplitude
210 !
220 CALL Ramp_wave
230 !
240 OUTPUT @Afg;"SOUR:FUNC:USER RAMP_OUT" !waveform sequence
250 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
```

Continued on Next Page
260 !
270 WAIT .1!allow interrupt to be serviced
280 OFF INTR 7
290 END
300 !
310 SUB Ramp_wave
320 Ramp_wave: !Subprogram which defines a ramp waveform and output
330 !sequence.
340 COM @Afg,Afg1
350 INTEGER Waveform(1:200) !Calculate waveform points as dac codes
360 FOR I=100 TO -99 STEP -1
370 Waveform(101-I)=((I*.050505)/.00125)+4096
380 NEXT I
390 !
400 OUTPUT @Afg;"SOUR:ARB:DAC:SOUR INT" !dac data source
410 OUTPUT @Afg;"SOUR:ARB:DAC:FORM UNS" !dac data format (unsigned)
420 OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL RAMP" !segment name
430 OUTPUT @Afg;" SOUR:LIST1:SEGM:DEF 200" !segment size
440 OUTPUT @Afg USING ",K";"SOUR:LIST1:SEGM:VOLT:DAC #0"
450 OUTPUT @Afg1;Waveform(*)
460 OUTPUT @Afg;CHR$(10);END !terminate with line feed (LF) and EOI
470 !
480 OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEL RAMP_OUT" !sequence name
490 OUTPUT @Afg;" SOUR:LIST1:SSEQ:DEF 1" !sequence size
500 OUTPUT @Afg;" SOUR:LIST1:SSEQ:SEQ RAMP" !segment order
510 SUBEND
520 !
530 SUB Rst
540 Rst: !Subprogram which resets the E1445.
550 COM @Afg,Afg1
560 OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
570 ENTER @Afg;Complete
580 SUBEND
590 !
600 SUB Wf_del
610 Wf_del: !Subprogram which deletes all sequences and segments.
620 COM @Afg,Afg1
630 OUTPUT @Afg;"FUNC:USER NONE" !select no sequences
640 OUTPUT @Afg;"LIST:SSEQ:DEL:ALL" !Clear sequence memory
650 OUTPUT @Afg;"LIST:SEG:M:DEL:ALL" !Clear segment memory
660 SUBEND
670 !
680 SUB Errmsg
690 Errmsg: !Subprogram which displays E1445 programming errors
700 COM @Afg,Afg1
710 DIM Message$[256]
720 !Read AFG status byte register and clear service request bit
730 B=SPOLL(@Afg)
740 !End of statement if error occurs among coupled commands
750 OUTPUT @Afg;""

Continued on Next Page
Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, DACBLOK2.FRM, is in directory "VBPROG" and the Visual C example program, DACBLOK2.C, is in directory "VCPROG" on the CD that came with your HP E1445A.
Using Combined Signed Data

The Combined Segment Lists transfers both the arbitrary waveform segment data and marker pulses to the AFG (see Chapter 6 for information on marker pulses). You can use either the Signed or Unsigned number format for the list.

The Combined Segment List can be sent as a comma separated list (see “Using Signed Data to Generate Waveforms” on page 225), or as Definite Length or Indefinite Length Arbitrary Block Data (see “Using Definite Length Arbitrary Blocks to Transfer Data” on page 231 and “Using Indefinite Length Arbitrary Blocks to Transfer Data” on page 235, respectively).

This section shows how to transfer the lists as DAC codes using the Signed number format.

---

**Note**
The AFG can only accept a single number format at a time. Thus, if the AFG currently contains Signed data and you wish to send Unsigned data, you **MUST** delete the data in memory first before enabling the AFG to receive Unsigned data.

---

**Combined Segment List Format**

Refer to Figure 7-1 for the 16-bit Integer Combined Segment List. Bit 1 (bit value 2) sets the marker pulse. Bits 3 to 15 are the DAC codes.

![Figure 7-1. Combined List Format](image-url)
Using the Combined List with the Signed Number Format

This section shows how to setup the AFG to receive a combined list in the Signed number format and how to generate the list from voltage values.

Transferring the List in the Signed Number Format

With the AFG set to receive codes in the Signed number format, it receives the codes in 16-bit two’s complement numbers. Use the [SOURce:]ARBitrary:DAC:FORMat SIGNed command to select the format.

Determining Codes in the Signed Number Format

For outputs into matched loads and with the amplitude set to maximum (+5.11875V), the following codes generate the following outputs:

- Code 0 outputs 0 V
- Code -32768 outputs -5.12 V or negative full scale voltage
- Code +32760 outputs +5.11875 V or positive full scale voltage

To calculate combined DAC codes from voltage values, use the formula:

\[ \text{Code} = \frac{\text{voltage value}}{0.00125} \times 2^3 \]

For example, to output -2V:

\[ \text{DAC Code} = \frac{-2}{0.00125} \times 2^3 = -12800 \]

To output a marker at a particular point of a waveform, add “2” to the combined list DAC code value of the point. For example, to add a marker bit of a point with a voltage value of 5 V:

\[ \text{Code} = \left( \frac{5}{0.00125} \times 2^3 \right) + 2 = 32000 + 2 = 32002 \]
The COMBSIGN program shows how to store a combined list (i.e., points and/or marker bit of an arbitrary waveform) into the AFG’s segment memory. The list is stored in the Signed number format. The data is transferred to the AFG using the Definite Length Arbitrary Block Data method. The example generates a 200 point \(-5\) V to \(+5\) V positive going ramp. A marker is output at the zero crossing (or center) of the ramp.

The commands are:

1. **Reset the AFG**
   
   \[ \text{RST} \]

2. **Clear the AFG Memory of All Sequence and Segment Data**
   
   \[
   \text{[SOURce:]LIST[1]:SSEQuence:DELete:ALL} \\
   \text{[SOURce:]LIST[1]:SEGMen]:DELete:ALL}
   \]

3. **Setup the AFG for Output**
   
   \[
   \text{[SOURce:]FREQuency[1][:CW | :FIXed] <frequency>} \\
   \text{[SOURce:]FUNCtion[:SHApe] USER} \\
   \text{[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>}
   \]

4. **Select the DAC Data Source**
   
   \[
   \text{[SOURce:]ARBitrary:DAC:SOURce INTernal}
   \]
   
   This command selects the source that transfers data to the DAC (see “DAC Sources” on page 280). Use INTernal to transfer the data using the [SOURce:]LIST[1] subsystem.

5. **Select the DAC Data Format**
   
   \[
   \text{[SOURce:]ARBitrary:DAC:FORMat SIGNed}
   \]
   
   This command selects the SIGNed data number format.
6. Set the Marker Output Source
   [SOURce:]MARKer:FEED "[SOURce:]LIST[1]
   This command selects the LIST[1] source as the source that outputs a marker pulse at the “Marker Out” front panel terminals (see Chapter 6 for information on other sources).

7. Setup the Waveform Segment
   [SOURce:]LIST[1]:SEGment:SELect <name>
   [SOURce:]LIST[1]:SEGment:DEFine <length>

8. Store the Waveform Segment as Combined Signed DAC Data
   [SOURce:]LIST[1]:SEGment:COMBined <combined_list>
   This command stores the waveform segment into segment memory in the Signed format set by the [SOURce:]ARBitrary:DAC:FORMat SIGNed command. The data is sent as a combined list with the marker bit selected.

9. Setup the Sequence and Generate Output
   [SOURce:]LIST[1]:SSEQUence:SELect <name>
   [SOURce:]LIST[1]:SSEQUence:DEFine <length>
   [SOURce:]LIST[1]:SSEQUence:SEQUence <segment_list>
   [SOURce:]FUNCTION:USER <name>
   INITiate[:IMMediate]

HP BASIC Program Example (COMBSIGN)

1  !RE-STORE"COMBSIGN"
2  !This program downloads an arbitrary waveform as a combined
3  !(voltage and marker) list of signed (2’s complement) DAC codes.
4  !The data is sent in an IEEE-488.2 definite length block in 16-bit
5  !integer format. The waveform is a 200 point, -5V to +5V ramp wave.
6  !
10  !Assign I/O path between the computer and E1445A.
20  ASSIGN @Afg TO 70910
30  ASSIGN @Afg1 TO 70910;FORMAT OFF !path for binary data
40  COM @Afg,@Afg1
50  !
60  !Set up error checking
70  ON INTR 7 CALL Ermsg
80  ENABLE INTR 7:2
90  OUTPUT @Afg;"CLS"
100 OUTPUT @Afg;"SRE 32"
110 OUTPUT @Afg;"ESE 60"
120  !
130  !Call the subprograms which reset the AFG and erase all waveform
140  !segments and sequences.
150  CALL Rst

Continued on Next Page
CALL Wf_del
!
OUTPUT @Afg;"SOUR:FREQ:FIX 200E3;"; !frequency
OUTPUT @Afg;"SOUR:FUNC:SHAP USER;"; !function
OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5.11875V" !amplitude
!
CALL Ramp_wave
!
OUTPUT @Afg;"SOUR:FUNC:USER RAMP_OUT" !waveform sequence
OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
!
WAIT .1!allow interrupt to be serviced
OFF INTR 7
END
!
SUB Ramp_wave
Ramp_wave: !Subprogram which defines a ramp waveform and output sequence.
COM @Afg,@Afg1
INTEGER Waveform(1:200) !Calculate waveform points as dac codes
FOR I=-100 TO 99
  IF I=0 THEN
    Waveform(I+101)=0+2 !set marker bit with this amplitude point
  ELSE
    Waveform(I+101)=(I*.050505)/.00125 !shift bits to dac code positions
    Waveform(I+101)=SHIFT(Waveform(I+101),-3)
  END IF
NEXT I
!
OUTPUT @Afg;"SOUR:ARB:DAC:SOUR INT" !dac data source
OUTPUT @Afg;"SOUR:ARB:DAC:FORM SIGN" !dac data format (signed)
!output marker as defined by segment list
OUTPUT @Afg;"SOUR:MARK:FEED ""SOUR:LIST1"
OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL RAMP" !segment name
OUTPUT @Afg;"SOUR:LIST1:SEGM:DEF 200" !segment size
OUTPUT @Afg USING ",";"SOUR:LIST1:SEGM:COMB #3402"
OUTPUT @Afg1;Waveform(*) !400 bytes: 3 digits (2 bytes/ampl point)
OUTPUT @Afg !CR LF
!
OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEL RAMP_OUT" !sequence name
OUTPUT @Afg;"SOUR:LIST1:SSEQ:DEF 1" !sequence size
OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEQ RAMP" !segment order

SUBEND
!
SUB Rst
Rst: !Subprogram which resets the E1445.
COM @Afg,Afg1
OUTPUT @Afg;"RST;""OPC?" !reset the AFG
ENTER @Afg;Complete

Continued on Next Page
SUBEND

SUB Wf_del
Wf_del: Subprogram which deletes all sequences and segments.
    COM @Afg,Afg1
    OUTPUT @Afg;"FUNC:USER NONE" !select no sequences
    OUTPUT @Afg;"LIST:SSEQ:DEL:ALL" !Clear sequence memory
    OUTPUT @Afg;"LIST:SEGM:DEL:ALL" !Clear segment memory
SUBEND

SUB Errmsg
Errmsg: Subprogram which displays E1445 programming errors
    COM @Afg,Afg1
    DIM Message$[256]
    !Read AFG status byte register and clear service request bit
    B=SPOLL(@Afg)
    !End of statement if error occurs among coupled commands
    OUTPUT @Afg;"
    OUTPUT @Afg;"ABORT" !abort output waveform
    REPEAT
    OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
    ENTER @Afg;Code,Message$
    PRINT Code,Message$
    UNTIL Code=0
    STOP
SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, COMBSIGN.FRM, is in directory “VBPROG” and the Visual C example program, COMBSIGN.C, is in directory “VCPROG” on the CD that came with your HP E1445A.

This program sends the combined list using Signed data as Definite Length Arbitrary Block Data. It is thus very similar to the SIGN_DAT program beginning on page 227 and DACBLOK1 example program beginning on page 232.
Using Combined Unsigned Data

The Combined Segment Lists transfers both the arbitrary waveform segment data and marker pulses to the AFG (see Chapter 6 for information on marker pulses). You can use either the Signed or Unsigned number format for the list.

The Combined Segment List can be sent as a comma separated list (see “Using Signed Data to Generate Waveforms” on page 225), or as Definite Length or Indefinite Length Arbitrary Block Data (see “Using Definite Length Arbitrary Blocks to Transfer Data” on page 231 and “Using Indefinite Length Arbitrary Blocks to Transfer Data” on page 235, respectively).

This section shows how to transfer the lists as DAC codes using the Unsigned number format.

---

**Note**

The AFG can only accept a single number format at a time. Thus, if the AFG currently contains Signed data and you wish to send Unsigned data, you **MUST** delete the data in memory first before enabling the AFG to receive Unsigned data.

---

### Using the Combined List with the Unsigned Number Format

This section shows how to setup the AFG to receive a combined list in the Unsigned number format and how to generate the list from voltage values.

### Transferring the List in the Unsigned Number Format

With the AFG set to receive codes in the Unsigned number format, it receives the codes as unsigned or offset binary numbers. Use the [SOURce:]ARBitrary:DAC:FORMat UNSigned command to select the format.

### Determining the Codes in the Unsigned Number Format

For outputs into matched loads and with the amplitude set to maximum (+5.11875V), the following DAC codes generate the following outputs:

- Code **-32768** outputs 0 V
- Code **0** outputs -5.12 V or negative full scale voltage
- Code **-8** outputs +5.11875 V or positive full scale voltage
To calculate combined list codes from NEGATIVE voltage values, use the formula:

\[ \text{DAC Code} = ((\text{voltage value} / 0.00125) \text{ shift left by 3}) + 32768 \]

For example, to output -2V:

\[ \text{DAC Code} = ((-2 / 0.00125) \text{ shift left by 3}) + 32768 = -12800 + 32768 = 19968 \]

To calculate combined list codes from POSITIVE voltage values, use the formula:

\[ \text{DAC Code} = ((\text{voltage value} / 0.00125) \text{ shift left by 3}) - 32768 \]

For example, to output +5V:

\[ \text{DAC Code} = ((5 / 0.00125) \text{ shift left by 3}) - 32768 = 32000 - 32768 = 768 \]

To output a marker at a particular point, add “2” to the combined list DAC code value of the point. For example, to add a marker bit of a point with a voltage value of 5V:

\[ \text{Code} = ((5 / 0.00125) \text{ shift left by 3}) -32768) + 2 = 32000 - 32768 + 2 = 766 \]

The COMBUNS program shows how to store a combined list (i.e., waveform segment and/or marker bit of an arbitrary waveform) into the AFG’s segment memory. The list is stored in the Unsigned number format. The data is transferred to the AFG using the Indefinite Length Arbitrary Block Data method. The example generates a 200 point +5 V to -5 V negative going ramp. A marker is output at the zero crossing (or center) of the ramp.
The commands are the same ones listed on page 241, except on how to select the Unsigned format and how to generate the data. These exceptions are as follows:

5. **Select the DAC Data Format**
   
   ```plaintext
   [SOURce:]ARBitrary:DAC:FORMat UNSi gned
   This command selects the UNSigned data number format.
   ```

8. **Store the Waveform Segment as Combined Signed DAC Data**
   
   ```plaintext
   [SOURce:]LIST[1]:SEGMEMt:COMBined <combined_list>
   This command stores the waveform segment into segment memory in the Unsigned format set by the [SOURce:]ARBitrary:DAC:FORMat UNSigned command. The data is sent as a comma separated combined list with the marker bit selected.
   ```

**HP BASIC Program Example (COMBUNS)**

The COMBUNS program is similar to the “COMBSIGN” program on page 242. The only differences are that this program generates and transfers the combined list using the Unsigned number format instead of the Signed format, and the list is transferred as Indefinite Length Arbitrary Block Data.

```
!RE-STORE"COMBUNS"
1 !This program downloads an arbitrary waveform as a combined
2 !(voltage and marker) list of unsigned DAC codes. The data is sent
3 !in an IEEE-488.2 indefinite length block in 16-bit integer format.
4 !The waveform is a 200 point, +5V to -5V ramp wave.
5 !Assign I/O path between the computer and E1445A.
6 ASSIGN @Afg TO 70910
20 ASSIGN @Afg1 TO 70910;FORMAT OFF !path for binary data
40 COM @Afg,@Afg1
50 !
60 !Set up error checking
70 ON INTR 7 CALL Errmsg
80 ENABLE INTR 7;2
90 OUTPUT @Afg:""CLS"
100 OUTPUT @Afg:""SRE 32"
110 OUTPUT @Afg:""ESE 60"
120 !
130 !Call the subprograms which reset the AFG and erase all waveform
140 !segments and sequences.
150 CALL Rst
160 CALL Wf_del
170 !
```
Continued on Next Page
SUB Wf_del

Wf_del: !Subprogram which deletes all sequences and segments.

COM @Afg,Afg1
OUTPUT @Afg;"FUNC:USER NONE" !select no sequences
OUTPUT @Afg;"LIST:SSEQ:DEL:ALL" !Clear sequence memory
OUTPUT @Afg;"LIST:SEGM:DEL:ALL" !Clear segment memory

SUBEND

SUB Errmsg

Errmsg: !Subprogram which displays E1445 programming errors

COM @Afg,Afg1
DIM Message$[256]
!Read AFG status byte register and clear service request bit
B=SPOLL(@Afg)
!End of statement if error occurs among coupled commands
OUTPUT @Afg;""
OUTPUT @Afg;"ABORT" !abort output waveform
REPEAT
OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
ENTER @Afg;Code,Message$
PRINT Code,Message$
UNTIL Code=0
STOP

SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, COMBUNS.FRM, is in directory "VBPROG" and the Visual C example program, COMBUNS.C, is in directory "VCPROG" on the CD that came with your HP E1445A.

This program sends the combined list using Unsigned data as Indefinite Length Arbitrary Block Data. It is thus very similar to the UNS_DAT program beginning on page 230 and the DACBLOK2 program beginning on page 236.
Using Combined Waveform Segments and Segment Sequences

Using Combined Segment Lists and Combined Segment Sequence Lists to generate arbitrary waveforms is one of the fastest methods to download or transfer waveform segments and segment sequences to the AFG. Both can be downloaded to the AFG either as Definite Length or Indefinite Length Arbitrary Block Data.

The Combined Segment Lists transfers both the arbitrary waveform segment data and marker pulses to the AFG. The lists are sent as 16-bit Integers in either the Signed or Unsigned number format. (See “Using Combined Signed Data” on page 239 for more information.)

The Combined Segment Sequence List selects the waveform segments, enables the marker output, and sets the repetition count for each waveform segment to be output. Each data code in a Combined Segment Sequence List is sent as a 32-bit Integer in the Unsigned number format.

Combined Segment Sequence List Format

Figure 7-2 shows a single 32-bit integer used for a Combined Segment Sequence List. Bits 0 through 16 select the combined or regular waveform segments for output, bit 18 enables the marker output, and bits 20 through 31 sets the repetition count.

A Combined Segment Sequence List determines the order and how often a waveform segment is to be executed. Thus, each waveform segment, marker enable, and repetition count has a unique data code.

32-Bit Combined List that defines the segment lists to be executed, enables the marker, and defines the repetition count for the segment lists.

Figure 7-2. Combined Sequence List Format
Selecting the Waveform Segments

To select a waveform segment, determine the address of the waveform segment and include the address in the Combined Sequence List. Do the following to determine the address:

1. **Select the Waveform Segment**
   Use the [SOURce:LIST[1][SEGMent]:SELect <name>] command, where <name> is the name of the waveform segment to be output.

2. **Get the Selected Waveform Segment Address**
   Use the [SOURce:LIST[1][SEGMent]:ADDRess?] query command to get the address. The address is the start location of the waveform segment in segment memory.

   To use the returned value in the Combined Sequence List, divide the returned value by 8. For example, if the returned value is 2048, the actual address is 2048 / 8 = 256. This is necessary due to the hardware requirements of the AFG.

3. **Add the Address to a Data Value in the Combined Segment Sequence List**

Selecting the Marker Enable

To select the marker enable, add the value of bit 18 to the Data Byte in the Combined Segment Sequence List.

Selecting the Repetition Count

Bits 20 through 31 select the repetition count. Do the following to set the repetition count:

1. **Select the Repetition Count Value**
   The repetition count bit value = 4096 - desired repetition count.
   For example, 2 repetition counts = 4096 - 2 = 4094.

2. **Shift the Repetition Count Value left by 20**

3. **Add the Shifted Repetition Count Value to the Data Byte in the Combined Segment Sequence List**
The COMBSEQ program shows how to transfer multiple Combined Segment Lists (i.e., waveform segments and/or marker bit of an arbitrary waveform) and a Combined Sequence List (waveform segments to be executed, marker enables, and repetition counts) into the AFG’s memory.

The waveform segments are transferred in the Signed number format and transferred as Definite Length Arbitrary Block Data. The segment sequence is transferred as Indefinite Length Arbitrary Block Data in the Unsigned number format.

The example generates two 5 V sine waves and a single 0 V to +5 V triangle wave. A marker is output at the center of the triangle.

The commands are:

1. **Reset the AFG**
   
   `*RST`

2. **Clear the AFG Memory of All Sequence and Segment Data**
   
   `[SOURce:]LIST[1]:SSEQuence:DELeTe:ALL
   [SOURce:]LIST[1]:SEGMent:DELeTe:ALL`

3. **Setup the AFG for Output**
   
   `[SOURce:]FREQuency[1][:CW | :FIXed] <frequency>
   [SOURce:]FUNCTION[:SHAPe] USER
   [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>`

4. **Select the DAC Data Source**
   
   `[SOURce:]ARBitrary:DAC:SOURce INTernal`
   
   This command selects the source that transfers data to the DAC (see “DAC Sources” on page 280). Use INTernal to transfer the data using the [SOURce:]LIST[1] subsystem.
5. **Select the DAC Data Format**
   
   ```
   [SOURce:]ARBitrary:DAC:FORMat SIGNed
   ```
   This command selects the SIGNED data (or UNSIGNED) data format.

6. **Setup the First Waveform Segment**
   
   ```
   [SOURce:]LIST[1][:SEGMent]:SELection <name>
   [SOURce:]LIST[1][:SEGMent]:DEFinition <length>
   ```

7. **Store the First Waveform Segment as Signed Combined Data**
   
   ```
   [SOURce:]LIST[1][:SEGMent]:VOLTage:DAC <voltage_list>
   ```
   This command stores the waveform segment into segment memory in the format set by the [SOURce:]ARBitrary:DAC:FORMat command. The data is sent as Definite Length Arbitrary Block Data (can also be sent as Indefinite Length Arbitrary Block Data).

8. **Setup the Second Waveform Segment**
   
   ```
   [SOURce:]LIST[1][:SEGMent]:SELection <name>
   [SOURce:]LIST[1][:SEGMent]:DEFinition <length>
   ```

9. **Store the Second Waveform Segment as Signed Combined Data**
   
   ```
   [SOURce:]LIST[1][:SEGMent]:VOLTage:DAC <voltage_list>
   ```
   This command stores the waveform segment and marker bit into segment memory in the Signed format set by the [SOURce:]ARBitrary:DAC:FORMat SIGNED command. The data is sent as Definite Length Arbitrary Block Data (can also be sent as Indefinite Length Arbitrary Block Data). In this example, the marker bit is set at the center of the triangle.

10. **Select the First Waveform Segment and Return its Address**
    
    ```
    [SOURce:]LIST[1][:SEGMent]:SELection <name>
    [SOURce:]LIST[1][:SEGMent]:ADDResse?
    ```
    These commands selects the first waveform segment and then returns the address. Divide the address by 8 and store it into the first element of the 32-bit Integer data array that is used to transfer the sequence list to the AFG.

11. **Add the First Waveform Segment’s Repetition Count**
    
    Add the repetition count (number of times the waveform segment is to be executed) of the first waveform segment to the value in the first element of the data array.

12. **Select the Second Waveform Segment and Return its Address**
    
    ```
    [SOURce:]LIST[1][:SEGMent]:SELection <name>
    [SOURce:]LIST[1][:SEGMent]:ADDResse?
    ```
    These commands selects the second waveform segment and then returns the address. Divide the address by 8 and store it into the second element of the data array.
13. **Add the Marker Enable**
Add the value of the marker enable bit of the second waveform segment to the value in the second element of the data array.

14. **Add the Second Waveform Segment’s Repetition Count**
Add the repetition count (number of times the waveform segment is to be executed) of the second waveform segment to the value in the second element of the data array.

15. **Setup the Sequence List**
   
   **[SOURce:]LIST[1]:SSEQUence:SELect <name>**
   **[SOURce:]LIST[1]:SSEQUence:DEFINE <length>**

16. **Store the Segment Sequence as Unsigned Combined Data**
   
   **[SOURce:]LIST[1]:SSEQUence:COMBined <combined_list>**
   This command stores the segment sequence in the data array into sequence memory. The list is in the Unsigned format and sent as Indefinite Length Arbitrary Block Data (can also be sent as Definite Length Arbitrary Block Data).

17. **Generate the Output**
   
   **[SOURce:]FUNCtion:USER <name>**
   **INITiate[:IMMediate]**
Sending the Combined Segment Lists is similar to the other HP BASIC programs in this chapter. However, since HP BASIC does not support 32-Bit Integer variables, sending a Combined Segment Sequence List is done differently.

The Combined Segment Sequence List must be treated in HP BASIC as 2, 16-Bit Integers. The first integer contains the repetition count, marker enable, and the most significant bit (MSB) of the segment address. The second bit contains the rest of the segment address. For example, Figure 7-3 shows two 16-Bit Integers for a combined sequence that contains a waveform segment with an address of 256, the marker enable bit set, and 2 repetition counts.

HP BASIC determines the value for the first integer as follows:

\[
\text{Repetition Count/Marker} = (\text{SHIFT}(4096 - \text{<repetition count>}, -4) + \text{<segment address>} \text{ DIV 65536}) + 4
\]

HP BASIC determines the value for the second integer as follows:

\[
\text{Segment Address} = \text{<segment address> MOD 65536} - 65536 \times (\text{<segment address> MOD 65536} \times 32767 ) \text{ DIV }
\]

DIV returns the integer portion of the Dividend. MOD returns the remainder of the division.

1 !RE-STORE"COMBSEQ"
2 !This program downloads two arbitrary waveforms as combined lists
3 !(voltage and marker) of signed (2’s complement) DAC codes. The
4 !lists are downloaded in definite length arbitrary blocks. The
5 !output sequence is a combined list (repetition count, marker, and
6 !waveform segment address) downloaded in an indefinite length
7 !arbitrary block.
8 !
9 10 !Assign I/O path between the computer and E1445A.
11 ASSIGN @Afg TO 70910
12 ASSIGN @Afg1 TO 70910;FORMAT OFF
13
Continued on Next Page
Continued on Next Page
Tri_wave: Subprogram which computes a triangle wave and downloads the corresponding dac codes as signed numbers (in a definite length block) to segment memory. Marker pulses coincide with the output voltages of Waveform(1024) through Waveform(1033).

```
590 COM @Afg,@Afg1
600 INTEGER Waveform(1:2048) !Calculate triangle wave (dac codes)
610 FOR I=1 TO 1023
620 Waveform(I)=I*.0048828/.00125
630 Waveform(I)=SHIFT(Waveform(I),-3) !shift bits to code positions
640 NEXT I
650 FOR I=1024 TO 1033
660 Waveform(I)=I*.0048828/.00125
670 Waveform(I)=(SHIFT(Waveform(I),-3))+2 !shift bits, set marker bit
680 NEXT I
690 FOR I=1034 TO 2048
700 Waveform(I)=(2048-I)*.0048828/.00125
710 Waveform(I)=SHIFT(Waveform(I),-3) !shift bits to code positions
720 NEXT I
730 !
740 !Output marker as defined by segment and sequence list
750 OUTPUT @Afg;"SOUR:MARK:FEED ""SOUR:LIST1"
760 OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL M2" !segment name
770 OUTPUT @Afg;" SOUR:LIST1:SEGM:DEF 2048" !segment size
780 OUTPUT @Afg USING ",K";" SOUR:LIST1:SEGM:COMB #44096"
790 OUTPUT @Afg1;Waveform(*) ! 4096 bytes: 4 digits (2 bytes/ampl point)
800 OUTPUT @Afg !CR LF
810 SUBEND
```

```
820 !
830 SUB Seq_list
840 Seq_list: !This subprogram downloads the sequence list as a combined (repetition count, marker, segment address) list in an indefinite length arbitrary block.
850 INTEGER Sequence(1:2,1:2)
860 REAL Addrm1,Addrm2
870 OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL M1" !determine segment address
880 OUTPUT @Afg;" SOUR:LIST1:SEGM:DEF 2048"
890 OUTPUT @Afg USING ",K";" SOUR:LIST1:SEGM:COMB #44096"
900 ENTER @Afg;Addrm1
910 Addrm1=Addrm1/4 !/4 to set starting address (boundary) of segment
920 !
930 !Sequence (1,1) is the repetition count and marker enable for segment M1. Sequence (1,2) is the starting address of segment M1.
940 !Sequence(1,1)=SHIFT(4096-2,-4)+Addrm1 DIV 65536
950 !Sequence(1,2)=Addrm1 MOD 65536-65536*(Addrm1 MOD 65536>32767)
960 !
970 OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL M2" !determine segment address
980 OUTPUT @Afg;" SOUR:LIST1:SEGM:DEF 2048"
990 OUTPUT @Afg USING ",K";" SOUR:LIST1:SEGM:COMB #44096"
1000 ENTER @Afg;Addrm2
1010 Addrm2=Addrm2/4 !/4 to set starting address (boundary) of segment
```

Continued on Next Page
!Sequence (2,1) is the repetition count and marker enable for segment M2. Sequence (2,2) is the starting address of segment M2.

Sequence(2,1)=(SHIFT(4096-1,-4)+Addrm2%65536)+4 !enable marker
Sequence(2,2)=Addrm2 MOD 65536-65536*(Addrm2 MOD 65536>32767)

!sequence name
!segments in sequence
!sequence list in indefinite length block
!terminate with Line Feed (LF) and EOI

SUBEND

!Subprogram which resets the E1445.

COM @Afg,Afg1
OUTPUT @Afg;"**RST;*OPC?" !reset the AFG
ENTER @Afg;Complete

SUBEND

!Subprogram which deletes all sequences and segments.

COM @Afg,Afg1
OUTPUT @Afg;"FUNC:USER NONE" !select no sequences
OUTPUT @Afg;"LIST:SSEQ:DEL:ALL" !Clear sequence memory
OUTPUT @Afg;"LIST:SEGM:DEL:ALL" !Clear segment memory

SUBEND

!Subprogram which displays E1445 programming errors

DIM Message$[256]
B=SPOLL(@Afg)
!End of statement if error occurs among coupled commands
OUTPUT @Afg;"
OUTPUT @Afg;"ABORT" !abort output waveform
REPEAT
OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
ENTER @Afg;Code,Message$
PRINT Code,Message$
UNTIL Code=0
STOP

SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, COMBSEQ.FRM, is in directory “VBPROG” and the Visual C example program, COMBSEQ.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Using the VXIbus Backplane

You can use the VXIbus backplane to download or transfer segment and sequence data to the AFG, and to set the phase modulation angle.

**Downloading Segment Data**

There are two ways to use the VXIbus backplane to download the data:

- downloading the list into memory to be executed later
- downloading directly to the DAC for immediate execution

**Downloading Segment Data into Memory**

Download a Combined Segment List and a Combined Sequence List into memory using the requirements in this section.

**Combined Waveform Segment List Format**

Figure 7-4 shows a single 16-bit integer used to download a Combined Waveform Segment List. Bits 3 through 15 are the DAC codes for the waveform voltage values, bit 1 is the marker bit, and bit 0 the last point.

- Store the list either as Signed or Unsigned Combined Segment Lists into memory. Use either Definite Length or Indefinite Length Arbitrary Block Data to store the data. The list uses a 16-bit word for each point of the waveform segment.

- Download the segment data directly into the AFG’s High Speed Data Register. The data must go to the register address with a 38 decimal (26 hex) offset in the AFG’s A24 address space.

- Be sure to set the last point bit (bit 0) in the list. This indicates to the AFG that all the segment data has been transferred. Set the bit at the third-to-last point of the waveform segments (the actual last point = -3). For example, for a Combined list with a size of 2048, set the bit at point number 2048 - 3 = 2045.
Combined Segment Sequence List Format

Figure 7-5 shows a single 32-bit integer used to download a Combined Segment Sequence List. Bits 0 through 16 select the combined waveform segments for output, bit 18 enables the marker output, and bits 20 through 31 sets the repetition count.

32-bit combined list that defines the segment lists to be executed, enables the marker, enables the last point, and defines the repetition count for the segment lists.

- A Combined Segment Sequence List determines the order and how often a waveform segment is to be executed. Thus, each waveform segment, marker enable, and repetition count has a unique data code.

- Select the combined waveform segments using their starting addresses in memory. Add the address to the Combined Segment Sequence List.

- Set bit 18 to enable the marker output for a segment sequence. Add the bit value to the Combined Segment Sequence List.

- Determine the repetition count using: 4096 – the repetition count value. Add the repetition count to the Combined Segment Sequence List.

- Store the list as a 32-bit wide value for each waveform segment in the list. Send the value as two 16-bit words with the most significant bit (MSB) sent first. Download the word with the most significant bit into the AFG’s Sequence Register with a 34 decimal (22 hex) offset in the AFG’s A24 address space. Download the word with the least significant bit into the AFG’s Sequence Register with a 36 decimal (24 hex) offset in the AFG’s A24 address space (see Appendix C for information on registers).
The VXIDOWN program shows how to download multiple Combined Segment Lists (i.e., waveform segment and/or marker bit of an arbitrary waveform) and a single Combined Segment Sequence List (waveform segments to be executed, marker enables, and repetition counts) into the AFG’s memory using the VXIbus backplane.

The combined segment lists are downloaded in the Signed format and as Definite Length Arbitrary Block Data.

The example generates two 5 V sine waves and a single 0 to +5 V triangle wave. A marker is output at the center of the triangle.

The commands are:

1. **Reset the AFG**
   
   `*RST`

2. **Clear the AFG Memory of All Sequence and Segment Data**
   
   `[SOURce:]LIST[1]:SSEQUence:DELete:ALL`
   
   `[SOURce:]LIST[1][:SEGMent]:DELete:ALL`

3. **Setup the AFG for Output**
   
   `[SOURce:]FREQuency[1][:CW | :FIXed] <frequency>`
   
   `[SOURce:]FUNCTION[:SHApe] USER`
   
   `[SOURce:]VOLTage[:LEVEL][:IMMediate][:AMPLitude] <amplitude>`

4. **Select the DAC Data Format**
   
   `[SOURce:]ARBitrary:DAC:FORMat SIGNed`
   
   This command selects the SIGNed (or UNSigned) number format.

5. **Set the Marker Output Source**
   
   `[SOURce:]MARKer:FEED “[SOURce:]LIST[1]”`
   
   This command selects the LIST[1] source as the source that outputs a marker pulse at the “Marker Out” front panel terminals (see Chapter 6 for information on other sources).

6. **Setup the First Combined Segment List**
   
   `[SOURce:]LIST[1][:SEGMent]:SELect <name>`
   
   `[SOURce:]LIST[1][:SEGMent]:DEFine <length>`

7. **Select the Download Source for the First Combined Segment List**
   
   `[SOURce:]ARBitrary:DOWNload <source>,<dest>,<length>`
   
   This command selects the source used to download DAC data into segment memory (see “DAC Sources” on page 280). The `<source>` parameter selects the download source, `<dest>` contains the name of the waveform segment to be downloaded, and `<length>` contains the size of the waveform segment in number of points (i.e., the same size set in `[SOURce:]LIST[1][:SEGMent]:DEFine <length>`).
8. **Place the AFG into Hold Until All Commands are Executed**
   *OPC?*
   This command prevents the AFG from receiving data over the VXIbus until it executes all the previous commands. If *OPC?* is not sent, the AFG will try to receive data, and thus generate an error, even before it completes executing the previous commands.

9. **Generate, Download, and Store the First Waveform Segment as a Combined Signed List**
   This step stores the Combined waveform segment into segment memory using the Signed number format set by the [SOURce:]ARBitrary:DAC:FORMat SIGNed command. The command or downloading method used depends on the device that downloads the data. For example, the device may be an embedded controller. (You can also use the command module, like the HP E1406A Command Module, but at a slower data transfer rate.)
   
   Be sure to set the last point bit at the appropriate point on the waveform.

10. **Notify the AFG that Downloading is Completed**
    [SOURce:]ARBitrary:DOWNload:COMPLETE
    Send this command to the AFG after all data is downloaded.

11. **Setup the Second Combined Segment List**
    [SOURce:]LIST[1][:SEGMENT]:SELection <name>
    [SOURce:]LIST[1][:SEGMENT]:DEFine <length>

12. **Select the Download Source for the Second Segment List**
    [SOURce:]ARBitrary:DOWNload <source>,<dest>,<length>
    This command selects the source used to download DAC data into segment memory (see “DAC Sources” on page 280). The <source> parameter selects the download source, <dest> contains the name of the waveform segment to be downloaded, and <length> contains the size of the waveform segment in number of points (i.e., the same size set in [SOURce:]LIST[1][:SEGMENT]:DEFine <length>).

13. **Place the AFG Into Hold Until All Commands are Executed**
    *OPC?*
    This command prevents the AFG from receiving data over the VXIbus until it executes all the previous commands. If *OPC?* is not sent, the AFG will try to receive data, and thus generate an error, even before it completes executing the previous commands.

14. **Generate, Download, and Store the Second Waveform Segment as a Combined Signed List**
    This step stores the Combined waveform segment into segment memory using the Signed number format set by the [SOURce:]ARBitrary:DAC:FORMat SIGNed command. The command or downloading method used depends on the device that downloads the data. For example, the device may be an embedded
controller. (You can also use the command module, like the HP E1406A Command Module, but at a slower data transfer rate.)

Be sure to set the last point bit and marker bits at the appropriate points on the waveform.

15. Notify the AFG that Downloading is Completed
   [SOURce:]ARBitrary:DOWNload:COMPLETE
   Send this command to the AFG after all data is downloaded.

16. Select the First Waveform Segment and Return its Address
   [SOURce:]LIST[1]:SEGMent:SELect <name>
   [SOURce:]LIST[1]:SEGMent:ADDress?
   These commands select the first waveform segment and then returns its address. Divide the address by 8; store it into the second element of the first 16-bit word array. Add the most significant bit of the segment address to the first element of the first 16-bit word array.

17. Add the First Segment List’s Repetition Count
   Add the repetition count (number of times the waveform segment is to be executed) of the first element of the first 16-bit word array.

18. Select the Second Waveform Segment and Return its Address
   [SOURce:]LIST[1]:SEGMent:SELect <name>
   [SOURce:]LIST[1]:SEGMent:ADDress?
   These commands select the second waveform segment and then returns its address. Divide the address by 8; store it into the second element of the second 16-bit word array. Add the most significant bit of the segment address to the first element of the second 16-bit word array.

19. Add the Marker Enable
   Add the value of the marker enable bit of the second waveform segment to the value in the first element of the second 16-bit word array.

20. Add the Second Segment List’s Repetition Count
   Add the repetition count (number of times the waveform segment is to be executed) of the first element of the second 16-bit word array.

21. Add the Last Point
   Add the value of the last point bit to the first element of the second 16-bit word array.

22. Setup the Sequence List
   [SOURce:]LIST[1]:SSEQUence:SELect <name>
   [SOURce:]LIST[1]:SSEQUence:DEFine <length>

23. Select the Download Source for the Segment Sequence List
   [SOURce:]ARBitrary:DOWNload <source>,<dest>,<length>
   This command selects the source used to download DAC data into segment sequence memory (see “DAC Sources” on page 280). The <source> parameter selects the download source, <dest> contains the
name of the segment sequence list to be downloaded, and \(<length>\) contains the size of the segment sequence list in number of segment lists (i.e., the same size set in \([\text{SOURCE:}]\text{LIST}[1]:\text{SEQUence:DEFine} \ \text{<length}>\)).

24. **Place the AFG into Hold Until All Commands are Executed**
   *OPC?*
   This command prevents the AFG from receiving data over the VXIbus until it executes all the previous commands. If *OPC?* is not sent, the AFG will try to receive data, and thus generate an error, even before it completes executing the previous commands.

25. **Download and Store the Segment Sequence List as a Combined List**
   This step stores the segment sequence list into memory. The command or downloading method used depends on the device that downloads the data. For example, the device may be an embedded controller. (You can also use the command module, like the HP E1406A Command Module, but at a slower data transfer rate.)

26. **Notify the AFG that Downloading is Completed**
   \([\text{SOURCE:}]\text{ARBITrary:DOWNLOAD:COMPLETE}\)
   Send this command to the AFG after all data is downloaded.

27. **Generate the Output**
   \(\text{INITiate:IMMediate}\)

**HP BASIC Program Example (VXIDOWN)**

This program is similar to the COMBSEQ program beginning on page 255, except on how the data is transferred to the AFG. The program uses a V360 Controller to download the data using the VXIbus instead of transferring it directly to the AFG using HP-IB.

```basic
1 !RE-STORE"VXIDOWN"
2 !This program downloads two arbitrary waveforms from the VXIbus
3 !backplane. The program loads segment memory by writing to the
4 !AFG's high-speed data register, and loads sequence memory by
5 !writing to the Sequence register. The program is written for a
6 !HP E1480 V/360 embedded controller, which allows direct access to
7 !the registers via the VXIbus.
8 !
9 !Assign I/O path between the computer and E1445A.
10 ASSIGN @Afg TO 1680
11 COM @Afg,Base_addr
12 !
13 !Set up error checking for the SCPI commands.
14 ON INTR 16 CALL Errmsg
15 ENABLE INTR 16;32
16 !
17 !RE-STORE"VXIDOWN"
```

*Continued on Next Page*
90 OUTPUT @Afg;"**SRE 32"
100 OUTPUT @Afg;"**ESE 60"
110 !
120 !Call the subprograms
130 CALL Rst
140 CALL Wf_del
150 CALL A24_offset
160 !
170 OUTPUT @Afg;"SOUR:FREQ1:FIX 2.048E6;"; !frequency
180 OUTPUT @Afg;"SOUR:FUNC:SHAP USER;"; !function
190 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5.11875V" !amplitude
200 OUTPUT @Afg;"SOUR:ARB:DAC:FORM SIGN" !dac data format (signed)
210 OUTPUT @Afg;"SOUR:MARK:FEED ""SOUR:LIST1""" !marker pulse source
220 !
230 CALL Sine_wave
240 CALL Tri_wave
250 CALL Seq_list
260 !
270 OUTPUT @Afg;"SOUR:FUNC:USER WAVE_OUT" !waveform sequence
280 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
290 !
300 WAIT .1 !allow interrupt to be serviced
310 OFF INTR 16
320 END
330 !
340 SUB Sine_wave
350 Sine_wave: !Subprogram which computes a sine wave and downloads
360 !the corresponding dac codes to segment memory over the
370 !VXIbus. A combined list is used but no marker pulse is
380 !specified.
390 COM @Afg,Base_addr
400 CONTROL 16,25,3 !access A24 space with WRITEIO
410 OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL SINE" !segment name
420 OUTPUT @Afg;"SOUR:LIST1:SEGM:DEF 2048" !segment size
430 !
440 INTEGER Waveform(1:2048)
450 !Calculate sine wave (dac codes) and shift bits to dac code positions
460 FOR I=1 TO 2048
470 Waveform(I)=5.*(SIN(2.*PI*(I/2048.)))/.00125
480 Waveform(I)=SHIFT(Waveform(I),-3)
490 NEXT I
500 !Set last point bit (actual last point - 3)
510 Waveform(2045)=Waveform(2045)+1
520 !
530 !Enable downloading from the VXIbus
540 OUTPUT @Afg;"ARB:DOWN VXI,SINE,2048"
550 OUTPUT @Afg;"**OPC?"
560 ENTER @Afg:Ready
570 !

Continued on Next Page
!Download the waveform segment to segment memory using WRITEIO and
!the AFG's high-speed data register. The register's address is
!located in A24 address space.
FOR I=1 TO 2048
  WRITEIO -16,Base_addr+IVAL("26",16);Waveform(I)
  NEXT I
OUTPUT @Afg;"SOUR:ARB:DOWN:COMP" !disable downloading
SUBEND

SUB Tri_wave
Tri_wave: !Subprogram which computes a triangle wave and downloads
!the corresponding dac codes to segment memory over the
!VXIbus. Marker pulses coincide with the output voltages
!Waveform(1024) through Waveform(1033).
COM @Afg,Base_addr
CONTROL 16,253 !access A24 space with WRITEIO
OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL TRI" !segment name
OUTPUT @Afg;" SOUR:LIST1:SEGM:DEF 2048" !segment size
!
!Calculate triangle wave (dac codes) and shift bits to code positions
INTEGER Waveform(1:2048)
FOR I=1 TO 1023
  Waveform(I)=I*.0048828/.00125
  Waveform(I)=SHIFT(Waveform(I),-3)
  NEXT I
FOR I=1024 TO 1033
  Waveform(I)=I*.0048828/.00125
  Waveform(I)=SHIFT(Waveform(I),-3)+2 !include marker bit
  NEXT I
FOR I=1034 TO 2048
  Waveform(I)=(2048-I)*.0048828/.00125
  Waveform(I)=SHIFT(Waveform(I),-3)
  NEXT I
!
!Set last point bit (actual last point - 3)
Waveform(2045)=Waveform(2045)+1
!
!Enable downloading from the VXIbus
OUTPUT @Afg;"ARB:DOWN VXI,TRI,2048"
OUTPUT @Afg;"*OPC?"
ENTER @Afg;Ready
!
!Download the waveform segment to segment memory using WRITEIO and
!the AFG's high-speed data register. The register's address is
!located in A24 address space.
FOR I=1 TO 2048
  WRITEIO -16,Base_addr+IVAL("26",16);Waveform(I)
  NEXT I
OUTPUT @Afg;"SOUR:ARB:DOWN:COMP" !disable downloading
SUBEND

Continued on Next Page
1080 !
1090 SUB Seq_list
1100 Seq_list: !This subprogram downloads the sequence list (repetition
1110 !count, marker, segment address) to sequence memory over
1120 !the VXIbus.
1130 INTEGER Sequence(1:2,1:2)
1140 REAL Addrm1, Addrm2
1150 COM @Afg, Base_addr
1160 CONTROL 16,25,3!access A24 space with WRITEIO
1170 OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL SINE" !determine segment address
1180 OUTPUT @Afg;"SOUR:LIST1:SEGM:ADDR?"
1190 ENTER @Afg; Addrm1
1200 Addrm1=Addrm1/8 !/8 to set starting address (boundary) of segment
1210 !
1220 !Sequence (1,1) is the repetition count and marker enable for
1230 !segment SINE. Sequence (1,2) is the starting address of segment SINE.
1240 Sequence(1,1)=(SHIFT(4096-2,-4)+Addrm1 DIV 65536)
1250 Sequence(1,2)=(Addrm1 MOD 65536-65536*(Addrm1 MOD 65536>32767))
1260 !
1270 OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL TRI" !determine segment address
1280 OUTPUT @Afg;"SOUR:LIST1:SEGM:ADDR?"
1290 ENTER @Afg; Addrm2
1300 Addrm2=Addrm2/8 !/8 to set starting address (boundary) of segment
1310 !
1320 !Sequence (2,1) is the repetition count, marker enable, and last point
1330 !indication for the segment sequence. Sequence (2,2) is the starting
1340 !address of segment TRI.
1350 Sequence(2,1)=(SHIFT(4096-1,-4)+Addrm2 DIV 65536)+12
1360 Sequence(2,2)=Addrm2 MOD 65536-65536*(Addrm2 MOD 65536>32767)
1370 !
1380 OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEL WAVE_OUT" !sequence name
1390 OUTPUT @Afg;"SOUR:LIST1:SSEQ:DEF 2" !segments in sequence
1400 OUTPUT @Afg;"SOUR:ARB:DOWN VXI,WAVE_OUT,2"
1410 OUTPUT @Afg;"*OPC?"
1420 ENTER @Afg;Ready
1430 !
1440 !Download the waveform sequence to sequence memory using WRITEIO
1450 !and the AFG's Sequence register. The register's address is
1460 !located in A24 address space.
1470 WRITEIO -16,Base_addr+IVAL("22",16);Sequence(1,1) !16 MS Bits
1480 WRITEIO -16,Base_addr+IVAL("24",16);Sequence(1,2) !16 LS Bits
1490 WRITEIO -16,Base_addr+IVAL("22",16);Sequence(2,1) !16 MS Bits
1500 WRITEIO -16,Base_addr+IVAL("24",16);Sequence(2,2) !16 LS Bits
1510 OUTPUT @Afg;"SOUR:ARB:DOWN:COMP" !disable downloading
1520 SUBEND
1530 !
1540 SUB A24_offset
1550 A24_offset:!Subprogram which determines the base address for
1560 !the AFG registers in A24 address space.
1570 COM @Afg, Base_addr

Continued on Next Page
CONTROL 16,25;2 !access A16 space with READIO and WRITEIO
A16_addr=DVAL("D400",16) !AFG A16 base address
Offset=READIO(-16,A16_addr+6) !read AFG offset register
Base_addr=Offset*256 !shift offset for 24-bit address
SUBEND

SUB Rst
Rst: !Subprogram which resets the E1445.
COM @Afg,Base_addr
OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
ENTER @Afg;Complete
SUBEND

SUB Wf_del
Wf_del: !Subprogram which deletes all sequences and segments.
COM @Afg,Base_addr
OUTPUT @Afg;"FUNC:USER NONE" !select no sequences
OUTPUT @Afg;"LIST:SSEQ:DEL:ALL" !Clear sequence memory
OUTPUT @Afg;"LIST:SEGM:DEL:ALL" !Clear segment memory
SUBEND

SUB Errmsg
Errmsg:!Subprogram which displays E1445 programming errors
COM @Afg,Base_addr
DIM Message$[256]
!Read AFG status byte register and clear service request bit
B=SPOLL(@Afg)
! End of statement if error occurs among coupled commands
OUTPUT @Afg;"" !abort output waveform
REPEAT
OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
ENTER @Afg;Code,Message$
PRINT Code,Message$
UNTIL Code=0
STOP
SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, VXIDOWN.FRM, is in directory “VBPROG” and the Visual C example program, VXIDOWN.C, is in directory “VCPROG” on the CD that came with your HP E1445A.

The above example programs use the HP E1406A Command Module to download the data into memory. However, the command module is only used to demonstrate the downloading method for those programs. A better method is to use an embedded controller. If you wish to use the HP E1406A Command Module to download data, use the method described in “Using Combined Waveform Segments and Segment Sequences” on page 250.
This method disables the AFG’s ARM subsystem and immediately outputs the DAC data point when received. The DAC code received by the AFG only sets the DAC to output to the received value. It thus does not disable the AFG’s DAC code format, triggering, marker selection and enabling, and amplitude setting. Send the DAC codes as Combined lists.

- The lists can be downloaded either in the Signed or Unsigned number formats, and as Definite Length or Indefinite Length Arbitrary Block Data.
- Since the AFG stores no data into memory, do not set the last point bit in the list.
- Download the segment data directly into the AFG’s High Speed Data Register. The data must go to the register address with a 38 decimal (26 hex) offset in the AFG’s A24 address space (see Appendix C for information on registers).

The VXISRCE program shows how to download segment data directly to the DAC. The program downloads the lists using the VXIbus.

The segment lists are downloaded in the Signed number format and as Indefinite Length Arbitrary Block Data. The example generates a 0 to +5 V triangle wave. The frequency of the triangle depends on the speed at which downloading occurs. The commands are:

1. **Reset the AFG**
   
   `*RST`

2. **Set the AFG’s Output Amplitude**
   
   `[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>`

3. **Select the DAC Data Format**
   
   `[SOURce:]ARBitrary:DAC:FORMat SIGNed`
   
   This command selects the SIGNed (or UNSigned) number format.

4. **Select the DAC Data Source**
   
   `[SOURce:]ARBitrary:DAC:SOURce VXI`
   
   This command selects the source that transfers data to the DAC (see “DAC Sources” on page 280). Use “VXI” to transfer data using the VXIbus.

5. **Place the AFG Into Hold Until All Commands are Executed**
   
   `*OPC?`
   
   This commands prevents the AFG from receiving data over the VXIbus until it executes all the previous commands. If `*OPC?` is not sent, the AFG will try to receive data, and thus generate an error, even before it completes executing the previous commands.

6. **Download the Waveform Segment as a Combined Signed List**
   
   This step directly downloads the Combined Waveform Segment List to the DAC using the Signed number format set by the
ARBITRARY:DAC:FORM SIGN command. The downloading method used depends on the device that downloads the data. For example, the device may be an embedded controller or a command module. The AFG output depends on the data received by the DAC and the currently selected amplitude.

HP BASIC Program Example (VXISRCE)

The program uses the V360 Controller to download the data using the VXIbus instead of transferring it directly to the AFG using HP-IB.

1 !RE-STORE "VXISRCE"
2 !This program uses the V/360 embedded controller to send waveform data directly to the AFG dac over the VXIbus backplane.
3 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 1680
30 COM @Afg,Addr
40 !
50 !Call the subprograms which reset the AFG and determine the base address of the registers in A24 address space.
60 !
70 CALL Rst
80 CALL A24_offset
90 !
100 !Scale the amplitude, set the dac data format and dac data source.
110 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5.11875V" !amplitude
120 OUTPUT @Afg;"SOUR:ARB:DAC:FORM SIGN" !dac data format (signed)
130 OUTPUT @Afg;"SOUR:ARB:DAC:SOUR VXI" !dac data source
140 OUTPUT @Afg;"*OPC?" !Wait for the SCPI commands to complete
150 ENTER @Afg;Complete
160 !
170 !Call the subprogram which sends data directly to the dac.
180 CALL Dac_drive
190 END
200 !
210 SUB A24_offset
220 A24_offset: !Subprogram which determines the base address for the AFG registers in A24 address space, then adds the offset and register number to the base to get the complete address.
230 !
240 !
250 !
260 COM @Afg,Addr
270 !CONTROL 16,25:2 !access A16 space with READIO and WRITEIO
280 A16_addr=DVAL("D400",16) !AFG A16 base address
290 Offset=READIO(-16,A16_addr+6) !read AFG offset register
300 Base_addr=Offset*256 !shift offset for 24-bit address
310 !Add the register number of the high speed data register to the A24 base address.
320 Addr=Base_addr+IVAL("26",16)
340 SUBEND

Continued on Next Page
!SUB Dac_drive

Dac_drive: !Subprogram which computes a 128 point, 5 Vpp triangle wave and writes the corresponding codes directly to the DAC via the VXIbus and High Speed Data register.

COM @Afg,Addr
!CONTROL 16,25;3 !access A24 space with WRITEIO
!
INTEGER I,Waveform(1:128) !Calculate triangle wave (dac codes)
FOR I=1 TO 64
    Waveform(I)=I*.0755/.00125
    Waveform(I)=SHIFT(Waveform(I),-3) !shift bits to dac code positions
    NEXT I
FOR I=65 TO 128
    Waveform(I)=(128-I)*.0755/.00125
    Waveform(I)=SHIFT(Waveform(I),-3) !shift bits to dac code positions
    NEXT I
!
!Continuously write data (in 16-bit words) to the dac via the VXIbus and High Speed Data register.
LOOP
FOR I=1 TO 128
    WRITEIO -16,Addr;Waveform(I)
    NEXT I
END LOOP
SUBEND

!SUB Rst
Rst: !Subprogram which resets the E1445.
COM @Afg,Addr
OUTPUT @Afg;"*RST;"OPC?" !reset the AFG
ENTER @Afg;Complete
SUBEND

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, VXISRCE.FRM, is in directory “VBPROG” and the Visual C example program, VXISRCE.C, is in directory “VCPROG” on the CD that came with your HP E1445A.

The above example programs use the HP E1406A Command Module to download the data to the DAC. However, the command module is only used to demonstrate the downloading method for those programs. A better method is to use an embedded controller.
Using the Front Panel’s “Digital Port In” Connector

You can use the “Digital Port In” connector to download data to the segment memory ([SOURce:]ARBitrary:DOWnload command), to change segment sequences without aborting the present operation, or to drive the DAC directly ([SOURce:]ARBitrary:DAC:SOURce command).

The WAVSELFp program selects three different sequences using the “Digital Port In” connector. Sequence 1 is a Sin (X)/X waveform, sequence 2 is a damped sine waveform, and sequence 3 is a sine wave with spikes waveform. The program downloads segment data as indefinite length arbitrary block data using the [SOURce]:LIST[1]:SEGMen[1]:COMBined command. Select the sequences as follows:

FPCLK is clocked, other data lines open – Sequence 3
FPCLK is clocked, FP000 to low – Sequence 2
FPCLK is clocked, FP001 to low – Sequence 1

HP BASIC Program Example (WAVSELFp)

```hp-basic
1 !RE-STORE "WAVSELFp"
2 !This program changes the output waveform sequence once the AFG has been
3 !INITiated by writing the location of a sequence’s base address to the
4 !Waveform Select register. All register reads and writes are 16 bit.
5 !The program uses the front panel ‘Digital Port In’ connector to
6 !change the sequences, as follows:
7 !FPCLK is clocked, other data lines open – Sequence 3
8 !FPCLK is clocked, FP000 to low – Sequence 2
9 !FPCLK is clocked, FP001 to low – Sequence 1
10 !
20 !Assign an I/O path between the computer and the AFG
30 ASSIGN @Cmd TO 80900
40 ASSIGN @Afg TO 80910
50 ASSIGN @Afg1 TO 80910;FORMAT OFF !path for binary data
60 Laddr=80 !logical address for AFG
70 COM @Cmd,@Afg,@Afg1,Base_addr,Seq1_addr,Seq2_addr,Seq3_addr
80 !
90 !Subprograms which reset the AFG and erase all existing waveforms.
100 CALL Rst
110 CALL Wf_del
120 !
130 !SCPI commands which configure the AFG
140 OUTPUT @Afg;"SOUR:FREQ1:FIX 4.096E6;"; !Sample rate
150 OUTPUT @Afg;"SOUR:FUNC:SHAP USER;"; !function
160 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 2.1V" !amplitude
170 OUTPUT @Afg;"SOUR:ARB:DATC:SOUR INT" !dac data source
180 OUTPUT @Afg;"SOUR:ARB:DATC:FORM SIGN" !dac data format
190 !

Continued on Next Page
```
Subprograms which define waveforms and load them into segment
and sequence memory, which determine the AFG’s register locations
in A24, and which configure the AFG’s sequence base memory.

CALL Waveform_def
CALL A24_offset(Laddr)
CALL Build_ram

Select an output sequence, and initiate (start) waveform output.
OUTPUT @Afg;"SOUR:FUNC:USER SEQ1"
OUTPUT @Afg;"INIT:IMM"

!Enable FP DPORT to control sequence selection
OUTPUT @Cmd;"DIAG:PEEK? ",Base_addr+8,"",16"
ENTER @Cmd;Traffic
Traffic=BINOR(BINAND(Traffic,IVAL("3FFF",16)),IVAL("4000",16))
OUTPUT @Cmd;"DIAG:POKE ",Base_addr+8,"",16,";Traffic
END

SUB Waveform_def
COM @Cmd,@Afg,@Afg1,Base_addr,Seq1_addr,Seq2_addr,Seq3_addr
CALL Sinx_def
CALL Sind_def
CALL Spike_def
SUBEND

SUB A24_offset(Laddr)
A24_offset: !Subprogram which determines the base address for
the AFG registers in A24 address space.
COM @Cmd,@Afg,@Afg1,Base_addr,Seq1_addr,Seq2_addr,Seq3_addr
OUTPUT @Cmd;"DIAG:PEEK? ",;DVAL("1FC000",16)+64*Laddr+6,"",16"
ENTER @Cmd;Traffic
Traffic=BINOR(BINAND(Traffic,IVAL("3FFF",16)),IVAL("4000",16))
OUTPUT @Cmd;"DIAG:POKE ",;Base_addr+8,"",16,";Traffic
Base_addr=Offset*256 !shift offset for 24-bit address
SUBEND

SUB Build_ram
Build_ram: !This subprogram configures the AFG’s sequence base memory
such that there are valid sequence base addresses in memory
before the AFG is INITiated and waveforms are selected.
COM @Cmd,@Afg,@Afg1,Base_addr,Seq1_addr,Seq2_addr,Seq3_addr
ENTER @Cmd;Traffic
Traffic=BINOR(BINAND(Traffic,IVAL("3FFF",16)),IVAL("8000",16))
Continued on Next Page
700 OUTPUT @Cmd;"DIAG:POKE ",;Base_addr+8",";Traffic
710 !
720 !Write the location of the sequence base address (waveform index)
730 !to the Waveform Select register. Write the base address of
740 !of the sequence in sequence memory to the Sequence Base register.
750 !
760 OUTPUT @Cmd;"DIAG:POKE ",;Base_addr+10",";8,252"
770 OUTPUT @Cmd;"DIAG:POKE ",;Base_addr+32",";16",";Seq3_addr
780 !
790 OUTPUT @Cmd;"DIAG:POKE ",;Base_addr+10",";8,253"
800 OUTPUT @Cmd;"DIAG:POKE ",;Base_addr+32",";16",";Seq1_addr
810 !
820 OUTPUT @Cmd;"DIAG:POKE ",;Base_addr+10",";8,254"
830 OUTPUT @Cmd;"DIAG:POKE ",;Base_addr+32",";16",";Seq2_addr
840 !
850 OUTPUT @Cmd;"DIAG:POKE ",;Base_addr+10",";8,255"
860 OUTPUT @Cmd;"DIAG:POKE ",;Base_addr+32",";16",";Seq3_addr
870 !
880 OUTPUT @Cmd;"DIAG:POKE ",;Base_addr+10",";8,0"
890 SUBEND
900 !
910 SUB Sinx_def
920 Sinx_def: !Define the waveform Sin(x)/x. Download the waveform data
930 !has a combined list (voltage and marker) of signed numbers
940 !an an indefinite length block. Download the sequence as a
950 !an combined list (repetition count, marker, and segment address)
960 !an indefinite length arbitrary block.
970 COM @Cmd,@Afg,@Afg1,Base_addr,Seq1_addr,Seq2_addr,Seq3_addr
980 INTEGER Waveform(1:4096)
990 INTEGER Sequence(1:2)
1000 REAL Addr_seg1
1010 FOR I=-2047 TO 2048
1020 IF I=0 THEN I=1.E-38
1030 Waveform(I+2048)=((SIN(2*PI*.53125*I/256))/(.53125*I/256)*.159154943092)/.00125
1040 !shift bits to dac code positions
1050 Waveform(I+2048)=SHIFT(Waveform(I+2048),-3)
1060 NEXT I
1070 !
1080 OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL SIN_X" !segment name
1090 OUTPUT @Afg;"SOUR:LIST1:SEGM:DEF 4096" !segment size
1100 OUTPUT @Afg USING ",K","SOUR:LIST1:SEGM:COMB #0" !waveform points
1110 OUTPUT @Afg1:Waveform(*) !indefinite length block
1120 OUTPUT @Afg;CHR$(10);END !terminate with line feed (LF) and EOI
1130 !
1140 OUTPUT @Afg;"SOUR:LIST1:SEGM:ADDR?"
1150 ENTER @Afg;Addr_seg1
1160 Addr_seg1=Addr_seg1/8 !/8 to set starting address (boundary) of segment
1170 !
1180 !Sequence (1) is the repetition count and marker enable for
1190 !segment SIN_X. Sequence (2) is the starting address of segment SIN_X.

Continued on Next Page
1200 Sequence(1)=SHIFT(4096-1,-4)+Addr_seg1 DIV 65536
1210 Sequence(2)=Addr_seg1 MOD 65536*(Addr_seg1 MOD 65536.32767)
1220
1230 OUTPUT @Af;"SOUR:LIST1:SSEQ:SEL SEQ1" !sequence name
1240 OUTPUT @Af;"SOUR:LIST1:SSEQ:DEF 1" !sequence size
1250 OUTPUT @Af USING "#,K";"SOUR:LIST1:SSEQ:COMB #0" !segment execution order
1260 OUTPUT @Af1;Sequence(*) !sequence list in indefinite length block
1270 OUTPUT @Af;CHR$(10);END !terminate with Line Feed (LF) and EOI
1280
1290 OUTPUT @Af;"SOUR:LIST1:SSEQ:ADDR?" !sequence location
1300 ENTER @Af;Seq1_addr
1310 SUBEND
1320
1330 SUB Sind_def
1340 Sind_def: !Compute the damped sine waveform. Download the data
1350 !as a combined list (voltage and marker) of signed numbers
1360 !an indefinite length block. Download the sequence as a
1370 !combined list (repetition count, marker, and segment address)
1380 !an indefinite length arbitrary block.
1390 COM @Cmd,@Af1,Base_addr,Seq1_addr,Seq2_addr,Seq3_addr
1400 INTEGER Waveform(1:4096)
1410 INTEGER Sequence(1:2)
1420 REAL Addr_seg2
1430 A=4/4096
1440 W=(2*PI)/50
1450 FOR T=1 TO 4096
1460 Waveform(T)=EXP(-A*T)*SIN(W*T)/.00125
1470 !shift bits to dac code positions
1480 Waveform(T)=SHIFT(Waveform(T),-3)
1490 NEXT T
1500
1510 OUTPUT @Af;"SOUR:LIST1:SEGM:SEL SIN_D" !segment name
1520 OUTPUT @Af;"SOUR:LIST1:SEGM:DEF 4096" !segment size
1530 OUTPUT @Af USING "#,K";"SOUR:LIST1:SEGM:COMB #0" !waveform points
1540 OUTPUT @Af1;Waveform(*) !indefinite length block
1550 OUTPUT @Af;CHR$(10);END !terminate with line feed (LF) and EOI
1560
1570 OUTPUT @Af;"SOUR:LIST1:SEGM:ADDR?" !segment address
1580 ENTER @Af;Addr_seg2
1590 Addr_seg2=Addr_seg2/8 !/8 to set starting address (boundary) of segment
1600
1610 !Sequence (1) is the repetition count and marker enable for
1620 !segment SIN_D. Sequence (2) is the starting address of segment SIN_D.
1630 Sequence(1)=SHIFT(4096-1,-4)+Addr_seg1 DIV 65536
1640 Sequence(2)=Addr_seg2 MOD 65536-65536*(Addr_seg2 MOD 65536.32767)
1650
1660 OUTPUT @Af;"SOUR:LIST1:SSEQ:SEL SEQ2" !sequence name
1670 OUTPUT @Af;"SOUR:LIST1:SSEQ:DEF 1" !sequence size
1680 OUTPUT @Af USING "#,K";"SOUR:LIST1:SSEQ:COMB #0"!segment execution order
1690 OUTPUT @Af1;Sequence(*) !sequence list in indefinite length block

Continued on Next Page
1700 OUTPUT @Afg;CHR$(10);END  !terminate with Line Feed (LF) and EOI
1710 !
1720 OUTPUT @Afg;"SOUR:LIST1:SSEQ:ADDR?"  !sequence location
1730 ENTER @Afg;Seq2_addr
1740 SUBEND
1750 !
1760 SUB Spike_def
1770 Spike_def: !Compute the waveform (sine wave with spike). Download the
1780  !data as a combined list (voltage and marker) of signed
1790  !numbers in an indefinite length block. Download the sequence as
1800  !a combined list (repetition count, marker, and segment address)
1810  !in an indefinite length arbitrary block.
1820 COM @Cmd,@Afg,@Afg1,Base_addr,Seq1_addr,Seq2_addr,Seq3_addr
1830 INTEGER Waveform(1:4096)
1840 INTEGER Sequence(1:2)
1850 REAL Addr_seg3
1860 FOR I=1 TO 4096
1870 Waveform(I)=SIN(2*PI*(I/4096))/.00125
1880 NEXT I
1890 Width=50
1900 FOR J=1 TO Width
1910 I=1024-Width+J
1920 Waveform(I)=Waveform(I)+.9*J/Width/.00125
1930 NEXT J
1940 FOR J=1 TO Width-1
1950 I=1024+Width-J
1960 Waveform(I)=Waveform(I)+.9*J/Width/.00125
1970 NEXT J
1980 !
1990 !shift bits to dac code positions
2000 FOR I=1 TO 4096
2010 Waveform(I)=SHIFT(Waveform(I),-3)
2020 NEXT I
2030 !
2040 OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL SPIKE"  !segment name
2050 OUTPUT @Afg;"SOUR:LIST1:SEGM:DEF 4096"  !segment size
2060 OUTPUT @Afg USING ",K;";"SOUR:LIST1:SEGM:COMB #0"!waveform points
2070 OUTPUT @Afg1;Waveform(*)  !indefinite length block
2080 OUTPUT @Afg;CHR$(10);END  !terminate with line feed (LF) and EOI
2090 !
2100 OUTPUT @Afg;"SOUR:LIST1:SEGM:ADDR?"
2110 ENTER @Afg;Addr_seg3
2120 Addr_seg3=Addr_seg3/8  !/8 to set starting address (boundary) of segment
2130 !
2140 !Sequence (1) is the repetition count and marker enable for
2150 !segment SPIKE. Sequence (2) is the starting address of segment SPIKE.
2160 Sequence(1)=SHIFT(4096-1,-4)+Addr_seg3 DIV 65536
2170 Sequence(2)=Addr_seg3 MOD 65536..65536.*(Addr_seg3 MOD 65536.32767)
2180 !
2190 OUTPUT @Afg;"SOUR:LIST1:SSEQ:SEL SEQ3"  !sequence name

Continued on Next Page
Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, WAVSELF.FRM, is in directory “VBPROG” and the Visual C example program, WAVSELF.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Figure 7-6 shows a pinout of the “Digital Port In” connector.

**Table: “Digital Port In” Connector Pinout**

<table>
<thead>
<tr>
<th>PIN</th>
<th>SIGNAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FPD00</td>
</tr>
<tr>
<td>2</td>
<td>FPD01</td>
</tr>
<tr>
<td>3</td>
<td>FPD03</td>
</tr>
<tr>
<td>4</td>
<td>FPD04</td>
</tr>
<tr>
<td>5</td>
<td>FPD06</td>
</tr>
<tr>
<td>6</td>
<td>FPD07</td>
</tr>
<tr>
<td>7</td>
<td>GND</td>
</tr>
<tr>
<td>8</td>
<td>FPD10</td>
</tr>
<tr>
<td>9</td>
<td>FPD11</td>
</tr>
<tr>
<td>10</td>
<td>FPD13</td>
</tr>
<tr>
<td>11</td>
<td>FPD14</td>
</tr>
<tr>
<td>12</td>
<td>FPCLK*</td>
</tr>
<tr>
<td>13</td>
<td>FPPAGE*</td>
</tr>
<tr>
<td>14</td>
<td>GND</td>
</tr>
<tr>
<td>15</td>
<td>FPD02</td>
</tr>
<tr>
<td>16</td>
<td>GND</td>
</tr>
<tr>
<td>17</td>
<td>FPD05</td>
</tr>
<tr>
<td>18</td>
<td>GND</td>
</tr>
<tr>
<td>19</td>
<td>FPD08</td>
</tr>
<tr>
<td>20</td>
<td>FPD09</td>
</tr>
<tr>
<td>21</td>
<td>GND</td>
</tr>
<tr>
<td>22</td>
<td>FPD12</td>
</tr>
<tr>
<td>23</td>
<td>GND</td>
</tr>
<tr>
<td>24</td>
<td>FPD15</td>
</tr>
<tr>
<td>25</td>
<td>GND</td>
</tr>
</tbody>
</table>

**Figure 7-6. HP E1445A “Digital Port In” Connector**
Using the “Digital Port In” Connector to Select a Sequence

Figure 7-7 shows the timing relationship to select a segment sequence.

![Figure 7-7. "Digital Port In" Data Timing](image)

The following explains the relationship.

1. The AFG generates an **FPPACE** signal after the next segment sequence has been selected, but before completing its output.

2. The AFG is now ready for a new segment sequence. Set the **FPCLK** line low to select a new segment sequence. If the line remains high, the AFG re-uses the last selected sequence.

3. The AFG now latches the least significant byte on the **FPDxx** (i.e., FPD00 through FPD15) data lines to select a new sequence. These data lines contain the address of the sequence to be selected.

4. Once the AFG uses the data to select a new sequence, it generates a new **FPPACE** signal and the process completes.

For correct operation, the **FPCLK** should occur 150 nS before the next **FPPACE** occurs. The minimum response delay \( (t_r) \) is 0 as is also the minimum data hold time \( (t_h) \). The minimum **FPPACE** pulse width \( (t_p) \) is 20 nS; its width depends on the selected sample rate.

To change the sequence, the sequence base memory must be loaded. See the “WAVSELFPP” example program beginning on page 272 to determine how to load the memory with the sequences.

Using the “Digital Port In” Connector to Download Data

To download data, ignore the **FPPACE** line but provide a **FPCLK** for each data point to be downloaded. The timing relationship between **FPCLK** and **FPDxx** is as shown in Figure 7-7, except without the **FPPACE** line. The data format is the same that is used to download segment data using the [SOURce:]ARBitrary:DOWNload command.
High Speed Operation Program Comments

The following comments give additional details on the program examples in this chapter.

Amplitude Effects on DAC Codes

The AFG stores the Signed or Unsigned DAC codes directly into memory. Thus, the amplitude setting has no affect on the codes. Unlike sending a voltage list, the output amplitude can be set to any of the values listed in Appendix B. The amplitude does not have to be ≥ to the maximum DAC code value.

Incorrect AFG Operation from Incorrect DAC Codes

The AFG requires that the data it receives must be correct, or it will not execute it correctly. Unlike using other data transfer methods, the AFG does not perform any error checking on the data when it is directly downloaded.

DAC Sources

The AFG has the following DAC sources available to download data to the DAC:

- **DPORt** – The front panel’s “Digital Port In” connector
- **LBUS** – The VXIbus Local Bus
- **VXI** – The VXIbus backplane

Download Sources

The AFG has the following sources available to download waveform segments and segment sequences into memory:

- **DPORt** – The front panel’s “Digital Port In” connector.
- **LBUS** – The VXIbus Local Bus.
- **VXI** – The VXIbus backplane.

Determining the Size of the Combined Segment List

Use [SOURce:]LIST[1]:SEGment:COMBined:POINts? to determine the size of the number of points of the waveform segment and marker pulse list of the currently selected waveform segment.

Determining the Size of the Combined Segment Sequence List

Use [SOURce:]LIST[1]:SSEQuence:COMBined:POINts? to determine the size of the number of waveform segments, marker pulse enable lists, and repetition count lists of the currently selected segment sequence.
Chapter Contents

This chapter describes the Standard Commands for Programmable Instruments (SCPI) command set and the IEEE 488.2 Common Commands for the HP E1445A Arbitrary Function Generator (AFG). Included in this chapter are the following sections:

- Command Types ........................................... Page 284
- SCPI Command Format ...................................... Page 284
- SCPI Command Parameters .................................. Page 286
- SCPI Command Execution .................................... Page 288
- SCPI Command Reference .................................... Page 289
- SCPI Command Quick Reference ............................ Page 409
- SCPI Conformance Information .............................. Page 414
- IEEE 488.2 Common Commands ............................. Page 416
- Common Commands Quick Reference ........................ Page 428

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORt</td>
<td>290</td>
</tr>
<tr>
<td>ARM</td>
<td>291</td>
</tr>
<tr>
<td>[:STARt</td>
<td>:SEQuence[1]]</td>
</tr>
<tr>
<td>[:LAYer[1]]</td>
<td>291</td>
</tr>
<tr>
<td>:COUNT</td>
<td>291</td>
</tr>
<tr>
<td>:LAYer2</td>
<td>292</td>
</tr>
<tr>
<td>:COUNT</td>
<td>292</td>
</tr>
<tr>
<td>[:IMMediate]</td>
<td>293</td>
</tr>
<tr>
<td>:SLOPe</td>
<td>293</td>
</tr>
<tr>
<td>:SOURce</td>
<td>294</td>
</tr>
<tr>
<td>:SWEep</td>
<td>:SEQuence3</td>
</tr>
<tr>
<td>[:IMMediate]</td>
<td>295</td>
</tr>
<tr>
<td>:LINK</td>
<td>296</td>
</tr>
<tr>
<td>:SOURce</td>
<td>297</td>
</tr>
<tr>
<td>CALibration</td>
<td>298</td>
</tr>
<tr>
<td>:COUNT?</td>
<td>298</td>
</tr>
<tr>
<td>:DATA</td>
<td>299</td>
</tr>
<tr>
<td>:AC[1]</td>
<td>299</td>
</tr>
<tr>
<td>:AC2</td>
<td>299</td>
</tr>
<tr>
<td>[:DC]</td>
<td>300</td>
</tr>
<tr>
<td>[:DC]</td>
<td>300</td>
</tr>
<tr>
<td>:BEGIN</td>
<td>300</td>
</tr>
<tr>
<td>:POINT?</td>
<td>301</td>
</tr>
<tr>
<td>:SECure</td>
<td>302</td>
</tr>
<tr>
<td>:CODE</td>
<td>302</td>
</tr>
<tr>
<td>[:STATe]</td>
<td>303</td>
</tr>
<tr>
<td>:STATe</td>
<td>304</td>
</tr>
<tr>
<td>:AC</td>
<td>304</td>
</tr>
<tr>
<td>:DC</td>
<td>305</td>
</tr>
<tr>
<td>INITiate</td>
<td>306</td>
</tr>
<tr>
<td>[:IMMediate]</td>
<td>306</td>
</tr>
<tr>
<td>OUTPut[1]</td>
<td>308</td>
</tr>
<tr>
<td>:FILTER</td>
<td>308</td>
</tr>
<tr>
<td>[:LPASs]</td>
<td>308</td>
</tr>
<tr>
<td>:FREQuency</td>
<td>308</td>
</tr>
<tr>
<td>[:STATe]</td>
<td>309</td>
</tr>
<tr>
<td>:IMPedance</td>
<td>309</td>
</tr>
<tr>
<td>:LOAD</td>
<td>310</td>
</tr>
<tr>
<td>:AUTO</td>
<td>311</td>
</tr>
<tr>
<td>[:STATe]</td>
<td>311</td>
</tr>
<tr>
<td>[SOURce:]</td>
<td>313</td>
</tr>
<tr>
<td>ARBitrary</td>
<td>313</td>
</tr>
<tr>
<td>:DAC</td>
<td>313</td>
</tr>
<tr>
<td>:FORMat</td>
<td>313</td>
</tr>
<tr>
<td>:SOURce</td>
<td>315</td>
</tr>
<tr>
<td>:DOWNload</td>
<td>316</td>
</tr>
<tr>
<td>:COMPLETE</td>
<td>318</td>
</tr>
<tr>
<td>Command</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>:SOURce:</td>
<td>319</td>
</tr>
<tr>
<td>FREQuency[1]</td>
<td>319</td>
</tr>
<tr>
<td>:CENTer</td>
<td>321</td>
</tr>
<tr>
<td>[:CW]:FIXed</td>
<td>322</td>
</tr>
<tr>
<td>:FSKey</td>
<td>323</td>
</tr>
<tr>
<td>:SOURce</td>
<td>324</td>
</tr>
<tr>
<td>:MODE</td>
<td>325</td>
</tr>
<tr>
<td>:RANGe</td>
<td>326</td>
</tr>
<tr>
<td>:SPAN</td>
<td>327</td>
</tr>
<tr>
<td>:STARt</td>
<td>328</td>
</tr>
<tr>
<td>:STOP</td>
<td>329</td>
</tr>
<tr>
<td>:SOURce:</td>
<td>330</td>
</tr>
<tr>
<td>FREQuency2</td>
<td>330</td>
</tr>
<tr>
<td>[:CW]:FIXed</td>
<td>331</td>
</tr>
<tr>
<td>:SOURce:</td>
<td>332</td>
</tr>
<tr>
<td>FUNCtion</td>
<td>332</td>
</tr>
<tr>
<td>[:SHAPe]</td>
<td>332</td>
</tr>
<tr>
<td>:USER</td>
<td>333</td>
</tr>
<tr>
<td>:SOURce:</td>
<td>334</td>
</tr>
<tr>
<td>LIST[1]</td>
<td>334</td>
</tr>
<tr>
<td>:FORMat</td>
<td>335</td>
</tr>
<tr>
<td>[:DATA]</td>
<td>335</td>
</tr>
<tr>
<td>[:SEGMen]t</td>
<td>336</td>
</tr>
<tr>
<td>:ADDress?</td>
<td>336</td>
</tr>
<tr>
<td>:CATalog?</td>
<td>336</td>
</tr>
<tr>
<td>:COMBined</td>
<td>337</td>
</tr>
<tr>
<td>:PIONts?</td>
<td>338</td>
</tr>
<tr>
<td>:DEFine</td>
<td>339</td>
</tr>
<tr>
<td>:DELete</td>
<td>340</td>
</tr>
<tr>
<td>:ALL</td>
<td>340</td>
</tr>
<tr>
<td>[:SELected]</td>
<td>340</td>
</tr>
<tr>
<td>:FREE?</td>
<td>341</td>
</tr>
<tr>
<td>:MARKer</td>
<td>342</td>
</tr>
<tr>
<td>:PIONts?</td>
<td>343</td>
</tr>
<tr>
<td>:POInt</td>
<td>343</td>
</tr>
<tr>
<td>:SELect</td>
<td>344</td>
</tr>
<tr>
<td>:VOLTage</td>
<td>345</td>
</tr>
<tr>
<td>:DAC</td>
<td>346</td>
</tr>
<tr>
<td>:PIONts?</td>
<td>347</td>
</tr>
<tr>
<td>:SSEQuence</td>
<td>347</td>
</tr>
<tr>
<td>:ADDress?</td>
<td>347</td>
</tr>
<tr>
<td>:CATalog?</td>
<td>348</td>
</tr>
<tr>
<td>:COMBined</td>
<td>348</td>
</tr>
<tr>
<td>:PIONts?</td>
<td>349</td>
</tr>
<tr>
<td>:DEFine</td>
<td>350</td>
</tr>
<tr>
<td>:DELete</td>
<td>351</td>
</tr>
<tr>
<td>:ALL</td>
<td>351</td>
</tr>
<tr>
<td>[:SELected]</td>
<td>351</td>
</tr>
<tr>
<td>:DWELl</td>
<td>352</td>
</tr>
<tr>
<td>:COUNt</td>
<td>352</td>
</tr>
<tr>
<td>:PIONts?</td>
<td>353</td>
</tr>
<tr>
<td>:FREE?</td>
<td>353</td>
</tr>
<tr>
<td>:MARKer</td>
<td>354</td>
</tr>
<tr>
<td>:PIONts?</td>
<td>355</td>
</tr>
<tr>
<td>:POInt</td>
<td>355</td>
</tr>
<tr>
<td>:SELect</td>
<td>356</td>
</tr>
<tr>
<td>:SEQuence</td>
<td>357</td>
</tr>
<tr>
<td>:SEGMen</td>
<td>ts?</td>
</tr>
<tr>
<td>:SOURce:</td>
<td>358</td>
</tr>
<tr>
<td>LIST2</td>
<td>358</td>
</tr>
<tr>
<td>:FORMat</td>
<td>358</td>
</tr>
<tr>
<td>[:DATA]</td>
<td>358</td>
</tr>
<tr>
<td>:FREQuency</td>
<td>359</td>
</tr>
<tr>
<td>:PIONts?</td>
<td>360</td>
</tr>
<tr>
<td>:MARKer</td>
<td>361</td>
</tr>
<tr>
<td>:ECLTrg&lt;</td>
<td>&gt;n</td>
</tr>
<tr>
<td>:FEED</td>
<td>361</td>
</tr>
<tr>
<td>[:STATe]</td>
<td>362</td>
</tr>
<tr>
<td>:FEED</td>
<td>363</td>
</tr>
<tr>
<td>:Polar</td>
<td>ity</td>
</tr>
<tr>
<td>[:STATe]</td>
<td>364</td>
</tr>
<tr>
<td>:SOURce:</td>
<td>365</td>
</tr>
<tr>
<td>PM</td>
<td>365</td>
</tr>
<tr>
<td>[:DEViation]</td>
<td>365</td>
</tr>
<tr>
<td>:SOURce</td>
<td>366</td>
</tr>
<tr>
<td>:STATe</td>
<td>367</td>
</tr>
<tr>
<td>:UNIT</td>
<td>367</td>
</tr>
<tr>
<td>[:ANGLe]</td>
<td>367</td>
</tr>
</tbody>
</table>
Command Types

Commands are separated into two types: IEEE 488.2 Common Commands and SCPI Commands.

Common Command Format

The IEEE 488.2 standard defines the Common Commands that perform functions like reset, self-test, status byte query, etc. Common commands are four or five characters in length, always begin with the asterisk character (*), and may include one or more parameters. The command keyword is separated from the first parameter by a space character. Some examples of Common Commands are shown below:

*RST, *CLS, *ESE <unmask>, *OPC?, *STB?

SCPI Command Format

The functions of the AFG are programmed using SCPI commands. SCPI commands are based on a hierarchical structure, also known as a tree system. In this system, associated commands are grouped together under a common node or root, thus, forming subtrees or subsystems. An example is the AFG’s ARM subsystem.

ARM

[:START]:SEQUence[1]
[:LAYER[1]]
  :COUNt <number>
  :LAYER2
  :COUNt <number>
  [:IMMediate] [no query]
  :SLOPe <edge>
  :SOURce <source>

:SWEep]:SEQUence3
  :COUNt <number>
  [:IMMediate] [no query]
  :LINK <link>
  :SOURce <source>

ARM is the root keyword of the command, [:START]:SEQUence1 and :SWEep]:SEQUence3 are second level keywords, :LAYER1 and :LAYER2 are third level keywords, and so on.
Command Separator

A colon (:) always separates one command keyword from a lower level command keyword as shown below:

```
ARM:LAY2:SOUR EXT
```

Abbreviated Commands

The command syntax shows most commands as a mixture of upper and lower case letters. The upper case letters indicate the abbreviated spelling for the command. For shorter program lines, send the abbreviated form. For better program readability, you may send the entire command. The AFG will accept either the abbreviated form or the entire command.

For example, if a command’s syntax contains the keyword COUNT, then COUN and COUNT are acceptable forms. Other forms of COUNT such as COU will generate an error.

You can use upper or lower case letters. Therefore, COUNT, coun, or Coun are all acceptable.

Implied (Optional) Commands

Implied or optional commands are those which appear in square brackets ([ ]) in the command syntax. The brackets are not part of the command, and are not sent to the AFG. Suppose you send the following command:

```
ARM:COUN 100
```

In this case, the AFG responds as if you had executed the command as:

```
ARM:STARt:LAYer1:COUNt 100
```

Variable Command Syntax

Some commands will have what appears to be a variable syntax. For example:

```
[SOURce:]MARKer:ECLTrg<n>[:STATe] <mode>
```

In this command, <n> is replaced by a number. No space is left between the keyword (ECLTrg) and the number because the number is part of the keyword.
SCPI Command Parameters

Parameters are enclosed in greater than/less than symbols (<>) in the command syntax and must always be separated from the keywords by a space. When more than one parameter is allowed, the parameters are separated by a vertical line (|).

The following information contains explanations and examples of the parameter types found in this chapter.

Parameter Types, Explanations, and Examples

- **Numeric**

  Accepts all commonly used decimal representations of numbers including optional signs, decimal points, and scientific notation:

  123, 123E2, -123, -1.23E2, .123, 1.23E-2, 1.23000E-01.

  Special cases include MINimum, MAXimum, and INFinity. The Comments section within the Command Reference will state whether a numeric parameter can also be specified in hex, octal, and/or binary:

  #H7B, #Q173, #B1111011

- **Boolean**

  Represents a single binary condition that is either true or false. Any non-zero value is considered true:

  ON, OFF, 1, 0

- **Discrete**

  Selects from a finite number of values. These parameters use mnemonics to represent each valid setting. An example is the TRIGger[:STARt]:SOURce <source> command where source can be BUS, ECLTrg0, ECLTrg1, EXTernal, HOLD, INTernal[1], INTernal[2], or TTLTrg0 through TTLTrg1.

- **Arbitrary Block Program Data**

  This parameter type is used to transfer a block of data in the form of bytes. The block of data bytes is preceded by a header which indicates either

  1. The number of data bytes which follow (definite length block), or

  2. That the following data block will be terminated upon receipt of a New Line message with the EOI signal true (indefinite length block).
The syntax for data in the blocks is as follows:

**Definite length block:**

#<non-zero digit><digit(s)><data byte(s)>

Where the value of <non-zero digit> equals the number of <digit(s)}. The value of <digit(s)> taken as a decimal integer indicates the number of <data byte(s)> in the block.

**Indefinite length block:**

#0<data byte(s)><NL^END>

Examples of sending 4 data bytes:

#14<byte><byte><byte><byte>
#3004<byte><byte><byte><byte>
#0<byte><byte><byte><byte><NL^END>

**Optional Parameters**

Command parameters shown within square brackets ([ ] ) are optional. The brackets are not part of the parameter, and are not sent to the AFG. If you do not specify a value for an optional parameter, the instrument chooses a default value.

For example, consider the ARM[:START]:LAYer[:1]:COUNT? [<MIN | MAX | INF>] command. If you send the command without specifying a parameter, the present ARM[:START]:LAYer[:1]:COUNT value is returned. If you send the MIN parameter, the command returns the minimum count available. If you send the MAX parameter, the command returns the maximum count available. There must be a space between the command and the parameter.

**Querying Parameter Settings**

Unless otherwise noted in the reference section, parameter settings can be queried by adding a question mark (?) to the command which set the parameter. For example:

SOUR:FREQ1:FIX 20E3

sets the frequency to 20 kHz. The value can be queried by executing:

SOUR:FREQ1:FIX?

The MINimum or MAXimum value of a parameter is determined as follows:

SOUR:FREQ1:FIX? MIN
SOUR:FREQ1:FIX? MAX

The minimum and maximum values returned are based on the settings of other AFG commands at that time.
SCPI Command Execution

The following information should be remembered when executing SCPI commands.

**Command Coupling**

Many of the AFG SCPI commands are value coupled. This means that sending a command can change parameter values set by previous commands. Often, this results in “Settings Conflict” errors when the program executes. To prevent these errors, the AFG commands must be executed in “Coupling Groups”. The coupling groups and associated commands are listed in Table B-2 in Appendix B.

The coupling groups identified in Table B-2 are frequency and voltage. Some commands (like [SOURce:]FUNCTION[:SHAPE]) are associated with both groups. These commands are a bridge linking (coupling) the two groups. Commands not in a coupling group must precede or follow commands in the coupling groups. Executing un-coupled commands in a coupling group breaks the coupling and can cause a “Settings Conflict” error. Command queries (commands with ?) are uncoupled commands and should be executed before or after coupled commands.

See “Executing Coupled Commands” on page 28 for information on executing coupled commands.

**MIN and MAX Parameters in Coupling Groups**

When MINimum or MAXimum is the parameter of a command in a coupling group, that command should be the last command executed in the group. Unlike other parameters that are set when an end-of-line indication is received, MIN and MAX are evaluated by the AFG processor when the command is parsed. Thus, the value of MIN or MAX is based on the values of the other (coupling group) commands at that time. “Settings conflict” errors will occur if the current values are incompatible with an intended MIN or MAX value. As a result, MIN and MAX are not recommended for specifying the value of a parameter.

**Linking Commands**

**Linking IEEE 488.2 Common Commands.**

Use a semicolon between the commands. For example:

```
*RST;*CLS;*OPC?
```

**Linking Multiple SCPI Commands.**

Use both a semicolon and a colon between the commands. For example:

```
SOUR:ROSC:SOUR INT1;:TRIG:STAR:SOUR INT1
```

**Command Choices**

Some commands are listed as two commands separated with a vertical bar (“|”). This means that either command name can be used. For example, use either :CW or :FIXed when :CW|FIXed is shown.
SCPI Command Reference

This section describes the SCPI commands for the HP E1445A Arbitrary Function Generator. Commands are listed alphabetically by subsystem and also within each subsystem. A command guide is printed in the top margin of each page. The guide indicates the subsystem listed on that page.
The `ABORt` command places the TRIGGER subsystem in the idle state, regardless of any other settings. The command halts waveform generation, but keeps the output voltage at the value generated when `ABORt` was executed. Only another `INITiate:IMMediate` command will restart waveform output.

**Subsystem Syntax**

```
ABORt [no query]
```

**Comments**

- `ABORt` does not affect any other settings of the HP E1445A.
- The Pending Operation Flag set true by the `INITiate:IMMediate` command will be set false as a consequence of entering the trigger idle state. Subsequent `*OPC`, `*OPC?`, and `*WAI` commands will therefore complete immediately.
- **Executable when Initiated:** Yes
- **Coupling Group:** None
- **Related Commands:** `*OPC`, `*OPC?`, `*WAI`, `INITiate:IMMediate`
- **RST Condition:** `*RST` places the HP E1445A in the trigger idle state, as if executing an `ABORt` command.

**Example** Aborting a Waveform

```
ABOR
```

*Places HP E1445A in idle state.*
The ARM subsystem operates with the TRIGger subsystem to control the starting of waveform output and frequency sweeps or list generation, as follows:

- The source and slope for arming (starting) waveform generation.
- The number of waveform start arms the HP E1445A will accept before trigger system returns to the idle state.
- The number of repetitions of a waveform that will be output for each start arm accepted.
- The number of sweep arms the HP E1445A will accept before the sweep system returns to the idle state.
- The source and slope for arming (starting) a frequency sweep or list generation.

### Subsystem Syntax

```plaintext
ARM
[:STARt]:SEQuence[1]
[:LAYer[1]]
:COUNt <number>
:LAYer2
:COUNt <number>
[:IMMediate] [no query]
:SLOPe <edge>
:SOURce <source>

:SWEep]:SEQuence3
:COUNt <number>
[:IMMediate] [no query]
:LINK <link>
:SOURce <source>
```

### [:STARt][:LAYer[1]]:COUNt

**ARM[:STARt][:LAYer[1]]:COUNt** selects the number of waveform repetitions to be output for each start arm accepted.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>numeric</td>
<td>1 through 65536</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9.9E+37</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>INFinity</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

MINimum selects 1 repetitions; MAXimum selects 65536 repetitions. 9.9E+37 is equivalent to INFinity.

**Comments**

- Use the ABORt or TRIGger:STOP[:IMMediate] command to terminate the output when ARM[:STARt]:LAYer1:COUNt is set to INFinity or 9.9E+37.

- For standard function sine waves, the actual number of cycles which appear at the output relative to the programmed count is approximate, and is not specified.
\textbf{Example} Setting Waveform Repetitions per Arm

\texttt{ARM:COUN 10} \hspace{1cm} \textit{Sets 10 repetitions/arm.}

\textbf{[:STARt]:LAYer2:COUNt}

\texttt{ARM[:STARt]:LAYer2:COUNt} \texttt{<number>} specifies the number of waveform start arms the HP E1445A will accept after an \texttt{INITiate[:IMMediate]} command before returning the trigger system to the idle state.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
\textbf{Parameter Name} & \textbf{Parameter Type} & \textbf{Range of Values} & \textbf{Default Units} \\
\hline
<number> & numeric & 1 through 65535 | MINimum | MAXimum | 9.9E+37 | INFinity & none \\
\hline
\end{tabular}
\end{table}

\textbf{Comments} • Use the \texttt{ABORt} command to return the trigger system to the idle state when \texttt{ARM[:STARt]:LAYer2:COUNt} set to \texttt{INFinity} or 9.9E+37.

• \textbf{Executable when Initiated:} Query form only

• \textbf{Coupling Group:} None

• \textbf{Related Commands:} ABORt, INITiate[:IMMediate]

• \textbf{*RST Condition:} ARM:STARt:LAYer1:COUNt INFinity

\textbf{Example} Setting the Start Arm Count

\texttt{ARM:LAY2:COUN 10} \hspace{1cm} \textit{Sets 10 start arms per INITiate.}
[:STARt]:LAYer2[:IMMediate]

**ARM[:STARt]:LAYer2[:IMMediate]** immediately arms the waveform regardless of the selected arm source. The trigger system must be initiated and the start trigger sequence must be in the wait-for-arm state. The selected start arm source remains unchanged.

**Comments**

- Executing this command with the start trigger sequence not in the wait-for-arm state generates Error -212,"Arm ignored".
- Executable when Initiated: Yes
- Coupling Group: None
- Related Commands: INITiate[:IMMediate]
- *RST Condition: None

**Example** Starting a Waveform

```
ARM:LAY2:SOUR HOLD
INIT
ARM:LAY2
```

Sets manual arm source.
Initiates trigger system.
Starts waveform.

[:STARt]:LAYer2:SLOPe

**ARM[:STARt]:LAYer2:SLOPe** `<edge>` selects the edge (rising or falling) on the HP E1445A’s front panel “Start Arm In” BNC which starts waveform generation. This edge is significant only with ARM[:STARt]:LAYer2:SOURce set to EXTernal. The programmed value is retained but not used when other sources are selected.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;edge&gt;</code></td>
<td>discrete</td>
<td>NEGative</td>
<td>POSitive</td>
</tr>
</tbody>
</table>

**Comments**

- Executable when Initiated: Query form only
- Coupling Group: None
- Related Commands: ARM[:STARt]:LAYer2:SOURce
- *RST Condition: ARM:STARt:LAYer2:SLOPe POSitive

**Example** Setting the Start Arm Slope

```
ARM:LAY2:SLOP NEG
```

Sets negative start arm slope.
ARM

[:START]:LAYer2:SOURce

**ARM[:START]:LAYer2:SOURce** `<source>` selects the source that will start waveform output. The available sources are:

- **BUS** – The Group Execute Trigger (GET) HP-IB command or the IEEE-488.2 *TRG common command.
- **ECLTrg0** and **ECLTrg1** – The VXIbus ECL trigger lines.
- **EXTernal** – The HP E1445A’s front panel “Start Arm In” BNC connector.
- **HOLD** – Suspend arming. Use the **ARM[:START]:LAYer2[:IMMediate]** command to start the waveform.
- **IMMediate** – Immediate arming. An arm is internally generated two to three reference oscillator cycles after the start trigger sequence enters the wait-for-arm state.
- **TTLTrg0** through **TTLTrg7** – The VXIbus TTL trigger lines.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;source&gt;</code></td>
<td>discrete</td>
<td>BUS</td>
<td>ECLTrg0</td>
</tr>
</tbody>
</table>

**Comments**

- Use the **ARM[:START]:LAYer2:SLOPe** command to select the active edge for the front panel “Start Arm In” BNC when used as the start arm source.
- **Executable when Initiated:** Query form only
- **Coupling Group:** None
- **Related Commands:** ARM[:START]:LAYer2:SLOPe
- **:*RST Condition:** ARM:START:LAYer2:SOURce IMMEDIATE

**Example** Setting the Start Arm Source

**ARM:LAY2:SOUR EXT**  
*Start arm source is front panel’s “Start Arm In” BNC.*
:SWEP:COUNt

**ARM:SWEep:COUNt <number>** specifies the number of sweep arms the HP E1445A will accept after an INITiate:IMMediate command before the sweep trigger sequence returns to the idle state. This command is equivalent to the [SOURce:]SWEP:COUNt command; either command may be used, and executing either one changes the value of the other.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>numeric</td>
<td>1 through 2147483647</td>
<td>INFinity</td>
</tr>
</tbody>
</table>

Minimum selects 1 arm; Maximum selects 2147483647 arms. 9.9E+37 is equivalent to INFinity.

### Comments

- **Executable when Initiated:** Query form only
- **Coupling Group:** Frequency
- **Related Commands:** INITiate[:IMMediate]
- **:*RST Condition:** ARM:SWEep:COUNt 1

### Example

Setting the Sweep Arm Count

```
ARM:SWE:COUN 10
```

Sets 10 sweep arms per INITiate.

:SWEep[:IMMediate]

**ARM:SWEep[:IMMediate]** starts a frequency sweep or list regardless of the selected sweep arm source. The trigger system must be initiated and the sweep trigger sequence must be in the wait-for-arm state. The selected sweep arm source remains unchanged.

### Comments

- Executing this command when frequency sweeps or lists are not enabled, or with the sweep trigger sequence not in the wait-for-arm state generates Error -212,"Arm ignored".
- **Executable when Initiated:** Yes
- **Coupling Group:** None
- **Related Commands:** INITiate[:IMMediate], [SOURce:]SWEP
- **:*RST Condition:** None
Example Starting a Frequency Sweep

SWE:START 1E3;STOP 10E3
Sets sweep frequency limits.
SWE:POIN 10
Sets 1 kHz steps.
ARM:SOUR IMM
Sets output to start immediately.
ARM:SWE:SOUR HOLD
Sets manual sweep arm.
INIT
Initiates trigger system.
ARM:SWE
Starts sweep.

:SWEep:LINK

ARM:SWEep:LINK <link> selects the internal event that starts a frequency sweep or list when ARM:SWEep:SOURce is set to LINK. The only defined internal event to start a sweep or list is “ARM[:STARt | :SEQ uence[1]]:LAYer2”.

There is no need to send this command since there is only one defined internal event. The command is included for SCPI compatibility purposes only.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;link&gt;</td>
<td>string</td>
<td>“ARM[:STARt</td>
<td>:SEQUence1]:LAYer2”</td>
</tr>
</tbody>
</table>

Comments

• Executable when Initiated: Yes
• Coupling Group: None
• Related Commands: ARM:SWEep:SOURce
• *RST Condition: ARM:SWEep:LINK “ARM[:STARt | :SEQ uence[1]]:LAYer2”

Example Linking the Sweep Arm

ARM:SWE:SOUR LINK
Links sweep arm to start arm.
ARM:SWE:LINK “ARM”
ARM:SWEep:SOURce <source> selects the source that starts a frequency sweep or list. The available sources are:

**BUS** – The Group Execute Trigger (GET) HP-IB command or the IEEE-488.2 *TRG* common command.

**HOLD** – Suspend sweep or list arming. Use ARM:SWEep[:IMMediate] to start the frequency sweep or list.

**IMMediate** – Immediate sweep or list arming. If the sweep advance trigger source (TRIGger:SWEep:SOURce command) is set to TIMer, the first frequency sweep or list starts when the first start arm is received. For multiple sweeps or lists, the last frequency point of each sweep or list is output for the same TRIGger:SWEep:TIMer time as between all other points of the sweep or list.

If TRIGger:SWEep:SOURce is set to any other source, the frequency sweep or list starts when the INITiate:IMMediate command is executed. For multiple sweeps or lists, a last frequency is output until the next sweep advance trigger is received.

**LINK** – The next valid start arm starts a sweep or list.

**TTLTrg0** through **TTLTrg7** – The VXIbus TTL trigger lines.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;source&gt;</td>
<td>discrete</td>
<td>BUS</td>
<td>HOLD</td>
</tr>
</tbody>
</table>

### Comments

- If ARM:SWEep:SOURce is set to TTLTrg<n> and you want to set TRIGger:SWEep:SOURce to TTLTrg<n>, both must be set to the same trigger line <n>.

- **Executable when Initiated:** Query form only

- **Coupling Group:** Frequency

- **Related Commands:** ARM[:START]:LAYer2:SOURce

- ***RST Condition:** ARM:SWEep:SOURce IMMEDIATE

### Example

**Setting the Sweep Arm Source**

```
ARM:SWE:SOUR TTLT1
```

Selects VXIbus trigger line TTLTRG1* as sweep arm source.
The CALibration subsystem has commands that calibrate the HP E1445A. The subsystem also includes commands to prevent and detect accidental or unauthorized calibration of the HP E1445A. The calibration procedure using these commands is located in the *HP E1445A Service Manual*.

### Subsystem Syntax

```
CALibration
 :COUNT?        [query only]
 :DATA
   :AC[1] <block>
   :AC2 <block>
   [:DC] <block>
 [:DC]
   :BEGIN          [no query]
   :POINt? <value> [query only]
 :SECure
   :CODE <code>    [no query]
   [:STATe] <mode[<code>]
 :STATe <state>
   :AC <state>
   :DC <state>
```

### :COUNT?

**CALibration:COUNT?** returns a number that shows how often the HP E1445A has been calibrated. Since executing CALibration:DATA:AC1, AC2, and DC commands and the CALibration:POINt? query (upon completion of the calibration procedure) increment the number, the CALibration:COUNT? command may be used to detect any accidental or unauthorized HP E1445A calibration.

**Comments**
- The HP E1445A was calibrated before it left the factory. Before using, read the calibration count to determine its initial value.
- The HP E1445A stores the calibration number in its non-volatile calibration memory which remains intact even with power off.
- The maximum value of the number is 2,147,483,647, after which it wraps around to 0.
- **Executable when Initiated:** Yes
- **Coupling Group:** None
- **Related Commands:** CALibration:SECure[:STATe], CALibration[:DC]:BEGIN
- **RST Condition:** Unaffected
Example  Querying the Calibration Count

CAL:COUN?  Queries calibration count.

:DATA:AC[1]

CALibration:DATA:AC[1] <block> transfers the 250 kHz filter portion of the HP E1445A’s calibration constants in IEEE-488.2 arbitrary block program data format. The query form returns this portion of the calibration constants in IEEE-488.2 definite block data format. Both forms require that calibration security have been previously disabled. See the HP E1445 Service Manual for detailed information on the use of this command.

Comments

• Executing this command with calibration security disabled increments the calibration count (CALibration:COUNt? query).

• Executable when Initiated: Yes

• Coupling Group: None

• Related Commands: CALibration:COUNt?, CALibration:SECure[:STATe]

• *RST Condition: Unaffected

:DATA:AC2

CALibration:DATA:AC2 <block> transfers the 10 MHz filter portion of the HP E1445A’s calibration constants in IEEE-488.2 arbitrary block program data format. The query form returns this portion of the calibration constants in IEEE-488.2 definite block data format. Both forms require that calibration security have been previously disabled.

Comments

• Executing this command with calibration security disabled increments the calibration count (CALibration:COUNt? query).

• Executable when Initiated: Yes

• Coupling Group: None

• Related Commands: CALibration:COUNt?, CALibration:SECure[:STATe]

• *RST Condition: Unaffected
CALibration

[:DATA[:DC]]

**CALibration:DATA[:DC]** `<block>` transfers the DC portion of the HP E1445A’s calibration constants in IEEE-488.2 arbitrary block program data format. The query form returns the current DC portion of the calibration constants in IEEE-488.2 definite block data format. Both forms require that calibration security have been previously disabled. See the *HP E1445A Service Manual* for detailed information on the use of this command.

**Comments**

- Executing this command with calibration security disabled increments the calibration count (**CALibration:COUNt?** query).
- **Executable when Initiated**: Yes
- **Coupling Group**: None
- **Related Commands**: **CALibration:COUNt?**, **CALibration:SECure[:STATe]**
- ***RST Condition**: Unaffected

[:DC]:BEGin

**CALibration[:DC]:BEGin** starts the DC calibration procedure for the HP E1445A. It sets the HP E1445A up for the first of the 44 measurements in the procedure. Calibration security must have been previously disabled. See the *HP E1445A Service Manual* for detailed information on the use of this command.

**Comments**

- Most of the HP E1445A’s commands cannot be executed while calibration is in progress. The *RST command may be used to prematurely terminate the calibration procedure without affecting the stored calibration constants.
- **Executable when Initiated**: No
- **Coupling Group**: None
- **Related Commands**: **CALibration[:DC]:POINt**, **CALibration:SECure[:STATe]**
- ***RST Condition**: None
CALibration[:DC]:POINt?

CALibration[:DC]:POINt? <value> takes the measured value for the current DC calibration point, computes needed calibration constants, and sets up the HP E1445A for the next measurement. When all measurements have been made, the calibration constants are checked for validity. If the validity check passes, the constants are stored in the HP E1445A’s non-volatile calibration memory and the calibration count (CALibration:COUNt? query) is incremented.

The *RST command should be sent after completing the calibration procedure to restore normal operation.

Calibration security must have been previously disabled. See the HP E1445A Service Manual for detailed information on the use of this command.

Comments

• Most of the HP E1445A’s commands cannot be executed while calibration is in progress. The *RST command may be used to prematurely terminate the calibration procedure without affecting the stored calibration constants.

• Executable when Initiated: No

• Coupling Group: None

• Related Commands: CALibration[:DC]:BEGin, CALibration:SECure[:STATe]

• *RST Condition: None
CALibration

:SECure:CODE

CALibration:SECure:CODE <code> sets the code which is required to disable calibration security. Calibration security must have been previously disabled.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;code&gt;</td>
<td>character data</td>
<td>0 through 12 chars</td>
<td>none</td>
</tr>
</tbody>
</table>

The code must start with a letter (“A” through “Z”) and may contain letters, digits, and underscores. Lower case letters are converted to upper case.

Comments

• Executing this command with calibration security enabled (CALibration:SECure:[STATe] ON set), generates the Error +1002,"Calibration security enabled". Disabling calibration security requires knowledge of the previous security code.

• Before shipping, the factory sets the calibration security code to “E1445A”. You should change it before you use your HP E1445A to prevent unauthorized calibration. Record the new security code and store it in a secure place. If you forget the new code, defeating the security involves instrument disassembly. See the HP E1445A Service Manual if this is required.

• The HP E1445A stores the security code in its non-volatile calibration memory which remains intact even with power off.

• Executable when Initiated: Yes

• Coupling Group: None

• Related Commands: CALibration:SECure:[STATe]

• *RST Condition: Unaffected

Example Changing the Factory-shipped Security Password

CAL:SEC:STAT OFF,E1445A  Disables security.
CAL:SEC:CODE NEWCODE   Sets new security code.
CAL:SEC ON              Re-enables security.
CALibration:SECure[:STATe] <mode>[,<code>] enables or disables calibration security. Calibration security must be disabled to calibrate the HP E1445A, read or write calibration data, change the security code, or change the protected user data.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mode&gt;</td>
<td>boolean</td>
<td>OFF</td>
<td>0</td>
</tr>
<tr>
<td>&lt;code&gt;</td>
<td>string</td>
<td>0 through 15 characters</td>
<td>none</td>
</tr>
</tbody>
</table>

Comments

- Attempting to disable calibration security without providing the <code> parameter generates Error -109,"Missing parameter". The value supplied must match the currently programmed security code or Error -224,"Illegal parameter value" will be generated. The HP E1445A will then wait 1 second before executing any subsequent commands.

- To enable security, the <code> parameter is not required, but is checked if it is present.

- Executable when Initiated: Yes

- Coupling Group: None


- *RST Condition: Unaffected

- Power-On Condition: CALibration:SECure[:STATe] ON

Example

Disabling Calibration Security

CAL:SEC:STATe OFF,E1445A

Disables security assuming factory-set security code.
CALibration

:STATe

CALibration:STATE <state> specifies whether corrections using the calibration constants are made or not. If STATE is OFF, then no corrections are made. If STATE is ON, DC and/or AC corrections will be made or not according to the states of the CALibration:STATE:DC and AC commands.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;state&gt;</td>
<td>boolean</td>
<td>OFF</td>
<td>0</td>
</tr>
</tbody>
</table>

Comments

- Executable when Initiated: Yes
- Coupling Group: None
- Related Commands: CALibration:STATE:AC, CALibration:STATE:DC
- *RST Condition: CALibration:STATE ON

Example

Disabling Calibration Corrections

CAL:STAT OFF

Disables corrections.

:STATe:AC

CALibration:STATE:AC <state> specifies whether AC corrections using the calibration constants are made or not. If state is OFF, then no AC corrections are made. If state is ON, AC corrections will be made if CALibration:STATE:AC is also set.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;state&gt;</td>
<td>boolean</td>
<td>OFF</td>
<td>0</td>
</tr>
</tbody>
</table>

Comments

- Executable when Initiated: Yes
- Coupling Group: None
- Related Commands: CALibration:STATE
- *RST Condition: CALibration:STATE:AC ON

Example

Disabling AC Calibration Corrections

CAL:STAT:AC OFF

Disables AC corrections.
CALibration:STATe:DC <state> specifies whether DC corrections using the calibration constants are made or not. If state is OFF, then no DC corrections are made. If state is ON, DC corrections will be made if CALibration:STATe ON is also set.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;state&gt;</td>
<td>boolean</td>
<td>OFF</td>
<td>0</td>
</tr>
</tbody>
</table>

Comments
- Executable when Initiated: Yes
- Coupling Group: None
- Related Commands: CALibration:STATe
- *RST Condition: CALibration:STATe:DC ON

Example
Disabling DC Calibration Corrections

CAL:STAT:DC OFF

Disables DC corrections.
The INITiate subsystem initiates the trigger subsystem and prepares the HP E1445A for waveform generation. Once initiated, a start arm received from the programmed arm source (TRIGger:STARt:SOURce command) starts the waveform output.

For frequency sweeping, the initial sample or waveform frequency is the START frequency when [SOURce:]FREQuency[1]:MODE SWEep is set, or the first frequency in the frequency list when [SOURce:]FREQuency[1]:MODE LIST is set.

**Subsystem Syntax**

INITiate

[:IMMediate] [no query]

**[::IMMediate]**

INITiate[:IMMediate] initiates the trigger system and places all trigger sequences in the wait-for arm or wait-for-trigger state, as appropriate. Waveform generation begins when the next start arm is received. When ARM[:START]:LAYer2:COUNt full arm cycles complete, the trigger system returns to the idle state, and waveform generation halts.

This command is an overlapped command as described by IEEE-488.2, Section 12. The exit from the idle state caused by INITiate::IMMediate shall cause its Pending Operation Flag to be set true. This Pending Operation Flag will be set false when the idle state is re-entered, either when the trigger cycle completes or when an ABORt or *RST command is executed.

The STATus:OPC:INITiate command controls whether *OPC, *OPC? and *WAI will test the Pending Operation Flag and wait until it is false (trigger system in the idle state).
Comments

- Use the ABORt command to prematurely halt the waveform generation and place the trigger system in the idle state.

- Waveform output begins immediately if ARM[:STARt]:LAYer2:SOURce IMMEDIATE is set.

- Executing this command when [SOURce:]FUNCTION[:SHApe] DC is set, when [SOURce:]ARBitrary:DAC:SOURce is not set to INTernal, or the trigger system is not in the idle state, Error -213, "Init ignored" is generated.

Executable when Initiated: No

Coupling Group: None

Related Commands: *OPC, *OPC?, *RST, *WAI, ABORt, ARM subsystem, STATus:OPC:INITiate, TRIGger subsystem

*RST Condition: The trigger system is in the idle state.

Example

Initiating Waveform Generation

INIT

Initiates waveform generation.
The `OUTPut[1]` subsystem controls the characteristics of the output waveform. The subsystem sets the low-pass output filter, sets the output source impedance, and enables or disables the output.

**Subsystem Syntax**

```
OUTPut[1]
  :FILTER
    [:LPASs]  [:FREQuency <frequency>  [:STATe <mode>]  [:IMPedance <impedance>]  [:LOAD <load>]  [:AUTO <mode>]  [:STATe <mode>]
```

**:FILTER[:LPASs]:FREQuency**

`OUTPut[1]:FILTER[:LPASs]:FREQuency <frequency>` sets the output filter’s cutoff frequency to either 250 kHz or 10 MHz.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;frequency&gt;</code></td>
<td>numeric</td>
<td>250 kHz</td>
<td>10 MHz</td>
</tr>
</tbody>
</table>

- MINimum selects the 250 kHz filter; MAXimum selects the 10 MHz filter.

**Comments**

- Selecting the cutoff frequency does not enable the output filter. Use the `OUTPut[1]:FILTER[:LPASs][:STATe]` command to enable or disable the output filter.

- **Executable when Initiated**: Yes

- **Coupling Group**: None

- **Related Commands**: `OUTPut[1]:FILTER[:LPASs][:STATe]`

- **RST Condition**: `OUTPut1:FILTer:LPASs:FREQuency 250 KHZ`

**Example** Setting the Low-pass Filter to 10 MHz

```
OUTP:FILT:FREQ 10 MHZ
OUTP:FILT ON
```

Selects 10 MHz output filter.

Enables output filtering.
OUTP[1]:FILTER[:LPASs][:STATE] <mode>

<mode> enables or disables the output filter.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mode&gt;</td>
<td>boolean</td>
<td>OFF</td>
<td>0</td>
</tr>
</tbody>
</table>

### Comments

- **Executable when Initiated:** Yes
- **Coupling Group:** None
- **Related Commands:** OUTP[1]:FILTER[:LPASs]:FREQuency
- **RST Condition:** OUTP1:FILTER[:LPASs]:STATE OFF

### Example

Enabling the 10 MHz Low-pass Filter

```
OUTP:FILT:FREQ 10 MHZ  
OUTP:FILT ON
```

```
Selects 10 MHz output filter.  
Enables output filtering.
```

OUTP[1]:IMPedance <impedance>

<impedance> sets the HP E1445A’s output impedance to either 50Ω or 75Ω.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;impedance&gt;</td>
<td>numeric</td>
<td>50</td>
<td>75</td>
</tr>
</tbody>
</table>

```
MINimum selects 50Ω output impedance; MAXimum selects 75Ω.
```

### Comments

- **Executable when Initiated:** Yes
- **Coupling Group:** Voltage
- **Related Commands:** OUTP[1]:LOAD, OUTP[1]:LOAD:AUTO
- **RST Condition:** OUTP1:IMPedance 50

### Example

Setting 75 Ω Output Impedance

```
OUTP:IMP 75
```

```
Sets 75 Ω output impedance.
```
OUTPut[1]:LOAD <load> indicates whether the actual load applied to the HP E1445A’s “Output 50/75Ω” is either matched to the output impedance specified by OUTPut[1]:IMPedance or is an open circuit. The output voltage into an open circuit is twice that into a matched load. Setting OUTPut[1]:LOAD INFinity compensates for this effect so that the [SOURce:]LIST[1]:SEGment:VOLTage and [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] and OFFSet commands will output the specified voltages into an open circuit.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;load&gt;</td>
<td>numeric</td>
<td>50</td>
<td>75</td>
</tr>
</tbody>
</table>

MINimum selects 50 Ω load impedance; MAXimum selects 75 Ω. Use 9.9E+37 or INFinity to indicate an open circuit output.

### Comments

- The <load> value specified by this command either must be the same as that specified by OUTPut[1]:IMPedance or must be 9.9E+37 or INFinity, or Error -221,"Settings conflict" will be generated.

- With OUTPut[1]:LOAD:AUTO ON set, the OUTPut[1]:LOAD value is coupled to (tracks) the OUTPut[1]:IMPedance value. Changing the IMPedance changes the LOAD value. Specifying a value for LOAD sets AUTO OFF.

- **Executable when Initiated:** Yes

- **Coupling Group:** Voltage

- **Related Commands:** OUTPut[1]:IMPedance, OUTPut[1]:LOAD:AUTO, [SOURce:]LIST subsystem, [SOURce:]VOLTage subsystem

- **RST Condition:** OUTPut[1]:LOAD:AUTO ON is set, and the OUTPut[1]:LOAD value is coupled to the OUTPut[1]:IMPedance value.

### Example

**Indicating Open Circuit Output Load**

OUTP:LOAD INF

*Indicates open circuit.*
:LOAD:AUTO

OUTPut[1]:LOAD:AUTO <mode> indicates whether the OUTPut[1]:LOAD value should be coupled to (track) the OUTPut[1]:IMPedance value.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mode&gt;</td>
<td>boolean</td>
<td>OFF</td>
<td>0</td>
</tr>
</tbody>
</table>

Comments

- With OUTPut[1]:LOAD:AUTO ON set, the OUTPut[1]:LOAD value is coupled to the OUTPut[1]:IMPedance value. Changing the IMPedance changes the LOAD value. Specifying a value for LOAD sets AUTO OFF. AUTO ONCE sets the LOAD value to the IMPedance value and sets AUTO OFF.
- Executable when Initiated: Yes
- Coupling Group: Voltage
- Related Commands: OUTPut[1]:IMPedance, OUTPut[1]:LOAD
- *RST Condition: OUTPut1:LOAD:AUTO ON

Example Uncoupling OUTPut[1]:IMPedance and OUTPut[1]:LOAD

OUTP:LOAD:AUTO OFF

Uncouples impedance and load.

[:STATe]

OUTPut[1][:STATe] <mode> closes or opens the HP E1445A’s output relay to enable or disable the analog output. Disabling the output does not stop waveform generation; however, the output appears as an open circuit.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mode&gt;</td>
<td>boolean</td>
<td>OFF</td>
<td>0</td>
</tr>
</tbody>
</table>

Comments

- Executable when Initiated: Yes
- Coupling Group: None
- *RST Condition: OUTPut1:STATe ON (not SCPI compliant)

Example Disabling the Output

OUTP OFF

Disables output.
The [SOURce:] subsystem is divided into multiple sections, each of which control a particular aspect of the HP E1445A. Each section of the subsystem is separately documented in the following sections of the command reference.

The [SOURce:] node itself is optional.

**Subsystem Syntax**  The first level [SOURce:] syntax tree is:

```plaintext
[SOURce:]
  ARBitrary . . . . . . . . . . . . Page 313
  FREQuency[1] . . . . . . . . . . Page 319
  FREQuency2 . . . . . . . . . . . . Page 330
  FUNCTION . . . . . . . . . . . . . Page 332
  LIST[1] . . . . . . . . . . . . . . Page 334
  LIST2 . . . . . . . . . . . . . . . Page 358
  MARKer . . . . . . . . . . . . . . Page 361
  PM . . . . . . . . . . . . . . . . . Page 365
  RAMP . . . . . . . . . . . . . . . . Page 368
  ROSeillator . . . . . . . . . . . Page 370
  SWEep . . . . . . . . . . . . . . . Page 372
  VOLTage . . . . . . . . . . . . . . Page 377
```
The [SOURce:]ARBitrary subsystem controls:

- The data format for the digital-to-analog converter (DAC).
- The DAC data source.
- Direct downloading of DAC data to the waveform segment memory.

**Subsystem Syntax**

```
[SOURce:] ARBitrary
    :DAC
    :FORMat <format>
    :SOURce <source>
    :DOWNLOAD <source>,<dest>,<length> [no query]
    :COMPLETE [no query]
```

`:DAC:FORMat` specifies the format for the DAC codes. The format controls how to send and receive DAC codes, and how the HP E1445A stores and interprets the waveform segment memory data.

**Note**

The DAC code format cannot be changed after storing the waveform segment data. Use `[SOURce:]LIST[1];SEGMENT;DELETE:ALL` to delete waveform segment data before changing the DAC code format.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;format&gt;</td>
<td>discrete</td>
<td>SIGNed</td>
<td>UNSigned</td>
</tr>
</tbody>
</table>

**Comments**

- **SIGNed**: Selects the two’s-complement format. The DAC code is a two’s complement number where 0 represents 0 V output, -4096 represents negative full scale output, +4095 represents positive full scale. The positive full scale output value is specified by the `[SOURce:]VOLTage[:LEVEL][:IMMediate][:AMPLitude]` command.

- **UNSsigned**: Selects the unsigned format. The DAC code is an unsigned number where 0 represents negative full scale output and 8191 represents positive full scale.
• There is no need to specify the DAC format with waveforms programmed in volts. The format should be specified if you are:

  a. Programming waveforms in DAC codes
     ([SOURce:]LIST[1]:[SEGMENT]:COMBined or
     [SOURce:]LIST[1]:[SEGMENT]:VOLTage:DAC commands).

  b. Driving the DAC directly ([SOURce:]ARBitrary:DAC:SOURce).

  c. Directly downloading waveform segments
     ([SOURce:]ARBitrary:DOWNload).

• Related Commands: [SOURce:]ARBitrary:DAC:SOURce,
  [SOURce:]ARBitrary:DOWNload, [SOURce:]LIST[1]:[SEGMENT]:COMBined,
  [SOURce:]LIST[1]:[SEGMENT]:VOLTage:DAC

• Executable when Initiated: Query form only

• Coupling Group: None

• *RST Condition: Unaffected

• Power-On Condition: [SOURce:]ARBitrary:DAC:FORMat SIGNed

Example  Setting Unsigned DAC Code Format

ARB:DAC:FORM UNS  

Sets unsigned format.
[SOURce:]ARBitrary

:DAC:SOURce

[SOURce:]ARBitrary:DAC:SOURce  <source> selects the DAC’s data source.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;source&gt;</td>
<td>discrete</td>
<td>DPORt</td>
<td>INTernal</td>
</tr>
</tbody>
</table>

Comments

- The available sources are:
  - **DPORt**: The HP E1445A’s front panel “Digital Port In” connector.
  - **LBUS**: The VXIbus local bus.
  - **INTernal**: The [SOURce:]LIST[1] subsystem or built-in waveforms.
  - **VXI**: The VXIbus data transfer bus.

- When driving the DAC from the VXIbus data transfer bus, the address for writing the data is offset 38 decimal (26 hex) in the HP E1445A’s A24 address space.

- Setting the DAC data source to a setting other than INTernal disables the ARM subsystem, the [INITiate] command, the [SOURce:] subsystem except for the [SOURce:]ARBitrary, [SOURce:]MARKer, and [SOURce:]VOLTage subsystems, and the TRIGger subsystem. The HP E1445A immediately outputs each DAC data point when received. Also, the output amplitude must be specified in terms of volts or volts peak (V or VPK).

- Use the [SOURce:]ARBitrary:DAC:FORMat command to select the format of the data (two’s-complement or unsigned) when directly driving the DAC from the VXIbus local bus, the front panel “Digital Port”, or the VXI backplane, or when programming waveforms using DAC codes via the [SOURce:]ARBitrary:DOWNload, [SOURce:]LIST[1][:SEGMent]:COMBined, or [SOURce:]LIST[1][:SEGMent]:VOLTage:DAC commands.

- **Executable when Initiated**: Query form only

- **Coupling Group**: Frequency and voltage

- **Related Commands**: VINstrument[:CONFigure]:LBUS[:MODE]

- **!*RST Condition**: SOURce:ARBitrary:DAC:SOURce  INternal

Example  Setting the DAC Data Source

ARB:DAC:SOUR DPOR  Selects front panel “Digital Port” connector as source.
:DOWNLOAD

[SOURce:]ARBitrary:DOWNLOAD <source>,<dest>,<length> enables the direct download mode to the waveform segment or segment sequence memory. It selects the download source, waveform segment or segment sequence name, and number of points. The available download sources are:

- **DPORT**: The HP E1445A’s front panel “Digital Port In” connector. Only waveform segment memory may be downloaded via this source.
- **LBUS**: The VXIbus local bus. Only waveform segment memory may be downloaded via this source.
- **VXI**: The VXIbus data transfer bus.

### Waveform Segment Data

The waveform segment data consists of a single 16-bit word for each voltage point. The format for downloaded waveform segment data is:

<table>
<thead>
<tr>
<th>Bits 15–3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC code</td>
<td>unused</td>
<td>marker</td>
<td>last point</td>
</tr>
</tbody>
</table>

The **DAC code** is a 13-bit two’s complement or unsigned number (see the [SOURce:]ARBitrary:DAC:FORMat command on page 313). With [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] 5.11875 V set and a matched output load, the least significant bit (LSB) represents 1.25 mV. If the **marker** bit is 1, a marker pulse will be output with this point if the **marker** bit in the segment sequence memory location generating this segment is also a 1. **Last point** is 1 for the waveform segment’s third-to-last point (actual last point - 3).

When downloading waveform segment data from the VXIbus data transfer bus, the address for writing the data is offset 38 decimal (26 hex) in the HP E1445A’s A24 address space.

### Segment Sequence Data

The segment sequence data consists of a 32-bit wide value for each segment in the sequence. The value should be sent as two 16-bit words with the most significant word sent first. The format for downloaded segment sequence data is:

<table>
<thead>
<tr>
<th>Bits 31–20</th>
<th>Bit 19</th>
<th>Bit 18</th>
<th>Bit 17</th>
<th>Bits 16–0</th>
</tr>
</thead>
<tbody>
<tr>
<td>repetition count</td>
<td>last point</td>
<td>marker enable</td>
<td>unused</td>
<td>segment address</td>
</tr>
</tbody>
</table>

The **repetition count** is 12-bit unsigned value that is (4096 - the desired repetition count): a value of 4095 in these bits indicates 1 repetition; a value of 0 indicates 4096 repetitions. **Last point** is 1 for the segment sequence’s last point. **Marker enable** is 1 to enable marker pulse generation for that waveform segment. **Segment address** is the starting address of the segment divided by 8. Use the [SOURce:]LIST[1][:SEGMen]t[:ADDRes]? query to obtain the address of a waveform segment.
When downloading segment sequence data from the VXIbus data transfer bus, the most significant 16 bits should be written to offset 34 decimal (22 hex) in the HP E1445A’s A24 address space. The least significant 16 bits should be written to offset 36 decimal (24 hex).

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;source&gt;</td>
<td>discrete</td>
<td>DPORt</td>
<td>LBUS</td>
</tr>
<tr>
<td>&lt;dest&gt;</td>
<td>char</td>
<td>defined waveform segment or segment sequence name</td>
<td>none</td>
</tr>
<tr>
<td>&lt;length&gt;</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

For waveform segments, the *length*, in terms of points, must be greater than or equal to 4 and less than or equal to the defined waveform segment length. The current waveform segment length is set to this length.

MINimum selects 4 points; MAXimum selects the defined waveform segment length.

For segment sequences, the *length*, in terms of points, must be greater than or equal to 1 and less than or equal to the defined segment sequence length. The current segment sequence length is set to this length.

MINimum selects 1 point; MAXimum selects the defined waveform segment or segment sequence length.

### Comments

- The waveform segment or segment sequence must have been previously defined (see the [SOURce:]LIST[1]:SEGment:DEFine and [SOURce:]LIST[1]:SSEQUence:DEFine commands).

- When downloading is complete, use the [SOURce:]ARBitrary:DOWNload:COMPLETE command to restore normal operation.

- No error checking is performed on downloaded data. Erratic operation may occur if invalid data (length or format) is downloaded.

- **Executable when Initiated**: No

- **Coupling Group**: None

- **Related Commands**: [SOURce:]LIST[1] commands

- **RST Condition**: Downloading disabled

### Example

Set up to download 512 points from the VXI backplane to waveform segment “ABC”

```
LIST:SEL ABC
LIST:DEF 512
ARB:DOWN VXI,ABC,512
download data
ARB:DOWN:COMP
```

* Creates segment name.
* Reserves 512 points of segment memory.
* Sets up for download.
* Indicates download complete.
:DOWNLOAD:COMPLETE

[SOURce:]ARBitrary:DOWNLOAD:COMPLETE disables direct downloading mode. Send it when downloading is complete.

**Comments**
- Executable when Initiated: No
- Coupling Group: None
- Related Commands: [SOURce:]ARBitrary:DOWNLOAD
- *RST Condition: Downloading disabled

**Example** Set up to download 512 points from the VXIbus to waveform segment “ABC”

```
LIST:SEL ABC
LIST:DEF 512
ARB:DOWN VXI,ABC,512
  download data
ARB:DOWN:COMP
```

  Creates segment name.
  Reserves 512 points of segment memory.
  Sets up for download.
  Indicates download complete.
The [SOURCE:FREQuency1] subsystem controls the first of the HP E1445A’s two frequency generators. ([SOURCE:FREQuency2 controls the second generator.] The first generator uses a direct digital synthesis (NCO) technique to generate the specified frequencies. It has an upper frequency limit of the reference oscillator frequency divided by 4 (the second generator operates up to the reference oscillator frequency). This generator has excellent resolution (.01 Hz with the 42.94 MHz reference oscillator) and allows frequency sweeping. Sine wave output is possible only with this generator. The second generator has better phase noise characteristics and permits higher frequency operation.

The values programmed by this subsystem are only used when TRIGGER[:START]:SOURCE is set to INTernal1.

**Coupling Rules**

The swept commands START, STOP, CENTer, and SPAN are coupled commands. When sending these commands, the following rules apply:

- If either START or STOP is sent singly, the value of the other is preserved, but the CENTer and SPAN values will change according to the following equations:
  \[
  \text{CENTer} = (\text{START} + \text{STOP})/2 \\
  \text{SPAN} = \text{STOP} - \text{START}
  \]

- If either CENTer or SPAN is sent singly, the value of the other is preserved, but the START and STOP values will change according to the following equations:
  \[
  \text{START} = \text{CENTer} - (\text{SPAN}/2) \\
  \text{STOP} = \text{CENTer} + (\text{SPAN}/2)
  \]

- If any two commands are sent as part of a frequency-coupled group within a single program message, then these two will be set as specified, and the other two will change. If more than two are sent in the group, the sweep will be determined by the last two received.

When MINimum and MAXimum are used with these commands, the values that will be set are the minimum and maximum values that will not cause any of the START, STOP, CENTer, and SPAN values to go beyond the minimum and maximum possible frequencies, given the coupling equations above. For example, if SPAN is currently set to 1 MHz, FREQuency1:CENTer MINimum would set 500 kHz.

The minimum possible frequency is 0 Hz, except in the case of logarithmic frequency sweeps. For logarithmic frequency sweeps, the minimum frequency is the maximum possible frequency divided by 1,073,741,824. The maximum possible frequency depends on the frequency of the currently selected reference oscillator source ([SOURCE:ROSCillator:SOURce), the waveform shape
([SOURce:]FUNCTION[:SHAPe]), and whether or not frequency doubling is enabled ([SOURce:]FREQuency[1]:RANGe), according to the following rules:

- **Arbitrary Waveforms and Sine Wave Outputs:** the maximum possible frequency is the current reference oscillator frequency divided by 4.
- **Square Wave Outputs:** the maximum possible frequency is the current reference oscillator frequency divided by 16.
- **Ramps and Triangle Outputs:** the maximum possible frequency is the current reference oscillator frequency divided by 4 further divided by the [SOURce:]RAMP:POINts value.

For non-sine wave outputs, multiply the above values by 2 if frequency doubling is in effect (see the [SOURce:]FREQuency[1]:RANGe command on page 326).

**Subsystem Syntax**

```
[SOURce:]FREQuency[1]
  :CENTer <center_freq>
  [:CW|:FIXed] <frequency>
  :FSKey <frequency1>,<frequency2>
  :SOURce <source>
  :MODE <mode>
  :RANGe <range>
  :SPAN <freq_span>
  :STARt <start_freq>
  :STOP <stop_freq>
```
[SOURce:]FREQuency[1]

:CENTER

[SOURce:]FREQuency[1]:CENTer <center_freq> sets the center sample rate or waveform frequency for a frequency-swept waveform.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;center_freq&gt;</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

The legal range for <center_freq>, as well as the MINimum and MAXimum values, are context-dependent. See “Coupling Rules” on page 319 for a description of the coupling between STARt, STOP, CENTer, and SPAN.

Comments

- **Executable when Initiated:** Query form only
- **Coupling Group:** Frequency
- **Related Commands:** TRIGger[:STARt]:SOURce, [SOURce:]FREQuency[1]:MODE, RANGE, SPAN, STARt, and STOP, [SOURce:]FUNCTION[:SHApe], [SOURce:]ROScillator commands
- **:*RST Condition:** SOURce:FREQuency1:CENTer 5.36870912 MHz

Example Setting the Center Frequency

FREQ:CENT 1E3

*Sets the center frequency to 1000 Hz.*
[:CW::FIXed]

[SOURce::FREQuency[1][::CW::FIXed]  <frequency> selects the non-swept sample rate for arbitrary waveforms or waveform frequency for the built-in waveforms (sine, square, etc.).

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;frequency&gt;</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

MINimum selects 0 Hz.

**Arbitrary Waveforms and Sine Wave Outputs**: MAXimum selects the current reference oscillator frequency divided by 4.

**Square Wave Outputs**: MAXimum selects the current reference oscillator frequency divided by 16.

**Ramps and Triangle Outputs**: MAXimum selects the current reference oscillator frequency divided by 4 further divided by the [SOURce::RAMP::POINts value.

For non-sine wave outputs, multiply the MAXimum value by 2 if frequency doubling is in effect (see the [SOURce::FREQuency[1]:RANGe command).

The above values bound the legal range for <frequency>.

Comments

- **Executable when Initiated**: Yes

- **Coupling Group**: Frequency

- **Related Commands**: TRIGger[:STARt]:SOURce, [SOURce::FREQuency[1]:MODE, [SOURce::FUNCTION[:SHAPe], [SOURce::ROSCillator commands

- ***RST Condition**: SOURce:FREQuency1:FIXed 10 kHz

Example Setting the Sample Rate or Waveform Frequency

FREQ 1E3  

Sets the frequency to 1000 Hz.
[SOURce:]FREQuency[1]

:FSKey

[SOURce:]FREQuency[1]:FSKey <frequency1>,<frequency2> sets the two sample rates or waveform frequencies for frequency-shift keying. [SOURce:]FREQuency[1]:FSKey:SOURce sets the source which selects between the two sample rates or waveform frequencies. A TTL high level on the selected source generates frequency1; a TTL low level generates frequency2.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;frequency1&gt;</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
<tr>
<td>&lt;frequency2&gt;</td>
<td></td>
<td></td>
<td>Hz</td>
</tr>
</tbody>
</table>

MINimum selects 0 Hz.

Arbitrary Waveforms and Sine Wave Outputs: MAXimum selects the current reference oscillator frequency divided by 4.

Square Wave Outputs: MAXimum selects the current reference oscillator frequency divided by 16.

Ramps and Triangle Outputs: MAXimum selects the current reference oscillator frequency divided by 4 further divided by the [SOURce:]RAMP:POINts value.

For non-sine wave outputs, multiply the MAXimum value by 2 if frequency doubling is in effect (see the [SOURce:]FREQuency[1]:RANGe command).

The above values bound the legal range for <frequency1> and <frequency2>.

Comments

- Executable when Initiated: Yes. However, the frequency being generated will not change until the FSK control source changes levels.

- Coupling Group: Frequency

- Related Commands: TRIGger[:STARt]:SOURce, [SOURce:]FREQuency[1]:MODE, [SOURce:]FREQuency[1]:RANGe, [SOURce:]FUNCTION[:SHAPe], [SOURce:]ROSCillator commands

- *RST Condition: SOURce:FREQuency1:FSKey 10 kHz, 10 MHz

Example

Setting the Frequency-Shift Frequencies

FREQ:FSK 1E6,1 KHZ

Sets 1 MHz and 1 kHz frequencies.
[SOURce:]FREQuency[1]:FSKey:SOURce

[SOURce:]FREQuency[1]:FSKey:SOURce <source> sets the source which will control which of the two FSKey sample rates or waveform frequencies is generated when [SOURce:]FREQuency[1]:MODE FSKey is selected. A high level on the source selects [SOURce:]FREQuency[1]:FSKey <frequency1>; a low level selects <frequency2>.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;source&gt;</td>
<td>discrete</td>
<td>EXTernal</td>
<td>TTLTrg0 through TTLTrg7</td>
</tr>
</tbody>
</table>

Comments

- The available sources are:
  - EXTernal: The HP E1445A’s front panel “Stop Trig/FSK/Gate In” BNC connector.
  - TTLTrg0 through TTLTrg7: The VXIbus TTL trigger lines.

- The front panel’s “Stop Trig/FSK/Gate In” BNC is a three-use connector; for FSK control, as a stop trigger source, or as a sample gate source. Only one of these uses may be active at any time.

- If a VXIbus TTLTrg trigger line is used for FSK control, then no TTLTrg trigger lines can be used as a stop trigger source or as a sample gate source.

- Executable when Initiated: Query form only

- Coupling Group: Frequency

- Related Commands: [SOURce:]FREQuency[1]:FSKey, [SOURce:]FREQuency[1]:MODE

- *RST Condition: SOURce:FSKey:SOURce EXTernal

Example

Setting the FSK Control Source

FREQ:FSK:SOUR TTL0

Selects VXIbus trigger line TTLTRG0 as FSK control source.
:[SOURce:]FREQuency[1]

**MODE**

:[SOURce:]FREQuency[1]:MODE  <mode> determines which set of commands control the frequency subsystem.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mode&gt;</td>
<td>discrete</td>
<td>CW</td>
<td>FIXed</td>
</tr>
</tbody>
</table>

### Comments

- The <mode> parameter has the following meanings:
  - **CW** or **FIXed**: Selects single-frequency mode. [SOURce:]FREQuency[1]:CW | :FIXed selects the sample rate or waveform frequency. CW and FIXed are equivalent.
  - **FSKey**: Selects frequency shift keying mode. [SOURce:]FREQuency[1]:FSKey and the front panel’s “Stop Trig/FSK/Gate In” BNC select the two sample rate or waveform frequencies.
  - **LIST**: Selects frequency list mode. [SOURce:]LIST2:FREQuency sets the sample rate or waveform frequencies.
  - **SWEep**: Selects frequency sweep mode. [SOURce:]FREQuency[1]:CENTer, SPAN, START and STOP commands set the sample or waveform frequency range. The [SOURce:]SWEep, ARM:SWEep, and TRIGger:SWEep subsystems control the sweep.

- **Executable when Initiated**: Query form only

- **Coupling Group**: Frequency

- **Related Commands**: TRIGger[:STARt]:SOURce, [SOURce:]FREQuency[1] subsystem, [SOURce:]LIST2 subsystem, [SOURce:]SWEep subsystem

- ***RST Condition**: SOURce:FREQuency1:MODE  FIXed

### Example

**Setting the Frequency Sweep Mode**

FREQ:MODE LIST  

*Sets the frequency sweep mode.*
[:RANGE]

[SOURce:]FREQuency[1]:RANGE <range> enables or disables frequency doubling for non-sine wave outputs. When doubling is enabled, the waveform is advanced on both edges, instead of one edge, of the square wave generated by the direct digital synthesis chip, thus doubling the maximum sample output rate. However, since the square wave symmetry is not perfect, doubling introduces some systematic jitter in the sample rate. Also, in doubled mode, the frequency resolution worsens by a factor of two.

Setting <range> to any value less than or equal to the maximum undoubled frequency, specified below, disables frequency doubling. Values greater than the maximum undoubled frequency enable frequency doubling.

- **Arbitrary Waveforms**: The maximum undoubled frequency is the current reference oscillator frequency divided by 4.
- **Square Wave Outputs**: The maximum undoubled frequency is the current reference oscillator frequency divided by 16.
- **Ramps and Triangle Outputs**: The maximum undoubled frequency is the current reference oscillator frequency divided by 4 further divided by the [SOURce:]RAMP:POINts value.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;range&gt;</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

MINimum selects 0 Hz; MAXimum selects twice the maximum undoubled frequency. The above values bound the legal range for <range>.

### Comments

- Since the maximum undoubled frequency depends on waveform shape and the reference oscillator frequency, frequency doubling may be alternately enabled and disabled as these settings change if <range> is greater than 0. Setting SOURce:FREQuency1:RANGE 0 is a good way to guarantee that frequency doubling is always disabled.
- **Executable when Initiated**: Query form only
- **Coupling Group**: Frequency
- **Related Commands**: TRIGger[:STARt]:SOURce, [SOURce:]FREQuency[1]:CENTer, MODE, STARt, and STOP, [SOURce:]FUNCTION[:SHAPE], [SOURce:]ROSCillator commands
- ***RST Condition**: SOURce:FREQuency1:RANGE 0.0 Hz
Example  Enabling Frequency Doubling

**FUNC:**SHAP  SQU  
*Selects square wave output.*

**ROSC:**SOUR  INT1  
*Selects 42.94 MHz oscillator.*

**FREQ:**RANG  5MHZ  
*Sets frequency range to 5 MHz.*

:SPAN

**[SOURce:]FREQuency[1]:SPAN**  
*<freq_span>*  
Sets the sample rate or waveform frequency span for a frequency-swept waveform.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;freq_span&gt;</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

The legal range for <freq_span>, as well as the MINimum and MAXimum values, are context-dependent. See “Coupling Rules” on page 319 for a description of the coupling between STARt, STOP, CENTER, and SPAN.

### Comments

- **Executable when Initiated:** Query form only
  - **Coupling Group:** Frequency
  - **Related Commands:** TRIGger[:STARt]:SOURce,  
    [SOURce:]FREQuency[1]:CENTer, MODE, RANGE, START, and STOP,  
    [SOURce:]FUNCTION[:SHApe], [SOURce:]ROSCillator commands
  - **RST Condition:** SOURce:FREQuency1:SPAN  10.73741824 MHz

Example  Setting the Frequency Span

**FREQ:**SPAN  1E3  
*Sets the frequency span to 1000 Hz.*
\texttt{[SOURce:]FREQuency[1]}:STARt

\texttt{[SOURce:]FREQuency[1]}:STARt \texttt{<start\_freq>} sets the starting sample rate or waveform frequency for a frequency-swept waveform.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{&lt;start_freq&gt;}</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

The legal range for \texttt{<start\_freq>}, as well as the MINimum and MAXimum values, are context-dependent. See "Coupling Rules" on page 319 for a description of the coupling between STARt, STOP, CENTER, and SPAN.

**Comments**

- **Executable when Initiated**: Query form only
- **Coupling Group**: Frequency
- **Related Commands**: TRIGger[:STARt]:SOURce, [SOURce:]FREQuency[1]:CENTer, MODE, RANGE, SPAN, and STOP, [SOURce:]FUNCTION[:SHAPE], [SOURce:]ROSCillator commands
- **RST Condition**: SOURce:FREQuency1:STARt 0.0 Hz

**Example**

**Setting the Starting Frequency**

\texttt{FREQ:STAR 1 KHZ} \hspace{1cm} \textit{Sets the starting frequency to 1000 Hz.}
[SOURce:]FREQuency[1]

:STOP

[SOURce:]FREQuency[1]:STOP \(<stop\_freq>\) sets the stopping sample rate or waveform frequency for a frequency-swept waveform.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;stop_freq&gt;)</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

The legal range for \(<stop\_freq>\), as well as the MINimum and MAXimum values, are context-dependent. See "Coupling Rules" on page 319 for a description of the coupling between STARt, STOP, CENTer, and SPAN.

Comments

- **Executable when Initiated:** Query form only

- **Coupling Group:** Frequency

- **Related Commands:** TRIGger[:STARt]:SOURce,
  [SOURce:]FREQuency[1]:CENTer, MODE, RANGE, SPAN, and STARt,
  [SOURce:]FUNCTION[:SHApe], [SOURce:]ROSCillator commands

- **\(^*\)RST Condition:** SOURce:FREQuency[1]:STOP 10.73741824 MHz

Example

**Setting the Stopping Frequency**

FREQ:STOP 1E3

*Sets the stopping frequency to 1000 Hz.*
The [SOURce:]FREQuency2 subsystem controls the second of the HP E1445A’s two frequency generators. ([SOURce:]FREQuency[1] controls the first generator.)

This second generator consists of a simple divide-by-\( n \) of the currently selected reference oscillator source, where \( n \) may be 1, 2, 3, or any even value between 4 and 131,072. This generator has better phase noise characteristics and permits higher frequency operation than the direct digital synthesis (NCO) technique used by the first generator. The first generator has finer resolution and frequency sweeping capability. Also, sine wave output is possible only with the first generator. Either generator may be used for square, ramp, triangle and arbitrary waveform output.

The values programmed by this subsystem are only used when TRIGger:STARt:SOURce is set to INternal2.

**Subsystem Syntax**

```
[SOURce:]
FREQuency2
  [:CW|:FIXed] <frequency>
```
[:CW | :FIXed]

[SOURce:]FREQuency2[:CW]:FIXed  \textless \textit{frequency} \textgreater  selects the sample rate for arbitrary waveforms or the frequency for the standard waveforms (square, ramp, triangle).

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textless \textit{frequency} \textgreater</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

**Arbitrary Waveforms:** MINimum selects the current reference oscillator frequency divided by 131,072; MAXimum selects the current reference oscillator frequency.

**Square Wave Outputs:** MINimum selects the current reference oscillator frequency divided by 524,288; MAXimum selects the current reference oscillator frequency divided by 4.

**Ramps and Triangles Outputs:** MINimum selects the current reference oscillator frequency divided by 131,072 further divided by the SOURce:RAMP:POINts value; MAXimum selects the current reference oscillator frequency divided by the SOURce:RAMP:POINts value.

The above values bound the valid range for \textless \textit{frequency} \textgreater. The \textless \textit{frequency} \textgreater value is rounded to the nearest frequency that can be produced using the divide-by-n technique of this generator.

**Comments**

- If the actual frequency generated differs from the specified frequency by greater than 1\%, the Frequency bit of the Questionable Signal Status Register will be set. See the STATus subsystem for more information.

- **Executable when Initiated:** Yes

- **Coupling Group:** Frequency

- **Related Commands:** TRIGger[:STARt]:SOURce, [SOURce:]FUNCTION[:SHAPe], [SOURce:]ROSCillator commands, STATus subsystem

- **\text{*RST Condition:** SOURce:FREQuency2:FIXed 10E3

**Example**  Setting the Sample Rate or Waveform Frequency

FREQ2 1E3  \textit{Sets frequency to 1000 Hz.}
The [SOURce:]FUNCTION subsystem controls what waveform shape (arbitrary, sinusoid, etc.) the HP E1445A generates. For arbitrary waveforms generation, the subsystem controls which of the 128 possible segment sequences are selected.

### Subsystem Syntax

```
[SOURce]
:FUNCTION
 [:SHAPE] <shape>
 :USER <name>
```

### [:SHAPE]

[SOURce:]FUNCTION[:SHAPE] <shape> selects what waveform shape the HP E1445A generates.

#### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;shape&gt;</td>
<td>discrete</td>
<td>DC</td>
<td>RAMP</td>
</tr>
</tbody>
</table>

#### Comments

- The <shape> parameter values are shown as follows:
  - **DC**: Generates a DC output voltage.
  - **RAMP**: Generates a stepped ramp. The [SOURce:]RAMP subsystem controls the polarity and number of points.
  - **SINusoid**: Generates a sinusoidal voltage. SINusoid requires that TRIGger[:STARt]:SOURce INTERNAL1 be selected.
  - **SQUare**: Generates a square wave. The [SOURce:]RAMP:POLarity command controls the polarity.
  - **TRIangle**: Generates a stepped triangle wave. The [SOURce:]RAMP subsystem controls the polarity and number of points.
  - **USER**: Generates an arbitrary waveform. The [SOURce:]FUNCTION:USER command selects the segment sequence to be generated.

- **For the DC function**: The voltage level is specified by [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude].

- **For the RAMP, SINusoid, SQUare, TRIangle, and USER functions**:
  - Use [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] to set output amplitude. For arbitrary (USER) waveforms, this command specifies the full-scale output voltage.
  - [SOURce:]VOLTage[:LEVel][:IMMediate]:OFFSet specifies the offset voltage.
- **TRIGger[:START]:SOURce** selects the sample source. SINusoid requires that TRIGger[:START]:SOURce INTernal1 be selected.

- The [SOURce]:FREQuency[1] or [SOURce]:FREQuency2 subsystems specify the signal frequency for RAMP, SINusoid, SQUare, and TRIangle waveforms. They specify the sample rate for arbitrary (USER) waveforms.

- When [SOURce]:FUNCTION[:SHAPE] RAMP or TRIangle is selected, the greater of the SOURce:RAMP:POINts value and 8 points of contiguous waveform segment memory must be available. When [SOURce]:FUNCTION[:SHAPE] SQUare is selected, 8 points of contiguous waveform segment memory must be available. Attempting to select one of these functions with less contiguous waveform segment memory available, or to set [SOURce]:RAMP:POINts to a value larger than the largest contiguous amount of available waveform segment memory when ramp or triangle wave output is selected, will generate Error +1000,"Out of memory".

- **Executable when Initiated:** Query form only

- **Coupling Group:** Frequency and voltage

- **^RST Condition:** SOURce:FUNCTION:SHAPE SINusoid

**Example** Selecting Square Wave Generation Mode

```
FUNC SQU
```

Selects square wave mode.

**:USER**

[SOURce:]FUNCTION:USER <name> selects which one of the 128 possible stored segment sequences the HP E1445A generates when arbitrary waveform generation is selected by [SOURce:]FUNCTION[:SHAPE] USER.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;name&gt;</td>
<td>character data</td>
<td>defined waveform sequence name</td>
<td>none</td>
</tr>
</tbody>
</table>

**Comments**

- **Executable when Initiated:** Query form only

- **Coupling Group:** None

- **Related Commands:** [SOURce:]FUNCTION[:SHAPE]

- **^RST Condition:** SOURce:FUNCTION:USER NONE

**Example** Selecting an Arbitrary Waveform

```
FUNC USER
FUNC:USER ABC
```

Selects arbitrary waveform mode.

Selects segment sequence.
The [SOURce:]LIST[1] subsystem defines the waveform segments and segment sequence for arbitrary waveform generation. The HP E1445A can simultaneously store up to 256 waveform segments and up to 128 segment sequences.

**Subsystem syntax**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FORMat</strong></td>
<td>List waveform format and length.</td>
</tr>
<tr>
<td><strong>:DATA</strong> &lt;format&gt;[,&lt;length&gt;]</td>
<td></td>
</tr>
<tr>
<td><strong>:SEGMent</strong></td>
<td>List segment address and length.</td>
</tr>
<tr>
<td><strong>:ADDRes?</strong></td>
<td>Query only.</td>
</tr>
<tr>
<td><strong>:CATalog?</strong></td>
<td>Query only.</td>
</tr>
<tr>
<td><strong>:COMBined</strong> &lt;combined_list&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>:POINts?</strong></td>
<td>Query only.</td>
</tr>
<tr>
<td><strong>:DEFine</strong> &lt;length&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>:DELeTe</strong></td>
<td>Delete segment.</td>
</tr>
<tr>
<td><strong>:ALL</strong></td>
<td>No query.</td>
</tr>
<tr>
<td><strong>[:SELeCted]</strong></td>
<td>No query.</td>
</tr>
<tr>
<td><strong>:FREE?</strong></td>
<td>Query only.</td>
</tr>
<tr>
<td><strong>:MARKer</strong> &lt;marker_list&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>:POINts?</strong></td>
<td>Query only.</td>
</tr>
<tr>
<td><strong>:SPOint</strong> &lt;point&gt;</td>
<td>No query.</td>
</tr>
<tr>
<td><strong>:SELeCt</strong> &lt;name&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>:VOLTage</strong> &lt;voltage_list&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>:DAC</strong> &lt;voltage_list&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>:POINts?</strong></td>
<td>Query only.</td>
</tr>
<tr>
<td><strong>:SSEQuence</strong></td>
<td>List sequence address and length.</td>
</tr>
<tr>
<td><strong>:ADDRes?</strong></td>
<td>Query only.</td>
</tr>
<tr>
<td><strong>:CATalog?</strong></td>
<td>Query only.</td>
</tr>
<tr>
<td><strong>:COMBined</strong> &lt;combined_list&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>:POINts?</strong></td>
<td>Query only.</td>
</tr>
<tr>
<td><strong>:DEFine</strong> &lt;length&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>:DELeTe</strong></td>
<td>Delete sequence.</td>
</tr>
<tr>
<td><strong>:ALL</strong></td>
<td>No query.</td>
</tr>
<tr>
<td><strong>[:SELeCted]</strong></td>
<td>No query.</td>
</tr>
<tr>
<td><strong>:DWELe</strong></td>
<td>Set dwell.</td>
</tr>
<tr>
<td><strong>:COUNT</strong> &lt;repetition_list&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>:POINts?</strong></td>
<td>Query only.</td>
</tr>
<tr>
<td><strong>:FREE?</strong></td>
<td>Query only.</td>
</tr>
<tr>
<td><strong>:MARKer</strong> &lt;marker_list&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>:POINts?</strong></td>
<td>Query only.</td>
</tr>
<tr>
<td><strong>:SPOint</strong> &lt;point&gt;</td>
<td>No query.</td>
</tr>
<tr>
<td><strong>:SELeCt</strong> &lt;name&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>:SEQUence</strong> &lt;segment_list&gt;</td>
<td></td>
</tr>
<tr>
<td><strong>:SEGMents?</strong></td>
<td>Query only.</td>
</tr>
</tbody>
</table>
[SOURce:]LIST[1]

:FORMat[:DATA]

[SOURce:]LIST[1]:FORMat[:DATA]  <format>[,<length>]  specifies the format of numeric waveform segment and segment sequence list return data in the [SOURce:]LIST[1] subsystem. The available numeric list return data formats are:

  ASCII: Returns numeric data as an NR1 or NR3 number as defined in IEEE-488.2.

  PACKed: Returns data in IEEE-488.2 definite block format. Internal to the block, the format depends on the query being executed, as list below. The most significant byte of each value is always sent first.

  – [SOURce:]LIST[1]:SEGment[:COMBined]?: The data is returned in the format described under the [SOURce:]LIST[1]:SEGment[:COMBined] command.

  – [SOURce:]LIST[1]:SEGment[:MARKer]?: The data is returned in 16-bit integer format.

  – [SOURce:]LIST[1]:SEGment[:VOLTage]?: The data is returned in IEEE-754 64-bit floating point format.

  – [SOURce:]LIST[1]:SEGment[:VOLTage:DAC]?: The data is returned as 16-bit signed or unsigned DAC codes as specified by the [SOURce:]ARBitrary:DAC:FORMat command.

  – [SOURce:]LIST[1]:SSEQUence:DWELl:COUNt?: The data is returned in 16-bit integer format.

  – [SOURce:]LIST[1]:SSEQUence:COMBined?: The data is returned in the format described under the [SOURce:]LIST[1]:SSEQUence:COMBined command.

  – [SOURce:]LIST[1]:SSEQUence:MARKer?: The data is returned in 16-bit integer format.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;format&gt;</td>
<td>discrete</td>
<td>ASCII</td>
<td>PACKed</td>
</tr>
<tr>
<td>&lt;length&gt;</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

If ASCII format is specified, <length> must either be omitted or must be 9 (or MINimum or MAXimum). Packed format ignores the <length> parameter.

Comments

  • Executable when Initiated: Query form only

  • Coupling Group: None

  • Related Commands: [SOURce:]LIST[1]:SEGment] commands, [SOURce:]LIST[1]:SSEQUence commands

  • *RST Condition: SOURce:LIST1:FORMat:DATA  ASCii

Example  Setting PACKed Return Data Format

LIST:FORM  PACK  

Sets packed format.
[SOURce:]LIST[1][:SEGMent]:ADDRess?

returns the address in the waveform segment memory at which the currently selected waveform segment is located.

Comments
- Executable when Initiated: Yes
- Coupling Group: None
- *RST Condition: No waveform segment is selected
- Power-On Condition: No waveform segments are defined

Example
Query Waveform Segment Memory Address
LIST:SEGM:ADDR? Queries segment address.

[:SEGMent]:CATalog?

returns a comma-separated list of quoted strings, each containing the name of a defined waveform segment. If no waveform segment names are defined, a single null string (""") is returned.

Comments
- Executable when Initiated: Yes
- Coupling Group: None
- *RST Condition: None
- Power-On Condition: No waveform segment names are defined

Example
Cataloging Waveform Segment Names
LIST:CAT? Catalog waveform segments.
[:SEGMen]T[:COMBi]ned

[:SEGMen]T[:COMBi]ned \(<combined\_list>\) defines in one step both the output voltage and marker pulse lists that constitute a waveform segment.

**Parameters**

The \(<combined\_list>\) may be either a comma-separated list of values or an IEEE-488.2 definite or indefinite length block containing the values in 16-bit integer format. Each value has the following format:

<table>
<thead>
<tr>
<th>Bits 15–3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC code</td>
<td>unused</td>
<td>marker</td>
<td>reserved</td>
</tr>
</tbody>
</table>

The DAC code is a 13-bit two’s complement or unsigned number (see the [SOURce::ARBitrary:DAc:FORMat] command). With [SOURce::VOLTage::LEVel::IMMediate::AMPLitude] 5.11875 V set and a matched output load, the least significant bit (LSB) represents 1.25 mV. If the marker bit is 1, a marker pulse will be output with this point if the marker bit in the segment sequence memory location generating this segment is also a 1.

**Comments**

- If the comma-separated list of values format is used, the values must be in two’s complement format, i.e., values should range from -32768 to +32767. If block format is used, the most significant byte of each value must be sent first.

- The combined list must be at least four points long but no longer than the reserved length specified by [SOURce::LIST[:SEGMen]:DEFine]. If the combined list length is less than the reserved length, only the number of points specified by the combined list is generated when outputting the waveform segment.

- Executing the query form of this command with voltage point and marker pulse lists defined with different lengths generates Error -221, "Settings conflict" unless the marker pulse list has a length of 1.

- Using combined lists is faster than separately defining the voltage point and marker pulse lists.

- **Executable when Initiated**: No

- **Coupling Group**: None

- **Related Commands**: [SOURce::LIST[1]:SEGMen]:MARKer, [SOURce::LIST[1]:SEGMen]:VOLTage, [SOURce::LIST[1]:SEGMen]:VOLTage:DAC

- ***RST Condition**: Unaffected

- **Power-On Condition**: No waveform segments are defined
Example  Defining a Waveform Segment Combined List

LIST:SEL ABC  
Selects waveform segment ABC.

LIST:DEF 8  
ABC is 8 points long.

LIST:COMB 16000,32000,16000,0,-16000,-32000,-16000,0  
Defines waveform segment.

[SEGMENT]:COMBined:POINts?

[SOURce:]LIST[1]:SEGMENT[:COMBined]:POINts? returns a number indicating the length of the currently selected waveform segment’s combined voltage point and marker pulse list.

Comments

- Executing this command with voltage point and marker pulse lists defined with different lengths generates Error -221,"Settings conflict" unless the marker pulse list has a length of 1. In this case, the length of the voltage point list is returned.

- Executable when Initiated: Yes

- Coupling Group: None

- *RST Condition: None

- Power-On Condition: No waveform segments are defined

Example  Query Combined Point List Length

LIST:SEL ABC  
Selects waveform segment ABC.

LIST:COMB:POIN?  
Queries combined point list length.
[SOURce:]LIST[1]

[:SEGMen]t[:DEFine]

[SOURce:]LIST[1][:SEGMen]t[:DEFine] <length> reserves enough waveform segment memory for a waveform segment of length points for the segment currently selected by [SOURce:]LIST[1][:SEGMen]t[:SELect].

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;length&gt;</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

The length will be rounded up, if needed, to a multiple of 8 points. All defined waveform segments share the waveform segment memory. Any one segment may use any part of or all of this memory.

MINimum reserves 8 points; MAXimum reserves the largest available contiguous piece of waveform segment memory (up to 262,144 points if no waveforms other than standard function sine waves exist).

**Comments**

- Once a waveform segment has been DEFINed, it must be deleted ([SOURce:]LIST[1][:SEGMen]t[:DELeTe[:SELe]cted] command) before its reserved length may be redefined. The voltage point and marker pulse list values and length may be changed repeatedly without re-executing the DEFINe command.

- [SOURce:]LIST[1][:SEGMen]t[:DEFine] initializes the waveform segment voltage point list to zero length and the marker pulse list to a length of 1 with a value of 0 (no markers will be generated).

- While the reserved length must be a multiple of 8, rounded up if necessary, the only restriction on the current waveform segment length (number of voltage points stored) is that it be at least four points long.

- **Executable when Initiated:** Yes

- **Coupling Group:** None

- **Related Commands:** [SOURce:]LIST[1][:SEGMen]t[:SELect]

- ***RST Condition:** Unaffected

- **Power-On Condition:** No waveform segments are defined

**Example**

Reserving Memory for a Waveform Segment

```
LIST:SEL ABC
LIST:DEF 1024
```

Selects waveform segment ABC.

Reserves 1024 points for ABC.
[:SEGMENT]:DELETE:ALL

[:SOURCE:]LIST[:1][:SEGMENT]:DELETE:ALL deletes all defined waveform segment definitions from memory and makes all of the waveform memory available for new waveform segment definitions.

Comments
- If any waveform segment is used in any segment sequence, executing this command generates Error +1102,"Segment in use". No waveform segments will be deleted.
- Use [:SOURCE:]LIST[:1][:SEGMENT]:DELETE[:SELECTED] to delete only the currently selected waveform segment definition.
- Executable when Initiated: No
- Coupling Group: None
- Related Commands: [:SOURCE:]LIST[:1][:SEGMENT]:DELETE[:SELECTED]
- *RST Condition: None
- Power-On Condition: No waveform segments are defined

Example Deleting All Waveform Segments
LIST:DEL:ALL Deletes all segments.

[:SEGMENT]:DELETE[:SELECTED]

[:SOURCE:]LIST[:1][:SEGMENT]:DELETE[:SELECTED] deletes the currently selected waveform segment definition and makes its memory available for new waveform segment definitions.

Comments
- If the waveform segment is used in any segment sequence, executing this command generates Error +1102,"Segment in use". The waveform segment will not be deleted.
- After deleting the currently selected waveform segment, no waveform segment is SELECTed.
- Use [:SOURCE:]LIST[:1][:SEGMENT]:DELETE:ALL to delete all waveform segment definitions with one command.
- Executable when Initiated: No
- Coupling Group: None
- Related Commands: [:SOURCE:]LIST[:1][:SEGMENT]:DELETE:ALL, [:SOURCE:]LIST[:1][:SEGMENT]:SELECT
- *RST Condition: None
• **Power-On Condition**: No waveform segments are defined

**Example**  Deleting a Waveform

- `LIST:SEL ABC` *Selects waveform segment ABC.*
- `LIST:DEL` *Deletes segment.*

[[:SEGMen]:FREE?]

[[:SOURce:]LIST[1]][:SEGMen]:FREE? returns information on waveform segment memory availability and usage. The return data format is:

```
<num_value>,<num_value>
```

The first numeric value shows the amount of waveform segment memory available in points; the second, the amount of waveform segment memory used in points.

**Comments**

- **Executable when Initiated**: Yes
- **Coupling Group**: None
- ***RST Condition**: None
- **Power-On Condition**: All of the waveform segment memory is available

**Example**  Querying Waveform Segment Memory Usage

- `LIST:FREE?` *Queries segment memory usage.*
[:SEGMen]::MARKer

[SOURce:]LIST[1][:SEGMen][:MARKer] <marker_list> defines, for each voltage point of a waveform segment, whether the HP E1445A may output a marker pulse. To actually output a marker pulse, the marker enable list value for the segment sequence entry for the segment must also be set to 1. [SOURce:]MARKer:FEED must be set to "SOURce:LIST” to output the marker pulse on the “Marker Out” BNC; [SOURce:]MARKer:ECLTrg<rn>:FEED must be set to “SOURce:LIST” to output the marker pulse on the corresponding VXIbus ECLTRG* line.

Parameters

The <marker_list> may be either a comma-separated list of values or an IEEE-488.2 definite or indefinite length block containing the values in 16-bit integer format. A value of 0 generates no marker pulse; any non-zero value enables marker pulse generation.

MINimum and MAXimum cannot be used with this command.

Comments

• If block format is used, the most significant byte of each value must be sent first.

• Marker pulses are one sample period wide (nominally 25 nS at 40 MHz clock rate). To widen the pulses, enable marker pulse generation on consecutive points.

• Usually, marker pulse generation is enabled on no more than one point of a waveform segment. The [SOURce:]LIST[1][:SEGMen][:MARKer]:SPOint command is the most efficient way to enable marker pulse generation on a single point.

• The waveform segment’s marker pulse list length must be the same length as its voltage point list or must have a length of 1. If not, executing the INITiate:IMMediate command generates Error +1104,"Segment lists of different lengths".

• A marker pulse list of length 1 is treated as though it were the same length as the voltage point list, with all marker pulse values the same as the specified value.

• The marker pulse list length must be no longer than the reserved length specified by [SOURce:]LIST[1][:SEGMen]:DEFine. If the marker pulse list length is less than the reserved length, only the number of points specified by the most recent marker pulse and voltage point lists is generated when the waveform segment is output.

• Changing marker pulse values preserves the waveform segment’s voltage point list, and vice versa.

• Executable when Initiated: No

• Coupling Group: None

• Related Commands: [SOURce:]LIST[1][:SEGMen]:COMBined

• *RST Condition: Unaffected

• Power-On Condition: No waveform segments are defined
Example  Defining a Waveform Segment Marker Pulse List

- LIST:SEL ABC  
  Selects waveform segment ABC.

- LIST:DEF 8  
  ABC is 8 points long.

- LIST:VOLT -1,.5,.5,.5,5,.5,0,-.5,-1  
  Defines waveform voltages.

- LIST:MARK 1,0,0,0,1,0,0,0  
  Outputs a marker pulse on first and fifth voltage points.

[:SEGMent]:MARKer:POINts?

[:SOURce:]LIST[1]:[SEGMent]:MARKer:POINts? returns a number indicating the length of the currently selected waveform segment’s marker pulse list.

Comments  
- Executable when Initiated: Yes
- Coupling Group: None
- *RST Condition: None
- Power-On Condition: No waveform segments are defined

Example  Query Marker Pulse List Length

- LIST:SEL ABC  
  Selects waveform segment ABC.

- LIST:MARK:POIN?  
  Queries the marker pulse list length.

[:SEGMent]:MARKer:SPOInt

[:SOURce:]LIST[1]:[SEGMent]:MARKer:SPOInt <point> is a short-cut method for defining a marker list with marker pulse generation enabled on a single point. It creates a marker list whose length is the same as the current voltage point list, and which enables marker generation only on the point specified. The voltage point list must have been previously defined.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;point&gt;</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

The valid range for <point> is 1 through the length of the current voltage point list. MINimum selects the first point of the current voltage point list; MAXimum selects the last point.

Comments  
- Executable when Initiated: No
- Coupling Group: None
- Related Commands: [:SOURce:]LIST[1]:[SEGMent]:MARKer
Example Creating a Single Point Marker List

LIST:SEL  ABC  \(\text{Selects waveform segment ABC.}\)
LIST:DEF 8  \(\text{ABC is 8 points long.}\)
LIST:VOLT -1,.5,.5,.5,5,.5,0,-.5,-1  \(\text{Defines waveform voltages.}\)
LIST:MARK:SPO 5  \(\text{Outputs a marker pulse on the fifth voltage point.}\)

[:SEGMent]:SELect

[SOURce:]LIST[1][:SEGMent]:SELect <name> selects a waveform segment for subsequent [SOURce:]LIST[1][:SEGMent] subsystem commands. This command will define the waveform segment name if it is undefined, but does not reserve any waveform segment memory.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;name&gt;</td>
<td>character data</td>
<td>1 through 12 characters</td>
<td>NONE</td>
</tr>
</tbody>
</table>

NONE selects no waveform segment

Comments

- Legal names must start with an alphabetic character and contain only alphabetic, numeric, and underscore (“_”) characters. Alphabetic character case (upper versus lower) is ignored. No waveform segment may have the same name as any segment sequence.

- A maximum of 256 waveform segment names may be defined at any time. Use the [SOURce:]LIST[1][:SEGMent]:DELe:ALL or SELected commands to delete names that are no longer needed.

- Executable when Initiated: Yes

- Coupling Group: None

- *RST Condition: Unaffected

Example Selecting a Waveform Segment

LIST:SEL  ABC  \(\text{Selects waveform segment ABC.}\)
[:SEGMENT]:VOLTage

[SOURce:]LIST[1]:[SEGMENT]:VOLTage <voltage_list> defines the series of output voltage points that constitute a waveform segment. The points are specified in terms of volts.

**Parameters**
The <voltage_list> may be either a comma-separated list of voltage values or an IEEE-488.2 definite or indefinite length block containing the values in IEEE-754 64-bit floating-point format.

The legal range for voltage values is specified by the [SOURce:]VOLTage[:LEVEL][:IMMediate][:AMPLitude] command. Default units are volts.

MINimum and MAXimum cannot be used with this command.

**Comments**
- If block format is used, the most significant byte of each value must be sent first.
- The voltage point list length must be at least four points long but no longer than the reserved length specified by [SOURce:]LIST[1]:[SEGMENT]:DEFINE. If the voltage point list length is less than the reserved length, only the number of points specified by the most recent voltage point and marker pulse list is generated when the waveform segment is output.
- The waveform segment’s marker pulse list length must be the same length as its voltage point list, or must have a length of 1. If not, executing the INITiate:IMMediate command generates Error +1104,"Segment lists of different lengths".
- Changing marker pulse values preserves the waveform segment’s voltage point list, and vice versa.
- The voltage values specified by this command are scaled relative to the full-scale output voltage specified by [SOURce:]VOLTage[:LEVEL][:IMMediate][:AMPLitude] in effect at the time the voltage point list is created. Subsequently changing the full-scale output voltage will change the actual output voltages that are generated, and also the values returned by the query form of this command.
- **Executable when Initiated:** No
- **Coupling Group:** None
- **Related Commands:** [SOURce:]LIST[1]:[SEGMENT]:COMBined, [SOURce:]LIST[1]:[SEGMENT]:VOLTage:DAC, [SOURce:]VOLTage[:LEVEL][:IMMediate][:AMPLitude]
- **:*RST Condition:** Unaffected
- **Power-On Condition:** No waveform segments are defined
Example  Defining a Waveform Segment Voltage Point List

```
LIST:SEL  ABC  Selects waveform segment ABC.
LIST:DEF 8  ABC is 8 points long.
LIST:VOLT .5,1,.5,0,-.5,-1,-.5,0  Defines waveform voltages.
```

[:SEGMen]t]:VOLTage:DAC

[:SOURce:]LIST[1][:SEGMen]t]:VOLTage:DAC  \(<voltage\_list>\) defines the series of output voltage points that constitute a waveform segment. The points are specified in terms of digital-to-analog converter (DAC) codes.

**Parameters**

The \(<voltage\_list>\) may be either a comma-separated list of DAC codes or an IEEE-488.2 definite or indefinite length block containing the DAC codes in 16-bit integer format.

The DAC code is a 16-bit two’s complement or unsigned number (see the [:SOURce:]ARBitrary:DAC:FORMat command). With [:SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLiitude] 5.11875 V set and a matched output load, the least significant bit (LSB) represents 1.25 mV. The legal range for the DAC codes is -4096 through +4095 for signed numbers, 0 through +8191 for unsigned numbers.

MINimum and MAXimum cannot be used with this command.

**Comments**

- If block format is used, the most significant byte of each value must be sent first.
- The voltage point list length must be at least four points long but no longer than the reserved length specified by [:SOURce:]LIST[1][:SEGMen]t]:DEFINE. If the voltage point list length is less than the reserved length, only the number of points specified by the most recent voltage point and marker pulse list is generated when the waveform segment is output.
- The waveform segment’s marker pulse list length must be the same length as its voltage point list or must have a length of 1. If not, executing the INITiate:IMMediate command generates Error +1104,"Segment lists of different lengths".
- Changing marker pulse values preserves the waveform segment’s voltage point list, and vice versa.
- **Executable when Initiated:** No
- **Coupling Group:** None
- **Related Commands:** [:SOURce:]LIST[1][:SEGMen]t]:VOLTage,
  [:SOURce:]LIST[1][:SEGMen]t]:COMBine,
  [:SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLiitude]
- **:*RST Condition:** Unaffected
• **Power-On Condition:** No waveform segments are defined

**Example**  Defining a Waveform Segment Voltage Point List

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARB:_DAC:FORM SIGN</td>
<td>Selects signed DAC code format.</td>
</tr>
<tr>
<td>LIST:SEL ABC</td>
<td>Selects waveform segment ABC.</td>
</tr>
<tr>
<td>LIST:DEF 8</td>
<td>ABC is 8 points long.</td>
</tr>
<tr>
<td>LIST:VOLT:DAC 400,800,400,0,-400,-800,-400,0</td>
<td>Defines waveform voltages.</td>
</tr>
</tbody>
</table>

[:SEGMENT]:VOLTage:POINts?

[:SEGMENT]:VOLTage:POINts? returns a number indicating the length of the currently selected waveform segment’s voltage point list.

**Comments**

- Executable when Initiated: Yes
- Coupling Group: None
- *RST Condition: None
- Power-On Condition: No waveform segments are defined

**Example**  Query Voltage Point List Length

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST:SEL ABC</td>
<td>Selects waveform segment ABC.</td>
</tr>
<tr>
<td>LIST:VOLT:POIN?</td>
<td>Queries voltage point list length.</td>
</tr>
</tbody>
</table>

:SSEQuence:ADDResS?

:SSEQuence:ADDResS? returns the address in the segment sequence memory at which the currently selected segment sequence is located.

**Comments**

- Executable when Initiated: Yes
- Coupling Group: None
- *RST Condition: None
- Power-On Condition: No segment sequences are defined

**Example**  Query Segment Sequence Memory Address

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST:SSEQ:ADDR?</td>
<td>Queries sequence address.</td>
</tr>
</tbody>
</table>
:SEQUence:CATalog?

[SOURce:]LIST[1]:SEQuence:CATalog? returns a comma-separated list of quoted strings, each containing the name of a defined segment sequence. If no segment sequence names are defined, a single null string ("") is returned.

Comments
- Executable when Initiated: Yes
- Coupling Group: None
- *RST Condition: None
- Power-On Condition: No segment sequence names are defined

Example Cataloging Segment Sequence Names

LIST:SSEQ:CAT?

Catalogs segment sequence names.

:SEQUence:COMBined

[SOURce:]LIST[1]:SEQuence:COMBined <combined_list> defines in one step the waveform segment, marker enable, and repetition count lists that constitute a segment sequence.

Parameters
The <combined_list> may be either a comma-separated list of values or an IEEE-488.2 definite or indefinite length block containing the values in 32-bit integer format. Each value has the following format:

<table>
<thead>
<tr>
<th>Bits 31–20</th>
<th>Bit 19</th>
<th>Bit 18</th>
<th>Bit 17</th>
<th>Bits 16–0</th>
</tr>
</thead>
<tbody>
<tr>
<td>repetition count</td>
<td>reserved</td>
<td>marker enable</td>
<td>unused</td>
<td>segment address</td>
</tr>
</tbody>
</table>

The repetition count is 12-bit unsigned value that is (4096 - the desired repetition count): a value of 4095 in these bits indicates 1 repetition; a value of 0 indicates 4096 repetitions. Marker enable is 1 to enable marker pulse generation for that waveform segment. Segment address is the starting address of the segment divided by 8. Use the [SOURce:]LIST[1]:SEGMent:ADDRess? query to obtain the address of a waveform segment.

MINimum and MAXimum cannot be used with this command.

Comments
- If the comma-separated list of values format is used, the values must be in two’s complement format (i.e., values should range from -2147483648 to +2147483647). If block format is used, the most significant byte of each value must be sent first.
- The combined list must be no longer than the reserved length specified by [SOURce:]LIST[1]:SEQuence:DEFine. If the combined list length is less than the reserved length, only the number of points specified by the combined list is generated when outputting the segment sequence.
• Using combined lists is faster than separately defining the waveform segment, marker enable, and repetition count lists.

• Executing this command with waveform segment, marker pulse, and repetition count lists defined with different lengths generates Error -221,"Settings conflict" unless the different length lists are the marker pulse and/or repetition count list and have a length of 1.

• **Executable when Initiated:** No

• **Coupling Group:** None

• **Related Commands:** [SOURce:]LIST[1]:SSEQUence:DWEEl:COUNt, [SOURce:]LIST[1]:SSEQUence:MARKer, [SOURce:]LIST[1]:SSEQUence:SEQUence

• **RST Condition:** Unaffected

• **Power-On Condition:** No segment sequences are defined

**Example**  
**Defining a Segment Sequence Combined List**

```
LIST:SSEQ:SEL  ABC  
LIST:SSEQ:DEF 1  
LIST:SSEQ:COMB -786432  
```

Selects sequence ABC.  
ABC is 1 point long.  
Outputs segment at address 0 one time with markers enabled.

**:SSEQUence:COMBined:POINts?**

[SOURce:]LIST[1]:SSEQUence:COMBined:POINts? returns a number indicating the length of the currently selected segment sequence’s combined waveform segment, marker pulse, and repetition count list.

**Comments**

• Executing this command with waveform segment, marker pulse, and repetition count lists defined with different lengths generates Error -221,"Settings conflict" unless the different length lists are the marker pulse and/or repetition count list and have a length of 1.  In this case, the length of the waveform segment list is returned.

• **Executable when Initiated:** Yes

• **Coupling Group:** None

• **RST Condition:** None

• **Power-On Condition:** No waveform segments are defined

**Example**  
**Query Combined Point List Length**

```
LIST:SSEQ:SEL  ABC  
LIST:SSEQ:COMB:POIN?  
```

Selects sequence ABC.  
Queries combined point list length.
[:SSEQ] :DEF

[SOURce:]LIST[1]:SSEQ :DEF <length> reserves enough segment sequence memory for a segment sequence of length segment names for the sequence currently selected by [SOURce:]LIST[1]:SEL.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;length&gt;</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

The segment sequence memory can store a maximum of 32,768 points (32,767 points if a square, ramp, or triangle wave exists). All defined segment sequences share this memory. Any one sequence can use any part of or all of this memory.

MINimum reserves 1 point; MAXimum reserves the largest available contiguous piece of segment sequence memory.

Comments

• Once a segment sequence has been DEFined, it must be deleted ([SOURce:]LIST[1]:SEL:DEL[:SELected] command) before its reserved length may be redefined. The contents and length of the list may be changed repeatedly without re-executing the DEFine command.

• By using the [SOURce:]LIST[1]:SSEQ:DWEL:COUNt command, up to 4096 repetitions of a waveform segment can take only one point in the segment sequence memory. This factor should be considered when reserving segment sequence memory space.

• [SOURce:]LIST[1]:SSEQ :DEF initializes the segment sequence’s waveform segment list to a zero current length and the repetition count and marker enable lists to a length of 1 with a value of 1 (single repetition of each segment, marker pulse generation enabled for all segments).

• Executable when Initiated: Yes

• Coupling Group: None

• Related Commands: [SOURce:]LIST[1]:SEL

• *RST Condition: Unaffected

• Power-on Condition: No segment sequences are defined.

Example

Reserving Memory for a Segment Sequence

LIST:SSEQ:SEL ABC
Selects sequence ABC.

LIST:SSEQ:DEF 1024
Reserves 1024 points for ABC.
**:SSEQuence:DELeTe:ALL**

[SOURce:]LIST[1]:SSEQuence:DELeTe:ALL deletes all defined segment sequence definitions from memory and makes all of the sequence memory available for new segment sequence definitions. "In use" sequences cannot be deleted.

**Comments**
- Use [SOURce:]LIST[1]:SSEQuence:DELeTe[:SELectioned] to delete a single segment sequence definition.

- **Executable when Initiated:** No
- **Coupling Group:** None
- **Related Commands:** [SOURce:]LIST[1]:SSEQuence:DELeTe[:SELectioned]
- ***RST Condition:** None
- **Power-On Condition:** No segment sequences are defined

**Example**

Deleting All Segment Sequences

LIST:SSEQ:DEL:ALL

* Deletes all segments.

**:SSEQuence:DELeTe[:SELectioned]**

[SOURce:]LIST[1]:SSEQuence:DELeTe[:SELectioned] deletes a single segment sequence definition and makes its memory available for new segment sequence definitions.

**Comments**
- Use [SOURce:]LIST[1]:SSEQuence:DELeTe:ALL to delete all segment sequence definitions with one command.

- **Executable when Initiated:** No
- **Coupling Group:** None
- **Related Commands:** [SOURce:]LIST[1]:SSEQuence:DELeTe:ALL, [SOURce:]LIST[1]:SSEQuence:SELect
- ***RST Condition:** None
- **Power-On Condition:** No segment sequences are defined

**Example**

Deleting a Segment Sequence

LIST:SSEQ:SEL ABC

Selects segment sequence ABC.

LIST:SSEQ:DEL

Deletes segment.
:SEQUence:DWELL:COUNT

[SOURce:]LIST[1]:SEQUence:DWELL:COUNT <repetition_list> defines, for each waveform segment of a segment sequence, how many times the waveform segment will be output before advancing to the next segment in the sequence.

Parameters

The <repetition_list> may be either a comma-separated list of repetition counts or an IEEE-488.2 definite or indefinite length block containing the counts in 16-bit integer format. The legal range for the counts is 1 to 4096.

MINimum and MAXimum cannot be used with this command.

Comments

• If block format is used, the most significant byte of each value must be sent first.

• The segment sequence’s repetition count list length must be the same length as its waveform segment and marker enable lists or must have a length of 1. If not, executing INITiate:IMMediate generates Error +1114,"Sequence lists of different lengths".

• A repetition count list of length 1 is treated as though it were the same length as the waveform segment list, with all repetition count values the same as the specified value.

• Changing repetition count values preserves the waveform segment and marker enable lists, and vice versa.

• Executable when Initiated: No

• Coupling Group: None

• Related Commands: [SOURce:]LIST[1]:SEQUence:COMBined

• *RST Condition: Unaffected

• Power-On Condition: No segment sequences are defined

Example

Defining a Segment Sequence Repetition Count List

LIST:SSEQ:SEL ABC
LIST:SSEQ:DEF 8
LIST:SSEQ:SEQ A,B,C,D,E,F,G,H
LIST:SSEQ:DWELL:COUN 6,1,1,1,1,1,1,1

Selects sequence ABC.

ABC is 8 points long.

Defines segment sequence.

Outputs segment A six times, others once.
**:SSEQuence:DWELI:COUNT:POINts?**

[SOURce:]LIST[1]:SSEQuence:DWELI:COUNT:POINts? returns a number indicating the length of the currently selected segment sequence’s repetition count list.

**Comments**
- Executable when Initiated: Yes
- Coupling Group: None
- *RST Condition: None
- Power-On Condition: No segment sequences are defined

**Example**  
**Query Repetition Count List Length**

```
LIST:SSEQ:SEL ABC
LIST:SSEQ:DWEL:COUN:POIN?
```

Sets segment sequence ABC.

Queries repetition count list length.

---

**:SSEQuence:FREE?**

[SOURce:]LIST[1]:SSEQuence:FREE? returns information on segment sequence memory availability and usage. The return data format is:

```
<numic_value>,<numeric_value>
```

The first numeric value shows the amount of segment sequence memory available in points; the second, the amount of segment sequence memory used in points.

**Comments**
- Executable when Initiated: Yes
- Coupling Group: None
- *RST Condition: None
- Power-On Condition: All of the segment sequence memory is available

**Example**  
**Querying Segment Sequence Memory Usage**

```
LIST:SSEQ:FREE?
```

Queries segment memory usage.
**:SSEQUence:MARKer**

[SOURce:]LIST[1]:SSEQUence:MARKer  <marker_list> defines, for each waveform segment of a segment sequence, whether the HP E1445A may output the marker pulses defined by the marker list for that waveform segment.

**Parameters**

The <marker_list> may be either a comma-separated list of values or an IEEE-488.2 definite or indefinite length block containing the values in 16-bit integer format. A value of 0 disables marker pulse generation for the waveform segment; any non-zero value enables marker pulse generation.

MINimum and MAXimum cannot be used with this command.

**Comments**

- If block format is used, the most significant byte of each value must be sent first.

- Frequently, marker pulse generation is enabled on no more than one waveform segment of a segment sequence. The [SOURce:]LIST[1]:SSEQUence:MARKer:SPOint command is the most efficient way to enable marker pulse generation for a single waveform segment.

- The segment sequence’s marker enable list length must be the same length as its waveform segment and repetition count lists or must have a length of 1. If not, executing INITiate:IMMediate generates Error +1114,"Sequence lists of different lengths”.

- A marker enable list of length 1 is treated as though it were the same length as the waveform segment list, with all marker enable values the same as the specified value.

- Changing marker enable values preserves the waveform segment and repetition count lists, and vice versa.

- **Executable when Initiated:** No

- **Coupling Group:** None

- **Related Commands:** [SOURce:]LIST[1]:SSEQUence:COMBined

- **RST Condition:** Unaffected

- **Power-On Condition:** No segment sequences are defined

**Example**

**Defining a Segment Sequence Marker Enable List**

```
LIST:SSEQ:SEL  ABC
LIST:SSEQ:DEF 8
LIST:SSEQ:SEQ A,B,C,D,E,F,G,H
LIST:SSEQ:MARK 1,0,0,0,1,0,0,0

Selects sequence ABC.
ABC is 8 points long.
Defines segment sequence.
Enables marker output on segments A and E.
```
?:SSEQ:MARK:POINTS

[SOURce:]LIST[1]:SSEQ:MARK:POINTS? returns a number indicating the length of the currently selected segment sequence’s marker pulse list.

Comments

• Executable when Initiated: Yes
• Coupling Group: None
• *RST Condition: None
• Power-On Condition: No segment sequences are defined

Example

Query Marker Pulse List Length

LIST:SSEQ:SEL ABC
Selects segment sequence ABC.

LIST:SSEQ:MARK:POIN?
Queries marker pulse list length.

?:SSEQ:MARK:SPOINT

[SOURce:]LIST[1]:SSEQ:MARK:SPoINT <point> is a short-cut method for defining a marker list with marker pulse generation enabled on a single waveform segment. It creates a marker list whose length is the same as the current waveform segment list, and which enables marker pulse generation only on the segment specified. The waveform segment list must have been previously defined.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;point&gt;</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

The valid range for <point> is 1 through the length of the current waveform segment list. MINimum selects the first segment of the current waveform segment list; MAXimum selects the last segment.

Comments

• Executable when Initiated: No
• Coupling Group: None
• Related Commands: [SOURce:]LIST[1]:SSEQ:MARKer
• *RST Condition: Unaffected
• Power-On Condition: No segment sequences are defined

Example

Creating a Single-Segment Marker List

LIST:SSEQ:SEL ABC
Selects segment sequence ABC.

LIST:SSEQ:DEF 8
ABC is 8 points long.

LIST:SSEQ:SEQ A,B,C,D,E,F,G,H
Defines segment sequence.

LIST:SSEQ:MARK:SPO 3
Enables marker pulse on segment C.
:SEQ:SELect

[SOURce:]LIST[1]:SEQ:SELect <name> selects a segment sequence for subsequent [SOURce:]LIST[1]:SEQ subsystem commands. This command will define the segment sequence name if it is undefined, but does not reserve any segment sequence memory.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;name&gt;</td>
<td>character data</td>
<td>1 through 12 characters</td>
<td>NONE</td>
</tr>
</tbody>
</table>

NONE selects no segment sequence

Comments

- Legal names must start with an alphabetic character and contain only alphabetic, numeric, and underscore ("_") characters. Alphabetic character case (upper versus lower) is ignored. No segment sequence may have the same name as any waveform segment.

- A maximum of 128 segment sequence names may exist at any time. Use the [SOURce:]LIST[1]:SEQ:DELEte:ALL or SELEcted commands to delete names that are no longer needed.

- Executable when Initiated: Yes

- Coupling Group: None

- Power-On Condition: SOURce:LIST1:SEQ:SELEct NONE

- *RST Condition: Unaffected

Example Selecting a Segment Sequence

LIST:SEQ:SEL  ABC  \(Selects \)segment sequence ABC
[:SSEQuence:SEQuence] defines the ordered sequence of waveform segments that constitute a full waveform.

Parameters
- The <segment_list> is a comma-separated list of waveform segment names. The waveform segment names must have been previously defined.

Comments
- The maximum length of the segment sequence is 32,768 points. By using the [:SOURce:]LIST[:1]:SSEQuence:DWEL:COUNt command, up to 4096 repetitions of a segment sequence name take only one point in the segment sequence memory.

- Executable when Initiated: No
- Coupling Group: None
- *RST Condition: Unaffected

Example
Defining a Segment Sequence

LIST:SSEQ:SEQ A,B,C  

Defines segment sequence.

[:SSEQuence:SEQuence:SEGMents?] returns a number indicating the length of the currently selected segment sequence’s waveform segment list.

Comments
- Executable when Initiated: Yes
- Coupling Group: None
- *RST Condition: None
- Power-On Condition: No segment sequences are defined

Example
Query Segment Sequence Length

LIST:SSEQ:SEQ:SEGM?  

Queries segment sequence length.
The [SOURce:]LIST2 subsystem defines the sample rate or frequencies list to be generated when [SOURce:]FREQuency[1]:MODE is set to LIST. Frequency list generation requires that TRIGger[:START]:SOURce INTernal1 and [SOURce:]FREQuency[1]:MODE LIST be set. Frequency list generation is started by a sweep arm (ARM:SWEep subsystem) and is advanced by a sweep advance trigger (TRIGger:SWEep subsystem).

### Subsystem Syntax

[SOURce:]LIST2

:FORMat [:DATA] <format>[,<length>]

:SREQuency <freq_list>

:POINts? [query only]

### :FORMat[:DATA]

[SOURce:]LIST2:FORMat[:DATA] <format>[,<length>] specifies the format of frequency list return data for the SOURce:LIST2:FREQuency command. The available frequency list return data formats are:

- **ASCii**: Returns the frequency list as NR3 numbers as defined in IEEE-488.2.
- **REAL**: Returns data in IEEE-488.2 definite block format containing the frequency values in IEEE-754 64-bit floating-point format.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;format&gt;</td>
<td>discrete</td>
<td>ASCII</td>
<td>REAL</td>
</tr>
<tr>
<td>&lt;length&gt;</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

If ASCII format is specified, <length> must either be omitted or must be 10 (or MINimum or MAXimum). If REAL format is specified, <length> must either be omitted or must be 64 (or MINimum or MAXimum).

### Comments

- **Executable when Initiated**: Query form only
- **Coupling Group**: None
- **Related Commands**: [SOURce:]LIST2:FREQuency
- ***RST Condition**: SOURce:LIST2:FORMat:DATA ASCII

### Example

**Setting REAL Return Data Format**

LIST:FORM REAL

Sets real format.
[:FREQuency]

[SOURce:]LIST2:FREQuency <freq_list> defines the list of sample rates or frequencies to be generated when SOURce:FREQuency[1]:MODE is set to LIST.

Parameters

The <freq_list> has one of the two following formats:

1. A comma-separated list of frequency values.
2. An IEEE-488.2 definite or indefinite length block containing the frequency values in IEEE-754 64-bit floating-point format.

The maximum length of the list is 256 frequency values.

The legal range for frequency values is given below. Default units are hertz. MINimum and MAXimum cannot be used with this command.

The minimum frequency is 0 Hz for all waveform shapes.

- **Arbitrary Waveforms and Sine Wave Outputs**: The maximum frequency is the current reference oscillator frequency divided by 4.
- **Square Wave Outputs**: The maximum frequency is the current reference oscillator frequency divided by 16.
- **Ramps and Triangle Outputs**: The maximum frequency is the current reference oscillator frequency divided by 4 further divided by the SOURce:RAMP:POINts value.

For non-sine wave outputs, multiply the maximum frequency by 2 if frequency doubling is in effect (see the SOURce:FREQuency[1]:RANGe command).

Comments

- When changing the frequency list length when SOURce:FREQuency[1]:MODE LIST is set, the SOURce:SWEep:TIME or the TRIGger:SWEep:TIMer value remains the same, depending on which command was most recently sent. The other value is changed based on the new frequency list length.

- **Executable when Initiated**: Query form only
- **Coupling Group**: Frequency
- **Related Commands**: TRIGger[:STARt]:SOURce, SOURce:FREQuency[1]:MODE, SOURce:SWEep
- ***RST Condition**: Unaffected
- **Power-On Condition**: No frequency list is defined

Example

Defining a Frequency List

LIST2:FREQ 1000,10e3,100e3,1 MHz  Defines the frequency list.
[:FREQuency:POINts?]

[:SOURce:]LIST2[:FREQuency:POINts?] returns a number that shows the length of the currently defined frequency list.

Comments

• Executable when Initiated: Yes

• Coupling Group: None

• *RST Condition: Unaffected

• Power-On Condition: No frequency list is defined

Example Query the Frequency List Length

LIST:FREQ2:POIN? Queries frequency list length.
The [SOURce:]MARKer subsystem controls:

- Which signal is routed to the “Marker Out” BNC.
- The polarity of the “Marker Out” BNC signal.
- Which signals, if any, are routed to the VXIbus ECL trigger lines.

**Subsystem Syntax**

```
[SOURce:]MARKer
:ECLTrg<n>:FEED <source>
   [:STATe] <mode>
   :FEED <source>
   :POLarity <polarity>
   [:STATe] <mode>
```

**:ECLTrg<n>:FEED**

```
[SOURce:]MARKer:ECLTrg<n>:FEED <source> selects the marker source for the specified VXIbus ECL trigger line (ECLTRG0 or ECLTRG1). The available sources are:

- “ARM[:STARt|:SEQuence[1]:LAYer[1]]”: For arbitrary waveforms, the marker level changes with the first waveform point of the first repetition. A marker pulse is then output with the last waveform point of each repetition. For sine waves, the marker is a 50% duty cycle square wave at the sine wave frequency.

- “ARM[:STARt|:SEQuen ce[1]:LAYer2”: Once a start arm is received, the marker is asserted when the first amplitude point is triggered. The marker is unasserted with the last amplitude point of the last waveform repetition, or following an ABORt.

- “[SOURce:]FREQuency[1]:CHANge”: Outputs a one sample period wide marker pulse that is output after a frequency change occurs. This shows that the new steady state frequency has been reached.

- “[SOURce:]LIST[1]”: Outputs marker pulses specified by the [SOURce:]LIST[1]:SEGMen[t]:MARKer and SSEQUence:MARKer commands. The pulse is normally one sample period wide, but may be widened by placing markers on consecutive output points. This source is only useful with [SOURce:]FUNCTION[:SHApe] USER (i.e., arbitrary waveform output).

- “[SOURce:]PM:DEViation:CHANge”: Outputs a one sample period wide marker pulse that is output after a phase change occurs. This shows that the new phase has been reached.

- “[SOURce:]ROSCillator”: The reference oscillator as selected by [SOURce:]ROSCillator:SOURce.

- “TRIGger[:STARt|:SEQuence[1]]”: Outputs a nominal 12 nS marker pulse for each point of the segment list.
Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;source&gt;</td>
<td>string</td>
<td>“ARM[:START</td>
<td>:SEQ uence[1]]:LAYer[1]”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“ARM[:START</td>
<td>:SEQ uence[1]]:LAYer2”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“[SOURce:]FREQuency[1]:CHANge”</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>“[SOURce:]LIST[1]”</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>“[SOURce:]PM:DEViation:CHANge”</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>“[SOURce:]ROSCillator”</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>“TRIGger[:START</td>
<td>:SEQ uence[1]]”</td>
</tr>
</tbody>
</table>

Comments

- Executable when Initiated: Yes

- Coupling Group: None

- Related Commands: [SOURce:]MARKer:ECLTrg<n>:STATE

- *RST Condition:
  [SOURce:]MARKer:ECLTrg0:FEED “ARM[:START|:SEQ uence[1]]:LAYer[1]”,
  [SOURce:]MARKer:ECLTrg1:FEED “TRIGger[:START|:SEQ uence[1]]”

Example

Setting the VXI ECLTRG0 Trigger Line Source

MARK:ECLT0:FEED “SOUR:LIST”  \(Sets\ \text{marker\ list\ as\ source.}\)

:ECLTrg<n>:STATE

[SOURce:]MARKer:ECLTrg<n>:STATE  \(<mode>\)  enables or disables the routing of the selected marker signal ([SOURce:]MARKer:ECLTrg<n>:FEED command) to the specified VXIbus ECL trigger line (ECLTRG0 or ECLTRG1).

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mode&gt;</td>
<td>boolean</td>
<td>OFF</td>
<td>ON</td>
</tr>
</tbody>
</table>

Comments

- Executable when Initiated: Yes

- Coupling Group: None

- Related Commands: [SOURce:]MARKer:ECLTrg<n>:FEED

- *RST Condition: SOURce:MARKer:ECLTrg<n>:STATE OFF

Example

Enabling Marker Output to ECLTRG0*

MARK:ECLT0 ON  \(Enables\ ECLTRG0*.\)
[SOURce:]MARKer:FEED <source> selects the source for the front panel
“Marker Out” BNC. The available sources are:

- “ARM[:START][:SEQUence[1]][:LAYer[1]]”: For arbitrary waveforms, the marker
  level changes with the first waveform point of the first repetition. A marker pulse is
  then output with the next-to-last waveform point of each repetition. For sine waves,
  the marker is a 50% duty cycle square wave at the sine wave frequency.

- “ARM[:START][:SEQUence[1]]:LAYer2”: Once a start arm is received, the
  marker is asserted when the first amplitude point is triggered. The marker is
  unasserted with the last amplitude point of the last waveform repetition, or
  following an ABORt.

- “[SOURce:]FREQuency[1]:CHANge”: Outputs a one sample period wide
  marker pulse that is output after a frequency change occurs. This shows that the
  new steady state frequency has been reached.

- “[SOURce:]LIST[1]”: Outputs marker pulses specified by the
  [SOURce:]LIST[1]:SEGment:MARKer and SEQuence:MARKer commands. The
  pulse is normally one sample period wide, but may be widened by placing
  markers on consecutive output points. This source is only useful with
  [SOURce:]FUNCTION:SHAPE USER (i.e., arbitrary waveform output).

- “[SOURce:]PM:DEVIation:CHANge”: Outputs a one sample period wide
  marker pulse that is output after a phase change occurs. This shows that the
  new phase has been reached.

- “[SOURce:]ROSCillator”: The reference oscillator as selected by
  [SOURce:]ROSCillator:SOURce.

- “TRIGger[:START][:SEQUence[1]]”: Outputs a nominal 12 nS marker pulse for
  each point of the segment list.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;source&gt;</td>
<td>string</td>
<td>“ARM[:START][:SEQUence[1]][:LAYer[1]]”</td>
<td>none</td>
</tr>
</tbody>
</table>

Comments

- Executable when Initiated: Yes
- Coupling Group: None
- Related Commands: [SOURce:]MARKer:POLarity, [SOURce:]MARKer:[STATe]
- *RST Condition:
  SOURce:MARKer:FEED “ARM[:START][:SEQUence[1]][:LAYer[1]]”
Example Setting the “Marker Out” BNC Source

MARK:FEED “SOUR:LIST” *Sets marker list as source.

:POLarity

[SOURce:]MARKer:POLarity <polarity> selects the polarity of the marker signal at the front panel “Marker Out” BNC. NORMAL polarity selects an active high marker output; INVerted an active low output.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;polarity&gt;</td>
<td>discrete</td>
<td>INVerted</td>
<td>NORMAL</td>
</tr>
</tbody>
</table>

Comments

• Executable when Initiated: Yes
• Coupling Group: None
• Related Commands: [SOURce:]MARKer:FEED, [SOURce:]MARKer[:STATe]
• *RST Condition: SOURce:MARKer:POLarity NORMAL

Example Setting the “Marker Out” BNC Polarity

MARK:POL INV *Sets active low output.

[:STATe]

[SOURce:]MARKer[:STATe] <mode> enables or disables the routing of the currently selected marker signal ([SOURce:]MARKer:FEED command) to the front panel “Marker Out” BNC.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mode&gt;</td>
<td>boolean</td>
<td>OFF</td>
<td>0</td>
</tr>
</tbody>
</table>

Comments

• Executable when Initiated: Yes
• Coupling Group: None
• Related Commands: [SOURce:]MARKer:FEED, [SOURce:]MARKer:POLarity
• *RST Condition: SOURce:MARKer:STATe ON

Example Enabling Marker Output to “Marker Out” BNC

MARK ON Enables “Marker Out” BNC.
The [SOURce:]PM (Phase Modulation) subsystem controls the modulation for sine wave output (only). Phase modulation is not possible with other waveform shapes.

Subsystem Syntax

```
[SOURce:]
PM
[:DEViation] <phase>
:SOURce <source>
:STATe <mode>
:UNIT
[:ANGLe] <units>
```

[:DEViation]

```
[SOURce:]PM[:DEViation] <phase> sets the modulation DEViation for a sine wave output when [SOURce:]PM:SOURce is set to INTernal.
```

The query form returns the amplitude in terms of the default units, specified by the [SOURce:]PM:UNIT[:ANGLe] command.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;phase&gt;</td>
<td>numeric</td>
<td>-π through π</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

MINimum selects -π; MAXimum selects π.

The default units for DEViation are specified by the [SOURce:]PM:UNIT[:ANGLe] command.

Acceptable units are <suffix_multiplier>RAD (radians) and <suffix_multiplier>DEG (degrees).

Comments

- **Executable when Initiated:** Yes
- **Coupling Group:** None
- **Related Commands:** [SOURce:]PM:SOURce, [SOURce:]PM:UNIT[:ANGLe]
- ***RST Condition:** SOURce:PM:DEViation 0

Example

**Setting Phase Deviation**

```
PM:DEV 180 DEG
```

Sets deviation to 180°.
[SOURce:]PM

:SOURce

[SOURce:]PM:SOURce \(<source>\) selects the source for phase modulation data.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;source&gt;)</td>
<td>discrete</td>
<td>DPORT</td>
<td>INTernal</td>
</tr>
</tbody>
</table>

Comments

The available sources are:

- **DPORT**: The HP E1445A’s front panel “Digital Port In” connector.
- **LBUS**: The VXIbus local bus.
- **INTERNAL**: The [SOURce:]PM[:DEViation] command.
- **VXI**: The VXIbus data transfer bus.

- When the source for phase deviation data is the VXIbus data transfer bus, the least significant byte of the data should be written either in the least significant bits of a word to offset 176 decimal (B0 hex). The most significant byte should be written in the least significant bits of a word to offset 178 decimal (B2 hex). After both bytes are written, a word write of any data to offset 138 decimal (8A hex) is required to activate the new phase deviation.

- Phase deviation may be changed at a maximum rate of one change every 5 reference oscillator cycles or 2 MHz, whichever is less.

- **Executable when Initiated**: Yes

- **Coupling Group**: Frequency

- **Related Commands**: [SOURce:]PM[:DEViation], [SOURce:]PM:STATe

- ***RST Condition**: SOURce:PM:SOURce  INTERNAL

Example

Setting Modulation Source

PM:SOUR DPOR

Sets “Digital Port In” connector as modulation source.
[:STATe]

[:STATe] <mode> enables or disables phase modulation for sine wave output. Phase modulation is always disabled for other waveform shapes.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mode&gt;</td>
<td>boolean</td>
<td>OFF</td>
<td>0</td>
</tr>
</tbody>
</table>

Comments

- **Executable when Initiated**: Yes
- **Coupling Group**: Frequency
- **Related Commands**: [SOURce:]FUNCTION[:SHPe]
- ***RST Condition**: SOURce:PM:STATe OFF

Example

Enabling Phase Modulation

```
FUNC:SHAP SIN
PM:STAT ON
INIT
PM:DEV .78648
```

[:UNIT[:ANGL]]

[:UNIT[:ANGL]] <units> sets the default angle units for subsequent [SOURce:]PM[:DEViation] commands. The available default units are:

- **DEG**: Degrees
- **RAD**: Radians

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;units&gt;</td>
<td>discrete</td>
<td>DEG</td>
<td>RAD</td>
</tr>
</tbody>
</table>

Comments

- **Executable when Initiated**: Yes
- **Coupling Group**: None
- **Related Commands**: [SOURce:]PM[:DEViation]
- ***RST Condition**: SOURce:PM:UNIT:ANGL RAD

Example

Setting the Default Angle Units

```
PM:UNIT:ANGL DEG
```

Sets default units to degrees.
The [SOURce:]RAMP subsystem selects the polarity of ramp waveforms, and the number of points on generated ramps and triangle waveforms.

**Subsystem Syntax**

```
[SOURce:]
RAMP
    :POINts <number>
    :POLarity <polarity>
```

**:POINts**

[SOURce:]RAMP:POINts <number> specifies the number of points to be used to generate the stepped ramp or triangle waveform.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

The valid range for <number> is 4 through the length of the largest available contiguous piece of waveform segment memory.

MINimum selects 4 points; MAXimum selects the largest available contiguous piece of waveform segment memory or 262,144 points, whichever is less (4 points minimum).

**Comments**

- For triangle waves, make <number> a multiple of 4 for best waveform shape.
- When [SOURce:]FUNCTION[:SHApe] RAMP or TRIangle is selected, the greater of the [SOURce:]RAMP:POINts value and 8 points of contiguous waveform segment memory must be available. When [SOURce:]FUNCTION[:SHApe] SQUARE is selected, 8 points of contiguous waveform segment memory must be available. Attempting to select one of these functions with less contiguous waveform segment memory available, or to set [SOURce:]RAMP:POINts to a value larger than the largest contiguous amount of available waveform segment memory when ramp or triangle wave output is selected, will generate Error +1000, “Out of memory”.
- **Executable when Initiated**: Query form only
- **Coupling Group**: Frequency and voltage
- **Related Commands**: [SOURce:]FUNCTION[:SHApe]
- **:*RST Condition**: SOURce:RAMP:POINts 100

**Example**

Setting Ramp Length

```
RAMP:POIN 1000
```

Sets ramp length.
[SOURce:]RAMP:POLarity <polarity> selects the polarity of the ramp, triangle, or square wave. For ramps, NORMal generates a positive-going ramp; INVerted generates a negative-going ramp. For triangles, NORMal generates a triangle with an initial positive-going slope; INVerted generates an initial negative-going slope. For square waves, NORMal generates a waveform with initial voltage being the more positive voltage; INVerted generates the more negative voltage first.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;polarity&gt;</td>
<td>discrete</td>
<td>INVerted</td>
<td>NORMal</td>
</tr>
</tbody>
</table>

### Comments
- **Executable when Initiated:** Query form only
- **Coupling Group:** Voltage
- **Related Commands:** [SOURce:]FUNCtion:[SHAPe]
- ***RST Condition:** SOURce:RAMP:POLarity NORMal

### Example
**Selecting Ramp Polarity**

```plaintext
FUNC:SHAP RAMP
RAMP:POL INV
```

*Selects ramp output.*

*Selects negative-going ramp.*
The [SOURce:]ROSCillator subsystem controls the reference oscillator’s source and indicates the frequency of an external oscillator. The HP E1445A uses the source and frequency information to generate sample output rate for arbitrary waveforms or waveform frequency for ramp, sine, square, and triangle wave output.

Subsystem Syntax

```
[SOURce:]ROSCillator
    :FREQuency
    :EXTernal <frequency>
    :SOURce <source>
```

:SOFReQuency:EXTernal

[SOURce:]ROSCillator:FREQuency:EXTernal <frequency> indicates to the HP E1445A the frequency of an external reference oscillator source. The [SOURce:]FREQuency[1] and [SOURce:]FREQuency2 subsystems use this value to generate sample rate and waveform frequencies when [SOURce:]ROSCillator:SOURce is set to EXTernal or ECLTrgn.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;frequency&gt;</td>
<td>numeric</td>
<td>1 Hz through 42.94967296 MHz</td>
<td>Hz</td>
</tr>
</tbody>
</table>

MINimum selects 1 Hz; MAXimum selects 42.94967296 MHz.

Comments

- Indicating an incorrect frequency for an external reference oscillator will cause incorrect sample rate and waveform frequencies to be generated by the [SOURce:]FREQuency[1] and [SOURce:]FREQuency2 subsystems.
- **Executable when Initiated:** Query form only
- **Coupling Group:** Frequency
- ***RST Condition:** SOURce:ROSCillator:FREQuency:EXTernal 42.94967296 MHz

Example

**Specifying the External Reference Oscillator Frequency**

```
ROSC:FREQ:EXT 5 MHZ
```

*External oscillator is 5 MHz.*
:SOURCE

[SOURce:]ROSCillator:SOURce <source> selects the reference oscillator source.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;source&gt;</td>
<td>discrete</td>
<td>CLK10</td>
<td>ECLTrg0</td>
</tr>
</tbody>
</table>

Comments

- The available sources are:
  - **CLK10**: The VXIbus CLK10 (10 MHz) line.
  - **EXTernal**: The HP E1445A’s front panel “Ref/Sample In” BNC.
  - **ECLTrg0** and **ECLTrg1**: The VXIbus ECL trigger lines.
  - **INTernal[1]**: The internal 42.94967296 MHz oscillator. Using this oscillator in conjunction with the [SOURce:]FREQuency[1] subsystem gives a resolution of .01 Hz for sine waves and arbitrary waveform sample rates.
  - **INTernal2**: The internal 40 MHz oscillator. Using this oscillator in conjunction with the [SOURce:]FREQuency2 subsystem allows that subsystem to exactly produce frequencies such as 1, 5, 10, and 20 MHz for arbitrary waveform sample rates.

- The reference oscillator is used to generate the sample rate and waveform frequencies specified in the [SOURce:]FREQuency[1] and [SOURce:]FREQuency2 subsystems.

- Use [SOURce:]ROSCillator:FREQuency:EXTernal to indicate the frequency of an external reference oscillator.

- **Executable when Initiated**: Query form only

- **Coupling Group**: Frequency

- **Related Commands**: [SOURce:]ROSCillator:FREQuency:EXTernal, [SOURce:]FREQuency[1] commands, [SOURce:]FREQuency2 commands

- **RST Condition**: SOURce:ROSCillator:SOURce INTernal1

Example

Setting the Reference Oscillator Source

ROSC:SOUR CLK10

Selects VXI CLK10 line as oscillator source.
The [SOURce:]SWEep subsystem selects:

- The number of frequency sweeps or repetitions of a frequency list to be performed.
- The direction of a frequency sweep.
- The number of points in a frequency sweep.
- A linear or logarithmic frequency sweep with respect to time.
- The sweep rate for frequency sweeps and frequency lists when TRIGger:SWEep:SOURce TIMer is set.

Frequency sweeping generation requires that TRIGger[:STARt:]SOURce INTERNAL1 and [SOURce:]FREQuency[1]:MODE SWEep be set. A sweep is started by a sweep arm (ARM:SWEep subsystem) and is advanced by a sweep advance trigger (TRIGger:SWEep subsystem).

Subsystem Syntax

[SOURce:]
SWEep
  :COUNT <number>
  :DIRection <direction>
  :POINts <number>
  :SPACing <mode>
  :TIME <time>

SWEep:COUNt

[SOURce:]SWEep:COUNt <number> specifies the number of sweeps or repetitions of a frequency list the HP E1445A will perform after an INITiate:IMMediate command before the sweep subsystem returns to the idle state. This command is equivalent to the ARM:SWEep:COUNt command; either command may be used, and executing either one changes the value of the other.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>numeric</td>
<td>1 through 2147483647</td>
<td>9.9E+37</td>
</tr>
</tbody>
</table>

Minimum selects 1 sweep; Maximum selects 2147483647 sweeps. 9.9E+37 is equivalent to INFinity.

Comments

- Executable when Initiated: Query form only
- Coupling Group: Frequency
- Related Commands: INITiate:IMMediate
- **RST Condition**: SOURce:SWEep:COUNt 1

**Example**  Setting the Sweep Count  
SWE:COUN 10  
*Sets 10 sweeps per INITiate.*

**:DIREction**

[SOURce:]SWEep:DIREction <direction>  
selects the direction of the frequency sweep.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;direction&gt;</td>
<td>discrete</td>
<td>DOWN</td>
<td>UP</td>
</tr>
</tbody>
</table>

**Comments**

- The available directions are:
  - **DOWN**: The sweep starts at the stop frequency specified by [SOURce:]FREQuency[1]:STARt and STOP, or CENTer and SPAN and ends at the start frequency.
  - **UP**: The sweep starts at the start frequency specified by [SOURce:]FREQuency[1]:STARt and STOP, or CENTer and SPAN and ends at the stop frequency.

- **Executable when Initiated**: Query form only

- **Coupling Group**: Frequency

- **Related Commands**: [SOURce:]FREQuency[1]:CENTer, MODE, SPAN, START, and STOP, [SOURce:]SWEep:POINts and SPACing

- **RST Condition**: SOURce:SWEep:DIREction UP

**Example**  Setting the Sweep Direction  
SWE:DIRE DOWN  
*Sweeps down in frequency.*
[SOURce:]SWEep

:POINts

[SOURce:]SWEep:POINts <number> selects the number of points in a frequency sweep.

The frequencies generated by the sweep are evenly spaced linearly or logarithmically, depending on the [SOURce:]SWEep:SPACing setting, between the frequencies specified by [SOURce:]FREQuency[1]:STARt and STOP, or CENTer and SPAN, inclusive.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>numeric</td>
<td>2 through 1073741824</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

MINimum selects 2 points; MAXimum selects 1073741824 points.

Comments

- [SOURce:]SWEep:POINts specifies the number of points with [SOURce:]FREQuency[1]:MODE set to SWEep; the length of the [SOURce:]LIST2:FREQuency list specifies the points with [SOURce:]FREQuency[1]:MODE set to LIST.

- When changing the [SOURce:]SWEep:POINts value when [SOURce:]FREQuency[1]:MODE SWEep set, the [SOURce:]SWEep:TIME or the TRIGger:SWEep:TIMer value remains the same, depending on which command was most recently sent. The other value is changed based on the new SWEep:POINts value.

- Executable when Initiated: Query form only

- Coupling Group: Frequency

- Related Commands: [SOURce:]FREQuency[1]:CENTer, MODE, SPAN, START, and STOP, [SOURce:]SWEep:DIRection and SPACing

- *RST Condition: SOURce:SWEep:POINts 800

Example

Setting the Number of Points in the Sweep

SWE:POIN 100

Sets 100 points in sweep.
**SPACing**

[SOURce:]SWEep:SPACing  `<mode>` selects either linear or logarithmic frequency sweep mode.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mode</code></td>
<td>discrete</td>
<td>LINear</td>
<td>LOGarithmic</td>
</tr>
</tbody>
</table>

**Parameters**

- **Comments**
  - The available modes are:
    - **LINear**: Selects the linear sweep mode. The sample rate or waveform frequency increases or decreases linearly between the start and stop frequencies selected by [SOURce:]FREQuency[1]:STARt and STOP, or CENTer and SPAN.
    - **LOGarithmic**: Selects the logarithmic sweep mode. The sample rate or waveform frequency increases or decreases logarithmically between the start and stop frequencies selected by [SOURce:]FREQuency[1]:STARt and STOP, or CENTer and SPAN.
  - **Executable when Initiated**: Query form only
  - **Coupling Group**: Frequency
  - **Related Commands**: [SOURce:]FREQuency[1]:CENTer, MODE, SPAN, START, and STOP, [SOURce:]SWEep:DIRection and POINts
  - ***RST Condition**: SOURce:SWEep:SPACing LINear

**Example** Setting the Frequency Sweep Spacing

SWE:SPAC LOG

*Selects logarithmic spacing.*
[:TIME]

[SOURce:]SWEep:TIME <number> selects the duration of the sweep or frequency list generation when TRIGger:SWEep:SOURce is set to TIMer. The duration is the time from the start of the sweep or list until when the last frequency begins to be output. The value set by this command is coupled to the TRIGger:SWEep:TIMer command value by the following equation:

\[
\text{TIME} = \text{TIMer} \times (\text{points} - 1)
\]

where points is the [SOURce:]SWEep:POINts value for frequency sweeps, or the length of the frequency list for frequency list generation.

When changing the frequency list length when [SOURce:]FREQuency[1]:MODE LIST is set, or the [SOURce:]SWEep:POINts value when any other MODE is set, the TIME or TIMer value remains the same, depending on which command, [SOURce:]SWEep:TIME or TRIGger:SWEep:TIMer respectively, was most recently sent. The other value is changed based on the new points value.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;time&gt;</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

MINimum selects 1.25 mS * (points - 1); MAXimum selects 4.19430375 S * (points - 1). The above values bound the valid range for time.

### Comments

- When performing multiple sweeps or list generations with ARM:SWEep:SOURce IMMEDIATE set, the last frequency point is output for the same length of time as all other points. The SWEep:TIME value is the time from the start of the sweep or list until the last frequency begins to be output and does not include the time for the last frequency point. Therefore, if a specific sweep repetition time is desired, SWEep:TIME should be set according to the following equation:

\[
\text{SWEep:TIME} = \text{time} \times (\text{points} - 1) / \text{points}
\]

Thus, to set a repetition time of 1 S for a 5 point sweep, SWEep:TIME should be set to .8 S.

- **Executable when Initiated:** Query form only

- **Coupling Group:** Frequency

- **Related Commands:** [SOURce:]LIST2:FREQuency, [SOURce:]SWEep:POINts, TRIGger:SWEep:SOURce, TRIGger:SWEep:TIMer

- **:*RST Condition:** SOURce:SWEep:TIME 1

### Example

**Setting the Duration of the Sweep**

**SWE:TIME 10**

Sets sweep to take 10 seconds.
The [SOURce:]VOLTage subsystem controls the amplitude and offset values for all output waveform shapes.

**Subsystem Syntax**

```
[SOURce:]
VOLTage
[:LEVel][:IMMediate][:AMPLitude] <amplitude>
    [:UNIT] <units>
    [:OFFSet] <offset>
```

[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude> sets the output amplitude when SOURce:FUNCTION[:SHAPE] is set to DC, RAMP, SINusoid, SQUARE, or TRIangle. It sets the positive full-scale output amplitude for arbitrary waveforms (SOURce:FUNCTION[:SHAPE] USER set); the least significant DAC code bit represents 1/4095 of this value.

Output amplitude for ramp, sine, square and triangle wave output may be programmed in volts, peak volts, peak-to-peak volts, RMS volts, or dBM. Output amplitude for DC must be programmed in volts; for arbitrary waveform output, volts or peak volts.

The query form returns the amplitude in terms of the default units, specified by the SOURce:VOLTage[:LEVel][:IMMediate][:AMPLitude]:UNIT[:VOLTage] command.
### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;amplitude&gt;</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

**DC Output:** When a matched load has been specified, MINimum selects -5.12 V; MAXimum selects 5.11875 V.

**Arbitrary Waveform, Ramp, Sine, Square, and Triangle Outputs:**
When a matched load has been specified, if the current offset voltage is less than or equal to 1 V, MINimum selects the equivalent of .16187 V (peak) in the default voltage units; if the current offset voltage is greater than 1 V, MINimum selects the equivalent of 1.02486 V in the current voltage units. MAXimum selects the equivalent of the lesser of (+6.025 V - |output offset value|) [rounded down to a value that is a multiple of .01 dB from 5.11875] and +5.11875 V.

For all waveform shapes, when an open circuit load has been specified, double the all the above voltages.

These values bound the legal range of values for <amplitude>.

Default units are specified by the [SOURce:]VOLTage[:LEVel][:IMMediate]:AMPLitude:UNIT[:VOLTage] command.

For all waveform shapes other than DC output, output amplitude control is implemented as a 0 to 30 dB attenuator with .01 dB resolution. For DC output, the amplitude is generated using the DAC; resolution is .00125 V into a matched load, .0025 V into an open circuit.

For DC output, acceptable units are V (volts). For arbitrary waveform output, acceptable units are V (volts) and VPK. For ramp, sine, square, and triangle outputs, acceptable units are V (volts), VPK (volts peak), VPP (volts peak-to-peak), VRMS (volts RMS), W (watts) and DBM or DBMW (dB referenced to 1 milliwatt). For W, DBM, and DBMW, the amplitude is referenced to the OUTPut[1]:LOAD value; they are meaningless and therefore unavailable if OUTPut[1]:LOAD INFinity is set.

### Comments
- **Related Commands:** OUTPut[1]:LOAD, [SOURce:]FUNCtion[:SHAPe], [SOURce:]VOLTage[:LEVel][:IMMediate]:AMPLitude:OFFSet
- **Executable when Initiated:** Yes
- **Coupling Group:** Voltage
- **RST Condition:** SOURce:VOLTage:LEVel:IMMediate:AMPLitude .16187 V

### Example

**Setting Output Voltage**

```
VOLT 5 VPP
```

Sets output amplitude to 5 volts peak-to-peak.
[:LEVel][:IMMediate][:AMPLitude]:UNIT[:VOLTage]

[:SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude]:UNIT[:VOLTage] <units>
sets the default units for subsequent
[:SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] commands.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;units&gt;</td>
<td>discrete</td>
<td>DBM</td>
<td>DBMW</td>
</tr>
</tbody>
</table>

Comments

- The available default units are:
  - **DBM | DBMW**: dB referenced to 1 milliwatt.
  - **V**: Volts. This is equivalent to VPK for time-varying waveforms.
  - **VPK**: Volts peak
  - **VPP**: Volts peak-to-peak
  - **VRMS**: Volts RMS
  - **W**: Watts

  For W, DBM, and DBMW, the amplitude is referenced to the OUTPut[1]:LOAD value; they are meaningless and therefore unavailable if OUTPut[1]:LOAD INFinity is set.

- Executable when Initiated: Yes

- Coupling Group: None

- Related Commands: OUTPut[1]:IMPedance, OUTPut[1]:LOAD,
  [:SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude]

- *RST Condition:
  SOURce:VOLTage:LEVel:IMMediate:AMPLitude:UNIT:VOLTage  V

Example

Setting the Default Voltage Units

VOLT:UNIT:VOLT  VPP  

Sets default units to volts peak-to-peak.
[:LEVel][:IMMediate]:OFFSet

[:SOURce:]VOLTage[:LEVel][:IMMediate]:OFFSet <offset> sets the output offset voltage for all waveform shapes except DC. Output offset amplitude is programmed in volts.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;offset&gt;</td>
<td>numeric</td>
<td>see below</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

DC Output: When a matched load has been specified, MINimum selects -5.0 V; MAXimum selects +5.0 V.

Arbitrary Waveform, Ramp, Sine, Square, and Triangle Outputs:
When a matched load has been specified, if the output amplitude, in volts, is greater than 1.02426 V (peak), MINimum selects the greater of (-6.025 V + output amplitude value) and -5.0 V, rounded down if needed to a multiple of 2.5 mV; MAXimum selects the lesser of (+6.025 V - output amplitude value) and +5.0 V, again rounded down. If the output amplitude in volts, is less than or equal to 1.02426 V, MINimum selects the greater of (-1.205 V + output amplitude value) and -.99993 V rounded down if needed to a multiple of .499966 mV; MAXimum selects the lesser of (+1.205 V - output amplitude value) and +.99993 V, again rounded down.

For all waveform shapes, when an open circuit load has been specified, double all the above voltages.

The above values bound the legal range for <offset>.

Comments

• Related Commands: [:SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude]

• Executable when Initiated: Yes

• Coupling Group: Voltage

• *RST Condition: [:SOURce:]VOLTage:LEVel:IMMediate:OFFSet 0 V

Example Setting Offset Voltage

VOLT:OFFS 3

Sets offset voltage to 3 volts.
The STATus subsystem controls the SCPI-defined Operation and Questionable Signal status registers. Each is comprised of a Condition Register, an Event Register, an enable mask, and negative and positive transition filters.

Each Status Register works as follows:

When a condition occurs, the appropriate bit in the Condition Register is set or cleared. If the corresponding transition filter is enabled for that bit, the same bit is set in the associated Event Register. The contents of the Event Register and the enable mask are logically ANDed bit-for-bit; if any bit of the result is set, the Summary bit for that register is set in the status byte. The Status Byte Summary bit for the Operation Status Register is bit 7; for the Questionable Signal Status Register, bit 3.

### Operation Status Register

Only bits 0 (calibrating), 3 (sweeping), 6 (waiting for arm), and 8 (initiated) are defined for the HP E1445A. All other bits are always zero.

- **Bit 0 - Calibrating:** Set (1) during the execution of the CALibration[:DC]:BEGin command. Cleared (0) at the end of DC calibration or if calibration is aborted.

- **Bit 3 - Sweeping:** Set (1) while a frequency sweep or list is in progress. Cleared (0) when waveform generation is halted, when frequency sweeping or lists are not selected, and at the end of each sweep or list.

- **Bit 6 - Waiting for Arm:ARM** Set (1) when waiting for a start arm. Cleared (0) when a start arm is accepted or when waveform generation is aborted.

- **Bit 8 - Initiated:** Set (1) by the INITiate:IMMEdiate command. Cleared (0) when waveform generation is complete and the trigger subsystem returns to the idle state.

### Questionable Signal Status Register

Only bits 5 (frequency) and 8 (calibration) are defined. All other bits are always 0.

- **Bit 5 - Frequency:** Set (1) when the [SOURce:]FREQuency2 divide-by-n frequency generator is selected and the generated frequency differs from the specified frequency by greater than 1%. Cleared (0) otherwise.

- **Bit 8 - Calibration:** Set (1) if an error has been detected in the non-volatile calibration memory. Cleared (0) otherwise.
**Subsystem Syntax**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;state&gt;</td>
<td>boolean</td>
<td>OFF</td>
<td>0</td>
</tr>
</tbody>
</table>

**Parameters**

- **Executable when Initiated:** Yes

- **Coupling Group:** None

- **Related Commands:** *OPC, *OPC?, *RST, *WAI, ABORt, INITiate:IMMediate, STATus:PRESet

- **RST Condition:** Unaffected

- **Power-on Condition:** STATus:OPC:INITiate ON

**Example**

**Setting Immediate Completion Mode**

STAT:OPC:INIT OFF  
Completes immediately for *OPC, etc.
:OPERation:CONDition?

**STATus:OPERation:CONDition?** returns the contents of the Operation Condition Register. Reading the register does not affect its contents.

**Comments**
- **Executable when Initiated:** Yes
- **Coupling Group:** None
- **Related Commands:** STATus commands, *SRE, *STB?
- ***RST Condition:** All bits of the Operation Condition Register are cleared as a result of the state present after *RST.

**Example**
Querying the Operation Condition Register

STAT:OPER:COND?

*Queries the Operation Condition Register.*

:OPERation:ENABle

**STATus:OPERation:ENABle** `<unmask>` specifies which bits of the Operation Event Register are included in its Summary bit. The Summary bit is the bit-for-bit logical AND of the Event Register and the unmasked bit(s).

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;unmask&gt;</code></td>
<td>numeric or non-decimal numeric</td>
<td>0 through +32767</td>
<td>none</td>
</tr>
</tbody>
</table>

The non-decimal numeric forms are the #H, #Q, or #B formats specified by IEEE-488.2.

**Comments**
- **Executable when Initiated:** Yes
- **Coupling Group:** None
- **Related Commands:** STATus commands, *SRE, *STB?
- ***RST Condition:** Unaffected
- **Power-on Condition:** STATus:OPERation:ENABle 0

**Example**
Setting the Operation Register Enable Mask

STAT:OPER:ENAB #H0040

*Enables summary on Waiting for Arm bit.*
:OPERation[:EVEN]? returns the contents of the Operation Event Register. Reading the register clears it to 0.

Comments
- The Operation Event Register is also cleared to 0 by the *CLS common command.
- Executable when Initiated: Yes
- Coupling Group: None
- Related Commands: STATus commands, *SRE, *STB?
- *RST Condition: Unaffected
- Power-on Condition: Operation Event Registers are cleared to 0.

Example
Querying the Operation Event Register

STAT:OPER:EVEN? Queries the Operation Event Register.

:OPERation:NTRansition

STATus:OPERation:NTRansition <unmask> sets the negative transition mask. For each bit unmasked, a 1-to-0 transition of that bit in the Operation Condition Register will set the same bit in the Operation Event Register.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;unmask&gt;</td>
<td>numeric or non-decimal numeric</td>
<td>0 through +32767</td>
<td>none</td>
</tr>
</tbody>
</table>

The non-decimal numeric forms are the #H, #Q, or #B formats specified by IEEE-488.2.

Comments
- Executable when Initiated: Yes
- Coupling Group: None
- Related Commands: STATus commands, *SRE, *STB?
- *RST Condition: Unaffected
- Power-on Condition: STATus:OPERation:NTRansition 0

Example
Setting the Operation Register Negative Transition Mask

STAT:OPER:NTR #H0008 Sets the Event bit when sweeping condition is cleared.
:OPERation:PTRansition

STATus:OPERation:PTRansition <unmask> sets the positive transition mask. For each bit unmasked, a 0-to-1 transition of that bit in the Operation Condition Register will set the same bit in the Operation Event Register.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;unmask&gt;</td>
<td>numeric or non-decimal numeric</td>
<td>0 through +32767</td>
<td>none</td>
</tr>
</tbody>
</table>

The non-decimal numeric forms are the #H, #Q, or #B formats specified by IEEE-488.2.

Comments

- Executable when Initiated: Yes
- Coupling Group: None
- Related Commands: STATus commands, *SRE, *STB?
- *RST Condition: Unaffected
- Power-on Condition: STATus:OPERation:PTRansition 32767

Example

Setting the Operation Register Positive Transition Mask

STAT:OPER:PTR #H0040

Sets the event bit when Waiting for Arm condition is set.

:PRESet

STATus:PRESet initializes the Enable Registers and transition masks for the Operation Status and Questionable Signal Status Registers and sets STATus:OPC:INITiate ON. For both Status Registers, the Enable Registers are set to 0, the negative transition masks are set to 0, and the positive transition masks are set to 32767.

Comments

- Executable when Initiated: Yes
- Coupling Group: None
- Related Commands: STATus commands, *SRE, *STB?
- *RST Condition: None

Example

Presetting the Status Subsystem

STAT:PRES

Presets the status subsystem.
:QUESTIONable:CONDition?

STATus:QUESTIONable:CONDition? returns the contents of the Questionable Signal Condition Register. Reading the register does not affect its contents.

Comments
- Executable when Initiated: Yes
- Coupling Group: None
- Related Commands: STATus commands, *SRE, *STB?
- *RST Condition: All bits of the Condition Register are cleared as a result of the state present after *RST, except for the Calibration bit, which will remain set if the condition persists.

Example Querying the Questionable Signal Condition Register

STAT:QUES:COND? Queries Questionable Signal Condition Register.

:QUESTIONable:ENABle

STATus:QUESTIONable:ENABle <unmask> specifies which bits of the Questionable Signal Event Register are included in its Summary bit. The Summary bit is the bit-for-bit logical AND of the Event Register and the unmasked bit(s).

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;unmask&gt;</td>
<td>numeric or non-decimal</td>
<td>0 through +32767</td>
<td>none</td>
</tr>
</tbody>
</table>

The non-decimal numeric forms are the #H, #Q, or #B formats specified by IEEE-488.2.

Comments
- Executable when Initiated: Yes
- Coupling Group: None
- Related Commands: STATus commands, *SRE, *STB?
- *RST Condition: Unaffected
- Power-on Condition: STATus:QUESTIONable:ENABle 0

Example Setting the Questionable Signal Register Enable Mask

STAT:QUES:ENAB #H0040 Enables summary on Waiting for Arm bit.
STATus:QUESTIONable[:EVENt]? returns the contents of the Questionable Signal Event Register. Reading the register clears it to 0.

Comments
- The Event Register is also cleared to 0 by the *CLS common command.
- Executable when Initiated: Yes
- Coupling Group: None
- Related Commands: STATus commands, *SRE, *STB?
- *RST Condition: Unaffected
- Power-on Condition: The Event Register is cleared to 0.

Example Querying the Questionable Signal Event Register

```
STAT:QUES:EVEN?  Queries the Questionable Signal Event Register.
```

:QUESTIONable:NTRansition

STATus:QUESTIONable:NTRansition <unmask> sets the negative transition mask. For each bit unmasked, a 1-to-0 transition of that bit in the Questionable Signal Condition Register will set the same bit in the Questionable Signal Event Register.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;unmask&gt;</td>
<td>numeric or non-decimal numeric</td>
<td>0 through +32767</td>
<td>none</td>
</tr>
</tbody>
</table>

The non-decimal numeric forms are the #H, #Q, or #B formats specified by IEEE-488.2.

Comments
- Executable when Initiated: Yes
- Coupling Group: None
- Related Commands: STATus commands, *SRE, *STB?
- *RST Condition: Unaffected
- Power-on Condition: STATus:QUESTIONable:NTRansition 0

Example Setting the Questionable Signal Register Negative Transition Mask

```
STAT:QUES:NTR #H0008  Sets the Event bit when sweeping condition is cleared.
```
STATus

:QUESTionable:PTRansition

STATus:QUESTionable:PTRansition <unmask> sets the positive transition mask. For each bit unmasked, a 0-to-1 transition of that bit in the Questionable Signal Condition Register will set the same bit in the Questionable Signal Event Register.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;unmask&gt;</td>
<td>numeric or non-decimal numeric</td>
<td>0 through +32767</td>
<td>none</td>
</tr>
</tbody>
</table>

The non-decimal numeric forms are the #H, #Q, or #B formats specified by IEEE-488.2.

Comments

- Executable when Initiated: Yes
- Coupling Group: None
- Related Commands: STATus commands, *SRE, *STB?
- *RST Condition: Unaffected
- Power-on Condition: STATus:QUESTionable:PTRansition 32767

Example

Setting the Questionable Signal Register Positive Transition Mask

STAT:QUES:PTR #H0040

Sets the event bit when Waiting for Arm condition is set.
The SYSTem subsystem returns error messages and the SCPI version number to which the HP E1445A complies.

### Subsystem Syntax

<table>
<thead>
<tr>
<th>Command</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTem</td>
<td>:ERRor?</td>
<td>[query only]</td>
</tr>
<tr>
<td></td>
<td>:VERSion?</td>
<td>[query only]</td>
</tr>
</tbody>
</table>

### :ERRor?

**SYSTem:ERRor?** returns the error messages in the error queue. See Table B-6 in Appendix B for a listing of possible error numbers and messages.

**Comments**

- The HP E1445A places any generated errors into the error queue. The queue is first-in, first out. With several errors waiting in the queue, the SYSTem:ERRor? command returns the oldest unread error message first.

- The error queue can hold 30 error messages. If the HP E1445A generates more than 30 messages that are not read, it replaces the last error message in the queue with Error -350,"Too many errors". No additional messages are placed into the queue until SYSTem:ERRor? reads some messages or the "CLS (clear status) command clears the queue.

- When the error queue is empty, SYSTem:ERRor? returns +0,"No error".

- **Executable when Initiated**: Yes

- ***RST Condition**: Unaffected

- **Power-On Condition**: No errors are in the error queue

### Example

**Reading the Error Queue**

SYST:ERR?  
*Queries the error queue.*
SYSTem

:VERSion?

SYSTem:VERSion? returns the SCPI version number to which the HP E1445A complies: “1991.0”.

Comment

•Executable when Initiated: Yes
•*RST Condition: None

Example Querying the SCPI Revision

SYSST:VERS? Queries SCPI revision.
The TRIGger subsystem operates with the ARM subsystem to control the behavior of the trigger system, as follows:

- The source and slope for generating the individual samples of a waveform.
- The source and slope of the signal that may gate sample generation.
- The source and slope for prematurely stopping one trigger cycle, without aborting the entire trigger system.
- The source for advancing a frequency sweep or list.

### Subsystem Syntax

```
TRIGger
{:STARt{:SEQuence[1]}
  :COUNt <number>
  :GATE
    :POLarity <polarity>
    :SOURce <source>
    :STATe <state>
  [:IMMediate] [no query]
  :SLOPe <edge>
  :SOURce <source>

:STOP{:SEQuence2
  [:IMMediate] [no query]
  :SLOPe <edge>
  :SOURce <source>

:SWEep{:SEQuence3
  [:IMMediate] [no query]
  :LINK <link>
  :SOURce <source>
  :TIMer <period>
```
TRIGger

[:START]:COUNT

TRIGger[:START]:COUNT <number> would normally specify the number of triggers the HP E1445A would accept after an INITiate:IMMediate command before returning the start trigger sequence to the wait-for-arm state. However, since this is equal to the length of the current waveform, and is not configurable here, the only legal value for this command is 9.91e37 or NaN (not a number).

There is no need to send this command. It is included for SCPI compatibility purposes only.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>numeric</td>
<td>9.91e37</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

MINimum and MAXimum select 9.91e37 triggers. 9.91E+37 is equivalent to NAN.

Comments

- **Executable when Initiated:** Query form only
- **Coupling Group:** None
- **Related Commands:** ABORt, INITiate:IMMediate
- ***RST Condition:** TRIGger:STARt:COUNt 9.91e37

Example

Setting the Start Trigger Count

TRIG:COUN  NAN
[:START]:GATE:POLarity

TRIGger[:START]:GATE:POLarity <polarity> selects the polarity of the HP E1445A’s front panel “Stop Trig/FSK/Gate In” BNC which gates the TRIGger:START subsystem. NORMAL polarity selects an active high gate; INVerted polarity selects an active low gate. This polarity is significant only when TRIGger[:START]:GATE SOURce is set to EXTernal. The programmed value is retained but not used when other sources are selected.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;polarity&gt;</td>
<td>discrete</td>
<td>INVerted</td>
<td>NORMAL</td>
</tr>
</tbody>
</table>

Comments

- **Executable when Initiated:** Query form only
- **Coupling Group:** Frequency
- **Related Commands:** TRIGger[:START]:GATE:SOURce
- **:*RST Condition:** TRIGger:START:GATE:POLarity INVerted

Example

**Setting the Sample Gate Polarity**

```
TRIG:START:GATE:POL NORM
```

Sets active high gate.

[:START]:GATE:SOURce

TRIGger[:START]:GATE:SOURce <source> selects the source which gates the TRIGger[:START] subsystem. The TRIGger[:START] subsystem is suspended (no new samples are generated) while the selected gate source is asserted. Normal sample generation resumes when the gate is unasserted.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;source&gt;</td>
<td>discrete</td>
<td>EXTernal</td>
<td>TTLTrg0 through TTLTrg7</td>
</tr>
</tbody>
</table>

Comments

- **The available sources are:**
  - EXTernal: The HP E1445A’s front panel “Stop Trig/FSK/Gate In” BNC connector.
  - TTLTrg0 through TTLTrg7: The VXIbus TTL trigger lines.
- When a VXIbus TTLTrg<n> line is selected as the gate source, the low level on the line asserts the gate. The TRIGger[:START]:GATE:POLarity command selects
TRIGger

the active level for the front panel’s “Stop Trig/FSK/Gate In” BNC when used as the gate source.

- The front panel’s “Stop Trig/FSK/Gate In” BNC is a three-use connector; for FSK control, as a stop trigger source, or as a sample gate source. Only one of these uses may be active at any time.

- If a VXIbus TTLTrg trigger line is used as the sample gate source, then no TTLTrg trigger lines can be used for FSK control or as a stop trigger source.

- **Executable when Initiated**: Query form only

- **Coupling Group**: Frequency

- **Related Commands**: TRIGger[:STARt]:GATE:POLarity, [SOURce:]FREQuency[1]:FSK:SOURce, TRIGger:STOP:SOURce

- **RST Condition**: TRIGger:STARt:GATE:SOURce EXTernal

**Example** Setting the Sample Gate Source

TRIG:GATE:SOUR TTL0

Selects VXIbus trigger line TTLTRG0* as sample gate source.

[STARt]:GATE:STATe

TRIGger[:STARt]:GATE:STATe `<mode>` enables or disables sample gating. When enabled, the TRIGger[:STARt] subsystem is suspended (no new samples are generated) while the gate source selected by TRIGger[:STARt]:GATE:SOURce is asserted. Normal sample generation resumes when the gate is unasserted.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;mode&gt;</code></td>
<td>boolean</td>
<td>OFF</td>
<td>0</td>
</tr>
</tbody>
</table>

**Comments**

- **Executable when Initiated**: Query form only

- **Coupling Group**: Frequency

- **Related Commands**: TRIGger[:STARt]:GATE:SOURce

- **RST Condition**: TRIGger:STARt:GATE:STATe OFF

**Example** Enabling Sample Gating

TRIG:GATE:STAT ON

Enables sample gating.
[:STARt][:IMMediate]

**TRIGger[:STARt][:IMMediate]** immediately advances to the next sample in a waveform regardless of the selected trigger source, provided that the trigger system has been initiated and a start arm received. The selected trigger source remains unchanged.

**Comments**
- Executing this command with the start trigger sequence not in the wait-for-trigger state generates Error -211, "Trigger ignored".
- **Executable when Initiated:** Yes
- **Coupling Group:** None
- **Related Commands:** INITiate[:IMMediate], TRIGger
- **^RST Condition:** None

**Example** Single Stepping a Waveform

- **ARM:** LAY2:SOUR IMM
- **TRIG:** SOUR HOLD
- **INIT**
- **TRIG**

### [:STARt]:SLOPe

**TRIGger[:STARt]:SLOPe** `<edge>` selects the edge (rising or falling) at the HP E1445A’s front panel “Ref/Sample In” BNC to advance the waveform. This edge is significant only with TRIGger[:STARt]:SOURce set to EXTernal. The programmed value is retained but not used when other sources are selected.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;edge&gt;</code></td>
<td>discrete</td>
<td>NEGative</td>
<td>POSitive</td>
</tr>
</tbody>
</table>

**Comments**
- Executable when Initiated: Query form only
- **Coupling Group:** None
- **Related Commands:** TRIGger[:STARt]:SOURce
- **^RST Condition:** TRIGger:STARt:SLOPe POSitive

**Example** Setting the Start Trigger Slope

- **TRIG:SLOP** NEG

  Sets negative trigger slope.
TRIGger

[:STAr]:SOURce

TRIGger[:STAr]:SOURce <source> selects the source that advances the waveform to the next sample point.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;source&gt;</td>
<td>discrete</td>
<td>BUS</td>
<td>ECLTrg0</td>
</tr>
</tbody>
</table>

Comments

- The sources available are:
  - **BUS**: The Group Execute Trigger (GET) HP-IB command or the IEEE-488.2 *TRG common command.
  - **ECLTrg0** and **ECLTrg1**: The VXIbus ECL trigger lines.
  - **EXternal**: The HP E1445A’s front panel “Ref/Sample In” BNC connector.
  - **HOLD**: Suspends sample generation. Use the TRIGger[:STAr][:IMMediate] command to advance the waveform.
  - **INTernal[1]**: The [SOURce]:FREQuency[1] subsystem. Sine wave output ([SOURce]:FUNCTION:[SHAPE] SINusoid set) requires that this source be selected.
  - **INTernal2**: The [SOURce]:FREQuency2 subsystem.
  - **TTLTrg0** through **TTLTrg7**: The VXIbus TTL trigger lines.

- Use the TRIGger[:STAr]:SLOPe command to select the active edge for the front panel “Ref/Sample In” BNC when used as the start trigger source.

- **Executable when Initiated**: No

- **Coupling Group**: Frequency

- **Related Commands**: TRIGger[:STAr]:SLOPe

- **:*RST Condition**: TRIGger:STAr:SOURce INTernal1

Example

**Setting the Start Trigger Source**

TRIG:SOUR EXT

*Trigger source is front panel’s “Ref/Sample In” BNC.*
**STOP[:IMMediate]**

**TRIGger**:**STOP[:IMMediate]** terminates the current start arm cycle at the end of the current waveform repetition regardless of the selected stop trigger source. The command aborts the remaining **ARM[:STARt][:LAYer[1]]:COUNt** repetitions of the current trigger cycle. The start trigger sequence is placed into the wait-for-arm state at the end of the current waveform repetition. The selected stop trigger source remains unchanged.

**Comments**

- Executing this command with the start trigger sequence in the idle or wait-for-arm states generates Error -211, "Trigger ignored".

- If the start trigger sequence is on the last of **ARM[:STARt]:LAYer2:COUNt** trigger cycles, or if **ARM[:STARt]:LAYer2:COUNt 1** is set, **TRIGger**:**STOP[:IMMediate]** places the trigger system in the idle state at the end of the current waveform repetition. An **INITiate[:IMMediate]** command must be executed to restart waveform generation.

- **TRIGger**:**STOP[:IMMediate]** differs from **ABORt** in that **ABORt** terminates all start arm cycles immediately, whereas **TRIGger**:**STOP[:IMMediate]** terminates only the current arm cycle, at the end of the current waveform repetition.

- **Executable when Initiated**: Yes

- **Coupling Group**: None

- **Related Commands**: **ABORt**, **INITiate[:IMMediate]**, **TRIGger[:STARt]:COUNt**

- ***RST Condition**: None

**Example**

**Stopping an Arm Cycle**

```
ARM:LAY2:COUN 5  \hspace{1cm} \text{Allows 5 arms.}
ARM:LAY2:SOUR HOLD \hspace{1cm} \text{Sets manual start arm.}
ARM:COUN 100 \hspace{1cm} \text{Sets 100 repetitions per arm.}
INIT \hspace{1cm} \text{Initiates trigger system.}
ARM:LAY2 \hspace{1cm} \text{Starts arm waveform.}
TRIG:STOP \hspace{1cm} \text{Terminates arm cycle at end of waveform repetition.}
ARM:LAY2 \hspace{1cm} \text{Starts arm waveform again.}
```
The TRIGger STOP SLOPe command selects the edge (rising or falling) on the HP E1445A’s front panel “Stop Trig/FSK/Gate In” BNC which terminates the current start arm cycle at the end of the current waveform repetition. This edge is significant only with TRIGger STOP SOURce set to EXTernal. The programmed value is retained but not used when other sources are selected.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;edge&gt;</td>
<td>discrete</td>
<td>NEGative</td>
<td>POSitive</td>
</tr>
</tbody>
</table>

### Comments
- Executable when Initiated: Query form only
- Coupling Group: Frequency
- Related Commands: TRIGger STOP SOURce
- *RST Condition: TRIGger STOP SLOPe POSitive

### Example

**Setting the Stop Trigger Slope**

```plaintext
TRIG:STOP:SLOP NEG
```

Sets negative stop trigger slope.

The TRIGger STOP SOURce command selects the source that can terminate the current start arm cycle at the end of the current waveform repetition. When the HP E1445A receives a stop trigger, the start trigger sequence is placed into the wait-for-arm state at the end of the current waveform repetition, aborting the remaining ARM[:STARt][:LAYer[1]]:COUNT repetitions of the current arm cycle.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;source&gt;</td>
<td>discrete</td>
<td>BUS</td>
<td>EXTernal</td>
</tr>
</tbody>
</table>

### Comments
- The available sources are:
  - **BUS**: The Group Execute Trigger (GET) HP-IB command or the IEEE-488.2 *TRG common command.
  - **EXTernal**: The HP E1445A’s front panel “Stop Trig/FSK/Gate In” BNC connector.
  - **HOLD**: Suspend stop triggering. Use the TRIGger STOP[:IMMediate] command to terminate a start arm cycle.
  - **TTLTrg0 through TTLTrg7**: The VXIbus TTL trigger lines.
• If a stop trigger is received while the start trigger sequence is in the idle or wait-for-arm states, it is ignored with no error generated.

• If the start trigger sequence is on the last of ARM[:START]:LAYer2:COUNt arm cycles, a stop trigger places the trigger system in the idle state at the end of the current waveform repetition.

• A stop trigger differs from the ABORt command in that ABORt terminates all start arm cycles immediately, whereas a stop trigger terminates only the current arm cycle, at the end of the current waveform repetition.

• Use the TRIGger:STOP:SLOPe command to select the active edge (rising or falling) for the front panel “Stop Trig/FSK/Gate In” BNC when used as the stop trigger source.

• The front panel “Stop Trig/FSK/Gate In” BNC is a three-use connector; for FSK control, as a stop trigger source, or as a sample gate source. Only one of these uses may be active at any time.

• If a VXIbus TTLTrg trigger line is used as the stop trigger source, then no TTLTrg trigger lines can be used for FSK control or as the gating source.

• Executable when Initiated: Query form only

• Coupling Group: Frequency

• Related Commands: ABORt, INITiate[:IMMediate], TRIGger[:START]:COUNt

• *RST Condition: TRIGger:STOP:SOURce HOLD

**Example**  Setting the Stop Trigger Source

TRIG:STOP:SOUR TTL1

*Selects VXIbus trigger line TTLTRG1* as source.

:s:IMmediate

TRIGger:SWEep[:IMMediate]  advances a frequency sweep or list to the next frequency regardless of the selected trigger source. The trigger system must have been initiated and the sweep trigger sequence must be in the wait-for-trigger state. The selected trigger source remains unchanged.

**Comments**  • Executing this command when frequency sweeps or lists are not enabled, or with the sweep trigger sequence not in the wait-for-trigger state generates Error -211,"Trigger ignored".

• Executable when Initiated: Yes

• Coupling Group: none

• Related Commands: INITiate:IMMediate, [SOURce:]SWEep commands
**Example** Advancing a Frequency Sweep

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWE:STAR 1E3;STOP 10E3</td>
<td>Sets sweep frequency limits.</td>
</tr>
<tr>
<td>SWE:POIN 10</td>
<td>Sets 1 kHz steps.</td>
</tr>
<tr>
<td>ARM:LAY2:SOUR IMM</td>
<td>Sets output to start immediately.</td>
</tr>
<tr>
<td>ARM:SWE:SOUR IMM</td>
<td>Sets sweep to start immediately.</td>
</tr>
<tr>
<td>TRIG:SWE:SOUR HOLD</td>
<td>Sets sweep to advance sweep manually.</td>
</tr>
<tr>
<td>INIT</td>
<td>Initiates trigger system.</td>
</tr>
<tr>
<td>TRIG:SWE</td>
<td>Advances to next frequency.</td>
</tr>
</tbody>
</table>

---

### :SWEep:LINK

**TRIGger:SWEep:LINK** `<link>` selects the internal event that advances a frequency sweep or list when **TRIGger:SWEep:SOURce** is set to **LINK**. The only defined internal event to advance a sweep or list is “ARM[:STARt|:SEQuence[1]]:LAYer2”.

There is no need to send this command since there is only one defined internal event. The command is included for SCPI compatibility purposes only.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;link&gt;</code></td>
<td>string</td>
<td>“ARM[:STARt</td>
<td>:SEQuence[1]]:LAYer2”</td>
</tr>
</tbody>
</table>

**Comments**

- **Executable when Initiated:** Yes
- **Coupling Group:** None
- **Related Commands:** TRIGger:SWEep:SOURce
- ***RST Condition:** TRIGger:SWEep:LINK “ARM[:STARt|:SEQuence[1]]:LAYer2”

**Example** Linking the Sweep Advance Trigger

- TRIG:SWE LINK
- TRIG:SWE:LINK "ARM:LAY2"

*Links sweep advance trigger to start arm.*
TRIGger:SWEP:SOURce

`TRIGger:SWEP:SOURce <source>` selects the source that causes a frequency sweep or list to advance to the next frequency.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;source&gt;</code></td>
<td>discrete</td>
<td>BUS</td>
<td>HOLD</td>
</tr>
</tbody>
</table>

### Comments

- **The available sources are:**
  - **BUS**: The Group Execute Trigger (GET) HP-IB command or the IEEE-488.2 *TRG* common command.
  - **HOLD**: Suspend sweep or list frequency advance triggering. Use `TRIGger:SWEep[:IMMediate]` to advance to the next frequency.
  - **LINK**: The next valid start arm advances the sweep or list. Thus, the frequency change always occurs at the start of `ARM[:STARt][:LAYer[1]]:COUNt` repetitions of the waveform.
  - **TIMer**: The `[SOURce:]SWEP:TIME` and `TRIGger:SWEP:TIMer` commands control the sweep or list frequency advance timing.
  - **TTLTrg0** through **TTLTrg1**: The VXIbus TTL trigger lines.

- If `TRIGger:SWEP:SOURce` is set to `TTLTrg<n>` and you want to set `ARM:SWEP:SOURce` to `TTLTrg<n>`, both must be set to the same trigger line `<n>`.

- **Executable when Initiated**: Query form only

- **Coupling Group**: Frequency

- **Related Commands**: `[SOURce:]SWEP:TIME`

- ***RST Condition**: `TRIGger:SWEP:SOURce TIMer`

### Example

**Setting the Sweep Advance Trigger Source**

`TRIG:SWE:SOUR TTL1` selects VXIbus trigger line `TTLTRG1*` as sweep advance source.
TRIGger

:SWEep:TIMer

TRIGger:SWEep:TIMer <period> selects the time between frequency values for sweep or frequency list generation when TRIGger:SWEep:SOURce is set to TIMer. This value set by command is coupled to the [SOURce:]SWEep:TIME command value by the following equation:

\[
\text{TIME} = \text{TIMer} \times (\text{points} - 1)
\]

where points is the [SOURce:]SWEep:POINts value for frequency sweeps, or the length of the frequency list for frequency list generation.

When changing the frequency list length when [SOURce:]FREQuency[1]:MODE LIST is set, or the [SOURce:]SWEep:POINts value when any other MODE is set, the TIME or TIMer value remains the same, depending on which command, [SOURce:]SWEep:TIME or TRIGger:SWEep:TIMer respectively, was most recently sent. The other value is changed based on the new points value.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;period&gt;</td>
<td>numeric</td>
<td>.00125 through 4.19430375</td>
<td>MINimum</td>
</tr>
</tbody>
</table>

MINimum selects 1.25 mS; MAXimum selects 4.19430375 S. The above values bound the valid range for <period>.

Comments

- Executable when Initiated: Query form only
- Coupling Group: Frequency
- Related Commands: [SOURce:]LIST2:FREQuency, [SOURce:]SWEep:POINts, [SOURce:]SWEep:TIME, TRIGger:SWEep:SOURce
- *RST Condition: SOURce:SWEep:TIME 1 is set; TRIGger:SWEep:TIMer is the dependent value.

Example

Setting the Sweep Advance Period

TRIG:SWE:TIM .1

Sets .1 S per frequency value.
The VINStrument subsystem operates with the [SOURce:]ARBitrary and [SOURce:]PM subsystems to control the virtual instrument features of the HP E1445A. These features include the ability to use the VXIbus Local Bus and normal data transfer bus to download data to the segment and segment sequence memories, directly drive the main output DAC, and provide phase deviations for sine waves.

**Subsystem Syntax**

```
VINStrument
[:CONFigure]
:LBUS
[:MODE] <mode>
:TEST
[:CONFigure <length> [no query]]
:DATA? [query only]
:VME
:RECeive
:ADDRess
:DATA? [query only]
:READy? [query only]
[:MODE] <edge>
:IDENtity?
```

### [:CONFigure]:LBUS[:MODE]

**VINStrument[:CONFigure]:LBUS[:MODE] <mode>** selects the operating mode for the VXIbus Local Bus.

#### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mode&gt;</td>
<td>discrete</td>
<td>CONSume</td>
<td>OFF</td>
</tr>
</tbody>
</table>

#### Comments

- **CONSume**: Local Bus data is used and not passed through. This mode must be selected when downloading segment and segment sequence memory data, directly driving the main output DAC, and providing phase deviations for sine waves.
- **OFF**: The Local Bus interface is disabled. Local Bus data is neither used nor passed through.
- **PIPipeline**: Local Bus data is passed through and not used. Select this mode when data should be transparently passed through the HP E1445A.
With VINStrument[:CONFigure]:LBUS[:MODE]:AUTO ON set, the Local Bus operation mode is automatically set to CONSume when downloading segment or segment sequence data ([SOURce:]ARBitrary:DOWNLOAD LBUS command), directly driving the main output DAC ([SOURce:]ARBitrary:DAC:SOURce LBUS command), or providing phase deviation data ([SOURce:]PM:SOURce LBUS command); the mode is set to OFF when none of these are active.

Executing the VINStrument[:CONFigure]:LBUS[:MODE] command sets VINStrument[:CONFigure]:LBUS[:MODE]:AUTO OFF.

• Executable when Initiated: Yes
• Coupling Group: None
• Related Commands: [SOURce:]ARBitrary:DAC:SOURce, [SOURce:]ARBitrary:DOWNLOAD, [SOURce:]PM:SOURce, VINStrument[:CONFigure]:LBUS[:MODE]:AUTO

• *RST Condition: VINStrument:CONfigure:LBUS:MODE OFF

Example Setting the Local Bus Operation Mode

VINS:CONF:LBUS PIP

Sets pipeline (pass through) mode.

[:CONFigure]:LBUS[:MODE]:AUTO

VINVIC:CONFigure:LBUS[:MODE]:AUTO <mode> indicates whether the VXIbus Local Bus operation mode should be automatically set to CONSume when downloading segment or segment sequence data ([SOURce:]ARBitrary:DOWNLOAD LBUS command), directly driving the main output DAC ([SOURce:]ARBitrary:DAC:SOURce LBUS command), or providing phase deviation data ([SOURce:]PM:SOURce LBUS command), and set to OFF when none of these are active. If AUTO ON is set, the Local Bus operation mode is changed as needed; if OFF is set, the mode must be explicitly set by the VINStrument[:CONFigure]:LBUS[:MODE] command.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mode&gt;</td>
<td>boolean</td>
<td>OFF</td>
<td>0</td>
</tr>
</tbody>
</table>

Comments

• Executable when Initiated: Yes
• Coupling Group: None
• Related Commands: [SOURce:]ARBitrary:DAC:SOURce, [SOURce:]ARBitrary:DOWNLOAD, [SOURce:]PM:SOURce, VINStrument[:CONFigure]:LBUS[:MODE]
- *RST Condition: VINstrument:CONFigure:LBUS:MODE:AUTO ON

Example  Uncoupling Local Bus Operation Mode
VINstrument:CONFigure:LBUS:AUTO OFF  Uncouple operation mode.

[:CONFigure]:TEST:CONFigure

VINstrument[:CONFigure]:TEST:CONFigure <length> configures the HP E1445A for Local Bus testing. The <length> parameter indicates that, during the test, that number of bytes will be sent to the HP E1445A. The data will be placed into unused waveform segment memory. When all data has been sent, use the VINstrument[:CONFigure]:TEST:DATA? query to retrieve what the HP E1445A received.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;length&gt;</td>
<td>numeric</td>
<td>see below</td>
<td>none</td>
</tr>
</tbody>
</table>

The valid range for <length> is 2 through the size of largest available contiguous piece of waveform segment memory in bytes (2 bytes per point). <length> must be an even number.

Minimum and Maximum cannot be used with this command.

Comments
- Executable when Initiated: Query form only
- Coupling Group: None
- Related Commands: VINstrument[:CONFigure]:TEST:DATA?
- *RST Condition: None
- Power-On Condition: Local Bus testing not configured

Example  Testing Local Bus Operation
VINstrument:CONFigure:LBUS:CONF 100  Configures for 100 byte test.

send data
VINInstrument:CONFigure:TEST:DATA?  Reads back test data.
[CONFigure]:TEST:DATA?

VINStrument[CONFigure]:TEST:DATA? returns the received VXIbus Local Bus test data. The data is returned in 16-bit integer format in an IEEE-488.2 definite block.

**Comments**
- **Executable when Initiated:** No
- **Coupling Group:** None
- **Related Commands:** VINStrument[CONFigure]:TEST:CONFigure
- ***RST Condition:** None
- **Power-On Condition:** Local Bus testing not configured

**Example**

Testing Local Bus Operation

```
VINS:CONF:TEST:CONF 100
Configure for 100 byte test
send data
VINS:CONF:TEST:DATA?
Read back test data.
```

[CONFigure]:VME[:MODE]

VINStrument[CONFigure]:VME[:MODE] <mode> selects the operating mode for the VXIbus data transfer bus. The only available mode is CONsume.

There is no need to send this command since there is only one available mode. The command is only included for compatibility with the HP Virtual Instrument/Local Bus System Specification.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mode&gt;</td>
<td>discrete</td>
<td>CONsume</td>
<td>none</td>
</tr>
</tbody>
</table>

**Comments**
- **Executable when Initiated:** Yes
- **Coupling Group:** None
- ***RST Condition:** VINStrument:CONFigure:VME:MODE CONsume

**Example**

Setting the VXIbus Data Transfer Bus Operation Mode

```
VINS:VME CONS
Sets CONsume mode.
```
[CONFigure]:VME:RECeive:ADDRess:DATA?

[CONFigure]:VME:RECeive:ADDRess:DATA? returns two values: A24, offset. A24 indicates that the HP E1445A’s A24 address space should be used for writing waveform segment, segment sequence, DAC, or phase deviation data, and offset is the offset into the A24 address space to be written to. The offset returned depends on which of the above operations is active when the ADDRess:DATA? query is executed. If none are active, Error +1022, “VXI data transfer bus not active” is generated.

Comments

• For segment sequence and phase deviation data, the offset returned is the offset of the first of the two words that must be written.

• Executable when Initiated: Yes

• Coupling Group: None

• Related Commands: [SOURce:]ARBitrary:DAC:SOURce, [SOURce:]ARBitrary:DOWNload, [SOURce:]PM:SOURce

• *RST Condition: The VXI data transfer bus is not active

Example

Querying the A24 Address Space Offset

```
LIST:SEL ABC;DEF 100  Creates waveform segment.
ARB:DOWN VXI,ABC,100  Starts download to segment.
```

[CONFigure]:VME:RECeive:ADDRess:READy?

[CONFigure]:VME:RECeive:ADDRess:READy? returns two values: A24, 112. A24 indicates that the HP E1445A’s A24 address space when writing waveform segment, segment sequence, DAC, or phase deviation data, and 112 is the offset into the A24 address space to be checked.

Actually, this indicated Status Register need never be checked. The HP E1445A will always handshake any data written to it; however, the data will be ignored if none of the above operations are active. Nevertheless, bit 1 of the Status Register indicates whether the HP E1445A is in the initiated state or the idle state: 1 indicates initiated, 0 indicates idle. This may useful when writing DAC and phase deviation data as it can be checked to indicate when these types of data will be ignored.

Comments

• Executable when Initiated: Yes

• Coupling Group: None

• Related Commands: [SOURce:]ARBitrary:DAC:SOURce, [SOURce:]ARBitrary:DOWNload, [SOURce:]PM:SOURce
**VINStrument**

- **RST Condition:** None

**Example** Querying the A24 Address Space Ready Indication Offset

```
VINS:VME:REC:ADDR:READ?  
```

Queries A24 offset for ready indication.

**:IDENTity?**

VINStrument:IDENTity? returns a response consisting of 4 fields, indicating the virtual instrument capability of the HP E1445A:

```
HEWLETT-PACKARD VIRTUAL INSTRUMENT,ANY DTOA,0,A.01.00
```

The first and last fields indicate that the HP E1445A conforms to revision A.01.00 of HP’s Virtual Instrument/Local Bus System Specification. The second field indicates that the HP E1445A is a digital-to-analog converter. The third field is reserved for future use.

**Comments**

- **Executable when Initiated:** Yes
- **Coupling Group:** None
- **RST Condition:** None

**Example** Querying Virtual Instrument Capability

```
VINS:IDENT?
```

Queries capability.
### Table 8-1. HP E1445A SCPI Commands

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRIGger</td>
<td>ABORt</td>
</tr>
</tbody>
</table>
| ARM       | ARM[:STARt][:LAYer[1]];COUNT <number>  
           | ARM[:STARt]:LAYer2;COUNT <number>  
           | ARM[:STARt]:LAYer2[:IMMediate]  
           | ARM[:STARt]:LAYer2:SLOPe <edge>  
           | ARM[:STARt]:LAYer2:SOURce <source>  
           | ARM:SWEep;COUNT <number>  
           | ARM:SWEep[:IMMediate]  
           | ARM:SWEep:LINK <link>  
           | ARM:SWEep:SOURce <source>  |
| CALibration | CALibration:COUNt  
              | CALibration:DATA:AC[1] <block>  
              | CALibration:DATA:AC2 <block>  
              | CALibration:DATA[:DC] <block>  
              | CALibration[:DC]:BEGin  
              | CALibration[:DC]:POINT? <value>  
              | CALibration:SECure:CODE <code>  
              | CALibration:SECure[:STATe]<mode>[,<code>]  
              | CALibration:STATe <state>  
              | CALibration:STATe:AC <state>  
              | CALibration:STATe:DC <state>  |
| INITiate  | INITiate[:IMMediate]  |
| OUTPut[1] | OUTPut[1]:FILTer[:LPASs];FREQuency <frequency>  
           | OUTPut[1]:FILTer[:LPASs][:STATe] <mode>  
           | OUTPut[1]:IMPedance <impedance>  
           | OUTPut[1]:LOAD <load>  
           | OUTPut[1]:LOAD:AUTO <mode>  
<pre><code>       | OUTPut[1][:STATe] &lt;mode&gt;  |
</code></pre>
<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>[SOURce:]ARBitr ary</td>
<td>[SOURce:]ARBitr ary:DAC:FORMat &lt;format&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]ARBitr ary:DAC:SOURce &lt;source&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]ARBitr ary:DOWNload &lt;source&gt;,&lt;dest&gt;,&lt;length&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]ARBitr ary:DOWNload:COMPLETE</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]FREQuency[1]:CENTer &lt;center_freq&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]FREQuency[1]:CW[FIXed] &lt;frequency&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]FREQuency[1]:FSKey &lt;frequency1&gt;,&lt;frequency2&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]FREQuency[1]:FSKey:SOURce &lt;source&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]FREQuency[1]:MODE &lt;mode&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]FREQuency[1]:RANGe &lt;range&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]FREQuency[1]:SPAN &lt;freq_span&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]FREQuency[1]:STARt &lt;start_freq&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]FREQuency[1]:STOP &lt;stop_freq&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]FREQuency2:[CW]:FIXed] &lt;frequency&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]FUNCtion:SHAPEs &lt;shape&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]FUNCtion:USER &lt;name&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:FORMat[:DATA] &lt;format&gt; [,&lt;length&gt;]</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SEGment:ADDRess?</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SEGment:CATalog?</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SEGment:COMBined &lt;combined list&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SEGment:COMBined:POINts?</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SEGment:DEFine &lt;length&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SEGment:DELETE:ALL</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SEGment:DELETE:SELECTed</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SEGment:FREE?</td>
</tr>
</tbody>
</table>
Table 8-1. HP E1445A SCPI Commands (continued)

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>[SOURce:]LIST[1]</td>
<td>[SOURce:]LIST[1][:SEGMent]:MARKer &lt;marker_list&gt;</td>
</tr>
<tr>
<td>(Cont’d)</td>
<td>[SOURce:]LIST[1][:SEGMent]:MARKer:POINts?</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1][:SEGMent]:MARKer:SPOint &lt;point&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1][:SEGMent]:SELect &lt;name&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1][:SEGMent]:VOLTage &lt;voltage_list&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1][:SEGMent]:VOLTage:DAC &lt;voltage_list&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1][:SEGMent]:VOLTage:POINts?</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SSEQ uence:ADDdress?</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SSEQ uence:CATalog?</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SSEQ uence:COMBined &lt;combined_list&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SSEQ uence:COMBined:POINts?</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SSEQ uence:DEFine &lt;length&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SSEQ uence:DELeTe:ALL</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SSEQ uence:DELeTe[:SELected]</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SSEQ uence:DWELl:CO UNt &lt;repetition list&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SSEQ uence:DWELl:CO UNt:POINts?</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SSEQ uence:FREE?</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SSEQ uence:MARKer &lt;marker_list&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SSEQ uence:MARKer:POINts?</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SSEQ uence:MARKer:SPOint &lt;point&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SSEQ uence:SELect &lt;name&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SSEQ uence:SEQUence &lt;segment_list&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST[1]:SSEQ uence:SEQUence:SEGments?</td>
</tr>
<tr>
<td>[SOURce:]LIST2</td>
<td>[SOURce:]LIST2:FORMat[:DATA] &lt;format&gt; [,&lt;length&gt;]</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST2:FREQuency &lt;freq_list&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]LIST2:FREQuency:POINts?</td>
</tr>
</tbody>
</table>
### Table 8-1. HP E1445A SCPI Commands (continued)

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>[SOURce:]MARKer</td>
<td>[SOURce:]MARKer:ECLTrg&lt;n&gt;:FEED &lt;source&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]MARKer:ECLTrg&lt;n&gt;:STATe &lt;mode&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]MARKer:FEED &lt;source&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]MARKer:POLarity &lt;polarity&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]MARKer[:STATe] &lt;mode&gt;</td>
</tr>
<tr>
<td>[SOURce:]PM</td>
<td>[SOURce:]:PM[:DEViation] &lt;phase&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]:PM:SOURce &lt;source&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]:PM:STATe &lt;mode&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]:PM:UNIT[:ANGLe] &lt;units&gt;</td>
</tr>
<tr>
<td>[SOURce:]RAMP</td>
<td>[SOURce:]:RAMP:POLarity &lt;polarity&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]:RAMP:POINTS &lt;number&gt;</td>
</tr>
<tr>
<td>[SOURce:]ROSCillator</td>
<td>[SOURce:]ROSCillator:FREQuency:EXTernal &lt;frequency&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]ROSCillator:SOURce &lt;source&gt;</td>
</tr>
<tr>
<td>[SOURce:]SWEep</td>
<td>[SOURce:]SWEep:COUNt &lt;number&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]SWEep:DIRection &lt;direction&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]SWEep:POINts &lt;number&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]SWEep:SPACing &lt;mode&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]SWEep:TIME &lt;number&gt;</td>
</tr>
<tr>
<td>[SOURce:]VOLTage</td>
<td>[SOURce:]VOLTage[:LEVEL][:IMMediate][:AMPLitude] &lt;amplitude&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]VOLTage[:LEVEL][:IMMediate][:AMPLitude]:UNIT[:VOLTage] &lt;units&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:]VOLTage[:LEVEL][:IMMediate]:OFFSet &lt;offset&gt;</td>
</tr>
<tr>
<td>STATus</td>
<td>STATus:OPC:INITiate &lt;state&gt;</td>
</tr>
<tr>
<td></td>
<td>STATus:OPERation</td>
</tr>
<tr>
<td></td>
<td>STATus:OPERation</td>
</tr>
<tr>
<td></td>
<td>STATus:OPERation</td>
</tr>
<tr>
<td></td>
<td>STATus:OPERation</td>
</tr>
<tr>
<td></td>
<td>STATus:OPERation</td>
</tr>
<tr>
<td></td>
<td>STATus:PRESet</td>
</tr>
<tr>
<td>Subsystem</td>
<td>Commands</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
</tr>
</tbody>
</table>
| SYSTEM    | SYSTem:ERRor?  
|          | SYSTem:VERsion? |
| TRIGger   | TRIGger[:STARt]:GATE:POLarity <polarity>  
|          | TRIGger[:STARt]:GATE:SOURce <source>  
|          | TRIGger[:STARt]:GATE:STATe <mode>  
|          | TRIGger[:STARt]:IMMediate  
|          | TRIGger[:STARt]:SLOPe <edge>  
|          | TRIGger[:STARt]:SOURce <source>  
|          | TRIGger:STOP[:IMMediate]  
|          | TRIGger:STOP:SLOPe <edge>  
|          | TRIGger:STOP:SOURce <source>  
|          | TRIGger:SWEep[:IMMediate]  
|          | TRIGger:SWEep:LINK <link>  
|          | TRIGger:SWEep:SOURce <source>  
|          | TRIGger:SWEep:TIMer <period> |
| VINStrument | VINStrument[:CONFigure]:LBUS[:MODE] <mode>  
| VINStrument | VINStrument[:CONFigure]:LBUS[:MODE]:AUTO <mode>  
| VINStrument | VINStrument[:CONFigure]:TEST:CONFigure <length>  
| VINStrument | VINStrument[:CONFigure]:TEST:DATA?  
| VINStrument | VINStrument[:CONFigure]:VME[:MODE] <mode>  
| VINStrument | VINStrument[:CONFigure]:VME:RECeive:ADDRess:DATA?  
| VINStrument | VINStrument[:CONFigure]:VME:RECeive:ADDRess:READY?  
| VINStrument | VINStrument:IDENtity? |
The HP E1445A Arbitrary Function Generator conforms to the SCPI-1991.0 standard.

Table 8-2 and 8-3 list all the SCPI confirmed, approved, and non-SCPI commands that the HP E1445A can execute.

### Table 8-2. SCPI Confirmed Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORt</td>
<td>:START [:SEQUence[1]] [[:LAYer[1]]]</td>
</tr>
<tr>
<td>ARM</td>
<td>:COUNT &lt;number&gt;</td>
</tr>
<tr>
<td></td>
<td>:LAYer2 [[:COUNT &lt;number&gt; [:IMMEDIATE]]]</td>
</tr>
<tr>
<td></td>
<td>:SLOPe &lt;edge&gt; :SOURce&lt;source&gt;</td>
</tr>
<tr>
<td>:SWEep [:SEQUence3]</td>
<td>:COUNT &lt;number&gt;</td>
</tr>
<tr>
<td></td>
<td>[:IMMEDIATE] :LINK &lt;link&gt; :SOURce&lt;source&gt;</td>
</tr>
<tr>
<td>INITiate</td>
<td>[:IMMEDIATE]</td>
</tr>
<tr>
<td>OUTPut[1]</td>
<td>:FILTER [:LPAS]</td>
</tr>
<tr>
<td></td>
<td>[:FREQuency &lt;frequency&gt; [:STATE] &lt;mode&gt;]</td>
</tr>
<tr>
<td></td>
<td>[:IMPedance &lt;impedance&gt; [:STATE] &lt;mode&gt;]</td>
</tr>
<tr>
<td>[SOURce:]</td>
<td>FREQuency[1]</td>
</tr>
<tr>
<td></td>
<td>:CENTer &lt;center_freq&gt; [:CW:FIXed] &lt;frequency&gt;</td>
</tr>
<tr>
<td></td>
<td>[:MODE &lt;mode&gt;] :SPAN &lt;freq_span&gt; :START &lt;start_freq&gt; :STOP &lt;stop_freq&gt;</td>
</tr>
<tr>
<td>FREQuency2</td>
<td>[:CW:FIXed] &lt;frequency&gt;</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>[:SHAPe] &lt;shape&gt;</td>
</tr>
<tr>
<td>LIST2</td>
<td>:FREQuency &lt;freq_list&gt; [:POINts?]</td>
</tr>
<tr>
<td>[SOURce:]</td>
<td>MARKer</td>
</tr>
<tr>
<td></td>
<td>[:STATe] &lt;mode&gt;</td>
</tr>
<tr>
<td>PM</td>
<td>[:DEViation] &lt;phase&gt; :SOURce &lt;source&gt; :STATe &lt;mode&gt; :UNIT [:ANGLe] &lt;units&gt;</td>
</tr>
<tr>
<td>:ROSCillator</td>
<td>:SOURce &lt;source&gt;</td>
</tr>
<tr>
<td>:SWEep</td>
<td>:COUNT &lt;number&gt; :DIRection &lt;direction&gt; :POINts &lt;number&gt; :SPACing &lt;mode&gt; :TIME &lt;time&gt;</td>
</tr>
<tr>
<td>SYSTem</td>
<td>:ERRor? :VERSion?</td>
</tr>
<tr>
<td>TRIGger</td>
<td>[:START [:SEQUence[1]] [:IMMEDIATE]</td>
</tr>
<tr>
<td></td>
<td>:COUNT &lt;number&gt; :SLOPe &lt;edge&gt; :SOURce &lt;source&gt;</td>
</tr>
<tr>
<td></td>
<td>:STOP [:IMMEDIATE] :SLOPe &lt;edge&gt; :SOURce &lt;source&gt;</td>
</tr>
<tr>
<td></td>
<td>[:IMMEDIATE] :LINK &lt;link&gt; :SOURce &lt;source&gt; :TIMEr &lt;period&gt;</td>
</tr>
</tbody>
</table>
### Table 8-3. Non-SCPI Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPut[1]</td>
<td>:LOAD &lt;load&gt; :AUTO &lt;mode&gt;</td>
</tr>
<tr>
<td></td>
<td>[SOURce:] ARBitrary :DAC :FORMat &lt;format&gt; :SOURce &lt;source&gt; :DOWNload &lt;source&gt;,&lt;dest&gt;,&lt;length&gt; :COMPLETE</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>USER :COMPLETE</td>
</tr>
<tr>
<td>LIST[1]</td>
<td>:FORMat [DATA] &lt;format&gt;[,&lt;length&gt;]</td>
</tr>
</tbody>
</table>
IEEE-488.2 Common Commands

*CLS

*CLS clears the Standard Event Status Register, the Operation Status Register, the Questionable Signal Register, and the error queue. This clears the corresponding summary bits (3, 5, and 7) in the Status Byte Register. *CLS does not affect the enable masks of any of the Status Registers.

Comments
• Executable when Initiated: Yes
• Coupling Group: None
• Related Commands: STATus:PRESet
• *RST Condition: None

*DMC

*DMC <name>,<data> creates a macro with the specified name and assigns zero, one, or a sequence of commands to the name. The sequence may be composed of SCPI and/or Common Commands. The sequence may be sent in IEEE-488.2 definite or indefinite block format, or as a quoted string.

Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;name&gt;</td>
<td>string data</td>
<td>1 through 12 characters</td>
<td>none</td>
</tr>
<tr>
<td>&lt;data&gt;</td>
<td>block data or string</td>
<td>any valid command sequence</td>
<td>none</td>
</tr>
</tbody>
</table>

Comments
• Legal macro names must start with an alphabetic character and contain only alphabetic, numeric, and underscore ("_") characters. Alphabetic character case (upper vs. lower) is ignored.

The name is allowed to be the same as a SCPI command, but may be not be the same as a Common Command. When the name is the same as a SCPI command, the macro rather than the command will be executed when the name is received if macro usage is enabled. The SCPI command will be executed if macro usage is disabled.

• Executable when Initiated: Yes
• Coupling Group: None
• Related Commands: *EMC, *GMC, *LMC, *RMC
• *RST Condition: None; macro definitions are unaffected
**Power-On Condition:** No macros are defined

**Example**  Define Macro to Restart Waveform

*DMC "RESTART",#19ABOR;INIT  

*EMC and *EMC?

*EMC <enable> enables and disables macro usage. When <enable> is zero, macro usage is disabled. Any non-zero value in the range of -32768 to +32767 enables macro usage.

*EMC? returns 1 if macro usage is enabled, 0 if disabled.

**Comments**  
- Macro definitions are not affected by this command.
- Executable when Initiated: Yes
- Coupling Group: None
- *RST Condition: Macro usage is disabled
- Power-On Condition: Macro usage is enabled

**ESE and *ESE?**

*ESE <mask> enables one or more event bits of the Standard Event Status Register to be reported in bit 5 (the Standard Event Status Summary Bit) of the Status Byte Register. The <mask> is the sum of the decimal weights of the bits to be enabled.

*ESE? returns the current enable mask.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mask&gt;</td>
<td>numeric</td>
<td>0 through 255</td>
<td>none</td>
</tr>
</tbody>
</table>

A 1 in a bit position enables the corresponding event; a 0 disables it.

**Comments**  
- Executable when Initiated: Yes
- Coupling Group: None
- Related Commands: *ESR?, *SRE, *STB?
- *RST Condition: Unaffected
• **Power-On Condition:** No events are enabled

**Example**  
**Enable All Error Events**

*ESE 60  
*ESR?  

*ESR? returns the value of the Standard Event Status Register. The register is then cleared (all bits 0).

**Comments**  
- **Executable when Initiated:** Yes  
- **Coupling Group:** None  
- ***RST Condition:** None  
- **Power-On Condition:** Register is cleared

**GMC?**

*GMC? *name* returns the definition of the specified macro in IEEE-488.2 definite block format.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>name</em></td>
<td>string data</td>
<td>defined macro name</td>
<td>none</td>
</tr>
</tbody>
</table>

**Comments**  
- **Executable when Initiated:** Yes  
- **Coupling Group:** None  
- **Related Commands:** *DMC  
- ***RST Condition:** None  
- **Power-On Condition:** No macros are defined

**Example**  
**Query Macro Definition**

*GMC? "RESTART"  

Queries macro definition.
**IDN?**

*IDN?* returns identification information for the HP E1445A. The response consists of four fields:

```
HEWLETT-PACKARD,E1445A,0,A.01.00
```

The first two fields identify this instrument as model number E1445A manufactured by Hewlett-Packard. The third field is 0 since the serial number of the HP E1445A is unknown to the firmware. The last field indicates the revision level of the firmware.

---

**Note**

The firmware revision field will change whenever the firmware is revised. A.01.00 is the initial revision. The first two digits indicate the major revision number, and increment when functional changes are made. The last two digits indicate bug fix level.

---

**Comments**

- **Executable when Initiated:** Yes
- **Coupling Group:** None
- ***RST Condition:** None
- **Power-On Condition:** Register is cleared

**LMC?**

*LMC?* returns a comma-separated list of quoted strings, each containing the name of a macro. If no macros are defined, a single null string ("") is returned.

**Comments**

- **Executable when Initiated:** Yes
- **Coupling Group:** None
- **Related Commands:** *DMC
- ***RST Condition:** None
- **Power-On Condition:** No macros are defined
*LRN?  

*LRN? returns a sequence of commands that may be resent to the HP E1445A to return it to its current programming state.

Only those commands that are affected by *RST are included in the sequence. Notable exceptions include the DAC code format (signed vs. unsigned), the [SOURce:]LIST commands, including waveform segment, segment sequence, and frequency list definitions, the STATUS subsystem commands, and the CALibration:SECure command state.

Note  

*LRN? should be sent singly in a program message, since the number of commands in the returned sequence is large, and may vary depending on firmware revision.

Comments  

• Executable when Initiated: Yes
• Coupling Group: None
• Related Commands: *RCL, *RST, *SAV
• *RST Condition: None

*OPC  

*OPC causes the HP E1445A to wait for all pending operations to complete. The Operation Complete bit (bit 0) in the Standard Event Status Register is then set.

If STATus:OPC:INITiate OFF is set, the Operation Complete bit will be set when all commands received prior to the *OPC have been executed. If ON is set, *OPC waits for waveform generation to complete before setting the Operation Complete bit. No other commands will be executed until the Operation Complete bit is set.

Comments  

• Executable when Initiated: Yes
• Coupling Group: None
• Related Commands: *OPC?, *WAI
• *RST Condition: None
*OPC?

*OPC? causes the HP E1445A to wait for all pending operations to complete. A single ASCII “1” is then placed in the output queue.

If STATus:OPC:INITiate OFF is set, the ASCII “1” will be placed in the output queue when all commands received prior to the *OPC? have been executed. If ON is set, *OPC? waits for waveform generation to complete before placing the “1” in the output queue. No other commands will be executed until the “1” is placed in the output queue.

Comments
• Executable when Initiated: Yes
• Coupling Group: None
• Related Commands: *OPC, *WAI
• *RST Condition: None

*PMC

*PMC purges all macro definitions.

Comments
• Use the *RMC command to purge an single macro definition.
• Executable when Initiated: Yes
• Coupling Group: None
• Related Commands: *DMC, *RMC
• *RST Condition: None
*PUD and *PUD?

*PUD <data> stores the specified data in the HP E1445A’s non-volatile calibration memory. The data must be sent in IEEE-488.2 definite or indefinite block format. Calibration security must have been previously disabled.

*PUD? returns the current protected user data in IEEE-488.2 definite block format. The query form may be executed regardless of the state of calibration security.

---

**Note**

When shipped from the factory, the protected user data area contains information regarding when the HP E1445A was last calibrated.

---

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mask&gt;</td>
<td>block data or string</td>
<td>0 through 63 characters</td>
<td>none</td>
</tr>
</tbody>
</table>

**Comments**

- **Executable when Initiated**: Yes
- **Coupling Group**: None
- ***RST Condition**: Unaffected
- **Power-On Condition**: Unaffected

**Example** Setting the Protected User Data

*PUD #17Unit #5  
Sets data to “Unit #5”.

---

422  Command Reference  Chapter 8
**RCL**

*RCL* <number> restores a previously stored programming state from one of the 10 possible stored state areas. The <number> indicates which of the stored state areas should be used.

This command affects the same command settings as does *RST*. Notable exceptions include the DAC code format (signed vs. unsigned), the [SOURce:]LIST commands, including waveform segment, segment sequence, and frequency list definitions, the STATus subsystem commands, and the CALibration:SECure command state.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>numeric</td>
<td>0 through 9</td>
<td>none</td>
</tr>
</tbody>
</table>

**Comments**

- **Executable when Initiated:** No
- **Coupling Group:** None
- **Related Commands:** *LRN?, *RST, *SAV
- ***RST Condition:** all saved states set to the same state as the *RST state

**RMC**

*RMC* <name> purges only the specified macro definition.

NOTE: At printing time, *RMC is a command proposed and accepted for a revision and re-designation of IEEE-488.2.

**Comments**

- **Use the *PMC command to purge all macro definitions in one command.**
- **Executable when Initiated:** Yes
- **Coupling Group:** None
- **Related Commands:** *DMC, *PMC
- ***RST Condition:** None
**RST**

*RST* resets the HP E1445A as follows:

- Sets all commands to their *RST* state.
- Aborts all pending operations including waveform generation.

*RST* does not affect:

- The state of VXIbus word serial protocol
- The output queue
- The Service Request Enable Register
- The Standard Event Status Enable Register
- The enable masks for the Operation Status and Questionable Signal Registers
- Calibration data
- Calibration security state
- Protected user data
- The DAC code format (signed vs. unsigned)
- Waveform segment, segment sequence, and frequency list definitions

**Comments**

- **Executable when Initiated:** Yes
- **Coupling Group:** None
- **RST Condition:** None

**SAV**

*SAV <number>* stores the current programming state into one of the 10 possible stored state areas. The <number> indicates which of the stored state areas should be used.

This command stores the states of all commands affected by *RST*. Notable exceptions include the DAC code format (signed vs. unsigned), the [SOURce:]LIST commands, including waveform segment, segment sequence, and frequency list definitions, the STATus subsystem commands, and the CALibration:SECure command state.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>numeric</td>
<td>0 through 9</td>
<td>none</td>
</tr>
</tbody>
</table>

**Comments**

- **Executable when Initiated:** Yes
- **Coupling Group:** None
- **Related Commands:** *LRN?*, *RCL*, *RST*

- **RST Condition:** unaffected

- **Power-on Condition:** all saved states set to the same state as the *RST* state

## *SRE and *SRE?*

*SRE* <mask> specifies which bits of the Status Byte Register are enabled to generate a service request (VXIbus reqt signal). Event and summary bits are always set and cleared in the Status Byte Register regardless of the enable mask. The <mask> is the sum of the decimal weights of the bits to be enabled.

*SRE?* returns the current enable mask.

### Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Range of Values</th>
<th>Default Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mask&gt;</td>
<td>numeric</td>
<td>0 through 255</td>
<td>none</td>
</tr>
</tbody>
</table>

A 1 in a bit position enables service request generation when the corresponding Status Byte Register bit is set; a 0 disables it.

### Comments

- **Executable when Initiated:** Yes
- **Coupling Group:** None
- **RST Condition:** Unaffected
- **Power-On Condition:** No bits are enabled

### Example

Enable Service Request on Message Available Bit

*SRE 16*

* Enables request on MAV.*

---

Chapter 8 Command Reference 425
### *STB?*

*STB? returns the value of the Status Byte Register. Bit 6 (decimal weight 64) is set if a service request is pending. STB? should not be used to read the Status Byte Register if a service request is generated by a message available (MAV) condition.

**Comments**
- *STB? is a query. Thus, sending the command in response to a MAV condition will generate Error -410 "Query interrupted".
- **Executable when Initiated:** Yes
- **Coupling Group:** None
- **Related Commands:** *SRE
- **RST Condition:** None

### *TRG*

*TRG* is the command equivalent of the HP-IB Group Execute Trigger and the VXIbus Trigger word serial command and has exactly the same effect.

**Comments**
- **Executable when Initiated:** Yes
- **Coupling Group:** None
- **Related Commands:** ARM and TRIGger subsystem, [SOURce:] commands
- **RST Condition:** None

### *TST?*

*TST?* causes the HP E1445A to execute its internal self-test and return a value indicating the results of the test.

A zero (0) response indicates that the self-test passed. A one (1) response indicates that the test failed. The failure also generates an error message with additional information on why the test failed.

When the test completes, all waveform segment and segment sequence definitions are deleted, and all other commands are set to their *RST values.

**Comments**
- **Executable when Initiated:** No
- **Coupling Group:** None
- **RST Condition:** None
*WAI

*WAI causes the HP E1445A to wait for all pending operations to complete before executing any further commands.

If STATus:OPC:INITiate OFF is set, command execution resumes when all commands received prior to the *WAI have been executed. If ON is set, *WAI waits for waveform generation to complete before resuming command execution.

Comments

- Executable when Initiated: Yes
- Coupling Group: None
- Related Commands: *OPC, *OPC?
- *RST Condition: None
This section describes the IEEE-488.2 Common Commands implemented in the HP E1445A. The table below shows the commands listed by functional group; however, commands are listed alphabetically in the reference. Examples are shown in the reference when the command has parameters or returns a non-trivial response; otherwise, the command string is as shown in the table. For additional information, refer to IEEE Standard 488.2-1987.

### Table 8-4. HP E1445A Common Commands

<table>
<thead>
<tr>
<th>Category</th>
<th>Command</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Data</td>
<td>*IDN?</td>
<td>Identification Query</td>
</tr>
<tr>
<td></td>
<td>*PUD &lt;data&gt;</td>
<td>Protected User Data Command</td>
</tr>
<tr>
<td></td>
<td>*PUD?</td>
<td>Protected User Data Query</td>
</tr>
<tr>
<td>Internal Operations</td>
<td>*LRN?</td>
<td>Learn Device Setup Query</td>
</tr>
<tr>
<td></td>
<td>*RST</td>
<td>Reset Command</td>
</tr>
<tr>
<td></td>
<td>*TST?</td>
<td>Self Test Query</td>
</tr>
<tr>
<td>Synchronization</td>
<td>*OPC</td>
<td>Operation Complete Command</td>
</tr>
<tr>
<td></td>
<td>*OPC?</td>
<td>Operation Complete Command</td>
</tr>
<tr>
<td></td>
<td>*WAI</td>
<td>Wait-to-Continue Command</td>
</tr>
<tr>
<td>Macro</td>
<td>*DMC &lt;name&gt;,&lt;data&gt;</td>
<td>Define Macro Command</td>
</tr>
<tr>
<td></td>
<td>*EMC &lt;enable&gt;</td>
<td>Enable Macro Command</td>
</tr>
<tr>
<td></td>
<td>*EMC?</td>
<td>Enable Macro Query</td>
</tr>
<tr>
<td></td>
<td>*GMC? &lt;name&gt;</td>
<td>Get Macro Contents Query</td>
</tr>
<tr>
<td></td>
<td>*LMC?</td>
<td>Learn Macro Query</td>
</tr>
<tr>
<td></td>
<td>*PMC</td>
<td>Purge Macros Command</td>
</tr>
<tr>
<td></td>
<td>*RMC &lt;name&gt;</td>
<td>Remove Individual Macro Command</td>
</tr>
<tr>
<td>Status and Event</td>
<td>*CLS</td>
<td>Clear Status Command</td>
</tr>
<tr>
<td></td>
<td>*ESE &lt;mask&gt;</td>
<td>Standard Event Status Enable Command</td>
</tr>
<tr>
<td></td>
<td>*ESE?</td>
<td>Standard Event Status Enable Query</td>
</tr>
<tr>
<td></td>
<td>*ESR?</td>
<td>Standard Event Status Register Query</td>
</tr>
<tr>
<td></td>
<td>*SRE</td>
<td>Service Request Enable Command</td>
</tr>
<tr>
<td></td>
<td>*SRE?</td>
<td>Service Request Enable Query</td>
</tr>
<tr>
<td></td>
<td>*STB?</td>
<td>Read Status Byte Query</td>
</tr>
<tr>
<td>Trigger</td>
<td>*TRG</td>
<td>Trigger Command</td>
</tr>
<tr>
<td>Stored Settings</td>
<td>*RCL</td>
<td>Recall Command</td>
</tr>
<tr>
<td></td>
<td>*SAV</td>
<td>Save Command</td>
</tr>
</tbody>
</table>
Introduction

This chapter describes the HP E1445A Arbitrary Function Generator status system. Included is information on the status groups used by the AFG, the conditions monitored by each group, and information on how to enable a condition to interrupt the computer.

This main sections of this chapter include:

- Status System Registers ........................................ Page 429
  - The Questionable Signal Status Group .................. Page 431
  - The Operation Status Group .............................. Page 435
  - The Standard Event Status Group ....................... Page 439
  - The Status Byte Status Group ......................... Page 442

Status System Registers

Operating conditions within the AFG are monitored by registers in various status groups. The status groups implemented by the AFG are:

- Questionable Signal Status Group
  - Condition Register
  - Transition Filter
  - Event Register
  - Enable Register

- Operation Status Group
  - Condition Register
  - Transition Filter
  - Event Register
  - Enable Register

- Standard Event Status Group
  - Standard Event Status Register
  - Standard Event Status Enable Register

- Status Byte Status Group
  - Status Byte Register
  - Service Request Enable Register

The relationship between the registers and filters in these groups is shown in Figure 9-1.
Figure 9-1. HP E1445A Status Groups and Associated Registers
The Questionable Signal Status Group

The Questionable Signal Status Group monitors the quality of various aspects of the output signal. In the AFG, the Questionable Signal Status Group monitors the frequency accuracy of the divide-by-n subsystem, and also error conditions in non-volatile calibration memory.

The Condition Register

Divide-by-n frequency accuracy and non-volatile calibration memory errors are monitored with the following bits in the Condition Register. All other bits are unused.

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>unused</td>
<td>CAL</td>
<td>unused</td>
<td>FREQ</td>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **FREQuency**: Bit 5 is set (1) when the frequency generated by the divide-by-n ([:SOURce:]FREQuency2) subsystem differs from the programmed frequency by greater than 1%. Otherwise, the bit remains cleared (0).
- **CALibration**: Bit 8 is set (1) when an error is detected in non-volatile calibration memory.

Reading the Condition Register

The settings of bits 5 and 8 can be determined by reading the Condition Register with the command:

```
STATus:QUEStionable:CONDition?
```

Bit 5 has a corresponding decimal value of 32 and bit 8 has a decimal value of 256. Reading the Condition Register does not affect the bit settings. The bits are cleared following a reset (*RST). Bit 8 calibration will remain set, however, if the error condition persists.

The Transition Filter

The Transition Filter specifies which type of bit transition in the Condition Register will set corresponding bits in the Event Register. Transition filter bits may be set for positive transitions (0 to 1), or negative transitions (1 to 0).

The commands used to set the transitions are:

```
STATus:QUEStionable:NTRansition <unmask>
STATus:QUEStionable:PTRansition <unmask>
```

NTRansition sets the negative transition. For each bit unmasked, a 1 to 0 transition of that bit in the Condition Register sets the associated bit in the Event Register.

PTRansition sets the positive transition. For each bit unmasked, a 0 to 1 transition of that bit in the Condition Register sets the associated bit in the Event Register.
<unmask> is the decimal, hexadecimal (#H), octal (#Q), or binary (#B) value of the Condition Register bit to be unmasked. (The decimal values of bits 5 and 8 are 32 and 256.)

**The Event Register**

The Event Register latches transition events from the Condition Register as specified by the Transition Filter. Bits in the Event Register are latched and remain set until the register is cleared by one of the following commands:

```
STATus:QUEStionable[:EVENt]? *CLS
```

**The Enable Register**

The Enable Register specifies which bits in the Event Register can generate a summary bit which is subsequently used to generate a service request. The AFG logically ANDs the bits in the Event Register with bits in the Enable Register, and ORs the results to obtain a summary bit.

The bits in the Enable Register that are to be ANDed with bits in the Event Register are specified (unmasked) with the command:

```
STATus:QUEStionable:ENABLE <unmask>
```

<unmask> is the decimal, hexadecimal (#H), octal (#Q), or binary (#B) value of the Enable Register bit to be unmasked. (The decimal values of bits 5 and 8 are 32 and 256.)

The Enable Register is cleared at power-on, or by specifying an <unmask> value of 0.

**Program Example**

The QSSG_RQS program sets up the Questionable Signal Status Group Registers to monitor the output frequency generated by the [SOURcer:FREQuency2 subsystem. If the programmed frequency differs from the actual output frequency by greater than 1%, a service request interrupt is sent to the computer which responds with a message indicating the condition.

The steps of the program are:

1. Set the bit transition which will latch the event (frequency error) in the Event Register.

   ```
   STATus:QUEStionable:NTRansition <unmask>
   or
   STATus:QUEStionable:PTRansition <unmask>
   ```
2. Unmask bit 4 (FREQ) in the Enable Register so that the event latched into the Event Register will generate a Questionable Signal Status Group summary bit.

```
STATus:QUEStionable:ENABle <unmask>
```

3. Unmask bit 3 (QUE) in the Service Request Enable Register so that a service request is generated when the Questionable Signal Status Group summary bit is received.

```
*SRE <unmask>
```

**HP BASIC Program Example (QSSG_RQS)**

```hpbasic
1 !RE-STORE "QSSG_RQS"
2 !This program generates a service request when the output frequency
3 !generated by the SOURce:FREQuency2 subsystem differs from the
4 !programmed frequency by more than 1%.
5 !
10 !Assign an I/O path between the computer and the E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !
50 !Reset the AFG
60 CALL Rst
70 !
80 !Set up the computer to respond to the service request.
90 ON INTR 7 CALL Disp_msg
100 ENABLE INTR 7;2
110 !
120 !Set up the AFG to monitor the output frequency.
130 OUTPUT @Afg;"*CLS" !clear Status Byte and Event Registers
140 OUTPUT @Afg;"STAT:QUES:PTR 32" !pos transition of FREQ bit
150 OUTPUT @Afg;"STAT:QUES:ENAB 32" !allow FREQ bit to generate summary bit
160 OUTPUT @Afg;"*SRE 8" !enable summary bit to generate RQS
170 !
180 !Call subprogram which outputs a signal using the SOURce:FREQ2
190 !subsystem.
200 CALL Freq2
210 !WAIT .1 !allow interrupts to be serviced
220 OFF INTR 7
230 END
240 !
250 SUB Freq2
260 Freq2: !Subprogram which outputs a 10 MHz square wave using the
270 !SOURce:FREQ2 subsystem.
280 COM @Afg
290 OUTPUT @Afg;"SOUR:ROSC:SOUR INT2;"; !reference oscillator
300 OUTPUT @Afg;"TRIG:STAR:SOUR INT2;"; !frequency generator
Continued on Next Page
```
OUTPUT @Afg;".*SOUR:FREQ2 10E6;" !frequency
OUTPUT @Afg;".*SOUR:FUNC:SHAP SQU;" !function
OUTPUT @Afg;".*SOUR:VOLT:LEV:IMM:AMPL 1V" !amplitude
OUTPUT @Afg;"INIT:IMM" !wait_for_arm state
LOOP !loop continuously until frequency error occurs
END LOOP
SUBEND
 !
SUB Disp_msg
Disp_msg: !Subprogram which is called when output frequency
!varies from 10 MHz by more than 1%.
COM @Afg
!Read Status Byte Register and clear service request bit (RQS)
B=SPOLL(@Afg)
LOOP
DISP "Output frequency error"
WAIT 1
DISP ""
WAIT 1
END LOOP
SUBEND
 !
SUB Rst
Rst: !Subprogram which resets the E1445.
COM @Afg
OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
ENTER @Afg;Complete
SUBEND

Comments
• This program runs continuously until a frequency change greater than 1% occurs between the programmed frequency and the output frequency. Resetting the computer stops the program.
• Clearing the Questionable Signal Event Register (line 130) allows new events to be latched into the Register. Clearing the service request bit (bit 6 (RQS)) in the Status Byte Register (line 440) when the interrupt is serviced allows the bit to be set again when the next summary bit is received.

Visual BASIC and Visual C/C++ Program Versions
The Visual BASIC example program, QSSG_RQS.FRM, is in directory “VBPROG” and the Visual C example program, QSSG_RQS.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
The Operation Status Group

The Operation Status Group monitors current operating conditions within the AFG. The specific conditions include: calibrating, sweeping, entering the wait-for-arm state, and execution of the INITiate:IMMediate command.

The Condition Register

Calibration, sweeping, waiting for an arm signal, and the INITiate:IMMediate command are monitored with the following bits in the Condition Register. All other bits are unused.

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>unused</td>
<td>INIT</td>
<td>ARM</td>
<td>unused</td>
<td>SWE</td>
<td>unused</td>
<td>CAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **CALibrating**: Bit 0 is set (1) during calibration. The bit is cleared (0) otherwise.
- **SWEeping**: Bit 3 is set (1) while a frequency sweep or list is in progress. The bit is cleared (0) when waveform generation is halted, when frequency sweeping or lists are not selected, and at the end of each sweep or list.
- **Waiting for ARM**: Bit 6 is set (1) when the AFG enters the wait-for-arm state. The bit is cleared (0) when a start arm is received or when waveform generation is aborted.
- **INITiated**: Bit 8 is set (1) when the INITiate:IMMediate command is executed. The bit is cleared (0) when waveform generation is complete and the AFG returns to the Idle state.

Reading the Condition Register

Bit settings in the Condition Register can be determined with the command:

```
STATus:OPERation:CONDition?
```

Bits 0, 3, 6, and 8 have corresponding decimal values of 1, 8, 64, and 256. Reading the Condition Register does not affect the bit settings. The bits are cleared following a reset (*RST).

The Transition Filter

The Transition Filter specifies which type of bit transition in the Condition Register will set corresponding bits in the Event Register. Transition filter bits may be set for positive transitions (0 to 1), or negative transitions (1 to 0). The commands used to set the transitions are:

```
STATus:OPERation:NTRansition <unmask>
STATus:OPERation:PTRansition <unmask>
```

NTRansition sets the negative transition. For each bit unmasked, a 1 to 0 transition of that bit in the Condition Register sets the associated bit in the Event Register.
PTRansition sets the positive transition. For each bit unmasked, a 0 to 1 transition of that bit in the Condition Register sets the associated bit in the Event Register.

<unmask> is the decimal, hexadecimal (#H), octal (#Q), or binary (#B) value of the Condition Register bit to be unmasked. (Bits 0, 3, 6, and 8 have corresponding decimal values of 1, 8, 64, and 256.)

**The Event Register**

The Event Register latches transition events from the Condition Register as specified by the Transition Filter. Bits in the Event Register are latched and remain set until the register is cleared by one of the following commands:

```
STATus:OPERation[:EVENt]? *CLS
```

**The Enable Register**

The Enable Register specifies which bits in the Event Register can generate a summary bit which is subsequently used to generate a service request. The AFG logically ANDs the bits in the Event Register with bits in the Enable Register, and ORs the results to obtain a summary bit.

The bits in the Enable Register that are to be ANDed with bits in the Event Register are specified (unmasked) with the command:

```
STATus:OPERation:ENABle <unmask>
```

<unmask> is the decimal, hexadecimal (#H), octal (#Q), or binary (#B) value of the Enable Register bit to be unmasked. (Bits 0, 3, 6, and 8 have corresponding decimal values of 1, 8, 64, and 256.)

The Enable Register is cleared at power-on, or by specifying an <unmask> value of 0.

**Program Example**

The OSG_RQS program sets up the Operation Status Group Registers to determine when the AFG enters the wait-for-arm state. When the AFG enters that state, a service request interrupt is sent to the computer which responds with a message indicating the state which exists.

The steps of the program are:

1. Set the bit transition which will latch the event (entering wait-for-arm state) in the Event Register.

   ```
   STATus:OPERation:NTRansition <unmask>
   or
   STATus:OPERation:PTRansition <unmask>
   ```
2. Unmask bit 6 (ARM) in the Enable Register so that the event latched into the Event Register will generate an Operation Status Group summary bit.

   STATus:OPERation:ENABle <unmask>

3. Unmask bit 7 (OPR) in the Service Request Enable Register so that a service request is generated when the Operation Status Group summary bit is received.

   *SRE <unmask>

**HP BASIC Program Example (OSG_RQS)**

1 !RE-STORE "OSG_RQS"
2 !This program generates a service request when the AFG enters the 
3 !wait-for-arm state.
4 !
10 !Assign an I/O path between the computer and the E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !
50 !Reset the AFG
60 CALL Rst
70 !
80 !Set up the computer to respond to the service request.
90 ON INTR 7 CALL Afg_ready
100 ENABLE INTR 7;2
110 !
120 !Set up the AFG to generate a service request when it enters the 
130 !wait-for-arm state.
140 OUTPUT @Afg;"*CLS" !clear Status Byte and Event Registers
150 OUTPUT @Afg;"STAT:OPER:PTR 64" !pos transition of ARM bit
160 OUTPUT @Afg;"STAT:OPER:ENAB 64" !allow ARM bit to generate summary bit
170 OUTPUT @Afg;"*SRE 128" !enable summary bit to generate RQS
180 OUTPUT @Afg;"STAT:OPC:INIT OFF" !allow intr branching after wait-for-arm
190 !
200 !Call subprogram which sets up and initiates the AFG.
210 !subsystem.
220 CALL Afg_setup
230 WAIT .1 'allow interrupt to be serviced
240 OFF INTR 7
250 END
260 !
270 SUB Afg_setup
280 Afg_setup: !Subprogram which sets up the AFG and places it in the 
290 !wait-for-arm state
300 COM @Afg
310 OUTPUT @Afg;"ABORT" !stop current waveform

*Continued on Next Page*
320 OUTPUT @Afg;"SOUR:ROSC:SOUR INT1;"; !reference oscillator
330 OUTPUT @Afg;"TRIG:STAR:SOUR INT1;"; !frequency generator
340 OUTPUT @Afg;"SOUR:FREQ1:FIX 1E3;"; !frequency
350 OUTPUT @Afg;"SOUR:FUNC:SHAP SIN;"; !function
360 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 1V" !amplitude
370 OUTPUT @Afg;"ARM:STAR:LAY2:SOUR HOLD" !hold off arm signal
380 OUTPUT @Afg;"INIT:IMM;"OPC?" !set wait-for-arm state
390 ENTER @Afg:Ready
400 OUTPUT @Afg;"ARM:STAR:LAY2:IMM" !arm AFG (output signal)
410 SUBEND
420 !
430 SUB Afg_ready
440 Afg_ready: !Subprogram which is called when the AFG enters the
450 !wait-for-arm state.
460 COM @Afg
470 !Read Status Byte Register and clear service request bit (RQS)
480 B=SPOLL(@Afg)
490 DISP "AFG is in the wait-for-arm state, press 'Continue' to send ARM"
500 PAUSE
510 DISP ""
520 SUBEND
530 !
540 SUB Rst
550 Rst: !Subprogram which resets the E1445.
560 COM @Afg
570 OUTPUT @Afg;"RST;"OPC?" !reset the AFG
580 ENTER @Afg:Complete
590 SUBEND

Comments

- Clearing the Operation Status Event Register (line 140) allows new
events to be latched into the register. Clearing the service request bit
(bit 6 (RQS)) in the Status Byte Register (line 480) when the
interrupt is serviced allows the bit to be set again when the next
summary bit is received.

- STAT:OPC:INIT OFF (line 180) allows the "OPC? command (line 380)
to execute following INIT:IMM, rather than waiting for the AFG to
return to the Idle state (Pending Operation Flag set false). Thus, when
the AFG enters the wait-for-arm state following INIT:IMM, "OPC? executes and allows time for the interrupt to be serviced (Afg_ready
called) before line 400 executes.

Refer to page 382 for more information on the STATus:OPC:INITiate
command.

Visual BASIC and
Visual C/C++ Program
Versions

The Visual BASIC example program, OSG_RQS.FRM, is in directory
“VBPROG” and the Visual C example program, OSG_RQS.C, is in
directory “VCProg” on the CD that came with your HP E1445A.
The Standard Event Status Group

The Standard Event Status Group monitors command execution, programming errors, and the power-on state. It is the status group used by the error checking routine in the HP BASIC example programs found throughout the manual.

The Standard Event Status Register

The conditions monitored by the Standard Event Status Register are identified below.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Power-on (PON)</td>
</tr>
<tr>
<td>6</td>
<td>Command Error (CME)</td>
</tr>
<tr>
<td>5</td>
<td>Execution Error (EXE)</td>
</tr>
<tr>
<td>4</td>
<td>Device Dependent Error (DDE)</td>
</tr>
<tr>
<td>3</td>
<td>Query Error (QYE)</td>
</tr>
<tr>
<td>2</td>
<td>Operation Complete (OPC)</td>
</tr>
<tr>
<td>1</td>
<td>unused</td>
</tr>
<tr>
<td>0</td>
<td>unused</td>
</tr>
</tbody>
</table>

- **Power-on (PON):** Bit 7 is set (1) when an off-to-on transition has occurred.
- **Command Error (CME):** Bit 5 is set (1) when an incorrect command header is received, or if an unimplemented common command is received.
- **Execution Error (EXE):** Bit 4 is set (1) when a command parameter is outside its legal range.
- **Device Dependent Error (DDE):** Bit 3 is set (1) when an error other than a command error, execution error, or query error has occurred.
- **Query Error (QYE):** Bit 2 is set (1) when the AFG output queue is read and no data is present, or when data in the output queue has been lost.
- **Operation Complete (OPC):** Bit 0 is set (1) when the *OPC command is received. *OPC is used to indicate when all pending (or previous) AFG commands have completed.

Note that bits 7, 5, 4, 3, 2, and 0 have corresponding decimal values of 128, 32, 16, 8, 4, and 1.

Reading the Standard Event Status Register

The settings of the Standard Event Status Register can be read with the command:

*ESR?

The bits are cleared at power-on, or by *ESR? or *CLS.
The Standard Event Status Enable Register

The Standard Event Status Enable Register specifies which bits in the Standard Event Status Register can generate a summary bit which is subsequently used to generate a service request. The AFG logically ANDs the bits in the Event Register with bits in the Enable Register, and ORs the results to obtain a summary bit.

The bits in the Enable Register that are to be ANDed with bits in the Event Register are specified (unmasked) with the command:

*ESE <unmask>

<unmask> is the decimal, hexadecimal (#H), octal (#Q), or binary (#B) value of the Enable Register bit to be unmasked. (Bits 7, 5, 4, 3, 2, 0 have corresponding decimal values of 128, 32, 16, 8, 4, 1.)

All unmasked bits in the Enable Register can be determined with the command:

*ESE?

The Standard Event Status Enable Register is cleared at power-on, or with an <unmask> value of 0.

Program Example

The ERRORCHK program sets up the Standard Event Status Group Registers to monitor programming errors. When a command error, execution error, device dependent error, or query error occurs, a service request interrupt is sent to the computer which then reads the AFG error queue and displays the error code and message.

The steps of the program are:

1. Unmask bits 5 (CME), 4 (EXE), 3 (DDE), 2 (QYE) in the Standard Event Status Enable Register so that the error will generate a Standard Event Status Group summary bit.

   *ESE <unmask>

2. Unmask bit 5 (ESB) in the Service Request Enable Register so that a service request is generated when the Standard Event Status Group summary bit is received.

   *SRE <unmask>
HP BASIC Program Example (ERRORCHK)

1 !RE-STORE"ERRORCHK"
2 !This program represents the method used to check for programming
3 !errors in HP BASIC programs.
4 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 70910
30 COM @Afg
40 !Define branch to be taken when an E1445A error occurs.
50 !Enable HP-IB interface to generate an interrupt when an error
60 !occurs.
70 ON INTR 7 CALL Errmsg
80 ENABLE INTR 7;2
90 !Clear all bits in the Standard Event Status Register, unmask the
100 !Standard Event Status Group summary bit in the E1445A Status Byte
110 !register (decimal weight 32), unmask the query error, device
120 !dependent error, execution error, and command error bits
130 ! (decimal sum 60) in the E1445A Standard Event Status Register.
140 OUTPUT @Afg;"CLS"
150 OUTPUT @Afg;"SRE 32"
160 OUTPUT @Afg;"ESE 60"
170 !
180 !Subprogram calls would be here
190 !
200 WAIT .1 !allow error branch to occur before turning intr off
210 OFF INTR 7
220 END
230 !
240 SUB Errmsg
250 Errmsg: !Subprogram which displays E1445 programming errors
260 COM @Afg
270 DIM Message$[256]
280 !Read AFG status byte register and clear service request bit
290 B=SPOLL(@Afg)
300 !End of statement if error occurs among coupled commands
310 OUTPUT @Afg;"
320 OUTPUT @Afg;"ABORT" !abort output waveform
330 REPEAT
340 OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
350 ENTER @Afg;Code,Message$
360 PRINT Code,Message$
370 UNTIL Code=0
380 STOP
390 SUBEND

Comments

- Clearing the service request bit (bit 6 (RQS)) in the Status Byte
  Register (line 290) when the interrupt is serviced allows the bit to be
  set again when the next summary bit is received.
Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, ERRORCHK.FRM, is in directory “VBPROG” and the Visual C example program, ERRORCHK.C, is in directory “VCPROG” on the CD that came with your HP E1445A.

The Status Byte Status Group

The registers in the Status Byte Status Group enable conditions monitored by the other status groups to generate a service request.

The Status Byte Register

The Status Byte Register contains the summary bits of the Questionable Signal Status Group (QUES), the Operation Status Group (OPER), and the Standard Event Status Group (ESB). The register also contains the message available bit (MAV) and the service request bit (RQS).

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPER</td>
<td>RQS</td>
<td>ESB</td>
<td>MAV</td>
<td>QUES</td>
<td>unused</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Questionable Signal Summary Bit (QUES):** Bit 3 is set (1) when a condition monitored by the Questionable Signal Status Group is present, when the appropriate bit is latched into the group’s Event Register, and when the bit is unmasked by the group’s Enable Register.

- **Message Available Bit (MAV):** Bit 4 is set (1) when data, such as a query response, is in the AFG’s output queue.

- **Standard Event Summary Bit (ESB):** Bit 5 is set (1) when a condition monitored by the Standard Event Status Group is present and the appropriate bit is set in the group’s Event Register, and when the bit is unmasked by the group’s Enable Register.

- **Service Request Bit (RQS):** Bit 6 is set (1) when any other bit in the Status Byte Register is set.

- **Operation Status Summary Bit (OPER):** Bit 7 is set (1) when a condition monitored by the Operation Status Group is present, when the appropriate bit is latched into the group’s Event Register, and when the bit is unmasked by the group’s Enable Register.

Reading the Status Byte Register

The Status Byte Register can be read with either of the following commands:

*STB?

SPOLL

Both commands return the decimal weighted sum of all set bits in the register. The difference between the commands is that *STB? does not clear bit 6 (RQS service request). The serial poll (SPOLL) does clear bit 6. All bits in the Status Byte Register with the exception of MAV are cleared with the command:

*CLS  (*CLS also aborts the current waveform)

MAV is cleared when data is read from the output queue.
The Service Request Enable Register

The Service Request Enable Register specifies which (status group) summary bit(s) will send a service request message to the computer over HP-IB. The bits are specified (unmasked) with the command:

*SRE <unmask>

All unmasked bits in the Enable Register can be determined with the command:

*SRE?

The Service Request Enable Register is cleared at power-on, or by specifying an <unmask> value of 0.

Presetting the Enable Register and Transition Filter

The Enable Registers and Transition Filters in the Questionable Signal and Operation Status Groups can be preset (initialized) with the command:

STATus:PRESet

All bits in the Enable Registers are masked (i.e. <unmask> is 0), and all bits in the Condition Registers set corresponding bits in the Event Registers on positive (0 to 1) transitions.
Chapter 10
Block Diagram Description

Chapter Contents

This chapter shows how the HP E1445A 13-Bit Arbitrary Function Generator (AFG) operates. The sections are as follows:

- AFG Description ........................................ Page 445
- Arbitrary Waveform Description..................... Page 446
- Generating Non-Sinusoid Arbitrary Waveforms.... Page 447
  - Output DAC............................................. Page 447
  - Memory.................................................. Page 448
  - Reference Oscillator................................. Page 448
  - Frequency Generators............................... Page 448
  - Trigger Circuitry..................................... Page 450
  - Output Circuitry...................................... Page 450
  - Microprocessor...................................... Page 450
- Generating Sinusoid Waveforms...................... Page 450
- Output Circuitry Description.......................... Page 451
  - Attenuator............................................. Page 451
  - Filter.................................................. Page 451
  - Output Amplifier.................................... Page 451
  - Offset Circuitry...................................... Page 451
- AFG Memory Description............................... Page 452

AFG Description

The AFG can output standard waveforms, like sinusoid, square, triangle, and ramp waveforms, and user defined arbitrary (i.e., USER function) waveforms. The AFG can also perform frequency sweeping, frequency-shift-keying, output frequency lists, and DC volts.

All waveforms that the AFG generates, except DC volts, are arbitrary waveforms. The only difference is that the AFG generates the data for the standard waveforms internally and the user supplies the data for the arbitrary waveforms.
Refer to Figure 10-1. An arbitrary waveform is equally divided into points that are the actual voltage points of the waveform. The AFG stores these points as a waveform segment in its segment memory. The waveform segments are stored as Digital-to-Analog Converter (DAC) codes. The codes set the output DAC to the voltage values of the waveform.

![Diagram of arbitrary waveform](image)

**Figure 10-1. Arbitrary Waveform**

The segment sequence selects the waveform segment to be output for waveform generation. The segment sequence is stored in the AFG’s sequence memory.

For square, ramp, and triangle functions, the AFG calculates the waveform segments and segment sequences, and stores them in memory. For the user generated waveforms, the user transfers the waveform segments and segment sequences to the AFG which stores them into memory. (See “Generating Sinusoid Waveforms” on page 450 for sinusoid waveforms.)
Refer to Figure 10-2. The following describes the blocks that generate non-sinusoid waveforms.

**Output DAC**

The AFG uses the 13-bit DAC to generate the waveforms. Each time the AFG’s frequency generator clocks the DAC, the DAC outputs a voltage value that corresponds to the point value in the waveform segment. The bits set in the DAC determine the voltage value. For non-sinusoid functions, the DAC codes in the AFG’s segment memory set the appropriate bits of the DAC. For the sinusoid function, the output of the frequency generator sets the bits to the appropriate value (see “Generating Sinusoid Waveforms” on page 450).

The DAC can also receive segment data from external sources, like the VXIbus. The external sources immediately set the DAC to an output voltage that corresponds to the DAC code value sent by the source. Each time the DAC receives a new code, the DAC’s output is set to the value in the new code. Thus, the waveform frequency depends on the rate at which the DAC receives the codes.

The output DAC’s voltage range is from -5.12 V to +5.11875 V.
Memory
Concurrent with the DAC, the frequency generator also clocks the segment memory to output the next code to set the DAC bits to the next point on the waveform. By clocking both the memory and DAC at a certain clock rate (i.e., sample rate), the AFG outputs a waveform at a frequency determined by the length and number of waveform segments and the sample rate. (See “AFG Memory Description” on page 452 for more information on how the memory operates.)

Waveform segments and segment sequences can also be stored into memory using external sources, like the VXIbus, for user generated waveforms.

Reference Oscillator
The reference oscillator provides the clock signal for the frequency generator. Thus, frequency stability depends on the stability of the reference oscillator. The oscillator also determines the frequency range of the frequency generators. The AFG allows for user supplied reference oscillator sources for custom frequency values.

Frequency Generators
The frequency generator generates the clock pulses to enable both the output DAC and memory to output a segment. The frequency generator thus determines the rate (i.e., the sample rate) at which the points of a waveform segment are output.

The AFG uses two different generators. One generator (Frequency1 generator) uses a Direct-Digital-Synthesis (NCO) technique to generate the sample frequencies. The other generator (Frequency2 generator) is a Divide-by-n generator.
DDS Frequency Generator (Frequency1 Generator)

Refer to Figure 10-3. This generator has excellent resolution and allows for frequency sweeping, frequency-shift-keying, and output frequency lists. However, its maximum frequency is the Reference Oscillator frequency divided by 4.

To generate precision frequencies for the memory and output DAC clock, the output of the DDS frequency generator is applied to a DAC. The DAC output is filtered and the resultant clock signals clocks the memory and output DAC to create the waveforms.

Divide-by-N Frequency Generator (Frequency2 Generator)

Refer to Figure 10-4. This generator has better phase noise characteristics and permits higher frequency operation (up to the Reference Oscillator Frequency). The output of this filter directly clocks the memory and output DAC.

Figure 10-3. Generating Waveforms Using a Frequency1 Generator

Figure 10-4. Generating Waveforms Using a Frequency2 Generator
Trigger Circuitry

The trigger circuitry advances the waveform to the next segment. The external trigger sources advances the waveform directly. Thus, the sample rate and resultant waveform frequency depend on the frequency of the applied triggers.

Output Circuitry

The output circuitry outputs the waveform at the front panel’s “Output” connector. The circuitry sets the output amplitude, offset voltages, output impedances, and has a 250 kHz and a 10 MHz low-pass filter. (See “Output Circuitry Description” on page 451 for more information.)

Microprocessor

The AFG uses a Motorola 68000 microprocessor to generates the waveform segments and segment sequences for the standard waveforms.

Generating Sinusoid Waveforms

Refer to Figure 10-5. The AFG uses the DDS (Frequency 1) frequency generator to generate Sinusoid waveforms. The generator output directly supplies the DAC data for the output DAC to generate the waveforms.

Figure 10-5. Generating Sinusoid Waveforms
Output Circuitry Description

Refer to Figure 10-6. The output circuitry consists of an output amplifier, attenuator, offset circuitry, and filter. The following describes the different parts of the circuitry.

**Attenuator**

The attenuator provides 30 dB attenuation in .01 dB steps for the output voltage. The AFG automatically sets the attenuator to the appropriate value dependent on the output amplitude selected by the user. The DC volts function does not use the attenuator. For this function, the output of the output DAC is directly output, through the output amplifier, to the “Output” connector.

**Filter**

The AFG provides a 250 kHz low-pass filter, 10 MHz low-pass filter, or no filter. The filters are used to filter the high frequency components, such as clock signals, of the output DAC’s output signal.

**Output Amplifier**

The output amplifier provides the necessary current to drive output loads of 50Ω and 75Ω loads applied to the “Output” connector. For matched loads, the output amplitude are from -5.12 V to +5.11875 V. In addition, the amplifier can also be set for open circuit or infinite loads applied to the “Output” connector. For open circuit outputs, the amplitude range is twice the matched load values.

**Offset Circuitry**

This circuitry offsets the output amplifier to provide an offset voltage.
AFG Memory Description

The segment memory that stores the segment list as DAC codes can store the codes either in the Signed or Unsigned number format. This memory uses 16-bit integer values for the codes. To change the number format to a different format, the memory must be completely empty before selecting the different format.

Because of hardware restrictions, the segment space in memory allocates for a multiple of 8 words for each waveform segment.
This appendix contains the HP E1445A Arbitrary Function Generator operating specifications. Except as noted, the specifications apply under the following conditions:

- **Period:** 1 year
- **Temperature:** 0°C–55°C
- **Relative Humidity:** ≤ 65% @ 0°C–40°C
- **Warm up Time:** 1 hour

“Typical”, “typ”, or “nominal” values are non-warranted supplementary information provided for applications assistance.

### Memory Characteristics

Segment Memory (contains DAC code and Marker Bit for each sample point):

- **DAC Word:** 13 bits
- **Digital Marker:** 1 bit (user-programmable)
- **Size:** 262144 (256K, 1K = 1024)
- **Segment Length:** from ≥ 4 points to 262,144 points.
- **Number of Segments:** 1 to 256
- **System Use:** (while one of the following waveforms is selected):
  - **Square:** 4 points
  - **Triangle, Ramp:** 100 points (default number, unless changed by user)
Sequence Memory (This memory concatenates segments into larger waveforms):

- **Size:** 32768 entries
- **Sequence Length:** 1 to 32768 entries
- **Number of Sequences:** 1 to 128
- **Contents of Each Entry:**
  - Designator of which segment to output
  - Loop Count: # times to repeat the designated segment (1 to 4096, default = 1)
  - "Marker Enable" Bit: a mask (default = enabled) for the marker data in the specified segment
- **System Use:** When square, triangle, or ramp waveform is selected, one entry is used

### Frequency and Sample Rate Characteristics

#### Tolerances
All internally-generated frequencies and rates are ±0.005% initial tolerance. Aging rate is 20 ppm/year.

#### Arbitrary Waveform Sample Rates

Maximum arbitrary waveform sample rate (internal or external rate):

- 40MSa/s (Sa/s = Samples per second)

#### Frequency Generator #1:

- **Internal Reference:** 42.94967296 MHz
- **Rate Generation Method:** Direct Digital Synthesis (DDS)
- **Basic Range:**
  - Minimum: 0.01 Sa/s
  - Maximum: 10.73741824 MSa/s
  - Resolution: 0.01 Sa/s
  - Jitter: 0.03% + 3 nsec (typical rms)
- **Extended Range:**
  - Minimum: 0.02 Sa/s
  - Maximum: 21.47483648 MSa/s
  - Resolution: 0.02 Sa/s
  - Jitter: 0.06% + 3 nsec (typical rms)
- **Pertinent SCPI Commands:**
  - [SOURce:]ROSCillator:SOURce INTernal1
  - [SOURce:]FREQuency1 subsystem
  - TRIGger:STARt:SOURce INTernal1
- **Frequency Agility:**
  - Sweep (linear or log), output frequencies from a list, frequency-shift keying (FSK), phase offset
- **Recommendation:** Use for most applications
Frequency Generator #2:

**Internal Reference:** 40.000000 MHz

**Rate Generation Method:** Divide-by-N, or direct use of reference

**Range:**
- **Minimum:** 305.175781 Sa/s (40/131072 MSa/s)
- **Maximum:** 40.000000 MSa/s
- **Resolution:** Not Applicable. Attainable rates are 40/N MSa/s where N = 1, 2, 3, and all even values up to 131072.
- **Jitter:** 80 psec (typical rms)

**Pertinent SCPI commands:**
- `[SOURce:]ROSCillator:SOURce INTernal2`
- `[SOURce:]FREQuency2 subsystem`
- `TRIGger:STARt:SOURce INTernal2`

**Frequency Agility:** No

**Recommendation:** Use if 40 MSa/s is required, or for lowest jitter at other sample rates.

Built-In Waveforms (using 42.94967296 MHz internal reference oscillator); in each case the frequency resolution equals the minimum frequency:

**Sine Waves:** 0.01 Hz to 10.73741824 MHz

**Square Waves:**
- 0.0025 Hz to 2.68435456 MHz (normal range); Average Duty cycle is 49.9 to 50.1% ± 3 nsec.
- 0.005 Hz to 5.36870912 MHz (doubled range); Average Duty cycle is 44% to 56% ± 3 nsec.
- Using frequency generator #1, square waves inherit the timing jitter characteristics given above for frequency generator #1. The sample period is 1/4 of the square wave period.

**Triangles, Ramps:**
- For the default setting of 100 points per cycle:
  - 0.0001 Hz to 107.3741824 kHz (normal range)
  - 0.0002 Hz to 214.7483648 kHz (doubled range)
- Higher frequencies are possible with fewer than 100 points per cycle. Points per cycle can be 4 to 262144.
Frequency Agility:

The capabilities in this section apply to all built-in standard waveforms and to all arbitrary waveforms generated with Frequency Generator #1 (i.e., the DDS timebase).

**Digital Sweep:**
Linear and Log phase-continuous
(0.2 to 800 points/sec typical) (Note 1)

**Frequency List:**
Up to 256 points phase-continuous
(0.2 to 800 points/sec typical) (Note 1)

**Frequency-Shift Keyed (FSK):**
Up to 2M or f(ref)/5 changes/sec
phase-continuous, whichever is smaller

**Digital Phase Modulation:**
See “Interface Characteristics” later in this appendix

**Note 1:** Sine waves can be leveled at each step of a frequency sweep or list. The speeds above include leveling.

Additional Waveform Control Characteristics:

Waveform repetitions per ARM:STARt:  1 to 65536 or INFinity
Not specified for built-in sine waves. For other waveforms, the final waveform repetition stops at the last sample point.

ARM:STARt events per INITiate:  1 to 65535 or INFinity
Amplitude Characteristics

Low-Pass Filtering: Programmable choice of three configurations:
- 250 kHz (nominal 3 dB point) 5th-order Bessel
- 10 MHz (nominal 3 dB point) 7th-order Bessel
- No Filter

Output Impedance (nominal): 50 Ω or 75 Ω (programmable)

Output Disconnect: Uses a relay. Output is unterminated when relay is open.

DAC Resolution (including sign): 13 bits (12 bits, sine waves only)
- monotonic to 11 bits

DC Volt Function:
Output: into 50 Ω or 75 Ω: -5.12 to +5.11875 volts in nominal steps of 1.25 mv
into open circuit: -10.24 to +10.2375 volts in nominal steps of 2.5 mv

Accuracy: (Temperature within 5 °C of temperature at calibration (Tcal); module calibrated at 18–28 °C; output impedance 50 Ω or 75 Ω; load 50 Ω or 75 Ω respectively, or INF):
- 0.3% of setting + 0.2% of full-scale (add for each ° C beyond 5°C from Tcal)
- 0.05% of setting + 0.015% of full-scale

All Built-In Waveforms:
Output Level: into 50 Ω or 75 Ω: 0.32374 to 10.2375 Vpp
into open circuit: 0.64748 to 20.475  Vpp
Output level adjustability is equivalent to 0–30 dB of attenuation in steps of 0.01 dB.

Sine Waves:
AC Accuracy, 1 kHz, maximum output: ± 0.1 dB
add beyond Tcal ± 5° C: ± 0.005 dB / ° C
Add if output is not at maximum: ± 0.05 dB
Add if frequency is not 1 kHz, flatness error relative to 1 kHz (specified for 50 Ω or 75 Ω only):
250 kHz filter:
- 0.1 Hz to 100 kHz: ± 0.05 dB
- 100 kHz to 250 kHz: ± 0.10 dB
10 MHz filter:
- 1 kHz to 10.73741824 MHz: ± 0.2 dB
(These flatness values are achieved by active compensation for filter frequency response in sine wave mode only, and do not imply dynamic characteristics of arbitrary waveforms.)
Sine Wave Spectral Purity

Output frequencies less than 250 kHz are characterized using the 250 kHz filter, higher output frequencies with the 10 MHz filter. Frequencies given below refer to the desired output sine wave (fc).

**Total Harmonic Distortion (through 9th harmonic):**

- 10 Hz–250 kHz: -60 dBc
- 250 kHz–4 MHz: \[-60 + 20 \log_{10} (fc / 250k)\] dBc
- 4 MHz–10 MHz: -36 dBc

**Nonharmonic Spurious and Clock Components (to 150 MHz):**

- 10 Hz–1 MHz: the greater of -60 dBc or -60 dBm
- 1 MHz–4 MHz: 50 dBc
- 4 MHz–10 MHz: -45 dBc

Arbitrary Waveforms (includes square, triangle, and ramp waveforms):

**DAC Full-Scale:**

- into 50 Ω or 75 Ω:
- into open circuit:

0.16187 to 5.11875 volts
0.32374 to 10.2375 volts

The output voltage corresponding to DAC full-scale can be adjusted, over the indicated 30 dB range, with resolution equivalent to steps of 0.01 dB.

**DC Accuracy:** 0.9% of setting.

- add for each °C beyond Tcal ± 5°: 0.05% of setting
- Add DC Offset (see below)

**Step Response (no filter) (typical)**

- Rise/Fall Time 10–90%: 17 nsec
- Precursors/Overshoot: <1%

**Slew Rate (no filter) (typical)**

- into 50 Ω: 750 V/μs
- into open circuit: 1470 V/μs

**DC Offset:**

- Resolution: 12 bits including sign
- Limit (of waveform + offset):
  - into 50 Ω or 75 Ω:
  - waveform peak(*) volts ≥ 1.02486: ± 5.5 v
  - waveform peak(*) volts < 1.02486: ± 1.1 v
  - into high impedance load:
  - waveform peak(*) volts ≥ 2.04972: ± 11 v
  - waveform peak(*) volts < 2.04972: ± 2.2 v

(*): As used in this table, “waveform peak volts” means the voltage corresponding to DAC full-scale.

**Accuracy:** 1% of setting + 0.2% of limit

Beyond Tcal ± 5° C, add 0.015% of limit per °C.
Interface Characteristics

BNC Connector Functions

TTL levels, except for analog output

Outputs

“Output 50/75 Ohm” - analog output 50 Ω or 75 Ω nominal; or open

“Marker Out” (This output is parallel-terminated with 50 Ω nominal)
- marker bits stored with arbitrary waveforms
- reference frequency
- waveform clock
- a pulse indicating each waveform repetition
- a level change at the start and the end of each burst of waveform repetitions
- frequency change
- phase change

Inputs

“Ref/Sample In” - external reference frequency (40 MHz maximum)
- trigger source (i.e., the waveform clock)

“Start Arm In” - start arm (enables waveform clock for a burst of waveform repetitions

“Stop Trig/FSK/Gate In” - waveform clock stop (causes the current waveform repetition to be the last, until another Start Arm)
- FSK Input
- waveform clock gate

NOTE: High impedance pulled up to +5 V through 4.7 KΩ resistor.
External source must be able to sink 1 mA.
**VXI ECLTrg Functions**

**Input Functions**
- reference frequency in trigger source in (i.e. the waveform clock) start arm in (enables waveform clock)

**Output Functions**
- marker bits stored with arbitrary waveforms
- reference frequency
- waveform clock
- a pulse indicating each waveform repetition
- a level change at the start and the end of each burst of waveform repetitions
- frequency change
- phase change

**VXI TTLTrg Functions**

**Input Functions**
- trigger source (i.e., the waveform clock)
- waveform clock gate/FSK Input
- start arm (enables waveform clock)
- waveform clock stop (causes the current waveform repetition to be the last)
- sweep arm (starts sweep or frequency list)
- sweep trigger (go to next point in sweep or frequency list)

**Output Functions**
- none

**Front Panel “Digital Port In” Connector**

**Connector Type:** 25-pin D-type receptacle

**Signal Lines:** 16 data, ext clock, int clock

**Logic Compatibility:** TTL

**Functions:**
- data to DAC
- download to segment memory
- waveform select (Note 2)
- phase modulation (8 bits)

**Data Rate:** 1M transfers/s typical

**Local Bus**

**Bus Type:** ECL

**Functions:**
- data to DAC
- download to segment memory
- waveform select (Note 2)
- phase modulation (8 bits)
- data pass-through

**Data Rate (typical):** 7Msa/s; 2M/s for phase modulation
VME Register Access

All hardware registers are mapped directly into VME A24 space, permitting advanced users to bypass the on-board uP. The manual documents a functional subset. While a waveform is running, waveform memory may not be loaded, but on-the-fly re-selection (Note 2) permits a new sequence to begin immediately upon completing the present sequence.

Note 2: “Waveform Select” Up to 128 waveforms (sequences) can be stored in memory, and then selected/re-selected on-the-fly by digital words arriving on the Local Bus (typ 7 Msa/s), the Faceplate Connector (typ 1 M/s), or the VME bus (typ 2 M/s).

General VXIbus Characteristics

Size: C
Slots: 1
Connectors: P1, P2
Weight (kg): 1.9
Device Type: Message-Based Servant
VXIbus Revision Compliance: 1.3
Register Level Documentation: Subset
SCPI Revision: 1991.0
Manufacturer Code: 4095 Decimal
Model Code: 418 Decimal
Slave: A16/A24 D08/D16
Master: A16/A24 D08/D16
(The HP E1445A can control the HP E1446A Summing Amplifier/DAC.)

Currents in Amps:

<table>
<thead>
<tr>
<th>Currents</th>
<th>+5 V</th>
<th>+12 V</th>
<th>-12 V</th>
<th>+24 V</th>
<th>-24 V</th>
<th>-5.2 V</th>
<th>-2 V</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM (IPM)</td>
<td>3.50</td>
<td>0.20</td>
<td>0.12</td>
<td>0.10</td>
<td>0.13</td>
<td>0.06</td>
<td>0.28</td>
</tr>
<tr>
<td>DM (IDM)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Average Watts/Slot: 40-44
dPressure (mm H2O): 0.5
Air Flow (liters/s): 3.5
Appendix Contents

The tables in this appendix contain information often referred to during HP E1445A programming. The tables in this appendix include:

- Table B-1. HP E1445A Example Program Listing .................................................. Page 464
- Table B-2. HP E1445A Command Coupling Groups ................................................ Page 467
- Table B-3. HP E1445A Frequency Limits ................................................................. Page 470
- Table B-4. HP E1445A Amplitude Limits ................................................................. Page 471
- Table B-5. HP E1445A Power-on/Reset Conditions ................................................ Page 472
- Table B-6. HP E1445A Error Messages ................................................................. Page 475
- Table B-7. HP E1445A Settings Conflict Error Messages ........................................ Page 480
## Table B-1. HP E1445A Example Program Listing

<table>
<thead>
<tr>
<th>Program Type</th>
<th>Program Name</th>
<th>Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RSTCLS</td>
<td>&quot;</td>
<td>Resetting and clearing the AFG.</td>
</tr>
<tr>
<td></td>
<td>LRN</td>
<td>&quot;</td>
<td>Power-on/reset configuration.</td>
</tr>
<tr>
<td></td>
<td>ERRORCHK</td>
<td>&quot;</td>
<td>Error checking program.</td>
</tr>
<tr>
<td></td>
<td>RSTSINE</td>
<td>&quot;</td>
<td>Sine wave output from reset settings.</td>
</tr>
<tr>
<td>Standard Functions</td>
<td>DCVOLTS</td>
<td>HP BASIC, Visual BASIC, Visual C/ C++</td>
<td>+5V DC voltage.</td>
</tr>
<tr>
<td></td>
<td>SINEWAVE</td>
<td>&quot;</td>
<td>1kHz, 5Vp sine wave.</td>
</tr>
<tr>
<td></td>
<td>SQUWAVE</td>
<td>&quot;</td>
<td>4V, 5 MHz square wave - 1V DC offset.</td>
</tr>
<tr>
<td></td>
<td>TRIWAVE</td>
<td>&quot;</td>
<td>200 point, 4V, 10 kHz triangle wave.</td>
</tr>
<tr>
<td></td>
<td>OUTPLOAD</td>
<td>&quot;</td>
<td>Sets AFG's output impedance and load.</td>
</tr>
<tr>
<td></td>
<td>OUTPUNIT</td>
<td>&quot;</td>
<td>Sets amplitude units to volts peak-to-peak.</td>
</tr>
<tr>
<td></td>
<td>PHS_MOD</td>
<td>&quot;</td>
<td>Shifts sine wave phase from 0 to 180 degrees.</td>
</tr>
<tr>
<td></td>
<td>MULSEG</td>
<td>&quot;</td>
<td>Arbitrary waveform with two segments.</td>
</tr>
<tr>
<td></td>
<td>AFGGEN1</td>
<td>&quot;</td>
<td>Ramp arbitrary waveform using the frequency1 generator.</td>
</tr>
<tr>
<td></td>
<td>AFGGEN2</td>
<td>&quot;</td>
<td>Ramp arbitrary waveform using the frequency2 generator.</td>
</tr>
<tr>
<td></td>
<td>SIN_X</td>
<td>&quot;</td>
<td>Sin(x)/x arbitrary waveform.</td>
</tr>
<tr>
<td></td>
<td>SIN_D</td>
<td>&quot;</td>
<td>Damped sine wave arbitrary waveform.</td>
</tr>
<tr>
<td></td>
<td>CHARGE</td>
<td>&quot;</td>
<td>Exponential charge/discharge waveform.</td>
</tr>
<tr>
<td></td>
<td>SPIKES</td>
<td>&quot;</td>
<td>Sine wave with spikes.</td>
</tr>
<tr>
<td></td>
<td>SIN_R</td>
<td>&quot;</td>
<td>1/2 wave rectified sine wave.</td>
</tr>
<tr>
<td></td>
<td>NOISE</td>
<td>&quot;</td>
<td>Pseudo-random noise.</td>
</tr>
<tr>
<td>Program Type</td>
<td>Program Name</td>
<td>Language</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>--------------</td>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Sweeping, Frequency Lists, Frequency-Shift Keying</td>
<td>SMPLSWP1</td>
<td>HP BASIC, Visual BASIC, Visual C/ C++</td>
<td>0 Hz to 1 MHz sweep using start and stop frequencies.</td>
</tr>
<tr>
<td>(Chapter 4)</td>
<td>LIST1</td>
<td>&quot;</td>
<td>1 kHz, 10 kHz, 100 kHz, 1 MHz frequency list.</td>
</tr>
<tr>
<td></td>
<td>SMPLSWP2</td>
<td>&quot;</td>
<td>1 kHz to 21 kHz sweep using start and span frequencies.</td>
</tr>
<tr>
<td></td>
<td>LISTDEF</td>
<td>&quot;</td>
<td>Definite length arbitrary block frequency list</td>
</tr>
<tr>
<td></td>
<td>LOG_SWP</td>
<td>&quot;</td>
<td>Seven point logarithmic frequency sweep.</td>
</tr>
<tr>
<td></td>
<td>SWP_PVST</td>
<td>&quot;</td>
<td>Setting the sweep time.</td>
</tr>
<tr>
<td></td>
<td>LIST_TME</td>
<td>&quot;</td>
<td>Setting the time through a frequency list.</td>
</tr>
<tr>
<td></td>
<td>SWP_ARB</td>
<td>&quot;</td>
<td>Sweeping an arbitrary waveform.</td>
</tr>
<tr>
<td></td>
<td>SWP_LEVL</td>
<td>&quot;</td>
<td>Sweep with output leveling.</td>
</tr>
<tr>
<td></td>
<td>FSK1</td>
<td>&quot;</td>
<td>Frequency-shift keying with the FSK In control source.</td>
</tr>
<tr>
<td></td>
<td>FSK2</td>
<td>&quot;</td>
<td>Frequency-shift keying with the TTLTrg control source.</td>
</tr>
<tr>
<td></td>
<td>FSK_ARB</td>
<td>&quot;</td>
<td>Frequency-shift keying of an arbitrary waveform.</td>
</tr>
<tr>
<td>Arming and Triggering</td>
<td>EXT_ARM</td>
<td>HP BASIC, Visual BASIC, Visual C/ C++</td>
<td>Arming the AFG with a signal applied to the Start Arm In BNC.</td>
</tr>
<tr>
<td>(Chapter 5)</td>
<td>BURST</td>
<td>&quot;</td>
<td>5 cycle burst for each external arm.</td>
</tr>
<tr>
<td></td>
<td>DIV_N</td>
<td>&quot;</td>
<td>10 MHz using the frequency2 generator.</td>
</tr>
<tr>
<td></td>
<td>LOCKSTEP</td>
<td>&quot;</td>
<td>Triggering Two AFGs with a common trigger signal.</td>
</tr>
<tr>
<td></td>
<td>STOPTRIG</td>
<td>&quot;</td>
<td>Aborting a cycle count using stop triggers.</td>
</tr>
<tr>
<td></td>
<td>GATE</td>
<td>&quot;</td>
<td>Gating the output on and off.</td>
</tr>
<tr>
<td></td>
<td>SWP_TRIG</td>
<td>&quot;</td>
<td>Arming and triggering a sweep using group execute trigger.</td>
</tr>
<tr>
<td></td>
<td>SWP_STEP</td>
<td>&quot;</td>
<td>Arming and triggering a sweep.</td>
</tr>
<tr>
<td></td>
<td>LIST_STP</td>
<td>&quot;</td>
<td>Arming and triggering a frequency list.</td>
</tr>
<tr>
<td>Marker Outputs</td>
<td>MARKSEG1</td>
<td>HP BASIC, Visual BASIC, Visual C/ C++</td>
<td>Outputting marker pulses with selected amplitude points.</td>
</tr>
<tr>
<td>(Chapter 6)</td>
<td>MARKSEG2</td>
<td>&quot;</td>
<td>Outputting a single marker pulse.</td>
</tr>
<tr>
<td></td>
<td>MARKTRG</td>
<td>&quot;</td>
<td>Outputting a marker pulse with each amplitude point.</td>
</tr>
<tr>
<td></td>
<td>DRIFT</td>
<td>&quot;</td>
<td>Two AFGs using the same reference osc.</td>
</tr>
<tr>
<td>Program Type</td>
<td>Program Name</td>
<td>Language</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------------</td>
<td>-----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>High-Speed Data Transfer</td>
<td>SIGN_DAT</td>
<td>HP BASIC, Visual BASIC, Visual C/ C++</td>
<td>Downloads arbitrary waveform data as signed DAC codes.</td>
</tr>
<tr>
<td>(Chapter 7)</td>
<td>UNS_DAT</td>
<td>&quot;</td>
<td>Downloads arbitrary waveform data as unsigned DAC codes.</td>
</tr>
<tr>
<td></td>
<td>DACBLOK1</td>
<td>&quot;</td>
<td>Downloads arbitrary waveform data as signed DAC codes in a definite length block</td>
</tr>
<tr>
<td></td>
<td>DACBLOK2</td>
<td>&quot;</td>
<td>Downloads arbitrary waveform data as unsigned DAC codes in an indefinite length block</td>
</tr>
<tr>
<td></td>
<td>COMBSIGN</td>
<td>&quot;</td>
<td>Downloads waveform amplitude and marker data as signed DAC codes in a definite length block</td>
</tr>
<tr>
<td></td>
<td>COMBUNS</td>
<td>&quot;</td>
<td>Downloads waveform amplitude and marker data as unsigned DAC codes in an indefinite length block</td>
</tr>
<tr>
<td></td>
<td>COMBSEQ</td>
<td>&quot;</td>
<td>Downloads waveform amplitude and marker data as signed DAC codes in definite length blocks. Downloads the output sequence (including repetition count, marker, and segment address) in an indefinite length block.</td>
</tr>
<tr>
<td></td>
<td>VXIDOWN</td>
<td>&quot;</td>
<td>Downloads waveform amplitude and marker data over the VXIbus backplane.</td>
</tr>
<tr>
<td></td>
<td>VXISRCE</td>
<td>&quot;</td>
<td>Writes data directly to the DAC from the VXIbus backplane. (See also Appendix C.)</td>
</tr>
<tr>
<td></td>
<td>WAVSELFIP</td>
<td>&quot;</td>
<td>Changes output waveform sequence by writing location of a sequence’s base address to the Waveform Select Register.</td>
</tr>
<tr>
<td>AFG Status</td>
<td>QSSG_RQS</td>
<td>HP BASIC, Visual C/ C++</td>
<td>Monitors conditions in the Questionable Signal Status Group.</td>
</tr>
<tr>
<td>(Chapter 9)</td>
<td>OSG_RQS</td>
<td>&quot;</td>
<td>Monitors conditions in the Operation Status Group.</td>
</tr>
<tr>
<td></td>
<td>ERRORCHK</td>
<td>&quot;</td>
<td>Monitors programming errors using the Standard Event Status Group.</td>
</tr>
<tr>
<td>Register-Based Applications</td>
<td>FREQ1REG</td>
<td>HP BASIC, Visual BASIC, Visual C/ C++</td>
<td>Changes the output frequency generated by the DDS (Direct-Digital-Synthesis) chip (Frequency1 generator) by writing directly to the registers.</td>
</tr>
<tr>
<td>(Appendix C)</td>
<td>FREQ2REG</td>
<td>&quot;</td>
<td>Changes the output frequency generated by the Divide-by-N chip (Frequency2 generator) by writing directly to the registers.</td>
</tr>
<tr>
<td></td>
<td>PHASCHNG</td>
<td>&quot;</td>
<td>Changes the signal phase by writing directly to the registers.</td>
</tr>
<tr>
<td></td>
<td>WAVE_SEL</td>
<td>&quot;</td>
<td>Changes the output waveform sequence by writing directly to the registers.</td>
</tr>
<tr>
<td></td>
<td>VXISRCE</td>
<td>&quot;</td>
<td>Writes data directly to the DAC from the VXIbus backplane.</td>
</tr>
</tbody>
</table>
# Command Coupling Groups

## Table B-2. HP E1445A Command Coupling Groups

<table>
<thead>
<tr>
<th>Coupling Group</th>
<th>Commands</th>
</tr>
</thead>
</table>
| None           | [SOURce:]LIST2:FORMat[:DATA]  
[SOURce:]LIST2:FREQuency:POINts?
[SOURce:]MARKer:ELTrg<n>:FEED  
[SOURce:]MARKer:ELTrg<n>[STATE]
[SOURce:]MARKer:FEED  
[SOURce:]MARKer:POLarity  
[SOURce:]MARKer[:STATE]
[SOURce]:PM[:DEViation]  
[SOURce]:PM:UNIT[:ANGLE]
[SOURce]:VOLTage[:LEVEL][:IMMediate][:AMPLitude]:UNIT[:VOLTage]
STATus:OPC:INITiate  
STATus:OPERation:CONDition?
STATus:OPERation:ENABLE  
STATus:OPERation[:EVENt]?
STATus:OPERation:NTRansition  
STATus:OPERation:PTRansition  
STATus:QUEStionable:CONDition?
STATus:QUEStionable:ENABLE  
STATus:QUEStionable[:EVENt]?
STATus:QUEStionable:NTRansition  
STATus:QUEStionable:PTRansition  
STATus:PRESet  
TRIGger[:STARt][:IMMediate]
TRIGger[:STARt]:SLOPe  
TRIGger:STOP[:IMMediate]
TRIGger:SWEep[:IMMediate]
VINStrument[CONfigure]:LBUS[:MODE]
VINStrument[CONfigure]:LBUS[:MODE]:AUTO  
VINStrument[CONfigure]:TEST:CONFigure  
VINStrument[CONfigure]:TEST:DATA?
VINStrument[CONfigure]:VME[:MODE]
VINStrument[CONfigure]:VME:REceive:ADDRess:DATA?
VINStrument[CONfigure]:VME:REceive:ADDRess:READy?
VINStrument:IDENtity?
| Frequency | ARM:SWEep:COUNt  
ARM:SWEep:SOURce |
<table>
<thead>
<tr>
<th>Coupling Group</th>
<th>Commands</th>
</tr>
</thead>
</table>
| Frequency      | [SOURce:]FREQuency[1]:CENTer  
[SOURce:]FREQuency[1]:CW:FIXed  
[SOURce:]FREQuency[1]:FSKey  
[SOURce:]FREQuency[1]:FSKey:SOURce  
[SOURce:]FREQuency[1]:MODE  
[SOURce:]FREQuency[1]:RANGe  
[SOURce:]FREQuency[1]:SPAN  
[SOURce:]FREQuency[1]:STARt  
[SOURce:]FREQuency[1]:STOP  
[SOURce:]FREQuency2[:CW]:FIXed  
[SOURce:]LIST2:FREQuency  
[SOURce:]PM:SOURce  
[SOURce:]PM:STATE  
[SOURce:]ROSCillator:FREQuency:EXTernal  
[SOURce:]ROSCillator:SOURce  
[SOURce:]SWEep:COUNT  
[SOURce:]SWEep:DIRection  
[SOURce:]SWEep:POINts  
[SOURce:]SWEep:SPACing  
[SOURce:]SWEep:TIME  
TRIGger[:STARt]:GATE:POLarity  
TRIGger[:STARt]:GATE:SOURce  
TRIGger[:STARt]:GATE:STATE  
TRIGger[:STARt]:SOURce  
TRIGger:STOP:SLOPe  
TRIGger:STOP:SOURce  
TRIGger:SWEep:SOURce  
TRIGger:SWEep:TIME |
| Voltage        | OUTPut[1]:IMPedance  
OUTPut[1]:LOAD  
OUTPut[1]:LOAD:AUTO  
[SOURce:]RAMP:POLarity  
[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude]  
[SOURce:]VOLTage[:LEVel][:IMMediate]:OFFSet |
| Frequency & Voltage | [SOURce:]ARBitrary:DAC:SOURce  
[SOURce:]FUNCtion[:SHAPe]  
[SOURce:]RAMP:POINts |
| None           | ABORt  
ARM[:STARt][:LAYer[1]]:COUNt  
ARM[:STARt][:LAYer2]:COUNt |
Table B-2. HP E1445A Command Coupling Groups (continued)

<table>
<thead>
<tr>
<th>Coupling Group</th>
<th>Commands</th>
</tr>
</thead>
</table>
| None           | ARM[:STARt]:LAYer2[:IMMediate]  
|                | ARM[:STARt]:LAYer2:SLOPe  
|                | ARM[:STARt]:LAYer2:SOURce  
|                | ARM:SWEep[:IMMediate]  
|                | INITiate[:IMMediate]  
|                | OUTPut[1]:FIlTer[:LPASs]:FREQuency  
|                | OUTPut[1]:FIlTer[:LPASs][:STATe]  
|                | OUTPut[1][:STATe]  
|                | [SOURce:]ARBitrary:DAC:FORMat  
|                | [SOURce:]ARBitrary:DOWNload  
|                | [SOURce:]ARBitrary:DOWNload:COMPLETE  
|                | [SOURce:]FUNCTion:USER  
|                | [SOURce:]LIST[1]:FORMat[:DATA]  
|                | [SOURce:]LIST[1]:SEGMen|t[:ADDRes|s?]  
|                | [SOURce:]LIST[1]:SEGMen|t[:CATalog?]  
|                | [SOURce:]LIST[1]:SEGMen|t[:COMB|ined]  
|                | [SOURce:]LIST[1]:SEGMen|t[:COMB|ined]:POIN|ts?]  
|                | [SOURce:]LIST[1]:SEGMen|t[:DEF|ine]  
|                | [SOURce:]LIST[1]:SEGMen|t[:DELe|te:A|LL]  
|                | [SOURce:]LIST[1]:SEGMen|t[:DELe|te[:SELe|cted|]]  
|                | [SOURce:]LIST[1]:SEGMen|t[:FRE|E?]  
|                | [SOURce:]LIST[1]:SEGMen|t[:MAR|Ker]  
|                | [SOURce:]LIST[1]:SEGMen|t[:MAR|Ker]:POIN|ts?]  
|                | [SOURce:]LIST[1]:SEGMen|t[:MAR|Ker]:SP|Oin  
|                | [SOURce:]LIST[1]:SEGMen|t[:SELe|ct]  
|                | [SOURce:]LIST[1]:SEGMen|t[:VOL|Tage]  
|                | [SOURce:]LIST[1]:SEGMen|t[:VOL|Tage:DA|C]  
|                | [SOURce:]LIST[1]:SEGMen|t[:VOL|Tage:POIN|ts?]  
|                | [SOURce:]LIST[1]:SSEQ|uence[:ADDRes|s?]  
|                | [SOURce:]LIST[1]:SSEQ|uence[:CATalog?]  
|                | [SOURce:]LIST[1]:SSEQ|uence[:COMB|ined]  
|                | [SOURce:]LIST[1]:SSEQ|uence[:COMB|ined]:POIN|ts?]  
|                | [SOURce:]LIST[1]:SSEQ|uence[:DEF|ine]  
|                | [SOURce:]LIST[1]:SSEQ|uence[:DELe|te:A|LL]  
|                | [SOURce:]LIST[1]:SSEQ|uence[:DELe|te[:SELe|cted|]]  
|                | [SOURce:]LIST[1]:SSEQ|uence[:DWE|L]:COUNT  
|                | [SOURce:]LIST[1]:SSEQ|uence[:DWE|L]:COUNT:POIN|ts?]  
|                | [SOURce:]LIST[1]:SSEQ|uence[:FRE|E?]  
|                | [SOURce:]LIST[1]:SSEQ|uence[:MAR|Ker]  
|                | [SOURce:]LIST[1]:SSEQ|uence[:MAR|Ker]:POIN|ts?]  
|                | [SOURce:]LIST[1]:SSEQ|uence[:MAR|Ker]:SP|Oin  
|                | [SOURce:]LIST[1]:SSEQ|uence[:SELe|ct]  
|                | [SOURce:]LIST[1]:SSEQ|uence[:SEQuence]  
|                | [SOURce:]LIST[1]:SSEQ|uence[:SEQuence:SEGM|ents?]  

Appendix B  
Useful Tables  469
## Frequency Limits

Table B-3. HP E1445A Frequency Limits

<table>
<thead>
<tr>
<th>Function</th>
<th>Trigger Start Source</th>
<th>Frequency Source</th>
<th>Low Limit</th>
<th>High Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>SINusoid</td>
<td>INTernal[1]</td>
<td>0 Hz</td>
<td>Ref Osc freq/4</td>
<td></td>
</tr>
<tr>
<td>SQUare</td>
<td>INTernal[1]*</td>
<td>0 Hz</td>
<td>Ref Osc freq/16</td>
<td></td>
</tr>
<tr>
<td>INTernal2</td>
<td>Ref Osc freq/4/131072</td>
<td>Ref Osc freq/4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRIangle</td>
<td>INTernal[1]*</td>
<td>0 Hz</td>
<td>Ref Osc freq/4</td>
<td>Ramp Points</td>
</tr>
<tr>
<td>INTernal2</td>
<td>Ref Osc freq/131072/Ramp Points</td>
<td>Ref Osc freq/Ramp Points</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAMP</td>
<td>INTernal[1]*</td>
<td>0 Hz</td>
<td>Ref Osc freq/4</td>
<td>Ramp Points</td>
</tr>
<tr>
<td>INTernal2</td>
<td>Ref Osc freq/131072/Ramp Points</td>
<td>Ref Osc freq/Ramp Points</td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER</td>
<td>INTernal[1]*</td>
<td>0 Hz</td>
<td>Ref Osc freq/4</td>
<td></td>
</tr>
<tr>
<td>INTernal2</td>
<td>Ref Osc freq/131072</td>
<td>Ref Osc freq</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Multiply High Limit frequency values by 2 if frequency doubling is selected by the [SOURce:]FREQuency[1]:RANGe command.
## Amplitude Limits

### Table B-4. HP E1445A Amplitude Limits

#### Amplitude Limits for Volts*

<table>
<thead>
<tr>
<th>Function</th>
<th>V (volts)</th>
<th>VPK (volts peak)</th>
<th>VPP (volts peak-to-peak)</th>
<th>VRMS (volts rms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC</td>
<td>-5.12 to +5.11875</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>SINusoid</td>
<td>+.161869088 to +5.11875</td>
<td>+.161869088 to +5.11875</td>
<td>+.323738175 to +10.2375</td>
<td>+.114458730 to +3.61950284</td>
</tr>
<tr>
<td>SQUare</td>
<td>+.161869088 to +5.11875</td>
<td>+.161869088 to +5.11875</td>
<td>+.323738175 to +10.2375</td>
<td>+.161869088 to +5.11875</td>
</tr>
<tr>
<td>TRLangle</td>
<td>+.161869088 to +5.11875</td>
<td>+.161869088 to +5.11875</td>
<td>+.323738175 to +10.2375</td>
<td>+.0934551614 to +2.9553117</td>
</tr>
<tr>
<td>RAMP</td>
<td>+.161869088 to +5.11875</td>
<td>+.161869088 to +5.11875</td>
<td>+.323738175 to +10.2375</td>
<td>+.0934551614 to +2.9553117</td>
</tr>
<tr>
<td>USER</td>
<td>+.161869088 to +5.11875</td>
<td>+.161869088 to +5.11875</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

*Double the values for Open Circuit Loads selected by OUTPut[1]:LOAD INFinity.

#### Amplitude Limits for Watts and dB**

| Function  | W (Watt @50Ω load) | W (Watt @75Ω load) | DBM|DBMW (dBmW @50Ω load) | DBM|DBMW (dBmW @75Ω load) |
|-----------|---------------------|---------------------|--------------------------|--------------------------|
| DC        | N/A                 | N/A                 | N/A                      | N/A                      |
| SINusoid  | +.000262016016 to +.262016016 | +.000174677344 to +.174677344 | -5.81672162 to +24.1832784 | -7.57763421 to +22.4223658 |
| SQUare    | +.000524032031 to +.524032031 | +.000349354678 to +.349354678 | -2.80642166 to +27.1935783 | -4.56733425 to +25.4326657 |
| TRLangle  | +.000174677344 to +.174677344 | +.000116451562 to +.116451562 | -7.57763421 to +22.4223658 | -9.33854680 to +20.6614532 |
| RAMP      | +.000174677344 to +.174677344 | +.000116451562 to +.116451562 | -7.57763421 to +22.4223658 | -9.33854680 to +20.6614532 |

**Not available with OUTPut[1]:LOAD INFinity selected

*Double the values for Open Circuit Loads selected by OUTPut[1]:LOAD INFinity.

Appendix B Useful Tables 471
## Power-On/Reset Configuration

Table B-5. HP E1445A Power-On/Reset Configuration (as returned by ‘LRN?)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Command</th>
<th>Power-on/Reset Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro usage</td>
<td>*EMC</td>
<td>+1</td>
</tr>
<tr>
<td>Calibration state</td>
<td>:CAL:STAT</td>
<td>1 (enabled)</td>
</tr>
<tr>
<td>AC calibration</td>
<td>:CAL:STAT:AC</td>
<td>1 (enabled)</td>
</tr>
<tr>
<td>DC calibration</td>
<td>:CAL:STAT:DC</td>
<td>1 (enabled)</td>
</tr>
<tr>
<td>DAC data source</td>
<td>:ARB:DAC:SOUR</td>
<td>INTernal</td>
</tr>
<tr>
<td>Phase modulation units</td>
<td>:PM:UNIT:ANGL</td>
<td>RADians</td>
</tr>
<tr>
<td>Waveform amplitude units</td>
<td>:VOLT:AMPL:UNIT:VOLT</td>
<td>V</td>
</tr>
<tr>
<td>Sample source</td>
<td>TRIG:SOUR</td>
<td>INTernal</td>
</tr>
<tr>
<td>Sample gate polarity</td>
<td>TRIG:GATE:POL</td>
<td>INVerted</td>
</tr>
<tr>
<td>Sample gate source</td>
<td>TRIG:GATE:SOUR</td>
<td>EXTernal</td>
</tr>
<tr>
<td>Gating State</td>
<td>TRIG:GATE:STAT</td>
<td>0 (off)</td>
</tr>
<tr>
<td>Output frequency</td>
<td>:FREQ:FIX</td>
<td>+1.000000000E+004</td>
</tr>
<tr>
<td>Frequency-shift keying (FSK)</td>
<td>:FREQ:FSK</td>
<td>+1.000000000E+004,</td>
</tr>
<tr>
<td>frequencies</td>
<td></td>
<td>+1.000000000E+007</td>
</tr>
<tr>
<td>FSK trigger source</td>
<td>:FREQ:FSK:SOUR</td>
<td>EXTernal</td>
</tr>
<tr>
<td>Frequency mode</td>
<td>:FREQ:MODE</td>
<td>FIXed</td>
</tr>
<tr>
<td>Frequency range</td>
<td>:FREQ:RANG</td>
<td>+0.000000000E+000</td>
</tr>
<tr>
<td>Sweep start frequency</td>
<td>:FREQ:STAR</td>
<td>+0.000000000E+000</td>
</tr>
<tr>
<td>Sweep stop frequency</td>
<td>:FREQ:STOP</td>
<td>+1.073741824E+007</td>
</tr>
<tr>
<td>Output frequency (divide-by-n</td>
<td>:FREQ2:FIX</td>
<td>+1.000000000E+004</td>
</tr>
<tr>
<td>generator)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference oscillator source</td>
<td>:ROSC:SOUR</td>
<td>INTernal1</td>
</tr>
<tr>
<td>External oscillator frequency</td>
<td>FREQ:EXT</td>
<td>+4.294967296E+007</td>
</tr>
<tr>
<td>Sweep count</td>
<td>:SWE:COUN</td>
<td>+1.000000000E+000</td>
</tr>
<tr>
<td>Sweep direction</td>
<td>:SWE:DIR</td>
<td>UP</td>
</tr>
<tr>
<td>Sweep points</td>
<td>:SWE:POIN</td>
<td>+800</td>
</tr>
<tr>
<td>Parameter</td>
<td>Command</td>
<td>Power-on/Reset Setting</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Sweep spacing (points)</td>
<td>:SWE:SPAC</td>
<td>LINear</td>
</tr>
<tr>
<td>Sweep time</td>
<td>:SWE:TIME</td>
<td>+1.000000000E+000</td>
</tr>
<tr>
<td>Stop trigger source</td>
<td>TRIG:STOP:SOUR</td>
<td>HOLD</td>
</tr>
<tr>
<td>Sweep start source</td>
<td>ARM:SWE:SOUR</td>
<td>IMMEDIATE</td>
</tr>
<tr>
<td>Sweep advance source</td>
<td>TRIG:SWE:SOUR</td>
<td>TIMER</td>
</tr>
<tr>
<td>Function</td>
<td>:FUNC:SHAP</td>
<td>SINusoid</td>
</tr>
<tr>
<td>Ramp/triangle waveform points</td>
<td>:RAMP:POIN</td>
<td>+100</td>
</tr>
<tr>
<td>Ramp/triangle waveform polarity</td>
<td>:RAMP:POL</td>
<td>NORMAL</td>
</tr>
<tr>
<td>Output amplitude</td>
<td>:VOLT:AMPL</td>
<td>+1.61869088E-001</td>
</tr>
<tr>
<td>DC offset</td>
<td>:VOLT:OFFS</td>
<td>+0.000000000E+000</td>
</tr>
<tr>
<td>Output impedance</td>
<td>OUTP:IMP</td>
<td>+5.000000000E+001</td>
</tr>
<tr>
<td>Output load</td>
<td>OUTP:LOAD</td>
<td>+5.000000000E+001</td>
</tr>
<tr>
<td>Load-Impedance coupling</td>
<td>OUTP:LOAD:AUTO</td>
<td>1 (on)</td>
</tr>
<tr>
<td>Waveform repetitions (burst)</td>
<td>ARM:COUN</td>
<td>+9.900000000E+037</td>
</tr>
<tr>
<td>Waveform arm count</td>
<td>ARM:LAY2:COUN</td>
<td>+1.000000000E+000</td>
</tr>
<tr>
<td>External arm slope</td>
<td>ARM:LAY2:SLOP</td>
<td>POS</td>
</tr>
<tr>
<td>Arm source</td>
<td>ARM:LAY2:SOUR</td>
<td>IMMEDIATE</td>
</tr>
<tr>
<td>Arbitrary waveform sequence</td>
<td>:FUNC:USER</td>
<td>NONE</td>
</tr>
<tr>
<td>Segment/sequence return data format and length</td>
<td>:LIST:FORM</td>
<td>ASCII,+9</td>
</tr>
<tr>
<td>Frequency list return data format and length</td>
<td>:LIST2:FORM</td>
<td>ASCII,+10</td>
</tr>
<tr>
<td>ECL trigger line 0 marker source</td>
<td>:MARK:ECLT0:FEED</td>
<td>&quot;ARM&quot;</td>
</tr>
<tr>
<td>Marker routing (ECLT0 line)</td>
<td>:MARK:ECLT0:STAT</td>
<td>0 (off)</td>
</tr>
<tr>
<td>ECL trigger line 1 marker source</td>
<td>:MARK:ECLT1:FEED</td>
<td>&quot;TRIG&quot;</td>
</tr>
<tr>
<td>Marker routing (ECLT1 line)</td>
<td>:MARK:ECLT1:STAT</td>
<td>0 (off)</td>
</tr>
<tr>
<td>&quot;Marker Out&quot; BNC source</td>
<td>:MARK:FEED</td>
<td>&quot;ARM&quot;</td>
</tr>
<tr>
<td>&quot;Marker Out&quot; signal polarity</td>
<td>:MARK:POL</td>
<td>NORM</td>
</tr>
</tbody>
</table>
Table B-5. HP E1445A Power-On/Reset Configuration (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Command</th>
<th>Power-on/Reset Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Marker Out&quot; BNC state</td>
<td>:MARK:STAT</td>
<td>1 (on)</td>
</tr>
<tr>
<td>Output state</td>
<td>OUTP:STAT</td>
<td>1 (on)</td>
</tr>
<tr>
<td>Output filter frequency</td>
<td>:FILT:FREQ</td>
<td>+2.50000000E+005</td>
</tr>
<tr>
<td>Output filter state</td>
<td>:FILT:STAT</td>
<td>0 (off)</td>
</tr>
<tr>
<td>Phase modulation deviation</td>
<td>:PM:DEV</td>
<td>+0.00000000E+000</td>
</tr>
<tr>
<td>Phase modulation source</td>
<td>:PM:SOUR</td>
<td>INTernal</td>
</tr>
<tr>
<td>Phase modulation state</td>
<td>:PM:STAT</td>
<td>0 (off)</td>
</tr>
<tr>
<td>External waveform advance trigger slope</td>
<td>TRIG:SLOP</td>
<td>POS</td>
</tr>
<tr>
<td>External stop trigger slope</td>
<td>TRIG:STOP:SLOP</td>
<td>POS</td>
</tr>
<tr>
<td>Local bus mode</td>
<td>:VINS:LBUS:REC:MODE</td>
<td>OFF</td>
</tr>
<tr>
<td>Local bus automatic mode</td>
<td>:MODE:AUTO</td>
<td>1 (on)</td>
</tr>
</tbody>
</table>
## Error Messages

### Table B-6. HP E1445A Error Messages

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-101</td>
<td>Invalid character</td>
<td>Unrecognized character in parameter.</td>
</tr>
<tr>
<td>-102</td>
<td>Syntax error</td>
<td>Command is missing a space or comma between parameters.</td>
</tr>
<tr>
<td>-103</td>
<td>Invalid separator</td>
<td>Parameter is separated by a character other than a comma.</td>
</tr>
<tr>
<td>-104</td>
<td>Data type error</td>
<td>The wrong data type (number, character, string, expression) was used when specifying the parameter.</td>
</tr>
<tr>
<td>-108</td>
<td>Parameter not allowed</td>
<td>Parameter specified in a command which does not require one.</td>
</tr>
<tr>
<td>-109</td>
<td>Missing parameter</td>
<td>Command requires a parameter(s).</td>
</tr>
<tr>
<td>-112</td>
<td>Program mnemonic too long</td>
<td>Command keyword &gt; 12 characters</td>
</tr>
<tr>
<td>-113</td>
<td>Undefined header</td>
<td>Command header (keyword) was incorrectly specified.</td>
</tr>
<tr>
<td>-121</td>
<td>Invalid character in number</td>
<td>A character other than a comma or number is in the middle of a number.</td>
</tr>
<tr>
<td>-123</td>
<td>Numeric overflow</td>
<td>A parameter value is greater than what can be represented with the number format.</td>
</tr>
<tr>
<td>-124</td>
<td>Too many digits</td>
<td>More than 256 digits were used to specify a number.</td>
</tr>
<tr>
<td>-128</td>
<td>Numeric data not allowed</td>
<td>A number was specified when a letter was required.</td>
</tr>
<tr>
<td>-131</td>
<td>Invalid suffix</td>
<td>Parameter suffix incorrectly specified (e.g. VO rather than VP).</td>
</tr>
<tr>
<td>-138</td>
<td>Suffix not allowed</td>
<td>Parameter suffix is specified when one is not allowed.</td>
</tr>
<tr>
<td>-141</td>
<td>Invalid character data</td>
<td>Discrete parameter specified is not a valid choice.</td>
</tr>
<tr>
<td>-144</td>
<td>Character data too long</td>
<td>A segment or sequence name is too long, or a discrete parameter is &gt; 12 characters. Segment and sequence names must be 12 characters or less.</td>
</tr>
<tr>
<td>-148</td>
<td>Character data not allowed</td>
<td>Discrete parameter was specified when another type (e.g. numeric, boolean) is required.</td>
</tr>
<tr>
<td>-151</td>
<td>Invalid string data</td>
<td>The string data specified (e.g. for the SOUR:MARK:FEED &lt;source&gt; command is not a valid choice.</td>
</tr>
<tr>
<td>-158</td>
<td>String data not allowed</td>
<td>A string was specified when another parameter type (i.e. discrete, numeric, boolean) is required.</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-161</td>
<td>Invalid block data</td>
<td>The number of bytes in a definite length data block does not equal the number of bytes indicated by the block header.</td>
</tr>
<tr>
<td>-168</td>
<td>Block data not allowed</td>
<td>Block data was specified when another parameter type (i.e. discrete, numeric, boolean) is required.</td>
</tr>
<tr>
<td>-178</td>
<td>Expression data not allowed</td>
<td>The parameter was specified as an expression (e.g. SOUR:FREQ1:FIX (A*B)).</td>
</tr>
<tr>
<td>-183</td>
<td>Invalid inside macro definition</td>
<td>Voltage or segment list is inside a macro.</td>
</tr>
<tr>
<td>-211</td>
<td>Trigger ignored</td>
<td>Trigger was received and the AFG was not in the wait-for-trigger state. Or, a trigger was received from a source other than the specified source.</td>
</tr>
<tr>
<td>-212</td>
<td>Arm ignored</td>
<td>Arm was received and the AFG was not in the wait-for-arm state. Or, an arm was received from a source other than the specified source.</td>
</tr>
<tr>
<td>-213</td>
<td>Init ignored</td>
<td>INITiate:IMMediate received while the AFG was currently initiated.</td>
</tr>
<tr>
<td>-221</td>
<td>Settings conflict</td>
<td>See &quot;Settings Conflict Error Messages&quot; at the end of this table.</td>
</tr>
<tr>
<td>-222</td>
<td>Data out of range</td>
<td>Parameter value is out of range for any AFG configuration (e.g. SOUR:FREQ1:FIX 1E9).</td>
</tr>
<tr>
<td>-224</td>
<td>Illegal parameter value</td>
<td>The calibration security code required to disable calibration security does not match the stored code.</td>
</tr>
<tr>
<td>-241</td>
<td>Hardware missing</td>
<td>Command was intended for the HP E1446A which was not present, or is outside the servant area of the HP E1445A AFG.</td>
</tr>
<tr>
<td>-270</td>
<td>Macro error</td>
<td>*RMC &lt;name&gt; was executed and name is not defined.</td>
</tr>
<tr>
<td>-272</td>
<td>Macro execution error</td>
<td>Macro program data sequence could not be executed due to a syntax error within the macro definition.</td>
</tr>
<tr>
<td>-273</td>
<td>Illegal macro label</td>
<td>The macro label defined in the *DMC command was too long, the same as a common command keyword, or contained invalid header syntax.</td>
</tr>
<tr>
<td>-276</td>
<td>Macro recursion error</td>
<td>A macro program data sequence could not be executed because the sequence leads to the execution of a macro being defined.</td>
</tr>
<tr>
<td>-277</td>
<td>Macro redefinition not allowed</td>
<td>A macro label in the *DMC command could not be executed because the macro label was already defined.</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-312</td>
<td>PUD memory lost</td>
<td>The protected user data saved by the *PUD command has been lost.</td>
</tr>
<tr>
<td>-313</td>
<td>Calibration memory lost</td>
<td>The nonvolatile calibration data used by the *CAL command has been lost.</td>
</tr>
<tr>
<td>-330</td>
<td>Self-test failed</td>
<td>Note the information associated with the message for a description of the failure.</td>
</tr>
<tr>
<td>-350</td>
<td>Too many errors</td>
<td>The HP E1445A error queue is full and additional errors have occurred.</td>
</tr>
<tr>
<td>-410</td>
<td>Query INTERRUPTED</td>
<td>The HP E1445A was sent a command before it was finished responding to a query command.</td>
</tr>
<tr>
<td>-420</td>
<td>Query UNTERMINATED</td>
<td>The controller (computer) attempts to read a query response from the HP E1445A without having first sent a complete query command.</td>
</tr>
<tr>
<td>-430</td>
<td>Query DEADLOCKED</td>
<td>The HP E1445A's input and output buffers are full and the AFG cannot continue.</td>
</tr>
<tr>
<td>-440</td>
<td>Query UNTERMINATED after indefinite response</td>
<td>Occurs when the *IDN? query is not the last query executed in a command string.</td>
</tr>
<tr>
<td>+1000</td>
<td>Out of memory</td>
<td>The HP E1445A segment or sequence memory is full.</td>
</tr>
<tr>
<td>+1002</td>
<td>Calibration security enabled</td>
<td>Calibration security must be disabled to calibrate the HP E1445A, to read or write calibration data, to change the security code, or to change the protected user data.</td>
</tr>
<tr>
<td>+1004</td>
<td>Calibration write fail</td>
<td>Writing calibration or protected user data (*PUD) to nonvolatile memory failed.</td>
</tr>
<tr>
<td>+1005</td>
<td>Calibration constant out of range</td>
<td>Illegal calibration constant was computed.</td>
</tr>
<tr>
<td>+1006</td>
<td>Calibration constant conflict</td>
<td>Calibration constants used during calibration set an illegal condition.</td>
</tr>
<tr>
<td>+1007</td>
<td>Calibration security defeated</td>
<td>CALibration secure state disabled and detected at power-on.</td>
</tr>
<tr>
<td>+1011</td>
<td>Illegal while download enabled or testing local bus</td>
<td>Commands such as SOUR:LIST1 ... cannot be executed under current conditions. Execute SOUR:ARB:DOWN:COMP to disable downloading, or VINS:CONF:TEST:DATA?, to complete the local bus test.</td>
</tr>
<tr>
<td>+1012</td>
<td>Illegal when not downloading</td>
<td>SOUR:ARB:DOWN:COMP disables downloading only after it has previously been enabled.</td>
</tr>
<tr>
<td>+1013</td>
<td>Illegal when not testing local bus</td>
<td>VINS:CONF:TEST:DATA? was executed and the local bus test was not performed.</td>
</tr>
<tr>
<td>+1014</td>
<td>Illegal while initiated</td>
<td>Command cannot be executed while the HP E1445A is in the initiated (instrument action) state.</td>
</tr>
</tbody>
</table>
Table B-6. HP E1445A Error Messages (continued)

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1015</td>
<td>Illegal when SOUR:ARB:DAC not INT</td>
<td>SOUR:LIST1 commands cannot be executed unless the DAC data source is internal.</td>
</tr>
<tr>
<td>+1016</td>
<td>Illegal when VIN:LBUS:MODE not CONS</td>
<td>The operating mode for the local bus is &quot;off&quot; and SOUR:ARB:DOWN is set to LBUS.</td>
</tr>
<tr>
<td>+1017</td>
<td>Illegal when SOUR:FUNC:SHAP RAMP</td>
<td>SQU</td>
</tr>
<tr>
<td>+1018</td>
<td>Illegal while calibrating</td>
<td>Commands cannot be sent to the HP E1445A while the device is calibrating.</td>
</tr>
<tr>
<td>+1019</td>
<td>Illegal while not calibrating</td>
<td>The command is only valid while the HP E1445A is calibrating.</td>
</tr>
<tr>
<td>+1020</td>
<td>Illegal while initiated and SOUR:PM:SOUR not INT</td>
<td>Frequency changes during phase modulation can only occur when SOUR:PM:SOUR is INTernal.</td>
</tr>
<tr>
<td>+1021</td>
<td>Test data byte count not even number</td>
<td>The length parameter for the command VINS:CONF:TEST:CONF is not an even number.</td>
</tr>
<tr>
<td>+1022</td>
<td>VXI data transfer bus not active</td>
<td>VINS:CONF:VME:REC:ADDR:DATA? is executed and A24 address space is not being written to.</td>
</tr>
<tr>
<td>+1100</td>
<td>Illegal segment name</td>
<td>Attempting to download to a segment that doesn’t exist, or selecting a segment name that’s the same as an existing sequence name.</td>
</tr>
<tr>
<td>+1101</td>
<td>Too many segment names</td>
<td>There are &gt;256 segment names defined. Use SOUR:LIST1:SEGM:DEL:SEL to delete the current (selected) segment, or SOUR:LIST1:SEGM:DEL:ALL to delete all segments.</td>
</tr>
<tr>
<td>+1102</td>
<td>Segment in use</td>
<td>Trying to delete a segment that is within a sequence.</td>
</tr>
<tr>
<td>+1103</td>
<td>Segments exist</td>
<td>Trying to change the data format of a segment that already exists.</td>
</tr>
<tr>
<td>+1104</td>
<td>Segment lists of different lengths</td>
<td>The length of a segment’s voltage list does not equal the length of its marker list and its marker list does not equal 1.</td>
</tr>
<tr>
<td>+1105</td>
<td>Segment list has zero length</td>
<td>Querying a voltage, marker, or dac code list that has no data.</td>
</tr>
<tr>
<td>+1106</td>
<td>Segment name not DEFined</td>
<td>Trying to load segment memory and memory has not been reserved by the SOUR:LIST1:SEGM:DEF command.</td>
</tr>
<tr>
<td>+1107</td>
<td>Segment name already defined</td>
<td>Defining a segment and a segment by that name already exists.</td>
</tr>
<tr>
<td>+1108</td>
<td>No segment name SELected</td>
<td>Trying to load a segment that has not been selected.</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>+1109</td>
<td>Segment list length less than minimum</td>
<td>Waveform segment has less than four points.</td>
</tr>
<tr>
<td>+1110</td>
<td>Illegal sequence name</td>
<td>Attempting to download to a sequence that doesn’t exist, or selecting a sequence name that’s the same as an existing segment name.</td>
</tr>
<tr>
<td>+1111</td>
<td>Too many sequence names</td>
<td>There are &gt;256 sequence names defined. Use SOUR:LIST1:SSEQ:DEL:SEL to delete the current (selected) sequence, or SOUR:LIST1:SSEQ:DEL:ALL to delete all sequences.</td>
</tr>
<tr>
<td>+1112</td>
<td>Sequence in use</td>
<td>Trying to delete a sequence currently selected by SOUR:FUNC:USER.</td>
</tr>
<tr>
<td>+1113</td>
<td>Sequence contains zero-length segment</td>
<td>Segment contains no voltage or dac code data.</td>
</tr>
<tr>
<td>+1114</td>
<td>Sequence lists of different lengths</td>
<td>The length of a sequence’s segment list does not equal the length of its marker list and its marker list does not equal 1.</td>
</tr>
<tr>
<td>+1115</td>
<td>Sequence list has zero length</td>
<td>Query of a marker list, dwell count list, or sequence segment list and no data is in the list. Also occurs following INIT:IMM or SOUR:FUNC:USER when no segments are in the sequence list.</td>
</tr>
<tr>
<td>+1116</td>
<td>Sequence name not DEFined</td>
<td>Trying to define an ordered sequence of waveform segments and sequence memory has not been reserved with the SOUR:LIST1:SSEQ:DEF command.</td>
</tr>
<tr>
<td>+1117</td>
<td>Sequence name already defined</td>
<td>Defining a sequence and a sequence by that name already exists.</td>
</tr>
<tr>
<td>+1118</td>
<td>No sequence name SELECTed</td>
<td>SOUR:LIST1:SSEQ subsystem command executed without a segment sequence first selected by SOUR:LIST1:SSEQ:SEL.</td>
</tr>
<tr>
<td>+1121</td>
<td>Frequency list has zero length</td>
<td>SOUR:FREQ1:MODE LIST is set and no frequency list exists.</td>
</tr>
<tr>
<td>+1122</td>
<td>Frequency list length less than minimum</td>
<td>Frequency list has less than two frequencies.</td>
</tr>
</tbody>
</table>
## Settings Conflict Error Messages

<table>
<thead>
<tr>
<th>Settings Conflict Error Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOUR:FREQ1:FIX frequency &lt; minimum; SOUR:FREQ1:FIX MIN set</td>
</tr>
<tr>
<td>SOUR:FREQ1:FIX frequency &gt; maximum; SOUR:FREQ1:FIX MAX set</td>
</tr>
<tr>
<td>SOUR:FREQ2:FIX frequency &lt; minimum; SOUR:FREQ2:FIX MIN set</td>
</tr>
<tr>
<td>SOUR:FREQ2:FIX frequency &gt; maximum; SOUR:FREQ2:FIX MAX set</td>
</tr>
<tr>
<td>SOUR:FREQ1:RANG frequency &gt; maximum; SOUR:FREQ1:RANG MAX set</td>
</tr>
<tr>
<td>SOUR:FREQ1:FSK frequency &lt; minimum; SOUR:FREQ1:FSK MIN set</td>
</tr>
<tr>
<td>SOUR:FREQ1:FSK frequency &gt; maximum; SOUR: FREQ1:FSK MAX set</td>
</tr>
<tr>
<td>TRIG:STAR:SOUR and SOUR:ROSC:SOUR both EXT; SOUR:ROSC:SOUR INT1 set</td>
</tr>
<tr>
<td>TRIG:STAR:SOUR and TRIG:STOP:SOUR both BUS; TRIG:STOP:SOUR HOLD set</td>
</tr>
<tr>
<td>OUTP:LOAD not equal to OUTP:IMP or INF; OUTP:LOAD set to OUTP:IMP value</td>
</tr>
<tr>
<td>SOUR:FUNC:SHAP DC and INIT; INIT ignored</td>
</tr>
<tr>
<td>SOUR:ARB:DAC:SOUR not INT and INIT; INIT ignored</td>
</tr>
<tr>
<td>Frequency list value out of range; SOUR:FREQ1:MODE FIX set</td>
</tr>
<tr>
<td>SOUR:FREQ1:MODE LIST and no frequency list defined; SOUR:FREQ1:MODE FIX set</td>
</tr>
<tr>
<td>SOUR:VOLT+ SOUR:VOLT:OFFS &lt; minimum; SOUR:VOLT:OFFS MIN set</td>
</tr>
<tr>
<td>SOUR:VOLT+ SOUR:VOLT:OFFS &gt; maximum; SOUR:VOLT:OFFS MAX set</td>
</tr>
<tr>
<td>SOUR2:VOLT:OFFS &lt; minimum; SOUR2:VOLT:OFFS MIN set</td>
</tr>
<tr>
<td>Settings Conflict Error Messages</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>SOUR2:VOLT:OFFS &gt; maximum; SOUR2:VOLT:OFFS MAX set</td>
</tr>
<tr>
<td>SOUR:FUNC:SHAP SIN and TRIG:STAR:SOUR not INT1; TRIG:STAR:SOUR INT1 set</td>
</tr>
<tr>
<td>SOUR:FREQ1:START &gt; SOUR:FREQ1:STOP; values exchanged</td>
</tr>
<tr>
<td>SOUR:FREQ1:STAR frequency &lt; minimum; SOUR:FREQ1:STAR MIN set</td>
</tr>
<tr>
<td>SOUR:FREQ1:STAR frequency &gt; maximum; SOUR:FREQ1:STAR MAX set</td>
</tr>
<tr>
<td>SOUR:FREQ1:STOP frequency &lt; minimum; SOUR:FREQ1:STOP MIN set</td>
</tr>
<tr>
<td>SOUR:FREQ1:STOP frequency &gt; maximum; SOUR:FREQ1:STOP MAX set</td>
</tr>
<tr>
<td>ARM:SWE:SOUR TTLT&lt;n&gt; and TRIG:SWE:SOUR TTLT&lt;n&gt;; ARM:SWE:SOUR IMM set</td>
</tr>
<tr>
<td>SWE:TIME &lt; minimum; SWE:TIME MIN set</td>
</tr>
<tr>
<td>SWE:TIME &gt; maximum; SWE:TIME MAX set</td>
</tr>
<tr>
<td>TRIG:SWE:TIM &lt; minimum; TRIG:SWE:TIM MIN set</td>
</tr>
<tr>
<td>TRIG:SWE:TIM &gt; maximum; TRIG:SWE:TIM MAX set</td>
</tr>
<tr>
<td>SOUR:FUNC:SHAP not SIN and SOUR:PM:STAT ON; SOUR:PM:STAT OFF set</td>
</tr>
<tr>
<td>SOUR:VOLT voltage &lt; minimum; SOUR:VOLT MIN set</td>
</tr>
<tr>
<td>SOUR:VOLT voltage &gt; maximum; SOUR:VOLT MAX set</td>
</tr>
<tr>
<td>SOUR:FUNC:SHAP not DC and SOUR:VOLT voltage &lt; 0.0V: absolute value of SOUR:VOLT set</td>
</tr>
<tr>
<td>OUTP:LOAD INF and current SOUR:VOLT unit W, DBM, OR DBMW: SOUR:VOLT:AMPL MIN (in V) set</td>
</tr>
<tr>
<td>SOUR:FUNC:SHAP DC and current SOUR:VOLT unit not V: SOUR:VOLT value converted to volts</td>
</tr>
<tr>
<td>Settings Conflict Error Messages (when HP E1445A is used with the HP E1446A Amplifier)</td>
</tr>
<tr>
<td>------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OUTP2:ATT 20 and OUTP2:IMP 0; OUTP2:IMP 50 set</td>
</tr>
<tr>
<td>SOUR2:VOLT:OFFS &lt; minimum; SOUR2:VOLT:OFFS MIN set</td>
</tr>
<tr>
<td>SOUR2:VOLT:OFFS &gt; maximum; SOUR2:VOLT:OFFS MAX set</td>
</tr>
</tbody>
</table>
Appendix C

Register-Based Programming

Appendix Contents

The HP E1445A Arbitrary Function Generator (AFG) is a message-based device. As such, it supports the VXI word-serial protocol used to transfer ASCII command strings and is capable of converting the SCPI commands it receives to reads and writes of its hardware registers.

Register-based programming allows the user to access the hardware registers directly. This increases the speed at which events in the AFG occur since the parsing (converting) of SCPI commands is eliminated. In addition to describing how to access selected AFG registers, this appendix explains how to do the following functions with register reads and writes:

- Accessing the Registers . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 484
  - Determining the A24 Base Address . . . . . . . . . . . . . . . . . . . . Page 484

- Changing the Output Frequency . . . . . . . . . . . . . . . . . . . . . Page 487
  - The Frequency Control Registers . . . . . . . . . . . . . . . . . . . . . Page 487
  - Frequency Control Programs . . . . . . . . . . . . . . . . . . . . . . . . . Page 489

- Changing the Signal Phase. . . . . . . . . . . . . . . . . . . . . . . . . . Page 495
  - The Phase Control Registers . . . . . . . . . . . . . . . . . . . . . . . . . Page 495
  - Phase Control Program . . . . . . . . . . . . . . . . . . . . . . . . . . . . Page 496

- Selecting the Waveform Sequence . . . . . . . . . . . . . . . . . . . . Page 498
  - The Waveform Sequence Registers . . . . . . . . . . . . . . . . . . . . . Page 498
  - Waveform Sequence Selection Program . . . . . . . . . . . . . . . . . Page 500

- Loading the DAC from the VXIbus . . . . . . . . . . . . . . . . . . . . . Page 506

This appendix does not identify all of the AFG registers nor does it cover all of the AFG functions from the register-based programming standpoint.
The example programs and programming techniques shown in this appendix are based on the following system configuration:

**Mainframe:** HP 75000 Series C (HP E1401)

**Controller:** HP E1480A V/360 (select code 16)

**Programming Language:** HP BASIC/UX (6.0)

**HP E1445A AFG:** Logical address = 80

Each program uses a combination of SCPI commands and register reads/writes. In most cases, SCPI commands set up the AFG and initiate the waveform. Register reads/writes are used to change the frequency, phase, waveform, etc., instantaneously.

**Accessing the Registers**

Access to the AFG’s operational registers is through addresses mapped into A24 address space. At power-on, the system resource manager reads the AFG’s Device Type Register (in A16 address space) to determine the amount of A24 memory the AFG requires. Once known, the resource manager allocates a block of A24 memory to the AFG and writes the base (starting) address into the AFG’s Offset Register.

**Determining the A24 Base Address**

When you are reading or writing to an AFG register, a hexadecimal or decimal register address is specified. The register address is the sum of:

\[
\text{A24 base address} + \text{register number}
\]

The base address of the AFG operational registers in A24 address space is determined by reading the AFG’s Offset Register and multiplying the value by 256 (10016). This converts the 16-bit value of the Offset Register to a 24-bit address.

The register number is identified in the register descriptions found in the following sections.
Reading the AFG's Offset Register

As shown in Figure C-1, the AFG's configuration registers are mapped into the upper 25% of A16 address space. The Offset Register is one of the AFG's configuration registers.

*Base Address = CO0016 + (Logical Address * 64)16
or
49,152 + (Logical Address * 64)10

Register Address = Base address + Register Offset

Figure C-1. HP E1445A AFG Registers within A16 Address Space
In a system using a V/360 (HP E1480) controller, for example, the base address of the configuration registers is computed as:

\[ \text{C000}_{16} + (\text{LADDR} \times 64)_{16} \]

or

\[ 49,152 + (\text{LADDR} \times 64) \]

where \( \text{C000}_{16} \) (49,152) is the starting location of the register addresses, LADDR is the AFG’s logical address, and 64 is the number of address bytes in A16 per VXI device.

The AFG’s factory set logical address is 80. If this address is not changed, the base address of the AFG’s configuration registers in A16 is:

\[ \text{C000}_{16} + (80 \times 64)_{16} \]

\[ \text{C000}_{16} + 1400_{16} = \text{D400}_{16} \]

or (decimal)

\[ 49,152 + (80 \times 64) \]

\[ 49,152 + 5120 = 54,272 \]

Given the base address and number of the Offset Register (06 in Figure C-1), the base address of the operational registers in A24 can be determined as in the following program.

### Reading the Offset Register

```
10 ASSIGN @Afg to 1680  !Path from V/360 to AFG via VXI backplane
20 COM @Afg,Base_addr
30 CALL A24_offset
40 END
50 !
60 SUB A24_offset
70 A24_offset: !Subprogram which determines the base address for
80 !the AFG registers in A24 address space.
90  COM @Afg,Base_addr
100 CONTROL 16,25;2  !access A16 space with READIO and WRITEIO
110 A16_addr=DVAL("D400",16) !convert A16 base address to decimal number
120 Offset=READIO(-16,A16_addr+6) !read AFG offset register
130 Base_addr=Offset*256 !multiply offset for 24-bit address
140 SUBEND
```

As mentioned, multiplying the value of the Offset Register by 256 (or \( 100_{16} \)) converts the 16-bit register value to a 24-bit address.
Changing the Output Frequency

This section explains how the frequency of the output signal is changed instantaneously by writing frequency codes to the appropriate registers. The section shows how to change the frequency when either the direct-digital-synthesis ([SOURce:]FREQuency[1]) or divide-by-n ([SOURce:]FREQuency2) frequency synthesis method is used.

The Frequency Control Registers

The following Frequency Control Registers are used to change the output frequency generated with the direct-digital-synthesis (DDS) and divide-by-n methods:

- Phase Increment Registers (DDS):
  \[ \text{base}_\text{addr} + A7_{16} \text{ through } \text{base}_\text{addr} + A1_{16} \]
- Frequency Load Strobe Register (DDS):
  \[ \text{base}_\text{addr} + 8D_{16} \]
- Sample/Hold and ROSC/N Control Register (DIV N):
  \[ \text{base}_\text{addr} + 63_{16} \]
- ROSC/N Divider Registers (DIV N):
  \[ \text{base}_\text{addr} + 7D_{16} \text{ through } \text{base}_\text{addr} + 7F_{16} \]

The Phase Increment Registers

Phase Increment Registers A7, A5, A3, and A1 contain the 32-bit phase increment data that is written to the DDS micro-chip. The phase increment value determines the output frequency.

<table>
<thead>
<tr>
<th>Address</th>
<th>15–8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>base + A7_{16} through</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>base + A1_{16}</td>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Register A7:** Bits 31–24 of the phase increment value. These are the most significant bits (MSBs).

**Register A5:** Bits 23–16 of the phase increment value.

**Register A3:** Bits 15–8 of the phase increment value.

**Register A1:** Bits 7–0 of the phase increment value. These are the least significant bits (LSBs).
The **Frequency Load Strobe Register**

Writing any value to the Frequency Load Strobe Register loads the contents of the Phase Increment Registers into the DDS micro-chip.

<table>
<thead>
<tr>
<th>Address</th>
<th>15–8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>base + 8D₁₆</td>
<td>unused</td>
<td>Strobe Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Strobe Data:** Writing any value to this register loads the contents of the Phase Increment Registers into the DDS micro-chip. Once the data has been loaded, it takes 20 reference oscillator clock cycles for the new frequency to appear at the output.

The **Sample/Hold and ROSC/N Control Register**

The Sample/Hold and ROSC/N Control Register enables and disables signal sampling, and specifies the N value used to generate ROSC/N frequencies.

<table>
<thead>
<tr>
<th>Address</th>
<th>15–8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>base + 63₁₆</td>
<td>unused</td>
<td>SHOLD</td>
<td>SMUX2</td>
<td>SMUX1</td>
<td>SMUX0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SHOLD:** Setting bit 7 to a ’1’ causes sample signals to be ignored. This bit is set while setting the divide-by-n counter.

**SMUX2–SMUX0:** bits 2–0 select N as follows:

- 0 0 0 = selects ROSC/1
- 0 0 1 = selects ROSC/2
- 0 1 0 = selects ROSC/3
- 0 1 1 = selects ROSC/2N

The **ROSC/N Divider Registers**

The ROSC/N Divider Registers contain the value of (N-1) when 2N is greater than or equal to 4. The reference oscillator (ROSC) will be divided by 4 through 131,072 when the registers are loaded with 1 through 65,535.

<table>
<thead>
<tr>
<th>Address</th>
<th>15–8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>base + 7D₁₆ through base + 7F₁₆</td>
<td>unused</td>
<td>value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Register 7D:** Contains the most significant byte of the value of (N-1).

**Register 7F:** Contains the least significant byte of the value of (N-1).
**Frequency Control Programs**

The following programs demonstrate how to change the signal frequency while the waveform is currently at the AFG output.

**DDS Frequency Control**

The FREQ_REG program changes the signal frequency that is generated using the DDS ([SOURce:]FREQ[1]) subsystem and the reference oscillator from any of the available sources. The program accesses the Phase Increment and Frequency Load Strobe Registers.

**HP BASIC Program Example (FREQ1_REG)**

```bASIC
1 !RE-STORE "FREQ1_REG"
2 !This program changes the output frequency generated by the direct-
3 !digital-synthesis (DDS) method by writing frequency-value data to
4 !the AFG's Phase Increment registers.
5 !
10 ASSIGN @Afg TO 1680
20 COM @Afg,Base_addr
30 !
40 !Call the subprograms which reset the AFG, which determine the base
50 !address of the AFG registers in A24 address space, and which set the
60 !output function.
70 CALL Rst
80 CALL A24_offset
90 CALL Output_function
100 !
110 DISP "Press 'Continue' to change frequency (register writes)"
120 PAUSE
130 DISP ""
140 !Call the subprogram which changes the output frequency, and pass the
150 !frequency, the number of waveform points, the reference oscillator
160 !frequency, and the frequency range (SOUR:FREQ1:RANGe command).
170 !(Note: sine waves and arb waves: npts=1, square waves: npts=4,
180 !ramp/triangle waves: npts=RAMP:POINts value.)
190 !
200 CALL Freq_change(2000.,1,4.294967296E+7,0)
210 END
220 !
230 SUB A24_offset
240 A24_offset: !Subprogram which determines the base address for
250 !the AFG registers in A24 address space.
260 !COM @Afg,Base_addr
270 !CONTROL 16,25:2!access A16 space with READIO and WRITEIO
280 A16_addr=DVAL("D400",16) !AFG A16 base address
290 Offset=READIO(-16,A16_addr+6) !read AFG offset register
300 Base_addr=Offset*256 !shift offset for 24-bit address
310 SUBEND
320 !
330 SUB Output_function
```

Continued on Next Page
Output_function: !Subprogram which uses SCPI commands to set the
42.94967296 MHz reference oscillator, to set DDS
frequency synthesis, to set the output frequency/
function/amplitude, and to start the waveform.

COM @Afg,Base_addr
OUTPUT @Afg;"SOUR:ROSC:SOUR INT1;"; !reference oscillator (42 MHz)
OUTPUT @Afg;":TRIG:STAR:SOUR INT1;"; !frequency1 generator
OUTPUT @Afg;":SOUR:FREQ1:FIX 1E3;"; !frequency
OUTPUT @Afg;":SOUR:FUNC:SHAP SIN;"; !function
OUTPUT @Afg;":SOUR:VOLT:LEV:IMM:AMPL 5V" !amplitude
OUTPUT @Afg;"INIT:IMM" !wait_for_arm state
OUTPUT @Afg;"STAT:OPC:INIT OFF;*OPC?" !wait for INIT to complete
ENTER @Afg;Complete

SUBEND

SUB Freq_change(Freq,Npts,Reference_osc,Range)
Freq_change: !Subprogram which changes the output frequency by writing
the frequency to registers on the AFG.

COM @Afg,Base_addr
CONTROL 16,25;3!access A24 space with READIO and WRITEIO

!Calculate frequency value written to registers
IF Range>0 THEN
  Phase$=DVAL$((Freq*Npts/Reference_osc/2)*4.294967296E+9,16)
ELSE
  Phase$=DVAL$((Freq*Npts/Reference_osc)*4.294967296E+9,16)
END IF

!Write the first byte of the frequency value to register A7, the
second byte to register A5, the third byte to register A3, and the
fourth byte to register A1.
WRITEIO -16,Base_addr+IVAL("A7",16);IVAL(Phase$[1;2],16)
WRITEIO -16,Base_addr+IVAL("A5",16);IVAL(Phase$[3;2],16)
WRITEIO -16,Base_addr+IVAL("A3",16);IVAL(Phase$[5;2],16)
WRITEIO -16,Base_addr+IVAL("A1",16);IVAL(Phase$[7;2],16)

!Generate the pulse which loads the new frequency. Once the pulse is
received, it takes 20 reference oscillator clock cycles before the
new frequency appears at the output.
WRITEIO -16,Base_addr+IVAL("8D",16);0

SUBEND

SUB Rst
Rst: !Subprogram which resets the E1445.

COM @Afg,Base_addr
OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
ENTER @Afg;Complete

SUBEND
Comments

• To simplify the program, SCPI commands are included to select the reference oscillator, the DDS subsystem, and to start the waveform. This requires that the only registers written to be the Phase Increment and Frequency Load Strobe Registers. This program executes as intended when the SCPI commands in subprogram Output_function are executed before the registers are written to.

• The subprogram Output_function sets the initial reference oscillator frequency to 42.94967296 MHz. If a different reference oscillator frequency is used (that is, 40 MHz or an externally supplied oscillator), specify that frequency when the Freq_change subprogram is called (line 200).

• If frequency doubling is in effect (SOUR:FREQ1:RANG command in subprogram Output_function), the doubled frequency can be changed to another doubled frequency by passing a number other than 0 as the fourth parameter to the Freq_change subprogram (line 200).

• Note the following when specifying the number of waveform points (Npts value passed to the Freq_change subprogram): sine waves and arbitrary waveforms, Npts = 1; square waves, Npts = 4; ramp and triangle waves, Npts = RAMP:POINts value.

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, FREQ1REG.FRM, is in directory “VBPROG” and the Visual C example program, FREQ1REG.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Divide-by-N Frequency Control

The FREQ2_REG program changes the signal frequency that is generated using the Divide-by-N ([SOURCE:FREQuency2]) subsystem and the reference oscillator from any of the available sources. The program accesses the Sample/Hold and ROSC/N Control Register, and the ROSC/N Divider Registers.

HP BASIC Program Example (FREQ2_REG)

1 !RE-STORE "FREQ2_REG"
2 !This program changes the output frequency generated with the divide-by-n frequency synthesis method by writing frequency data to the Sample/Hold and ROSC/N Control register, and to the ROSC/N Divider registers.
6 !
10 ASSIGN @Afg TO 1680
20 COM @Afg,Base_addr
30 !
40 !Call the subprograms which reset the AFG, which determine the base address of the AFG registers in A24 address space, and which set the output function.
70 CALL Rst
80 CALL A24_offset
90 CALL Output_function
100 !
110 DISP "Press 'Continue' to change frequency (register writes)"
120 PAUSE
130 DISP ""
140 !Call the subprogram which changes the output frequency. Pass the reference oscillator frequency, the new output frequency, and the number of waveform points. (Note arbitrary waveforms: npts=1 square waves: npts=4, ramp/triangle waves: npts=RAMP:POINts value.)
170 !
190 !
200 CALL Divide_by_n(4.E+7,2.5E+6,4)
210 END
210 !
230 SUB A24_offset
240 A24_offset: !Subprogram which determines the base address for the AFG registers in A24 address space.
250 !
260 COM @Afg,Base_addr
270 CONTROL 16,25,2 !access A16 space with READIO and WRITEIO
280 A16_addr=DVAL("D400",16) !AFG A16 base address
290 Offset=READIO(-16,A16_addr+6) !read AFG offset register
300 Base_addr=Offset*256 !shift offset for 24-bit address
310 SUBEND
320 !
330 SUB Output_function

Continued on Next Page
340  Output_function:  !Subprogram which uses SCPI commands to set the
350  !40 MHz reference oscillator, to set divide-by-n
360  !frequency synthesis, to set the output frequency/
370  !function/amplitude, and to start the waveform.
380  COM @Afg,Base_addr
390  OUTPUT @Afg;"SOUR:ROSC:SOUR INT2;"; !reference oscillator (40 MHz)
400  OUTPUT @Afg;"::TRIG:STAR:SOUR INT2;"; !frequency generator
410  OUTPUT @Afg;"::SOUR:FREQ2:FIX 1E6;"; !frequency
420  OUTPUT @Afg;"::SOUR:FUNC:SHAP SQU;"; !function
430  OUTPUT @Afg;"::SOUR:VOLT:LEV:IMM:AMPL 5V" !amplitude
440  OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
450  OUTPUT @Afg;"STAT:OPC:INIT OFF;"OPC?" !wait for INIT to complete
460  ENTER @Afg;Complete
470  SUBEND
480  !
490  SUB Divide_by_n(Reference_osc,Frequency,Points)
500  Divide_by_n:  !Subprogram which changes the output frequency by writing
510  !to the register which controls divide-by-n frequency
520  !synthesis.
530  COM @Afg,Base_addr
540  INTEGER Divider
550  CONTROL 16,25;3 !access A24 space with READIO and WRITEIO
560  !
570  !Read register 63. Write to register 63 setting the SHOLD bit (bit 7)
580  !so sample signals are ignored.
590  Sample_hold=READIO(-16,Base_addr+IVAL("63",16))
600  Sample_hold=BINIOR(Sample_hold,128) !set bit 7
610  WRITEIO -16,Base_addr+IVAL("63",16);Sample_hold
620  !
630  !Set the reference oscillator divider based on the new frequency.
640  !Also load the new divider value if n is greater than 3.
650  Divider=Reference_osc/Frequency/Points
660  SELECT Divider
670  CASE 1
680  Sample_hold=BINAND(Sample_hold,248)+0
690  CASE 2
700  Sample_hold=BINAND(Sample_hold,248)+1
710  CASE 3
720  Sample_hold=BINAND(Sample_hold,248)+2
730  CASE ELSE
740  Sample_hold=BINAND(Sample_hold,248)+3
750  Divider=Divider/2-1
760  WRITEIO -16,Base_addr+IVAL("7D",16);SHIFT(Divider,8)
770  WRITEIO -16,Base_addr+IVAL("7F",16);BINAND(Divider,255)
780  END SELECT
790  !
800  WRITEIO -16,Base_addr+IVAL("63",16);Sample_hold
810  !
820  !Clear sample/hold bit which activates new frequency

Continued on Next Page
830     WRITEIO -16,Base_addr+IVAL("63","16");BINAND(Sample_hold,127)
840     \n
850     SUBEND
860     !
870     Rst:   !Subprogram which resets the E1445.
880     COM @Afg,Base_addr
890     OUTPUT @Afg;"*RST;*OPC?"       !reset the AFG
900     ENTER @Afg;Complete
910     SUBEND

Comments

- To simplify the program, SCPI commands are included to select the reference oscillator, the divide-by-n subsystem and to start the waveform. This requires that the only registers written to be the Sample/Hold and ROSC/N Control Register, and the ROSC/N Divider Registers. This program executes as intended when the SCPI commands in subprogram Output_function are executed before the registers are written to.

- The subprogram Output_function sets the initial reference oscillator frequency to 40 MHz. If a different reference oscillator frequency is used (that is, 42.94967296 MHz or an externally supplied oscillator), specify that frequency when the Divide_by_n subprogram is called (line 200).

- Standard function sine waves are not available with the divide-by-n subsystem ([SOURce:]FREQuency2). Frequency doubling should not be used with the divide-by-n subsystem.

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, FREQ2REG.FRМ, is in directory “VBPROG” and the Visual C example program, FREQ2REG.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Changing the Signal Phase

This section explains how the phase of a sine wave generated by the DDS ([SOURce:FREQuency[1])] subsystem is changed by writing phase data to the Phase Modulation Registers.

The Phase Control Registers

The following phase control registers are used to change the phase of the sine wave generated by the DDS subsystem:

- **Phase Modulation Registers:**
  
  base_addr + B3\textsubscript{16} through base_addr + B1\textsubscript{16}

- **Phase Load Strobe Register:**
  
  base_addr + 8B\textsubscript{16}

The Phase Modulation Registers

Phase Modulation Registers B3 and B1 contain the 12-bit phase modulation data that is added to the output of the phase accumulator.

<table>
<thead>
<tr>
<th>Address</th>
<th>15–8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>base + B3\textsubscript{16} through base + B1\textsubscript{16}</td>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Phase modulation value</td>
</tr>
</tbody>
</table>

**Register B3:** This register contains the eight most significant bits of the 12-bit phase modulation value (bits 11–4).

**Register B1:** Bits 7–4 of this register are the four least significant bits of the 12-bit phase modulation value (bits 3–0). Bits 3–0 of register B1 are ignored.

The Phase Load Strobe Register

Writing any value to the Phase Load Strobe Register adds the data in the Phase Modulation Registers to the output of the phase accumulator.

<table>
<thead>
<tr>
<th>Address</th>
<th>15–8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>base + 8B\textsubscript{16}</td>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Strobe Data</td>
</tr>
</tbody>
</table>

**Strobe Data:** Writing any value to this register adds the data in the Phase Modulation Registers to the output of the phase accumulator. Once the phase has been added, it takes 14 reference oscillator clock cycles for the new phase to appear at the output.
Phase Control Program

The PHAS_CHNG program demonstrates how to change the sine wave signal phase while the waveform is currently at the AFG output.

**HP BASIC Program Example (PHAS_CHNG)**

```hpbasic
1 !RE-STORE "PHAS_CHNG"
2 !This program changes the phase of the output signal by writing
3 !phase offset data to the phase modulation registers.
4 !
10 ASSIGN @Afg TO 1680
20 COM @Afg,Base_addr
30 !
40 !Call the subprograms which reset the AFG, which determine the base
50 !address of the AFG registers in A24 address space, and which set the
60 !output function.
70 CALL Rst
80 CALL A24_offset
90 CALL Output_function
100 !
110 DISP "Press 'Continue' to change the signal phase (register writes)"
120 PAUSE
130 DISP ""
140 !Call the subprogram which changes the signal phase, and pass the new
150 !phase value.
160 !
170 CALL Phase_change(180)
180 END
190 !
200 SUB A24_offset
210 A24_offset: !Subprogram which determines the base address for
220 !the AFG registers in A24 address space.
230 COM @Afg,Base_addr
240 CONTROL 16,25:2 !access A16 space with READIO and WRITEIO
250 A16_addr=DVAL("D400",16) !AFG A16 base address
260 Offset=READIO(-16,A16_addr+6) !read AFG offset register
270 Base_addr=Offset*256 !shift offset for 24-bit address
280 SUBEND
290 !
300 SUB Output_function
310 Output_function: !Subprogram which uses SCPI commands to set DDS
320 !frequency synthesis, to set the output frequency/
330 !function/amplitude, to set up phase modulation, and
340 !to start the waveform.
350 COM @Afg,Base_addr
360 OUTPUT @Afg;"TRIG:STAR:SOUR INT1;"; !frequency generator
370 OUTPUT @Afg;"SOUR:FREQ1:FIX 60;"; !frequency
380 OUTPUT @Afg;"SOUR:PM:SOUR INT;"; !phase modulation source
390 OUTPUT @Afg;"SOUR:PM:STAT ON;"; !enable phase modulation
400 OUTPUT @Afg;"SOUR:FUNC:SHAP SIN;"; !function
```

Continued on Next Page
Phase change(Phase)

Phase_change: Subprogram which changes the phase of the output signal by writing phase data to the registers on the AFG.

Phase1 = Phase MOD 360
IF Phase1 < 0 THEN Phase1 = Phase1 + 360
Phase_data$ = IVAL$(4096 * (16 * Phase1 / 360) - 65536 * (Phase1 >= 180), 16)
WRITEIO -16,Base_addr+IVAL("B3",16);IVAL(Phase_data$[1,2],16)
WRITEIO -16,Base_addr+IVAL("B1",16);IVAL(Phase_data$[3,2],16)

Generate pulse which loads the new phase. Once the pulse is received, it takes 14 reference oscillator clock cycles before the new phase appears at the output.

Comments

To simplify the program, SCPI commands are included to configure the AFG, enable phase modulation, and start the waveform. Thus, the only registers written to are the Phase Modulation and Phase Load Strobe Registers. This program executes as intended when the SCPI commands in subprogram Output_function are executed before the registers are written to.

Phase modulation is only available with standard function sine waves. Standard function sine waves are only available with the DDS ([SOURce:FRQency[1]) subsystem.

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, PHASCHNG.FRM, is in directory “VBPROG” and the Visual C example program, PHASCHNG.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Selecting the Waveform Sequence

This section shows how to select and output an arbitrary waveform without aborting the current waveform and re-initializing the AFG.

The Waveform Sequence Registers

The following Waveform Sequence Registers are used to change the output waveform sequence:

- Traffic Register:
  \[
  \text{base}_\text{addr} + 8_{16}
  \]

- Waveform Select Register:
  \[
  \text{base}_\text{addr} + A_{16}
  \]

- Sequence Base Register:
  \[
  \text{base}_\text{addr} + 20_{16}
  \]

- Status Register:
  \[
  \text{base}_\text{addr} + 2_{16}
  \]

The Traffic Register

The Traffic Register specifies the source which selects the waveform sequence.

<table>
<thead>
<tr>
<th>Address</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7–0</th>
</tr>
</thead>
<tbody>
<tr>
<td>base + 8_{16}</td>
<td>Sequencer data source</td>
<td>High-speed clock source</td>
<td>High-speed data source</td>
<td>other control bits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Sequencer Data Source**: The Sequencer data source field specifies the source which selects addresses in sequence base memory that, in turn, select the waveform sequences. The available sources are:

- 00 = sequencer data source is Local Bus
- 01 = sequencer data source is Front Panel
- 10 = sequencer data source is Waveform Select Register

The source specified in the Sequence Selection program is the Waveform Select Register. Note that when the **sequencer data source** is specified, the contents of the other register fields must remain unchanged.
The Waveform Select Register

The Waveform Select Register contains the location of the output sequence’s base address in sequence base memory.

<table>
<thead>
<tr>
<th>Address</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7–0</th>
</tr>
</thead>
<tbody>
<tr>
<td>base + A16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Waveform Index</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>other control bits</td>
</tr>
</tbody>
</table>

**Waveform Index:** The Waveform Index is the location in sequence base memory where the base address of the sequence in sequence memory is located.

When specifying a waveform index it is recommended that you begin with an index of 1, then 2, and so on. Index 0 is reserved for SCPI usage.

The Sequence Base Register

The Sequence Base Register contains the base address of the selected sequence in sequence memory.

<table>
<thead>
<tr>
<th>Address</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>base + 2016</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Sequence Base Address</td>
</tr>
</tbody>
</table>

**Sequence Base Address:** The Sequence Base Address is the location of the sequence in sequence memory.

The Status Register

The Status Register is used to determine when a new waveform can be selected from sequence base memory.

<table>
<thead>
<tr>
<th>Address</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>base + 216</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>other status bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>WFUSED</td>
</tr>
</tbody>
</table>

**WFUSED:** A ‘0’ to ‘1’ transition of this bit indicates that a new waveform can be selected from sequence base memory. This bit is cleared by reading, and then writing to the Waveform Select Register.
**Sequence Selection Program**

The WAVE_SEL program shows how to change the output waveform (sequence) without aborting the current waveform and re-initializing the AFG.

**HP BASIC Program Example (WAVE_SEL)**

```hpbasic
1 !RE-STORE "WAVE_SEL"
2 !This program changes the output waveform sequence once the AFG has been
3 !INITiated by writing the location of a sequence's base address to the
4 !Waveform Select register. All register reads and writes are 16 bit.
5 !
10 !Assign an I/O path between the computer and the AFG
20 ASSIGN @Afg TO 1680
30 ASSIGN @Afg1 TO 1680;FORMAT OFF !path for binary data
40 COM @Afg,@Afg1,Base_addr,Seq1_addr,Seq2_addr,Seq3_addr
50 !
60 !Subprograms which reset the AFG and erase all existing waveforms.
70 CALL Rst
80 CALL Wf_del
90 !
100 !SCPI commands which configure the AFG
110 OUTPUT @Afg;"SOUR:FREQ1:FIX 4.096E6;"; !Sample rate
120 OUTPUT @Afg;"SOUR:FUNC:SHAPE USER;"; !function
130 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 2.1V" !amplitude
140 OUTPUT @Afg;"SOUR:ARB:DAC:SOUR INT" !dac data source
150 OUTPUT @Afg;"SOUR:ARB:DAC:FORM SIGN" !dac data format
160 !
170 !Subprograms which define waveforms and load them into segment
180 !and sequence memory, which determine the AFG's register locations
190 !in A24, and which configure the AFG's sequence base memory.
200 CALL Waveform_def
210 CALL A24_offset
220 CALL Build_ram
230 !
240 !Select an output sequence, and initiate (start) waveform output.
250 OUTPUT @Afg;"SOUR:FUNC:USER SEQ1" !waveform sequence
260 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
270 !
280 !Subprogram which changes the output sequence with register writes.
290 CALL Wave_change
300 END
310 !
320 SUB Waveform_def
330 COM @Afg,@Afg1,Base_addr,Seq1_addr,Seq2_addr,Seq3_addr
340 CALL Sinx_def
350 CALL Sind_def
360 CALL Spike_def
370 SUBEND
380 !
390 SUB A24_offset
```

*Continued on Next Page*
A24_offset: !Subprogram which determines the base address for
the AFG registers in A24 address space.

COM @Afg,@Afg1,Base_addr(Seq1_addr,Seq2_addr,Seq3_addr
CONTROL 16,25;2!access A16 space with READIO and WRITEIO
A16_addr=DVAL("D400",16) !AFG A16 base address
Offset=READIO(-16,A16_addr+6) !read AFG offset register
Base_addr=Offset*256 !shift offset for 24-bit address

SUBEND

SUB Build_ram

Build_ram: !This subprogram configures the AFG's sequence base memory
such that there are valid sequence base addresses in memory
before the AFG is INITiated and waveforms are selected.

COM @Afg,@Afg1,Base_addr(Seq1_addr,Seq2_addr,Seq3_addr
CONTROL 16,25;3!access A24 space with READIO and WRITEIO

!Preserve Traffic register contents. Set bits 15-14 to 1 0 to set
!the Waveform Select register as the source which selects the output
!waveform sequence.

Traffic=BINAND(READIO(-16,Base_addr+IVAL("8",16)),IVAL("3FFF",16))
WRITEIO -16,Base_addr+IVAL("8",16);BINIOR(Traffic,IVAL("8000",16))

!Write the location of the sequence base address (waveform index)
!to the Waveform Select register. Write the base address of
!of the sequence in sequence memory to the Sequence Base register.

Wav_sel=BINAND(READIO(-16,Base_addr+IVAL("A",16)),IVAL("00FF",16))
WRITEIO -16,Base_addr+IVAL("A",16);BINIOR(Wav_sel,IVAL("0100",16))
WRITEIO -16,Base_addr+IVAL("20",16);Seq1_addr !sequence mem base addr

Wav_sel=BINAND(READIO(-16,Base_addr+IVAL("A",16)),IVAL("00FF",16))
WRITEIO -16,Base_addr+IVAL("A",16);BINIOR(Wav_sel,IVAL("0200",16))
WRITEIO -16,Base_addr+IVAL("20",16);Seq2_addr !sequence mem base addr

Wav_sel=BINAND(READIO(-16,Base_addr+IVAL("A",16)),IVAL("00FF",16))
WRITEIO -16,Base_addr+IVAL("A",16);BINIOR(Wav_sel,IVAL("0300",16))
WRITEIO -16,Base_addr+IVAL("20",16);Seq3_addr !sequence mem base addr

SUBEND

SUB Wave_change

Wave_change: !Once the AFG has been INITiated, this subprogram changes
!the output waveform sequence by writing the location of the
!sequence's base address in sequence base memory to the
!Waveform Select register.

!Read the waveform select register and write back the value read, in
!order to clear the WFUSED bit in the Status register.

Continued on Next Page
900 Wav_sel=READIO(-16,Base_addr+IVAL("A",16))
910 WRITEIO -16,Base_addr+IVAL("A",16);Wav_sel
920 ! Select a waveform by writing to the Waveform Select register
930 ! following a 0-to-1 transition of the WFUSED bit in the Status
940 ! register. The transition indicates a new waveform can be selected.
950 ! 256 selects sequence 1, 512 selects sequence 2, and 768 selects
960 ! sequence 3.
970 Wav_sel=BINAND(READIO(-16,Base_addr+IVAL("A",16)),IVAL("00FF",16))
980 LOOP
1000 FOR I=256 TO 768 STEP 256
1010 WRITEIO -16,Base_addr+IVAL("A",16);BINIOR(Wav_sel,I)
1020 REPEAT
1030 UNTIL BIT(READIO(-16,Base_addr+2),10)
1040 NEXT I
1050 END LOOP
1060 SUBEND
1070 !
1080 SUB Sinx_def
1090 Sinx_def: ! Define the waveform Sin(x)/x. Download the waveform data
1100 ! as a combined list (voltage and marker) of signed numbers
1110 ! lin an indefinite length block. Download the sequence as a
1120 ! combined list (repetition count, marker, and segment address)
1130 ! lin an indefinite length arbitrary block.
1140 COM @Afg,@Afg1,Base_addr,Seq1_addr,Seq2_addr,Seq3_addr
1150 INTEGER Waveform(1:4096)
1160 INTEGER Sequence(1:2)
1170 REAL Addr_seg1
1180 FOR I=-2047 TO 2048
1190 IF I=0 THEN I=1.E-38
1200 Waveform(I+2048)=((SIN(2*PI*.53125*I/256))/(.53125*I/256)*.159154943092)/.00125
1210 ! shift bits to dac code positions
1220 Waveform(I+2048)=SHIFT(Waveform(I+2048),-3)
1230 NEXT I
1240 !
1250 OUTPUT @Afg;"SOUR:LIST1:SEGM:SEL SIN_X" ! segment name
1260 OUTPUT @Afg;"SOUR:LIST1:SEGM:DEF 4096" ! segment size
1270 OUTPUT @Afg USING ",#,K";"SOUR:LIST1:SEGM:COMB #0" ! waveforms points
1280 OUTPUT @Afg1;Waveform(*) ! indefinite length block
1290 OUTPUT @Afg;CHR$(10);END ! terminate with line feed (LF) and EOI
1300 !
1310 OUTPUT @Afg;"SOUR:LIST1:SEGM:ADDR?"
1320 ENTER @Afg;Addr_seg1
1330 Addr_seg1=Addr_seg1/8 ! /8 to set starting address (boundary) of segment
1340 !
1350 ! Sequence (1) is the repetition count and marker enable for
1360 ! segment SIN_X. Sequence (2) is the starting address of segment SIN_X.
1370 Sequence(1)=SHIFT(4096-1,-4)+Addr_seg1 DIV 65536.
1380 Sequence(2)=Addr_seg1 MOD 65536.-65536.*(Addr_seg1 MOD 65536.>32767)

Continued on Next Page
SUB Sind_def

!Compute the damped sine waveform. Download the data
!as a combined list (voltage and marker) of signed numbers
!in an indefinite length block. Download the sequence as a
!combined list (repetition count, marker, and segment address)
!in an indefinite length arbitrary block.
COM @Afg1,Base_addr,Seq1_addr,Seq2_addr,Seq3_addr

INTEGER Waveform(1:4096)
INTEGER Sequence(1:2)
REAL Addr_seg2

A=4/4096
W=(2*PI)/50
 FOR T=1 TO 4096
    Waveform(T)=EXP(-A*T)*SIN(W*T)/.00125
    !shift bits to dac code positions
    Waveform(T)=SHIFT(Waveform(T),-3)
 NEXT T

!Sequence (1) is the repetition count and marker enable for
!segment SIN_D. Sequence (2) is the starting address of segment SIN_D.
Sequence(1)=SHIFT(4096-1,-4)+Addr_seg1 DIV 65536.
Sequence(2)=Addr_seg2 MOD 65536.-65536.*(Addr_seg2 MOD 65536.>32767)

SUBEND

Continued on Next Page
Spike_def: !Compute the waveform (sine wave with spike). Download the

!data as a combined list (voltage and marker) of signed

!numbers in an indefinite length block. Download the sequence as

!a combined list (repetition count, marker, and segment address)

!in an indefinite length arbitrary block.

COM @Afg,@Afg1,Base_addr,Seq1_addr,Seq2_addr,Seq3_addr

INTEGER Waveform(1:4096)

INTEGER Sequence(1:2)

REAL Addr_seg3

FOR I=1 TO 4096
  Waveform(I)=SIN(2*PI*(I/4096))/ .00125
NEXT I

Width=50

FOR J=1 TO Width
  I=1024-Width+J
  Waveform(I)=Waveform(I)+.9*J/Width/.00125
NEXT J

FOR J=1 TO Width-1
  I=1024+Width-J
  Waveform(I)=Waveform(I)+.9*J/Width/.00125
NEXT J

!shift bits to dac code positions

FOR I=1 TO 4096
  Waveform(I)=SHIFT(Waveform(I),-3)
NEXT I

!Sequence (1) is the repetition count and marker enable for

!segment SPIKE. Sequence (2) is the starting address of segment SPIKE.

Sequence(1)=SHIFT(4096-1,-4)+Addr_seg3 DIV 65536.

Sequence(2)=Addr_seg3 MOD 65536-65536.*(Addr_seg3 MOD 65536>.32767)

!sequence name

Continued on Next Page
Comments

- SCPI commands are included in this program to load segment and sequence memory, and initialize the AFG. This program executes as intended when the SCPI commands are executed prior to writing to the registers.

- The sequence in which the Waveform Selection Registers are written to and the register contents are summarized below.

The Traffic Register selects the source which specifies addresses in sequence base memory that, in turn, select the waveform sequences. The Waveform Select Register (selected by the Traffic Register) contains the waveform index which is the location in sequence base memory where the base address of the sequence in sequence memory is located.

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, WAVE_SEL.FRM, is in directory “VBPROG” and the Visual C example program, WAVE_SEL.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Loading the DAC from the VXIbus

This section shows how to load waveform data into the AFG’s DAC directly from the VXIbus backplane. For additional information on loading the DAC directly, refer to Chapter 7.

The High Speed Data Register

Waveform data from the VXIbus is loaded into the DAC via the following register.

- **High-Speed Data Register:**

  \[
  \text{base\_addr} + 26_{16}
  \]

<table>
<thead>
<tr>
<th>Address</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>base + 26_{16}</td>
<td>DAC code</td>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DAC Code:** The DAC code is a 13-bit signed (2’s complement) or unsigned number. With [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] set to 5.12 V and a matched output load, the least significant bit (LSB) is 1.25 mV.

HP BASIC Program Example (VXISRCE)

The program uses the V360 Controller to download the data using the VXIbus instead of transferring it directly to the AFG using HP-IB.

```
1 !RE-STORE"VXISRCE"
2 !This program uses the V/360 embedded controller to send waveform
3 !data directly to the AFG dac over the VXIbus backplane.
4 !
10 !Assign I/O path between the computer and E1445A.
20 ASSIGN @Afg TO 1680
30 COM @Afg,Addr
40 !
50 !Call the subprograms which reset the AFG and determine the base
60 !address of the registers in A24 address space.
70 CALL Rst
80 CALL A24_offset
90 !
100 !Scale the amplitude, set the dac data format and dac data source.
110 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5.11875V" !amplitude
120 OUTPUT @Afg;"SOUR:ARB:DAC:FORM SIGN" !dac data format (signed)
130 OUTPUT @Afg;"SOUR:ARB:DAC:SOUR VXI" !dac data source
140 OUTPUT @Afg;"*OPC?" !Wait for the SCPI commands to complete
150 ENTER @Afg;Complete
160 !
170 !Call the subprogram which sends data directly to the dac.
```

*Continued on Next Page*
CALL Dac_drive
END

SUB A24_offset
A24_offset: !Subprogram which determines the base address for
the AFG registers in A24 address space, then adds the
offset and register number to the base to get the
complete address.
COM @Afg,Addr
!CONTROL 16,25;2 !access A16 space with READIO and WRITEIO
A16_addr=DVAL("D400",16) !AFG A16 base address
Offset=READIO(-16,A16_addr+6) !read AFG offset register
Base_addr=Offset*256 !shift offset for 24-bit address
!Add the register number of the high speed data register
to the A24 base address.
Addr=Base_addr+IVAL("26",16)
SUBEND

SUB Dac_drive
Dac_drive: !Subprogram which computes a 128 point, 5 Vpp triangle wave and
writes the corresponding codes directly to the DAC via
the VXIbus and High Speed Data register.
COM @Afg,Addr
!CONTROL 16,25;3 !access A24 space with WRITEIO
INTEGER I,Waveform(1:128) !Calculate triangle wave (dac codes)
FOR I=1 TO 64
  Waveform(I)=I*.0755/.00125
  Waveform(I)=SHIFT(Waveform(I),-3) !shift bits to dac code positions
NEXT I
FOR I=65 TO 128
  Waveform(I)=(128-I)*.0755/.00125
  Waveform(I)=SHIFT(Waveform(I),-3) !shift bits to dac code positions
NEXT I
!Continuously write data (in 16-bit words) to the dac via the
VXIbus and High Speed Data register.
LOOP
FOR I=1 TO 128
  WRITEIO -16,Addr;Waveform(I)
NEXT I
END LOOP
SUBEND

SUB Rst
Rst: !Subprogram which resets the E1445.
COM @Afg,Addr
OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
ENTER @Afg;Complete
SUBEND
Comments

- To simplify the program, SCPI commands are included so that the only register written to is the High-Speed Data Register. This program executes as intended when those SCPI commands which configure the AFG are executed before the register is written to.

- This program was written using the system configuration described on page 484. Data is written to the DAC at a rate of 115 µs per amplitude point, which is limited by the execution speed of HP BASIC.

Visual BASIC and Visual C/C++ Program Versions

The Visual BASIC example program, VXISRCE.FRM, is in directory “VBPROG” and the Visual C example program, VXISRCE.C, is in directory “VCPROG” on the CD that came with your HP E1445A.
Index

HP E1445A AFG Module User's Manual

*CLS, 47, 416
*DMC, 416
*EMC, 417
*EMC?, 417
*ESE, 417
*ESE?, 417
*ESR?, 418
*GMC?, 418
*IDN?, 419
*LMC?, 419
*LRN?, 48, 420
*OPC, 420
*OPC?, 421
*PMC, 421
*PUD, 422
*PUD?, 422
*RCL, 423
*RMC, 423
*RST, 47, 424
*SAV, 424
*SRE, 425
*SRE?, 425
*STB?, 426
*TRG, 426
*TST?, 46, 426
*WAI, 427

32-Bit Integer Data
how HP BASIC transfers, 255

A

A24 Address
base address, 484, 486
query space, 407

Abbreviated SCPI Commands, 285

ABORt
subsystem, 290
using, 196

Aborting
waveforms, 196

AC Calibration Corrections, 304
AC Output Leveling, 144-146, 160-161

amplitude errors, 160-161
Accessing the Registers, 484, 486

Address
A24 space query, 407
command module HP-IB port, 22
HP-IB, 22
logical, 21-22
primary HP-IB, 22
query segment sequence, 347
query waveform segment, 336
secondary HP-IB, 22

AFG
arming count, 199, 292
arming sources, 199
arming the, 165-171, 291-297
block diagram description, 445-452
bus request level, setting, 24
calibration, 298-305
description, 445
frequency modes, 155, 198
frequency synthesis modes, 197
gating polarity, 200
gating sources, 200
high speed operation, 223-280
initiating the, 306-307
installing in mainframe, 25
lock-stepping multiple, 176-180
memory description, 452
operating multiple AFGs together, 218-222
servant area, 23
specifications, 453-462
status, 429-444
triggering the, 172-185

AFGGEN1 Example Program, 102
AFGGEN2 Example Program, 104

Amplitude
characteristics, 457-458
effects on DAC codes, 280
errors, AC leveling, 160-161
levels, selecting, 72-74
limits, 471
output, default voltage units, 379
output, setting, 72-74, 377-380
voltage list effect, 113

Arbitrary Block
data, 156-157
frequency lists using, 130-132
SCPI command parameters, 286

Arbitrary Waveforms
commands flowchart, 84-85
description, 446
doubling frequency, 155
FSK, using, 152-153
generating, 83-116
generating, damped sine waves, 107-108
generating, description, 86-87
generating, exponential charge/discharge waveform, 108-109
generating, half rectified sine waves, 111
generating, marker pulses, 206
generating, noise, 112
generating, non-sinusoid, 447
generating, sample programs, 104-109, 111-112
generating, simple, 88-92
generating, sin(x)/x, 105-106
generating, spiked sine waves, 109
generating, with different frequency generators, 99-102, 104
generating, with different waveform segments, 93-98
generating, with single waveform segment, 88-92
minimum frequency, 155
number of waveform points, 157
output frequency, 160
program comments, 113-116
sample rate, setting, 331
sweep direction, 157
sweeping, 141

ARBWAVE.C Program Example, 41-45
ARBWAVE.FRM Example Program, 91-92
ARBWAVE.FRM Program Example, 33-39

ARM Subsystem, 291-297
ARM[:STAR][:LAY[1]];COUN, 291
ARM[:STAR]:LAY2:COUN, 292
ARM[:STAR]:=LAY2[:IMM], 293
ARM[:STAR]:LAY2:SLOP, 293
ARM[:STAR]:LAY2:SOUR, 294
ARM:SWEep:COUN, 295
ARM:SWEep[:IMM], 295
ARM:SWEep:LINK, 296
ARM:SWEep:SOUR, 297

ARM-TRIG, 163-202
configuration, 164
flowchart, 164
states, 164

Arming
and triggering, 163-202
and triggering, a frequency list, 193-195
and triggering, a frequency sweeps, 190-192

and triggering, frequency sweeps and lists, 186-189
and triggering, immediately, 201
and triggering, program comments, 197-201
commands, 165, 291-297
count, 199, 292
frequency sweeps or lists, 295
setting number of, 169-171
slope, setting, 293
sources, 199
sources, setting, 166-168, 294
sweep sources, 297
sweeps, linking, 296
sweeps, setting, 295, 372
sweeps, sources, 297
the AFG, 165-171, 291-297
waveforms immediately, 293

ASCii Data Format, 335, 358
Assigning the AFG to a Commander, 21

Attenuator Description, 451

B

Backplane
downloading segment data, 259-268
using VXIbus, 259-271, 506-508

Base Address, 484, 486

Bits
operation condition register, 383
operation event summary bit, 383
questionable signal condition register, 386
questionable signal event summary bit, 386

Block Diagram
description, 445-452

Boolean Command Parameters, 286

BURST Example Program, 170-171

Bus
data transfer bus, 24
request level, 24
request level guidelines, 24
request lines, 24

Byte
size, definite length block data, 231
size, indefinite length block data, 235

C

CALibration Subsystem, 298-305
CAL:COUNt?, 298
CAL:DATA:AC[1], 299
CAL:DATA:AC2, 299
CAL:DATA[:DC], 300
CAL[:DC]:BEGIN, 300
CAL[:DC]:POINt?, 301
CAL:SECure:CODE, 302
CAL:SECure[:STATe], 303
CAL:STATe, 304
CAL:STATe:AC, 304
CAL:STATe:DC, 305

Cataloging
segment sequence names, 348
waveform segment names, 336

Certification, 13

Changing
output frequency, 487
security passwords, 302
signal phase, 495

CHARGE Example Program, 108-109

Checking for Errors, 49

Clearing AFG, 47
example program, 47
*CLS, 47, 416

Combined
segment list, 239-250
segment list, defining, 337
segment list, determining size, 280
segment list, format, 239-240, 245
segment list, query, 338
segment sequence list, 250, 348-349
segment sequence list format, 250, 260
segment sequence list, determining size, 280
segment sequence list, query, 349
segments and sequences, 250-258
sequences, using, 250-258
signed data, using, 239-244
unsigned data, using, 245-249

waveform segment list format, 259

COMBSEQ Example Program, 255-258

COMBSIGN Example Program, 242-244

COMBUNS Example Program, 247-249

Command Module
HP-IB port address, 22

Command Reference, 281-312

ABORt subsystem, 290
ARM subsystem, 291-297
CALibration subsystem, 298-305
*CLS, 47, 416
*DMC, 416
*EMC, 417
*EMC?, 417
*ESE, 417
*ESE?, 417
*ESR?, 418
*GMC?, 418
*IDN?, 419
INITiate subsystem, 306-307
*LMC?, 419

*LRN?, 48, 420
*OPC, 420
*OPC?, 421
OUTPut[1] subsystem, 308-311
*PMC, 421
*PUD, 422
*PUD?, 422
*RCL, 423
*RMC, 423
*RST, 47, 424
*SAV, 424
[SOURce:] subsystem, 312-380
[SOURce]ARBitrary subsystem, 313-318
[SOURce]:FREQuency[1] subsystem, 319-329
[SOURce]:FREQuency2 subsystem, 330-331
[SOURce]:FUNCtion subsystem, 332-333
[SOURce]:LIST[1] subsystem, 334-357
[SOURce]:LIST2 subsystem, 358-360
[SOURce]:MARKeR subsystem, 361-364
[SOURce]:PM subsystem, 365-367
[SOURce]:RAMP subsystem, 368-369
[SOURce]:ROSCillator subsystem, 370-371
[SOURce]:SWEep subsystem, 372-376
[SOURce]:VOLTage subsystem, 377-380
*SRE, 425
*SRE?, 425
STATus subsystem, 381-388
*STB?, 426
SYSTem subsystem, 389-390
*TRG, 426
TRIGger subsystem, 391-402
*TST?, 46, 426
VINStrument subsystem, 403-408
*WAI, 427

Commander, assigning AFG to, 21

Commands
ABORt, 290
arbitrary waveform generation, 84-85
ARM, 291-297
arming, 165, 291-297
CALibration, 298-305
*CLS, 47, 416
common, 416-428
coupling, 27-28, 288
coupling groups, 27-28, 467-469
*DMC, 416
*EMC, 417
*EMC?, 417
ew, 417
*ESE, 417
*ESE?, 417
*ESR?, 418

FSK programming, 118-119
Common (*) Commands, 416-428
*CLS, 47, 416
*DMC, 416
*EMC, 417
*EMC?, 417
*ESE, 417
*ESE?, 417
*ESR?, 418
format, 284
*GMC?, 418
*IDN?, 419
linking, 288
*LMC?, 419
*LRN?, 48, 420
marker pulses, 204, 342-343, 354-355, 362-364
*OPC, 420
*OPC?, 421
OUTPut[1], 308-311
*PMC, 421
*PUD, 422
*PUD?, 422
*RCL, 423
*RM, 423
*RST, 47, 424
*SAV, 424
SCPI structure, 26
[SOURce: ], 312-380
[SOURce]:ARBitrary, 313-318
[SOURce]:FREQuency[1], 319-329
[SOURce]:FREQuency2, 330-331
[SOURce]:FUNCtion, 332-333
[SOURce]:LIST[1], 334-357
[SOURce]:LIST2, 358-360
[SOURce]:MARKer, 361-364
[SOURce]:PM, 365-367
[SOURce]:RAMP, 368-369
[SOURce]:ROSCillator, 370-371
[SOURce]:SWEep, 372-376
[SOURce]:VOLTage, 377-380
*SRE, 425
*SRE?, 425
standard waveform generation, 54-55
STATus, 381-388
*STB?, 426
SYSTem, 389-390
*TRG, 426
TRIGger, 391-402
triggering, 172, 391-402
*TST?, 46, 426
types, 284
VINstrument, 403-408
*WAI, 427
Comment Sheet, reader, 17
Common (*) Commands, 416-428
*CLS, 47, 416
*DMC, 416
*EMC, 417
*EMC?, 417
*ESE, 417
*ESE?, 417
*ESR?, 418
*GMC?, 418
*IDN?, 419
linking, 288
*LMC?, 419
*LRN?, 48, 420
marker pulses, 204, 342-343, 354-355, 362-364
*OPC, 420
*OPC?, 421
*PMC, 421
*PUD, 422
*PUD?, 422
quick reference, 428
*RCL, 423
*RMC, 423
*RST, 47, 424
*SAV, 424
*SRE, 425
*SRE?, 425
*STB?, 426
*TRG, 426
*TST?, 46, 426
*WAI, 427
Condition Register
operation status group, 435
query bits, 383, 386
questionable signal status group, 431
reading, 431, 435
Configuration
ARM-TRIG, 164
VXIbus local bus testing, 405
Conformity, declaration, 15
Connector
digital port in, 272-279
digital port in, downloading, 279
digital port in, pinout, 278
digital port in, selecting a sequence, 279
digital port in, specifications, 460
marker out BNC, 204, 342, 363-364
ref/sample in BNC, 395, 398
start arm in BNC, 293
stop trig/FSK/gate in BNC, 393
Count
arming, 199, 292
frequency list repetitions, 156
sweeps, 156, 295, 372
waveform repetition, 199, 291
waveform segment sequence outputs, 352-353
Coupled Commands
coupling, 27, 288
coupling groups, 288, 467-469
executing, 28, 288
frequency 1 generator, 319
Cycles
arming, setting, 169-171
arming, stopping, 397-398
waveform, setting, 169-171

D

DAC
data source, selecting, 315
downloading data directly into, 269-271, 506-508
loading from VXIbus, 506-508
output description, 447
sources, 280

DAC Codes
amplitude effects on, 280
incorrect, 280
output voltage points, 346
sending segment data with, 114
specifying format, 313-314
transferring, in signed number format, 225, 240
transferring, in unsigned number format, 229, 245

DACBLOK1 Example Program, 232-234
DACBLOK2 Example Program, 236-238

Damped Sine Waves, generating, 107-108

Data
arbitrary block, 156-157
byte size, definite length blocks, 231
byte size, indefinite length blocks, 235
downloading directly into the DAC, 269-271, 506-508
downloading segment into memory, 259-268
downloading using backplane, 259
downloading using digital in port, 279
format, ASCII, 335, 358
format, definite length blocks, 231
format, indefinite length blocks, 235
format, PACKed, 335
format, REAL, 358
phase modulation, 366
segment sequence, 316
sending DAC codes, 114
signed, combined, 239-244
signed, to generate waveforms, 225-228
transfer bus, 24
transfer bus, operating mode, 406
transfer methods, 224
transferring 32-bit integer, 255
unsigned, combined, 245-249
unsigned, to generate waveforms, 229-230
VXIbus local bus test, 406
waveform segment, 316

DC
calibration, corrections, 305
calibration, starting, 300-301
calibration, starting, 300-301
calibration, starting, 300-301
calibration, starting, 300-301
voltage, generating, 56-57

DCVOLTS Example Program, 56-57
DDS Frequency Generator
control, 489-491
description, 449
generating waveforms with, 99-102
ranges, 79

Declaration of Conformity, 15

Definite Length Arbitrary Blocks, 130-132, 156
data byte size, 231
data format, 231
transferring data using, 231-234

Deviation
angle, default units, 367
angle, setting, 76, 365
units for phase modulation, 80, 365, 367

Digital Port In Connector
downloading data, 279
pinout, 278
selecting a sequence, 279
specifications, 460
using, 272-279

Disabling
analog output, 311
calibration corrections, 304-305
downloading mode, 318
ECL trigger lines, 362
frequency doubling, 326
low-pass output filter, 309
marker signal, 362, 364
phase modulation, 367
sample gating, 394
security code, 303
trigger system, 290

Discrete Command Parameters, 286
DIV_N Example Program, 174-175

Divide-by-N Frequency Generator
control, 492-494
description, 449
generating waveforms with, 99-101, 104, 174-175
*DMC, 416

Documentation History, 14

Doubling Frequency Ranges, 155, 326

Downloading, 316-317
combined segment sequence list, 260
combined waveform segment list, 259
data directly into DAC, 269-271, 506-508
data using digital in port, 279
disabling, 318
segment data into DAC, 269-271, 506-508
segment data into memory, 259-268
segment data using backplane, 259
sources, 280
DRIFT Example Program, 220-222
Driving TTLTrgn Trigger Lines, 162

E

ECLTrgn
functions, 460
trigger lines, 204-205, 361-362

Embedded Computer
See Embedded Controller

Embedded Controller
using, 23
*EMC, 417
*EMC?, 417

Enable Register, 383, 386
operation status group, 436
questionable signal status group, 432

Enabling
analog output, 311
calibration corrections, 304-305
ECL trigger lines, 362
frequency doubling, 326
gate, 200
low-pass output filter, 309
marker signal, 362, 364
phase modulation, 367
sample gating, 394
security code, 303

End-Of-Line Terminator, suppressing, 31

Error
AC leveling amplitude, 160-161
checking for, 49
messages, in error queue, 389
messages, list of, 475-479
messages, settings conflict, 480-482
numbers, 475-479
queue, 389
ERRORCHK Example Program, 441
ERRORCHK Program Example, 49
*ESE, 417
*ESE?, 417
*ESR?, 418

Event Register
operation status group, 436
query contents, 384, 387
questionable signal status group, 432
summary bit, 383, 386

Example Programs
AFGGEN1, 102
AFGGEN2, 104
ARBWAVE.C, 41-45
ARBWAVE.FRM, 33-39, 91-92
BURST, 170-171

CHARGE, 108-109
COMBSEQ, 255-258
COMBSIGN, 242-244
COMBUNS, 247-249
DACBLOK1, 232-234
DACBLOK2, 236-238
DCVOLTS, 56-57
DIV_N, 174-175
DRIFT, 220-222
ERRORCHK, 49, 441
EXT_ARM, 167-168
FREQ1_REG, 489-491
FREQ2_REG, 492-494
FSK1, 148-149
FSK2, 150-151
FSK_ARB, 152-153
GATE, 184-185
list of, 464-466
LIST1, 125-126
LIST_STP, 194-195
LIST_TME, 139-140
LISTDEF, 131-132
LOCKSTEP, 177-180
LOG_SWP, 133-134
LRN, 48
MARKSEG1, 209-211
MARKSEG2, 213
MARKTRG, 215-217
MULSEG, 96-98
NOISE, 112
OSG_RQS, 437
OUTPLOAD, 70-71
OUTPUNIT, 73-74
PHAS_CHNG, 496-497
PHS_MOD, 76-77
QSSG_RQS, 433-434
RSTCLS, 47
RSTSINE, 51
SIGN_DAT, 227-228
SIN_D, 107-108
SIN_R, 111
SIN_X, 105-106
SINEWAVE, 59-60
SLFTST, 47
SMPLSWP1, 122-123
SMPLSWP2, 128-129
SPIKES, 109
SQUWAVE, 63-64
STOPTRIG, 181-182
SWP_ARB, 141-143
SWP_LEVL, 30-31, 145-146
SWP_PVST, 136-137
SWP_STEP, 191-192
Executing
  coupled commands, 28, 288
  SCPI commands, 288
  several waveform segments, 93-98

Exponential Charge/Discharge Waveform, generating, 108-109
EXT_ARm Example Program, 167-168

External
  reference oscillator frequency, 370
  stop trigger slope, 200

F

Factory Settings, 20
Filter
description, 451
  low-pass output, 308-309
  negative transition, 384, 387, 431, 435
  positive transition, 385, 388, 431, 435
Flowchart
  arbitrary waveform commands, 84-85
  ARM-TRIG, 164
  frequency, lists commands, 118-119
  frequency, sweep commands, 118-119
  frequency-shift keying (FSK) commands, 118-119
  marker pulse commands, 204
  standard waveform commands, 54-55
Format
  combined segment list, 239-240, 245
  combined segment sequence list, 250, 260
  combined waveform segment list, 259
  common commands, 284
  DAC codes, 313-314
  definite length block data, 231
  indefinite length block data, 235
  SCPI commands, 284-285
  signed number, 225-226
  signed number, combined list, 240
  unsigned number, 229
  unsigned number, combined list, 245
FREQ1_REG Example Program, 489-491
FREQ2_REG Example Program, 492-494
Frequency
  agility, 456
  characteristics, 454-456
  external reference oscillator, 370

  generator, DDS, 79, 99-102, 449, 489-491
generator, description, 448-449
generator, divide-by-n, 99-101, 104, 174-175, 449, 492-494
  limits, 470
  list, advance trigger, 201
  list, arming, 193-195, 201, 295
  list, defining, 359
  list, query length, 360
  list, setting up, 124-126
  list, specifying, 124-126
  list, triggering, 193-195
  list, using arbitrary blocks, 130-132
  list, versus time, 138-140, 159
  lists, and sweeping, 120
  lists, commands flowchart, 118-119
  lists, frequency range, 155
  lists, maximum number, 155
  lists, repetition count, 155
  load strobe register, 488
  logarithmic spacing, 133-134
  low-pass output filter, 308
  modes, 155, 198
  output, arbitrary waveform, 160
  output, changing, 487
  points, 157
  range, doubling, 155, 326
  range, frequency lists and FSK, 155
  range, sweeping and sampling, 155
  repetition, determining, 87
  span, 327
  standard waveform, setting, 331
  start and span, 127-129
  start and stop, 121-123
  sweep, advance source, 158
  sweep, advance trigger, 201
  sweep, arming, 190-192, 201, 295
  sweep, commands flowchart, 118-119
  sweep, count, 372
  sweep, direction, 157, 373
  sweep, points, 374
  sweep, sample rate, 160
  sweep, spacing, 157, 375
  sweep, time, 158-159, 376
  sweep, triggering, 190-192, 399-402
  sweep, using triggers, 186-189
  synthesis modes, 197
  vs. number of points, 79
Frequency Control
  programs, 489-494
  registers, 487-488
Frequency Shift Keying
  See FSK
Frequency 1 Generator
  characteristics, 454
  coupling rules, 319
  description, 449
  range, 116
  SCPI commands, 319-329
Frequency 2 Generator
  characteristics, 455
  description, 449
  SCPI commands, 330-331
FSK, 147-153
  and sweeping, 117-162
  command reference, 120
  control sources, 161
  delay, 162
  frequency range, 155
  program comments, 154-162
  programming flowchart, 118-119
  sample rate, setting, 323
  sample rate, source, 324
  using, arbitrary waveform, 152-153
  using, FSK control source, 147-149
  using, TTLTrgn control source, 150-151
FSK1 Example Program, 148-149
FSK2 Example Program, 150-151
FSK_ARB Example Program, 152-153
G
GATE Example Program, 184-185
Gating
  and signal phase, 200
  disabling, 394
  enabling, 200, 394
  polarity, 200, 393
  sources, 200, 393
  trigger signals, 183-185
  using, 196
Generating
  arbitrary wave ramp, Visual BASIC program, 33-39
  arbitrary wave ramp, Visual C/C++ program, 41-45
  arbitrary waveforms, 83-116
  arbitrary waveforms, damped sine waves, 107-108
  arbitrary waveforms, description, 86-87
  arbitrary waveforms, exponential charge/discharge waveform, 108-109
  arbitrary waveforms, half rectified sine waves, 111
  arbitrary waveforms, noise, 112
  arbitrary waveforms, non-sinusoid, 447
  arbitrary waveforms, sample programs, 104-109, 111-112
  arbitrary waveforms, sample rate, 331
  arbitrary waveforms, simple, 88-92
  arbitrary waveforms, sin(x)/x, 105-106
  arbitrary waveforms, spiked sine waves, 109
  arbitrary waveforms, with different frequency generators, 99-102, 104
  arbitrary waveforms, with different waveform segments, 93-98
  arbitrary waveforms, with single waveform segment, 88-92
  DC voltages, 56-57
  exponential charge/discharge waveform, 108-109
  marker pulses, arbitrary waveforms, 206
  marker pulses, each waveform point, 214-217
  marker pulses, multiple, 207-211
  marker pulses, single, 212-213
  noise, 112
  ramp waves, 65-68
  sine waves, 51, 58-60, 450
  sine waves, damped, 107-108
  sine waves, half rectified, 111
  sine waves, spiked, 109
  square waves, 61-64
  standard waveforms, 53-82
  standard waveforms, frequencies, 331
  standard waveforms, phase modulation, data sources, 366
  standard waveforms, phase modulation, default angle units, 367
  standard waveforms, phase modulation, disabling, 367
  standard waveforms, phase modulation, enabling, 367
  standard waveforms, phase modulation, selecting deviation units, 80, 365
  standard waveforms, phase modulation, using, 75-77
  standard waveforms, ramp waves, 65-68
  standard waveforms, selecting amplitude levels, 72-74
  standard waveforms, selecting output loads, 69-71
  standard waveforms, selecting output units, 72-74
  standard waveforms, sine waves, 58-60
  standard waveforms, square waves, 61-64
  standard waveforms, triangle waves, 65-68
  triangle waves, 65-68
  waveforms, shape selection, 332
  waveforms, using signed data, 225-228
  waveforms, using unsigned data, 229-230
Generators
  frequency 1, characteristics, 454
  frequency 1, description, 449
  frequency 1, range, 116
  frequency 1, SCPI commands, 319-329
  frequency 2, characteristics, 455
  frequency 2, description, 449
  frequency 2, SCPI commands, 330-331
Getting Started, 19-52
*GMC?, 418
Half Rectified Sine Waves, generating, 111
High Speed Data Register, 506
High Speed Operation, 223-280
  program comments, 280
How
HP BASIC transfers 32-bit integer data, 255
  the AFG generates arbitrary waveforms, 86-87
to free memory, 113
HP BASIC
  how to transfer 32-bit integer data, 255
HP BASIC Language Programs, 29-31, 464-466
  AFGGEN1, 102
  AFGGEN2, 104
  ARBWAVE, 91-92
  BURST, 170-171
  CHARGE, 108-109
  COMBSEQ, 255-258
  COMBSIGN, 242-244
  COMBUNS, 247-249
  DACBLOK1, 232-234
  DACBLOK2, 236-238
  DCVOLTS, 56-57
  DIV_N, 174-175
  DRIFT, 220-222
  end-of-line terminator, 31
  ERRORCHK, 49, 441
  EXT_ARM, 167-168
  FREQ1_REG, 489-491
  FREQ2_REG, 492-494
  FSK1, 148-149
  FSK2, 150-151
  FSK_ARB, 152-153
  GATE, 184-185
  LIST1, 125-126
  LIST_STP, 194-195
  LIST_TME, 139-140
  LISTDEF, 131-132
  LOCKSTEP, 177-180
  LOG_SWP, 133-134
  LRN, 48
  MARKSEG1, 209-211
  MARKSEG2, 213
  MARKTRG, 215-217
  MULSEG, 96-98
  NOISE, 112
  OSG_RQS, 437
  OUTUPLOAD, 70-71
  OUTPUNIT, 73-74
  PHAS_CHNG, 496-497
  PHS_MOD, 76-77
  QSSG_RQS, 433-434
  RSTCL, 47
  RSTSSINE, 51
  SIGN_DAT, 227-228
  SIN_D, 107-108
  SIN_R, 111
  SIN_X, 105-106
  SINEWAVE, 59-60
  SLFTST, 47
  SMPLSWP1, 122-123
  SMPLSWP2, 128-129
  SPIKES, 109
  SQUWAVE, 63-64
  STOPTRIG, 181-182
  SWP_ARB, 141-143
  SWP_LEVL, 30-31, 145-146
  SWP_PVST, 136-137
  SWP_STEP, 191-192
  SWP_TRIG, 188-189
  system configuration, 29
  TRIWAVE, 67-68
  UNS_DAT, 230
  VXIDOWN, 264-268
  VXISRCE, 270-271, 506-508
  WAVE_SEL, 500-505
  WAVESELFP, 272-277
HP E1446A
  settings conflict error messages, 482
  using AFG with, 23
HP-IB
  address, 22
  command module port, 22
  interface card, 22
  primary address, 22
  secondary address, 22

Idle State, 164
*IDN?, 419
IEEE-488.2 Common Commands
  See Common (*) Commands
Immediate
  arming and triggering, 201
  arming of waveform, 293
  frequency sweep or list, 295
Impedance
  output, setting, 69-71, 309
Implied SCPI Commands, 27, 285
Increasing Speed, 223-280
Indefinite Length Arbitrary Blocks, 130-132, 157
  data byte size, 235
  data format, 235
transferring data using, 235-238
INITiate Subsystem, 306-307
INITiate:IMMediate, 306-307
Initiating
the AFG, 306-307
waveforms, 165
Installing Module, 25
Instrument
action state, 164
virtual, commands, 403-408
Instrument Language (SCPI), 26
Interface
characteristics, 459
local bus, 403-406
select code, 22
Introductory Programs, 46
AFG self-test, 46
checking for errors, 49
generating sine waves, 51
query power-on/reset configuration, 48
resetting and clearing AFG, 47

L
LADDR, 22
Linear Sweeping, 375
Linking
commands, 288
sweep advance trigger, 400
List of Example Programs, 464-466
LIST1 Example Program, 125-126
LIST_STP Example Program, 194-195
LIST_TME Example Program, 139-140
LISTDEF Example Program, 131-132
*LMC?, 419
Loading DAC from VXIbus, 505-508
Local Bus
operating mode, 403-404
specifications, 460
test data, 406
testing configuration, 405
Lock-Stepping Multiple AFGs, 176-180
LOCKSTEP Example Program, 177-180
LOG_SWP Example Program, 133-134
Logarithmic Sweeping, 133-134, 319, 375
Logical Address, 22
description, 21
Low-pass Output Filter
cut-off frequency, 308
disabling, 309
enabling, 309
LRN, 48
LRN Program Example, 48
*LRN?, 48, 420

M
Mainframe
installing modules, 25
removing modules from, 25
Marker
enable, 251
out BNC, disabling marker signal, 364
out BNC, enabling marker signal, 364
out BNC, marker pulses, 204
out BNC, output pulse, 342
out BNC, selecting polarity, 364
out BNC, selecting sources, 363
outputs, 203-222
outputs, program comments, 222
points, determining number of, 222
polarity, 364
program comments, 222
pulse, arbitrary generated, 206
pulse, commands, 204, 342-343, 354-355, 362-364
pulse, each waveform point, 214-217
pulse, enable flowchart, 204
pulse, front panel BNC, 204
pulse, multiple, 207-211
pulse, segment sequence, 354-355
pulse, single, 212-213
pulse, waveform segment, 337-338, 342-343
signal, disabling, 362, 364
signal, enabling, 362, 364
sources available, 361-363
sources, available, 205-206
sources, front panel BNC, 363
MARKSEG1 Example Program, 209-211
MARKSEG2 Example Program, 213
MARKTRG Example Program, 215-217
Maximum
arbitrary waveform frequency, 155
arbitrary waveform sample rates, 454
frequencies in frequency list, 155
parameters, in coupling groups, 288
parameters, using, 81
ramp wave frequency, 155
sine wave frequency, 155
square wave frequency, 155
triangle wave frequency, 155
Memory
characteristics, 453-454
deleting segment sequence definitions, 351
deleting waveform segment definitions, 340
description, 448, 452
determining amount, 113
donloading segment data into, 259-268
freeing, 113
query segment sequence, 347, 353
query waveform segment, 336, 341
reserving for segment sequence, 350
reserving for waveform segment, 339
segment characteristics, 453
sequence characteristics, 454
storing segment sequence, 87
storing waveform segment, 86
Microprocessor Description, 450
Minimum
arbitrary waveform frequency, 155
parameters, in coupling groups, 288
parameters, using, 81
ramp wave frequency, 155
sine wave frequency, 155
square wave frequency, 155
triangle wave frequency, 155
Module
getting started, 19-52
installing in mainframe, 25
removing from mainframe, 25
MULSEG Example Program, 96-98
Multiple
AFG operations, 203-222
AFG operations, program comments, 222
AFGs, lock-stepping, 176-180
marker pulses, 207-211
SCPI commands, linking, 288
segment lists, 207-211

N
Naming Segment Sequences
query, 348
Naming Waveform Segments, 86
query, 336
Negative Transition Filter, 384, 387, 431, 435
NOISE Example Program, 112
Noise, generating, 112
Non-Sinusoid Arbitrary Waveforms, 447
Number
of arms, setting, 169-171
of frequency sweep points, 374
of marker points, determining, 222
of points, 87, 374
of points vs. frequency, 79
of sweep arms, setting, 295, 372
of waveform cycles, setting, 169-171
signed number format, 225-226
unsigned number format, 229
unsigned number format, combined list, 245
Numeric Command Parameters, 286

O
Offset
A24 address space, 407, 485-486
circuitry description, 451
register, reading, 485-486
*OPC, 420
*OPC?, 421
Operation
from incorrect DAC codes, 280
high speed, 223-280
multiple AFGs, 203-222
multiple AFGs, together, 218-222
Operation Status
condition register, 383, 435
enable register, 383, 436
event register, 383, 436
event register, query contents, 384
group, 435-437
negative transition filter, 384
positive transition filter, 385
register, 381, 435-437
summary bit, 383
transition filter, 435
using, 436-437
Optional SCPI Commands, 27, 285
parameters, 287
Oscillator Sources, 78, 115, 154, 197
OSG_RQS Example Program, 437
OUTPLOAD Example Program, 70-71
OUTPUNIT Example Program, 73-74
Output
amplifier description, 451
amplitude, default voltage units, 379
amplitude, setting, 72-74, 377-379
circuitry, 450
circuitry, description, 451
DAC description, 447
filters, 160
filters, cutoff frequency, 308
filters, disabling, 309
filters, enabling, 309
frequency, arbitrary waveform, 160
frequency, changing, 487
impedance, setting, 69-71, 309
leveling, AC, 144-146, 160-161
loads, comments, 79
loads, selecting, 69-71, 310-311
marker, 203-222
marker, program comments, 222
marker, pulse, 342-343, 354-355
offset voltage, 380
relay, 311
units, comments, 80
units, selecting, 72-74
voltage, waveform segment, 337-338, 345-347
waveforms, 86-87
waveforms, suspending, 183
OUTPut[1] Subsystem, 308-311
OUTP:FILT[:LPAS]:FREQ, 308
OUTP:FILT[:LPAS][:STAT], 309
OUTP:IMPedance, 309
OUTP:LOAD, 310
OUTP:LOAD:AUTO, 311
OUTP[:STATe], 311

P
PACKed Data Format, 335
Parameters
arbitrary block, 286
boolean, 286
discrete, 286
MAXimum, using, 81
MINimum, using, 81
numeric, 286
optional, 287
query settings, 287
SCPI commands, 286-287
Password, changing, 302
PHAS_CHNG Example Program, 496-497
Phase
control program, 496-497
control registers, 495
increment registers, 487
load strobe register, 495
Phase Modulation, 75
commands, 365-367
data source, 366
default units, 367
deviation, 76, 365
disabling, 367
enabling, 367
registers, 495
selecting deviation units, 80, 365
using, 75-77
PHS_MOD Example Program, 76-77
Pinout, digital port in connector, 278-279
plug&play
See VXIplug&play Online Help
PM

See Phase Modulation
*PMC, 421
Points
combined list length query, 338, 349
frequency sweep, 374
marker pulse list length query, 343, 355
output voltage, 345-347
output voltage list length query, 347
ramp waves, 368
segment sequence repetition query, 353
single, 343, 355
triangle waves, 368
versus time, 135-137, 159
Polarity
gating, 200
marker signal, 364
ramp waves, 369
sample gate, 393
square waves, 369
triangle waves, 369
Positive Transition Filter, 385, 388
Power-On Configuration, 48, 472-474
Preparation For Use, 19
Primary HP-IB Address, 22
Program
frequency control, 489-494
introductory, 46
languages, 29
register-based, 483-508
Program Comments
arbitrary waveforms, 113-116
arming and triggering, 197-201
high speed operation, 280
marker outputs, 222
multiple AFG operations, 222
standard waveforms, 78
sweeping and FSK, 154-162
Pseudo-Noise
generating, 112
sweeping, 141-143
*PUD, 422
*PUD?, 422

Q
QSSG_RQS Example Program, 433-434
Query
A24 address space offset, 407
calibration, 298
error queue, 389
frequency list length, 360
marker list size, 222
operation condition register, 383
operation event register, 384  
parameter settings, 287  
power-on/reset configuration, 48  
questionable signal condition register, 386  
questionable signal event register, 387  
repetition count list length, 116, 353  
SCPI version number, 390  
segment sequence address, 347  
segment sequence combined list, 349  
segment sequence list length, 357  
segment sequence list names, 116, 348  
segment sequence marker pulse list, 355  
segment sequence memory, 353  
virtual instrument capacity, 408  
VXIbus local bus test data, 406  
waveform segment address, 336  
waveform segment combined list, 338  
waveform segment marker pulse list, 343  
waveform segment memory, 341  
waveform segment names, 116, 336  
waveform segment size, 116  
waveform segment voltage point list, 347  

Questionable Signal Status  
condition register, 386, 431  
enable register, 386, 432  
event register, 386, 432  
event register, query contents, 387  
group, 431-434  
negative transition filter, 387, 431, 435  
positive transition filter, 388  
register, 381, 431-434  
summary bit, 386  
using, 432-434  

Quick Reference  
common commands, 428  
SCPI commands, 409-413  

R  

Ramp Waves  
doubling frequency, 156  
generating, 65-68  
minimum frequency, 155  
number of waveform points, 157, 368  
polarity, 369  

Ranges  
DDS frequency generator, 79  
frequency, 155  
frequency 1 generator, 116  
*RCL, 423  

Reader Comment Sheet, 17  

Reading  
condition register, 431, 435  

error queue, 389  
offset register, 485-486  
standard event status register, 439  
status byte status register, 442  

REAL Data Format, 358  
Rectified Sine Waves, generating, 111  
Ref/Sample In BNC  
trigger slope, 395, 398  

Reference Oscillator  
description, 448  
frequency, 370  

sources, 78, 115, 154, 197, 371  

Register-Based Programming, 483-508  
accessing registers, 484  
changing output frequency, 487  
changing signal phase, 495-497  
frequency control programs, 489-494  
frequency control registers, 487-488  
offset register, reading, 485-486  
phase control program, 496-497  
selecting waveform sequence, 498-505  
system configuration, 484  

Registers  
accessing, 484, 486  
condition register, 431, 435  
enable register, 383, 386, 432, 436  
event register, 432, 436  
event register, query contents, 384, 387  
event register, summary bit, 383, 386  
frequency control, 487-488  
frequency load strobe, 488  
high speed data, 506  
offset, reading, 485-486  
operation status, 381, 436-437  
operation status group, 435  
phase control, 495  
phase increment, 487  
phase load strobe, 495  
phase modulation, 495  
questionable signal status, 381, 431-434  
ROSC/N divider, 488  
sample/hold and ROSC/N control, 488  
sequence base, 499  
standard event status, 439  
standard event status enable, 440  
standard event status group, 439-441  
status byte, 442  
status byte status group, 442  
status register, 499  
traffic register, 498  
transition filter, 431, 435  
waveform select, 499  
waveform sequence, 498-499  

*RCL, 423
Removing Modules, 25
Repetition Count, 251
  segment sequence list, 353
  segment sequence output, 352
Repetition Frequency, determining, 87
Reset Configuration
  list, 472-474
  query, 48
Reset Configuration Query, 48
Resetting AFG, 47
  example program, 47
Returning
  ASCii data format, 335, 358
  definite block data format, 335, 358
  PACKed data format, 335
  REAL data format, 358
  repetition count list length, 116
  segment sequence list names, 116
  waveform segment names, 116
*RMC, 423
ROSC/N Divider Register, 488
*RST, 47, 424
RSTCLS Program Example, 47
RSTSINE Program Example, 51

S

Safety Warnings, 14
Sample
  frequency range, 155
  gate, disabling, 394
  gate, enabling, 394
  gate, polarity, 393
  gate, source, 393
  programs, 104-109, 111-112
  rate, 87, 160
  rate, arbitrary waveforms, 331, 454
  rate, characteristics, 454-456
  rate, frequency span, 327
  rate, FSK source, 324
  rate, non-swept, 322
  rate, setting center, 321
  rate, setting FSK, 323
  rate, starting, 328
  rate, stopping, 329
  sources, arbitrary waveforms, 115
  sources, frequency lists, 154
  sources, FSK, 154
  sources, standard waveforms, 78
  sources, sweeping, 154
Sample/Hold and ROSC/N Control Register, 488
*SAV, 424
SCPI
  conformance information, 414-415
  instrument language, 26
  programming, 26
  version number, 390
SCPI Commands, 281
  abbreviated, 285
  ABORt subsystem, 290
  arbitrary block parameters, 286
  ARM subsystem, 291-297
  boolean parameters, 286
  CALibration subsystem, 298-305
  command separator, 285
  conformance information, 414-415
  coupling, 27-28, 288, 467-469
  discrete parameters, 286
  execution, 288
  format, 284-285
  implied, 27
  implied (optional), 285
  INITiate subsystem, 306-307
  linking, 27, 288
  numeric parameters, 286
  optional, 27
  OUTPut[1] subsystem, 308-311
  parameters, 286-287
  quick reference, 409-413
  reference, 281-312
  root keyword, 26
[SOURce:] subsystem, 312-380
[SOURce:]ARBitrary subsystem, 313-318
[SOURce:]FREQuency[1] subsystem, 319-329
[SOURce:]FREQuency2 subsystem, 330-331
[SOURce:]FUNCtion subsystem, 332-333
[SOURce:]LIST[1] subsystem, 334-357
[SOURce:]LIST2 subsystem, 358-360
[SOURce:]MARker subsystem, 361-364
[SOURce:]PM subsystem, 365-367
[SOURce:]RAMP subsystem, 368-369
[SOURce:]ROSCillator subsystem, 370-371
[SOURce:]SWEep subsystem, 372-376
[SOURce:]VOLTage subsystem, 377-380
  square bracket description, 287
STATus subsystem, 381-388
  structure, 26
SYSTem subsystem, 389-390
TRIGger subsystem, 391-402
  variable command syntax, 285
  vertical line, description, 286, 288
VINStrument subsystem, 403-408

Secondary HP-IB Address, 22
Security Code
  disabling, 303
  enabling, 303
  setting, 302
Segment
  data, downloading into DAC, 269-271, 506-508
  data, downloading into memory, 259-268
  data, downloading using backplane, 259
  list, combined, 239-250
  list, combined waveform format, 259
  list, determining size, 280
  list, multiple, 207-211
  marker pulses, determining number, 222
  marker pulses, multiple, 207-211
  marker pulses, single, 212-213
  memory, characteristics, 453
  memory, determining amount, 113
  memory, freeing, 113
  number of points, 87
  sample rate, 87
  sequence list, combined, 250
  sequence list, combined format, 250, 260
  sequence list, determining size, 280
  sequence list, query length, 357
  sequence, combined list, 348-349
  sequence, data, 316
  sequence, defining, 357
  sequence, defining outputs, 352-353
  sequence, deleting from memory, 351
  sequence, marker pulses, 354-355
  sequence, query list names, 116, 348
  sequence, query memory, 347, 353
  sequence, query repetition count list, 116
  sequence, reserving memory for, 350
  sequence, selecting, 356
  sequence, sending, 114
  sequence, storing in memory, 87
  single, marker pulses, 212-213
  single, waveform, 212-213
  using different segments, 93-98
  waveform, deleting from memory, 340
  waveform, marker pulses, 337-338, 342-343
  waveform, naming, 86
  waveform, output voltage, 337-338, 345-347
  waveform, query memory, 336, 341
  waveform, query names, 116, 336
  waveform, reserving memory for, 339
  waveform, selecting, 251, 344
  waveform, storing in memory, 86
Select Code, 22
Selecting
  amplitude levels, 72-74
  DAC data source, 315
  deviation units, 80, 365, 367
  marker enable, 251
  output, 73-74
  output loads, 69-71, 310-311
  output units, 72
  repetition count, 251
See also Setting
  waveform segments, 251, 344
  waveform sequence, 498-505
Self-Test, 46
  codes, 46
  example program, 47
Sequence
  base register, 499
  memory, characteristics, 454
  memory, determining amount, 113
  memory, freeing, 113
  segment, combined list, 348-349
  segment, defining, 357
  segment, defining outputs, 352-353
  segment, deleting from memory, 351
  segment, determining marker pulses, 222
  segment, marker pulses, 354-355
  segment, query list length, 357
  segment, query list names, 116, 348
  segment, query memory, 347, 353
  segment, query repetition count list, 116
  segment, reserving memory for, 350
  segment, selecting, 356
  segment, sending, 114
  segment, storing in memory, 87
  selecting, 498-505
  selection program, 500-505
Servant Area
  setting, 23
  switch, 23
Setting
  AFG bus request level, 24
  arming sources, 166-168, 294
  arming sweeps, 295-296, 372
  arming sweeps, sources, 297
  frequency lists, 124-126
  logical address switch, 22
  number of arms, 169-171
  output amplitude, 72-74, 377-380
  output impedance, 69-71, 309
  phase modulation deviation, 76, 365
  security code, 302
See also Selecting
  servant area, 23
  servant area switch, 23
  start arm slope, 293
  start arm source, 294
start trigger slope, 395
stop trigger slope, 398
waveform cycles per arm, 169-171
SIGN_DAT Example Program, 227-228

Signal
marker, disabling, 362, 364
marker, enabling, 362, 364
marker, polarity, 364
phase and gating, 200
phase, changing with registers, 495
questionable status register, 381, 431-434

Signed
data, combined, 239-244
data, generating waveforms with, 225-228
number, format, 225-226, 240

Sin(X)/X Waveforms, 105-106
sweeping, 141-143

SIGN_D Example Program, 107-108
SIGN_R Example Program, 111
SIGN_X Example Program, 105-106

Sine Waves
generating, 51, 58-60, 450
generating, damped, 107-108
generating, half rectified, 111
generating, noise, 112
generating, spiked, 109
minimum frequency, 155
number of waveform points, 157
See also Sinusoid

SINEWAVE Example Program, 59-60

Single
marker pulses, 212-213
waveform segments, 212-213
Single Point Marker, 343, 355

Sinusoid
function requirements, 78
phase modulation, 75, 365-367
waveforms, generating, 450

Sinusoid Waves
See Sine Waves

SLFTST Program Example, 47

Slope
eexternal stop trigger, 200
start arm, setting, 293
start trigger, setting, 395
stop trigger, setting, 398
SMPLSWP1 Example Program, 122-123
SMPLSWP2 Example Program, 128-129

Soft Front Panel
See VXIplug&play Online Help
[SOURce:] Subsystem, 312-380
[SOURce:]ARBitrary Subsystem, 313-318
[:DAC:SOURce, 315
[:DOWNload, 316-317
[:DOWNload:COMPLETE, 318
[SOURce:]FREQuency[1] Subsystem, 319-329
[:CENTer, 321
[:CW]:FIXed, 322
:FSKey, 323
:FSKey:SOURce, 324
:MODE, 325
:RANGe, 326
:SPAN, 327
:STARt, 328
:STOP, 329
[SOURce:]FREQuency2 Subsystem, 330-331
[:CW]:FIXed, 331
[SOURce:]FUNCTION Subsystem, 332-333
[:SHAPE], 332
[:USER, 333
[SOURce:][:LIST[1] Subsystem, 334-357
:FORMat[:DATA], 335
[:SEGment]:ADDRes?, 336
[:SEGment]:CATalogs?, 336
[:SEGment]:COMBined, 337
[:SEGment]:COMBined:POINts?, 338
[:SEGment]:DEFine, 339
[:SEGment]:DELETE:ALL, 340
[:SEGment]:DELETE:SELeected, 340
[:SEGment]:FREE?, 341
[:SEGment]:MARKer, 342
[:SEGment]:MARKer:POINts?, 343
[:SEGment]:MARKer:SPoint, 343
[:SEGment]:SELECT, 344
[:SEGment]:VOLTage, 345-346
[:SEGment]:VOLTage:DAC, 346
[:SEGment]:VOLTage:POINts?, 347
:SEQUence:ADDRes?, 347
:SEQUence:CATalogs?, 348
:SEQUence:COMBined, 348
:SEQUence:COMBined:POINts?, 349
:SEQUence:DEFine, 350
:SEQUence:DELETE:ALL, 351
:SEQUence:DELETE:SELeected, 351
:SEQUence:DWELl:COUNT, 352
:SEQUence:DWELl:COUNT:POINts?, 353
:SEQUence:FREE?, 353
:SEQUence:MARKer, 354
:SEQUence:MARKer:POINts?, 355
:SEQUence:MARKer:SPoint, 355
:SEQUence:SELECT, 356
:SEQUence:SEQUence, 357
:SEQUence:SEQUence:SEGments?, 357
[SOURce:][:LIST2 Subsystem, 358-360
:FORMat[:DATA], 358
:FREQuency, 359  
:FREQuency:POINts?, 360  
[SOURce:]MARKer Subsystem, 361-364  
:ECLTrgn:FEED, 361  
:ECLTrgn[:STATe], 362  
:FEED, 363  
:POLarity, 364  
[:STATe], 364  
[SOURce:]PM Subsystem, 365-367  
[:DEViation], 365  
:SOURce, 366  
:STATe, 367  
:UNIT[:ANGLe], 367  
[SOURce:]RAMP Subsystem, 368-369  
:POINts, 368  
:POLarity, 369  
[SOURce:]ROSCillator Subsystem, 370-371  
:FREQuency:EXTernal, 370  
:SOURce, 371  
[SOURce:]SWEepe Subsystem, 372-376  
:COUnT, 372  
:DIRection, 373  
:POINts, 374-375  
:TIME, 376  
[SOURce:]PM Subsystem, 377-380  
[:LEV][:IMM][:AMPL], 377-378  
[:LEV][:IMM][:AMPL]:UNIT[:VOLT], 379  
[:LEV][:IMM]:OFFSet, 380  
Sources  
arming, 199  
arming, setting, 166-168, 294  
DAC, 280, 315  
downloading, 280  
FSK control, 161  
FSK sample rate, 324  
gating, 200, 393  
marker, available, 205-206, 361-363  
phase modulation data, 366  
reference oscillator, 78, 115, 154, 197, 371  
sample, arbitrary waveform, 115  
sample, frequency lists, 154  
sample, FSK, 154  
sample, standard waveform, 78  
sample, sweeping, 154  
start trigger, 396  
stop trigger, 199, 398  
sweeping, setting, 297  
Span  
frequencies, sweeping with, 127-129, 327  
Specifying  
frequency lists, 124-126  
sweep times, 158  
Speed  
comparisons, 224  
increasing, 223-280  
Spiked Sine Waves, generating, 109  
SPIKES Example Program, 109  
Square Waves  
doubling frequency, 155  
generating, 61-64  
minimum frequency, 155  
number of waveform points, 157  
polarity, 369  
SQUWAVE Example Program, 63-64  
*SRE, 425  
*SRE?, 425  
Standard Event Status  
enable register, 440  
group, 439-441  
register, 439  
Standard Waveforms  
commands flowchart, 54-55  
frequencies, setting, 331  
generating, 53-82  
generating, ramp waves, 65-68  
generating, sine waves, 58-60  
generating, square waves, 61-64  
generating, triangle waves, 65-68  
phase modulation, data sources, 366  
phase modulation, default angle units, 367  
phase modulation, disabling, 367  
phase modulation, enabling, 367  
phase modulation, selecting deviation units, 80, 365  
phase modulation, using, 75-77  
program comments, 78  
selecting, amplitude levels, 72-74  
selecting, output loads, 69-71  
selecting, output units, 72-74  
Start  
arm slope, setting, 293  
arm source, setting, 294  
DC calibration, 300-301  
frequencies, 121-123  
frequencies, setting, 328  
frequencies, sweeping with, 127-129  
trigger slope, setting, 395  
trigger sources, 396  
Start Arm In BNC  
arm slope, 293  
Status  
register, 499  
system registers, 429-430  
Status Byte  
register, 442  
status group, 442  
Status Register, 429-430
description, 381
operation, 381
operation status group, 435-437
questionable signal, 381
questionable signal group, 431-434
standard event status group, 439-441
status byte status group, 442
STATus Subsystem, 381-388
STAT:OPC:INITiate, 382
STAT:OPERation:CONDition?, 383
STAT:OPERation:ENABle, 383
STAT:OPERation[:EVENt]?, 384
STAT:OPERation:NTRansition, 384
STAT:OPERation:PTRansition, 385
STAT:PRESet, 385
STAT:QUEStionable:CONDition?, 386
STAT:QUEStionable:ENABle, 386
STAT:QUEStionable[:EVENt]?, 387
STAT:QUEStionable:NTRansition, 387
STAT:QUEStionable:PTRansition, 388
*STB?, 426
Stop
arm cycle, 397-398
frequencies, 121-123
frequencies, setting, 329
trigger slope, setting, 398
trigger sources, 398
triggers, external slope, 200
triggers, sources, 199
triggers, using, 180-182, 196
Stop Trig/FSK/Gate In BNC
gating polarity, 393
STOPTRIG Example Program, 181-182
Subsystems (SCPI Commands)
ABORt, 290
ARM, 291-297
CALibration, 298-305
INITiate, 306-307
OUTPut[1], 308-311
[SOURce:], 312-380
[SOURce:]ARBitrary, 313-318
[SOURce:]FREQuency[1], 319-329
[SOURce:]FREQuency2, 330-331
[SOURce:]FUNCtion, 332-333
[SOURce:]LIST[1], 334-357
[SOURce:]LIST2, 358-360
[SOURce:]MARKer, 361-364
[SOURce:]PM, 365-367
[SOURce:]RAMP, 368-369
[SOURce:]ROSCillator, 370-371
[SOURce:]SWEep, 372-376
[SOURce:]VOLTage, 377-380
STATus, 381-388
SYSTem, 389-390
TRIGger, 391-402
VINstrument, 403-408
Summing Amplifier/DAC
See HP E1446A
Suspending Output Waveforms, 183
Sweep
advance source, 158
advance trigger, 201, 399
advance trigger, linking, 400
advance trigger, source, 401
advance trigger, time, 402
arm, linking, 296
arming, 190-192, 201, 295-297, 372
count, 156
count, setting, 295, 372
direction, 157, 373
linear, 375
points, 374
points versus time, 135-137, 159
sources, setting, 297
spacing, 157
time, 158, 376
time, specifying, 158
time, versus points, 135-137, 159
triggering, 190-192, 399-402
using triggers, 186-189
with output leveling, 30-31
Sweeping
and frequency lists, 120
and frequency-shift keying, 117-162
arbitrary waveforms, 141
frequency range, 155
logarithmic frequency spacing, 133-134
program comments, 154-162
pseudo-random noise, 141-143
sin(x)/x, 141-143
using start and span frequencies, 127-129
using start and stop frequencies, 121-123
Switches
logical address, 22
servant area, 23
SWP_ARB Example Program, 141-143
SWP_LEVL, 30-31
SWP_LEVL Example Program, 145-146
SWP_PVST Example Program, 136-137
SWP_STEP Example Program, 191-192
SWP_TRIG Example Program, 188-189
System Configuration
HP BASIC programming, 29
Visual BASIC programming, 32
Visual C/C++ programming, 40
SYSTem Subsystem, 389-390
SYStem:ERRor?, 389
SYStem:VERSion?, 390

T

Time
sweep advance trigger, 402
sweeps, 376
versus frequency lists, 138-140, 159
versus points, 135-137, 159
Traffic Register, 498

Transferring
 calibration constants, 299-300
 combined list, in signed number format, 240
 combined list, in unsigned number format, 245
 DAC codes, in signed number format, 225, 240
 DAC codes, in unsigned number format, 229, 245
data, 32-bit integer in HP BASIC, 255
data, using definite length arbitrary blocks, 231-234
data, using indefinite length arbitrary blocks, 235-238

Transition Filters
 negative, 384, 387, 431, 435
 operation status group, 435
 positive, 385, 388, 431, 435
 questionable signal status group, 431

*TRG, 426

Triangle Waves
doubling frequency, 156
generating, 65-68
minimum frequency, 155
number of waveform points, 157, 368
polarity, 369

Trigger
circuitry description, 450
gating signals, 183-185
lines, ECLTrgn, 204-205, 361-362
lines, TTLTrgn, 150-151, 162
slope, setting, 395, 398
stop triggers, external slope, 200
stop triggers, sources, 199
stop triggers, using, 180-182, 196
system, ABORt command, 290
system, ARM commands, 291-297
system, INITiate command, 306-307

TRIGger Subsystem, 391-402
TRIG[:START]:COUNt, 392
TRIG[:START]:GATE:POLarity, 393
TRIG[:START]:GATE:SOURCe, 393
TRIG[:START]:GATE:STATe, 394
TRIG[:START][:IMMediate], 395
TRIG[:START]:SLOPe, 395
TRIG[:START]:SOURCe, 396
TRIG:STOP[:IMMediate], 397
TRIG:STOP:SLOPe, 398
TRIG:STOP:SOURCe, 398
TRIG:SWEep[:IMMediate], 399
TRIG:SWEep:LINK, 400
TRIG:SWEep:SOURCe, 401
TRIG:SWEep:TIMer, 402

Triggering
 and arming, 163-202
 and arming, a frequency list, 193-195
and arming, a frequency sweep, 190-192
and arming, frequency sweeps and lists, 186-189
and arming, immediately, 201
and arming, program comments, 197-201
commands, 172, 391-402
the AFG, 172-185

TRIWAVE Example Program, 67-68

*TST?, 46, 426

TTLTrgn
 control source, 150-151
 functions, 460
trigger lines, driving the, 162

U

UNS_DAT Example Program, 230

Unsigned
data, combined, 245-249
data, generating waveforms with, 229-230
number, format, 229, 245

Useful Tables, 463-482
 amplitude limits, 471
command coupling groups, 467-469
error messages, 475-479
error messages, settings conflict, 480-482
equipped program listing, 464-466
frequency limits, 470
power-on/reset configuration, 472-474

Using
 ABORt, 196
arbitrary blocks, 130-132
arbitrary blocks, definite length, 231-234
arbitrary blocks, indefinite length, 235-238
arbitrary waveforms, 152-153
combined waveform segments, 250-258
combined waveform sequences, 250-258
DAC codes to send data, 114
different frequency generators, 99-102, 104
digital port in connector, 272-279
divide-by-n frequency generator, 174-175
embedded controller, 23
FSK control source, 147-149
gating, 196
HP E1446A with AFG, 23
maximum parameters, 81
minimum parameters, 81
operation status group, 436-437
phase modulation, 75-77, 80, 365-367
questionable signal status group, 432-434
signed data, combined, 239-244
signed data, number format, 225-228, 240
signed data, to generate waveforms, 225-228
standard event status group, 440-441
start and span frequencies, 127-129
start and stop frequencies, 121-123
stop triggers, 180-182, 196
TTLTrgn control source, 150-151
unsigned data, combined, 245-249
unsigned data, number format, 229-230, 245
unsigned data, to generate waveforms, 229-230
VXIbus backplane, 259-271, 506-508

V

Variable Command Syntax, 285
Virtual Instrument Subsystem, 403-408
  VINS[:CONF]:LBUS[:MODE], 403
  VINS[:CONF]:LBUS[:MODE]:AUTO, 404
  VINS[:CONF]:TEST:CONF, 405
  VINS[:CONF]:TEST:DATA?, 406
  VINS[:CONF]:VME[:MODE], 406
  VINS[:CONF]:VME:REC:ADDR:READ?, 407
  VINS:IDENtity?, 408
Virtual Instrument Commands, 403-408
Visual BASIC Language Programs, 32-39
  ARBWAVE.FRM, 33-39
  list of, 464-466
  running a program, 32
  system configuration, 32
  using SICL, 32
Visual C/C++ Language Programs, 40-45
  ARBWAVE.C, 41-45
  compiler used, 40
  list of, 464-466
  running a program, 40
  system configuration, 40
  using HP SICL, 40
VME Register Access, 461
Voltage
  generating DC, 56-57
  list, amplitude effect on, 113
  output, offset, 380
  output, waveform segment, 337-338, 345-347
VXIplug&play Example Programs
  See VXIplug&play Online Help
VXIplug&play Function Reference

See VXIplug&play Online Help
VXIplug&play Programming
  See VXIplug&play Online Help
VXIplug&play Soft Front Panel
  See VXIplug&play Online Help
VXIbus
  A24 address space query, 407
  backplane, downloading segment data, 259-268
  backplane, using, 259-271, 506-508
  characteristics, 461
  data transfer bus, operating mode, 406
  ECL trigger lines, 361-362
  factory settings, 20
  loading DAC from, 506-508
  local bus, operating mode, 403-404
  local bus, testing, 405-406
VXIDOWN Example Program, 264-268
VXISRCE Example Program, 270-271, 506-508

W

*WAI, 427
Wait-for-Arm State, 164
Wait-for-Trigger State, 164
WARNINGS, 14
Warranty, 13
WAVE_SEL Example Program, 500-505
Waveform Select Register, 499
Waveforms
  aborting, 196
  arming immediately, 293
  FSK using arbitrary, 152-153
  generating arbitrary, 83-116
  generating arbitrary, command flowchart, 84-85
  generating arbitrary, damped sine waves, 107-108
  generating arbitrary, description, 86-87
  generating arbitrary, exponential charge/discharge waveform, 108-109
  generating arbitrary, half rectified sine waves, 111
  generating arbitrary, noise, 112
  generating arbitrary, sample programs, 104-109, 111-112
  generating arbitrary, sample rate, setting, 331
  generating arbitrary, simple, 88-92
  generating arbitrary, sin(x)/x, 105-106
  generating arbitrary, spiked sine waves, 109
  generating arbitrary, with different frequency generators, 99-102, 104
  generating arbitrary, with different waveform segments, 93-98
  generating arbitrary, with single waveform segment, 88-92
  generating standard, 53-82
generating standard, command flowchart, 54-55
generating standard, frequencies, setting, 331
generating standard, ramp waves, 65-68
generating standard, selecting amplitude levels, 72-74
generating standard, selecting output loads, 69-71
generating standard, selecting output units, 72-74
generating standard, sine waves, 58-60
generating standard, square waves, 61-64
generating standard, triangle waves, 65-68
generating, using signed data, 225-228
generating, using unsigned data, 229-230
initiating, 165
outputting, 86-87
phase modulation, data sources, 366
phase modulation, default angle units, 367
phase modulation, disabling, 367
phase modulation, enabling, 367
phase modulation, selecting deviation units, 80, 365
point, marker pulses for each, 214-217
repetition count, 199, 291
repetition frequency, determining, 87
repetition per start arm, 291
segment, data, 316
segment, deleting from memory, 340
segment, determining marker points, 222
segment, determining memory, 113
segment, freeing memory, 113
segment, marker pulses, 337-338, 342-343
segment, multiple marker pulses, 207-211
segment, naming, 86
segment, number of points, 87
segment, output voltage, 337-338, 345-347
segment, query memory, 336, 341
segment, query names, 116, 336
segment, query size, 116
segment, reserving memory for, 339
segment, sample rate, 87
segment, selecting, 251, 344
segment, single marker pulses, 212-213
segment, storing in memory, 86-87
segment, using combined, 250-258
segment, using combined sequence, 250-258
segment, using different, 93-98
sequence, determining marker points, 222
sequence, determining memory, 113
sequence, freeing memory, 113
sequence, registers, 498-499
sequence, selecting, 498-505
sequence, using combined, 250-258
setting cycle count, 169-171
shape selection, 332
sweeping arbitrary, 141
using DDS generator, 99-102

using divide-by-n generator, 99-101, 104, 174-175
WAVSELF Example Program, 272-277