

Contents

HP E1340A Arbitrary Function Generator User's Manual

Warranty	9
WARNINGS	10
Safety Symbols	10
Declaration of Conformity	11
Reader Comment Sheet	13
1. Getting Started with the HP E1340A	15
Chapter Contents	15
Preparation for Use	15
Installing the AFG	15
Downloading the AFG Device Driver	15
AFG Logical Address	17
Addressing the AFG	18
Instrument and Programming Languages	19
SCPI Programming	19
SCPI Command Structure	19
Command Coupling	20
How to Execute Coupled Commands	21
HP IBASIC Language Programs	22
C Language Programs	25
QuickBASIC Language Programs	31
Introductory Programs	36
AFG Self-Test	36
Resetting and Clearing the AFG	37
Querying the Power-On/Reset Configuration	37
Checking for Errors	38
Generating a Sine Wave	40
HP E1340A Example Programs	40
2. Generating Standard Waveforms with the HP E1340A	41
Chapter Contents	41
Standard Waveforms Flowchart	41
Generating DC Voltages	43
C and QuickBASIC Program Versions	44
Generating Sine, Triangle, and Square Waves	45
C and QuickBASIC Program Versions	46
Generating Ramp Waves	47
C and QuickBASIC Program Versions	49
Selecting the Amplitude Levels and Output Units	50
C and QuickBASIC Program Versions	51
Program Comments	52
Reference Oscillator Sources	52
Output Load Comments	52

Output Units Comments	52
Using MINimum and MAXimum Parameters	53
3. Generating Arbitrary Waveforms with the HP E1340A	55
Chapter Contents	55
Arbitrary Waveforms Flowchart	55
How the AFG Generates Arbitrary Waveforms	56
Generating a Simple Arbitrary Waveform	59
C and QuickBASIC Program Versions	62
Executing Several Waveform Segments	65
C and QuickBASIC Program Versions	68
Arbitrary Waveform Hopping	69
C and QuickBASIC Program Versions	72
Generating Built-In Arbitrary Waveforms	73
C and QuickBASIC Program Versions	75
Sample Programs	76
Generating a Damped Sine Wave	76
Generating an Exponential Charge/Discharge Waveform	78
Generating a Sine Wave with Spikes	79
Generating a Half-Rectified Sine Wave	80
Program Comments	82
Amplitude Effects on Voltage List	82
Reference Oscillator Sources	82
Waveforms in the EPROM	82
4. HP E1340A Sweeping and Frequency-Shift Keying	83
Chapter Contents	83
The Command Reference	83
Programming Flowchart	83
Sweeping	85
Sweeping Considerations	85
Sweeping Using Start and Stop Frequencies	85
C and QuickBASIC Programs	87
Sweeping Using Start and Span Frequencies	88
C and QuickBASIC Programs	89
Sweep Points Vs. Sweep Time	90
C and QuickBASIC Programs	91
Frequency-Shift Keying	92
Program Example	92
Program Comments	94
Reference Oscillator Sources	94
AFG Frequency Modes	94
Frequency Points	94

5. HP E1340A Arming and Marker Outputs	95
Chapter Contents	95
The ARM Configuration	95
The ARM States	95
Initiating Waveforms	96
Arming the AFG	96
Arming Commands	96
Setting the	
Arm Source	97
Setting the Arm and Waveform Cycle Count	99
Gating the Waveforms	101
Aborting Waveforms	102
Marker Outputs	103
Marker Commands	103
Available Marker Sources	103
Generating Zero Crossing Marker Pulses	104
Program Comments	107
Reference Oscillator Sources	107
AFG Frequency Modes	107
AFG Arming Sources	107
AFG Arm Count	107
Waveform Repetition Count	107
Enabling the Gate	108
6. HP E1340A High Speed Operation	109
Chapter Contents	109
Using DAC Data to Generate Waveforms	109
Program Example	110
Using Definite Length Blocks to Transfer Data	113
Definite Length Block Data Format	113
Data Byte Size	113
Program Example	114
Using the VXIbus Backplane	120
Using the VXIbus Backplane to Download Segment Data	120
Downloading Segment Data into Memory	120
Downloading Directly into the DAC	120
Program that Downloads and Executes Waveform Segments in Memory	120
Program to Download Directly to the DAC	126
Program Comments	129
Amplitude Effects on DAC Codes	129
Incorrect AFG Operation from Incorrect DAC Codes	129

7. HP E1340A Command Reference	131
Chapter Contents	131
Command Types	132
Common Command Format	132
SCPI Command Format	133
Command Separator	133
Abbreviated Commands	133
Implied (Optional) Keywords	134
SCPI Command Parameters	134
Parameter Types, Explanations, and Examples	134
Optional Parameters	135
Querying	
Parameter	
Settings	136
SCPI Command Execution	136
Command Coupling	136
Linking Commands	137
SCPI Command Reference	137
ABORt	138
ARM	139
[:START][:LAYer[1]] :COUNT	139
[:START]:LAYer2 :COUNT	140
[:START]:LAYer2 :SLOPe	140
[:START]:LAYer2 :SOURce	141
CALibration	142
:AC:BEGin	142
:AC:POINt	143
:DATA:AC	143
:DATA[:DC]	143
[:DC]:BEGin	144
[:DC]:POINt	144
:STATe:AC	145
INITiate	146
[:IMMediate]	146
OUTPut	147
:IMPedance	147
:LOAD	148
[SOURce:]ARBitrary	149
:DAC:SOURce	149
:DOWNload	150
:DOWNload :COMplete	152
[SOURce:]FREQuency	153
:CENTer	155
[:CW]:FIXed]	155
:FSKey	156
:MODE	156
:SPAN	157
:STARt	158

:STOP	158
[SOURce:]FUNctIon	159
[:SHAPe]	159
:USER	160
[SOURce:]LIST	161
[:SEGment] :CATalog?	161
[:SEGment]:DEFine?	162
[:SEGment]:SElect	162
[:SEGment]:VOLTagE	163
[:SEGment]:VOLTagE:DAC	164
[:SEGment]:VOLTagE:POINts?	164
:SSEquence :CATalog?	165
:SSEquence :DEFine?	165
:SSEquence:SElect	165
:SSEquence :SEquence?	166
:SSEquence :SEquence :SEGments?	166
[SOURce:]MARKer	167
:FEED	167
:POLarity	168
[SOURce:]RAMP	169
:POLarity	169
[SOURce:]ROSCillator	170
:FREQuency :EXTernal	170
:GATE:STATe	171
:SOURce	171
[SOURce:]SWEep	173
:COUNT	174
:POINts	175
:TIME	175
[SOURce:]VOLTagE	177
[:LEVel][:IMMediate] [:AMPLitude]	177
[:LEVel][:IMMediate] [:AMPLitude]:UNIT [:VOLTagE]	179
[:LEVel][:IMMediate]:OFFSet	180
STATus	181
:OPERation QUEStionable :CONDition?	182
:OPERation QUEStionable :ENABle	182
:OPERation QUEStionable [:EVENT]?	183
:OPERation QUEStionable :NTRansition	183
:OPERation QUEStionable :PTRansition	184
:PRESet	184
SYSTem	185
:ERRor?	185
:VERsion?	185
TRIGger	186
[:START]:COUNT	186
[:START]:SOURce	187
IEEE-488.2 Common Commands	188

*CLS	189
*DMC	189
*EMC and *EMC?	190
*ESE and *ESE?	190
*ESR?	190
*GMC?	191
*IDN?	191
*LMC?	192
*LRN?	192
*OPC	192
*OPC?	193
*PMC	193
*RCL	193
*RMC	194
*RST	194
*SAV	194
*SRE and *SRE?	195
*STB?	195
*TST?	196
*WAI	196
SCPI Conformance Information	200
8. HP E1340A AFG Status	203
Introduction	203
Status System Registers	203
The Operation Status Group	205
The Standard Event Status Group	207
The Status Byte Status Group	208
Using the Standard Event Status Group	210
9. HP E1340A Block Diagram Description	211
Chapter Contents	211
AFG Description	211
What is an Arbitrary Waveform?	211
Generating Waveforms	212
Output DAC	212
Memory	213
Reference Oscillator	213
Frequency Generators	213
Output Circuitry	214
Microprocessor	214
Input/Output Connectors	214
A. HP E1340A Specifications	215
Appendix Contents	215
Instrument Specifications	215
General and VXI Characteristics	217

B. HP E1340A Useful Tables	219
Appendix Contents	219
C. HP E1340A Register-Based Programming	229
Appendix Contents	229
Register Addressing	229
The Base Address	229
Computer Configurations	232
Throughput Speed	232
Embedded Computer Programming (C-Size Systems)	232
IBASIC Programming	232
External Computer Programming	233
Programming Guidelines	234
Register Descriptions	234
The READ Registers	234
The ID Register	234
The Device Type Register	235
The Status Register	236
The Query Response Register	237
The WRITE Registers	237
The Control Register	237
The Command and Parameter Registers	238
The Input Data Register	238
Command Descriptions and Formats	239
Setting the Output Function and Loading RAM	243
Setting the Signal Frequency	244
Setting the Output Attenuation	245
Setting the Amplitude Offset	246
Setting the Sweep Rate	247
Setting the Burst Count	247
The AFG Output Mode	248
Starting the Waveform	251
Querying the AFG	251
AFG Soft Reset	252
Aborting the Waveform	252
Program Timing and Execution	252
AFG Reset Sequence	253
Configuring the AFG	254
Querying AFG Parameters	255
Example Programs	256
Opcode/Parameter Quick Reference	256
Generating a	
Sine Wave	258
Multiple Waveforms	259
Internally Triggering a Burst of Cycles	260
Externally Triggering a Burst of Cycles	261
Frequency-Shift Keying	262

Waveform Hopping	263
Sweeping	264
Gating the Output	265
Downloading an Arbitrary Waveform	266
Fast Frequency Changes	268
Sending Data Directly to the DAC	271
Example Program Subprograms	272
Querying AFG Parameters	280

Notes

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 prepay 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 circuit 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.

Restricted Rights Legend

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, California 94304 U.S.A.

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19 (c) (1,2).



HP E1340A Arbitrary Function Generator Module User's Manual
Edition 3
Copyright © 1995 Hewlett-Packard Company. All Rights Reserved.

Documentation History

All Editions and Updates of this manual and their creation date are listed below. The first Edition of the manual is Edition 1. The Edition number increments by 1 whenever the manual is revised. Updates, which are issued between Editions, contain replacement pages to correct or add additional information to the current Edition of the manual. Whenever a new Edition is created, it will contain all of the Update information for the previous Edition. Each new Edition or Update also includes a revised copy of this documentation history page.

Edition 1 January, 1992
Update 1 August, 1992
Edition 2 March, 1993
Edition 3 April, 1995

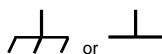
Safety Symbols



Instruction manual symbol affixed to product. Indicates that the user must refer to the manual for specific WARNING or CAUTION information to avoid personal injury or damage to the product.



Indicates the field wiring terminal that must be connected to earth ground before operating the equipment—protects against electrical shock in case of fault.



Frame or chassis ground terminal—typically connects to the equipment's metal frame.



Alternating current (AC).



Direct current (DC).



Indicates hazardous voltages.

WARNING

Calls attention to a procedure, practice, or condition that could cause bodily injury or death.

CAUTION

Calls attention to a procedure, practice, or condition that could possibly cause damage to equipment or permanent loss of data.

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: E1340A

Product Options: All

conforms to the following Product Specifications:

Safety: IEC 1010-1 (1990) Incl. Amend 1 (1992)/EN61010-1 (1993)
CSA C22.2 #1010.1 (1992)
UL 1244

EMC: CISPR 11:1990/EN55011 (1991): Group1 Class A
IEC 801-2:1991/EN50082-1 (1992): 4kVCD, 8kVAD
IEC 801-3:1984/EN50082-1 (1992): 3 V/m
IEC 801-4:1988/EN50082-1 (1992): 1kV Power Line
.5kV 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 and carries the CE-marking accordingly.

Tested in a typical configuration in an HP B-Size VXI mainframe.

April, 1995


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).

Notes

Please fold and tape for mailing

Reader Comment Sheet

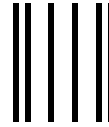
HP E1340A Arbitrary Function Generator Module User's Manual
Edition 3

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).

_____ Your Name	_____ City, State/Province
_____ Company Name	_____ Country
_____ Job Title	_____ Zip/Postal Code
_____ Address	_____ Telephone Number with Area Code

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

fold here

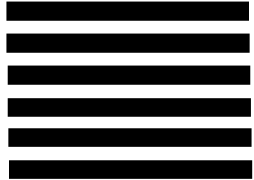


NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 37 LOVELAND, CO

POSTAGE WILL BE PAID BY ADDRESSEE

HEWLETT-PACKARD COMPANY
Measurement Systems Division
Learning Products Department
P.O. Box 301
Loveland, CO 80539-9984



cut along this line



fold here

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.

Disagree	←————→					Agree
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Please write any comments or suggestions below--be specific.

Chapter Contents

This chapter shows you how to configure, install, and begin using the HP E1340A Arbitrary Function Generator (AFG). The main sections of this chapter include:

- Preparation for Use Page 15
 - Installing the AFG Page 15
 - Downloading the AFG Device Driver Page 15
 - AFG Logical Address Page 17
 - Addressing the AFG Page 18
- Instrument and Programming Languages Page 19
 - HP IBASIC Language Programs Page 22
 - C Language Programs Page 25
 - QuickBASIC Language Programs Page 31
- Introductory Programs Page 36
 - AFG Self-Test Page 36
 - Resetting and Clearing the AFG Page 37
 - Querying the Power-On/Reset Configuration Page 37
 - Checking for Errors Page 38
 - Generating a Sine Wave Page 40
- HP E1340A Example Programs Page 40

Preparation for Use

This section shows how to prepare the AFG for use. Included are instructions to install the AFG, to download the AFG Device Driver, and to address the AFG.

Installing the AFG

The AFG can be installed in any mainframe slot, except slot 0. For installation instructions in a B-Size VXIbus System, refer to the *B-size Installation and Getting Started Guide*. For installation instructions in C-Size VXIbus Systems, refer to the *C-size Installation and Getting Started Guide*.

Downloading the AFG Device Driver

The HP E1340A AFG driver allows the AFG to operate in an HP E1300/E1301 Mainframe or HP E1405/E1406 Command Module using the Standard Commands for Programmable Instruments (i.e., SCPI) instrument command language. The driver is supplied on the AFG's example program disks (see "Instrument and Programming Languages" on page 19 for more information on the disks).

The procedure for downloading the driver is contained in the HP Installation Note *Downloading Device Drivers*. The *E1300/E1301 Mainframe Manual*, *HP E1405 Command Module Manual*, and *E1406 Command Module Manual* also contain the installation procedure.

Mainframe/Command Module Requirements

To download a driver, the ROM version number of the HP E1300/E1301 Mainframe or HP E1405 Command Module must be A.06.00 or above. To determine the version number, send the IEEE 488.2 common command *IDN?. A typical returned value (for the HP E1300A Mainframe) is:

```
HEWLETT-PACKARD , E1300A , 0 , A . 06 . 01
```

where "A.06.01" is the version number.

Recommended Systems to Download the Drivers

The AFG instrument driver can be downloaded with the following systems.

HP BASIC with an External HP-IB Controller

This system downloads the driver over the Hewlett-Packard Interface Bus (HP-IB*) with an HP BASIC controller, like the HP Series 200/300 computer. Be sure to connect the computer to the Mainframe's / Command Module's HP-IB port. The program to download the driver is named "VXIDLD_GET", and is in GET/SAVE format. The program is located on the HP IBASIC example program disk (HP part number E1340-10035). This disk is formatted in the LIF format.

HP IBASIC in an HP E1300/E1301 Mainframe or HP E1405/E1406 Command Module

This system downloads the drivers using the HP IBASIC (i.e., HP Instrument BASIC) program language. The language comes with options to the HP E1300/E1301 Mainframe and the HP E1405/E1406 Command Modules. IBASIC reads disks formatted in either the LIF or MS-DOS® format and can use either example program disk (dependent on your disk drive). To download the driver from the LIF disk or the DOS disk, use "AUTOST".

Externally Connected MS-DOS® Computer

This system downloads the driver using an MS-DOS computer over the RS-232 serial interface. Connect the computer to the Mainframe's / Command Module's RS-232 port. The program to download the driver is named "VXIDLD.EXE" and is located on the DOS example program disk (HP part number E1340-10036). This disk is formatted for MS-DOS.

* HP-IB is Hewlett-Packard's implementation of IEEE Std 488.1-1978

AFG Logical Address

The AFG is shipped from the factory with a logical address setting of 80, as shown in Figure 1-1.

The E1340A AFG logical address is used:

- to place the AFG in the servant area of a commander (e.g. HP E1405/E1406 Command Module).
- to address the AFG (see “Addressing the AFG” later in this chapter).

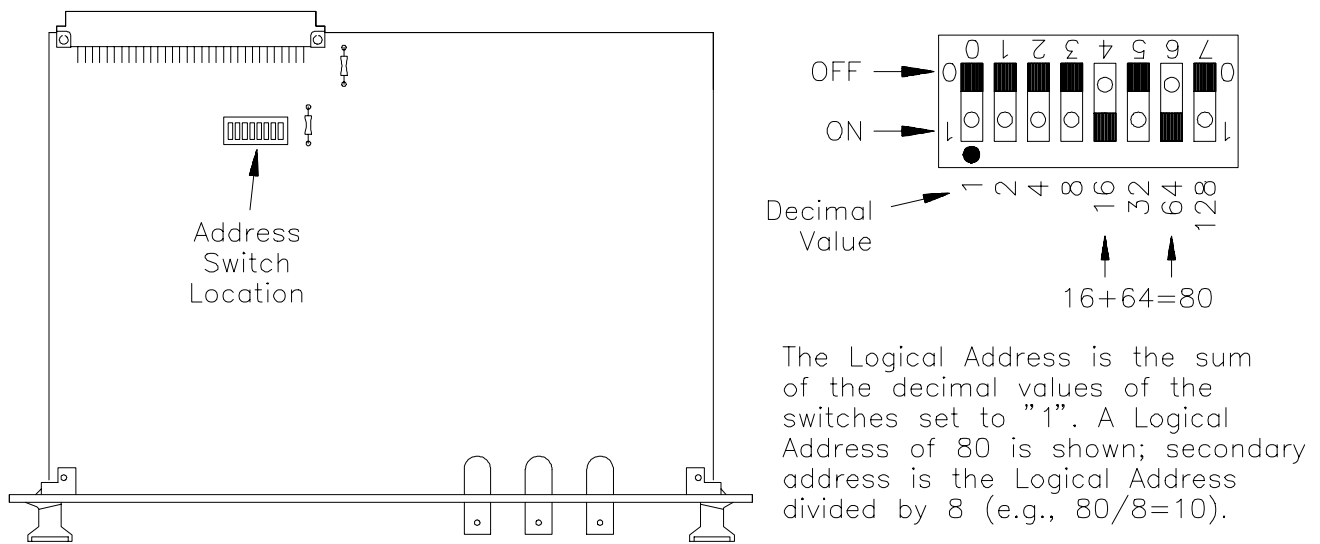


Figure 1-1. Setting the Logical Address

Assigning the AFG to a Commander

In VXibus systems, each device must be in the servant area of a commander. Note the following when assigning the E1340A AFG:

- In a B-size system, the system instrument is the commander. Any AFG logical address (1 - 255) places the AFG in the commander servant area. The AFG logical address should remain at 80, or if necessary, set to some other multiple of 8 (see “Addressing the AFG”).
- The E1340A AFG is a register-based device. If the AFG’s driver is downloaded in the HP E1405/E1406 Command Module, then the AFG must be in the servant area of the Command Module to program with SCPI commands.

Addressing the AFG

The following information explains how the AFG is addressed using different computer (controller) configurations.

IBASIC

To program the AFG in a B-size or C-size system using IBASIC, the AFG must have a unique address. The address is a combination of the IBASIC interface select code, the Mainframe’s/Command Module’s primary HP-IB address, and the AFG’s secondary HP-IB address. An address in this form in an HP IBASIC statement appears as:

```
OUTPUT 80910;"SOUR:ROSC:SOUR INT"
```

Interface Select Code (8): The IBASIC interface select code is 8. This select code is used to access all devices installed in B-size and C-size systems under IBASIC control.

Primary HP-IB Address (09): This is the address of the HP-IB port on the Mainframe or Command Module. Valid addresses are 0 to 30. The mainframe/command 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. For the E1340A AFG factory set logical address of 80, the secondary address is 10.

External Controller and PC

When programming the AFG from an external computer such as a HP Series 300 or PC, the addressing convention used is the same as IBASIC except for the interface select code. An example is shown below:

```
OUTPUT 70910;"SOUR:ROSC:SOUR INT"
```

Interface Select Code (7): Determined by the address of the HP-IB interface card in the computer. In most Hewlett-Packard computers, this card has a factory set address of 7, including the HP 82335 HP-IB Interface Card (this card was used with an HP Vectra personal computer to create the C and QuickBASIC example programs).

The **Primary HP-IB Address** and **Secondary HP-IB Address** are the same as described previously for IBASIC programming.

Instrument and Programming Languages

The purpose of this manual is to teach you how to use the E1340A AFG. To do this, the manual uses block diagrams, flowcharts, and example programs. The programs shown in the manual are SCPI programs written in HP IBASIC. These programs, and QuickBASIC and C language versions of the programs are on the following disks which ship with the manual:

- HP E1340A Instrument Driver and HP IBASIC Example Programs - 3.5" 720 kbyte disk (E1340-10035)
- HP E1340A Instrument Driver and QuickBASIC and C Language Example programs - 3.5" 1.44 Mbyte disk (E1340-10036)

SCPI Programming

SCPI (Standard Commands for Programmable Instruments) is an ASCII-based instrument command language designed for test and measurement instruments. The HP E1300/E1301 Mainframe or HP E1405/E1406 Command Module with the AFG driver installed interprets the ASCII command strings and sets the AFG accordingly. The Mainframe/Command Module does this by writing to the AFG registers.

SCPI Command Structure

The AFG SCPI command set is found in Chapter 7. 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 shown below.

```
ARM
  [:START|SEQUence[1]]
    [:LAYer[1]]
      :COUNT <number>
    :LAYer2
      :COUNT <number>
      :SLOPe <edge>
      :SOURce <source>
```

ARM is the root keyword of the command, :START|SEQUence1 is the second level keyword, :LAYer1 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:MODE <mode>
```

Command headers enclosed in brackets are optional. UPPER CASE letters in the header are required, lower case letters can be omitted. **To aid in learning the AFG command set, all headers are included in the example programs; however, the headers are abbreviated.** In an example program, the previous statement with a *mode* parameter of FIX would appear as:

```
SOUR:FREQ:MODE FIX
```

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 a "Coupling Group".

The AFG uses one coupling group. Commands not in the coupling group must precede or follow commands in the coupling group. Executing uncoupled commands in the coupling group breaks the coupling and can cause a "Settings Conflict" error.

The coupling group and associated commands can be found in Appendix B, Table B-2.

How to Execute Coupled Commands

Command coupling determines the AFG programming sequence. 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 (HP IBASIC programs only).

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 INT;;SOUR:FUNC:SHAP SIN
```

or

```
SOUR:ROSC:SOUR INT;  
:SOUR:FUNC:SHAP SIN
```

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” earlier in this chapter).

Note

See “HP IBASIC Language Programs” later in this chapter for information on suppressing the end-of-line terminator.

HP IBASIC Language Programs

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

System Configuration

Except where noted, the example programs in HP IBASIC were developed (and tested) on the following system:

Mainframe: HP E1301A Mainframe
(IBASIC Option)

HP E1340A Logical Address: 80

Instrument Language: SCPI

Program Structure

The structure of an example program in HP IBASIC is shown below:

```
1  !RE-SAVE"ARB_GEN"
2  !This program generates a 4096 point, 0 to 5V ramp waveform.
3  !The data is transferred to the AFG as voltages.
4  !
10 !Assign I/O path between IBASIC and the E1340A.
20 ASSIGN @Afg TO 80910
30 COM @Afg,Waveform(1:4096)
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 subprogram which resets the AFG.
130 CALL Rst
140 !Set AFG parameters
150 OUTPUT @Afg;"SOUR:ROSC:SOUR INT;"; !reference oscillator
160 OUTPUT @Afg;":SOUR:FREQ:FIX 1E3;"; !frequency
170 OUTPUT @Afg;":SOUR:FUNC:SHAP USER;"; !function
180 OUTPUT @Afg;":SOUR:VOLT:LEV:IMM:AMPL 5.1V"!scale
    amplitude
190 !
200 !Call subprogram which defines waveform segment.
210 CALL Ramp_wave
220 !
230 !Select output sequence and initiate waveform
240 OUTPUT @Afg;"SOUR:FUNC:USER A"
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 8
290  END
300  !
310  SUB Ramp_wave
320  Ramp_wave: !Subprogram which defines a ramp waveform
330      COM @Afg,Waveform(*)
340      FOR I=1 TO 4096
350          Waveform(I)=I*.00122
360      NEXT I
370  !
380      OUTPUT @Afg;"SOUR:LIST:SEGM:SEL A"!Select segment name
390      OUTPUT @Afg;" SOUR:LIST:SEGM:VOLT";Waveform(*)!load
                                                waveform points

400  SUBEND
410  !
420  SUB Rst
430  Rst: !Subprogram which resets the E1340.
440      COM @Afg,Waveform(*)
450      OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
460      ENTER @Afg;Complete
470  SUBEND
480  !
490  SUB Errmsg
500  Errmsg: !Subprogram which displays E1340 programming errors
510      COM @Afg,Waveform(*)
520      DIM Message$[256]
530      !Read AFG status byte register and clear service request bit
540      B=SPOLL(@Afg)
550      !End of statement if error occurs among coupled commands
560      OUTPUT @Afg;"
570      OUTPUT @Afg;"ABORT"                !abort output waveform
580      REPEAT
590      OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
600      ENTER @Afg;Code,Message$
610      PRINT Code,Message$

620      UNTIL Code=0
630      STOP
640  SUBEND

```

Turning Off (Suppressing) the End-Of-Line Terminator

As mentioned, 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 IBASIC programs, the EOL terminator is suppressed by placing a semicolon (;) following the quotation mark (“ ”) which closes the command string. In the program above, the commands in lines 150 - 180 are in the coupling group. The semicolons following the command strings in lines 150 through 170 suppress the EOL terminator; therefore lines 150 - 180 are sent as a single statement.

Getting Programs

The HP IBASIC programs are stored on the disk as ASCII files. The program name is shown in the program title and in line 1 of the program (e.g., 1 !RE-SAVE"ARB_GEN"). To get a program, type:

```
GET "file name"    (e.g., GET "ARB_GEN")
```

Declaring IBASIC Variables in COM (common) Memory

When writing or modifying IBASIC programs, array variables can be declared in COM (common) memory. Variables not in COM memory reside in the IBASIC stack. The 'stack' is a 32 kByte (default) segment of memory which contains components such as pointers and local variables for subprograms and declarations. When too many variables (or too large an array) are in the stack, Error 2 - Memory Overflow will occur. If a memory overflow occurs, the stack size can be changed with the command `PROGram:MALLocate <nbytes>` (see the *Instrument BASIC User's Manual* for more information).

C Language Programs

All of the C Language example programs in this manual are written for the HP 82335 HP-IB Interface Card using the HP-IB Command Library for C. Unless otherwise noted, the library functions used in the programs are compatible with the ANSI C standard.

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 C programs were developed on the following system:

Controller: HP Vectra 386/25 personal computer
(386 processor operated at 25 MHz)

HP-IB Interface Card: HP 82335 HP-IB Interface with
Command Library

Mainframe: HP 75000 Series B/C

Slot0/Resource Manager: HP E1301A Mainframe/E1405/E1406
Command Module

HP E1340A Logical Address:80

Instrument Language: SCPI

C Compiler Used

Unless otherwise noted, all C Language programs were compiled (and tested) using the following compilers:

- Microsoft[®] C Version 5.10
- Microsoft[®] QuickC[®] Version 2.0
- Borland Turbo C[®] Version 2.0
- Borland Turbo C++[®] Version 1.0

Running a Program

To run a program, first compile and link the program to make an executable file. To compile a program:

- Link the appropriate HP-IB C library (located on the HP-IB Command Library disk that came with the HP-IB Interface Card). Use the following libraries:
 - Microsoft[®] C and QuickC[®]: clhplib.lib
 - Turbo C[®] and C++[®]: tchplib.lib
- If NOT compiling in the Large/Huge memory model, include the “cfunc.h” header file in the program (located on the HP-IB Command Library disk that came with the HP-IB Interface Card).

- Be sure the necessary paths have been added to the AUTOEXEC.BAT file for the compilers to find the library and header files (see the appropriate C Language manual to set the proper paths).

To compile the programs from the DOS command line using the Large memory model, execute the following:

- Microsoft[®] C:

```
cl /AL <path \program name> path \clhplib.lib
```

- Microsoft[®] QuickC[®]:

```
qcl /AL <path \program name> path \clhplib.lib
```

- Turbo C[®] and Turbo C++[®]:

```
tcc -ml <path \program name> path \tchhplib.lib
```

Change the “/AL” and “-ml” parameters to the appropriate types when compiling in the smaller memory models (see the appropriate C Language manual for the parameter type).

C Program Example

Following is an example program written in C using the HP 82335 HP-IB Interface Card. The program:

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

```

/* ARB_GEN.C - This program generates a 4096 points ramp. The data to */
/* generate the ramp is transferred to the AFG as voltages */

/* Include the following header files */
#include <stdio.h>
#include <string.h>
#include <malloc.h> /* Use "alloc.h" for Turbo C(c) or C++(c) */
#include <cfunc.h> /* This file is from the HP-IB Command Library Disk */

#define ISC 7L /* Assigns the HP-IB select code */
#define ADDR 70910L /* Assign an I/O path between the computer and the AFG */

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

/*****/
void main(void) /* Run the program */
{
rst_clr(); /* Reset the AFG */
gen_seg(); /* Generate segment list and output sequence */
run_query(); /* Query waveform segment memory */
}
/*****/
void gen_seg(void)
{
char static *set_commands[] =/* Use "set_commands" to setup the AFG */
{
"SOUR:ROSC:SOUR INT;" /* Select the Ref. Oscillator */
":SOUR:FREQ:FIX 1e3;" /* Set waveform frequency */
":SOUR:FUNC:SHAP USER;" /* Command to select the user function */
":SOUR:VOLT:LEV:IMM:AMPL 5.1V", /* Set the amplitude */

"SOUR:LIST:SEGM:SEL a" /* Select the segment name */
},

```

Continued on next page

```

*seg_commands =          /* Use "seg_commands" to store segments */

"SOUR:LIST:SEGM:VOLT ", /* Command to send volts data */

*out_commands[] =       /* Use the "out_commands" array to generate output */
{
"SOUR:FUNC:USER a",     /* Select the user name */
"INIT:IMM"              /* Start waveform generation */
};

float  *Wave_seg;
int    loop,
      seg_size = 4096;    /* Set the segment size to 4096 points */

/* 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*));

/* 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 * .00122);

/* Call function to execute the final command with segment data */
send_data(seg_commands, Wave_seg, seg_size);

/* 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*));

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

```

Continued on next page

```

        /* Free the allocated memory */
free (Wave_seg);
}
/*****/
void cmd_exe(char *commands[], int length)
{
int loop;

/* Execute each command group using a loop */
for (loop = 0; loop < length; loop++)
IOOUTPUTS(ADDR, commands[loop], strlen(commands[loop]));
}

/*****/
void send_data(char *commands, float *Wave_seg, int num_size)
{
char static state[2] = {13, 10};

/* First disable EOI and EOL to send continuous data to the AFG; send last
command */
IOEOI (ISC, 0);IOEOL (ISC, " ", 0);
IOOUTPUTS (ADDR, commands, strlen(commands));

/* Re-enable EOL and EOI for normal HP-IB operation; then send the data */
IOEOI (ISC, 1);IOEOL (ISC, state, 2);
IOOUTPUTA (ADDR, Wave_seg, num_size);
}
/*****/
void run_query(void)
{
char name[6];
int length = 5;

/* Query segment memory */
IOOUTPUTS(ADDR, "SOUR:LIST:SEGM:SEL?", 19);
IOENTERS(ADDR, name, &length);

printf("\nWaveform Segment currently selected: %s", name);
}

```

Continued on next page

```

/*****/
void rst_clr(void)
{
float into;

/* Executes the commands that resets the AFG and clears its error register */
IOOUTPUTS(ADDR, "*RST;*OPC?", 10);
IOENTER(ADDR, &into);
IOOUTPUTS(ADDR, "*CLS;*OPC?", 10);
IOENTER(ADDR, &into);
}
/*****/

void check_error(char *func_tion)
{
char into[161];
int length = 160;

IOOUTPUTS(ADDR, "SYST:ERR?", 9); /* Query error register */
IOENTERS(ADDR, into, &length); /* Enter error message */

if (atoi(into) != 0) /* Determine if error is present */
/* If errors present, print and exit */
{
while (atoi(into) != 0)
{
printf("Error %s in function %s\n\n", into, func_tion);
IOOUTPUTS(ADDR, "SYST:ERR?", 9);
IOENTERS(ADDR, into, &length);
}

exit(1);
}
}
}

```

QuickBASIC Language Programs

All of the QuickBASIC Language example programs in this manual are written for the HP 82335 HP-IB Interface Card using the HP-IB Command Library for BASIC.

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 QuickBASIC programs were developed on the following system:

Controller: HP Vectra 386/25 personal computer
(386 processor operated at 25 MHz)

HP-IB Interface Card: HP 82335 HP-IB Interface with
Command Library

Mainframe: HP 75000 Series B/C

Slot0/Resource Manager: HP E1301A Mainframe/E1405/E1406
Command Module

HP E1340A Logical Address:80

Instrument Language: SCPI

Running a Program

To run a program, first compile and link the program to make an executable file. You can either compile and link the program in the QuickBASIC environment, or compile and link separately on the command line as follows.

Using the QuickBASIC Environment

To compile and link a program, select the QuickBASIC environment as follows:

```
qb <path \program name> /l \path \qbhplib
```

This loads both the program and HP-IB library into the QuickBASIC environment. Note that this only selects the QuickBASIC version 4.0 and above environment. For other versions of QuickBASIC, refer to the “HP 82335 Using the HP-IB Interface and Command Library” manual.

Using the Command Line

To compile and link using the command line is a two step method as follows:

First, compile the program by executing:

```
bc <path \program name>
```

This generates an object listing that has the same name as the program name with an OBJ extension. This object file must be linked to the HP-IB library.

Second, link the HP-IB library:

```
link <path \program name>
```

After prompted for a run file, press the Enter key. The executable file with an EXE extension is then generated.

QuickBASIC Program Example

The following is an example program written in QuickBASIC using the HP 82335 HP-IB Interface Card. The program:

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

```
' ARB_GEN.BAS - This program generates a 4096 points ramp. The data to
' generate the ramp is transferred to the AFG as voltages

DECLARE SUB GenSeg ()
DECLARE SUB CmdExe (Commands$( ), Length%)
DECLARE SUB SendData (Commands$, WaveSeg!( ), NumSize%)
DECLARE SUB RstClr ()
DECLARE SUB CheckError (FuncType$)
DECLARE SUB RunQuery ()

COMMON SHARED ADDR&, ISC&

REM $INCLUDE: 'QBSETUP'

ISC& = 7 'Assigns the HP-IB select code
ADDR& = 70910 'Assign an I/O path between the computer and the AFG

CALL RstClr ' Reset the AFG
CALL GenSeg ' Generate segment list and output sequence
CALL RunQuery ' Query segment name

END

SUB CheckError (FuncType$)

Max.Length% = 160: Actual.Length% = 0
```

Continued on next page


```

Info$ = SPACE$(Max.Length%)

CALL IOOUTPUTS(ADDR&, "SYST:ERR?", 9) ' Query error register
CALL IOENTERS(ADDR&, Info$, Max.Length%, Actual.Length%)
Info$ = LEFT$(Info$, Actual.Length%)

IF VAL(Info$) <> 0 THEN
CLS
WHILE VAL(Info$) <> 0
    PRINT :PRINT Info$; " in function: "; FuncType$
    Actual.Length% = 0

    CALL IOOUTPUTS(ADDR&, "SYST:ERR?", 9)
    CALL IOENTERS(ADDR&, Info$, Max.Length%, Actual.Length%)
    Info$ = LEFT$(Info$, Actual.Length%)
WEND
END
END IF

END SUB

SUB CmdExe (Commands$, Length%)

FOR I = 1 TO Length%
CALL IOOUTPUTS(ADDR&, Commands$(I), LEN(Commands$(I)))
NEXT I

END SUB

SUB GenSeg

DIM SetCommands$(1 TO 2)
DIM OutCommands$(1 TO 2)
DIM WaveSeg!(1 TO 4096)

SetCommands$(1) = "SOUR:ROSC:SOUR INT;" ' Select the Ref. Oscillator
SetCommands$(1) = SetCommands$(1) + ":SOUR:FREQ:FIX 1e3;" ' Set
                                                    waveform frequency
SetCommands$(1) = SetCommands$(1) + ":SOUR:FUNC:SHAP USER;"
                                                    ' Select user function

```

Continued on next page

```

SetCommands$(1) = SetCommands$(1) + ":SOUR:VOLT:LEV:IMM:AMPL 5.1V"
' Set the amplitude

SetCommands$(2) = "SOUR:LIST:SEGM:SEL a" ' Select the segment name

SegCommands$ = "SOUR:LIST:SEGM:VOLT " ' Command to send volts data

OutCommands$(1) = "SOUR:FUNC:USER a" ' Select the user name

OutCommands$(2) = "INIT:IMM" ' Start waveform generation

' Setup the AFG
CALL CmdExe(SetCommands$(1), 2)

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

' Calculate the segments; store in memory
FOR I = 1 TO 4096
WaveSeg!(I) = I * .00122
NEXT I

' Call function to execute the final command with segment data
CALL SendData(SegCommands$, WaveSeg!(), 4096)

' Call function to check for AFG errors
CALL CheckError("GenSeg (SegCommands)")

' Setup the AFG for output
CALL CmdExe(OutCommands$(1), 2)

' Call function to check for AFG errors
CALL CheckError("GenSeg (OutCommands)")

END SUB

SUB RstClr

CALL IOOUTPUTS(ADDR&, "**RST;*OPC?", 10)
CALL IOENTER(ADDR&, Into!)

```

Continued on next page

```

CALL IOOUTPUTS(ADDR&, "*CLS;*OPC?", 10)
CALL IOENTER(ADDR&, Into!)

END SUB

SUB RunQuery

Max.Length% = 20: Actual.Length = 0
Info$ = SPACE$(Max.Length%)

' Query segment name
CALL IOOUTPUTS(ADDR&, "SOUR:LIST:SEGM:SEL?", 19)
CALL IOENTERS(ADDR&, Info$, Max.Length%, Actual.Length%)
Info$ = LEFT$(Info$, Actual.Length%)

PRINT : PRINT "Segment Name: "; Info$

END SUB

SUB SendData (Commands$, WaveSeg!(), NumSize%)

Endline$ = CHR$(13) + CHR$(10)

' First disable EOI and EOL to send continuous data to the AFG; send last command
CALL IOEOI(ISC&, 0)
CALL IOEOL(ISC&, " ", 0)
CALL IOOUTPUTS(ADDR&, Commands$, LEN(Commands$))

' Re-enable EOL and EOI for normal HP-IB operation; then send the data
CALL IOEOI(ISC&, 1)
CALL IOEOL(ISC&, Endline$, LEN(Endline$))
CALL IOOUTPUTA(ADDR&, SEG WaveSeg!(1), NumSize%)

END SUB

```

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

AFG Self-Test The AFG self-test is executed with the command:

```
*TST?
```

Table 1-1. HP E1340A Self-Test Codes.

Self-Test Code	Description
0	Test passed
1	Test failed. An error message describes the failure.

The self-test checks communication between the E1340A and the E1300/01 mainframe or Command Module. Upon completion of the test, one of the self-test codes listed in Table 1-1 is returned.

HP IBASIC (SLFTST)

```
1    !RE-SAVE "SLFTST"  
10   !Send the self-test command, enter and display the result.  
20   DIM Message$(256)  
30   OUTPUT 80910;"*TST?"  
40   ENTER 80910;Rslt  
50   IF Rslt <>0 THEN  
60     REPEAT  
70       OUTPUT 80910;"SYST:ERR?"  
80       ENTER 80910;Code,Message$  
90       PRINT Code,Message$  
100  UNTIL Code=0  
110  END IF  
120  PRINT Rslt  
130  END
```

C and QuickBASIC Programs

The C and QuickBASIC versions of SLFTST are on the C language example programs disk (HP P/N E1340-10036).

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 8.

HP IBASIC (RSTCLS)

```
1   !RE-SAVE"RSTCLS"
10  !Assign an I/O path between IBASIC and the AFG.
20  ASSIGN @Afg TO 80910
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
```

C and QuickBASIC Programs

The C and QuickBASIC versions of RSTCLS are on the C language example programs disk (HP P/N E1340-10036).

Querying the Power-On/Reset Configuration

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

The command which queries each AFG parameter setting is:

```
*LRN?
```

HP IBASIC (LRN)

```
1  !RE-SAVE "LRN"
10 !Assign an I/O path between IBASIC and the AFG.
20 ASSIGN @Afg TO 80910
30 !Call the subprogram
40 CALL Lrn_conf(@Afg)
50 END
60 !
70 SUB Lrn_conf(@Afg)
80 Lrn_conf: !subprogram which queries the AFG reset configuration
90 DIM Lrn$(1000),Temp$(40)
100 OUTPUT @Afg;"*LRN?"
110 ENTER @AFG;Lrn$
120 Temp$=""
130 FOR I=1 TO LEN(Lrn$)
140 IF Lrn$(I,I)=";" THEN
150 PRINT Temp$ !print Temp$ when ; is received
160 Temp$=""
170 ELSE
180 Temp$=Temp$&Lrn$(I,I) !build Temp$ from *LRN data
190 END IF
200 NEXT I
210 SUBEND
```

C and QuickBASIC Programs

The C and QuickBASIC versions of LRN are on the C language example programs disk (HP P/N E1340-10036).

Checking for Errors

The following HP IBASIC program shows the lines and subprogram added to the HP IBASIC 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 to force an "end of statement" condition should a syntax error occur in a group of commands. Otherwise, the ABORT command in line 320 could be considered as part of the group and be ignored by the AFG SCPI interpreter.

Note

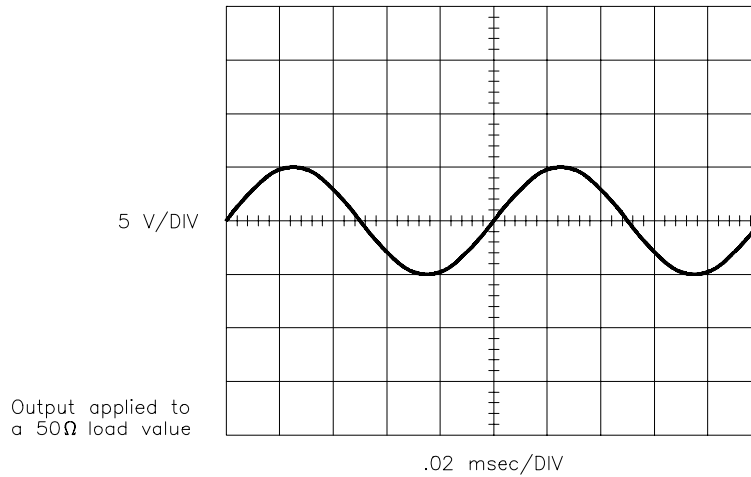
An alternative HP IBASIC error checking program is in the *C-Size VXibus Systems Installation and Getting Started Guide*. Error checking routines for C language and QuickBASIC programs are found in programs ARB_GEN.C and ARB_GEN.BAS, listed previously in this chapter.

HP IBASIC (ERRORCHK)

```
1  !RE-SAVE"ERRORCHK"
2  !This program represents the method used to check for programming
3  !errors in HP IBASIC programs.
10 !Assign I/O path between IBASIC and the E1340A.
20 ASSIGN @Afg TO 80910
30 COM @Afg
40 !Define branch to be taken when an E1340A error occurs.
50 !Enable IBASIC interface to generate interrupt when error occurs.
60 ON INTR 8 CALL Errmsg
70 ENABLE INTR 8;2
80 !Clear all bits in the standard event status register, unmask the
90 !standard event status group summary bit in the E1340A status byte
100 !register (decimal weight 32), unmask the query error, device
110 !dependent error, execution error, and command error bits
120 !(decimal sum 60) in the E1340A standard event status register.
130 OUTPUT @Afg;"*CLS"
140 OUTPUT @Afg;"*SRE 32"
150 OUTPUT @Afg;"*ESE 60"
160 !
170 !Subprogram calls would be here
180 !
190 WAIT .1 !allow error branch to occur before turning intr off
200 OFF INTR 8
210 END
220 !
230 SUB Errmsg
240 Errmsg: !Subprogram which displays E1340 programming errors
250     COM @Afg
260     DIM Message$[256]
270     !Read AFG status byte register and clear service request bit
280     B=SPOLL(@Afg)
290     !End of statement if error occurs among coupled commands
300     OUTPUT @Afg;" "
310     OUTPUT @Afg;"ABORT" !abort output waveform
320     REPEAT
330         OUTPUT @Afg;"SYST:ERR?" !read AFG error queue
340         ENTER @Afg;Code,Message$
350         PRINT Code,Message$
360     UNTIL Code=0
370     STOP
380 SUBEND
```

Generating a Sine Wave

To output a sine wave, only an amplitude needs to be specified and the AFG placed in the wait-for-trigger state (INIT). Using the minimum amount of AFG commands, this program generates a 10 kHz, 10 Vp-p sine wave.



HP IBASIC (RSTSINE)

```
1  !RE-SAVE"RSTSINE"
2  !This program outputs a sine wave based on a specified amplitude
3  !and the reset settings of the AFG.
4  !
10 !Assign an I/O path between IBASIC and the AFG.
20 ASSIGN @Afg TO 80910
30 COM @Afg
40 !
50 !Reset the AFG
60 CALL Rst
70 OUTPUT @AFG;"SOUR:VOLT:LEV:IMM:AMPL 5" !set amplitude
80 OUTPUT @Afg;"INIT:IMM" !output sine wave using reset conditions
90 END
100 !
110 SUB Rst
120 Rst: !subprogram which resets the AFG.
130 COM @Afg
140 OUTPUT @Afg;"*RST;*OPC?" !reset the AFG
150 ENTER @Afg;Complete
160 SUBEND
```

C and QuickBASIC Programs

The C and QuickBASIC versions of RSTSINE are on the C language example programs disk (HP P/N E1340-10036).

HP E1340A Example Programs

For easy reference, names and descriptions of the example programs in this manual are summarized in Appendix B, Table B-1.

Chapter 2 Generating Standard Waveforms with the HP E1340A

Chapter Contents

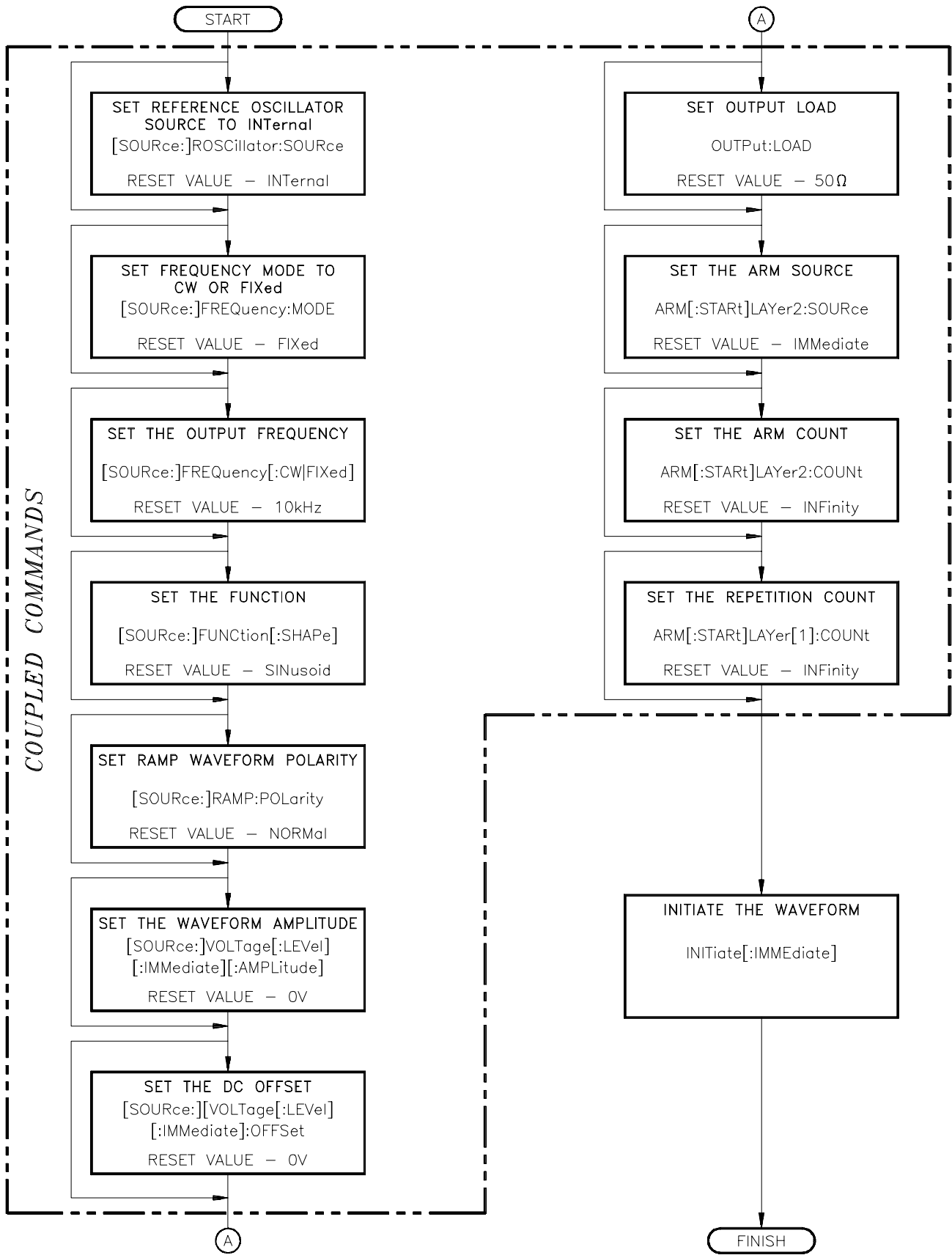
This chapter shows how to generate standard waveforms (SINusoid, SQUare, TRIangle, and RAMPS) using the HP E1340A 12-Bit Arbitrary Function Generator (hereafter called the “AFG”).

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 (i.e., V, Vpeak, etc.), and how to set the waveform amplitude and offset. The sections are as follows:

- Standard Waveforms Flowchart Page 41
- Generating DC Voltages Page 43
- Generating Sine, Triangle, and Square Waves Page 45
- Generating Ramp Waves Page 47
- Selecting the Amplitude Levels and Output Units Page 50
- Program Comments Page 52

Standard Waveforms Flowchart

The flowchart on page 42 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.



Generating DC Voltages

This program outputs a +5 V DC voltage. The commands are:

1. Reset the AFG - *RST

The *RST command aborts any waveform output and selects the sinusoid function.

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 50 for more information.

HP IBASIC Program Example (DCVOLTS)

```
1  !RE-SAVE“DCVOLTS”
2  !This program outputs 5V dc.
3  !
10 !Assign I/O path between IBASIC and the E1340A.
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 subprogram which resets the AFG.
130 CALL Rst
140 !
150 !Set AFG parameters
160 OUTPUT @Afg;“SOUR:FUNC:SHAP DC” !function
170 OUTPUT @Afg;“SOUR:VOLT:LEV:IMM:AMPL 5V” !amplitude
180 !
190 WAIT .1                               !allow interrupt to be serviced
```

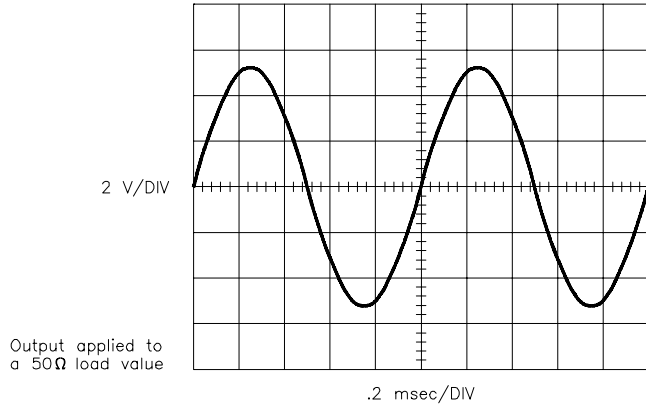
Continued on next page

```
200 OFF INTR 8
210 END
220 !
230 SUB Rst
    •
    •
300 SUB Errmsg
    •
    •
```

C and QuickBASIC Program Versions

The C example program, DCVOLTS.C, is in directory “CPROG” and the QuickBASIC example program, DCVOLTS.BAS, is in directory “QBPROG” on the C and QuickBASIC example disk (part number E1340-10036).

Generating Sine, Triangle, and Square Waves



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

1. Reset the AFG - *RST

The *RST command aborts any waveform output and selects the sinusoid function.

2. Set the Waveform Frequency -
[SOURce:]FREQuency[:FIXed] <frequency>

This command specifies the waveform frequency. Refer to 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.) For the TRIangle and SQUare Wave functions, use the TRIangle and SQUare parameters, respectively, instead of SINusoid.

4. Set the Amplitude -
[SOURce:]VOLTag[e][:LEVel][:IMMediate][:AMPLitude] <amplitude>

This command specifies the amplitude. Refer to the section called “Selecting the Amplitude Levels and Output Units” on page 50 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.

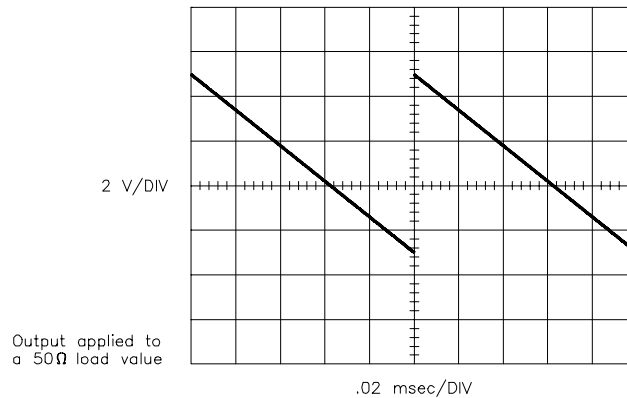
HP IBASIC Program Example (SINEWAVE)

```
1  !RE-SAVE"SINEWAVE"
2  !This program generates a 5V, 1 kHz sine wave.
3  !
10 !Assign I/O path between IBASIC and the E1340A.
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 subprogram which resets the AFG.
130 CALL Rst
140 !
150 !Set AFG parameters
160 OUTPUT @Afg;"SOUR:FREQ:FIX 1E3;"!frequency
170 OUTPUT @Afg;"SOUR:FUNC:SHAP SIN;"!function
180 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5V"!amplitude
190 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
200 !
210 WAIT .1 !allow interrupt to be serviced
220 OFF INTR 8
230 END
240 !
250 SUB Rst
    •
    •
320 SUB Errmsg
    •
    •
```

C and QuickBASIC Program Versions

The C example program, SINEWAVE.C, is in directory "CPROG" and the QuickBASIC example program, SINEWAVE.BAS, is in directory "QBPROG" on the C and QuickBASIC example disk (part number E1340-10036).

Generating Ramp Waves



This program outputs a negative going Ramp at 10 kHz, 4 V output level, and +1 V offset. The commands are:

1. Reset the AFG - *RST

The *RST command aborts any waveform output, selects the sinusoid function, selects the 42.9 MHz reference oscillator source, 10 kHz frequency, arm start immediate, 0 V offset, and 0 V output.

2. Set the Frequency -
[SOURce:]FREQUency[:FIXed] <frequency>

This command specifies the frequency. Refer to Appendix B for the frequency limits.

3. Select the Function -
[SOURce:]FUNCTION[:SHAPe] RAMP

This command selects the RAMP function.

4. Select the Ramp Polarity -
[SOURce:]RAMP:POLarity INVerted

This command selects the polarity of the RAMP wave. Use NORMAl for the initial voltage to go positive; use INVerted for the initial voltage to go negative.

5. 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 50 for more information.

6. 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 50 for more information.

7. 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 IBASIC Program Example (RAMPWAVE)

```
1  !RE-SAVE"RAMPWAVE"
2  !This program generates a 4V, 10 kHz negative-going ramp wave.
3  !
10 !Assign I/O path between IBASIC and the E1340A.
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 subprogram which resets the AFG.
130 CALL Rst
140 !
150 !Set AFG parameters
160 OUTPUT @Afg;"SOUR:ROSC:SOUR INT;" ;!reference oscillator
170 OUTPUT @Afg;"SOUR:FREQ:FIX 10E3;" ;!frequency
180 OUTPUT @Afg;"SOUR:FUNC:SHAP RAMP;" ;!function
190 OUTPUT @Afg;"SOUR:RAMP:POL INV;" ;!ramp polarity
200 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 4V;" ;!amplitude
210 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:OFFS 1V" ;!offset
220 OUTPUT @Afg;"INIT:IMM" ;!wait-for-arm state
230 !
240 WAIT .1 ;!allow interrupt to be serviced
250 OFF INTR 8
260 END
270 !
280 SUB Rst
```

Continued on next page


```
•  
•  
350 SUB Errmsg  
•  
•
```

C and QuickBASIC Program Versions

The C example program, RAMPWAVE.C, is in directory “CPROG” and the QuickBASIC example program, RAMPWAVE.BAS, is in directory “QBPROG” on the C and QuickBASIC example disk (part number E1340-10036).

Selecting the Amplitude Levels and Output Units

This 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

The *RST command aborts any waveform output, selects the sinusoid function, selects the 42.9 MHz reference oscillator source, arm start immediate, 0 V offset, and 0 V output.

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 required 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 Appendix B for the amplitude limits. The maximum value of the combined amplitude and offset voltages must remain within the 7 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 IBASIC Program Example (OUTPUNIT)

```
1  !RE-SAVE"OUTPUNIT"
2  !This program sets the AFG's output units to volts peak-to-peak.
3  !
10 !Assign I/O path between IBASIC and the E1340A.
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 subprogram which resets the AFG.
130 CALL Rst
140 !
150 !Set AFG parameters
160 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL:UNIT:VOLT VPP"
      !output units
      •
      •
200 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 8;"!amplitude
210 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:OFFS 1"!offset
220 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
230 !
240 WAIT .1 !allow interrupt to be serviced
250 OFF INTR 8
260 END
270 !
280 SUB Rst
      •
      •
350 SUB Errmsg
      •
      •
```

C and QuickBASIC Program Versions

The C example program, OUTPUNIT.C, is in directory "CPROG" and the QuickBASIC example program, OUTPUNIT.BAS, is in directory "QBPROG" on the C and QuickBASIC example disk (part number E1340-10036).

Program Comments

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

Reference Oscillator Sources

- The SINusoid, SQUare, TRlangle, and RAMP functions can use any of the Reference Oscillator Sources. The sources, selected by [SOURce:]ROSCillator:SOURce, are:

INTernal - 42.94967296 MHz (power-on value)

EXTernal - User provided value (the front panel “Aux In” BNC)

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

Output Load Comments

- For correct output amplitude values, the load applied to the AFG “Signal Output” terminals must be the same value as the selected AFG output impedance value.

- To output to an open circuit, execute OUTPUT:LOAD INFinity or 9.9E+37. The HP E1340A 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:

```
[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] 5V
```

changes the unit type to volts (i.e., 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, TRlangle, and RAMPS.
- The default unit type only applies for amplitudes and not for offsets. The unit for offsets must always be specified in “V” for volts. For example, executing:

```
[SOURce:]VOLTage:OFFSet .1VPP
```

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

```
[SOURce:]VOLTage:OFFSet 0.1
```

or

```
[SOURce:]VOLTage:OFFSet 0.1V
```

- The W, DBM, and DBMW unit types references the amplitude levels to the 50 Ω output load value. Thus, the W, DBM, and DBMW values are meaningless and not available when selecting an open circuit load.

Using MINimum and MAXimum Parameters

You can execute many commands (like [SOURCE:]FREQUENCY[:CW]:FIXed]) using the MINimum or MAXimum parameters instead of a number value. However, MIN and MAX are internally converted immediately to numeric values when received using the values of other commands in effect at that time. The values of other commands affecting the MIN and MAX computation are subsequently changed, but the already computed MIN and MAX values do **not** change.

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 with the HP E1340A

Chapter Contents

This chapter shows how to generate arbitrary waveforms using the HP E1340A 12-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 55
- Generating a Simple Arbitrary Waveform Page 59
- Executing Several Waveform Segments Page 65
- Arbitrary Waveform Hopping Page 69
- Generating Built-In Arbitrary Waveforms Page 73
- Sample Programs Page 76
 - Generating a Damped Sine Wave Page 76
 - Generating an Exponential Charge/Discharge
Waveform Page 78
 - Generating a Sine Wave with Spikes Page 79
 - Generating a Half-Rectified Sine Wave Page 80
- Program Comments Page 82

Arbitrary Waveforms Flowchart

The flowchart on page 58 shows the commands and the command execution order to generate standard 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.

How the AFG Generates Arbitrary Waveforms

Refer to Figure 3-1. An arbitrary waveform consists of two parts, a waveform segment (or all points on a waveform) and a segment sequence. The segments contain the actual voltage points of the waveform. The segment sequence determines which waveform segments are to be output.

The AFG has four waveform segments available: A, B, C, and D. The size of each waveform segment is 4096 points. Thus, to store data into a waveform segment, the number of data items (or waveform points) must be 4096.

To output a waveform, set the user function to output one or more of the waveform segments. To output a waveform with a single waveform segment, use one of the A through D waveform segments. To output waveforms with multiple waveform segments, use a combination of AB or ABCD waveform segments. The AB combination outputs the 8192 point waveform defined by the A and B waveform segments. Thus, the A segment is output first and then the B segment. The ABCD combination outputs the 16384 point waveform defined by the A, B, C, and D waveform segments. Thus, the A segment is output first and the D segment is output last.

Similarly, the AB combination outputs a waveform with 8192 points and the ABCD combination outputs a waveform with 16384 points.

The segment values can be either sent as voltage values (use [SOURCE]:LIST[:SEGMENT]:VOLT <voltage_list>) or DAC (digital-to-analog converter) codes (use [SOURCE]:LIST[:SEGMENT]:VOLT:DAC <voltage_list>). If sent as voltage values, the AFG converts them to DAC codes before storing them in memory.

To output a waveform, the AFG sets the DAC to the voltage value of the selected waveform segment. The waveform frequency determines the waveform repetition rate.

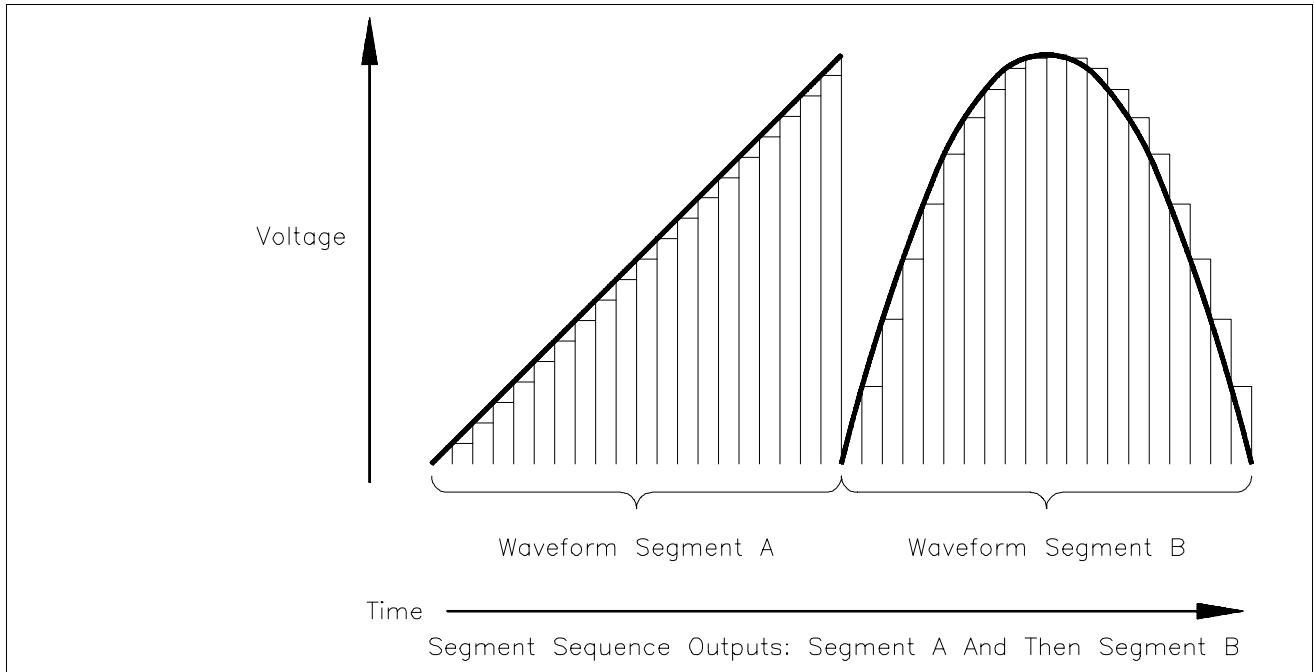
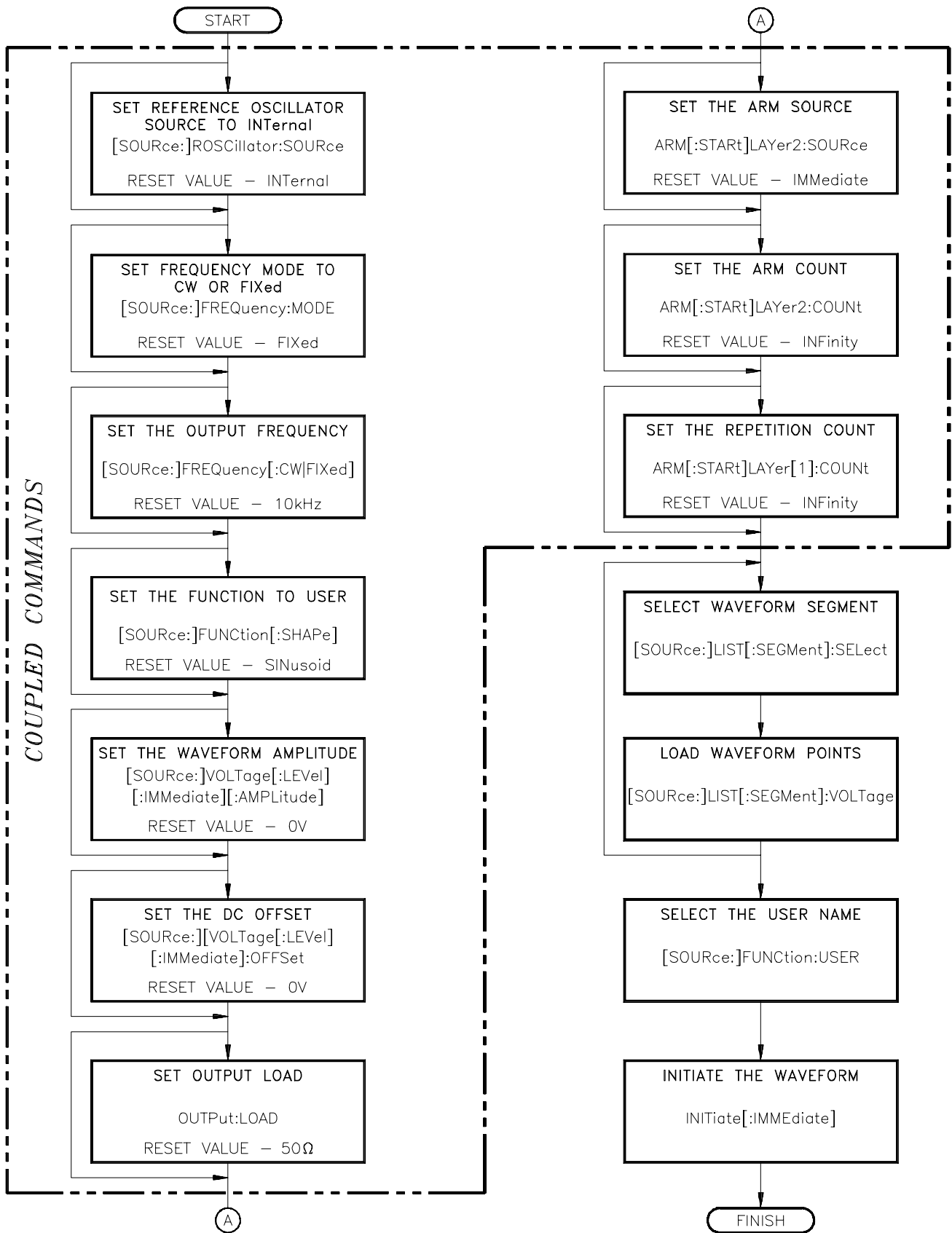
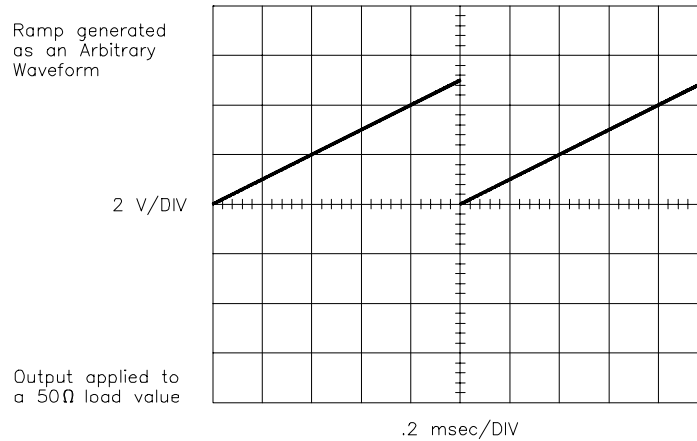


Figure 3-1. Generating Arbitrary Waveforms



Generating a Simple Arbitrary Waveform



This program shows how to generate an arbitrary waveform with a single waveform segment. The example generates a 4096 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. Select the Reference Oscillator -
[SOURCE:]ROSCillator:SOURCE <source>

This command selects the Reference Oscillator Source (see “Program Comments” later in this chapter).

3. Set the Waveform Frequency -
[SOURCE:]FREQUENCY[:CW|FIXed] <frequency>

This command sets the repetition rate of the waveform. See Appendix B for the frequency limits.

4. Select the Arbitrary Waveform Function -
[SOURCE:]FUNCTION[:SHAPE] USER

This command selects the arbitrary waveform function.

5. 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 Appendix B for the amplitude limits.

6. Select the Waveform Segment -
[SOURce:]LIST[:SEGMENT]:SElect <name>

Use either A, B, C, or D in <name> for the waveform segment.

7. Store the Waveform Segment as Voltages -
[SOURce:]LIST[: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.

8. Select the User Name -
[SOURce:]FUNCTION:USER <name>

This command selects the waveform segment to be output. Make the *name* in this command the same name as the waveform segment to be output.

9. 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.

10. Query the Waveform Segment (Optional) -
[SOURce:]LIST[:SEGMENT]:SElect?

This command returns the currently selected waveform segment.

HP IBASIC Program Example (ARB_GEN)

```
1  !RE-SAVE"ARB_GEN"
2  !This program generates a 4096 point, 0 to 5V ramp waveform.
3  !The data is transferred to the AFG as voltages.
4  !
10 !Assign I/O path between IBASIC and the E1340A.
20 ASSIGN @Afg TO 80910
30 COM @Afg,Waveform(1:4096)
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 subprogram which resets the AFG.
130 CALL Rst
140 !Set AFG parameters
150 OUTPUT @Afg;"SOUR:ROSC:SOUR INT;";!reference oscillator
160 OUTPUT @Afg;":SOUR:FREQ:FIX 1E3;";!frequency
170 OUTPUT @Afg;":SOUR:FUNC:SHAP USER;";!function
180 OUTPUT @Afg;":SOUR:VOLT:LEV:IMM:AMPL 5.1V";!scale
                                         amplitude
190 !
200 !Call subprogram which defines waveform segment.
210 CALL Ramp_wave
220 !
230 !Select output sequence and initiate waveform
240 OUTPUT @Afg;"SOUR:FUNC:USER A"
250 OUTPUT @Afg;"INIT:IMM"           !wait-for-arm state
260 !
270 WAIT .1                          !allow interrupt to be serviced
280 OFF INTR 8
290 END
300 !
310 SUB Ramp_wave
320 Ramp_wave: !Subprogram which defines a ramp waveform
330     COM @Afg,Waveform(*)
340     FOR I=1 TO 4096
350         Waveform(I)=I*.00122
360     NEXT I
370 !
380     OUTPUT @Afg;"SOUR:LIST:SEGM:SEL A";!Select segment name
```

Continued on next page

```

390     OUTPUT @Afg;" SOUR:LIST:SEGM:VOLT";Waveform(*)!load
                                           waveform points
400 SUBEND
410 !
420 SUB Rst
    •
    •
490 SUB Errmsg
    •
    •

```

C and QuickBASIC Program Versions

The C example program, ARB_GEN.C, is in directory “CPROG” and the QuickBASIC example program, ARB_GEN.BAS, is in directory “QBPROG” on the C and QuickBASIC example disk (part number E1340-10036).

Transferring Data in C

To transfer data to the AFG requires that the data sent with the [SOURCE:]LIST[:SEGMENT]:VOLTage command must be contiguous. To do this, send no carriage return (CR) and line feed (LF) before transferring all the data. The send_data function does this as follows (refer to the C function below):

1. Disable EOI and EOL using IOEOI (ISC, 0) and IOEOL (ISC, “ ”, 0), respectively. ISC contains the HP-IB select code, and the 0 and “ ” (NULL) values disables the carriage return (CR) and line feed (LF) to allow the AFG to receive a command string without a terminator.
2. Next, sent to the AFG the SCPI [SOURCE:]LIST[:SEGMENT]:VOLTage command string using IOOUTPUTS (ADDR, commands, strlen(commands)). ADDR contains the AFG HP-IB address, commands contains the SCPI command string, and strlen(commands) contains the string length of commands. This enables the AFG to receive voltage data. Since CR or LF is disabled, the AFG holds execution until all the data is sent.
3. Enable EOI and EOL using IOEOI (ISC, 1) and IOEOL (ISC, state, 2), respectively. The variable state contains the decimal codes for CR and LF. Although EOL and EOI are enabled, no CR and LF is sent until all voltage data transfer is completed (next step).

4. The voltage data values are sent next using the array Wave_seg in IOOUTPUTA (ADDR, Wave_seg, num_size). The values in Wave_seg are in a floating point format. IOOUTPUTS automatically separates each value in the arrays by a comma (,).
5. After all the data is sent, the data transfer terminates with a CR and LF. This lets the AFG know that it can receive a new command.

```

void send_data(char *commands, float *Wave_seg, int num_size)
{
    char static state[2] = {13, 10};

    /* First disable EOI and EOL; then send the last command */
    IOEOI (ISC, 0);IOEOL (ISC, " ", 0);
    IOOUTPUTS (ADDR, commands, strlen(commands));

    /* Re-enable EOL and EOI for normal HP-IB operation; then
       send the last data */
    IOEOI (ISC, 1);IOEOL (ISC, state, 2);
    IOOUTPUTA (ADDR, Wave_seg, num_size);
}

```

Transferring Data in QuickBASIC

To transfer data to the AFG requires that the data sent with the [SOURCE:]LIST[:SEGMENT]:VOLTage command must be contiguous. To do this, send no carriage return (CR) and line feed (LF) before transferring all the data. The SendData routine in the QuickBASIC program do this as follows (refer to the QuickBASIC routine below):

1. Disable EOI and EOL using IOEOI (ISC&, 0) and IOEOL (ISC&, " ", 0), respectively. ISC& contains the HP-IB select code, the 0 and " " (NULL) values disables the carriage return (CR) and line feed (LF) to allow the AFG to receive a command string without a terminator.
2. Next, sent to the AFG the SCPI [SOURCE:]LIST[:SEGMENT]:VOLTage command string using IOOUTPUTS(ADDR&, Commands\$, LEN(Commands\$)). ADDR& contains the AFG HP-IB address, Commands\$ contains the SCPI command string, and LEN(Commands\$) contains the string length of Commands\$. This enables the AFG to receive voltage data. Since CR or LF is disabled, the AFG holds execution until all the data is sent.
3. Enable EOI and EOL using IOEOI(ISC&, 1) and IOEOL(ISC&, Endline\$, LEN(Endline\$)), respectively. The variable Endline\$ contains the decimal codes for CR and LF. Although EOL and EOI are enabled, no CR and LF is sent until all voltage data transfer is completed (next step).

4. The voltage data values are sent next using the array WaveSeg!() in IOOUTPUTA(ADDR&, SEG WaveSeg!(1), NumSize%). The values in WaveSeg! are in a floating point format. IOOUTPUTS automatically separates each value in the arrays by a comma (,).
5. After all the data is sent, the data transfer terminates with a CR and LF. This lets the AFG know that it can receive a new command.

```
SUB SendData (Commands$, WaveSeg!(), NumSize%)
```

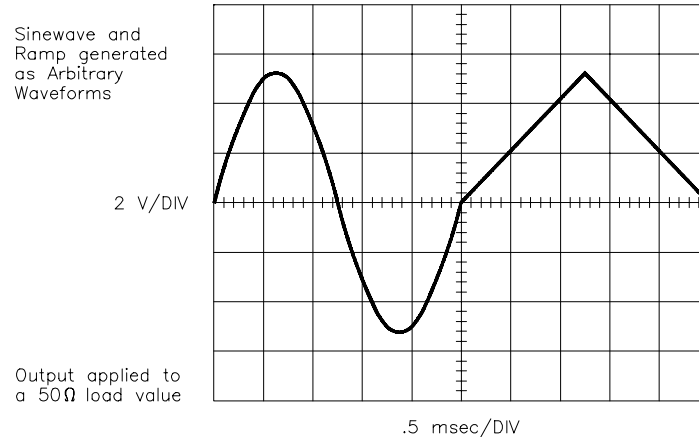
```
Endline$ = CHR$(13) + CHR$(10)
```

```
    ' First disable EOI and EOL; then send the last command  
CALL IOEOI(ISC&, 0)  
CALL IOEOL(ISC&, " ", 0)  
CALL IOOUTPUTS(ADDR&, Commands$, LEN(Commands$))
```

```
    ' Re-enable EOL and EOI for normal HP-IB operation; then send the data  
CALL IOEOI(ISC&, 1)  
CALL IOEOL(ISC&, Endline$, LEN(Endline$))  
CALL IOOUTPUTA(ADDR&, SEG WaveSeg!(1), NumSize%)
```

```
END SUB
```


Executing Several Waveform Segments



This program shows how to generate an arbitrary waveform that combines two different waveform segments. One waveform segment generates a 1 kHz, 5 V sine wave. The other one generates a 1 kHz, 5 V peak triangle. The commands are:

1. Reset the AFG - *RST

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

2. Set the Waveform Frequency -
[SOURce:]FREQuency[:CW|FIXed] <frequency>

This command sets the repetition rate of the waveform. See Appendix B for the frequency limits.

3. Select the Arbitrary Waveform Function -
[SOURce:]FUNctio[n][:SHAPE] USER

This command selects the arbitrary waveform function. Couple the command to the previous frequency command.

4. Set the Maximum Output Amplitude -
[SOURce:]VOLtag[e][: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 Appendix B for the amplitude limits

5. Select the First Waveform Segment -
[SOURce:]LIST[:SEGMENT]:SElect A
Use A to select the first waveform segment.

6. Store the First Waveform Segment as Voltages -
[SOURce:]LIST[:SEGMENT]:VOLTage <voltage_list>

This command stores the points of the “A” 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.

7. Select the Second Waveform Segment -
[SOURce:]LIST[:SEGMENT]:SELEct B

Use B to select the second waveform segment.

8. Store the Second Waveform Segment as Voltages -
[SOURce:]LIST[:SEGMENT]:VOLTage <voltage_list>

This command stores the points of the B 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.

9. Select the User Name -
[SOURce:]FUNCTION:USER AB

This command selects the waveform segments to be output (i.e., AB).

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 IBASIC Program Example (MULSEG)

```
1  !RE-SAVE“MULSEG”
2  !This program generates an arbitrary waveform comprised of two
3  !waveform segments. One segment is a sine wave and the other
4  !segment is a triangle wave.
5  !
10 !Assign I/O path between IBASIC and the E1340A.
20 ASSIGN @Afg TO 80910
30 COM @Afg,Waveform1(1:4096),Waveform2(1:4096)
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 subprogram which resets the AFG.
130 CALL Rst
140 !
150 !Set AFG parameters
160 OUTPUT @Afg;“SOUR:FREQ:FIX 10E3;”;!frequency
170 OUTPUT @Afg;“:SOUR:FUNC:SHAP USER;”;!function
180 OUTPUT @Afg;“:SOUR:VOLT:LEV:IMM:AMPL 5.1V” !scale
amplitude
190 !
200 !Call subprogram which defines waveform segments A and B.
210 CALL Sine_tri_wave
220 !
230 !Select output sequence and initiate waveform
240 OUTPUT @Afg;“SOUR:FUNC:USER AB”
250 OUTPUT @Afg;“INIT:IMM” !wait-for-arm state
260 !
270 WAIT .1 !allow interrupt to be serviced
280 OFF INTR 8
290 END
300 !
310 SUB Sine_tri_wave
320 Sine_tri_wave: !Subprogram which defines waveforms
330     COM @Afg,Waveform1(*),Waveform2(*)
340     FOR I=1 TO 4096
350         Waveform1(I)=5*(SIN(2*PI*(I/4096)))
360     NEXT I
370     !
```

Continued on next page

```

380     OUTPUT @Afg;"SOUR:LIST:SEGM:SEL A"
           !select segment name
390     OUTPUT @Afg;" SOUR:LIST:SEGM:VOLT";Waveform1(*)
           !load waveform points
400     !

420         Waveform2(I)=I*.0024414
430     NEXT I
440     FOR I=2048 TO 4096
450         Waveform2(I)=(4096-I)*.0024414
460     NEXT I
470     !
480     OUTPUT @Afg;"SOUR:LIST:SEGM:SEL B" !select segment name
490     OUTPUT @Afg;" SOUR:LIST:SEGM:VOLT";Waveform2(*)
           !load waveform points

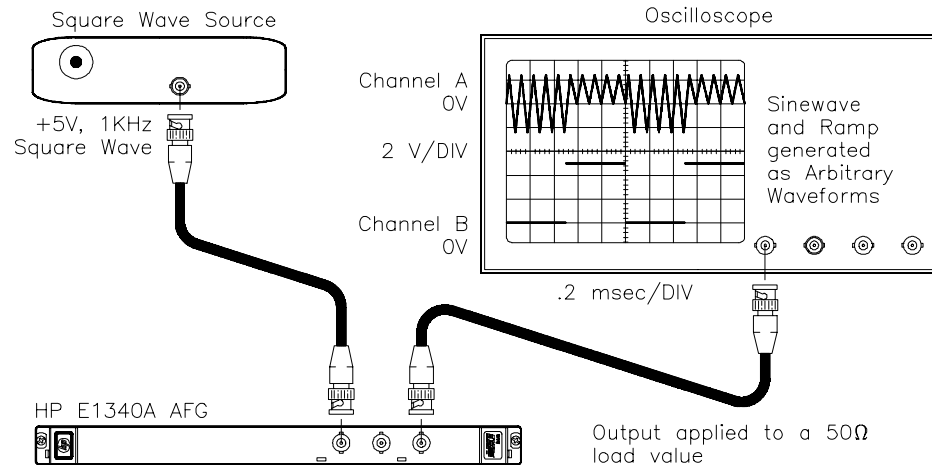
500 SUBEND
510 !
520 SUB Rst
    •
    •
590 SUB Errmsg
    •
    •

```

C and QuickBASIC Program Versions

The C example program, MULSEG.C, is in directory "CPROG" and the QuickBASIC example program, MULSEG.BAS, is in directory "QBPROG" on the C and QuickBASIC example disk (part number E1340-10036).

Arbitrary Waveform Hopping



This program performs waveform hopping of waveform segments A and B. Use a very stable 1 kHz, 0 to 5 V square wave applied to the AFG's "Aux In" connector to switch between the waveform segments. The positive level of the square wave outputs waveform segment A to generate a 5 cycle sine wave. The zero level of the square wave outputs waveform segment B to generate a 5 cycle triangle wave. (You may need to adjust the 1 kHz square wave to output a stable arbitrary waveform.) There will be about a 20 msec delay between a transition on the "Aux In" connector and the output switching.

1. Reset the AFG - *RST

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

2. Setup the AFG for an Arbitrary Waveform Output -
[SOURce:]FREQuency[:CW|FIXed] <frequency>
[SOURce:]FUNCTion[:SHAPe] USER
[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>

These commands setup the AFG to output arbitrary waveforms.

3. Select the First Waveform Segment -
[SOURce:]LIST[:SEGMENT]:SElect A

Use A to select the first waveform segment.

4. Store the First Waveform Segment as Voltages -
[SOURce:]LIST[:SEGMENT]:VOLTage <voltage_list>

This command stores the points of the "A" 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.

5. Select the Second Waveform Segment -
[SOURce:]LIST[:SEGMENT]:SELEct B

Use B to select the second waveform segment.

6. Store the Second Waveform Segment as Voltages -
[SOURce:]LIST[:SEGMENT]:VOLTage <voltage_list>

This command stores the points of the “B” 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.

7. Select the User Name -
[SOURce:]FUNCTION:USER HOP_AB

This command setup the AFG to hop between waveform segments A and B.

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 IBASIC Program Example (ARB_HOP)

```
1  !RE-SAVE“ARB_HOP”
2  !This program hops (selects) between two waveform segments based
3  !on the level of a 1 kHz, 0 to 5V square wave applied to the AFG’s
4  !“Aux In” connector. A low level selects segment A (sine wave), a
5  !high level selects segment B (triangle wave).
6  !
10 !Assign I/O path between IBASIC and the E1340A.
20 ASSIGN @Afg TO 80910
30 COM @Afg,Waveform1(1:4096),Waveform2(1:4096)
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 subprogram which resets the AFG.
130 CALL Rst
140 !Set AFG parameters
150 OUTPUT @Afg;“SOUR:FREQ:FIX 10E3;”;!frequency
160 OUTPUT @Afg;“:SOUR:FUNC:SHAP USER;”; !function
```

Continued on next page

```

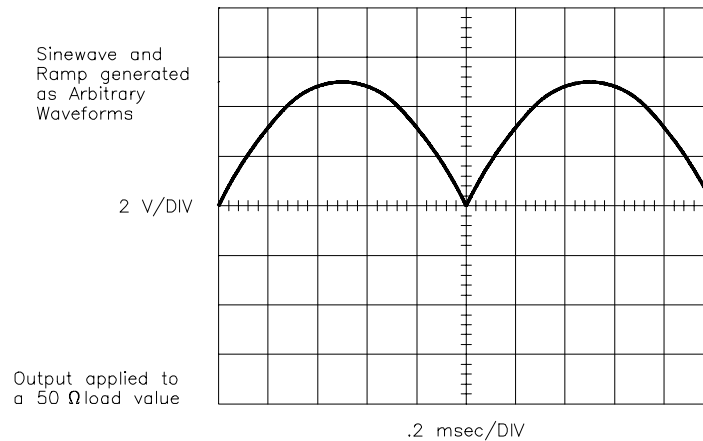
170 OUTPUT @Afg;“:SOUR:VOLT:LEV:IMM:AMPL 5.1V” !scale
      amplitude
180 !
190 !Call subprogram which defines waveform segments A and B.
200 CALL Sine_tri_wave
210 !
220 !Select output sequence and initiate waveform
230 OUTPUT @Afg;“SOUR:FUNC:USER HOP_AB”
240 OUTPUT @Afg;“INIT:IMM” !wait-for-arm state
250 !
260 WAIT .1 !allow interrupt to be serviced
270 OFF INTR 8
280 END
290 !
300 SUB Sine_tri_wave
310 Sine_tri_wave: !Subprogram which defines
      waveforms
320 COM @Afg,Waveform1(*),Waveform2(*)
350 FOR I=1 TO 4096
360 Waveform1(I)=5*(SIN(2*PI*(I/4096)))
370 NEXT I
380 !
390 OUTPUT @Afg;“SOUR:LIST:SEGM:SEL A”!Select segment name
400 OUTPUT @Afg;“ SOUR:LIST:SEGM:VOLT”;Waveform1(*)
      !load waveform points
410 !
420 FOR I=1 TO 2047
430 Waveform2(I)=I*.0024414
440 NEXT I
450 FOR I=2048 TO 4096
460 Waveform2(I)=(4096-I)*.0024414
470 NEXT I
480 !
490 OUTPUT @Afg;“SOUR:LIST:SEGM:SEL B”!Select segment name
500 OUTPUT @Afg;“ SOUR:LIST:SEGM:VOLT”;Waveform2(*)
      !load waveform points
510 SUBEND
520 !
530 SUB Rst
      •
      •
600 SUB Errmsg
      •
      •

```

C and QuickBASIC Program Versions

The C example program, ARB_HOP.C, is in directory “CPROG” and the QuickBASIC example program, ARB_HOP.BAS, is in directory “QBPROG” on the C and QuickBASIC example disk (part number E1340-10036).

Generating Built-In Arbitrary Waveforms



The AFG has 16 built-in arbitrary waveforms available for use. All of these waveforms have 4096 points and are stored in an EPROM (see “Program Comments” for the waveform listing). To execute the waveforms, download the data points into waveform segment A, B, C, or D. The following program downloads and generates the 16th waveform.

1. Reset the AFG - *RST

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

2. Setup the AFG for an Arbitrary Waveform Output -
[SOURCE:]FREQUENCY[:CW|FIXed] <frequency>
[SOURCE:]FUNCTION[:SHAPE] USER
[SOURCE:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>

These commands setup the AFG to output an arbitrary waveform.

3. Download the Waveform Data from the EEPROM into Memory -
[SOURCE:]ARbitrary:DOWNload <source>, <dest>, <length>

This command downloads the segment data from the EPROM where <source> selects the waveform block in the EPROM, <dest> is the waveform segment name (i.e., A, B, C, or D), and <length> is the waveform segment length (i.e., 4096).

4. Select the User Name -
[SOURCE:]FUNCTION:USER <name>

This command selects the waveform segment to be output. Make the *name* in this command the same name as the waveform segment to be output.

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 IBASIC Program Example (ROM_DOWN)

```
1  !RE-SAVE"ROM_DOWN"
2  !This program downloads a 4096 point waveform to waveform segment
3  !memory from block 16 (EEPROM) of the AFG's waveform EEPROM.
4  !
10 !Assign I/O path between IBASIC and the E1340A.
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 subprogram which resets the AFG.
130 CALL Rst
140 !
150 !Set AFG parameters
160 OUTPUT @Afg;"SOUR:FREQ:FIX 1E3;"!frequency
170 OUTPUT @Afg;"SOUR:FUNC:SHAP USER;"!function
180 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5V"!amplitude
190 OUTPUT @Afg;"SOUR:ARB:DOWN EEPROM16,A,4096"!download
                                     waveform
200 OUTPUT @Afg;"SOUR:FUNC:USER A"!select sequence
210 OUTPUT @Afg;"INIT:IMM"           !wait-for-arm state
220 !
230 WAIT .1                          !allow interrupt to be serviced
240 OFF INTR 8
250 END
260 !
270 SUB Rst
    •
    •
340 SUB Errmsg
    •
    •
```

C and QuickBASIC Program Versions

The C example program, ROM_DOWN.C, is in directory “CPROG” and the QuickBASIC example program, ROM_DOWN.BAS, is in directory “QBPROG” on the C and QuickBASIC example disk (part number E1340-10036).

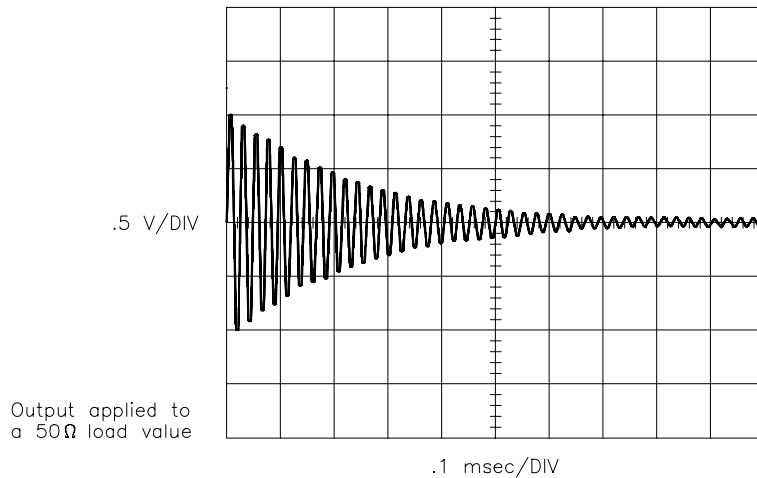
Sample Programs

The following programs generate various arbitrary waveforms. All programs output waveforms at a 1 kHz waveform frequency and 5 V amplitude.

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 Damped Sine Wave

This program generates a Damped Sine Wave using 4096 segments or points.



HP IBASIC Program Example (SIN_D)

```
1 !RE-SAVE"SIN_D"
2 !This program generates a damped sine wave.
3 !
10 !Assign I/O path between IBASIC and the E1340A.
20 ASSIGN @Afg TO 80910
30 COM @Afg,Waveform(1:4096)
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 !
```

Continued on next page

```

120 !Call the subprogram which resets the AFG.
130 CALL Rst
140 !
150 !Set AFG parameters
160 OUTPUT @Afg;"SOUR:FREQ:FIX 1E3,";!frequency
170 OUTPUT @Afg;"SOUR:FUNC:SHAP USER,";!function
180 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 1.1V" !scale
                                         amplitude

190 !
200 !Call subprogram which defines waveform segment.
210 CALL Damped_sine
220 !
230 !Select output sequence and initiate waveform
240 OUTPUT @Afg;"SOUR:FUNC:USER A"
250 OUTPUT @Afg;"INIT:IMM"           !wait-for-arm state
260 !
270 WAIT .1                          !allow interrupt to be serviced
280 OFF INTR 8
290 END
300 !
310 SUB Damped_sine
320 Damped_sine: !Subprogram which defines a damped sine wave
330     COM @Afg,Waveform(*)
340     A=4/4096
350     W=(2*PI)/50
360     FOR T=1 TO 4096
370         Waveform(T)=EXP(-A*T)*SIN(W*T)
380     NEXT T
390     !
400     OUTPUT @Afg;"SOUR:LIST:SEGM:SEL A"!select segment name
410     OUTPUT @Afg;"SOUR:LIST:SEGM:VOLT";Waveform(*)!load
                                         waveform points

420 SUBEND
430 !
440 SUB Rst
    •
    •
510 SUB Errmsg
    •
    •

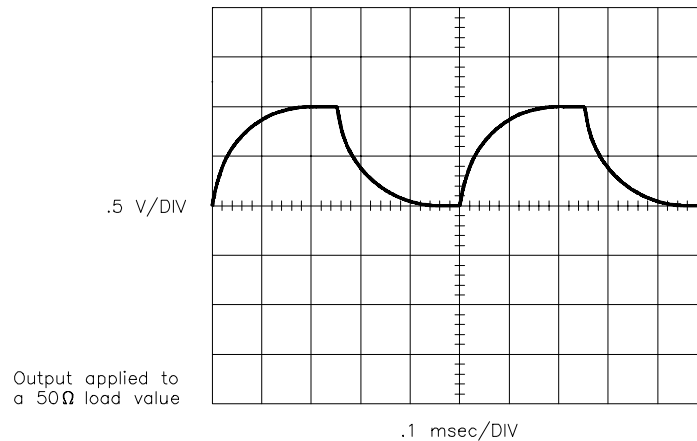
```

C and QuickBASIC Program Versions

The C example program, SIN_D.C, is in directory "CPROG" and the QuickBASIC example program, SIN_D.BAS, is in directory "QBPROG" on the C and QuickBASIC example disk (part number E1340-10036).

Generating an Exponential Charge/Discharge Waveform

This program generates an Exponential Charge/Discharge Waveform with 4096 segments or points.



HP IBASIC Program Example (CHARGE)

This program is similar to the “SIN_D” BASIC program on page 76, with the following differences:

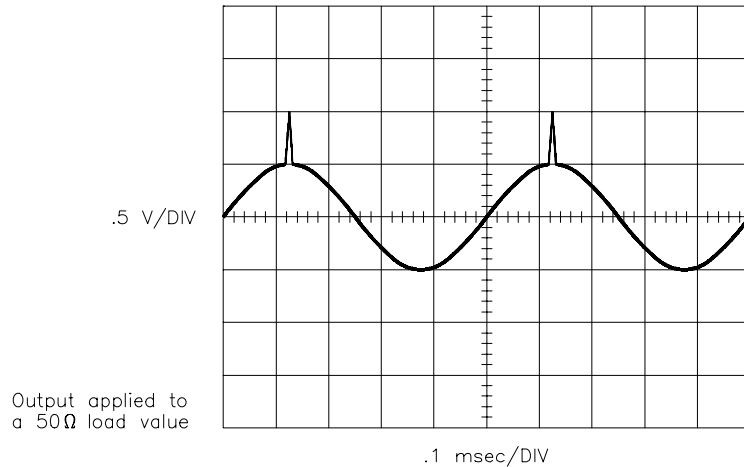
```
1  !RE-SAVE“CHARGE”
2  !This program generates an exponential charge/discharge waveform.
   •
   •
210 CALL Charge
   •
   •
310 SUB Charge
320 Charge:!Subprogram which defines an exponential charge/discharge
330      !waveform.
340      COM @Afg,Waveform(*)
350      Rc=400
360      FOR T=1 TO 4096
370          IF T>=0 AND T<2047 THEN
380              Waveform(T)=1*(1-EXP(-T/Rc))
390          END IF
400          IF T>=2047 THEN
410              Waveform(T)=1*(1-EXP(-2048/Rc))-1*(1-EXP(-(T-2047)/Rc))
420          END IF
430      NEXT T
440      !
450      OUTPUT @Afg;“SOUR:LIST:SEGM:SEL A”!select segment name
460      OUTPUT @Afg;“ SOUR:LIST:SEGM:VOLT”;Waveform(*)!load
                                           waveform points
470 SUBEND
```

C and QuickBASIC Program Versions

The C example program, CHARGE.C, is in directory "CPROG" and the QuickBASIC example program, CHARGE.BAS, is in directory "QBPROG" on the C and QuickBASIC example disk (part number E1340-10036).

Generating a Sine Wave with Spikes

This program generates a Sine Wave with Spikes using 4096 segments or points.



HP IBASIC Program Example (SPIKES)

This program is similar to the "SIN_D" BASIC program on page 76, with the following differences:

```
1  !RE-SAVE"SPIKES"
2  !This program generates a sine wave with a spike.
   •
   •
150 !Set AFG parameters
160 OUTPUT @Afg;"SOUR:FREQ:FIX 1E3,";!frequency
170 OUTPUT @Afg;"SOUR:FUNC:SHAP USER,";!function
180 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 1.5V" !scale
   amplitude
190 !
200 !Call subprogram which defines waveform segment.
210 CALL Sine_spike
   •
   •
310 SUB Sine_spike
320 Sine_spike: !Subprogram which defines a sine wave with a spike
330   COM @Afg,Waveform(*)
340   FOR I=1 TO 4096
350     Waveform(I)=.5*SIN(2*PI*(I/4096))
360   NEXT I
370   Width=50
```

Continued on next page

```

380   FOR J=1 TO Width/2
390       Waveform(J+1024)=Waveform(J+1024)+J*.04
400   NEXT J
410   FOR J=1 TO Width/2
420       Waveform(J+1024+Width/2)=Waveform(J+1024+Width/2)
         +1-(J*.04)
430   NEXT J
440   !
450   OUTPUT @Afg;"SOUR:LIST:SEGM:SEL A"!select segment name
460   OUTPUT @Afg;" SOUR:LIST:SEGM:VOLT";Waveform(*)!load
         waveform points
470   SUBEND

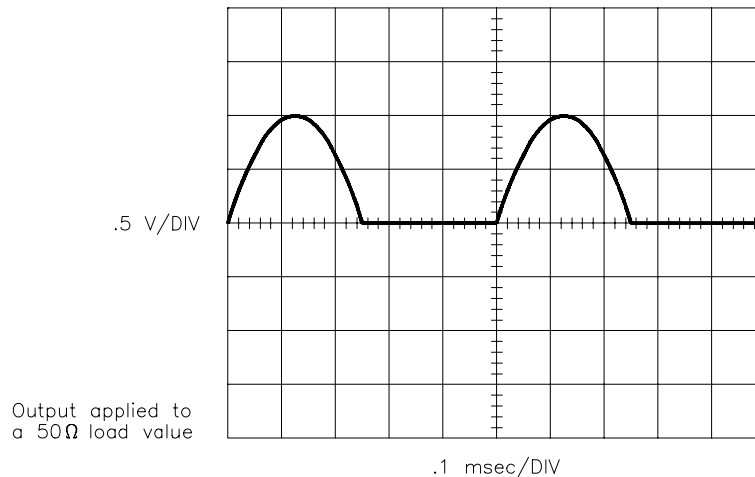
```

C and QuickBASIC Program Versions

The C example program, SPIKES.C, is in directory "CPROGRAM" and the QuickBASIC example program, SPIKES.BAS, is in directory "QBPROGRAM" on the C and QuickBASIC example disk (part number E1340-10034).

Generating a Half-Rectified Sine Wave

This program generates a Half-Rectified Sine Wave using 4096 segments or points.



HP IBASIC Program Example (SIN_R)

This program is similar to the “SIN_D” BASIC program on page 76, with the following differences:

```
1    !RE-SAVE“SIN_R”
2    !This program generates a 1/2 wave rectified sine wave.
    •
    •
210  CALL Rec_sine
    •
    •
310  SUB Rec_sine
320  Rec_sine:    !Subprogram which defines a rectified sine wave
330      COM @Afg,Waveform(*)
340      FOR I=1 TO 4096
350          Waveform(I)=SIN(2*PI*(I/4096))
360      NEXT I
370      FOR I=2049 TO 4096
380          Waveform(I)=0
390      NEXT I
400      !
410      OUTPUT @Afg;“SOUR:LIST:SEGM:SEL A”!select segment name
420      OUTPUT @Afg;“SOUR:LIST:SEGM:VOLT”;Waveform(*)!load
                                         waveform points
430  SUBEND
```

C and QuickBASIC Program Versions

The C example program, SIN_R.C, is in directory “CPROG” and the QuickBASIC example program, SIN_R.BAS, is in directory “QBPROG” on the C and QuickBASIC example disk (part number E1340-10036).

Program Comments

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

Amplitude Effects on Voltage List

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.

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 - 42.94967296 MHz (power-on value)

EXTernal - User provided value (the front panel "Aux In" BNC)

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

Waveforms in the EPRom

The following are the waveforms located in the EPRom:

EEPRom1	Sine Wave
EEPRom2	Triangle Wave
EEPRom3	Sin(x)/x for 8.25 radians
EEPRom4	Haversine
EEPRom5	Square Wave
EEPRom6	First ten terms of a Square Wave
EEPRom7	First four terms of a Square Wave
EEPRom8	Falling Ramp
EEPRom9	First 20 terms of a Falling Ramp
EEPRom10	Rising Ramp
EEPRom11	First 20 terms of a Rising Ramp
EEPRom12	White Noise
EEPRom13	Modulated White Noise
EEPRom14	A 3rd, 4th, 5th Harmonic Chord
EEPRom15	8 cycles of a linear rising Sine Wave
EEPRom16	Positive half-cycle Sine Wave

Chapter 4

HP E1340A Sweeping and Frequency-Shift Keying

Chapter Contents

This chapter covers the sweeping and frequency-shift keying (FSK) features of the HP E1340A Arbitrary Function Generator (AFG). The chapter is organized as follows:

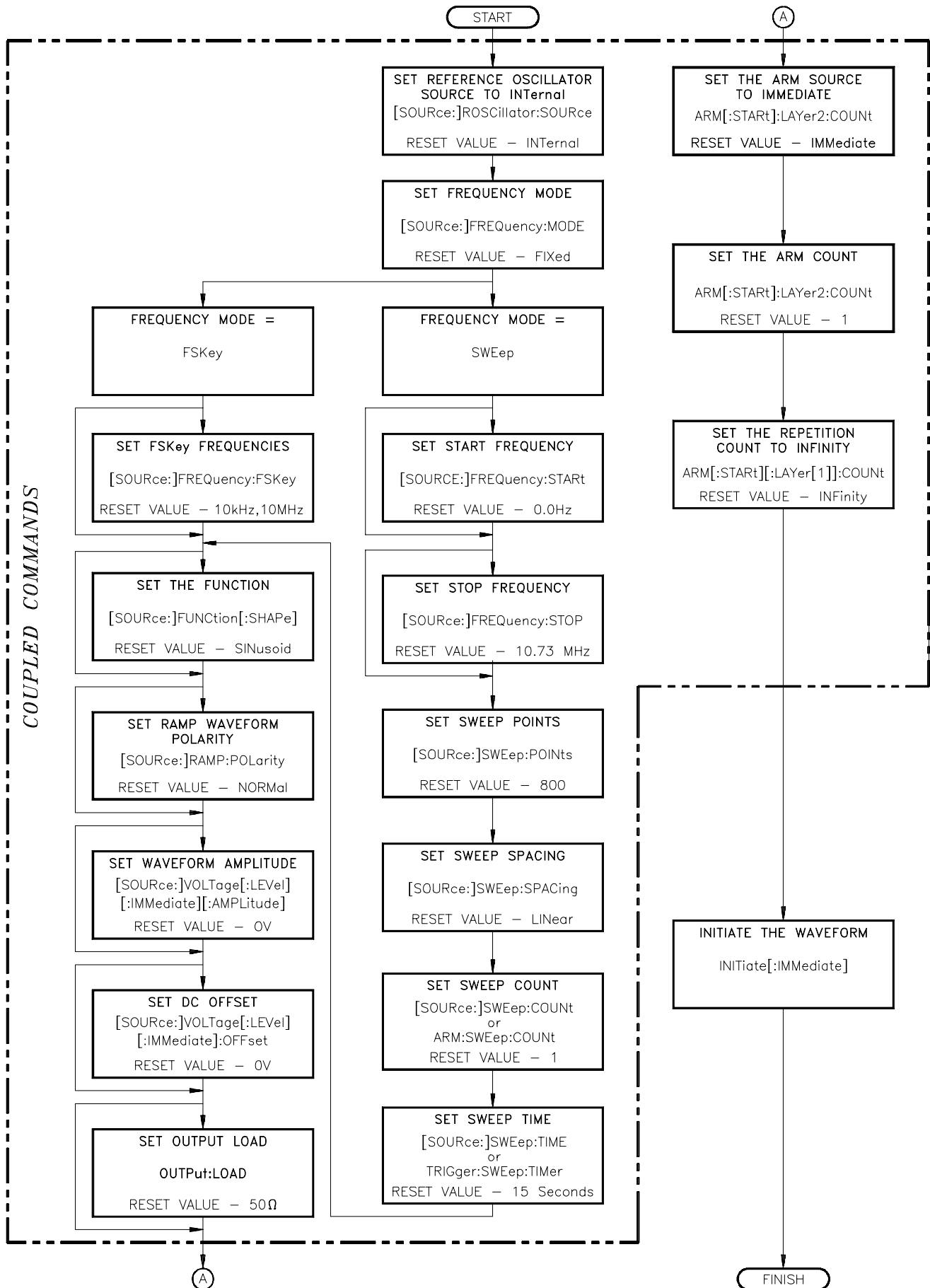
- Sweeping Page 85
 - Sweeping Using Start and Stop FrequenciesPage 85
 - Sweeping Using Start and Span FrequenciesPage 88
 - Sweep Points Vs. Sweep Time Page 90
- Frequency-Shift Keying..... Page 92
- Program CommentsPage 94

The Command Reference

Detailed information on the commands introduced in this chapter can be found in Chapter 7 under “Command Reference”. The commands in this chapter are shown in their entirety (optional headers included) to help you locate them in the reference.

Programming Flowchart

The flowchart on page 84 shows the commands and the command execution order to generate standard 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.



Sweeping

The AFG offers linear frequency sweeping of standard waveforms (i.e. sine, square, triangle, ramp) and arbitrary waveforms from 0.0 Hz to 15 MHz for sine waves, and 0.0 Hz to 1 MHz for square, triangle, and ramp waves.

The command subsystems covered in this section include:

- [SOURce:]ROSCillator
- [SOURce:]FREQuency
- [SOURce:]SWEep

Sweeping Considerations

The AFG only allows for continuous repetitive sweeping. The INITiate[:IMMEDIATE] command starts the sweeping at a fixed rate determined by the number of points in the sweep and the sweep time. For more information on sweeping, see Chapter 7 under the [SOURce:]SWEep subsystem.

Sweeping Using Start and Stop Frequencies

The following 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 as a guide, the steps of this program are:

1. Select the Reference Oscillator -
[SOURce:]ROSCillator:SOURce <source>
2. Select the Frequency Sweep Mode -
[SOURce:]FREQuency:MODE SWEep
3. Set the Start Frequency -
[SOURce:]FREQuency:STARt <start_freq>
4. Set the Stop Frequency -
[SOURce:]FREQuency:STOP <stop_freq>
5. Select 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 IBASIC (SMPLSWP1)

```
1  !RE-SAVE"SMPLSWP1"
2  !This program sweeps a sine wave from 0 Hz to 1 MHz by specifying
3  !start and stop frequencies.
4  !
10 !Assign I/O path between IBASIC and the E1340A.
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 subprogram which resets the AFG.
130 CALL Rst
140 !
150 !Set AFG parameters
160 OUTPUT @Afg;"SOUR:ROSC:SOUR INT;"!reference oscillator
170 OUTPUT @Afg;"SOUR:FREQ:MODE SWE;"!frequency mode
180 OUTPUT @Afg;"SOUR:FREQ:STAR 0;"!start frequency
190 OUTPUT @Afg;"SOUR:FREQ:STOP 1E6;"!stop frequency
200 OUTPUT @Afg;"SOUR:FUNC:SHAP SIN;"!function
210 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5V"!amplitude
220 OUTPUT @Afg;"INIT:IMM"!wait-for-arm state
230 !
240 !Call subprogram which queries sweep parameters
250 CALL Query
260 !
270 WAIT .1!allow interrupt to be serviced
280 OFF INTR 8
290 END
300 !
310 SUB Query
320 Query: !Subprogram which queries sweep parameters
330     COM @Afg
340     OUTPUT @Afg;"SOUR:FREQ:STAR?"
350     ENTER @Afg;Sweep1$
360     OUTPUT @Afg;"SOUR:FREQ:STOP?"
370     ENTER @Afg;Sweep2$
380     OUTPUT @Afg;"SOUR:FREQ:CENT?"
390     ENTER @Afg;Sweep3$
```

Continued on next page

```

400     OUTPUT @Afg;"SOUR:FREQ:SPAN?"
410     ENTER @Afg;Sweep4$
420     PRINT "START = ";Sweep1$,"STOP = ";Sweep2$
430     PRINT
440     PRINT "CENTER = ";Sweep3$,"SPAN = ";Sweep4$
450 SUBEND
460 !
470 SUB Rst
    •
    •
540 SUB Errmsg
    •
    •

```

The start, stop, center, and span values returned are:

```

START = +0.000000000E+000      STOP = +1.000000000E+006
CENTER = +5.000000000E+005     SPAN = +1.000000000E+006

```

C and QuickBASIC Programs

The C program SMPLSWP1.C is in directory "CPROG", and the QuickBASIC program SMPLSWP1.BAS is in directory "QBPROG" on the example program disk HP P/N E1340-10036.

Sweeping Using Start and Span Frequencies

The following 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 as a guide, the steps of this program are:

1. Select the Frequency Sweep Mode -
[SOURce:]FREQuency:MODE SWEep
2. Set the Start Frequency -
[SOURce:]FREQuency:STARt <start_freq>
3. Set the Frequency Span -
[SOURce:]FREQuency:SPAN <freq_span>
4. Set the Output Function -
[SOURce:]FUNCTio[n][:SHAPE] <shape>
5. Set the Signal Amplitude -
[SOURce:]VOLTag[e][:LEVel][:IMMediate][:AMPLitude] <amplitude>
6. Place the AFG in the Wait-for-arm State -
INITiate[:IMMediate]

HP IBASIC (SMPLSWP2)

```
1  !RE-SAVE"SMPLSWP2"
2  !This program sweeps a sine wave from 1 kHz to 21 kHz by
3  !specifying a start frequency and a frequency span.
4  !
10 !Assign I/O path between IBASIC and the E1340A.
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 subprogram which resets the AFG.
130 CALL Rst
140 !
150 !Set AFG parameters
160 OUTPUT @Afg;"SOUR:FREQ:MODE SWE;" ;!frequency mode
170 OUTPUT @Afg;" :SOUR:FREQ:STAR 1E3;" ;!start frequency
180 OUTPUT @Afg;" :SOUR:FREQ:SPAN 20E3;" ;!frequency span
190 OUTPUT @Afg;" :SOUR:FUNC:SHAP SIN;" ;!function
```

Continued on next page


```

200 OUTPUT @Afg;“:SOUR:VOLT:LEV:IMM:AMPL 5V” !amplitude
210 OUTPUT @Afg;“INIT:IMM” !wait-for-arm state
220 !
230 !Call subprogram which queries sweep parameters
240 CALL Query
250 !
260 WAIT .1 !allow interrupt to be serviced
270 OFF INTR 8
280 END
290 !
300 SUB Query
310 Query: !Subprogram which queries sweep parameters
320 COM @Afg
330 OUTPUT @Afg;“SOUR:FREQ:STAR?”
340 ENTER @Afg;Sweep1$
350 OUTPUT @Afg;“SOUR:FREQ:STOP?”
360 ENTER @Afg;Sweep2$
370 OUTPUT @Afg;“SOUR:FREQ:CENT?”
380 ENTER @Afg;Sweep3$
390 OUTPUT @Afg;“SOUR:FREQ:SPAN?”
400 ENTER @Afg;Sweep4$
410 PRINT “START = ”;Sweep1$,”STOP = ”;Sweep2$
420 PRINT
430 PRINT “CENTER = ”;Sweep3$,”SPAN = ”;Sweep4$
440 SUBEND
450 !
460 SUB Rst
    •
    •
530 SUB Errmsg
    •
    •

```

The start, stop, center, and span values returned are:

```

START = +1.000000000E+003      STOP = +2.100000000E+004
CENTER = +1.100000000E+004    SPAN = +2.000000000E+004

```

C and QuickBASIC Programs

The C program SMPLSWP2.C is in directory “CPROGRAM”, and the QuickBASIC program SMPLSWP2.BAS is in directory “QBPROGRAM” on the example program disk HP P/N E1340-10036.

Sweep Points Vs. Sweep Time

To demonstrate the relationship between the number of points (frequencies) in a frequency sweep and the time of the sweep, the following program uses 100 frequency points to continuously sweep from 5 kHz to 15 kHz in 0.5 seconds. (See “Program Comments” to determine the sweep time.)

Using the flowchart as a guide, the steps of this program are:

1. Select the Frequency Sweep Mode -
[SOURce:]FREQuency:MODE SWEEp
2. Set the Start Frequency -
[SOURce:]FREQuency:STARt <start_freq>
3. Set the Stop Frequency -
[SOURce:]FREQuency:STOP <stop_freq>
4. Set the Number of Points (frequencies) in the Frequency Sweep -
[SOURce:]SWEep:POINts <number>
5. Set the Sweep Repetition Time -
[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 IBASIC (SWP_PVST)

```
1 !RE-SAVE"SWP_PVST"
2 !This program sweeps a sine wave from 5 kHz to 15 kHz
3 !in 0.5 seconds to demonstrate how to set the sweep time.
4 !
10 !Assign I/O path between IBASIC and the E1340A.
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 subprogram which resets the AFG.
130 CALL Rst
```

Continued on next page

```

140  !
150  !Set AFG parameters
160  OUTPUT @Afg;“SOUR:FREQ:MODE SWE;”;!frequency mode
170  OUTPUT @Afg;“:SOUR:FREQ:STAR 5E3;”;!start frequency
180  OUTPUT @Afg;“:SOUR:FREQ:STOP 15E3;”;!stop frequency
190  OUTPUT @Afg;“:SOUR:SWE:POIN 100;” !sweep points
200  OUTPUT @Afg;“:SOUR:SWE:TIME .495;” !sweep time
210  OUTPUT @Afg;“:SOUR:FUNC:SHAP SIN;” !function
220  OUTPUT @Afg;“:SOUR:VOLT:LEV:IMM:AMPL 5V” !amplitude
230  OUTPUT @Afg;“INIT:IMM” !wait-for-arm state
240  !
250  WAIT .1 !allow interrupt to be serviced
260  OFF INTR 8
270  END
280  !
290  SUB Rst
    •
    •
360  SUB Errmsg
    •
    •

```

C and QuickBASIC Programs

The C program SWP_PVST.C is in directory “CPROG”, and the QuickBASIC program SWP_PVST.BAS is in directory “QBPROG” on the example program disk HP P/N E1340-10036.

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 20 MHz. The frequency-shift keying control source is applied to the front panel “Aux In” BNC Connector.

The command subsystems associated with frequency-shift keying include:

- [SOURce:]ROSCillator
- [SOURce:]FREQuency
 - FSK mode and related commands

Program Example

The following program shows the basic steps involved in setting up and using the frequency-shift keying function of the AFG. A 5V, 1 kHz square wave control signal is applied to the AFG’s front panel “FSK” port. Output frequencies of 20 kHz and 10 kHz occur as the level of the 1 kHz signal changes.

Using the flowchart as a guide, the steps of this program are:

1. Select the Reference Oscillator -
[SOURce:]ROSCillator:SOURce <source>
2. Select the Frequency-shift Keying Mode -
[SOURce:]FREQuency:MODE FSK
3. Select the FSK Frequencies -
[SOURce:]FREQuency:FSKey <frequency1>,<frequency2>
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 IBASIC (FSK1)

```
1  !RE-SAVE"FSK1"
2  !This program sets up frequency-shift keying using the AFG's "Aux In"
3  !port as the control source. A high signal on the port selects the
4  !first frequency. A low signal on the port selects the second frequency.
5  !
10 !Assign I/O path between IBASIC and the E1340A.
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 subprogram which resets the AFG.
130 CALL Rst
140 !
150 !Set AFG parameters
160 OUTPUT @Afg;"SOUR:ROSC:SOUR INT;"!reference oscillator
170 OUTPUT @Afg;"SOUR:FREQ:MODE FSK;"!frequency mode
180 OUTPUT @Afg;"SOUR:FREQ:FSK 10E3,20E3;"!FSK frequencies
190 OUTPUT @Afg;"SOUR:FUNC:SHAP SIN;"!function
200 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5V" !amplitude
210 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
220 !
230 WAIT .1 !allow interrupt to be serviced
240 OFF INTR 8
250 END
260 !
270 SUB Rst
    •
    •
340 SUB Errmsg
    •
    •
```

FSK frequencies are: +1.000000000E+004,+2.000000000E+004

C and QuickBASIC Programs

The C program FSK1.C is in directory "CPROG", and the QuickBASIC program FSK1.BAS is in directory "QBPROG" on the example program disk HP P/N E1340-10036.

Program Comments

The following information is associated with sweeping 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 two reference oscillator sources for the AFG which are selected by the [SOURCE:]ROSCillator:SOURCE command:

EXTernal: The AFG's front panel "Aux In" BNC .

INTernal: The internal 42.94967296 MHz oscillator.

The INTernal reference oscillator gives a frequency resolution of .01 Hz.

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

AFG Frequency Modes

There are three frequency "modes" available in the AFG. The modes selected by the [SOURCE:]FREQUENCY:MODE command are:

CW|FIXed: Single frequency mode.

FSKey: Frequency shift keying mode.

SWEep: Frequency sweep mode.

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 1001. The number of points is set with the SOURCE:SWEep:POINTs command and applies to sweeps only.

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).

The duration (t) of each frequency 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:

$$\text{Sweep time}_{\text{specified}} = \text{Sweep repetition time}_{\text{desired}} * ((\text{points} - 1)/\text{points})$$

Chapter Contents

This chapter shows you how to arm and gate the HP E1340A AFG in order to start and advance standard and arbitrary waveforms. This chapter also shows how generate the different marker outputs at the "Aux Out" BNC connector. The sections of this chapter include:

- The ARM Configuration Page 95
- Initiating Waveforms Page 96
- Arming the AFG Page 96
 - Setting the Arm Source Page 97
 - Setting the Arm and Waveform Cycle Count Page 99
- Gating the Waveforms Page 101
- Aborting Waveforms Page 102
- Marker Outputs Page 103
- Program Comments Page 107

The ARM Configuration

The HP E1340A AFG uses an ARM triggering configuration to output the standard and arbitrary waveforms. When initiated, an arm signal enables the AFG to output the waveforms each time an arm signal is received.

The ARM States

The AFG operates within three states: Idle, Wait-for-arm, and Instrument Action.

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 INITiate[:IMMEDIATE] command.

The AFG starts instrument action when it receives an arm from the specified arm source.

After the instrument action (waveform is output) occurs, 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:

```
INITiate[:IMMEDIATE]
```

INITiate is an uncoupled command and is generally the last command executed before a waveform is output:

```
100 SUB Sine_wave
110 Sine_wave: !Subprogram which outputs a sine wave
120     COM @Afg
130     OUTPUT @Afg;"SOUR:FREQ:FIX 1E3;"!frequency
140     OUTPUT @Afg;"SOUR:FUNC:SHAP SIN;"!function
150     OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5 V"!amplitude
160     OUTPUT @Afg;"INIT:IMM"    !wait-for-arm state
170 SUBEND
```

If 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 output a waveform, it 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 INITiate[:IMMEDIATE] command
- the number of waveform cycles per arm

The arming commands include:

```
ARM
[:START[SEQUENCE[1]]
[:LAYER[1]]
:COUNT <number>
:LAYER2
:COUNT <number>
:SLOPE <edge>
:SOURCE <source>
```

The arming commands for continuous waveforms are coupled commands.

Note

Detailed information on the commands introduced in this chapter can be found in Chapter 7 - "Command Reference". The commands in this chapter are shown in their entirety (optional headers included) to help you locate them in the reference.

Setting the Arm Source

The following program shows how to select the source which arms the AFG. The program selects the AFG's "Aux In" BNC connector as the arming source. When the AFG receives an arming signal, it outputs a 10 kHz, 1 Vpp square wave.

The steps of this program are:

1. Select the FIXed Frequency Mode -
[SOURce:]FREQUency:MODE *<mode>*
2. Set the Output Frequency -
[SOURce:]FREQUency[:CW]:FIXed *<frequency>*
3. Set the Output Function -
[SOURce:]FUNCTION[:SHAPE] *<shape>*
4. Set the Signal Amplitude -
[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] *<amplitude>*
5. Set the Arm Source -
ARM[:START]:LAYer2:SOURce *<source>*
6. Set the Edge of the External Arm Signal -
ARM[:START]:LAYer2:SLOPe *<edge>*
7. Place the AFG in the Wait-for-arm State -
INITiate[:IMMediate]

HP IBASIC (EXT_ARM)

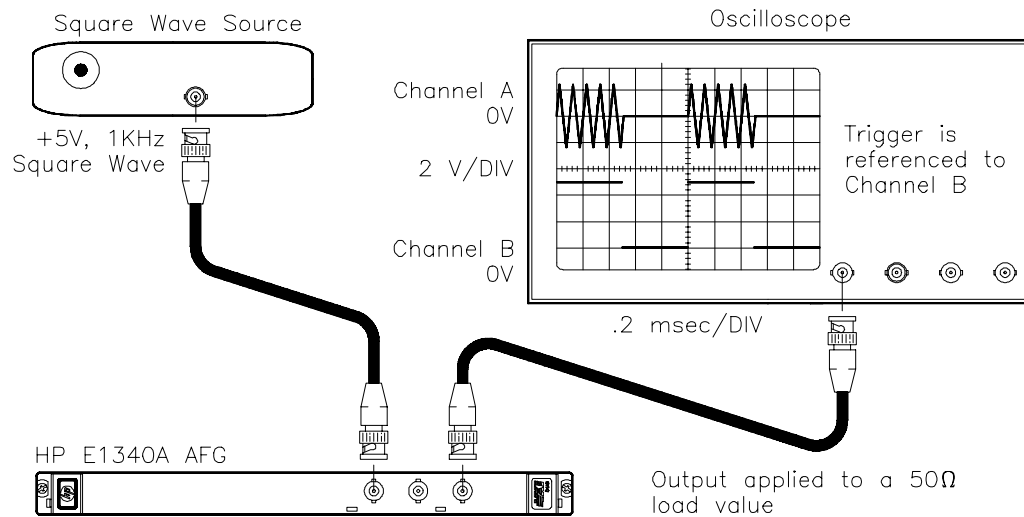
```
1  !RE-SAVE"EXT_ARM"
2  !This program arms the AFG with an external signal applied to the
3  !AFG's "Aux In" port. When armed, the AFG outputs 100 cycles
4  !of a 10 kHz, 1 Vpp square wave.
5  !
10 !Assign I/O path between IBASIC and the E1340A.
20 ASSIGN @Afg TO 80910
30 COM @Afg
40 !
50 !Set up 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 subprogram which resets the AFG.
130 CALL Rst
140 !
150 !Set AFG parameters
160 OUTPUT @Afg;"SOUR:FREQ:MODE FIX;"!frequency mode
170 OUTPUT @Afg;"SOUR:FREQ:FIX 1E3;"!frequency
180 OUTPUT @Afg;"SOUR:FUNC:SHAP SQU;"!function
190 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 1VPP;"!amplitude
200 OUTPUT @Afg;"ARM:STAR:LAY2:SOUR EXT;"!arm source
210 OUTPUT @Afg;"ARM:STAR:LAY2:COUN INF;"!arm count
220 OUTPUT @Afg;"ARM:STAR:LAY1:COUN 1E3"!cycle count
230 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
240 !
250 WAIT .1 !allow interrupt to be serviced
260 OFF INTR 8
270 END
280 !
290 SUB Rst
    •
    •
360 SUB Errmsg
    •
    •
```

C and QuickBASIC Programs

The C program EXT_ARM.C is in directory "CPROG", and the QuickBASIC program EXT_ARM.BAS is in directory "QBPROG" on the example program disk HP P/N E1340-10036.

Setting the Arm and Waveform Cycle Count

The following 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 the AFG receives an external arm. There will be about a 50 msec delay between a transition on the "Aux In" connector and the output switching.

The steps of this program are:

1. Set the Output (Burst) Frequency -
[SOURce:]FREQUency[:CW]:FIXed] <frequency>
2. Set the Output Function -
[SOURce:]FUNCtion[:SHAPE] <shape>
3. Set the Signal Amplitude -
[SOURce:]VOLTage[:LEVEL][:IMMEDIATE][:AMPLitude] <amplitude>
4. Set the Arm Source -
ARM[:START]:LAYer2:SOURce <source>
5. Set the Edge of the External Arm Signal -
ARM[:START]:LAYer2:SLOPe <edge>
6. Set the Arm Count -
ARM[:START]:LAYer2:COUNt <number>
7. Set the Number of Waveform Cycles (Burst Count) -
ARM[:START][:LAYer[1]]:COUNt <number>
8. Place the AFG in the Wait-for-arm State -
INITiate[:IMMEDIATE]

HP IBASIC (BURST)

```
1  !RE-SAVE“BURST”
2  !This program sets the arm count to infinity and the waveform
3  !repetition count to 5. The arm source is set to external and a
4  !1 kHz square wave is applied to "Aux In" connector. The AFG outputs
5  !a 5 cycle burst on each positive edge of the external arm signal.
6  !
10 !Assign I/O path between IBASIC and the E1340A.
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 subprogram which resets the AFG.
130 CALL Rst
140 !
150 !Set AFG parameters
160 OUTPUT @Afg;“SOUR:FREQ:FIX 10E3;”;!frequency
170 OUTPUT @Afg;“:SOUR:FUNC:SHAP SIN;”; !function
180 OUTPUT @Afg;“:SOUR:VOLT:LEV:IMM:AMPL 2.5VPP;”; !amplitude
190 OUTPUT @Afg;“:ARM:STAR:LAY2:SOUR EXT;”; !arm source
200 OUTPUT @Afg;“:ARM:STAR:LAY2:SLOP POS;”; !arm slope
210 OUTPUT @Afg;“:ARM:STAR:LAY2:COUN INF;”; !arm count
220 OUTPUT @Afg;“:ARM:STAR:LAY1:COUN 5” !cycle count
230 OUTPUT @Afg;“INIT:IMM” !wait-for-arm state
240 !
250 WAIT .1 !allow interrupt to be serviced
260 OFF INTR 8
270 END
280 SUB Rst
    •
    •
360 SUB Errmsg
    •
    •
```

C and QuickBASIC Programs

The C program BURST.C is in directory “CPROG”, and the QuickBASIC program BURST.BAS is in directory “QBPROG” on the example program disk HP P/N E1340-10036.

Gating the Waveforms

Gating a waveform is the process of suspending the output waveform. This is accomplished by gating the reference oscillator on and off. An active gate suspends waveform output at the point when the AFG receives the gate. Set the gate inactive and waveform output resumes.

The following example shows how to use the AFG's "Aux In" BNC to suspend waveform output. A high (TTL level) 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 Output (Burst) Frequency -
[SOURce:]FREQUENCY[:CW]:FIXed <frequency>
3. Set the Output Function -
[SOURce:]FUNCTION[:SHAPE] <shape>
4. Set the Signal Amplitude -
[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>
5. Set the Gate State On -
[SOURce:]ROSCillator:GATE:STATE <state>
6. Place the AFG in the Wait-for-arm State -
INITiate[:IMMediate]

HP IBASIC (GATE_SIG)

```
1  !RE-SAVE"GATE_SIG"
2  !This program gates the output of a 10 kHz triangle wave. When
3  ! the gate signal is high, the gate is active and the waveform is
4  !suspended. When the gate signal is low, the gate is inactive and
5  !the waveform resumes.
6  !
10 !Assign I/O path between IBASIC and the E1340A.
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"
```

Continued on next page

```

100 OUTPUT @Afg;"*ESE 60"
110 !
120 !Call the subprogram which resets the AFG.
130 CALL Rst
140 !
150 !Set AFG parameters
160 OUTPUT @Afg;"SOUR:FREQ:FIX 10E3,";!frequency
170 OUTPUT @Afg;"SOUR:FUNC:SHAP TRI,";!function
180 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5V"!amplitude
190 OUTPUT @Afg;"SOUR:ROSC:GATE:STAT ON"!enable gating
200 OUTPUT @Afg;"INIT:IMM"!wait-for-arm state
210 !
220 WAIT .1!allow interrupt to be serviced
230 OFF INTR 8
240 END
250 !
260 SUB Rst
    •
    •
330 SUB Errmsg
    •
    •

```

C and QuickBASIC Programs

The C program GATE_SIG.C is in directory "CPROG", and the QuickBASIC program GATE_SIG.BAS is in directory "QBPROG" on the example program disk HP P/N E1340-10036.

Aborting Waveforms

Aborting a waveform places the AFG in the Idle state. The waveform is halted and the output remains at the last amplitude point when the abort was executed. The command which aborts a waveform is:

ABORT

Marker Outputs

The following describes the different marker outputs generated at the AFG's front panel "Aux Out" BNC connector.

Marker Commands

The marker commands are:

```
[SOURce:]  
MARKer  
:FEED <source>  
:POLarity <polarity>
```

The marker commands are coupled commands.

Available Marker Sources

There are three marker sources available for output at the AFG's front panel "Aux Out" BNC. The different marker sources are as follows:

OUTPut:ZERO

This source outputs a marker pulse whenever the waveform crosses through zero. Set the output polarity to INVerted using the [SOURce:]MARKer:POLarity command. This outputs a positive marker pulse whenever the waveform crosses zero. Marker output is high when the main output is <0V and low when the main output is >0V. Note that since an analog comparator detects zero-crossing, it may not detect exact crossing. Also, at high frequencies, the comparator's sensitivity declines and it may not detect signal excursions much less than the DAC's full scale range for both negative and positive polarities.

SEGMENT

This source outputs a marker pulse at the completion of a segment. For multiple segment waveforms (AB or ABCD), the AFG outputs a marker pulse at the end of each segment 2 or 4 pulses per repetition of the waveforms. Set the output polarity to NORMal using the [SOURce:]MARKer:POLarity command. This outputs a positive marker pulse.

[SOURce:]ROSCillator

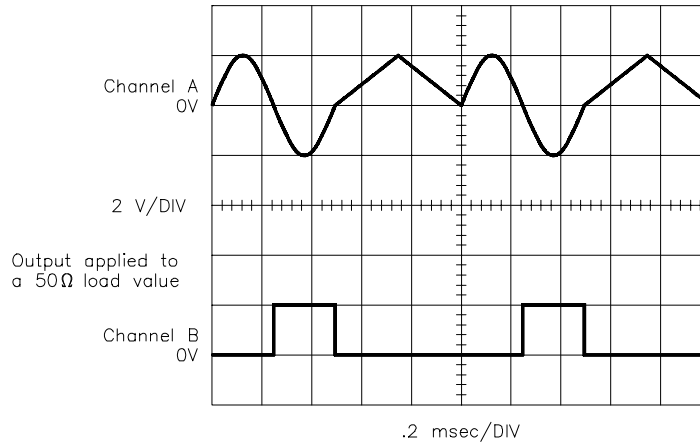
The source outputs the internal reference oscillator signal. Set the output polarity to NORMal using the [SOURce:]MARKer:POLarity command. This outputs a TTL compatible oscillator signal.

SWEep

This source outputs a marker pulse on the last point of a frequency sweep. Set the output polarity either to **NORMAL** or **INVERTed** using the `[SOURCE:]MARKer:POLarity` command.

Generating Zero Crossing Marker Pulses

This program outputs a marker pulse whenever the waveform crosses through zero. The program generates an arbitrary waveform using two different waveform segments. One waveform segment generates a 1 kHz, 5 V sine wave. The other one generates a 1 kHz, 5 V peak triangle.



The commands are:

1. Reset the AFG - `*RST`
2. Setup the AFG for an Arbitrary Waveform Output -
`[SOURCE:]FREQuency[:CW|FIXed] <frequency>`
`[SOURCE:]FUNCTion[:SHAPE] USER`
`[SOURCE:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>`
3. Select the Marker Source -
`[SOURCE:]MARKer:FEED <source>`
4. Select the Marker Pulse Polarity -
`[SOURCE:]MARKer:POLarity <polarity>`
5. Generate and Store the First Waveform Segment -
`[SOURCE:]LIST[:SEGMENT]:SElect A`
`[SOURCE:]LIST[:SEGMENT]:VOLTage <voltage_list>`
6. Generate and Store the Second Waveform Segment -
`[SOURCE:]LIST[:SEGMENT]:SElect B`
`[SOURCE:]LIST[:SEGMENT]:VOLTage <voltage_list>`
7. Generate the Output -
`[SOURCE:]FUNCTion:USER AB`
`INITiate[:IMMediate]`

HP IBASIC (MARK_OUT)

```
1  !RE-SAVE"MARK_OUT"
2  !This program outputs marker pulses with all amplitude points less
3  !than 0V. The output sequence consists of two arbitrary waveforms.
4  !
10 !Assign I/O path between IBASIC and the E1340A.
20 ASSIGN @Afg TO 80910
30 COM @Afg,Waveform1(1:4096),Waveform2(1:4096)
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 subprogram which resets the AFG.
130 CALL Rst
140 !
150 !Set AFG parameters
160 OUTPUT @Afg;"SOUR:FREQ:FIX 10E3;"!frequency
170 OUTPUT @Afg;"SOUR:FUNC:SHAP USER;"!function
180 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5.1V"!scale
                                         amplitude
190 OUTPUT @Afg;"SOUR:MARK:FEED ""OUTP:ZERO"";"!marker
                                         source
200 OUTPUT @Afg;"SOUR:MARK:POL INV"!marker source polarity
210 !
220 !Call subprogram which defines waveform segments A and B.
230 CALL Sine_tri_wave
240 !
250 !Select output sequence and initiate waveform
260 OUTPUT @Afg;"SOUR:FUNC:USER AB"
270 OUTPUT @Afg;"INIT:IMM"           !wait-for-arm state
280 !
290 WAIT .1                          !allow interrupt to be serviced
300 OFF INTR 8
310 END
320 !
330 SUB Sine_tri_wave
340 Sine_tri_wave: !Subprogram which defines waveforms
350     COM @Afg,Waveform1(*),Waveform2(*)
360     FOR I=1 TO 4096
370         Waveform1(I)=5*(SIN(2*PI*(I/4096)))
380     NEXT I
```

Continued on next page

```

390      !
400      OUTPUT @Afg;"SOUR:LIST:SEGM:SEL A"!select segment name
410      OUTPUT @Afg;" SOUR:LIST:SEGM:VOLT";Waveform1(*)
           !load waveform points

420      !
430      FOR I=1 TO 2047
440          Waveform2(I)=I*.0024414
450      NEXT I
460      FOR I=2048 TO 4096
470          Waveform2(I)=(4096-I)*.0024414
480      NEXT I
490      !
500      OUTPUT @Afg;"SOUR:LIST:SEGM:SEL B"!select segment name
510      OUTPUT @Afg;" SOUR:LIST:SEGM:VOLT";Waveform2(*)
           !load waveform points

520  SUBEND
530  !
540  SUB Rst
      •
      •
610  SUB Errmsg
      •
      •

```

C and QuickBASIC Programs

The C program MARK_OUT.C is in directory "CPROG", and the QuickBASIC program MARK_OUT.BAS is in directory "QBPROG" on the example program disk HP P/N E1340-10036.

Program Comments

The following information is associated with arming operation of the AFG.

Reference Oscillator Sources

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

EXTernal: The AFG's front panel "Aux In" BNC.

INTernal: The internal 42.94967296 MHz oscillator (default source).

AFG Frequency Modes

There are three frequency "modes" available in the AFG using ([SOURCE:]FREQUENCY). The modes selected by the [SOURCE:]FREQUENCY:MODE command are:

CW|FIXed: Single frequency mode.

FSKey: Frequency shift keying 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.

AFG Arming Sources

The arming sources set by the ARM[:START]:LAYER2:SOURCE <source> command are:

EXTernal: The HP E1340A's front panel "Start Arm In" BNC connector (TTL levels). Available for repetitive burst output only.

IMMEDIATE: Immediate arming. An arm is internally generated when the start trigger sequence enters the wait-for-arm state.

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 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 65534, or INFINITY. The default value is INFINITY.

Enabling the Gate

Before the AFG reference oscillator can be gated, the gate must be enabled. This is done with the [SOURCE:]ROSCillator:GATE:STATE command. When the mode is ON, gating is enabled. When OFF, gating is disabled.

Chapter Contents

This chapter explains how to use the HP E1340A Arbitrary Function Generator at faster speeds and other operations.

Chapter 3 shows how to transfer waveform segments to the AFG as voltage values. This is the slowest method to transfer the lists to the AFG. This chapter shows faster ways to transfer the data to the AFG. The sections are as follows:

- Using DAC Data to Generate Waveforms Page 109
- Using Definite Length Blocks to Transfer Data Page 113
- Using the VXIbus Backplane. Page 120
- Program Comments Page 129

Using DAC Data to Generate Waveforms

Transferring waveform segments as Digital-to-Analog Converter (DAC) Codes to the AFG is faster than transferring a voltage list. The following shows how to transfer the lists as DAC codes using 16-bit integers. The DAC codes are transferred to the AFG as a comma (,) separated list. Be sure to set the upper 4 bits of the integers to 0.

Determining DAC Codes

For outputs into matched loads and with the amplitude set to maximum (+5.11750V), the following DAC codes generate the following outputs:

- Code **0** outputs -5.12 V or negative full scale voltage
- Code **+2048** outputs 0 V
- Code **+4095** outputs +5.11750 V or positive full scale voltage

To calculate DAC codes from voltage values, use the formula:

$$\text{DAC Code} = (\text{voltage value} / .0025) + 2048$$

For example, to output -2V:

$$\text{DAC Code} = (-2 / .0025) + 2048 = -800 + 2048 = 1248$$

Program Example

This program shows how to store a waveform segment (i.e., points of an arbitrary waveform) into the AFG's segment memory. The program transfers the data to the AFG as a comma (,) separated list. The example generates a 4096 point +5 V to -5 V negative going ramp.

1. Reset the AFG - *RST
2. Setup the AFG for Output -
[SOURce:]FREQuency[:CW|FIXed] <frequency>
[SOURce:]FUNction[:SHAPE] USER
[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>
3. Select the DAC Data Source -
[SOURce:]ARBitrary:DAC:SOURce INTERNAL

This command selects the source that transfers data to the DAC (see "Program Comments"). Use INTERNAL to transfer the data using the [SOURce:]FUNction USER subsystem. (Reset automatically sets this command; it is given here for good programming practice.)

4. Select the Waveform Segment -
[SOURce:]LIST[:SEGment]:SElect <name>

Use either A, B, C, or D in <name> for the waveform segment.

5. Store the Waveform Segment as DAC Data -
[SOURce:]LIST[:SEGment]:VOLTage:DAC <voltage_list>

This command stores the waveform segment into segment memory using DAC codes consisting of 16-bit integers.

6. Generate the Output -
[SOURce:]FUNction:USER <name>
INITiate[:IMMediate]

HP IBASIC Program Example (UNS_DAT)

```
1  !RE-SAVE"UNS_DAT"  
2  !This program downloads AFG data in the form of DAC codes  
3  !(unsigned numbers). The data represents a 5V ramp waveform.  
4  !  
10 !Assign I/O path between IBASIC and the E1340A.  
20 ASSIGN @Afg TO 80910  
30 COM @Afg,Waveform(1:4096)  
40 !  
50 !Set up error checking  
60 ON INTR 8 CALL Errmsg  
70 ENABLE INTR 8;2
```

Continued on next page

```

80  OUTPUT @Afg;"*CLS"
90  OUTPUT @Afg;"*SRE 32"
100 OUTPUT @Afg;"*ESE 60"
110 !
120 !Call the subprogram which resets the AFG.
130 CALL Rst
140 !
150 !Set AFG parameters
160 OUTPUT @Afg;"SOUR:FREQ:FIX 1E3,";!frequency
170 OUTPUT @Afg;"SOUR:FUNC:SHAP USER,";!function
180 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5.11750V"!scale
amplitude

190 !
200 !Call subprogram which defines waveform segment A (ramp wave).
210 CALL Ramp_wave
220 !
230 !Select output sequence and initiate waveform
240 OUTPUT @Afg;"SOUR:FUNC:USER A"
250 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
260 !
270 WAIT .1 !allow interrupt to be serviced
280 OFF INTR 8
290 END
300 !
310 SUB Ramp_wave
320 Ramp_wave: !Subprogram which defines ramp waveform
330     COM @Afg,Waveform(*)
340     FOR I=2048 TO -2047 STEP -1!calculate waveform points as dac
codes
350         Waveform(2049-I)=((I*.0024426)/.0025)+2048
360     NEXT I
370     !
380     OUTPUT @Afg;"SOUR:ARB:DAC:SOUR INT"!dac data source
(internal)
390     OUTPUT @Afg;"SOUR:LIST:SEGM:SEL A"!select segment name
400     OUTPUT @Afg;"SOUR:LIST:SEGM:VOLT:DAC ";Waveform(*)
!load ampl points

410 SUBEND
420 !
430 SUB Rst
    •
    •
500 SUB Errmsg
    •
    •

```

**C and QuickBASIC
Program Versions**

The C example program, UNS_DAT.C, is in directory “CPROG” and the QuickBASIC example program, UNS_DAT.BAS, is in directory “QBPROG” on the C and QuickBASIC example disk (part number E1340-10036).

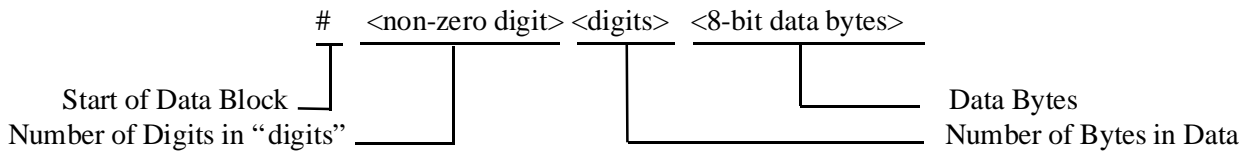
This program is very similar to the example programs used in Chapter 3. The only difference is that this program transfers the segment data as DAC codes instead of voltage values.

Using Definite Length Blocks to Transfer Data

The AFG can receive DAC codes as Definite Length Arbitrary Block Data consisting of 16-bit integers. This is a much faster method to transfer data than using a comma (,) separated list that was used in “Using DAC Data to Generate Waveforms” on page 109. Be sure to set the upper 4 bits of the integers to 0.

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 4096, the <non_zero digit> value equals 3 or 4, respectively
- “<digits>” shows the number of data bytes to be sent; for example, if 4096 data bytes are to be sent, <digits> equals 4096 (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 8192 8-bit data bytes is:

#48192<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. Note that the AFG requires that the most significant bit of each 16-bit integer be sent first.

For example, to send a waveform segment consisting of 4096 DAC codes (4096 points), the actual number of “digits” and “8-bit data bytes” equals: $4096 * 2 = 8192$.

Program Example

This program shows how to store a waveform segment (i.e., points of an arbitrary waveform) into the AFG's segment memory. The waveform segment are sent as DAC codes using the Definite Length Block Data transfer method. The example generates a 4096 point 0 to +5 V positive going ramp.

1. Reset the AFG - *RST
2. Setup the AFG for Output -
[SOURce:]FREQuency[:CW|FIXed] <frequency>
[SOURce:]FUNctio[:SHAPE] USER
[SOURce:]VOLTag[:LEVel][:IMMediate][:AMPLitude] <amplitude>
3. Select the DAC Data Source -
[SOURce:]ARBitrary:DAC:SOURce INTernal

This command selects the source that transfers data to the DAC (see "Program Comments"). Use INTernal to transfer the data using the [SOURce:]LIST[1] subsystem.

4. Select the Waveform Segment -
[SOURce:]LIST[:SEGMENT]:SElect <name>

Use either A, B, C, or D in <name> for the waveform segment.

5. Store the Waveform Segment as DAC Data -
[SOURce:]LIST[:SEGMENT]:VOLTag:DAC <voltage_list>

This command stores the waveform segment into segment memory using DAC codes sent as Definite Length Block Data.

6. Generate the Output -
[SOURce:]FUNctio:USER <name>
INITiate[:IMMediate]

HP IBASIC Program Example (DACBLOK)

To transfer Definite Length Block Data to the AFG requires that the data sent with the [SOURCE:]LIST[:SEGMENT]:VOLTage:DAC command must be contiguous. To do this, send no carriage return (CR) and line feed (LF) before all the data is transferred.

```
1  !RE-SAVE"DACBLOK"
2  !This program downloads AFG amplitude data in the form of DAC codes
3  !(unsigned numbers). The data is sent in a IEEE-488.2 definite length
4  !block in 16-bit integer format. The waveform is a 4096 point ramp wave.
5  !
10 !Assign I/O paths between IBASIC and the E1340A.
20 ASSIGN @Afg TO 80910
30 ASSIGN @Afg1 TO 80910;FORMAT OFF !path for binary data
40 COM @Afg,@Afg1,INTEGER Waveform(1:4096)
50 !
60 !Set up error checking
70 ON INTR 8 CALL Errmsg
80 ENABLE INTR 8;2
90 OUTPUT @Afg;"*CLS"
100 OUTPUT @Afg;"*SRE 32"
110 OUTPUT @Afg;"*ESE 60"
120 !
130 !Call the subprogram which resets the AFG.
140 CALL Rst
150 !
160 !Set AFG parameters
170 OUTPUT @Afg;"SOUR:FREQ:FIX 1E3;"!frequency
180 OUTPUT @Afg;"SOUR:FUNC:SHAP USER;"!function
190 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5V" !scale amplitude
200 !
210 !Call subprogram which defines waveform segment A (ramp wave).
220 CALL Ramp_wave
230 !
240 !Select output sequence and initiate waveform
250 OUTPUT @Afg;"SOUR:FUNC:USER A"
260 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
270 !
280 WAIT .1 !allow interrupt to be serviced
290 OFF INTR 8
300 END
310 !
```

```

320 SUB Ramp_wave
330 Ramp_wave: !Subprogram which defines ramp waveform
340 COM @Afg,@Afg1,INTEGER Waveform(*)
350 FOR I=1 TO 4096 !calculate waveform points as dac
           codes
360 Waveform(I)=((I*.00122)/.0025)+2048
370 NEXT I
380 !
390 OUTPUT @Afg;"SOUR:ARB:DAC:SOUR INT" !dac data source
           (internal)
400 OUTPUT @Afg;"SOUR:LIST:SEGM:SEL A" !select segment name
410 OUTPUT @Afg USING "#,K";"SOUR:LIST:SEGM:VOLT:DAC #48192"
420 OUTPUT @Afg1;Waveform(*) !8192 bytes: 4 digits
           (2 bytes/ampl point)
430 OUTPUT @Afg !CR LF
440 SUBEND
450 !
460 SUB Rst
    •
    •
530 SUB Errmsg
    •
    •

```

C and QuickBASIC Program Versions

The C example program, DACBLOK.C, is in directory "CPROG" and the QuickBASIC example program, DACBLOK.BAS, is in directory "QBPROG" on the C and QuickBASIC example disk (part number E1340-10036).

Transferring Data in C

To transfer Definite Length Block Data to the AFG requires that the data sent with the [SOURCE:LIST:SEGMENT]:VOLTage:DAC command must be contiguous. To do this, send no carriage return (CR) and line feed (LF) before transferring all the data. The send_data function does this as follows (refer to the C function below):

1. Disable EOI and EOL using IOEOI (ISC, 0) and IOEOL (ISC, " ", 0), respectively. ISC contains the HP-IB select code, and the 0 and " " (NULL) values disables the carriage return (CR) and line feed (LF) to allow the AFG to receive a command string without a terminator.
2. Next, sent to the AFG the SCPI [SOURCE:LIST[1]:SEGMENT]:VOLTage:DAC command string using IOOUTPUTS (ADDR, commands, strlen(commands)). ADDR contains the AFG HP-IB address, commands contains the SCPI command string, and strlen(commands) contains the string length of commands. This enables the AFG to receive voltage data. Since CR or LF is disabled, the AFG holds execution until all the data is sent.

3. Enable EOI and EOL using IOEOI (ISC, 1) and IOEOL (ISC, state, 2), respectively. The variable state contains the decimal codes for CR and LF. Although EOL and EOI are enabled, no CR and LF is sent until all voltage data transfer is completed (next step).
4. The DAC data values are sent next using the array Wave_seg in IOOUTPUTAB (ADDR, Wave_seg, num_bytes). Wave_seg contains the <8-bit data bytes>, num_bytes contains the value for the number of bytes to be transferred, and swap contains the swap value (see “Swapping the Data Bytes” later in this chapter). IOOUTPUTAB’s main purpose is to transfer data as Definite Length Arbitrary Block Data. It automatically determines the “<non-zero digit>” and “<digits>” and sends the complete “#<non-zero digit><digits><8-bit data bytes>” block to the AFG.

Since the DAC codes are in a 16-bit integer format, the programs send 2 “<8-bit data bytes>” for each DAC code; all values are sent in one data block. The number of data bytes transferred depends on the number in num_bytes.

5. After all the data is sent, the function sends blank data using IOOUTPUTS(ADDR, "", 0) to terminate the data transfer with a CR LF. This lets the AFG know that it can receive a new command.

```
void send_data(char *commands, int *Wave_seg, int num_bytes, int
swap)
{
    /* First disable EOI and EOL to send continuous data to the AFG;
then sent the data */
    IOEOI (ISC, 0);IOEOL (ISC, " ", 0);
    IOOUTPUTS (ADDR, commands, strlen(commands));

    /* Send the last command and data */
    IOOUTPUTAB(ADDR, Wave_seg, num_bytes, swap);

    /* Re-enable EOL and EOI for normal HP-IB operation; then send CR/LF */
    IOEOI (ISC, 1);IOEOL (ISC, state, 2);
    IOOUTPUTS(ADDR, "", 0);
}
```

Transferring Data in QuickBASIC

To transfer Definite Length Block Data to the AFG requires that the data sent with the [SOURCE:]LIST[:SEGMENT]:VOLTAGE:DAC command must be contiguous. To do this, send no carriage return (CR) and line feed (LF) before transferring all the data. The SendData routine does this as follows (refer to the QuickBASIC routine below):

1. Disable EOI and EOL using IOEOI (ISC&, 0) and IOEOL (ISC&, “ ”, 0), respectively. ISC& contains the HP-IB select code, and the 0 and “ ” (NULL) values disables the carriage return (CR) and line feed (LF) to allow the AFG to receive a command string without a terminator.
2. Next, sent to the AFG the SCPI [SOURCE:]LIST[1][:SEGMENT]:VOLTAGE:DAC command string using IOOUTPUTS(ADDR&, Commands\$, LEN(Commands\$)). ADDR& contains the AFG HP-IB address, Commands\$ contains the SCPI command string, and LEN(Commands\$) contains the string length of Commands\$. This enables the AFG to receive voltage data. Since CR or LF is disabled, the AFG holds execution until all the data is sent.
3. Enable EOI and EOL using IOEOI(ISC&, 1) and IOEOL(ISC&, Endline\$, LEN(Endline\$)), respectively. The variable Endline\$ contains the decimal codes for CR and LF. Although EOL and EOI are enabled, no CR and LF is sent until all voltage data transfer is completed (next step).
4. The DAC data values are sent next using the array WaveSeg%(1) in IOOUTPUTAB(ADDR&, SEG WaveSeg%(1), NumSize%, SwapSize%). WaveSeg% contains the <8-bit data bytes>, NumSize% contains the value for the number of bytes to be transferred, and SwapSize% contains the swap value (see “Swapping the Data Bytes” on the following page).

IOOUTPUTAB's main purpose is to transfer data as Definite Length Arbitrary Block Data. It automatically determines the “<non-zero digit>” and “<digits>” and sends the complete “#<non-zero digit><digits><8-bit data bytes>” block to the AFG.

Since the DAC codes are in a 16-bit integer format, the programs send 2 “<8-bit data bytes>” for each DAC code; all values are sent in one data block. The number of data bytes transferred depends on the number in NumSize%.

5. After all the data is sent, the function sends blank data using IOOUTPUTS(ADDR&,"",0) to terminate the data transfer with a CR LF. This lets the AFG know that it can receive a new command.

```
SUB SendData (Commands$, WaveSeg%(), NumSize%, SwapSize%)

  Endline$ = CHR$(13) + CHR$(10)

  ' First disable EOI and EOL to send continuous data to the AFG; then send
  the last command
  CALL IOEOI(ISC&, 0)
  CALL IOEOL(ISC&, " ", 0)
  CALL IOOUTPUTS(ADDR&, Commands$, LEN(Commands$))

  ' Send the data
  CALL IOOUTPUTTAB(ADDR&, SEG WaveSeg%(1), NumSize%,
  SwapSize%)

  ' Re-enable EOL and EOI for normal HP-IB operation; then send CR/LF
  CALL IOEOI(ISC&, 1)
  CALL IOEOL(ISC&, Endline$, LEN(Endline$))
  CALL IOOUTPUTS(ADDR&, "", 0)

END SUB
```

Swapping the Data Bytes

The C and QuickBASIC programs used with a PC type computer sends the least significant bit (LSB) of the data byte first and the most significant bit (MSB) last. Thus, IOOUTPUTTAB must swap the data bytes in order to send the LSB last. To do this, set swap in IOOUTPUTTAB(ADDR, Wave_seg, num_bytes, swap) (C program) or SwapSize% in IOOUTPUTTAB(ADDR&, SEG WaveSeg%(1), SwapSize%) (QuickBASIC program) to the size of the Wave_seg or WaveSeg% array type (not the size of the array). Since the DAC code is a 16-bit integer, the array type is an integer. The swap or SwapSize% value is thus the size of an integer (i.e., 2). This lets IOOUTPUTTAB know when to swap the byte.

Using the VXIbus Backplane

You can use the VXIbus Backplane to download or transfer segment data to the AFG.

Using the VXIbus Backplane to Download 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

This method downloads the segment data as DAC codes (see “Using DAC Data to Generate Waveforms” on page 109). The method transfers the codes as 16-bit integers in the Definite Length Arbitrary Block data format. The upper 4 bits of the integers must be set to 0.

Send the segment data directly into the AFG’s register address 12 decimal (0C hex) offset in the AFG’s A16 address space (see Appendix C for information on registers).

Downloading Directly into the DAC

This method immediately outputs the DAC data point when received. The DAC code received by the AFG sets the DAC to output to the received value. Send the DAC codes as 16-bit integers with the upper 4 bits set to 0.

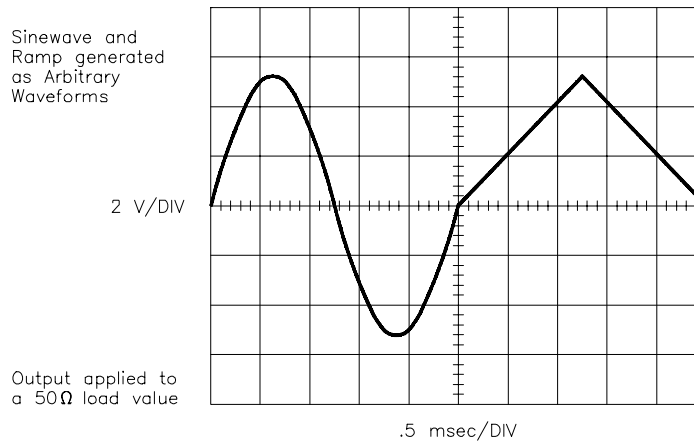
Send the DAC data directly into the AFG’s register with address 12 decimal (0C hex) offset in the AFG’s A16 address space (see Appendix C for information on registers).

Program that Downloads and Executes Waveform Segments in Memory

This program shows how to download multiple waveform segments into the AFG’s memory.

The example generates a 5 V sine wave and a 0 to +5 V triangle wave (see next figure). The commands are:

1. Reset the AFG - *RST
2. Setup the AFG for Output -
[SOURce:]FREQuency[:CW|FIXed] <frequency>
[SOURce:]FUNctio[n]:SHAPE] USER
[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>



3. Select the DAC Data Source -
[SOURce:]ARBitrary:DAC:SOURce INTernal

This command selects the source that transfers data to the DAC (see “Program Comments”). Use INTernal to transfer the data using the [SOURce:]FUNCTION USER subsystem.

4. Setup the AFG to Store the First Waveform Segment -
[SOURce:]ARBitrary:DOWNload <source>, <dest>, <length>

The <source> is “VXI” for the VXIbus, the <dest> is the name of the waveform segment to receive the data (i.e., A, B, C, or D), and <length> is the segment size (i.e., 4096).

5. Wait Until All Commands are Executed - *OPC?

This command allows the AFG to wait until it has executed all commands before the computer begins sending the data. If the computer sends the data too soon, it will be lost.

6. Setup the AFG to Store the Second Waveform Segment -
[SOURce:]ARBitrary:DOWNload <source>, <dest>, <length>

The <source> is “VXI” for the VXIbus, the <dest> is the name of the waveform segment to receive the data (i.e., A, B, C, or D), and <length> is the segment size (i.e., 4096).

7. Wait Until All Commands are Executed - *OPC?

This command allows the AFG to wait until it has executed all commands before the computer begins sending the data. If the computer sends the data too soon, it will be lost.

8. Generate, Download, and Store the First Waveform Segment as DAC Codes -

This step stores the DAC codes of the Waveform Segment 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 like the HP E1480A V360/Controller. (You can also use the HP E1405A/E1406A Command Module or HP E1300A/E1301A Mainframe, but at a slower data transfer rate.)

9. Notify the AFG that Downloading is Completed -
[SOURce:]ARBitrary:DOWNload:COMPLete

Send this command to the AFG after all data is downloaded.

10. Generate, Download, and Store the Second Waveform Segment as DAC Codes -

This step stores the DAC codes of the Waveform Segment 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 like the HP E1480A V360/Controller. (You can also use the HP E1405A/E1406A Command Module or HP E1300A/E1301A Mainframe, but at a slower data transfer rate.)

11. Notify the AFG that Downloading is Completed -
[SOURce:]ARBitrary:DOWNload:COMPLete

Send this command to the AFG after all data is downloaded.

12. Generate the Output -
[SOURce:]FUNctIon:USER <name>
INITiate[:IMMediate]

HP IBASIC Program Example (VXIDOWN)

This program is similar to the DACBLOK program on page 115, except on how the data is transferred to the AFG. The program uses IBASIC to download the data using the VXIbus.

```
1  !RE-SAVE"VXIDOWN"
2  !This program generates an arbitrary waveform comprised of two
3  !waveform segments. The segment data (DAC codes) are transferred
4  !from IBASIC to the AFG's input data register over the VXI backplane.
5  !This method of data transfer is recommended when amplitude data
6  !computation (by IBASIC) is not intensive, but fast downloading is
7  !required.
8  !
10 !Assign I/O path between IBASIC and the E1340A.
20 ASSIGN @Afg TO 80910
30 !
40 !Compute the E1340 base address in A16 address space. The base
50 !address is used with the AFG's input data register offset to form
60 !the complete register address.
70 Base_addr=DVAL("1FC000",16)+(80*64)
80 !
90 COM @Afg,Base_addr,INTEGER Wave1(1:4096),INTEGER
   Wave2(1:4096)
100 !
110 !Set up error checking
120 ON INTR 8 CALL Errmsg
130 ENABLE INTR 8;2
140 OUTPUT @Afg;"*CLS"
150 OUTPUT @Afg;"*SRE 32"
160 OUTPUT @Afg;"*ESE 60"
170 !
180 !Call the subprogram which resets the AFG.
190 CALL Rst
200 !
210 !Set AFG parameters
220 OUTPUT @Afg;"SOUR:FREQ:FIX 10E3;"!frequency
230 OUTPUT @Afg;"SOUR:FUNC:SHAP USER;"!function
240 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5.1175V"!scale
   amplitude
250 !
260 !Call subprogram which defines waveform segments A and B.
270 CALL Sine_tri_wave
280 !
290 !Select output sequence and initiate waveform
300 OUTPUT @Afg;"SOUR:FUNC:USER AB"
310 OUTPUT @Afg;"INIT:IMM" !wait-for-arm state
```

Continued on next page

```

320  !
330  WAIT .1                               !allow interrupt to be serviced
340  OFF INTR 8
350  END
360  !
370  SUB Sine_tri_wave
380  Sine_tri_wave: !Subprogram which defines and downloads waveforms
390      COM @Afg,Base_addr,INTEGER Wave1(*),INTEGER Wave2(*)
400      FOR I=1 TO 4096 !calculate waveform points as dac codes
410          Wave1(I)=(5*(SIN(2*PI*(I/4096)))/.0025)+2048
420      NEXT I
430      !
440      OUTPUT @Afg;"SOUR:ARB:DAC:SOUR INT" !dac data source
                                         (internal)
450      OUTPUT @Afg;"SOUR:ARB:DOWN VXI,A,4096"!enable
                                         download mode
460      OUTPUT @Afg;"*OPC?"             !allow AFG set up to complete
470      ENTER @Afg;Ready
480      FOR I=1 TO 4096
490          WRITEIO -9826,Base_addr+12;Wave1(I) !load waveform
                                                  points
500      NEXT I
510      OUTPUT @Afg;"*OPC?"             !allow downloading to complete
520      ENTER @Afg;Ready
530      OUTPUT @Afg;"SOUR:ARB:DOWN:COMP"!disable
                                         downloading from VXIbus
540      !
550      FOR I=1 TO 2047 !calculate waveform points as dac codes
560          Wave2(I)=((I*.0024414)/.0025)+2048
570      NEXT I
580      FOR I=2048 TO 4096
590          Wave2(I)=(((4096-I)*.0024414)/.0025)+2048
600      NEXT I
610      !
620      OUTPUT @Afg;"SOUR:ARB:DAC:SOUR INT" !dac data source
                                         (internal)
630      OUTPUT @Afg;"SOUR:ARB:DOWN VXI,B,4096"!enable
                                         download mode
640      OUTPUT @Afg;"*OPC?"             !allow AFG set up to complete
650      ENTER @Afg;Ready
660      FOR I=1 TO 4096
670          WRITEIO -9826,Base_addr+12;Wave2(I)!load waveform
                                                  points
680      NEXT I
690      OUTPUT @Afg;"*OPC?"             !allow downloading to complete
700      ENTER @Afg;Ready

```

Continued on next page

```

710      OUTPUT @Afg;"SOUR:ARB:DOWN:COMP"!disable
                                           downloading from VXIbus
720  SUBEND
730  !
740  SUB Rst
      •
      •
810  SUB Errmsg
      •
      •

```

C and QuickBASIC Program Version

The C example program, VXIDOWN.C, is in directory "CPROG" and the QuickBASIC example program, VXIDOWN.BAS, is in directory "QBPROG" on the C and QuickBASIC example disk (part number E1340-10036).

The C and QuickBASIC example programs use the HP E1405A/E1406A Command Module (version A.06.00 or above) or the HP E1300A/E1301A Mainframe (version A.06.00 or above) to download the data into memory. However, the command module/mainframe is only used to demonstrate the downloading method for C and QuickBASIC. A better method is to use an embedded controller. If you wish to use the HP E1405A/E1406A Command Module or HP E1300A/E1301A Mainframe to download data, use the method described in "Using Definite Length Blocks to Transfer Data" on page 113.

Program to Download Directly to the DAC

This program shows how to download segment data directly to the DAC as DAC codes.

The program downloads waveform segments using the VXIbus. The example program generates 500, 4096 point triangle waves.

1. Reset the AFG - *RST
2. Setup the AFG for Output -
[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude>
3. Turn Amplitude Correction Off -
CALibration:STATe:AC OFF

The AFG cannot correct the amplitude when transferring data directly to the DAC and thus must be turned off.

4. Select the DAC Data Source -
[SOURce:]ARBitrary:DAC:SOURce VXI

This command selects the source that transfers data to the DAC (see “Program Comments”). Use VXI to transfer the data directly to the DAC.

5. Wait Until All Commands are Executed - *OPC?

This command allows the AFG to wait until it has executed all commands before the computer begins sending the data. If the computer sends the data too soon, it will be lost.

6. Generate and Download the DAC Codes -

This step transfers the DAC codes to the DAC. The command or downloading method used depends on the device that downloads the data. For example, the device may be an embedded controller like the HP E1480A V360/Controller, or IBASIC installed in the HP E1300/E1301 Mainframe or HP E1405/E1406 Command Module. (You can also use an external HP-IB computer connected to the HP E1405A/E1406A Command Module or HP E1300A/E1301A Mainframe, but at a slower data transfer rate.)

HP IBASIC Program Example (VXISRCE)

This program uses the input data register to download the data directly to the DAC over the VXIbus (see Appendix C for register programming information).

```
1  !RE-SAVE"VXISRCE"
2  !This program downloads dac codes to the AFG dac from the input
3  !data register. Corresponding amplitude points are output as the
4  !codes are received.
5  !
10 !Assign I/O path between IBASIC and the E1340A.
20 ASSIGN @Afg TO 80910
30 !
40 !Compute the E1340 base address in A16 address space. The base
50 !address is used with the input data register offset to form the
60 !complete register address.
70 Base_addr=DVAL("1FC000",16)+(80*64)
80 !
90 COM @Afg,Base_addr,INTEGER Waveform(1:4096)
100 !
110 !Set up error checking
120 ON INTR 8 CALL Errmsg
130 ENABLE INTR 8;2
140 OUTPUT @Afg;"*CLS"
150 OUTPUT @Afg;"*SRE 32"
160 OUTPUT @Afg;"*ESE 60"
170 !
180 !Call the subprogram which resets the AFG.
190 CALL Rst
200 !Set AFG parameters
210 OUTPUT @Afg;"SOUR:VOLT:LEV:IMM:AMPL 5.1175V" !scale
amplitude
220 !
230 !Call subprogram which defines a triangle waveform.
240 CALL Triwave
250 !
260 WAIT .1 !allow interrupt to be serviced
270 OFF INTR 8
280 END
290 !
300 SUB Triwave
```

Continued on next page

```

310 Triwave: !Subprogram which defines a triangle waveform and
320           ! downloads it continuously to the DAC.
330   COM @Afg,Base_addr,INTEGER Waveform(*)
340   !
350   FOR I=1 TO 2047           !calculate waveform points as dac
                               !codes
360       Waveform(I)=((I*.0024414)/.0025)+2048
370   NEXT I
380   FOR I=2048 TO 4096
390       Waveform(I)=(((4096-I)*.0024414)/.0025)+2048
400   NEXT I
410   !
420   OUTPUT @AFG;"CAL:AC:STAT OFF"! turn ac cal off
430   OUTPUT @Afg;"SOUR:ARB:DAC:SOUR VXI" !dac data source
440   OUTPUT @Afg;"INIT:IMM"           !wait-for-arm state
450   OUTPUT @Afg;"*OPC?"             !allow AFG setup to complete
460   ENTER @Afg;Ready
470   LOOP
480       FOR I=1 TO 4096           !continually download dac codes
490           WRITEIO -9826,Base_addr+12;Waveform(I)
500       NEXT I
510   END LOOP
520 SUBEND
530 !
540 SUB Rst
    •
    •
610 SUB Errmsg
    •
    •

```

C and QuickBASIC Program Version

The C example program, VXISRCE.C, is in directory "CPROG" and the QuickBASIC example program, VXISRCE.BAS, is in directory "QBPROG" on the C and QuickBASIC example disk (part number E1340-10036).

The C and QuickBASIC example programs use the HP E1405A/E1406A Command Module or HP E1300A/E1301A Mainframe (version A.06.00 or above) to download the data to the DAC. However, the command module/mainframe is only used to demonstrate the downloading method for C and QuickBASIC. A better method is to use an embedded controller.

Program Comments

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

Amplitude Effects on DAC Codes

The AFG stores the 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 \geq 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.

Chapter 7

HP E1340A Command Reference

Chapter Contents

This chapter describes the **Standard Commands for Programmable Instruments** (SCPI) command set and the **IEEE 488.2 Common Commands** for the HP E1340A Arbitrary Function Generator (AFG). Included in this chapter are the following sections:

- Command Types Page 132
- SCPI Command Format Page 133
- SCPI Command Parameters Page 134
- SCPI Command Execution Page 136
- SCPI Command Reference Page 137
- Common Command Reference Page 188
- HP E1340A Command Quick Reference Page 197
- SCPI Conformance Information Page 200

ABORt	138	INITiate	146
ARM	139	[:IMMediate]	146
[:STARt SEQuence[1]]	139	OUTPut	147
[:LAYer[1]]	139	:IMPedance	147
:COUNt	139	:LOAD	148
:LAYer2	140	[SOURce:]	149
:COUNt	140	ARBitrary	149
:SLOPe	140	:DAC	149
:SOURce	141	:SOURce	149
CALibration	142	:DOWNload	150
:AC	142	:COMPlete	152
:BEgin	142	[SOURce:]	153
:POINt?	143	FREQuency	154
:DATA	142	:CENTer	155
:AC	143	[:CW]:FIXed]	155
[:DC]	143	:FSKey	156
[:DC]	144	:MODE	156
:BEgin	144	:SPAN	157
:POINt?	144	:STARt	158
:STATe	144	:STOP	158
:AC	145		

[SOURce:]	159	[SOURce:]	173
FUNction	159	SWEEp	174
[:SHAPE]	159	:COUNt	174
:USER	160	:POINts	175
		:TIME	175
[SOURce:]	161	SOURce:]	177
LIST	161	VOLtagE	177
[:SEGMENT]	161	[:LEVEl]	177
:CATalog?	161	[:IMMediate]	177
:DEFine?	162	[:AMPLitude]	177
:SElect	162	:UNIT	179
:VOLtagE	163	[:VOLtagE]	179
:DAC	164	:OFFSet	180
:POINts?	164		
:SSEquence	165	STATus	181
:CATalog?	165	:OPERation QUEStionable	182
:DEFine?	165	:CONDition?	182
:SElect	165	:ENABle	182
:SEquence?	166	[:EVENT]?	183
:SEGMents?	166	:NTRansition	183
		:PTRansition	184
[SOURce:]	167	:PRESet	184
MARKer	167	SYSTem	185
:FEED	167	:ERRor?	185
:POLarity	168	:VERsion?	185
[SOURce]	169	TRIGger	186
:RAMP	169	[:STARt SEquence[1]]	186
:POLarity	169	:COUNt	186
[SOURce]	170	:SOURce	187
:ROSCillator	170		
:FREQuency	170		
:EXTernal	170		
:GATE	171		
:STATe	171		
:SOURce	171		

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 shown below.

```
ARM
  [:START|SEQUence[1]]
    [:LAYer[1]]
      :COUNT <number>
    :LAYer2
      :COUNT <number>
      :SLOPe <edge>
      :SOURce <source>
```

ARM is the root keyword of the command, :START|SEQUence1 is the second level keyword, :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) Keywords

Implied or optional keywords 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:COUN 100
```

SCPI Command Parameters

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 ARM[:STARt]:LAYer2:SOURce <source> command where *source* can be EXTernal or IMMEDIATE.

- 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>] 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:FREQ:FIX 20E3
```

sets the frequency to 20 kHz. The value can be queried by executing:

```
SOUR:FREQ:FIX?
```

The MINimum or MAXimum value of a parameter is determined as follows:

```
SOUR:FREQ:FIX? MIN
```

```
SOUR:FREQ: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 a “coupling group”. The coupling group and associated commands are listed in Appendix B, Table B-2.

Commands not in the coupling group must precede or follow commands in the coupling group. Executing uncoupled commands in a coupling group breaks the coupling and can cause a “Settings Conflict” error. Command queries (commands with a question mark) are uncoupled commands and should be executed before or after coupled commands.

Chapter 1 also contains information on executing coupled commands.

MIN and MAX Parameters in Coupling Groups

When MINimum or MAXimum is the parameter of a command in the 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 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 a semicolon (;) and a colon (:) to link commands within different subsystems. For example:

```
SOUR:ROSC:SOUR INT;:ARM:STAR:LAY1:COUN 20
```

Commands within the same subsystem are linked with a semicolon (;). For example:

```
ARM:STAR:LAY1:COUN 20;LAY2:COUN 5
```

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 contains the SCPI commands for the HP E1340A 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 first command listed on that page.

ABORt

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 E1340A.
- 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
- **Coupled command:** No
- **Related Commands:** *OPC, *OPC?, *WAI, INITiate:IMMEDIATE
- ***RST Condition:** *RST places the HP E1340A in the trigger idle state, as if executing an ABORt command.

Example Aborting a waveform

ABOR

Place HP E1340A in idle state

ARM

The ARM subsystem operates with the TRIGger subsystem to control the starting of waveform output as follows:

- The source and slope for arming (starting) waveform generation.
- The number of waveform start arms the HP E1340A will accept before the trigger system returns to the idle state.
- The number of repetitions of a waveform that will be output for each start arm accepted.

Subsystem Syntax

```
ARM
[:START|SEQuence[1]]
[:LAYer[1]]
:COUNT <number>
:LAYer2
:COUNT <number>
:SLOPe <edge>
:SOURce <source>
```

[:START][:LAYer[1]] :COUNT ARM[:START][:LAYer[1]]:COUNT <number> selects the number of waveform repetitions to be output for each start arm accepted.

Parameter Name	Parameter Type	Range of Values	Default Units
<i>number</i>	numeric	<i>see below</i> MINimum MAXimum	none
Ramp, Sine, Square, and Triangle Outputs: MINimum selects 1 repetition; MAXimum selects 65534 repetitions. 9.9E+37 is equivalent to INFINITY. Arbitrary Waveform Output: MINimum selects 1 repetition. If SOURce:FUNction:USER A B C D HOP_AB is selected, MAXimum selects 65534 repetitions. If SOURce:FUNction:USER AB is selected, MAXimum selects 32767 repetitions. If SOURce:FUNction:USER ABCD is selected, MAXimum selects 16383 repetitions.			

Comments

- Use the ABORt command to terminate the output when ARM:START:LAYer1:COUNT is set to INFINITY.
- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- **Related Commands:** ABORt
- ***RST Condition:** ARM:START:LAYer1:COUNT INFINITY

Example Setting waveform repetitions per arm

```
ARM:COUN 10
```

Set 10 repetitions/arm

ARM[:START]:LAYer2 :COUNT

[[:START]:LAYer2 :COUNT

ARM[:START]:LAYer2:COUNT <number> specifies the number of waveform start arms the HP E1340A will accept after an INITiate:IMMEDIATE command before returning the trigger system to the idle state.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>number</i>	numeric	1 9.9e37 INFINity MINimum MAXimum	none
MINimum and MAXimum select 1 arm. 9.9E+37 is equivalent to INFINity.			

Comments

- If ARM:START:LAYer2:COUNT 1 is set, then ARM:START:LAYer2:SOURce must be set to IMMEDIATE. If ARM:START:LAYer2:COUNT INFINity is set, ARM:START:LAYer2:SOURce must be set to EXTERNAL.
- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- **Related Commands:** ABORt, INITiate:IMMEDIATE, ARM:START:LAYer2:SOURce
- ***RST Condition:** ARM:START:LAYer2:COUNT 1

Example Setting the start arm count

ARM:LAY2:COUN INF;SOUR EXT *Start arm count is infinity, and source is front panel "Aux In" BNC*

[[:START]:LAYer2 :SLOPe

ARM[:START]:LAYer2:SLOPe <edge> selects the edge (rising or falling) on the HP E1340A's front panel "Aux 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

Parameter Name	Parameter Type	Range of Values	Default Units
<i>edge</i>	discrete	NEGative POSitive	none

Comments

- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- **Related Commands:** ARM:START:LAYer2:SOURce
- ***RST Condition:** ARM:START:LAYer2:SLOPe POSitive

Example Setting the start arm slope

ARM:LAY2:SLOP NEG *Set negative start arm slope*

[:START]:LAYer2
:SOURce

ARM[:START]:LAYer2:SOURce <source> selects the source that will start waveform output. The available sources are:

EXTernal: The HP E1340A's front panel "Aux In" BNC connector.

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.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>source</i>	discrete	EXTernal IMMediate	none

Comments

- The "Aux In" BNC is a multiple-use input: for FSK control, for waveform hopping control, as a reference oscillator source, for reference oscillator gating, or as a start arm source. Only one of these uses may be active at any time.
- If ARM:START:LAYer2:SOURce IMMediate is set, then ARM:START:LAYer2:COUNT must be set to 1. If ARM:START:LAYer2:SOURce EXTernal is set, ARM:START:LAYer2:COUNT must be set to INFinity.
- The ARM:START:LAYer2:SLOPe command selects the active edge for the front panel "Aux In" BNC when used as the start arm source.
- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- **Related Commands:** ARM:START:LAYer2:COUNT, ARM:START:LAYer2:SLOPe
- ***RST Condition:** ARM:START:LAYer2:SOURce IMMediate

Example Setting the start arm source

```
ARM:LAY2:SOUR EXT;COUN INF
```

Start arm source is front panel "Aux In" BNC, and count is infinity

CALibration

The CALibration subsystem has commands that calibrates the HP E1340A. The subsystem also includes commands to prevent and detect accidental or unauthorized calibration of the HP E1340A.

Subsystem Syntax

```

CALibration
:AC
    :BEgin                [no query]
    :POINT? <value>      [query only]
:DATA
    :AC <block>
    [:DC] <block>
[:DC]
    :BEgin                [no query]
    :POINT? <value>      [query only]
:STATe
    :AC <state>

```

:AC:BEgin

CALibration:AC:BEgin starts the AC calibration procedure for the HP E1340A. It sets the HP E1340A up for the first of the measurements in the procedure. See the “Calibration” section of the *HP E1340A Service Manual* for detailed information on the use of this command.

Comments

- Most of the HP E1340A’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
- **Coupled command:** No
- **Related commands:** CALibration:AC:POINT
- ***RST Condition:** none

:AC:POINT **CALibration:AC:POINT** *<value>* takes the measured value for the current AC calibration point, computes needed calibration constants, and sets up the HP E1340A for the next measurement. Each calibration constant is checked for validity when it is computed. When all measurements and computations are completed, the constants are stored in the HP E1340A's non-volatile calibration memory.

The *RST command should be sent after completing the calibration procedure to restore normal operation.

See the "Calibration" section of the *HP E1340A Service Manual* for detailed information on the use of this command.

Comments

- Most of the HP E1340A'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
- **Coupled command:** No
- **Related commands:** CALibration:AC:BEIn
- ***RST Condition:** none

:DATA:AC **CALibration:DATA:AC** *<block>* transfers the AC portion of the HP E1340A'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. See the "Calibration" section of the *HP E1340A Service Manual* for detailed information on the use of this command.

Comments

- **Executable when initiated:** No
- **Coupled command:** No
- **Related commands:** none
- ***RST Condition:** unaffected

:DATA[:DC] **CALibration:DATA[:DC]** *<block>* transfers the DC portion of the HP E1340A'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. See the "Calibration" section of the *HP E1340A Service Manual* for detailed information on the use of this command.

Comments

- **Executable when initiated:** No
- **Coupled command:** No
- **Related commands:** none
- ***RST Condition:** unaffected

CALibration[:DC]:BEGin

[:DC]:BEGin

CALibration[:DC]:BEGin starts the DC calibration procedure for the HP E1340A. It sets the HP E1340A up for the first of the measurements in the procedure. See the “Calibration” section of the *HP E1340A Service Manual* for detailed information on the use of this command.

Comments

- Most of the HP E1340A’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
- **Coupled command:** No
- **Related commands:** CALibration:DC:POINT
- ***RST Condition:** none

[: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 E1340A for the next measurement. Each calibration constant is checked for validity when it is computed. When all measurements and computations are completed, the constants are stored in the HP E1340A’s non-volatile calibration memory.

The *RST command should be sent after completing the calibration procedure to restore normal operation.

See the “Calibration” section of the *HP E1340A Service Manual* for detailed information on the use of this command.

Comments

- Most of the HP E1340A’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
- **Coupled command:** No
- **Related commands:** CALibration:DC:BEGin
- ***RST Condition:** none

:STATe:AC CALibration:STATe:AC <state> specifies whether AC amplitude correction using the calibration constants is enabled or disabled. If STATe is OFF, AC correction is disabled. If STATe is ON, AC correction is enabled.

AC amplitude correction is only possible when SOURce:ARBitrary:DAC:SOURce is set to INTernal and the current reference oscillator frequency is 42.94967296 MHz, either internal or external. CALibration:STATe:AC must be set to OFF when SOURce:ARBitrary:DAC:SOURce VXI is set or when a different external reference oscillator frequency is selected.

CAUTION The HP E1340A uses an active amplitude correction technique, boosting the output signal to compensate for filter and other roll-offs. When waveform generation at high frequencies is halted either because ARM:LAYer1:COUNT cycles have been generated, or because an ABORt command was executed, filter loss disappears, and the output level may exceed the level specified by SOURce:VOLTagE:LEVel:IMMediate:AMPLitude. It is recommended that AC amplitude correction be used only for continuous output.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>state</i>	boolean	OFF 0 ON 1	none

- Comments**
- **Executable when initiated:** Query form only
 - **Coupled command:** Yes
 - **Related commands:** SOURce:ROSCillator:FREQuency:EXTernal
 - ***RST Condition:** CALibration:STATe:AC ON

Example Disabling AC amplitude correction

CAL:STAT:AC OFF

Disable amplitude correction

INITiate

The INITiate subsystem initiates the trigger subsystem and prepares the HP E1340A for waveform generation. Once initiated, a start arm received from the programmed arm source (ARM:STARt:SOURce command) starts the waveform output.

For frequency sweeping, the initial waveform frequency is the STARt frequency when SOURce:SWEep:DIRection UP is set, or the STOP frequency when SOURce:SWEep:DIRection DOWN is set.

Subsystem Syntax

```
INITiate
[:IMMediate]           [no query]
```

[:IMMediate]

INITiate[:IMMediate] initiates the trigger system. Waveform generation begins immediately if ARM:STARt:SOURce IMMEDIATE is set; otherwise, waveform generation begins when the first start arm is received. Only the ABORt and *RST commands or a Device Clear will halt waveform generation.

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.

Comments

- Use the ABORt command to prematurely halt the waveform generation and place the trigger system in the idle state.
- Executing this command when SOURce:FUNCTion:SHAPE DC is set, when SOURce:ARBITrary:DAC:SOURce is not set to INTernal, or with the trigger system not in the idle state generates error -123, "Init ignored".
- **Executable when initiated:** No
- **Coupled command:** No
- **Related Commands:** *OPC, *OPC?, *RST, *WAI, ABORt, ARM subsystem, TRIGger subsystem
- ***RST Condition:** The trigger system is in the idle state.

Example Initiating waveform generation

```
INIT Initiate waveform generation
```

OUTPut

The OUTPut subsystem controls the characteristics of the output waveform. The subsystem sets the output source impedance, and enables or disables the output.

Subsystem Syntax OUTPut
 :IMPedance <*impedance*>
 :LOAD <*load*>

:IMPedance **OUTPut:IMPedance <*impedance*>** sets the HP E1340A's output impedance.

Since the HP E1340A only provides 50Ω output impedance, there is no need to send this command. It is provided for SCPI compatibility only.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>impedance</i>	numeric	50 MINimum MAXimum	Ohms
MINimum and MAXimum select 50Ω output impedance.			

- Comments**
- Impedance values between 25 and 62.5 will be accepted and rounded to 50 Ω. Values outside that range will cause error -222,"Argument out of range" to occur.
 - **Executable when initiated:** Yes
 - **Coupled command:** Yes
 - **Related commands:** OUTPut:LOAD
 - ***RST Condition:** OUTPut:IMPedance 50

Example **Querying output impedance**

OUTP:IMP?

50 will be returned with this query

:LOAD **OUTPut:LOAD <load>** indicates whether the actual load applied to the HP E1340A's "Aux Out" BNC is either matched to the output impedance specified by **OUTPut:IMPedance** or is an open circuit. The output voltage into an open circuit is twice that into a matched load. Setting **OUTPut:LOAD INFinity** compensates for this effect so that the **SOURce:LIST:SEGment:VOLTag**e and **SOURce:VOLTag:LEVel:IMMEDIATE:AMPLitude** and **OFFSet** commands will output the specified voltages into an open circuit.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>load</i>	numeric	50 9.9E+37 INFinity MINimum MAXimum	Ohms
MINimum and MAXimum both select 50Ω load impedance. Use 9.9E+37 or INFinity to indicate an open circuit output.			

- Comments**
- **Executable when initiated:** Query form only
 - **Coupled command:** Yes
 - **Related commands:** **OUTPut:IMPedance**, **SOURce:LIST subsystem**, **SOURce:VOLTag**e subsystem
 - ***RST Condition:** **OUTPut:LOAD 50**

Example **Indicating open circuit output load**

OUTP:LOAD INF *Indicate open circuit*

[SOURce:]ARBitrary

The [SOURce:]ARBitrary subsystem controls:

- The DAC data source.
- Direct downloading of DAC data to the waveform segment memory.

Subsystem Syntax

```
[SOURce:]
ARBitrary
  :DAC
    :SOURce <source>
    :DOWNload <source>,<dest>,<length> [no query]
    :COMPLete [no query]
```

:DAC:SOURce [SOURce:]ARBitrary:DAC:SOURce <source> selects the DAC's data source. The available sources are:

INTernal: The waveforms specified by the SOURce:FUNC or SOURce:FUNC:USER commands.

VXI: The VXIbus data transfer bus.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>source</i>	discrete	INTernal VXI	none

Comments

- When driving the DAC from the VXIbus data transfer bus, the address for writing the data is offset 12 decimal (0C hex) in the HP E1340A's A16 address space.
- Setting the DAC data source to a setting other than INTernal disables the ARM subsystem, the SOURce subsystem except for the SOURce:ARBitrary and SOURce:VOLTage subsystems, and the TRIGger subsystem. The HP E1340A immediately outputs each DAC data point when received. The DAC data must be provided as integer numbers between 0 and 4095; 0 generates negative full scale output, 4095 generates positive full scale. The SOURce:VOLTage:LEVel:IMMediate:AMPLitude specifies the positive full-scale output voltage, and must be specified in terms of volts or volts peak (V or VPK).
- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- **Related commands:** SOURce:VOLTage:LEVel:IMMediate:AMPLitude
- ***RST Condition:** SOURce:ARBitrary:DAC:SOURce INTernal

Example Setting the DAC data source

```
ARB:DAC:SOUR VXI
```

Select VXI A16 address space as data source

:DOWNload [SOURce:]ARbitrary:DOWNload <source>,<dest>,<length> enables the direct download mode to the waveform segment memory. It selects the download source, waveform segment name, and number of points. The available download sources are:

EEPRom<n>: EEPRom1 through EEPRom16, where <n> selects which block of the built-in waveform EPROM to use. These blocks contain 4096 points of the following waveforms:

EEPRom1: Sine wave

EEPRom2: Triangular wave

EEPRom3: $\sin(x)/x$ for 8.25 radians

EEPRom4: Haversine

EEPRom5: Square wave

EEPRom6: First ten terms of a square wave

EEPRom7: First four terms of a square wave

EEPRom8: Falling ramp

EEPRom9: First 20 terms of a falling ramp

EEPRom10: Rising ramp

EEPRom11: First 20 terms of a rising ramp

EEPRom12: White noise

EEPRom13: Modulated white noise

EEPRom14: A 3rd, 4th, 5th harmonic chord

EEPRom15: 8 cycles of a linear rising sine wave

EEPRom16: Positive half-cycle sine wave

VXI: The VXIbus data transfer bus.

Waveform Data The waveform data consists of a single 16-bit word for each voltage point (only 12 bits are used), with zero representing negative full scale and 4095 representing positive full scale. The most significant four bits of each 16-bit word are not used and must be zero.

With SOURce:VOLTage:LEVel:IMMediate AMPLitude 5.1175 V set and a matched output load, the least significant bit (LSB) represents 2.5 mV.

When downloading waveform segment data from the VXIbus data transfer bus, the address for writing the data is offset 12 decimal (0C hex) in the HP E1340A's A16 address space.

Parameter Name	Parameter Type	Range of Values	Default Units
<i>source</i>	discrete	EEPROM1 through EEPROM16 VXI	none
<i>dest</i>	char	A B C D	none
<i>length</i>	numeric	4096 MINimum MAXimum	none

Comments

- When downloading from the VXIbus and downloading is complete, use the SOURCE:ARBITRARY:DOWNLOAD:COMPLETE command to restore normal operation. This command should not be sent when downloading from the built-in waveform EPROM blocks.
- No error checking is performed on downloaded data. Similarly, the number of points sent is not checked against the length specified. Erratic operation may occur if invalid data is downloaded.
- Waveform segment contents are unspecified at power-on and must be loaded before the segment may be output.
- **Executable when initiated:** No
- **Coupled command:** No
- **Related Commands:** SOURCE:ARBITRARY:DOWNLOAD:COMPLETE, SOURCE:VOLTAGE:LEVEL:IMMEDIATE:AMPLITUDE
- ***RST Condition:** downloading disabled

Example Download 4096 points from the VXI backplane to waveform segment "C"

```
ARB:DOWN VXI,C,4096           Set up for download
download data
ARB:DOWN:COMP                 Indicate download complete
```

Example Download 4096 points from EEPROM block 3 (Sine(x)/x) to waveform segment "C"

```
ARB:DOWN EEPR3,C,4096       Set up for download
```

[SOURce:]ARbitrary:DOWNload :COMPLete

:DOWNload :COMPLete [SOURce:]ARbitrary:DOWNload:COMPLete disables direct downloading from VXIbus mode. Send it when VXIbus downloading is complete.

Comments

- **Executable when initiated:** No
- **Coupled command:** No
- **Related Commands:** SOURce:ARbitrary:DOWNload
- ***RST Condition:** downloading disabled

Example Download 4096 points from the VXI backplane to waveform segment "C"

ARB:DOWN VXI,C,4096	<i>Set up for download</i>
download data	
ARB:DOWN:COMP	<i>Indicate download complete</i>

[SOURCE:]FREQUENCY

The [SOURCE:]FREQUENCY subsystem controls the HP E1340A's frequency generator.

The generator uses a direct digital synthesis (NCO) technique to generate the specified frequencies. It has an upper frequency limit of the reference oscillator frequency multiplied by .46874999767 and has a resolution of .01 Hz when used with the built-in 42.94 MHz reference oscillator.

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:

$$\begin{aligned} \text{CENTER} &= (\text{START} + \text{STOP})/2 \\ \text{SPAN} &= \text{STOP} - \text{START} \end{aligned}$$

- 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:

$$\begin{aligned} \text{START} &= \text{CENTER} - (\text{SPAN}/2) \\ \text{STOP} &= \text{CENTER} + (\text{SPAN}/2) \end{aligned}$$

- 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, FREQUENCY:CENTER MINIMUM would set 500 kHz. An exception to this rule exists, however: since the minimum SPAN is .01 Hz per sweep point (assuming the internal reference oscillator), a command string such as FREQUENCY:CENTER 0;SPAN MINIMUM has no valid value for SPAN, and will attempt to set the START frequency to a negative value. Similarly, FREQUENCY:START 15e6;STOP MINIMUM will attempt to set STOP to value greater than 15 MHz.

The minimum possible frequency is 0 Hz. The maximum possible frequency depends on the frequency of the currently selected reference oscillator source (SOURce:ROSCillator:SOURce) and the waveform shape (SOURce:FUNcTION:SHAPE), according to the following rules:

- **Sine Wave Output:** the maximum possible frequency is the current reference oscillator frequency multiplied by .34924596548 (15 MHz when the 42.94967296 MHz internal reference oscillator is used).
- **Square Wave, Ramp, and Triangle Outputs:** the maximum possible frequency is the current reference oscillator frequency multiplied by 0.0232830643654 (1 MHz when the 42.94967296 MHz internal reference oscillator is used).
- **SOURce:FUNcTION:SHAPE:USER:** the maximum possible frequency depends upon how many segments are being output. For SOURce:FUNcTION:USER A | B | C | D | HOP_AB, only one segment is output, and the maximum possible frequency is the reference oscillator frequency multiplied by .34924596548 (15 MHz when the 42.94967296 MHz internal reference oscillator is used). For SOURce:FUNcTION:USER AB, two segments are output, and the maximum possible frequency is half the above value. For SOURce:FUNcTION:USER ABCD, four segments are output, and the maximum possible frequency is one-fourth the above value.
- Waveforms in the HP E1340A are generated by indexing through 4,096 points of memory data (8,192 for SOURce:FUNcTION:USER AB or 16,384 for SOURce:FUNcTION:USER ABCD) within the specified waveform period. At the frequency of (reference oscillator frequency / memory points) each point is output for exactly one reference oscillator period. At lower frequencies, some or all points are output for multiple clock periods, with all points being output for the same number of clock periods at frequencies that are a power of two (2, 4, 8, etc.) less than the above. At higher frequencies, some points are skipped on each repetition.

Subsystem Syntax

```
[SOURce:]
FREQUency
:CENTer <center_freq>
[:CW]:FIXed] <frequency>
:FSKey <frequency1>,<frequency2>
:MODE <mode>
:SPAN <freq_span>
:STARt <start_freq>
:STOP <stop_freq>
```

:CENTer [SOURce:]FREQUency:CENTer <*center_freq*> sets the center frequency for a frequency-swept waveform.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>center_freq</i>	numeric	<i>see below</i> MINimum MAXimum	Hz
The legal range for <i>center_freq</i> , as well as the MINimum and MAXimum values, are context-dependent. See the section "Coupling Rules" at the beginning of the SOURce:FREQUency subsystem for a description of the coupling between START, STOP, CENTer, and SPAN.			

Comments

- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- **Related Commands:** SOURce:FREQUency:MODE, SPAN, START, and STOP, SOURce:FUNCTion:SHAPE, SOURce:ROSCillator
- ***RST Condition:** SOURce:FREQUency:CENTer 7.5 MHz

Example Setting the center frequency

```
FREQ:CENT 1E3
```

Set center frequency to 1000 Hz

[:CW]:FIXed [SOURce:]FREQUency[:CW]:FIXed <*frequency*> selects the non-swept waveform frequency.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>frequency</i>	numeric	<i>see below</i> MINimum MAXimum	Hz
MINimum selects 0 Hz.			
MAXimum selects the current reference oscillator frequency multiplied by 0.468749999767.			
The above values bound the legal range for <i>frequency</i> .			

Comments

- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- **Related Commands:** SOURce:FREQUency:MODE, SOURce:FUNCTion:SHAPE, SOURce:ROSCillator
- ***RST Condition:** SOURce:FREQUency:FIXed 10 kHz

Example Setting the waveform frequency

```
FREQ 1E3
```

Set frequency to 1000 Hz

:FSKey [SOURce:]FREQUency:FSKey <*frequency1*>,<*frequency2*> sets the two waveform frequencies for frequency-shift keying. A TTL-compatible high level on the "Aux In" BNC generates *frequency1*; a TTL-compatible low level generates *frequency2*.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>frequency1</i> <i>frequency2</i>	numeric	see below MINimum MAXimum	Hz
MINimum selects 0 Hz.			
MAXimum selects the current reference oscillator frequency multiplied by 0.468749999767.			
The above values bound the legal range for <i>frequency1</i> and <i>frequency2</i> .			

Comments

- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- **Related Commands:** SOURce:FREQUency:MODE, SOURce:FUNCTion:SHAPE and USER, SOURce:ROSCillator
- ***RST Condition:** SOURce:FREQUency:FSKey 10 kHz, 15 MHz

Example

Setting the frequency-shift frequencies

```
FREQ:FSK 1E6,1 KHZ
```

Set 1 MHz and 1 kHz frequencies

:MODE

[SOURce:]FREQUency:MODE <*mode*> determines which set of commands control the frequency subsystem. The settings have the following meanings:

CW or FIXed: Selects single-frequency mode.

SOURce:FREQUency:CW|FIXed selects the waveform frequency. CW and FIXed are equivalent.

FSKey: Selects frequency shift keying mode.

SOURce:FREQUency:FSKey activates the mode, and TTL levels on the HP E1340A's front panel "Aux In" BNC are used to toggle between the two waveform frequencies.

SWEEp: Selects frequency sweep mode.

SOURce:FREQUency:CENTER, SPAN, START and STOP commands set the frequency range. The SOURce:SWEEp subsystem controls the sweep.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>mode</i>	discrete	CW FIXed FSKey SWEep	none

Comments

- The "Aux In" BNC is a multiple-use input: for FSK control, for waveform hopping control, as a reference oscillator source, for reference oscillator gating, or as a start arm source. Only one of these uses may be active at any time.
- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- **Related Commands:** SOURce:FREQUency subsystem, SOURce:SWEep subsystem
- ***RST Condition:** SOURce:FREQUency:MODE FIXed

Example Setting frequency sweep mode

```
FREQ:MODE SWEep
```

Set frequency sweep mode

:SPAN [SOURce:]FREQUency:SPAN *<freq_span>* sets the frequency span for a frequency-swept waveform.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>freq_span</i>	numeric	<i>see below</i> MINimum MAXimum	Hz
The legal range for <i>freq_span</i> , as well as the MINimum and MAXimum values, are context-dependent. See the section "Coupling Rules" at the beginning of the SOURce:FREQUency subsystem for a description of the coupling between START, STOP, CENTER, and SPAN.			

Comments

- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- **Related Commands:** SOURce:FREQUency:CENTer, MODE, START, and STOP, SOURce:FUNCTion:SHAPE, SOURce:ROSCillator
- ***RST Condition:** SOURce:FREQUency:SPAN 15.0 MHz

Example Setting the frequency span

```
FREQ:SPAN 1E3
```

Set frequency span to 1000 Hz

[SOURce:]FREQUency:STARt

:STARt [SOURce:]FREQUency:STARt <*start_freq*> sets the starting frequency for a frequency-swept waveform.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>start_freq</i>	numeric	<i>see below</i> MINimum MAXimum	Hz

The legal range for *start_freq*, as well as the MINimum and MAXimum values, are context-dependent. See the section "Coupling Rules" at the beginning of the SOURce:FREQUency subsystem for a description of the coupling between STARt, STOP, CENTer, and SPAN.

- Comments**
- **Executable when initiated:** Query form only
 - **Coupled command:** Yes
 - **Related Commands:** SOURce:FREQUency:CENTer, MODE, SPAN, and STOP, SOURce:FUNCTion:SHAPE, SOURce:ROSCillator
 - ***RST Condition:** SOURce:FREQUency:STARt 0.0 Hz

Example Setting the starting frequency

FREQ:STAR 1 KHZ *Set starting frequency to 1000 Hz*

:STOP [SOURce:]FREQUency:STOP <*stop_freq*> sets the stopping frequency for a frequency-swept waveform.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>stop_freq</i>	numeric	<i>see below</i> MINimum MAXimum	Hz

The legal range for *stop_freq*, as well as the MINimum and MAXimum values, are context-dependent. See the section "Coupling Rules" at the beginning of the SOURce:FREQUency subsystem for a description of the coupling between STARt, STOP, CENTer, and SPAN.

- Comments**
- **Executable when initiated:** Query form only
 - **Coupled command:** Yes
 - **Related Commands:** SOURce:FREQUency:CENTer, MODE, SPAN, and STARt, SOURce:FUNCTion:SHAPE, SOURce:ROSCillator
 - ***RST Condition:** SOURce:FREQUency:STOP 15.0 MHz

Example Setting the stopping frequency

FREQ:STOP 1E3 *Set stopping frequency to 1000 Hz*

[SOURce:]FUNCTION

The [SOURce:]FUNCTION subsystem controls which waveform shape (arbitrary, sinusoid, etc.) the HP E1340A generates. For arbitrary waveform generation, the subsystem controls which of the seven possible segment sequences are selected.

Subsystem Syntax

```
[SOURce:]
  FUNCTION
    [:SHAPE] <shape>
    :USER <name>
```

[:SHAPE] [SOURce:]FUNCTION[:SHAPE] <shape> selects what waveform shape the HP E1340A generates, shown as follows:

DC: Generates a DC output voltage.

RAMP: Generates a stepped ramp. The SOURce:RAMP subsystem controls the polarity.

SINusoid: Generates a sinusoidal voltage.

SQUare: Generates a square wave.

TRiangle: Generates a stepped triangle wave.

USER: Generates an arbitrary waveform. The SOURce:FUNCTION:USER command selects the segment sequence to be generated.

Note Selecting a shape other than DC or USER causes the voltage data contained in waveform segment "D" to be overwritten by the voltage data for the selected waveform shape. Segment "D" must be reloaded before it can be output as part of a segment sequence.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>shape</i>	discrete	DC RAMP SINusoid SQUare TRIangle USER	none

Comments

- **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.
 - The SOURce:FREQUency subsystem specifies the signal frequency for all waveform shapes.
- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- ***RST Condition:** SOURce:FUNction:SHAPE SINusoid

Example **Selecting square wave generation mode**

FUNC SQU *Select square wave mode*

:USER [SOURce:]FUNction:USER <name> selects which of the seven pre-defined segment sequences the HP E1340A will output when arbitrary waveform generation is selected by SOURce:FUNction:SHAPE USER.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>name</i>	character data	A B C D AB ABCD HOP_AB NONE	none
NONE selects no segment sequence			

Comments

- Segment sequences A, B, C, and D output the corresponding segments. Sequence AB outputs segments A and B. Sequence ABCD outputs all four segments. Sequence HOP_AB allows waveform hopping between segments A and B. The "Aux In" BNC selects which segment is output: a TTL high level selects segment A; a TTL low, segment B.
- The "Aux In" BNC is a multiple-use input: for FSK control, for waveform hopping control, as a reference oscillator source, for reference oscillator gating, or as a start arm source. Only one of these uses may be active at any time.
- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- **Related Commands:** SOURce:FUNction:SHAPE
- ***RST Condition:** SOURce:FUNction:USER NONE

Example **Selecting an arbitrary waveform**

FUNC USER *Select arbitrary waveform mode*
 FUNC:USER AB *Select segment sequence AB*

[SOURce:]LIST

The [SOURce:]LIST subsystem provides edit access to the waveform segments, and creation of segment sequences for arbitrary waveform generation.

There are four predefined segment names: A, B, C, and D. Each of these is 4096 points in length. There are six predefined segment sequences: A, B, C, D, AB, and ABCD. The sequences are predefined to output segment A, B, C, D, A and B, and all four segments, respectively.

Values may be loaded into segment memory by using the LIST:VOLT:DAC or LIST:VOLT commands in conjunction with LIST:SElect. The remaining commands are queries and are provided primarily for compatibility and completeness reasons.

Subsystem syntax

```
[SOURce:]
LIST
  [:SEGment]
    :CATalog?           [query only]
    :DEFine?           [query only]
    :SElect <name>
    :VOLTage <voltage_list> [no query]
      :DAC <voltage_list> [no query]
      :POINTs?         [query only]

    :SSEquence
      :CATalog?       [query only]
      :DEFine?       [query only]
      :SElect <name>
      :SEquence?     [query only]
      :SEGmentS?    [query only]
```

[[:SEGment] :CATalog?] [SOURce:]LIST[:SEGment]:CATalog? returns a comma-separated list of four quoted strings containing the predefined waveform segment names: "A", "B", "C", "D".

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- ***RST Condition:** none
- **Power-On Condition:** Segment names A, B, C, and D are defined

Example

Cataloging waveform segment names

```
LIST:CAT?
```

Catalog waveform segments

[SOURce:]LIST[:SEGMENT]:DEFine?

[[:SEGMENT]:DEFine? [SOURce:]LIST[:SEGMENT]:DEFine? returns the amount of the waveform segment memory allocated for the segment currently selected by SOURce:LIST:SEGMENT:SElect. Since all waveform segments are 4096 points in length, this query will always return 4096.

- Comments**
- **Executable when initiated:** Yes
 - **Coupled command:** No
 - **Related Commands:** SOURce:LIST:SEGMENT:SElect
 - ***RST Condition:** unaffected
 - **Power-On Condition:** Each waveform segment is defined to a length of 4096

Example Querying the size of a waveform segment

LIST:SEL D *Select waveform segment D*
LIST:DEF? *Query size of waveform segment D*

[[:SEGMENT]:SElect [SOURce:]LIST[:SEGMENT]:SElect <name> selects a waveform segment for subsequent SOURce:LIST:SEGMENT subsystem commands.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>name</i>	discrete	A B C D NONE	none
NONE selects no waveform segment			

- Comments**
- There are only 4 legal segment names: A, B, C, D. Alphabetic character case (upper vs. lower) is ignored.
 - **Executable when initiated:** Yes
 - **Coupled command:** No
 - ***RST Condition:** unaffected
 - **Power-On Condition:** SOURce:SEGMENT:SElect NONE

Example Selecting a waveform segment

LIST:SEL A *Select waveform segment A*

[:SEGMENT]:VOLTage

[SOURce:]LIST[: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 voltage value must be sent first.
- The voltage point list length must contain exactly 4096 points.
- 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.
- Waveform segment contents are unspecified at power-on and must be loaded before the segment may be output.
- **Executable when initiated:** No
- **Coupled command:** No
- **Related Commands:** SOURce:LIST:SEGMENT:VOLTage:DAC, SOURce:VOLTage:LEVel:IMMediate:AMPLitude
- ***RST Condition:** unaffected
- **Power-On Condition:** waveform segment contents are unspecified

Example**Defining a waveform segment voltage point list**

LIST:SEL B	<i>Select waveform segment B</i>
LIST:VOLT .5,1,.5,0,-.5,.....,-1	<i>Define 4096 waveform voltages</i>

[SOURce:]LIST[:SEGMENT]:VOLTage:DAC

**[[:SEGMENT]:VOLTage
:DAC**

[SOURce:]LIST[:SEGMENT]: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 length block containing the DAC codes in 16-bit integer format. The upper four bits of the 16-bit integer must be 0.

The DAC code is a 12-bit number, with 0 representing negative full scale, 4095 representing positive full scale, and 2048 representing zero. With SOURce:VOLTage:LEVel:IMMediate:AMPLitude 5.11750 V, and OUTPut:LOAD 50, the least significant bit (LSB) represents 2.50 mV.

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.
- Waveform segment contents are unspecified at power-on and must be loaded before the segment may be output.
- **Executable when initiated:** No
- **Coupled command:** No
- **Related Commands:** SOURce:LIST:SEGMENT:VOLTage, SOURce:VOLTage:LEVel:IMMediate:AMPLitude
- ***RST Condition:** unaffected
- **Power-On Condition:** waveform segment contents are unspecified

Example

Defining a waveform segment voltage point list

```
LIST:SEL B Select waveform segment B  
LIST:VOLT:DAC 4,1048,4095,.....,2048 Define 4096 waveform voltages
```

**[[:SEGMENT]:VOLTage
:POINTS?**

[SOURce:]LIST[:SEGMENT]:VOLTage:POINTS? returns a number indicating the length of the currently selected waveform segment's voltage point list. The length is considered 0 at power-on. For segment D, it is set to 0 when a waveform shape other than DC or USER is selected. The length is set to 4096 after the segment contents have been loaded via the SOURce:ARbitrary:DOWNload, SOURce:LIST:SEGMENT:VOLTage, or SOURce:LIST:SEGMENT:VOLTage:DAC commands.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- ***RST Condition:** unaffected
- **Power-On Condition:** waveform segment lengths are 0

Example

Query voltage point list length

```
LIST:SEL C Select waveform segment C  
LIST:VOLT:POIN? Query voltage point list length
```

**:SSEquence
:CATalog?**

[SOURCE:]LIST:SSEquence:CATalog? returns a comma-separated list of six quoted strings containing the predefined segment sequence names: "A", "B", "C", "D", "AB", "ABCD".

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- ***RST Condition:** unaffected
- **Power-On Condition:** Sequence names A, B, C, D, AB, and ABCD are defined

Example

Cataloging segment sequence names

LIST:SSEQ:CAT?

Catalog segment sequence names

**:SSEquence
:DEFine?**

[SOURCE:]LIST:SSEquence:DEFine? returns the length of the segment sequence currently selected by SOURCE:LIST:SSEquence:SElect. Sequences A, B, C, and D have a length of 1; sequence AB has a length of 2; and sequence ABCD has a length of 4.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related Commands:** SOURCE:LIST:SSEquence:SElect
- ***RST Condition:** unaffected
- **Power-On Condition:** segment sequence lengths are predefined as specified above

Example

Querying the length of a segment sequence

LIST:SSEQ:SEL AB

Select sequence AB

LIST:SSEQ:DEF?

Query the length of the segment sequence

:SSEquence:SElect

[SOURCE:]LIST:SSEquence:SElect <name> selects a segment sequence for subsequent SOURCE:LIST:SSEquence subsystem commands.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>name</i>	discrete	A B C D AB ABCD NONE	none
NONE selects no segment sequence			

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- ***RST Condition:** unaffected
- **Power-On Condition:** SOURCE:LIST:SSEquence:SElect NONE

[SOURCE:]LIST:SSEquence :SEquence?

Example Selecting a segment sequence

LIST:SSEQ:SEL AB *Select segment sequence AB*

**:SSEquence
:SEquence?** [SOURCE:]LIST:SSEquence:SEquence? returns a comma-separated list of the segment names comprising the currently selected segment sequence. Sequences A, B, C, and D contain only the same-named waveform segment. Sequence AB contains segments A and B. Sequence ABCD contains segments A, B, C, and D.

- Comments**
- **Executable when initiated:** Yes
 - **Coupled command:** No
 - ***RST Condition:** unaffected
 - **Power-On Condition:** segment sequences are pre-defined as specified above

Example Querying a segment sequence

LIST:SSEQ:SEL AB *Select segment sequence AB*
LIST:SSEQ:SEQ? *Query segment sequence*

**:SSEquence
:SEquence
:SEGMents?** [SOURCE:]LIST:SSEquence:SEquence:SEGMents? returns a number indicating the number of waveform segments contained in the currently selected segment sequence. Sequences A, B, C, and D contain only 1 segment. Sequence AB contains 2 segments. Sequence ABCD contains 4.

- Comments**
- **Executable when initiated:** Yes
 - **Coupled command:** No
 - ***RST Condition:** none
 - **Power-On Condition:** segment sequence lengths are predefined as specified above

Example Query segment sequence length

LIST:SSEQ:SEQ:SEGM? *Query segment sequence length*

[SOURce:]MARKer

The [SOURce:]MARKer subsystem controls:

- Which signal is routed to the "Aux Out" BNC.
- The polarity of the "Aux Out" BNC signal.

Subsystem Syntax

```
[SOURce:]
MARKer
:FEED <source>
:POLarity <polarity>
```

:FEED [SOURce:]MARKer:FEED <source> selects the source for the front panel "Aux Out" BNC. The available sources are:

"OUTPut:ZERO": Outputs marker pulses whenever the waveform crosses through zero. SOURce:MARKer:POLarity must be set to INVerted when using this source.

"SEGment": Outputs a pulse at the completion of a segment. Note that multiple segment waveforms like AB or ABCD will have 2 and 4 marker pulses per cycle respectively. SOURce:MARKer:POLarity must be set to NORMAl when using this source.

"[SOURce:]ROSCillator": The internal 42.94967296 MHz reference oscillator. SOURce:MARKer:POLarity must be set to NORMAl when using this source.

"[SOURce:]SWEeP": Outputs a marker pulse on the last point of a frequency sweep. SOURce:MARKer:POLarity may be either NORMAl or INVerted with this source.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>source</i>	string	"OUTPut:ZERO" "SEGment" "[SOURce:]ROSCillator" "[SOURce:]SWEep"	none

Comments

- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- **Related Commands:** SOURce:MARKer:POLarity
- ***RST Condition:** SOURce:MARKer:FEED "SEGment"

Example Setting the "Aux Out" BNC source

```
MARK:FEED "SOUR:ROSC"
```

Set marker reference oscillator as source

[SOURce:]MARKer:POLarity

:POLarity [SOURce:]MARKer:POLarity <*polarity*> selects the polarity of the marker signal at the front panel "Aux Out" BNC. NORMal polarity selects an active high marker output; INVerted an active low output.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>polarity</i>	discrete	INVerted NORMal	none

Comments

- NORMal must be selected when MARKer:FEED "SEGment" or "SOURce:ROSCillator" is selected. INVerted must be selected when MARKer:FEED "OUTPut:ZERO" is selected. Either NORMal and INVerted may be selected when MARKer:FEED "[SOURce:]SWEep" is selected.
- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- **Related Commands:** SOURce:MARKer:FEED
- ***RST Condition:** SOURce:MARKer:POLarity NORMal

Example Setting the "Aux Out" BNC polarity

MARK:FEED "SWE"

Generate marker pulse at start of sweep

MARK:POL INV

Set active low output

[SOURce:]RAMP

The [SOURce:]RAMP subsystem selects the polarity of ramp waveforms.

Subsystem Syntax [SOURce:]
 RAMP
 :POLarity <*polarity*>

:POLarity [SOURce:]RAMP:POLarity <*polarity*> selects the polarity of the ramp waveform. NORMal generates a positive-going ramp; INVerted generates a negative-going ramp.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>polarity</i>	discrete	INVerted NORMal	none

- Comments**
- **Executable when initiated:** Query form only
 - **Coupled command:** Yes
 - **Related Commands:** SOURce:FUNCTion:SHAPe
 - ***RST Condition:** SOURce:RAMP:POLarity NORMal

Example Selecting ramp polarity

FUNC:SHAP RAMP
RAMP:POL INV

Select ramp output
Select negative-going ramp

[SOURce:]ROSCillator

The [SOURce:]ROSCillator subsystem controls the reference oscillator's source and indicates the frequency of an external oscillator. The HP E1340A uses the source and frequency information to generate the waveform frequency for all waveform shapes.

Subsystem Syntax

```
[SOURce:]
ROSCillator
:FREQUENCY
:EXTernal <frequency>
:GATE
:STATe <state>
:SOURce <source>
```

:FREQUENCY [SOURce:]ROSCillator:FREQUENCY:EXTernal <frequency> indicates to the HP E1340A the frequency of an external reference oscillator source. The SOURce:FREQUENCY subsystem uses this value to generate the waveform frequency when SOURce:ROSCillator:SOURce is set to EXTernal.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>frequency</i>	numeric	1 Hz through 42.94967296 MHz MINimum MAXimum	Hz
MINimum selects 1 Hz; MAXimum selects 42.94967296 MHz.			

Comments

- AC amplitude correction can only be made if the current reference oscillator frequency is 42.94967296 MHz, either internal or external. CALibration:STATe:AC must be set to OFF when a different external reference oscillator frequency is selected.
- Indicating an incorrect frequency for an external reference oscillator will cause incorrect sample rate and waveform frequencies to be generated by the SOURce:FREQUENCY subsystem.
- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- **Related commands:** CALibration:STATe:AC, SOURce:ROSCillator:SOURce
- ***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

:GATE:STATE [SOURce:]ROSCillator:GATE:STATE <state> indicates whether gating of the reference oscillator is enabled or disabled. When gating is enabled, a TTL high signal on the "Aux In" BNC will stop the waveform output at the present point; a TTL low on the "Aux In" BNC will resume/continue normal operation.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>state</i>	boolean	ON OFF 0 1	none

- Comments**
- The "Aux In" BNC is a multiple-use input: for FSK control, for waveform hopping control, as a reference oscillator source, for reference oscillator gating, or as a start arm source. Only one of these uses may be active at any time.
 - **Executable when initiated:** Query form only
 - **Coupled command:** No
 - ***RST Condition:** SOURce:ROSCillator:GATE:STATE OFF

Example Enable gating of the waveform via the "Aux In" BNC

ROSC:GATE:STATE ON *Enable gating*

:SOURce [SOURce:]ROSCillator:SOURce <source> selects the reference oscillator source. The available sources are:

EXTernal: The HP E1340A's front panel "Aux In" BNC.

INTernal: The internal 42.94967296 MHz oscillator. Using this oscillator with the SOURce:FREQUENCY subsystem gives a frequency resolution of .01 Hz.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>source</i>	discrete	EXTernal INTernal	none

- Comments**
- The reference oscillator is used to generate the waveform frequencies specified in the SOURce:FREQUENCY subsystem.
 - Use SOURce:ROSCillator:FREQUENCY:EXTernal to indicate the frequency of an external reference oscillator.
 - The "Aux In" BNC is a multiple-use input: for FSK control, for waveform hopping control, as a reference oscillator source, for reference oscillator gating, or as a start arm source. Only one of these uses may be active at any time.
 - **Related Commands:** SOURce:ROSCillator:FREQUENCY:EXTernal, SOURce:FREQUENCY

[SOURce:]ROSCillator:SOURce

- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- ***RST Condition:** SOURce:ROSCillator:SOURce INTernal

Example **Setting the reference oscillator source**

ROSC:SOUR EXTernal

Select an external oscillator source

[SOURCE:]SWEep

The [SOURCE:]SWEep subsystem selects:

- The number of points in a frequency sweep.
- The sweep rate for frequency sweeps.

Frequency sweeping generation requires that SOURCE:FREQUENCY:MODE SWEep be set.

Sweep Considerations

Frequency sweeping is controlled entirely by the on-board microprocessor on the HP E1340A module. Only continuous repetitive sweeping is supported. A sweep is started by the INITiate:IMMEDIATE command and advances at a fixed rate determined by the number of points in the sweep and the sweep time.

The minimum and maximum number of frequency points in a sweep are constrained by the frequency span and the sweep time.

Looking at the span considerations, and assuming that the internal 42.94967296 MHz reference oscillator is used, the minimum and maximum frequency increments per frequency point of a 4,096 point waveform are .01 Hz and 15 MHz. Thus, when sweeping a 4,096 point waveform from 1 kHz to 10 kHz, the minimum number of points is 2, and the maximum is $(10 \text{ kHz} - 1 \text{ kHz}) / .01 \text{ Hz} + 1$, or 900,001 points.

Considering sweep time, the fastest the HP E1340A can sweep is 187 μS per frequency point with AC amplitude correction disabled, or 250 μS per point with AC amplitude correction enabled. Up to 25.5 mS additional delay per frequency point is possible in 100 μS increments. Thus, the fastest amplitude corrected sweep will proceed at a rate of 15 MHz/point / 250 μS /point or 60 GHz per second. The slowest non-amplitude corrected sweep will proceed at a rate of .01 Hz/point / 25.687 mS/point or about 0.3893 Hz per second.

Also, sweeps are limited to a minimum of 2 points (the start and stop frequencies) and a maximum of 1,073,741,825 points (not a real concern, since each sweep would take a minimum of 3.1 days to complete).

The SOURCE:SWEep:POINTS command considers all these constraints when MINimum or MAXimum is specified. The SOURCE:SWEep:TIME command considers the currently specified number of points and whether AC amplitude correction is enabled when MINimum or MAXimum is specified.

[SOURce:]SWEep:COUNT

Subsystem Syntax

[SOURce:]
SWEep
:COUNT <number>
:POINts <number>
:TIME <time>

:COUNT [SOURce:]SWEep:COUNT <number> specifies the number of sweeps the HP E1340A will perform.

Since the only valid value for this count is INFINITY, there is no need to send this command. It is included for SCPI compatibility purposes only.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>number</i>	numeric	9.9E+37 INFINITY MINIMUM MAXIMUM	none
MINIMUM and MAXIMUM both select INFINITY sweeps. 9.9E+37 is equivalent to INFINITY.			

Comments

- **Executable when initiated:** Yes
- **Coupled command:** Yes
- **Related Commands:** INITiate:IMMEDIATE, ABORt
- ***RST Condition:** SOURce:SWEep:COUNT INFINITY

Example Setting the sweep count

SWE:COUN INF

Set infinite sweeps

:POINTs [SOURce:]SWEep:POINTs <*number*> selects the number of points in a frequency sweep.

The frequencies generated by the sweep are evenly spaced linearly between the frequencies specified by SOURce:FREQuency:START and STOP, or CENTER and SPAN, inclusive.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>number</i>	numeric	<i>see below</i> MINimum MAXimum	none
The legal range for <i>number</i> , as well as the MINimum and MAXimum values, is context-dependent. See the section "Sweep Considerations" at the beginning of the SOURce:SWEep subsystem for a description of the dependencies.			

- Comments**
- **Executable when initiated:** Query form only
 - **Coupled command:** Yes
 - **Related Commands:** SOURce:FREQuency:CENter, MODE, SPAN, START, and STOP
 - ***RST Condition:** SOURce:SWEep:POINTs 1001

Example Setting the number of points in the sweep

SWE:POIN 100

Set 100 points in sweep

:TIME [SOURce:]SWEep:TIME <*time*> selects the duration of the sweep. The duration is the time from the start of the sweep until when the last frequency begins to be output.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>time</i>	numeric	<i>see below</i> MINimum MAXimum	seconds
<p>With CALibration:STATE:AC OFF set: MINimum selects 0.187 mS * (<i>points</i> - 1); MAXimum selects 25.687 mS * (<i>points</i> - 1).</p> <p>With CALibration:STATE:AC ON set: MINimum selects 0.250 mS * (<i>points</i> - 1); MAXimum selects 25.75 mS * (<i>points</i> - 1).</p> <p><i>points</i> is specified by the SOURce:SWEep:POINTs command. The above values bound the valid range for <i>time</i>.</p>			

[SOURce:]SWEep:TIME

Comments

- 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 each sweep 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:

$$\text{SWEep:TIME} = \text{time} * (\text{points} - 1) / \text{points}$$

Thus, to set a repetition time of 1 S for a 100 point sweep, SWEep:TIME should be set to .99 S.

- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- **Related Commands:** SOURce:SWEep:POINTS, CAL:STATe:AC
- ***RST Condition:** SOURce:SWEep:TIME 1.05

Example Setting the duration of the sweep

SWE:TIME 20

Set sweep to take 20 seconds

[SOURce:]VOLTage

The [SOURce:]VOLTage subsystem controls the amplitude and offset values for all output waveform shapes.

Amplitude and Offset Considerations

Amplitude control is accomplished by the combination of an analog multiplier and a 20 dB attenuator. The analog multiplier scales the output of the main output DAC. It has a resolution of 11 bits where 2047 represents maximum output and 0 no output. Any specified offset voltage is summed with the multiplier output before the 20 dB attenuator. If **both** the output amplitude and offset are less than or equal to 10% of full scale (.51175 V into a 50 Ω load), the 20 dB attenuator will be used, and the analog multiplier and offset DAC values will be scaled up by a factor of 10. This results in finer low-level amplitude and offset control, as well as reduced noise on the output. If **either** the output amplitude or offset values are greater than 10% of full scale, the 20 dB attenuator will not be used. Thus, a very small amplitude signal riding on a large offset will have relatively poor amplitude resolution and will tend to be noisy.

Subsystem Syntax

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

[:LEVel][:IMMediate] [:AMPLitude]

[SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] <amplitude> sets the output amplitude when SOURce:FUNCTION:SHAPE is set to DC, RAMP, SINusoid, SQUare, TRIangle. It sets the positive full-scale output amplitude for arbitrary waveforms (SOURce:FUNCTION:SHAPE USER set); the least significant DAC code bit represents 1/2047 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

Parameter Name	Parameter Type	Range of Values	Default Units
<i>amplitude</i>	numeric	<i>see below</i> MINimum MAXimum	<i>see below</i>
<p>DC Output: When a matched load has been specified, MINimum selects -5.12 V; MAXimum selects 5.11750 V.</p> <p>Arbitrary Waveform, Ramp, Sine, Square, and Triangle Outputs: MINimum will select the equivalent of 0 V in the current default units. MAXimum selects the equivalent of the lesser of (+6.0 V - output offset value) and +5.11750 V.</p> <p>For all waveform shapes, when an open circuit load has been specified, double all the above voltages.</p> <p>These values bound the legal range of values for <i>amplitude</i>.</p> <p>Default units are specified by the SOURce:VOLTage:LEVel:IMMediate:AMPLitude:UNIT:VOLTage command.</p>			

For DC output, the amplitude is generated using the DAC; resolution is .0025 V into a matched load, .005 V into an open circuit. For all other waveform shapes, output amplitude control is implemented via a output multiplier circuit, followed by a switchable 20 dB step attenuator. The step attenuator is used only when both amplitude and offset are each below one tenth of their full scale values.

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 a 50 ohm load.

Comments

- An amplitude of -9.9e37 DBM or DBMW is equivalent to 0 V.
- **Related Commands:** SOURce:FUNCTion:SHAPE, SOURce:VOLTage:LEVel:IMMediate:OFFSet
- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- ***RST Condition:** SOURce:VOLTage:LEVel:IMMediate:AMPLitude 0 V

Example Setting output voltage

VOLT 5 VPP

Set output amplitude to 5 volts peak-to-peak

[SOURce:]VOLTage[:LEVel][:IMMediate] [:AMPLitude]:UNIT [:VOLTage]

[: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. 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:LOAD value; they are meaningless and therefore unavailable if OUTPut:LOAD INFinity is set. These units are also meaningless and unavailable for user-defined waveforms (SOURce:FUNCTion:USER set) and when directly driving the DAC from the VXIbus (SOURce:ARBitrary:DAC:SOURce VXI set).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>units</i>	discrete	DBM DBMW V VPK VPP VRMS W	none

Comments

- **Related Commands:** OUTPut:LOAD, SOURce:VOLTage:LEVel:IMMediate:AMPLitude
- **Executable when initiated:** Yes
- **Coupled command:** No
- ***RST Condition:**
SOURce:VOLTage:LEVel:IMMediate:AMPLitude:UNIT:VOLTage V

Example Setting the default voltage units

VOLT:UNIT:VOLT VPP

Set default units to volts peak-to-peak

[SOURce:]VOLTage[:LEVel][:IMMEDIATE]:OFFSet

[[: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

Parameter Name	Parameter Type	Range of Values	Default Units
<i>offset</i>	numeric	<i>see below</i> MINimum MAXimum	volts
DC Output: When a matched load has been specified, MINimum selects -5.12 V; MAXimum selects +5.11750 V.			
Arbitrary Waveform, Ramp, Sine, Square, and Triangle Outputs: When a matched load has been specified, MINimum selects the greater of (-6.0 V + output amplitude value) and -5.12 V, rounded down if needed to a multiple of 2.5 mV; MAXimum selects the lesser of (+6.0 V - output amplitude value) and +5.1175 V, again rounded down.			
For all waveform shapes, when an open circuit load has been specified, double all the voltage values above.			
The above values bound the legal range for <i>offset</i> .			

Comments

- **Related Commands:**
SOURce:VOLTage:LEVel:IMMEDIATE:AMPLitude
- **Executable when initiated:** Query form only
- **Coupled command:** Yes
- ***RST Condition:** SOURce:VOLTage:LEVel:IMMEDIATE:OFFSet 0 V

Example Setting offset voltage

VOLT:OFFS 3

Set offset voltage to 3 volts

STATus

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), and 8 (INITiated) are defined for the HP E1340A. All other bits are always zero.

Bit 0 - CALibrating: Set (1) during calibration. Cleared (0) otherwise.

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 8 - INITiated: Set (1) by the INITiate:IMMEDIATE command. Cleared (0) by the ABORt command. When outputting a single burst of waveform repetitions (ARM:LAYer2:COUNT 1 set and ARM:LAYer1:COUNT not set to INFinity), it is also cleared by any other command executed after waveform generation completes.

Questionable Signal Status Register

All bits are always 0. This register is implemented only for SCPI compatibility purposes.

Subsystem Syntax

```

STATus
:OPERation|QUEStionable
:CONDition?                [query only]
:ENABLE <unmask>
[:EVENT]?                  [query only]
:NTRansition <unmask>
:PTRansition <unmask>
:PRESet                    [no query]
    
```

STATus:OPERation|QUESTionable:CONDition?

:OPERation |QUESTionable :CONDition?

STATus:OPERation|QUESTionable:CONDition? returns the contents of the appropriate Condition Register. Reading the register does not affect its contents.

Comments

- **Executable when initiated:** Yes
- **Coupling group:** none
- **Related commands:** STATus subsystem, *SRE, *STB?
- ***RST Condition:** all bits of both Condition Registers are cleared as a result of the state present after *RST.

Example Querying the Operation Condition Register

STAT:OPER?

Query Operation Condition Register

:OPERation |QUESTionable :ENABLE

STATus:OPERation|QUESTionable:ENABLE <unmask> specifies which bits of the associated 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

Parameter Name	Parameter Type	Range of Values	Default Units
<i>unmask</i>	numeric or non-decimal numeric	0 through +32767	none

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 subsystem, *SRE, *STB?
- ***RST Condition:** unaffected
- **Power-on Condition:** STATUS:OPERation|QUESTionable:ENABLE 0

Example Setting the Operation Register enable mask

STAT:OPER:ENAB #H0100

Enable summary on Initiated bit

**:OPERation
|QUESTionable
[:EVENT]?**

STATus:OPERation|QUESTionable[:EVENT]? returns the contents of the appropriate Event Register. Reading the register clears it to 0.

Comments

- Both Event Registers are also cleared to 0 by the *CLS common command.
- **Executable when initiated:** Yes
- **Coupling group:** none
- **Related commands:** STATus subsystem, *SRE, *STB?
- ***RST Condition:** unaffected
- **Power-on Condition:** Both Event Registers are cleared to 0.

Example

Querying the Operation Event Register

STAT:EVEN?

Query Operation Event Register

**:OPERation
|QUESTionable
:NTRansition**

STATus:OPERation|QUESTionable:NTRansition <unmask> sets the negative transition mask. For each bit unmasked, a 1-to-0 transition of that bit in the associated Condition Register will set the same bit in the associated Event Register.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>unmask</i>	numeric or non-decimal numeric	0 through +32767	none

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 subsystem, *SRE, *STB?
- ***RST Condition:** unaffected
- **Power-on Condition:**
STATus:OPERation|QUESTionable:NTRansition 0

Example

Setting the Operation Register negative transition mask

STAT:OPER:NTR #H0008

Set event bit when sweeping condition is cleared

**:OPERation
|QUESTionable
:PTRansition**

STATus:OPERation|QUESTionable:PTRansition <unmask> sets the positive transition mask. For each bit unmasked, a 0-to-1 transition of that bit in the associated Condition Register will set the same bit in the associated Event Register.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>unmask</i>	numeric or non-decimal numeric	0 through +32767	none

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 subsystem, *SRE, *STB?
- ***RST Condition:** unaffected
- **Power-on Condition:**
STATus:OPERation|QUESTionable:PTRansition 32767

Example

Setting the Operation Register positive transition mask

STAT:OPER:PTR #H0100

Set event bit when initiated condition is set

:PRESet

STATus:PRESet initializes the Enable Registers and transition masks for the Operation 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 subsystem, *SRE, *STB?
- ***RST Condition:** none

Example

Presetting the STATus subsystem

STAT:PRES

Preset STATus subsystem

SYSTem

The SYSTem subsystem returns error messages and the SCPI version number to which the HP E1340A complies.

Subsystem Syntax

```
SYSTem
:ERRor?           [query only]
:VERSion?        [query only]
```

:ERRor? **SYSTem:ERRor?** returns the error messages in the error queue. See Table B-4 in Appendix B for a listing of possible error numbers and messages.

Comments

- The HP E1340A 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? returns the oldest unread error message first.
- The error queue can hold 30 error messages. If the HP E1340A 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? Query the error queue
```

:VERSion? **SYSTem:VERSion?** returns the SCPI version number to which the HP E1340A complies: "1991.0".

Comment

- **Executable when initiated:** Yes
- ***RST Condition:** none

Example Querying the SCPI revision

```
SYST:VERS? Query SCPI revision
```

TRIGger

The TRIGger subsystem operates with the ARM subsystem to control the behavior of the trigger system, as follows:

- The source for generating the individual samples of a waveform.
- The total number of samples (triggers).

Subsystem Syntax

```
TRIGger
[:START|SEQUence[1]]
:COUNT <number>
:SOURce <source>
```

[:START]:COUNT

TRIGger[:START]:COUNT <number> would normally specify the number of triggers the HP E1340A 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

Parameter Name	Parameter Type	Range of Values	Default Units
<i>number</i>	numeric	9.91e37 NAN MINimum MAXimum	none
MINimum and MAXimum select 9.91e37 triggers. 9.91E+37 is equivalent to NAN.			

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related Commands:** ABORt, INITiate:IMMEDIATE
- ***RST Condition:** TRIGger:STARt:COUNt 9.91e37

Example Setting the start trigger count

```
TRIG:COUN NAN
```

[[:START]:SOURce TRIGger[:START]:SOURce <*source*> selects the source that advances the waveform to the next sample point. The only available source is INTERNAL.

There is no need to send this command. It is included for SCPI compatibility purposes only.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>source</i>	discrete	INTERNAL	none

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related Commands:** ABORt
- ***RST Condition:** TRIGger:STARt:SOURce INTERNAL

Example Setting the start trigger source

TRIG:SOUR INT

Trigger source is internal

IEEE-488.2 Common Commands

This section describes the IEEE-488.2 Common Commands implemented in the HP E1340A. 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.

Category	Command	Title
System Data	*IDN?	Identification Query
Internal Operations	*LRN? *RST *TST?	Learn Device Setup Query Reset Command Self Test Query
Synchronization	*OPC *OPC? *WAI	Operation Complete Command Operation Complete Query Wait-to-Continue Command
Macro	*DMC <name>,<data> *EMC <enable> *EMC? *GMC? <name> *LMC? *PMC *RMC <name>	Define Macro Command Enable Macro Command Enable Macro Query Get Macro Contents Query Learn Macro Query Purge Macros Command Remove Individual Macro Command
Status & Event	*CLS *ESE <mask> *ESE? *ESR? *SRE *SRE? *STB?	Clear Status Command Standard Event Status Enable Command Standard Event Status Enable Query Standard Event Status Register Query Service Request Enable Command Service Request Enable Query Read Status Byte Query
Stored Settings	*RCL *SAV	Recall Command Save Command

- *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.

c

- Comments**
- **Executable when initiated:** Yes
 - **Coupled command:** No
 - **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

Parameter Name	Parameter Type	Range of Values	Default Units
<i>name</i>	character data	1 through 12 characters	none
<i>data</i>	block data	any valid command sequence	none

- 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 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
- **Coupled command:** No
- **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           Define macro
```

***EMC and *EMC?**

***EMC and *EMC?** ***EMC <enable>** enables and disables macro usage. When *enable* is zero, macros usage is disabled. Any non-zero value in the range of -32768 through +32767 enables macro usage.

The query form returns 1 if macro usage is enabled, 0 if disabled.

Comments

- Macro definitions are not affected by this command.
- **Executable when initiated:** Yes
- **Coupled command:** No
- ***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. *Mask* is the sum of the decimal weights of the bits to be enabled.

The query form returns the current enable mask.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>mask</i>	numeric	0 through 255	none

A 1 in a bit position enables the corresponding event; a 0 disables it.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related Commands:** *ESR?, *SRE, *STB?
- ***RST Condition:** unaffected
- **Power-On Condition:** no events are enabled

Example

Enable all error events

*ESE 60

Enable error events

***ESR?**

***ESR?** returns the value of the Standard Event Status Register. The register is then cleared (all bits 0).

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- ***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

Parameter Name	Parameter Type	Range of Values	Default Units
<i>name</i>	character data	<i>defined macro name</i>	none

- Comments**
- **Executable when initiated:** Yes
 - **Coupled command:** No
 - **Related Commands:** *DMC
 - ***RST Condition:** none
 - **Power-On Condition:** no macros are defined

Example Query macro definition

```
*GMC? RESTART Query macro definition
```

***IDN?** ***IDN?** returns identification information for the E1340A. The response consists of four fields:

```
HEWLETT-PACKARD , E1340A , 0 , A . 01 . 00
```

The first two fields identify this instrument as model number E1340A manufactured by Hewlett-Packard. The third field is 0 since the serial number of the E1340A 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
 - **Coupled command:** No
 - ***RST Condition:** none
 - **Power-On Condition:** register is cleared

*LMC?

***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
- **Coupled command:** No
- **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 E1340A 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:LIST1|2 commands, including waveform segment, segment sequence, and frequency list definitions, the STATus subsystem commands, and the CALibration:STATE and SECurity command states.

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
- **Coupled command:** No
- **Related commands:** *RCL, *RST, *SAV
- ***RST Condition:** none

***OPC** *OPC causes the E1340A 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
- **Coupled command:** No
- **Related commands:** *OPC?, *WAI
- ***RST Condition:** none

***OPC?** ***OPC?** causes the E1340A 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
 - **Coupled command:** No
 - **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
 - **Coupled command:** No
 - **Related commands:** *DMC, *RMC
 - ***RST Condition:** none

***RCL** ***RCL <number>** restores a previously stored programming state from one of the 10 possible stored state areas. *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 SOURce:LIST commands, including waveform segment definitions, the STATus subsystem commands, and the CALibration:STATE:AC state.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>number</i>	numeric	0 through 9	none

- Comments**
- **Executable when initiated:** No
 - **Coupled command:** No
 - **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
- **Coupled command:** No
- **Related commands:** *DMC, *PMC
- ***RST Condition:** none

***RST** ***RST** resets the HP E1340A as follows:

- Sets all commands to their *RST state.
- Aborts all pending operations including waveform generation.
- Sets all stored (*SAV) states to the same state as the *RST state.

*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
- Waveform segment definitions

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- ***RST Condition:** none

***SAV** ***SAV <number>** stores the current programming state into one of the 10 possible stored state areas. *Number* indicates which of the stored state areas should be used.

This command stores the states of all commands affected by *RST. Notable exceptions the SOURce:LIST commands, including waveform segment definitions, the STATus subsystem commands, and the CALibration:STATe:AC state.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>number</i>	numeric	0 through 9	none

Comments

- **Executable when initiated:** No
- **Coupled command:** No
- **Related Commands:** *LRN?, *RCL, *RST
- ***RST 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 IEEE-488.1 service request. Event and summary bits are always set and cleared in the Status Byte Register regardless of the enable mask. *Mask* is the sum of the decimal weights of the bits to be enabled.

The query form returns the current enable mask.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
<i>mask</i>	numeric	0 through 255	none

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
- **Coupled command:** No
- ***RST Condition:** unaffected
- **Power-On Condition:** no bits are enabled

Example

Enable service request on Message Available bit

*SRE 16

Enable request on MAV

***STB?**

***STB?** returns the value of the Status Byte Register. Bit 6 (decimal weight 64) is set if a service request is pending.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related commands:** *SRE
- ***RST Condition:** none

***TST?**

***TST?** causes the E1340A to execute its internal self-test and returns a value indicating the results of the test. The test includes only a test of the command module-to-card microprocessor communications.

A zero response indicates that the self-test passed. Any non-zero response indicates that the test failed. See the HP E1340A service manual for information on interpreting non-zero self-test responses.

The settings for all SCPI commands are unchanged by this command.

Comments

- **Executable when initiated:** No
- **Coupled command:** No
- ***RST Condition:** none

***WAI** causes the E1340A 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
- **Coupled command:** No
- **Related commands:** *OPC, *OPC?
- ***RST Condition:** none

Table 7-1. HP E1340A Command Quick Reference

Subsystem	Commands
TRIGger	ABORt
ARM	ARM[:STARt][:LAYer[1]]:COUNt <number> ARM[:STARt]:LAYer2:COUNt <number> ARM[:STARt]:LAYer2:SLOPe <edge> ARM[:STARt]:LAYer2:SOURce <source>
CALibration	CALibration:AC:BEGiN CALibration:AC:POINt? <value> CALibration:DATA:AC <block> CALibration:DATA[:DC] <block> CALibration[:DC]:BEGiN CALibration[:DC]:POINt? <value> CALibration:STATe:AC <state>
INITiate	INITiate[:IMMediate]
OUTPut	OUTPut:IMPedance <impedance> OUTPut:LOAD <load>
[SOURce:]ARBitrary	[SOURce:]ARBitrary:DAC:SOURce <source> [SOURce:]ARBitrary:DOWNload <source>,<dest>,<length> [SOURce:]ARBitrary:DOWNload:COMPLete
[SOURce:]FREQUency	[SOURce:]FREQUency:CENTer <center_freq> [SOURce:]FREQUency[:CW]:FIXed <frequency> [SOURce:]FREQUency:FSKey <frequency1>,<frequency2> [SOURce:]FREQUency:MODE <mode> [SOURce:]FREQUency:SPAN <freq_span> [SOURce:]FREQUency:STARt <start_freq> [SOURce:]FREQUency:STOP <stop_freq>
[SOURce:]FUNCTion	[SOURce:]FUNCTion[:SHAPE] <shape> [SOURce:]FUNCTion:USER <name>

Table 7-1. HP E1340A Command Quick Reference (continued)

Subsystem	Commands
[SOURce:]LIST	[SOURce:]LIST[:SEGMENT]:CATalog? [SOURce:]LIST[:SEGMENT]:DEFine? [SOURce:]LIST[:SEGMENT]:SElect <name> [SOURce:]LIST[:SEGMENT]:VOLTage <voltage_list> [SOURce:]LIST[:SEGMENT]:VOLTage:DAC <voltage_list> [SOURce:]LIST[:SEGMENT]:VOLTage:POINts? SOURce:]LIST:SSEquence:CATalog? [SOURce:]LIST:SSEquence:DEFine? [SOURce:]LIST:SSEquence:SElect <name> [SOURce:]LIST:SSEquence:SEquence? [SOURce:]LIST:SSEquence:SEquence:SEGMents?
[SOURce:]MARKer	[SOURce:]MARKer:FEED <source> [SOURce:]MARKer:POLarity <polarity>
[SOURce:]RAMP	[SOURce:]RAMP:POLarity <polarity>
[SOURce:]ROSCillator	[SOURce:]ROSCillator:FREQUENCY:EXTernal <frequency> [SOURce:]ROSCillator:GATE:STATE <state> [SOURce:]ROSCillator:SOURce <source>
[SOURce:]SWEep	[SOURce:]SWEep:COUNT <number> [SOURce:]SWEep:POINts <number> [SOURce:]SWEep:TIME <time>
[SOURce:]VOLTage	[SOURce:]VOLTage[:LEVel][:IMMEDIATE][:AMPLitude] <amplitude> [SOURce:]VOLTage[:LEVel][:IMMEDIATE][:AMPLitude]:UNIT[:VOLTage] <units> [SOURce:]VOLTage[:LEVel][:IMMEDIATE]:OFFSet <offset>

Table 7-1. HP E1340A Command Quick Reference (continued)

Subsystem	Commands
STATus	STATus:OPERation QUEStionable:CONDition? STATus:OPERation QUEStionable:ENABle <unmask> STATus:OPERation QUEStionable[:EVENT]? STATus:OPERation QUEStionable:NTRansition <unmask> STATus:OPERation QUEStionable:PTRansition <unmask> STATus:PRESet
SYSTEM	SYSTem:ERRor? SYSTem:VERsion?
TRIGger	TRIGger[:STARt]:COUNT <number> TRIGger[:STARt]:SOURce <source>

SCPI Conformance Information

The HP E1340A Arbitrary Function Generator conforms to the SCPI-1991.0 standard.

The following tables list all the SCPI confirmed, approved, and non-SCPI commands that the HP E1340A can execute.

Table 7-2. SCPI Confirmed Commands

ABORt	[SOURce:] ROSCillator
ARM	:SOURce <source>
[:START SEQuence[1]]	SWEep
[:LAYer[1]]	:COUNT <number>
:COUNT <number>	:POINTs <number>
:LAYer2	:TIME <time>
:COUNT <number>	
:SLOPe <edge>	STATus
:SOURce<source>	:OPERation QUEStionable
	:CONDition?
INITiate	:ENABle <unmask>
[:IMMEDIATE]	:NTRansition <unmask>
	:PTRansition <unmask>
OUTPut	:PRESet
:IMPedance <impedance>	
[SOURce:]	SYSTem
FREQUency	:ERRor?
:CENTer <center_freq>	:VERSion?
[:CW FIXed] <frequency>	
:MODE <mode>	TRIGger
:SPAN <freq_span>	[:START SEQuence[1]]
:START <start_freq>	:COUNT <number>
:STOP <stop_freq>	:SOURce <source>
FUNCTion	
[:SHAPe] <shape>	

Table 7-3. Non-SCPI Commands

CALibration	[SOURce:]
:AC	LIST
:BEgin	[:SEGMent]
:POINt? <value>	:CATalog?
:DATA	:DEFine?
:AC <block>	:SElect <name>
[:DC] <block>	:VOLTage <voltage_list>
[:DC]	:DAC <voltage_list>
:BEgin	:POINts?
:POINt? <value>	:SSEquence
:STATe	:CATalog?
:AC <state>	:DEFine?
	:SElect <name>
	:SEquence?
	:SEGMents?
OUTPut	MARKer
:LOAD <load>	:FEED <source>
	:POLarity <polarity>
[SOURce:]	RAMP
ARBitrary	:POLarity <polarity>
:DAC	ROSCillator
:SOURce <source>	:FREQuency
:DOWNload <source>,<dest>,<length>	:EXTernal <frequency>
:COMPLete	:GATE
FUNction	:STATe <state>
:USER <name>	

Introduction

This chapter describes the HP E1340A 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 203
 - The Operation Status Group. Page 205
 - The Standard Event Status Group Page 207
 - The Status Byte Status Group Page 208
 - Using the Standard Event Status Group. Page 210

Status System Registers

Operating conditions within the AFG are monitored by registers in various status groups. The status groups implemented by the AFG are:

- 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 8-1.

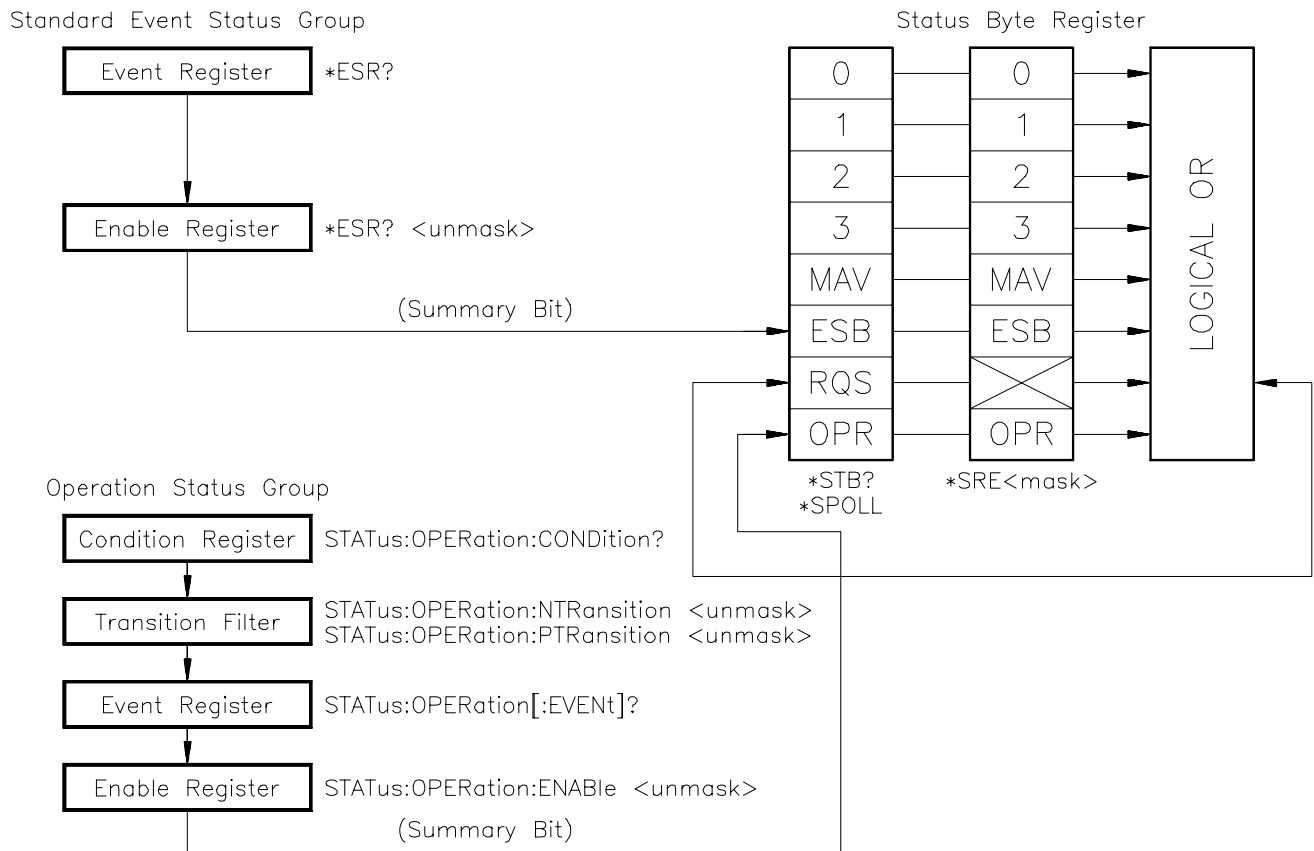


Figure 8-1. HP E1340A Status Groups and Associated Registers

The Operation Status Group

The Operation Status Group monitors current operating conditions within the AFG. The specific conditions include: CALibrating, SWEeping, and execution of the INITiate:IMMEDIATE command.

The Condition Register

Calibration, sweeping, and the INITiate:IMMEDIATE command are monitored with the following bits in the Condition Register. All other bits are unused.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
unused							INIT	unused				SWE	unused		CAL

CALibrating: Bit 0 is set (1) during the execution of the CALibration:ALL? query. The bit is cleared (0) otherwise.

SWEeping: Bit 3 is set (1) while a frequency sweep is in progress. The bit is cleared (0) when waveform generation is halted and frequency sweeping is not selected.

INITiated: Bit 8 is set (1) when the INITiate:IMMEDIATE command is executed. Waveform generation begins immediate if ARM:START:SOURce IMMEDIATE is set; otherwise waveform generation begins when the AFG receives the first start arm. The bit is cleared (0) when waveform generation is complete after the waveform is aborted (by an ABORT, or *RST command, or a Device Clear). The AFG then 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, and 8 have corresponding decimal values of 1, 8, 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, and 8 have corresponding decimal values of 1, 8, 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, and 8 have corresponding decimal values of 1, 8, and 256.)

The Enable Register is cleared at power-on, or by specifying an *<unmask>* value of 0.

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 IBASIC example programs found throughout the manual.

The Standard Event Status Register

The conditions monitored by the Standard Event Status Register are identified below.

7	6	5	4	3	2	1	0
PON	unused	CME	EXE	DDE	QYE	unused	OPC

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 un-implemented 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.

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).

7	6	5	4	3	2	1	0
OPER	RQS	ESB	MAV	QUES	unused		

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.

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.

Using the Standard Event Status Group

The following 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>

Chapter 9

HP E1340A Block Diagram Description

Chapter Contents

This chapter shows how the HP E1340A 12-Bit Arbitrary Function Generator (AFG) operates. The sections are as follows:

- AFG Description Page 211
- What is an Arbitrary Waveform?. Page 211
- Generating Waveforms Page 212

AFG Description

The AFG is a register-based VXIbus B-size card that can output standard waveforms, like SINusoid, SQUare, TRIangle, and RAMP waveforms, various arbitrary, and user defined arbitrary (i.e., USER function) waveforms. The AFG can also perform frequency sweeping, frequency-shift-keying, and output DC volts.

All waveforms that the AFG generates, except DC volts, are arbitrary waveforms. The only difference is that the user supplies the data for the arbitrary waveforms while the standard waveforms are already stored in an EPROM.

What is an Arbitrary Waveform?

Refer to Figure 9-1. 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 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. Each waveform segment consists of 4096 points.

n

Generating Waveforms

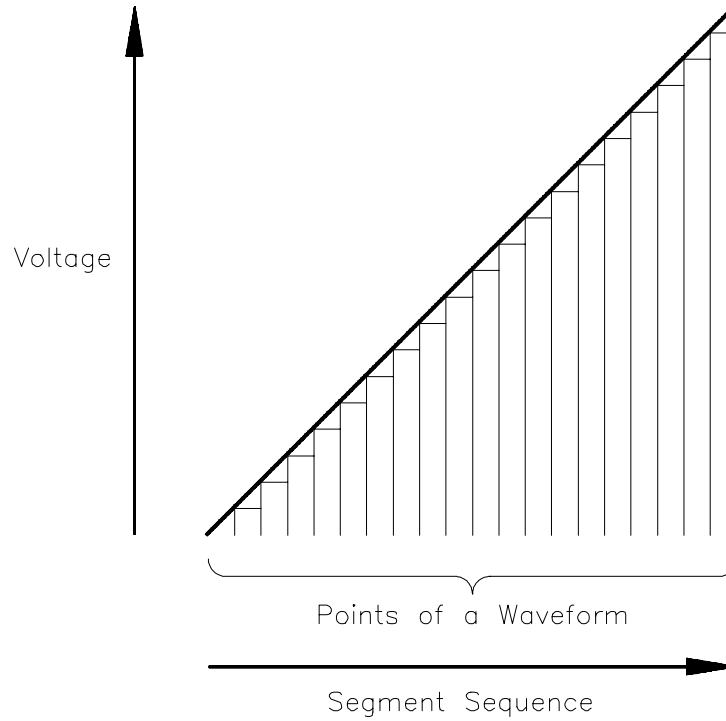


Figure 9-1. Arbitrary Waveform

Refer to Figure 9-2. The following describes the blocks that generate waveforms.

Output DAC

The AFG uses a 12-bit DAC to generate the waveforms. Each time the AFG's frequency generator clocks the DAC latch, the DAC outputs a voltage value that corresponds to the point value in memory. The bits set in the DAC determine the voltage value. The DAC codes in memory set the appropriate bits of the DAC. The RAM receives its data either from EPROMs or over the VXIbus through a register.

Besides receiving data from memory, the DAC can also receive segment data directly over the VXIbus. In this case, the data on the VXIbus immediately set the DAC to an output voltage that corresponds to the DAC code value sent. 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 +5.1175 V.

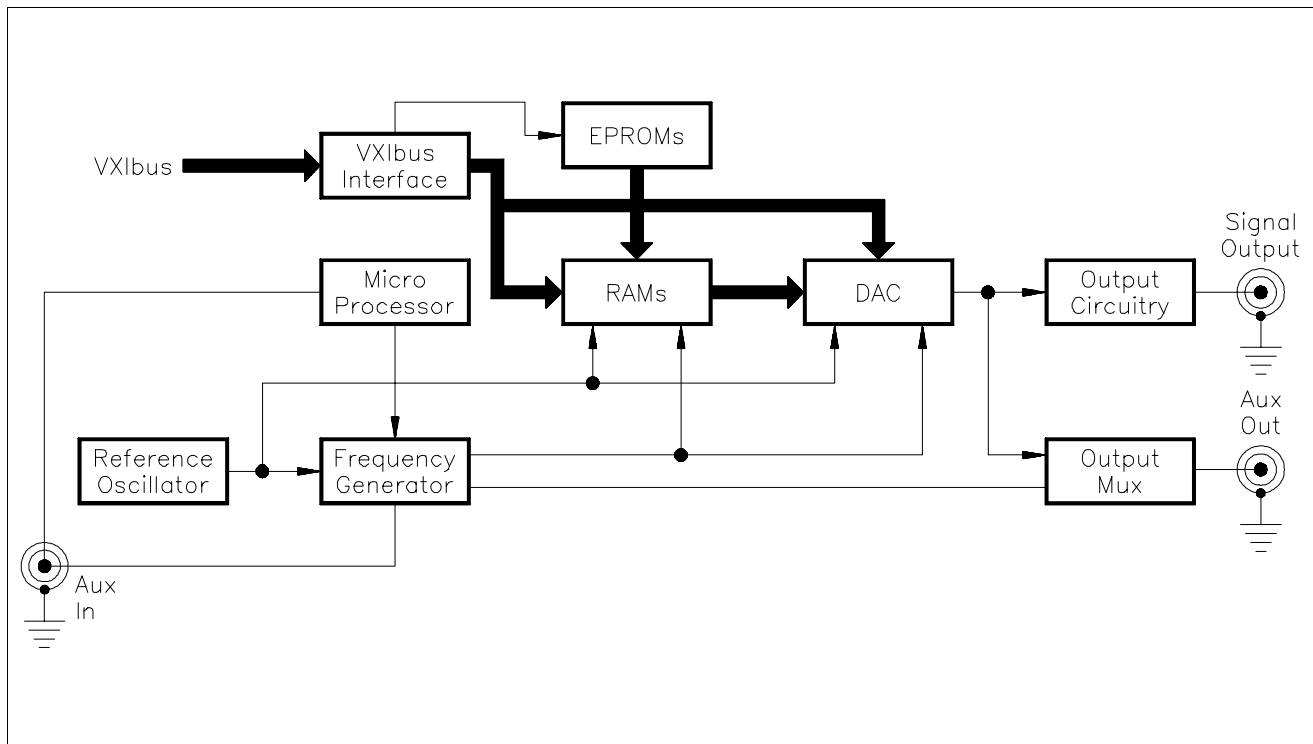


Figure 9-2. AFG Simplified Block Diagram

Memory

The memory consists of four areas of RAM in which the AFG stores the DAC codes. Concurrently with the DAC, the frequency generator also clocks a latch to output the next code in the RAM to set the DAC bits to the next point on the waveform.

The standard waveform DAC codes are stored in EPROMs. To output a standard waveform, the codes in the EPROMs are transferred to the RAM. For user generated arbitrary waveforms, the DAC codes are directly stored into the RAMs. The same operation then takes place as stated above.

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 AFG allows for user supplied reference oscillator sources for custom frequency values. The reference oscillator also clocks the RAM and the DAC.

Frequency Generators

The frequency generator sets the waveform frequency by determining the speed at which the RAM address is addressed.

Output Circuitry	The output circuitry outputs the waveform at the front panel's "Signal Output" connector. The circuitry sets the output amplitude, offset voltages, and output impedance.
Attenuator	The attenuator provides 20 dB attenuation for the output voltage. The AFG automatically sets the attenuator to the appropriate value dependent on the output amplitude selected by the user.
Output Amplifier	The output amplifier provides the necessary current to drive output loads of 50Ω. For matched loads, the output amplitude is 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 "Signal 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.
Microprocessor	The purpose of the microprocessor is to monitor the "Aux In" input line to help control frequency-shift keying, burst, and waveform hopping operation. It also interprets low-level commands received over the VME bus.
Input/Output Connectors	The AFG uses three connectors, a "Signal Output", "Aux In", and "Aux Out" connector.
Signal Output	This connector outputs the AFG's standard and arbitrary waveforms.
Aux In	This connector is used for frequency-shift keying control, waveform hopping, reference oscillator gating, reference oscillator source, or arm start source.
Aux Out	This connector outputs the marker sources selected by the [SOURCE:]MARKER command. The Output MUX connects the appropriate signals to the connector.

Appendix A

HP E1340A Specifications

Appendix Contents

This appendix contains the HP E1340A Arbitrary Function Generator operating specifications. Except as noted, the specifications apply under the following conditions:

- **Period:** 1 year
- **Temperature:** 0° - 55° C
- **Relative humidity:** 65% @ 0° - 40° C
- **Warm up time:** 1 hour

"Typical", "typ", or "nominal" values are non-warranted supplementary information provided for application assistance.

Instrument Specifications

Amplitude Resolution:	12 bits
Timebase:	Frequency: 42.949673 MHz \pm 50 PPM
Number of Active Waveforms in Memory:	4
Maximum Voltage Amplitude:	10.24V
DC Offset Range:	\pm 5.11 V into 50 Ω
Built-in Waveforms:	Sine, Square, Ramp, Triangle, Sin(x)/x Noise, Haversine
Maximum Waveform Frequency:	15 MHz - Sine 1 MHz Square, Ramp, Triangle
Output Impedance:	50 Ω \pm 1%
Fine Attenuator Resolution:	11 bits
Coarse Attenuator:	0 or 20 dB

DC Accuracy: $\pm 2.7\%$ Full Scale

AC Accuracy: ± 0.43 dB

Total Harmonic Distortion (Sine):

Frequency	Amplitude (dBm)	
	+23 to -20	10
1 to 100 kHz	-60 dB	-63 dB
100 kHz to 1 MHz	-49 dB	-54 dB
1 MHz to 10 MHz	-33 dB	-47 dB
10 MHz to 15 MHz	-30 dB	-41 dB

ic

Sine Non-Harmon Distortion:

0 to 1 MHz: -65 dBc or -45 dBm, whichever is greater
1 MHz to 10 MHz: -43 dBc or -45 dBm, whichever is greater
10 MHz to 15 MHz: -34 dBc or -45 dBm, whichever is greater

Sine Flatness: +23 dBm to -20 dBm
(referenced to 1 kHz)
frequency up to 10 MHz: ± 0.2 dB
frequency up to 15 MHz: ± 0.5 dB

Note: Maximum output at 11 MHz to 15 MHz is only guaranteed to +20 dBm.

Integral Non-Linearity: ± 5 lsb

Differential Non-Linearity: ± 5 lsb

Jitter: 23 nS

Frequency Sweep Range: 0.01 Hz to 15 MHz

Sweep Rate of Change: 0.4 Hz/sec to 2.6 MHz/sec

Frequency-Shift (FSK) Rate: to 4 kHz

Waveform Repetitions: 1 to 64534 or continuous

General and VXI Characteristics

Module Size:	B
Device Type:	Register-Based
Connectors Used:	P1
Number Slots:	1
VXIbus Interface Capability:	A16
Interrupt Level:	Not Used
Power Requirements:	
	Voltage: +5 V +12 V
	Peak Module Current: 1.2 A 0.6 A
	Peak Dynamic Current: 0.01 A 0.15 A
Watts per Slot:	13.2
Cooling per Slot:	0.11 mm H ₂ O @ 1.06 liters/s
Storage Temperature:	-40° to 75° C
EMC, RFI, Safety:	meets: FTZ 1046/1984 CSA 556B IEC 348 UL 1244
Net Weight:	0.6 kg

Appendix B

HP E1340A Useful Tables

Appendix Contents

The tables in this appendix contain information often referred to during E1340A programming. The tables in this appendix include:

- Table B-1. HP E1340A Example Program Listing Page 220
- Table B-2. HP E1340A Coupled Commands Page 222
- Table B-3. HP E1340A Power-on/Reset Page 223
- Table B-4. HP E1340A Error Messages Page 224

Table B-1. HP E1340A Example Program Listing

Program Type	Program Name	Language	Description
Introductory (Chapter 1)	SLFTST	HP IBASIC, C, QuickBASIC	E1340A Self Test.
	RSTCLS	"	Resetting and clearing the AFG.
	LRN	"	Power-on/reset configuration.
	ERRORCHK	HP IBASIC	Error checking program.
	RSTSINE	HP IBASIC, C, QuickBASIC	Sine wave output from reset settings.
Standard Waveforms (Chapter 2)	DCVOLTS	HP IBASIC, C, QuickBASIC	+5V DC voltage.
	SINEWAVE	"	1kHz, 5Vp sine wave.
	RAMPWAVE	"	4V, 10 kHz negative-going ramp.
	OUTPUNIT	"	Sets amplitude units to volts peak-to-peak.
Arbitrary Waveforms (Chapter 3)	ARB_GEN	HP IBASIC, C, QuickBASIC	Procedure for generating an arbitrary waveform (0 to 5V ramp waveform).
	MULSEG	"	Arbitrary waveform with two segments.
	ARB_HOP	"	Selects between two waveforms based on the level of the signal on the 'Aux In' BNC.
	ROM_DOWN	"	Downloads a waveform to waveform segment memory from the waveform EPROM.
	SIN_D	"	Damped sine wave arbitrary waveform.
	CHARGE	"	Exponential charge/discharge waveform.
	SPIKES	"	Sine wave with spikes.
	SIN_R	"	1/2 wave rectified sine wave.
Sweeping and Frequency-Shift Keying (Chapter 4)	SMPLSWP1	HP IBASIC, C, QuickBASIC	0 Hz to 1 MHz sweep using start and stop frequencies.
	SMPLSWP2	"	1 kHz to 21 kHz sweep using start and span frequencies.
	SWP_PVST	"	Setting the sweep time (5 kHz to 15 kHz in 0.5 seconds).
	FSK1	"	Frequency-shift keying.

Table B-1. HP E1340A Example Program Listing (Cont'd)

Program Type	Program Name	Language	Description
Arming and Marker Outputs (Chapter 5)	EXT_ARM	HP IBASIC, C, QuickBASIC	Arming the AFG with a signal applied to the 'Aux In' BNC.
	BURST	"	5 cycle burst for each external arm.
	GATE_SIG	"	Gating the output on and off.
	MARK_OUT	"	Outputs marker pulses with all amplitude points less than 0V.
High-Speed Operation (Chapter 6)	UNS_DAT	HP IBASIC, C, QuickBASIC	Downloads arbitrary waveform data as (unsigned) DAC codes.
	DACBLOK	"	Downloads arbitrary waveform data as DAC codes in a definite length block.
	VXIDOWN	"	Downloading to the AFG's Input Data register from the VXI backplane.
	VXISRCE	"	Downloading directly to the DAC from the Input Data register.
AFG Status (Chapter 9)	ERRORCHK	HP IBASIC	Monitors programming errors using the Standard Event Status Group.

Table B-2. Coupled Commands

Group	Commands
Coupled Commands	ARM[:START][:LAYer[1]]:COUNT ARM[:START]:LAYer2:COUNT ARM[:START]:LAYer2:SLOPe ARM[:START]:LAYer2:SOURce OUTPut:IMPedance OUTPut:LOAD [SOURce:]ARBitrary:DAC:SOURce [SOURce:]FREQuency:CENTer [SOURce:]FREQuency[:CW FIXed] [SOURce:]FREQuency:FSKey [SOURce:]FREQuency:MODE [SOURce:]FREQuency:SPAN [SOURce:]FREQuency:STARt [SOURce:]FREQuency:STOP [SOURce:]FUNCTion[:SHAPE] [SOURce:]FUNCTion:USER [SOURce:]MARKer:FEED [SOURce:]MARKer:POLarity [SOURce:]RAMP:POLarity [SOURce:]ROSCillator:FREQuency:EXTernal [SOURce:]ROSCillator:SOURce [SOURce:]SWEep:COUNT [SOURce:]SWEep:POINTs [SOURce:]SWEep:TIME [SOURce:]VOLTage[:LEVel][:IMMEDIATE][:AMPLitude] [SOURce:]VOLTage[:LEVel][:IMMEDIATE]:OFFSet
Un-coupled Commands	ABORt INITiate[:IMMEDIATE] [SOURce:]ARBitrary:DOWNload [SOURce:]ARBitrary:DOWNload:COMPLete [SOURce:]LIST[:SEGMENT]:CATalog? [SOURce:]LIST[:SEGMENT]:DEFine? [SOURce:]LIST[:SEGMENT]:SELect [SOURce:]LIST[:SEGMENT]:VOLTage [SOURce:]LIST[:SEGMENT]:VOLTage:DAC [SOURce:]LIST[:SEGMENT]:VOLTage:POINTs? [SOURce:]LIST:SSEquence:CATalog? [SOURce:]LIST:SSEquence:DEFine [SOURce:]LIST:SSEquence:SELect [SOURce:]LIST:SSEquence:SEQUence? [SOURce:]LIST:SSEquence:SEQUence:SEGMENTS? [SOURce:]ROSCillator:GATE:STATe [SOURce:]VOLTage[:LEVel][:IMMEDIATE][:AMPLitude]:UNIT[:VOLTage] TRIGger[:START]:COUNT TRIGger[:START]:SOURce

Table B-3. HP E1340A Power-On/Reset Configuration (as returned by *LRN?)

Parameter	Command	Power-on/Reset Setting
Macro usage	*EMC	+0
AC calibration	:CAL:STAT:AC	1 (enabled)
DAC data source	:ARB:DAC:SOUR	INTernal
Waveform amplitude units	:VOLT:AMPL:UNIT:VOLT	V
External reference osc. frequency	:ROSC:FREQ:EXT	+4.294967296E+007
Reference oscillator source	:ROSC:SOUR	INTernal
Gating State	GATE:STAT	0 (off)
Output frequency	:FREQ:FIX	+1.000000000E+004
Frequency-shift keying (FSK) frequencies	FSK	+1.000000000E+004, +1.500000000E+007
Frequency mode	MODE	FIXed
Sweep start frequency	STAR	+0.000000000E+000
Sweep stop frequency	STOP	+1.500000000E+007
Sweep count	:SWE:COUN	+9.900000000E+037
Sweep points	POIN	+1.001000000E+003
Sweep time	TIME	+1.050000000E+000
Function	:FUNC:SHAP	SINusoid
Ramp waveform polarity	:RAMP:POL	NORMal
Output amplitude	:VOLT:AMPL	+0.000000000E+000
DC offset	OFFS	+0.000000000E+000
Output load	:OUTP:LOAD	+5.000000000E+001
Waveform repetitions (burst)	:ARM:COUN	+9.900000000E+037
Waveform arm count	LAY2:COUN	+1.000000000E+000
External arm slope	SLOP	POSitive
Arm source	SOUR	IMMediate
'Aux Out' BNC source	:MARK:FEED	"SEGM"
'Aux Out' signal polarity	POL	NORMal
Arbitrary waveform sequence	:FUNC:USER	NONE

Table B-4. HP E1340A Error Messages

Code	Message	Description
-101	Invalid character	Unrecognized character in parameter.
-102	Syntax error	Command is missing a space or comma between parameters.
-103	Invalid separator	Parameter is separated by a character other than a comma.
-104	Data type error	The wrong data type (number, character, string, expression) was used when specifying the parameter.
-108	Parameter not allowed	Parameter specified in a command which does not require one.
-109	Missing parameter	Command requires a parameter(s).
-112	Program mnemonic too long	Command keyword > 12 characters.
-113	Undefined header	Command header (keyword) was incorrectly specified.
-121	Invalid character in number	A character other than a comma or number is in the middle of a number.
-123	Numeric overflow	A parameter value is greater than what can be represented with the number format.
-124	Too many digits	More than 256 digits were used to specify a number.
-128	Numeric data not allowed	A number was specified when a letter was required.
-131	Invalid suffix	Parameter suffix incorrectly specified (e.g. VO rather than VP).
-138	Suffix not allowed	Parameter suffix is specified when one is not allowed.
-141	Invalid character data	Discrete parameter specified is not a valid choice.
-144	Character data too long	A segment or sequence name is too long, or a discrete parameter is > 12 characters. Segment and sequence names must be 12 characters or less.
-148	Character data not allowed	Discrete parameter was specified when another type (e.g. numeric, boolean) is required.
-151	Invalid string data	The string data specified (e.g. for the SOUR:MARK:FEED <source> command is not a valid choice.
-158	String data not allowed	A string was specified when another parameter type (i.e. discrete, numeric, boolean) is required.

Table B-4. HP E1340A Error Messages (Cont'd)

Code	Message	Description
-161	Invalid block data	The number of bytes in a definite length data block does not equal the number of bytes indicated by the block header.
-168	Block data not allowed	Block data was specified when another parameter type (i.e. discrete, numeric, boolean) is required.
-178	Expression data not allowed	The parameter was specified as an expression (e.g. SOUR:FREQ1:FIX (A*B)).
-183	Invalid inside macro definition	Voltage or segment list is inside a macro.
-213	Init ignored	INITiate:IMMediate received while the AFG was currently initiated.
-221	Settings conflict	See "Settings Conflict Error Messages" at the end of this table.
-222	Data out of range	Parameter value is out of range for any AFG configuration (e.g. SOUR:FREQ1:FIX 1E9).
-240	Hardware error	The HP E1340A is not responding. Possible hardware failure.
-270	Macro error	*RMC <name> was executed and name is not defined.
-272	Macro execution error	Macro program data sequence could not be executed due to a syntax error within the macro definition.
-273	Illegal macro label	The macro label defined in the *DMC command was too long, the same as a common command keyword, or contained invalid header syntax.
-276	Macro recursion error	A macro program data sequence could not be executed because the sequence leads to the execution of a macro being defined.
-277	Macro redefinition not allowed	A macro label in the *DMC command could not be executed because the macro label was already defined.
-330	Self-test failed	Note the information associated with the message for a description of the failure.
-350	Too many errors	The HP E1340A A error queue is full and additional errors have occurred.
-410	Query INTERRUPTED	The HP E1340A was sent a command before it was finished responding to a query command.
-420	Query UNTERMINATED	The controller (computer) attempts to read a query response from the HP E1340A without having first sent a complete query command.

Table B-4. HP E1340A Error Messages (Cont'd)

Code	Message	Description
-430	Query DEADLOCKED	The HP E1340A's input and output buffers are full and the AFG cannot continue.
-440	Query UNTERMINATED after indefinite response	Occurs when the *IDN? query is not the last query executed in a command string.
1005	Calibration constant out of range	Illegal calibration constant was computed.
1006	Calibration constant conflict	Calibration constants used during calibration set an illegal condition.
1011	Illegal while download enabled	Commands such as SOUR:LIST cannot be executed under current conditions. Execute SOUR:ARB:DOWN:COMP to disable downloading.
1012	Illegal when not downloading	SOUR:ARB:DOWN:COMP disables downloading only after it has previously been enabled.
1014	Illegal while initiated	Command cannot be executed while the HP E1340A is in the initiated (instrument action) state.
1015	Illegal when SOUR:ARB:DAC not INT	SOUR:LIST command cannot be executed unless the DAC data source is internal.
1018	Illegal while calibrating	Command cannot be sent to the HP E1340A while the device is calibrating.
1019	Illegal while not calibrating	The command is only valid when the HP E1340A is calibrating.
1100	Illegal segment name	Attempting to download to a segment that doesn't exist, or selecting a segment name that's the same as an existing sequence name.
1102	Segment in use	Trying to delete a segment that is within a sequence.
1108	No segment name SElected	Trying to load a segment that has not been selected.
1110	Illegal sequence name	Attempting to download to a sequence that doesn't exist, or selecting a sequence name that's the same as an existing segment name.
1113	Sequence contains zero-length segment	Segment contains no voltage or DAC code data.
1118	No sequence name SElected	SOUR:LIST:SSEQ subsystem command executed without a segment sequence first selected by SOUR:LIST:SSEQ:SEL.
1119	No sequence name selected for output	INITiate command sent while SOUR:FUNC:USER NONE selected.

Table B-4. HP E1340A Error Messages (Cont'd)

Settings Conflict Error Messages
The following shows the conflicting settings and the change(s) made internally by the AFG.
SOUR:FREQ:FIX frequency < minimum; SOUR:FREQ:FIX MIN set
SOUR:FREQ:FIX frequency > maximum; SOUR:FREQ:FIX MAX set
ARB:DAC:SOUR VXI and CAL:STAT:AC ON; CAL:STAT:AC OFF set
ROSC:SOUR EXT and ROSC:FREQ:EXT not 42.94967296 MHz and CAL:STAT:AC ON; CAL:STAT:AC OFF set
multiple Aux In BNC uses; SOUR:ROSC:GATE:STAT OFF set
multiple Aux In BNC uses; ARM:LAY2:SOUR IMM set
multiple Aux In BNC uses; SOUR:FREQ:MODE FIX set
multiple Aux In BNC uses; SOUR:FUNC:USER A set
SOUR:FREQ:MODE FSK and ARM:LAY1:COUN not INF; ARM:LAY1:COUN INF set
SOUR:FREQ:FSK frequency < minimum; SOUR:FREQ:FSK MIN set
SOUR:FREQ:FSK frequency > maximum; SOUR:FREQ:FSK MAX set
ARM:LAY1:COUN INF and ARM:LAY2:SOUR EXT; ARM:LAY2:SOUR IMM set
FUNC:USER:AB and ARM:LAY1:COUN >32767; ARM:LAY1:COUN 32767 set
FUNC:USER:ABCD and ARM:LAY1:COUN > 16383; ARM:LAY1:COUN 16383 set
OUTP:LOAD INF and SOUR:VOLT unit W, DBM, or DBMW; SOUR:VOLT:AMPL MIN (in V) set
ARM:LAY2:COUN 1 and ARM:LAY2:SOUR EXT; ARM:LAY2:SOUR IMM set
ARM:LAY2:COUN >1 and ARM:LAY2:SOUR IMM; ARM:LAY2:SOUR EXT set
SOUR:FREQ:MODE SWE and ARM:LAY1:COUN not INF; ARM:LAY1:COUN INF set
SOUR:FREQ:MODE SWE and ARM:LAY2:SOUR not IMM; ARM:LAY2:SOUR IMM set
SOUR:FREQ:SPAN frequency < minimum; SOUR:FREQ:MODE FIX set

Table B-4. HP E1340A Error Messages (Cont'd)

Settings Conflict Error Messages (Cont'd)
SOUR:FREQ:STAR > SOUR:FREQ:STOP; values exchanged
SOUR:FREQ:STAR frequency < minimum; SOUR:FREQ:STAR MIN set
SOUR:FREQ:STAR frequency > maximum; SOUR:FREQ:STAR MAX set
SOUR:FREQ:STOP frequency < minimum; SOUR:FREQ:STOP MIN set
SOUR:FREQ:STOP frequency > maximum; SOUR:FREQ:STOP MAX set
SWE:POIN > maximum; SWE:POIN MAX set
SWE:TIME < minimum; SWE:TIME MIN set
SWE:TIME > maximum; SWE:TIME MAX set
SOUR:VOLT+SOUR:VOLT:OFFS < minimum; SOUR:VOLT:OFFS MIN set
SOUR:VOLT+SOUR:VOLT:OFFS > maximum; SOUR:VOLT:OFFS MAX set
SOUR:VOLT voltage < minimum; SOUR:VOLT MIN set
SOUR:VOLT voltage > maximum; SOUR:VOLT MAX set
SOUR:FUNC:SHAP not DC and SOUR:VOLT voltage < 0.0V; absolute value of SOUR:VOLT set
SOUR:ARB:DAC:SOUR not INT or SOUR:FUNC:SHAP USER and SOUR:VOLT unit not V/VPK; SOUR:VOLT:AMPL MIN (in V) set
SOUR:FUNC:SHAP DC and SOUR:VOLT unit not V: SOUR:VOLT value converted to volts
MARK:FEED "ROSC" and MARK:POL INV; MARK:POL NORM set
MARK:FEED "SEGM" and MARK:POL INV; MARK:POL NORM set
MARK:FEED "OUTP:ZERO" and MARK:POL NORM; MARK:POL INV set

Appendix C

HP E1340A Register-Based Programming

Appendix Contents

The HP E1340A Arbitrary Function Generator (AFG) is a register-based device which does not support the VXIbus word serial protocol. When a SCPI command is sent to the AFG, the AFG driver in the HP E1300/01 Mainframe (Series B) or in the HP E1405/E1406 Command Module (Series C) parses the command and writes the information to the AFG registers.

Register-based programming is a series of reads and writes **directly** to the AFG registers. This increases throughput speed since command parsing is eliminated and the registers can be accessed from the VXI backplane (with IBASIC or an embedded controller).

This appendix contains the information you need for register-based programming. The contents include:

- Register Addressing Page 229
- Computer Configurations Page 232
- Register Descriptions Page 234
- Command Descriptions and Formats Page 239
- Program Timing and Execution Page 252
- Example Programs Page 256

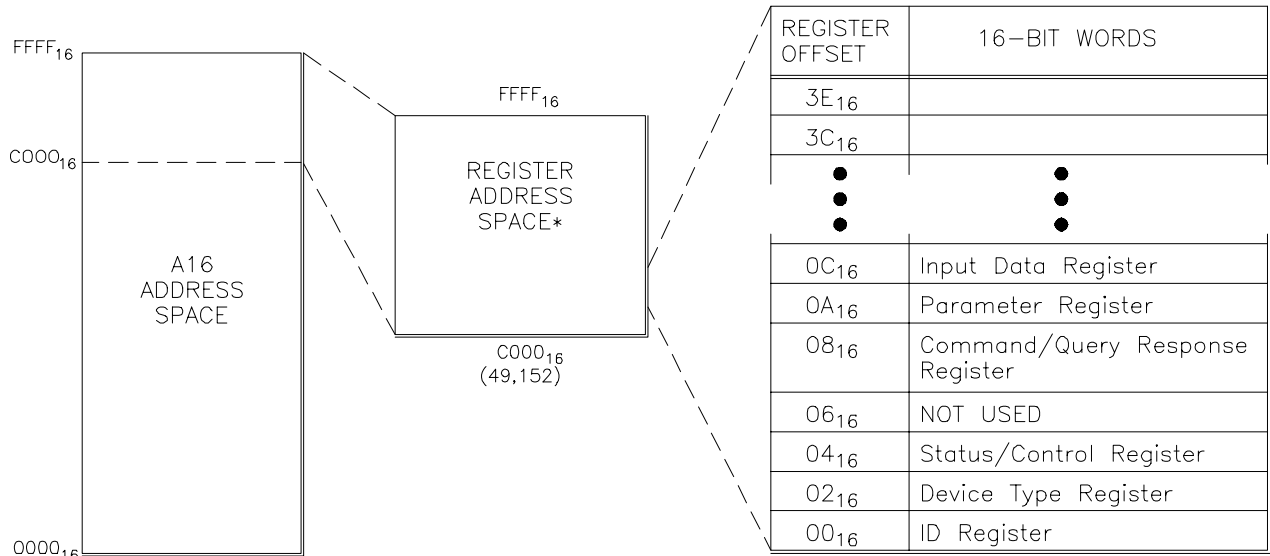
Register Addressing

Register addresses for register-based devices are located in the upper 25% of VXI A16 address space. Every VXI device (up to 256 devices) is allocated a 32 word (64 byte) block of addresses. The AFG uses six of the 64 addresses allocated.

Figure C-1A shows the register address location within A16. Figure C-1B shows the location of A16 address space in the HP E1300/01 Mainframe and HP E1405/E1406 Command Module.

The Base Address

When you are reading or writing to an AFG register, a hexadecimal or decimal register address is specified. This address consists of a A16 base address plus a register offset or register number.

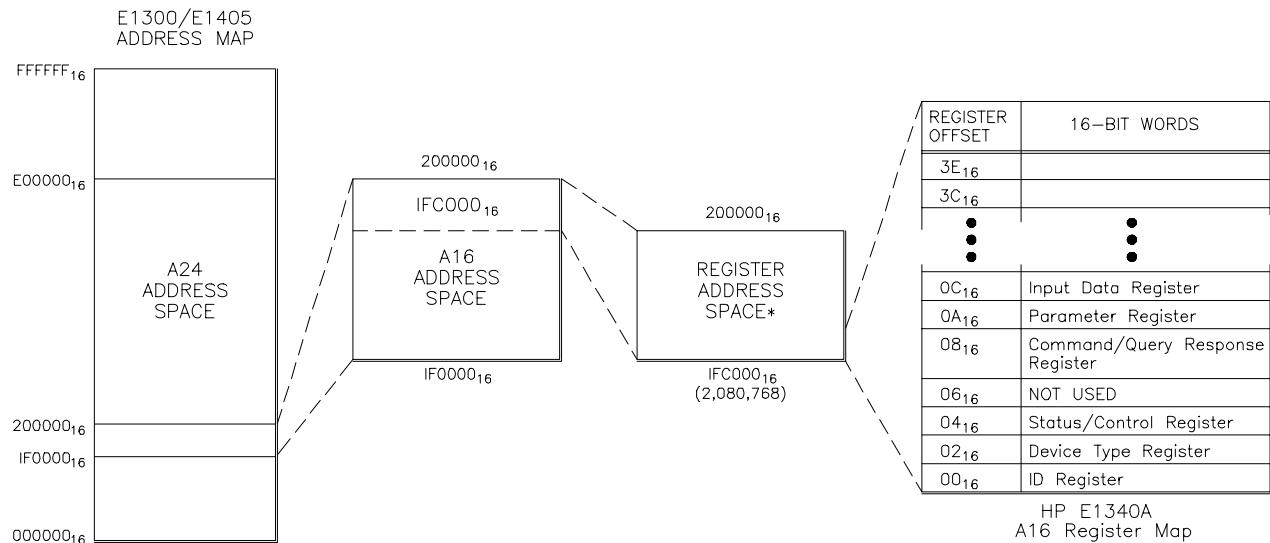


HP E1340A
A16 Register Map

* Base Address = C000₁₆ + (logical address * 64)₁₆
or
49,152 + (logical address * 64)₁₀

Register Address = Base Address + Register Offset

Figure C-1A. HP E1340A AFG Registers within A16 Address Space



HP E1340A
A16 Register Map

* Base Address = IFC000₁₆ + (logical address * 64)₁₆
or
2,080,768 + (logical address * 64)₁₀

Register Address = Base Address + Register Offset

Figure C-1B. AFG Registers within HP E1300/E1405 A16 Address Space

The A16 base address used in register-based programming depends on whether the A16 address space is located inside the E1300(01) Mainframe/E1405/E1406 Command Module or elsewhere (e.g. embedded computer). Figures C-1A, C-1B, and Table C-1 enable you to determine the base address for the following computer configurations:

- E1300/E1301 Instrument BASIC (IBASIC)
- External Computer over HP-IB to E1300/E1301 Mainframe or E1405/E1406 Command Module
- V/360 Embedded Controller (C-Size system)

Table C-1. Computer Configurations used with the HP E1340A

Computer	Programming Method	Base Address
E1300/E1301 IBASIC (Absolute Addressing)	READIO (-9826,Base_addr + offset) WRITEIO -9826,Base_addr + offset;data (positive select code = byte read or write negative select code = word read or write)	Base_addr = $1FC000_{16} + (LADDR * 64)_{16}$ or = $2,080,768 + (LADDR * 64)$ offset = register offset (Figure C-1A)
(Select Code 8)	READIO (8,Base_addr + reg number) WRITEIO 8,Base_addr + reg number;data	Base_addr = $LADDR * 256$ reg number = offset (Figure C-1A)/2
External Computer (over HP-IB to E1300/E1301 Mainframe or E1405/E1406 Command Module)	VXI:READ? logical_address,offset VXI:WRITE logical_address,offset,data DIAG:PEEK? Base_addr + offset,width DIAG:POKE Base_addr + offset,width,data	AFG logical address setting (LADDR) offset = register offset (Figure C-1A) Base_addr = $1FC000_{16} + (LADDR * 64)_{16}$ or = $2,080,768 + (LADDR * 64)$ offset = register offset (Figure C-1A)
V/360 Embedded Computer (C-Size system)	READIO (-16,Base_addr + offset) WRITEIO -16,Base_addr + offset;data (positive select code = byte read or write negative select code = word read or write)	Base_addr = $C000_{16} + (LADDR * 64)_{16}$ or = $49,152 + (LADDR * 64)$ offset = register offset (Figure C-1B)
LADDR : AFG logical address. ($LADDR * 64)_{16}$: multiply quantity, then convert to a hexadecimal number (e.g. $80 * 64)_{16} = 1400_{16}$. When using DIAG:PEEK? and DIAG:POKE, the width (number of bits) is 8 or 16.		

Computer Configurations

This section contains performance and functional information on the computer configurations that can be used with register-based programming.

Throughput Speed

Throughput speed is based on the amount of command parsing and whether the registers are accessed from the VXI backplane or from the HP-IB. The computer configurations which allow faster throughput relative to each other are summarized below:

1. HP E1480A V/360 Controller with READIO and WRITEIO (register access is from VXI backplane).
2. E1300/01 IBASIC absolute addressing with READIO and WRITEIO (register access is from VXI backplane).
3. E1300/01 IBASIC select code 8 with READIO and WRITEIO (register access is from VXI backplane).
4. External Computer using DIAG:PEEK? and DIAG:POKE (register access is over HP-IB).
5. External Computer using VXI:READ? and VXI:WRITE (register access is over HP-IB).

Embedded Computer Programming (C-Size Systems)

If the E1340A AFG is part of a C-Size VXI system, the fastest throughput is achieved using an embedded computer such as the HP E1480 V/360. The embedded computer allows you to access the registers from the VXIbus backplane, and since READIO and WRITEIO are used, there is no parsing of SCPI command headers.

IBASIC Programming

When the E1340A AFG is programmed using the E1300/E1301 Mainframe's Instrument BASIC (IBASIC), two methods of accessing the registers are through absolute addressing or using select code 8.

Absolute Addressing and Select Code 8

Absolute addressing is faster than select code 8 since the complete register address (including the A16 starting location $1FC000_{16}$) is specified. When select code 8 is used, the IBASIC processor must calculate the complete register address based on the logical address specified (Table C-1).

The Register Offset and Register Number

Depending on whether absolute addressing or select code 8 is used, either a register offset or register number is specified as part of the register address. Absolute addressing specifies a register offset, which is the register's location in the block of 64 address bytes. For example, the AFG's Command Register has an offset of 08_{16} . When you write a command to this register, the offset is added to the base address to form the register address (using a logical address of 80):

$$\begin{aligned}\text{register address} &= \text{base address} + \text{register offset} \\ &= 1FC000_{16} + (80 * 64)_{16} + 08_{16} \\ &= 1FC000_{16} + 1400_{16} + 08_{16} = \mathbf{1FD408}_{16}\end{aligned}$$

or

$$\begin{aligned}&= 2,080,768 + (80 * 64) + 8 \\ &= 2,080,768 + 5120 + 8 = \mathbf{2,085,896}\end{aligned}$$

Using select code 8 requires that you specify a **register number**. The register number is the register offset/2. Referring to Figure C-1A, the Command Register with an offset of 08 , is register number 4.

Note

The example programs found in this appendix are IBASIC programs using absolute addressing.

External Computer Programming

When the AFG is programmed by an external computer through the E1300/E1301 mainframe or E1405/E1406 Command Module, the registers are accessed using DIAG:PEEK? and DIAG:POKE, or VXI:READ? and VXI:WRITE.

DIAG:PEEK?/DIAG:POKE and VXI:READ?/VXI:WRITE

Throughput speed using DIAG:PEEK? and DIAG:POKE is faster than VXI:READ? and VXI:WRITE because the complete register address (including the A16 starting location $1FC000_{16}$) is specified. VXI:READ? and VXI:WRITE specify the device logical address and register offset only. Thus, the E1300/01/E1405/06 processor must calculate the complete register address which decreases throughput speed.

IBASIC programming using absolute addressing or select code 8 is faster than either DIAG:PEEK? and DIAG:POKE or VXI:READ? and VXI:WRITE because the registers are accessed from the VXIbus backplane rather than from the HP-IB. Also, READIO and WRITEIO are not parsed.

Programming Guidelines

As noted, the example programs in this appendix are written in IBASIC. If your application involves arbitrary waveforms, and waveform computation is intensive and download time is not critical, an external controller should be used. If download time is important and computation time is not critical, IBASIC enables fast downloading through direct access to the VXI backplane.

Declaring IBASIC Variables in COM (common) Memory

When writing or modifying IBASIC programs, array variables can be declared in COM (common) memory. Variables not in COM memory reside in the IBASIC stack. The "stack" is a 32 kByte (default) segment of memory which contains components such as pointers and local variables for subprograms and declarations. When too many variables (or too large an array) are in the stack, Error 2 - "Memory Overflow" will occur. If a memory overflow occurs, the stack size can be changed with the command `PROG:MALLocate <nbytes>` (see the *Instrument BASIC User's Manual* for more information).

Register Descriptions

There are four READ and four WRITE registers on the AFG. This section contains a description and a bit map of each register.

The READ Registers

The following READ registers are located on the AFG.

- ID Register (base + 00₁₆)
- Device Type Register (base + 02₁₆)
- Status Register (base + 04₁₆)
- Query Response Register (base + 08₁₆)

Examples and program statements in this appendix use 16-bit reads. In most cases, however, only the lower eight bits are used.

The ID Register

The AFG's ID Register indicates the classification, addressing mode, and the manufacturer of the device.

Address	15	14	13	12	11 - 0
base + 00 ₁₆	Device Class		Address Mode		Manufacturer ID

Device Classification. Bits 15 and 14 classify a device as one of the following:

- 0 0 memory device
- 0 1 extended device
- 1 0 message-based device
- 1 1 register-based device

The HP E1340 Arbitrary Function Generator is a register-based device.

Addressing Mode. Bits 13 and 12 indicate the addressing mode used by the device:

- 0 0 A16/A24 address mode
- 0 1 A16/A32 address mode
- 1 0 RESERVED
- 1 1 A16 address mode

The HP E1340 AFG uses the A16 address mode.

Manufacturer ID. Bits 11 through 0 identify the manufacturer of the device. Hewlett-Packard's ID number is 4095, which corresponds to bits 11 - 0 being set to "1".

Given the device classification, addressing space, and manufacturer of the HP E1340 AFG, reading the ID register returns FFFF₁₆.

The Device Type Register

The Device Type Register contains a model code which identifies the device.

Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
base + 02 ₁₆	Model Code															

Model Code. The model code of the HP E1340 AFG is FFA0₁₆.

The Status Register

The flow and control of register-based programs are determined by the Status Register. This register is continually monitored to determine when to send a command, when to send a parameter, and when data is available.

Address	15 - 8	7	6	5	4	3	2	1	0
base + 04 ₁₆	FF ₁₆	DONE	1	Burst Status	0	Pass/SysFail	Pass/Sys Fail	Resp Buffer Full	Cmd Buffer empty

DONE. A zero (0) in bit 7 indicates that the operation performed by the current command has finished. Bit 7 is set to one (1) when a command is received and is being processed.

The validity of this bit is determined by bit 0. See “Status Bit Precedence” for more information.

Burst Status. A zero (0) in bit 5 indicates the burst mode is set and a burst is in progress. A one (1) indicates the burst mode is set and the burst is complete. The bit is undefined if the burst mode is not set.

Pass/SysFail. A zero (0) in bit 3 indicates the AFG is executing a reset, or is executing or has failed its self-test. A one (1) indicates the reset is finished or the self-test passed.

Bit 2 performs the same function.

Response Buffer Full. A one (1) in bit 1 indicates data returned by a query is in the Query Response Register. The bit is cleared (0) when the response is read from the register.

Command Buffer Empty. A one (1) in bit 0 indicates a command or parameter can be written to the Command or Parameter Register. The bit is cleared (0) when the command or parameter is received. Bit 0 also determines the validity of bit 7. See “Status Bit Precedence” for more information.

Status Bit Precedence

In addition to the conditions the bits monitor, certain status bits indicate the validity of other bits in the Status Register. This solves race situations between selected bits.

When bit 0 is zero (0), **bit 7 is invalid.** This allows the AFG to set bit 7 (set it to one (1)) to indicate that a command or parameter is being processed. When **Bit 7** is zero (0), **bit 1 is invalid.** This allows the AFG time to set those bits to the correct states based on the conditions they represent.

The Query Response Register

When the AFG's configuration (e.g. function, frequency,...) is queried, the response is sent to the Query Response Register.

Address	15 - 8	7	6	5	4	3	2	1	0
base + 08 ₁₆	not used	Query Response							

Query Response. The response returned to the register is the contents of an AFG memory location containing information about the queried parameter. Responses are 16-bits, of which only the lower eight bits are used. See the "Example Programs" section for more information.

The WRITE Registers

The following WRITE registers are located on the AFG.

- Control Register (base + 04₁₆)
- Command Register (base + 08₁₆)
- Parameter Register (base + 0A₁₆)
- Input Data Register (base + 0C₁₆)

Examples and program statements in this appendix use 16-bit writes. Except for the Input Data Register, the WRITE registers only use the lower eight bits.

The Control Register

The Control Register is used to perform a hardware reset of the AFG and to disable the AFG from driving the SYSFAIL line.

Address	15 - 8	7	6	5	4	3	2	1	0
base + 04 ₁₆	Not Used	X	X	X	X	X	X	SYSFAIL Inhibit	Reset

Resetting the AFG. Writing a one (1) to bit 0 (hardware) resets the AFG. Writing a zero (0) turns the reset function off. Bit 0 must be a "1" for at least 2 μs for the reset to occur. If bit 0 remains a 1, the AFG continually resets.

Inhibiting SYSFAIL. Writing a one (1) to bit 1 prevents the AFG from driving the SYSFAIL line. Writing a zero (0) allows the AFG to drive SYSFAIL during a reset.

Every program (except "Querying AFG Parameters") in the "Programming Examples" section begins by resetting the AFG.

The Command and Parameter Registers

Commands and parameters are opcodes or data bytes written to the Command and Parameter Registers.

Address	15 - 8	7	6	5	4	3	2	1	0
base + 08 ₁₆	not used	Command Opcode							

Address	15 - 8	7	6	5	4	3	2	1	0
base + 0A ₁₆	not used	Parameter Byte							

Command Format. A command consists of an opcode written to the Command Register, plus zero to six parameters written to the Parameter Register. A command opcode written to the Command Register might appear as:

8,1

where "8" is the offset of the Command Register and "1" is the opcode. A parameter might appear as:

10,128

where "10" is the offset of the Parameter Register and 128 is the value of the parameter byte.

The AFG commands and parameters are covered in the section "Command Descriptions and Formats". The sequence and timing involved in sending commands are covered in the section "Program Timing and Execution".

The Input Data Register

The Input Data Register is used to download amplitude points to RAM or directly to the AFG DAC.

Address	15 - 12	11	10	9	8	7	6	5	4	3	2	1	0
Base + 0C ₁₆	Not Used	Amplitude data											

Command Descriptions and Formats

The AFG commands and parameters are in the form of opcodes and data bytes. Table C-2 lists the command opcodes and the format of the parameters.

Table C-2. HP E1340A Command Opcodes and Parameter Formats

Setting the function and loading RAM					Command Opcode = 5			
Bit	7	6	5	4	3	2	1	0
Bit Weight	128	64	32	16	8	4	2	1
Purpose	DAC Source	Download Mode	Waveform selection (waveform EEPROM)			RAM		
Setting	0 - Input Data Register 1 - waveform EEPROM	0 - waveform appears at output as it is loaded into RAM from the Input Data Register. 1 - output remains at 0V as waveform is loaded into RAM from the Input Data Register.	0 0 0 0 - Sine wave 0 0 0 1 - Triangle wave 0 0 1 0 - Sin (x)/x for 8.25 radians 0 0 1 1 - Haversine 0 1 0 0 - Square wave 0 1 0 1 - First 10 terms of a square wave 0 1 1 0 - First 4 terms of a square wave 0 1 1 1 - Falling ramp 1 0 0 0 - First 20 terms of a falling ramp 1 0 0 1 - Rising ramp 1 0 1 0 - First 20 terms of a rising ramp 1 0 1 1 - White noise 1 1 0 0 - Modulated white noise 1 1 0 1 - A 3rd, 4th, 5th harmonic chord 1 1 1 0 - 8 cycles of linear rising sine wave 1 1 1 1 - Positive half-cycle sine wave			0 0 - RAM 0 0 1 - RAM 1 1 0 - RAM 2 1 1 - RAM 3		

Setting Frequency Register 1: Fixed frequency (CW) Sweep start frequency 1st FSK frequency					Command Opcode = 1			
Bits	7	6	5	4	3	2	1	0
Byte 1 ----->	Most Significant Byte of Frequency							
Byte 2 ----->	2nd Significant Byte of Frequency							
Byte 3 ----->	3rd Significant Byte of Frequency							
Byte 4 ----->	Least Significant Byte of Frequency							

Setting Frequency Register 2: Fixed frequency (CW) Sweep stop frequency 2nd FSK frequency					Command Opcode = 2			
Bits	7	6	5	4	3	2	1	0
Byte 1 ----->	Most Significant Byte of Frequency							
Byte 2 ----->	2nd Significant Byte of Frequency							
Byte 3 ----->	3rd Significant Byte of Frequency							
Byte 4 ----->	Least Significant Byte of Frequency							

Table C-2. HP E1340A Command Opcodes and Parameter Formats (Cont'd)

Setting the Attenuation				Command Opcode = 4				
Bit	7	6	5	4	3	2	1	0
Bit Weight	128	64	32	16	8	4	2	1
Byte 1 ----->	Atten	Corr	not used		MS nibble of attenuation			
Setting	0 - enable 20 dB attenuator 1 - disable 20 dB attenuator	0 - enable sine wave correction 1 - disable sine wave correction	X					
Byte 2 ----->	Least Significant Byte of Attenuation							

Setting the Offset:					Command Opcode = 3			
Bits	7	6	5	4	3	2	1	0
Byte 1 ----->	not used				MS Nibble of Offset			
Byte 2 ----->	Least Significant Byte of Offset							

Setting the Sweep Rate					Command Opcode = 7			
Bits	7	6	5	4	3	2	1	0
Byte 1 ----->	Most Significant Byte of Phase Increment							
Byte 2 ----->	2nd Significant Byte of Phase Increment							
Byte 3 ----->	3rd Significant Byte of Phase Increment							
Byte 4 ----->	Least Significant Byte of Phase Increment							
Byte 5 ----->	Delay (tic) Count							
Byte 6 ----->	0 (Linear Sweep)							

Setting the Burst Count					Command Opcode = 8			
Bits	7	6	5	4	3	2	1	0
Byte 1 ----->	Most Significant Byte of Burst Count							
Byte 2 ----->	Least Significant Byte of Burst Count							

Table C-2. HP E1340A Command Opcodes and Parameter Formats (Cont'd)

Setting the Clock Source and Output Mode						Command Opcode = 9		
Byte 1								
Bit	7	6	5	4	3	2	1	0
Bit Weight	128	64	32	16	8	4	2	1
Purpose	Aux Out Level	Aux In Level	Clock Source			Aux Out (Marker)		Aux In
Setting	0 - Normal 1 - Inverted	0 - Normal 1 - Inverted	0 0 0 - External 0 0 1 - Gated 0 1 0 - Burst 0 1 1 - Internal 1 0 0 - Input Data Register (base + 0C ₁₆) 1 0 1 - reserved 1 1 0 - No clock 1 1 1 - No clock			0 0 - 42 MHz osc 0 1 - Pulse/cycle 1 0 - Zero cross 1 1 - Sweep sync		0 - Wave hop - FSK - Triggered Burst 1 - External Clock - Gate (internal clock)
Byte 2								
Bit	7	6	5	4	3	2	1	0
Bit Weight	128	64	32	16	8	4	2	1
Purpose	RAM Bank		Frequency Register	Output Mode			RAM Size	
Setting	0 0 - RAM 0 0 1 - RAM 1 1 0 - RAM 2 1 1 - RAM 3		0 - Register 1 1 - Register 2	0 0 0 - Continuous (fixed) 0 0 1 - FSK 0 1 0 - Wave hop 0 1 1 - Internal burst 1 0 0 - External burst 1 0 1 - Sweep 1 1 0 - Direct DAC access 1 1 1 - Fast frequency change			0 0 - 4k 0 1 - 8k 1 0 - 16k	

Reading AFG Settings					Command Opcode = 13			
Bits	7	6	5	4	3	2	1	0
Byte 1 ----->	Query Byte							

Soft Reset					Command Opcode = 14			
Bits	7	6	5	4	3	2	1	0
	no parameter							

Setting the Output Function and Loading RAM

The output function is set by selecting the DAC source, selecting a waveform from the waveform EPROM, and then loading the waveform into RAM.

The function is a **one byte** parameter. The value of the byte is the sum of the bits set (bit weights). For example, to set a sine wave as the output function and to load it into RAM 0, the value of the byte would be 128 which sets the bit pattern:

1 0 0 0 0 0 0 0

To set a triangle wave as the output function and load it into RAM 1, the value of the byte would be 133, which sets the bit pattern:

1 0 0 0 0 1 0 1

Downloading an Arbitrary Waveform

The DAC source when downloading an arbitrary waveform into RAM is the Input Data Register (base + 0C16). Arbitrary waveforms are loaded into RAM in the same manner as standard waveforms from the waveform EPROM. For example, to download an arbitrary waveform into RAM 0, the value of the "function" byte would be 0 which sets the bit pattern:

0 0 0 0 0 0 0 0

Arbitrary waveforms must have exactly 4096 points. Undefined points appear at the output as random values. The "Example Programs" section demonstrates how to download and output arbitrary waveforms.

Download Mode

Bit 6 of the "setting the function and loading RAM" byte (Table C-2) is used to control whether or not arbitrary waveform data appears at the output as the data is loaded into RAM from the Input Data Register. Setting bit 6 to "1" sets the output to 0V as the waveform is downloaded into RAM. When bit 6 is set to "0", the output tracks the Input Data Register as the data is loaded into RAM.

Multiple Waveforms in a Single Output Sequence

The AFG's output signal can consist of up to four waveform segments (RAM 0 through RAM 3). Each RAM is loaded from the waveform EPROM, from the Input Data Register, or from both. An example of this feature is located in the "Example Programs" section.

Setting the Signal Frequency

There are five frequency parameters used with the AFG:

- fixed (continuous) frequency
- sweep start frequency
- sweep stop frequency
- FSK frequency 1
- FSK frequency 2

Frequency Register 1 (command opcode 1) is used to set a fixed frequency, the sweep start frequency, and the first FSK frequency. Frequency Register 2 (command opcode 2) is used to set a fixed frequency, the sweep stop frequency, and the second FSK frequency.

Format The frequency is a **four-byte** parameter expressed in 10's of millihertz. The format of the frequency value set from the Parameter Register is as follows:

$$\text{Frequency} = (F_o * 2^{32} / \text{Ref osc})$$

Where F_o is the output frequency and Ref osc is the reference oscillator frequency. If the AFG's internal 42.94967296 MHz oscillator is used, the equation simplifies to:

$$\text{Frequency} = (F_o * 100)$$

Frequency is converted to the form:

$$(F_1 * 2^{24} + F_2 * 2^{16} + F_3 * 2^8 + F_4) / 100$$

For example, to send 1kHz in four bytes (using the AFG internal oscillator):

$$\begin{aligned} \text{Frequency} &= (1\text{E}3 * 100) = 100,000 \\ &= 00 \quad 01 \quad 86 \quad \text{A0}_{16} \\ &= 0 \quad 1 \quad 134 \quad 160_{10} \\ &\quad \text{byte 1} \quad \text{byte 2} \quad \text{byte 3} \quad \text{byte 4} \end{aligned}$$

Arbitrary Waveform Frequency

The maximum frequency of an arbitrary waveform using the AFG internal oscillator is 10.48576 kHz (42.94967296E6/4096). Higher frequencies can be achieved, however; the arbitrary waveform cannot contain frequency components greater than 15 MHz. At 10.48576 kHz, each amplitude point is output once. At lower frequencies, some or all points may be output more than once. At higher frequencies, some points are skipped on each repetition.

Frequency of Multiple Segment Waveforms

When two or four waveform segments make up the output waveform, the waveform repetition frequency is 1/2 or 1/4 of the programmed frequency. For example, if the RAM size is 8k, the waveforms in RAM 1 and RAM 2 comprise the output waveform. If the programmed frequency is 1 kHz, the output frequency is 500 Hz. Similarly, if the RAM size is 16k, the waveforms in RAM 0 through RAM 3 make up the output waveform. If the programmed frequency is 1 kHz, the output frequency is 250 Hz. For 1 kHz, you must either double or quadruple the programmed frequency.

Setting the Output Attenuation

The maximum output voltage is +5.1175V / -5.12V into 50 Ω.

The amplitude is controlled by the 12-bit output attenuator. The attenuation value written to the AFG is based on the desired amplitude (volts peak) and the output load:

Amplitude > + 0.51175 into 50 Ω:

$$\text{Atten} = (\text{Ampl} / 5.1175) * 2047 \quad (\text{no attenuation})$$

Amplitude < + 0.51175 into 50 Ω:

$$\text{Atten} = (\text{Ampl} / 0.51175) * 2047 \quad (20 \text{ dB attenuation})$$

Amplitude > + 1.0235 into open circuit:

$$\text{Atten} = (\text{Ampl} / 10.235) * 2047 \quad (\text{no attenuation})$$

Amplitude < + 1.0235 into open circuit:

$$\text{Atten} = (\text{Ampl} / 1.0235) * 2047 \quad (20 \text{ dB attenuation})$$

Atten is the attenuation value set from the Parameter Register and Ampl is the desired output amplitude in volts peak.

Format

The attenuation is a **two-byte** parameter whose bits are defined in Table C-2. The attenuation value (Atten) is converted to the form:

$$A_1 * 2^8 + A_2$$

As an example, to set 2 V_{pk} into 50Ω:

$$\text{Atten} = (2 / 5.1175) * 2047$$

$$= 03 \quad 20_{16}$$

$$= 3 \quad 32_{10} + 128$$

$$= 131 \quad 32_{10}$$

byte 1 byte 2

128 is added to set bit 7, which turns the 20 dB attenuator off.

Turning Attenuation Off

Setting bit 7 (Atten) of the MSB to "1" sets no attenuation (Table C-2). Bit 7 should be set to "1" for amplitudes > 0.51175V into 50 Ω, or for amplitudes > 1.0235V into an open circuit. Setting bit 7 to "0" attenuates the output by 20 dB. Bit 7 should be set to "0" for amplitudes < 0.51175V into 50Ω, or for amplitudes < 1.0235V into an open circuit.

Amplitude Correction

Setting bit 6 (Corr) to "1" (adding 64) turns off amplitude correction (Table C-2). Setting bit 6 to "0" enables amplitude correction. When enabled, the output is corrected such that the amplitude remains level for all frequencies up to 15 MHz. The correction computation is only valid and should only be enabled with a reference oscillator frequency (internal or external) of 42.94967296 MHz.

Caution

If amplitude correction is in effect for a programmed number of cycles (burst), correction will remain in effect after the burst completes. This can result in an output signal 2.5 times the programmed value. Sine wave correction is recommended for a continuous signal only.

Setting the Amplitude Offset

The offset of the output signal is controlled by the 12-bit DAC. The offset value written to the AFG is based on the desired offset and the output load:

Offset with no signal attenuation into 50 Ω:

$$\text{Offset} = (V_{\text{off}} / .0025) + 2048$$

Offset with no signal attenuation into open circuit:

$$\text{Offset} = (V_{\text{off}} / .005) + 2048$$

Offset with 20 dB attenuation into 50 Ω:

$$\text{Offset} = ((V_{\text{off}} / .0025) * 10) + 2048$$

Offset with 20 dB attenuation into open circuit:

$$\text{Offset} = ((V_{\text{off}} / .005) * 10) + 2048$$

Offset is the offset value set from the Parameter Register and V_{off} is the desired offset.

Format The offset is a **two-byte** parameter whose bits are defined in Table C-2. The offset value (Offset) is converted to the form:

$$A_1 * 2^8 + A_2$$

For example, to add a 1V dc offset to a 2 Vp signal (into 50 Ω):

$$\begin{aligned} \text{Offset} &= (1 / .0025) + 2048 \\ &= 09 \quad 90_{16} \\ &= 9 \quad 144_{10} \\ &\quad \text{byte1} \quad \text{byte 2} \end{aligned}$$

Setting the Sweep Rate

The AFG can sweep (linearly) from 0.01 Hz to 15 MHz. The fastest the AFG can sweep is 187 μS per frequency point with amplitude correction disabled, or 250 μS per point with amplitude correction enabled. Up to 25.5 mS additional delay per frequency point is possible in 100 μS increments (tic count).

Format The sweep rate is a **six-byte** parameter (Table C-2). The first four bytes contain the frequency increment. The fifth byte (tic count) controls how often the increment occurs. An increment occurs every 187 or 250 μS plus (tic count - 1) delays of 100 μS. The sixth byte is always 0 for a linear sweep.

Setting the Burst Count

The burst count specifies the number of waveform cycles to occur when the "burst" output mode is set. For all other output modes the burst count need not be set since an infinite number of cycles are output.

Format The burst count is a **two-byte** parameter (Table C-2). The count is converted to the form:

$$A_1 * 2^8 + A_2$$

For example, a 1,000 cycle burst is sent as:

$$\begin{aligned} 1,000 \text{ cycles} &= 03 \quad E8_{16} \\ &= 3 \quad 232_{10} \\ &\quad \text{byte 1} \quad \text{byte 2} \end{aligned}$$

Internal and External Triggered Bursts

The AFG burst modes are "internal" and "external". An internal burst of the specified burst count occurs when an internal trigger is received. An external burst of the specified burst count occurs when the signal on the "Aux In" BNC port is low. The burst mode is set with the output mode bytes covered in the next section. The "Example Programs" section demonstrates how to set up internal and external triggered bursts.

The AFG Output Mode

The AFG output mode consists of the following:

- Setting the "Aux In" BNC port function.
- Setting the "Aux Out" (marker) BNC port condition.
- Setting the time base (reference) source.
- Setting the "Aux In" and "Aux Out" sense levels.
- Setting the waveform length
- Setting the output mode
- Selecting the Frequency Register
- Selecting the RAM to output to the DAC.

Format The output mode is a **two byte** parameter. The format of the bytes is shown in Table C-2.

Aux Out/Aux In Level Bit 7 of byte 1 sets the polarity of the signal at the "Aux Out" (marker) BNC for the selected condition (bits 2 and 1). Bit 6 sets the polarity which activates the selected event (bit 0). For bits 7 and 6, normal ("0") sets an "active high" (positive transition) signal. Inverted ("1") sets an "active low" (negative transition) signal.

Clock Source The clock source bits (5 - 3) specify and control the AFG's reference clock (oscillator). The sources:

External - external source is applied to the "Aux In" BNC. Bit 0 must be set to "1".

Gated - allows the reference to be gated by a gate signal applied to the "Aux In" BNC. Bit 0 must be set to "1". A TTL "high" level stops the output. A TTL "low" level enables the output.

Burst - allows a counted burst (see "Setting the Burst Count") of cycles to occur when an internal or external trigger is received. This source is set together with either the "internal burst" or "external burst" mode (bits 4 - 2 of byte 2).

Internal - this source is the AFG's internal 42.94 MHz oscillator.

Input Data Register - allows waveform data to be downloaded to the AFG's DAC through the Input Data Register (base + 0C16). This source is set for the "direct DAC access" mode (bits 4 - 2 of byte 2).

No clock - setting this condition stops the internal reference clock or the acceptance of an external clock.

Aux Out The AFG's "Aux Out" BNC port outputs the following TTL level signals. The polarity of the signal is set by bit 7.

42 MHz osc. - this square wave signal is the AFG's internal 42.94 MHz reference clock. Bit 7 (polarity) is ignored.

Pulse/cycle - this signal is a marker pulse that is output at the beginning of each waveform output segment (1 pulse per 4k RAM bank). Bit 7 (polarity) is ignored.

Zero Cross - this signal is a marker pulse that is output each time the output waveform crosses through 0V (positive or negative transition). Bit 7 (polarity) is ignored.

Sweep sync - this signal is a marker pulse that is output at the beginning of each sweep. Bit 7 (polarity) can be used.

Aux In The "Aux In" BNC connector accepts TTL level signals which activate the following events based on the setting of bit 0. The bit is set based on the clock source setting and mode setting.

Setting bit 0 to "0" enables an external signal to select the output signal (wave hop), to select the output frequency (FSK), or to trigger a counted burst. The external signal's rising edge selects the waveform in RAM 0. Its falling edge selects the waveform in RAM 1. Similarly, its rising edge selects FSK frequency 1, and its falling edge selects FSK Frequency 2. The sensing of the signal for wave hopping, FSK, and external bursts can be inverted with bit 6.

Setting bit 0 to "1" enables an external reference clock to be applied to the port. Bit 0 is also set to "1" when the output is to be gated on and off.

RAM Bank The RAM Bank bits (bits 7 - 6 of byte 2) are used to select the waveform stored in the RAM bank when the output function is set (see "Setting the Output Function"). The output mode (bits 4 - 2) or RAM size (bits 1 - 0) may override the RAM bank bits.

Frequency Register The Frequency Register bit (bit 5) selects the output frequency stored in Frequency Register 1 or Frequency Register 2 (command opcodes 1 and 2).

Frequency Register 1 contains:

- fixed (continuous) frequency
- sweep start frequency
- first FSK frequency

Frequency Register 2 contains:

- fixed (continuous) frequency
- sweep stop frequency
- second FSK frequency

Output Mode The AFG output modes (bits 4 - 2) are:

Continuous (fixed) - output signal has a continuous (fixed) frequency. This is the default mode.

FSK - output mode is frequency-shift keying. Bit 0 of byte 1 must be set to "0" in order for a control signal (applied to the "Aux In" BNC) to shift frequencies. The rising edge of the control signal selects the frequency in Frequency Register 1. The falling edge of the control signal selects Frequency Register 2. When this mode is set, the Frequency Register bit (bit 5) is ignored.

Wave hop - output mode is wave hopping. Bit 0 of byte 1 must be set to "0" in order for a control signal (applied to the "Aux In" BNC) to select the output waveform. The rising edge of the control signal selects the waveform in RAM 0. The falling edge of the control signal selects the waveform in RAM 1. When this mode is set, the RAM bank bits (7 - 6) and the RAM size bits (1 - 0) are ignored.

Internal burst - this mode will output the number of cycles specified by the burst count (opcode 8) and stop. This mode is set together with the "burst" clock source (bits 5 - 3 of byte 1).

External burst - this mode will output the number of cycles specified by the burst count (opcode 8) each time an external trigger is received ("Aux In" BNC). This mode is set together with the "burst" clock source (bits 5 - 3 of byte 1).

Sweep - this mode sets a linear sweep from the (start) frequency in Frequency Register 1 to the (stop) frequency in Frequency Register 2. The RAM bank bits (7 - 6) select the waveform which is swept.

The Frequency Register (bit 5) and RAM size (bits 1 - 0) bits are ignored.

Direct DAC access - this mode sends waveform data to the DAC directly from the Input Data Register (base + 0C₁₆). When this mode is set, the RAM bank, Frequency Register, and RAM size bits are ignored. The frequency is set to DC, and the output remains at the given voltage until the Input Data Register is written to again. This mode is used to generate an arbitrary waveform whose frequency and size (number of points) is limited by a controller's speed and its amount of memory. This mode is set together with the "Input Data Register" clock source (bits 5 -3 of byte 1).

Fast Frequency change - this mode is used to continuously change the output signal frequency in order to make custom frequency profiles. The AFG outputs a new frequency each time four (frequency) bytes are written to the Parameter Register. This mode uses the 4k RAM size. The Frequency Register and RAM size bits are ignored.

RAM Size The RAM size (bits 1 - 0 of byte 2) specifies whether one, two, or four waveforms will comprise the output waveform. If the RAM size is 8k or 16k, the RAM bank (bits 7 - 6) is ignored.

Starting the Waveform When register-based programming, the waveform is output when byte 2 of the output mode is processed by the AFG. If the "external burst" mode is set, the waveform is output when an external trigger is received.

Querying the AFG AFG parameters are queried by sending command opcode 13 followed by a query opcode. The response to the query is returned to the Query Response Register (base + 08₁₆).

The AFG parameters that can be queried include:

- function
- frequency (registers 1 and 2)
- attenuation
- offset
- burst count
- sweep rate
- mode

Each time opcode 13 and a query opcode are sent one byte is returned. For example, to query the frequency, opcode 13 must be sent four times (with a different query code) to return the four bytes which set the frequency.

The “Program Timing and Execution” section shows the register access sequence used to query the parameters. Examples of querying parameters are contained in the “Example Programs” section. The query opcodes are listed in Table C-4.

AFG Soft Reset

Sending command opcode 14 sets all AFG parameters to their power-on values. This command does not reset the AFG hardware as does the (hardware) reset described in the “Program Timing and Execution” section and used in the example programs.

Aborting the Waveform

Sending command opcode 6 halts waveform generation by stopping the clock. No other AFG settings are affected.

Program Timing and Execution

When programming the AFG at the register level, the structure of the program will generally be as follows:

- Resetting the AFG
- Setting the function
- Setting the frequency
- Setting the attenuation and offset
- Setting the sweep rate
- Setting the burst count
- Setting the output mode

This section contains generalized flowcharts and comments for performing these and other procedures such as querying parameters. The flowcharts identify the registers used and the status bits monitored to ensure proper execution of the program.

AFG Reset Sequence

The AFG is reset as indicated in Figure C-2.

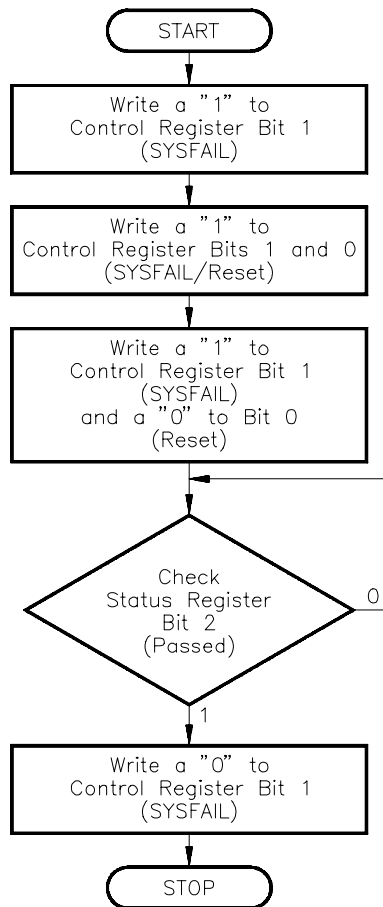


Figure C-2. Register Access to Reset the AFG

Comments

1. The registers used are:
 - Control Register (base + 04₁₆)
 - Status Register (base + 04₁₆)
2. Writing a "1" to bit 1 prevents the AFG from asserting the SYSFAIL line when it is reset. If SYSFAIL is enabled when a reset occurs, the AFG is taken off-line by the system Resource Manager.
3. Writing a "1" to bits 1 and 0 resets the AFG while preventing the AFG from asserting SYSFAIL . Writing a "0" to bit 0 turns the reset function off.
4. Bit 2 of the Status Register is monitored to determine when the reset is finished.
5. Writing a "0" to bit 1 re-enables SYSFAIL.

Configuring the AFG

The (suggested) register-based programming sequence and register access sequence is shown in Figure C-3.

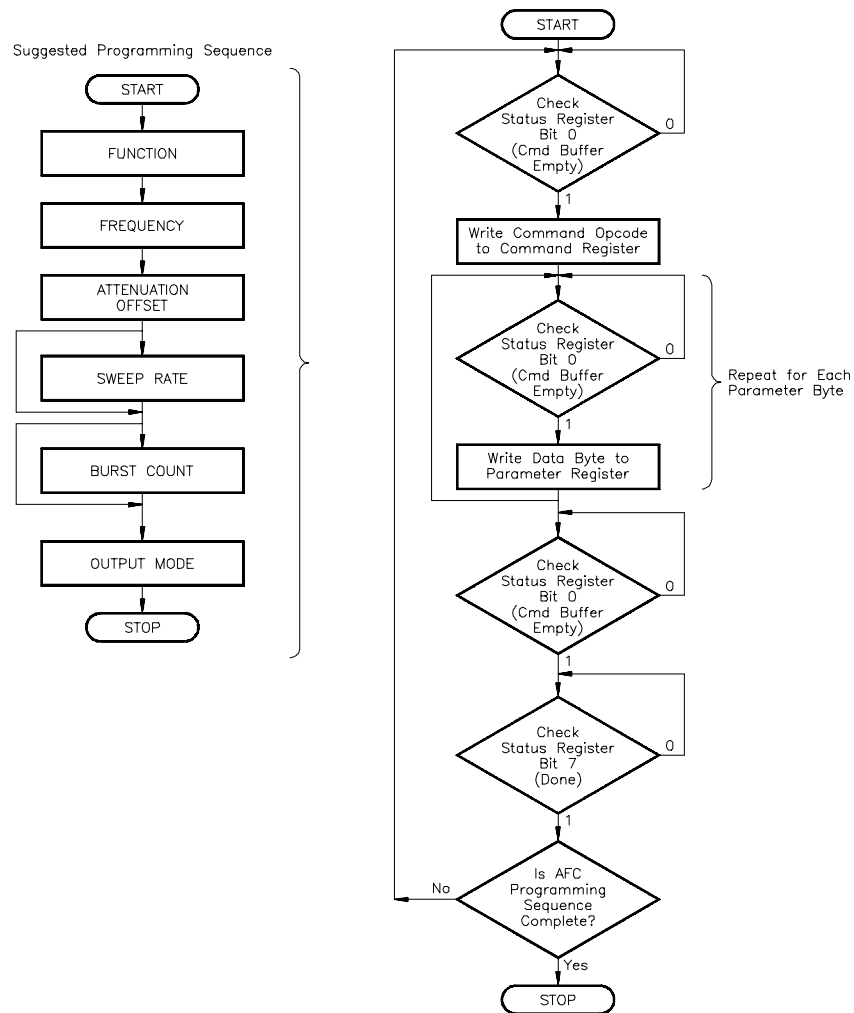


Figure C-3. Register Access to Configure the AFG

Comments

1. The registers used are:
 - Status Register (base + 04₁₆)
 - Command Register (base + 08₁₆)
 - Parameter Register (base + 0A₁₆)
2. Status Register bit 0 is monitored to determine when a command and parameters can be written to the Command and Parameter Registers.
3. Status Register bit 7 is monitored to determine when the AFG has finished processing the current command and parameter(s) and when the next command can be sent.
4. Repeated passes through the flowchart are made until each AFG parameter (e.g. function, frequency, etc.) has been set.

Querying AFG Parameters

The register access sequence for querying the AFG is shown in Figure C-4.

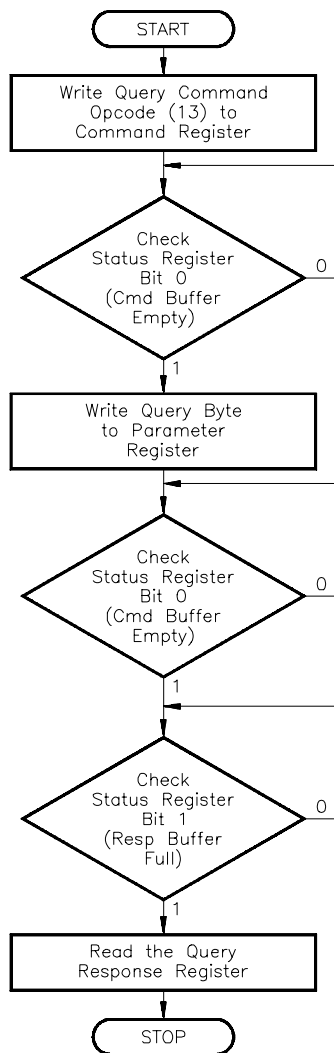


Figure C-4. Register Access to Query the AFG

Comments

1. The registers used are:
 - Status Register (base + 04₁₆)
 - Command Register (base + 08₁₆)
 - Parameter Register (base + 0A₁₆)
 - Query Response Register (base + 08₁₆)
2. Command opcode 13 is written to the Command Register.
3. Status Register bit 0 is monitored to determine when the query opcode can be written to the Parameter Register.
4. Status bit 1 is monitored to determine when the response to the query is in the Query Response Register.

Example Programs

The examples in this section program the E1340A AFG at the register level. The program listings are HP Instrument BASIC (IBASIC) programs, and are contained on the example programs disk HP P/N E1340-10035. C and QuickBASIC versions of these programs are on example programs disk HP P/N E1340-10036.

The examples in this section include:

- Generating a Sine Wave
- Multiple Waveforms
- Internally Triggering a Burst of Cycles
- Externally Triggering a Burst of Cycles
- Frequency-Shift Keying
- Waveform Hopping
- Sweeping
- Gating the Output
- Downloading an Arbitrary Waveform
- Fast Frequency Changes
- Sending Data Directly to the DAC
- Querying AFG Settings

IBASIC Subprograms

The IBASIC programs, which are based on the flowcharts found in the “Program Timing and Execution” section, pass opcodes and parameters to a series of subprograms. The subprograms are in a separate file (file name "SUBS") that is accessed by each IBASIC program. The subprograms are listed after the program "Sending Data Directly to the DAC".

Opcode/Parameter Quick Reference

Table C-3 lists the command opcodes and parameter values (where applicable) as they are used in the example programs.

Table C-3. Command Opcode and Parameter Quick Reference

Setting	Opcode	Parameter																																																
Function	5	<p>Function:</p> <table> <tr> <td>0 - Sine wave</td> <td>8 - 20 terms of falling ramp</td> </tr> <tr> <td>1 - Triangle wave</td> <td>9 - Rising ramp</td> </tr> <tr> <td>2 - Sin(x)/x for 8.25 radians</td> <td>10 - 20 terms of rising ramp</td> </tr> <tr> <td>3 - Haversine</td> <td>11 - White noise</td> </tr> <tr> <td>4 - Square wave</td> <td>12 - Modulated white noise</td> </tr> <tr> <td>5 - 10 terms of square wave</td> <td>13 - 3rd, 4th, 5th harm chord</td> </tr> <tr> <td>6 - 4 terms of square wave</td> <td>14 - 8 cycle rising sine wave</td> </tr> <tr> <td>7 - Falling ramp</td> <td>15 - Pos. 1/2 cycle sine wave</td> </tr> </table> <p>RAM:</p> <table> <tr> <td>0 - RAM 0</td> <td>2 - RAM 2</td> </tr> <tr> <td>1 - RAM 1</td> <td>3 - RAM 3</td> </tr> </table>	0 - Sine wave	8 - 20 terms of falling ramp	1 - Triangle wave	9 - Rising ramp	2 - Sin(x)/x for 8.25 radians	10 - 20 terms of rising ramp	3 - Haversine	11 - White noise	4 - Square wave	12 - Modulated white noise	5 - 10 terms of square wave	13 - 3rd, 4th, 5th harm chord	6 - 4 terms of square wave	14 - 8 cycle rising sine wave	7 - Falling ramp	15 - Pos. 1/2 cycle sine wave	0 - RAM 0	2 - RAM 2	1 - RAM 1	3 - RAM 3																												
0 - Sine wave	8 - 20 terms of falling ramp																																																	
1 - Triangle wave	9 - Rising ramp																																																	
2 - Sin(x)/x for 8.25 radians	10 - 20 terms of rising ramp																																																	
3 - Haversine	11 - White noise																																																	
4 - Square wave	12 - Modulated white noise																																																	
5 - 10 terms of square wave	13 - 3rd, 4th, 5th harm chord																																																	
6 - 4 terms of square wave	14 - 8 cycle rising sine wave																																																	
7 - Falling ramp	15 - Pos. 1/2 cycle sine wave																																																	
0 - RAM 0	2 - RAM 2																																																	
1 - RAM 1	3 - RAM 3																																																	
Frequency Register 1	1	four bytes																																																
Frequency Register 2	2	four bytes																																																
Attenuation	4	<p>Attenuation: 0 - 20 dB attenuation 1 - 0 dB attenuation</p> <p>Correction: 0 - enable amplitude correction 1 - disable amplitude correction</p>																																																
Offset	3	two bytes																																																
Sweep Rate	7	six bytes																																																
Burst Count	8	two bytes																																																
Clock Source/ Output Mode	9	<table> <tr> <td>Aux Out/Aux In level:</td> <td>Clock Source:</td> <td>Aux Out:</td> </tr> <tr> <td>0 - Normal</td> <td>0 - External</td> <td>0 - 42 MHz osc.</td> </tr> <tr> <td>1 - Inverted</td> <td>1 - Gated</td> <td>1 - Pulse/cycle</td> </tr> <tr> <td></td> <td>2 - Burst</td> <td>2 - Zero cross</td> </tr> <tr> <td></td> <td>3 - Internal</td> <td>3 - Sweep sync</td> </tr> <tr> <td></td> <td>4 - Input Data Register</td> <td></td> </tr> <tr> <td>RAM Bank:</td> <td>Frequency Register:</td> <td>RAM Size:</td> </tr> <tr> <td>0 - RAM 0</td> <td>1 - Register 1</td> <td>0 - 4k</td> </tr> <tr> <td>1 - RAM 1</td> <td>2 - Register 2</td> <td>1 - 8k</td> </tr> <tr> <td>2 - RAM 2</td> <td></td> <td>2 - 16k</td> </tr> <tr> <td>3 - RAM 3</td> <td></td> <td></td> </tr> <tr> <td>Mode:</td> <td></td> <td></td> </tr> <tr> <td>0 - Continuous (fixed)</td> <td>4 - Triggered burst</td> <td></td> </tr> <tr> <td>1 - FSK</td> <td>5 - Sweep</td> <td></td> </tr> <tr> <td>2 - Wave hop</td> <td>6 - Direct DAC access</td> <td></td> </tr> <tr> <td>3 - Single burst</td> <td>7 - Fast frequency change</td> <td></td> </tr> </table>	Aux Out/Aux In level:	Clock Source:	Aux Out:	0 - Normal	0 - External	0 - 42 MHz osc.	1 - Inverted	1 - Gated	1 - Pulse/cycle		2 - Burst	2 - Zero cross		3 - Internal	3 - Sweep sync		4 - Input Data Register		RAM Bank:	Frequency Register:	RAM Size:	0 - RAM 0	1 - Register 1	0 - 4k	1 - RAM 1	2 - Register 2	1 - 8k	2 - RAM 2		2 - 16k	3 - RAM 3			Mode:			0 - Continuous (fixed)	4 - Triggered burst		1 - FSK	5 - Sweep		2 - Wave hop	6 - Direct DAC access		3 - Single burst	7 - Fast frequency change	
Aux Out/Aux In level:	Clock Source:	Aux Out:																																																
0 - Normal	0 - External	0 - 42 MHz osc.																																																
1 - Inverted	1 - Gated	1 - Pulse/cycle																																																
	2 - Burst	2 - Zero cross																																																
	3 - Internal	3 - Sweep sync																																																
	4 - Input Data Register																																																	
RAM Bank:	Frequency Register:	RAM Size:																																																
0 - RAM 0	1 - Register 1	0 - 4k																																																
1 - RAM 1	2 - Register 2	1 - 8k																																																
2 - RAM 2		2 - 16k																																																
3 - RAM 3																																																		
Mode:																																																		
0 - Continuous (fixed)	4 - Triggered burst																																																	
1 - FSK	5 - Sweep																																																	
2 - Wave hop	6 - Direct DAC access																																																	
3 - Single burst	7 - Fast frequency change																																																	

Generating a Sine Wave

This program generates a 4 V_{p-p}, 10 kHz sine wave. The program selects the sine wave in the waveform EPROM and places it in RAM 0. The frequency, attenuation (amplitude), and the various mode parameters are set in the sequence indicated in Figure C-3.

HP IBASIC (RGSINE)

```
1 !RE-SAVE"RGSINE"
2 GET "SUBS",500,3 !place subprograms at line 500, continue w/line 3
3 !Program which demonstrates how to output a sine wave using register
4 !reads and writes.
5 !
10 !Compute the E1340 base address in E1300 A16 address space
20 COM Base_addr
30 Base_addr=DVAL("1FC000",16)+(80*64) !base address
40 !
50 !Reset the AFG
60 Afg_reset
70 !Set the function and load RAM
80 Function(0,0) !sine wave in RAM 0
90 !Set the frequency (frequency register 1)
100 Frequency1(1000.) !1 kHz
110 !Set amplitude, offset, impedance, correction
120 Amplitude(2,0,50,0) !2 Vpk, 0V, 50 ohms, correction enabled
130 !Set the Aux Out level, Aux In level, clock source, Aux Out signal,
140 !RAM bank, frequency register, and output mode.
150 Mode(0,0,3,1,0,0,0) !norm,norm,internal,pulse/cycle,bank 0,reg 1,cont
160 END
170 !
```

C and QuickBASIC Programs

The C program RGSINE.C is in directory "CPROG", and the QuickBASIC program RGSINE.BAS is in directory "QBPROG" on the example program disk HP P/N E1340-10036.

Multiple Waveforms

This program loads RAM 0 through RAM 3 from the waveform EPROM and outputs each waveform as part of a single waveform sequence.

HP IBASIC (MULTFUNC)

```
1 !RE-SAVE"MULTFUNC"
2 GET "SUBS",500,3 !place subprograms at line 500, continue w/line 3
3 !Program which demonstrates loading memory (RAM 0 through RAM 3)
4 !and then outputting the waveforms in a single sequence.
5 !
10 !Compute the E1340 base address in E1300 A16 address space
20 COM Base_addr
30 Base_addr=DVAL("1FC000",16)+(80*64) !base address
40 !
50 !Reset the AFG
60 Afg_reset
70 !Load RAM 0 - RAM 3 from the waveform EEPROM
80 Function(9,0) !rising ramp in RAM 0
90 Function(10,1) !first 20 terms of rising ramp in RAM 1
100 Function(7,2) !falling ramp in RAM 2
110 Function(8,3) !first 20 of falling ramp in RAM 3
120 !Set the frequency (frequency register 1)
130 Frequency1(1000.) !1 kHz (output frequency is 250 Hz)
140 !Set amplitude, offset, impedance, correction
150 Amplitude(5,0,50,0) !5 Vpk, 0V, 50 ohms, correction enabled
160 !Set the Aux Out level, Aux In level, clock source, Aux Out signal,
170 !RAM bank, frequency register, and output mode.
180 Mode(0,0,3,1,16,0,0) !norm,norm,internal,pulse/cycle,16k,reg 1,cont
190 END
200 !
```

Comments

1. To output waveforms from RAM 0 - RAM 3 in a single sequence, the RAM size must be set to 16k, which overrides the RAM bank setting. In line 180, "16" is passed to the "Mode" subprogram which sets RAM size to 16k (see "SELECT Ram_bank" in subprogram "Mode").
2. When register-based programming, if two waveform segments (RAM 1 and RAM 2) comprise the output waveform, the frequency is 1/2 of the programmed frequency. If four waveform segments (RAM 0 through RAM 3) are used, the output frequency is 1/4 of the programmed frequency. The actual frequency of the waveform in this program is 250 Hz.

C and QuickBASIC Programs

The C program MULTFUNC.C is in directory "CPROG", and the QuickBASIC program MULTFUNC.BAS is in directory "QBPROG" on the example program disk HP P/N E1340-10036.

Internally Triggering a Burst of Cycles

The following program outputs a burst of 50,000 waveform cycles when the AFG's internal trigger occurs.

HP IBASIC (INT_BRST)

```
1  !RE-SAVE"INT_BRST"
2  GET "SUBS",500,3 !place subprograms at line 500, continue w/line 3
3  !Program which demonstrates how to output a burst of waveform cycles
4  !when an internal trigger is received.
5  !
10 !Compute the E1340 base address in E1300 A16 address space
20 COM Base_addr
30 Base_addr=DVAL("1FC000",16)+(80*64) !base address
40 !
50 !Reset the AFG
60 Afg_reset
70 !Set function and load RAM
80 Function(0,0) !sine wave in RAM 0
90 !Set the frequency (frequency register 2)
100 Frequency2(10000.) !10 kHz
110 !Set amplitude, offset, impedance, correction
120 Amplitude(5,0,50,0) !5 Vpk, 0V, 50 ohms, correction enabled
130 !Set the burst count
140 Burst(5.0E+4) !50k cycles
150 !Set the Aux Out level, Aux In level, clock source, Aux Out signal,
160 !RAM bank, frequency register, and output mode.
170 Mode(0,0,2,1,0,1,3) !norm,norm,burst,pulse/cycle,bank 0,reg 2,inbrst
180 END
190 !
```

Comments

1. To output a counted burst, the clock source must be set to "Burst" and the output mode set to "Internal Burst". The burst count range is 1 to 65534. The reset value is undefined (register-based).
2. This program sets the output frequency from Frequency Register 2. Either Frequency Register 1 or Register 2 and be used to set a continuous (fixed) frequency.

C and QuickBASIC Programs

The C program INT_BRST.C is in directory "CPROG", and the QuickBASIC program INT_BRST.BAS is in directory "QBPROG" on the example program disk HP P/N E1340-10036.

Externally Triggering a Burst of Cycles

The following program outputs a burst of 50,000 waveform cycles when an external trigger is received.

HP IBASIC (EXT_BRST)

```
1  !RE-SAVE"EXT_BRST"
2  GET "SUBS",500,3 !place subprograms at line 500, continue w/line 3
3  !Program which demonstrates how to output a burst of waveform cycles
4  !when an external trigger is received.
5  !
10 !Compute the E1340 base address in E1300 A16 address space
20 COM Base_addr
30 Base_addr=DVAL("1FC000",16)+(80*64) !base address
40 !
50 !Reset the AFG
60 Afg_reset
70 !Set function and load RAM
80 Function(0,0) !sine wave in RAM 0
90 !Set the frequency (frequency register 1)
100 Frequency1(10000.) !10 kHz
110 !Set amplitude, offset, impedance, correction
120 Amplitude(5,0,50,0) !5 Vpk, 0V, 50 ohms, correction enabled
130 !Set the burst count
140 Burst(5.0E+4) !50k cycles
150 !Set the Aux Out level, Aux In level, clock source, Aux Out signal,
160 !RAM bank, frequency register, and output mode.
170 Mode(0,0,2,1,0,0,4) !norm,norm,burst,pulse/cycle,bank 0,reg 1,exbrst
180 END
190 !
```

Comments

1. To output a counted burst, the clock source must be set to "Burst" and the output mode set to "External Burst". The burst count range is 1 to 65534. The reset value is undefined (register-based).
2. The external burst occurs when the signal on the "Aux In" BNC port goes low (TTL levels). The "Aux In level" parameter can be used to invert the sensing of the "Aux In" port in order to start a burst when the "Aux In" port goes high.

C and QuickBASIC Programs

The C program EXT_BRST.C is in directory "CPROG", and the QuickBASIC program EXT_BRST.BAS is in directory "QBPROG" on the example program disk HP P/N E1340-10036.

Frequency-Shift Keying

The following program sets the AFG's frequency-shift keying mode. The frequency of the output signal switches between 1 kHz and 10 kHz depending on the level of the signal applied to the "Aux In" BNC port.

HP IBASIC (FSK)

```
1  !RE-SAVE"FSK"
2  GET "SUBS",500,3 !place subprograms at line 500, continue w/line 3
3  !Program which demonstrates frequency-shift keying (FSK).
4  !
10 !Compute the E1340 base address in E1300 A16 address space
20 COM Base_addr
30 Base_addr=DVAL("1FC000",16)+(80*64) !base address
40 !
50 !Reset the AFG
60 Afg_reset
70 !Set function and load RAM
80 Function(0,0) !sine wave in RAM 0
90 !Set the first FSK frequency (frequency register 1)
100 Frequency1(1000.) !1 kHz
110 !Set the second FSK frequency (frequency register 2)
120 Frequency2(1.0E+4)!10 kHz
130 !Set amplitude, offset, impedance, correction
140 Amplitude(5,0,50,0) !5 Vpk, 0V, 50 ohms, correction enabled
150 !Set the Aux Out level, Aux In level, clock source, Aux Out signal,
160 !RAM bank, frequency register, and output mode.
170 Mode(0,0,3,1,0,0,1) !norm,norm,internal,pulse/cycle,bank 0,reg 1,FSK
180 END
```

Comments

1. The first FSK frequency is set with Frequency Register 1. The second FSK frequency is set with Frequency Register 2. A control signal applied to the "Aux In" BNC port selects the current output frequency. A rising edge selects Frequency Register 1. A falling edge selects Frequency Register 2. The "Aux In level" parameter can be used to invert the sensing of the "Aux In" port in order to select a frequency with the other edge.
2. Since the output frequency is selected by the "Aux In" port, the Frequency Register parameter (bit 5 of byte 2 - Table C-2) is ignored. Sending any command to the AFG while the FSK mode is set aborts the mode.

C and QuickBASIC Programs

The C program FSK.C is in directory "CPROG", and the QuickBASIC program FSK.BAS is in directory "QBPROG" on the example program disk HP P/N E1340-10036.

Waveform Hopping

The following program selects between two waveforms based on the level of a control signal applied to the "Aux In" BNC.

HP IBASIC (WAVE_HOP)

```
1  !RE-SAVE"WAVE_HOP"
2  GET "SUBS",500,3 !place subprograms at line 500, continue w/line 3
3  !Program which demonstrates switching between waveforms in RAM 0
4  !and RAM 1.
5  !
10 !Compute the E1340 base address in E1300 A16 address space
20 COM Base_addr
30 Base_addr=DVAL("1FC000",16)+(80*64) !base address
40 !
50 !Reset the AFG
60 Afg_reset
70 !Load sine wave into RAM 0
80 Function(0,0)
90 !Load triangle wave into RAM 1
100 Function(1,1)
110 !Set the frequency (frequency register 1)
120 Frequency1(1000.) !1 kHz
130 !Set amplitude, offset, impedance, correction
140 Amplitude(5,0,50,1) !5 Vpk, 0V, 50 ohms, correction disabled
150 !Set the Aux Out level, Aux In level, clock source, Aux Out signal,
160 !RAM bank, frequency register, and output mode.
170 Mode(0,0,3,1,0,0,2) !norm,norm,internal,pulse/cycle,bank 0,reg 1,w_hop
180 END
190 !
```

Comments

1. The waveforms used in waveform hopping are stored in RAM 0 and RAM 1. The waveforms can be waveforms from the waveform EPROM or arbitrary waveforms loaded from the Input Data Register. The output waveform is selected by the "Aux In" BNC port. The rising edge of the control signal selects RAM 0. The falling edge selects RAM 1. The "Aux In level" parameter can be used to invert the sensing of the "Aux In" port in order to select a waveform with the other edge.
2. Sending any command to the AFG while the Wave hop mode is set aborts the mode.

3. When the "Wave hop" mode is set, the RAM bank and RAM size bits (Table C-2) are not used.

C and QuickBASIC Programs

The C program WAVE_HOP.C is in directory "CPROG", and the QuickBASIC program WAVE_HOP.BAS is in directory "QBPROG" on the example program disk HP P/N E1340-10036.

Sweeping

The following program generates a linear sweep from 1 kHz to 10 kHz in 5 seconds.

HP IBASIC (LINSWEEP)

```
1  !RE-SAVE"LINSWEEP"
2  GET "SUBS",500,3 !place subprograms at line 500, continue w/line 3
3  !Program which demonstrates how to set up a linear sweep.
4  !
10 !Compute the E1340 base address in E1300 A16 address space
20 COM Base_addr
30 Base_addr=DVAL("1FC000",16)+(80*64) !base address
40 !
50 !Reset the AFG
60 Afg_reset
70 !Set function and load RAM
80 Function(0,0) !sine wave in RAM 0
90 !Set amplitude, offset, impedance, correction
100 Amplitude(2,0,50,0) !2 Vpk, 0V, 50 ohms, correction enabled
110 !Set the start and stop frequency, sweep rate, and sweep type
120 Linsweep(1000,10000.,5)!start freq, stop freq, rate
130 !Set the Aux Out level, Aux In level, clock source, Aux Out signal,
140 !RAM bank, frequency register, and output mode.
150 Mode(0,0,3,3,0,0,5) !norm,norm,internal,sweep sync,bank 0,reg 1,sweep
160 END
170 !
```

C and QuickBASIC Programs

The C program LINSWEEP.C is in directory "CPROG", and the QuickBASIC program LINSWEEP.BAS is in directory "QBPROG" on the example program disk HP P/N E1340-10036.

Gating the Output

The following program sets the AFG's gating mode. The internal 42 MHz oscillator is gated on and off based on the level of a control signal applied to the 'Aux In' BNC.

HP IBASIC (GATE)

```
1 !RE-SAVE"GATE"
2 GET "SUBS",500,3 !place subprograms at line 500, continue w/line 3
3 !Program which demonstrates how to gate the output signal on and off.
4 !A "low" on the 'Aux In' BNC enables the AFG output. A "high" gates
5 !the output off.
6 !
10 !Compute the E1340 base address in E1300 A16 address space
20 COM Base_addr
30 Base_addr=DVAL("1FC000",16)+(80*64) !base address
40 !
50 !Reset the AFG
60 Afg_reset
70 !Set function and load RAM
80 Function(1,3) !triangle wave in RAM 3
90 !Set the frequency (frequency register 1)
100 Frequency1(5.0E+4) !50 kHz
110 !Set amplitude, offset, impedance, correction
120 Amplitude(2.5,0,50,0) !2.5 Vpk, 0V, 50 ohms, correction enabled
130 !Set the Aux Out level, Aux In level, clock source, Aux Out signal,
140 !RAM bank, frequency register, and output mode.
150 Mode(0,0,1,1,3,0,0) !norm,norm,gate,pulse/cycle,bank 3,reg 1,cont
160 END
170 !
```

Comments

1. Gating is enabled by setting the clock source to 'Gated' and the 'Aux In' bit (bit 0) to '1' (Table C-2). When the signal applied to the 'Aux In' BNC is low, the clock (reference oscillator) is allowed to run. When the signal is high, the clock is halted. The output remains at the last amplitude point clocked. The waveform resumes with the next point when the clock is enabled.

C and QuickBASIC Programs

The C program GATE.C is in directory "CPROG", and the QuickBASIC program GATE.BAS is in directory "QBPROG" on the example program disk HP P/N E1340-10036.

Downloading an Arbitrary Waveform

The following program downloads an arbitrary waveform (sine wave with a spike) to RAM 0 from the Input Data Register.

HP IBASIC (ARBWAVE)

```
1  !RE-SAVE"ARBWAVE"
2  !Program which demonstrates how to download an arbitrary waveform
3  !to the AFG.
4  !
10 !Compute the E1340 base address in E1300 A16 address space
20 COM Base_addr,REAL Waveform(0:4095)
30 Base_addr=DVAL("1FC000",16)+(80*64) !base address
40 !
50 !Compute the arbitrary waveform points
60 FOR I=0 TO 4095
70   Waveform(I)=SIN(2*PI*(I/4095))
80 NEXT I
90 Width=50
100 FOR J=1 TO (Width/2)
110   Waveform(J+1023)=Waveform(J+1023)+J*.04
120 NEXT J
130 FOR J=1 TO (Width/2)
140   Waveform(J+1023+Width/2)=Waveform(J+1023+Width/2)+1-(J*.04)
150 NEXT J
160 !
170
Scale_factor=2047/(MAX(MAX(Waveform(*),ABS(MIN(Waveform(*))))))
180 FOR I=0 TO 4095
190   Waveform(I)=(Waveform(I)*Scale_factor)+2047
200 NEXT I
210 !
220 !Reset the AFG
230 Afg_reset
240 !Set function and load RAM
250 Function(16,0) !arbitrary waveform in RAM 0
260 !Load RAM 0 from the Input Data Register
270 FOR I=0 TO 4095
280   WRITEIO -9826,Base_addr+12;Waveform(I)
290 NEXT I
300 !Set the frequency (frequency register 1)
310 Frequency1(10000) !10 kHz
320 !Set amplitude, offset, impedance, correction
330 Amplitude(5,0,50,0) !5 Vpk, 0V, 50 ohms, correction enabled
```

```

340 !Set the Aux Out level, Aux In level, clock source, Aux Out signal,
350 !RAM bank, frequency register, and output mode.
360 Mode(0,0,3,1,0,0,0) !norm,norm,internal,pulse/cycle,bank 0,reg 1,cont
370 END
380 !
500 !Subprograms used for register-based programming (file name 'SUBS').
501 !
509 SUB Function(Function,Ram)
519 Function: !Subprogram which selects the output waveform and places it
529           !in the specified RAM
539 COM Base_addr,Waveform(*)

```

Comments

1. Before running this program, type ' GET "SUBS",500 '. This adds the subprograms to program ARBWAVE starting at line 500. Add the array variable "Waveform(*)" to the COM statement found in selected subprograms.
2. Arbitrary waveforms are loaded into RAM from the Input Data Register (base + 0C₁₆). Arbitrary waveforms must be 4096 points long. Undefined points are output as random amplitudes. Since arbitrary waveforms are 4096 points, the frequency of the waveform is the repetition rate, rather than the sample rate of each point.
3. The 'Function' subprogram used with program ARBWAVE sets bit 6 of the 'output function and RAM' byte (Table C-2). This sets the output to 0V as the waveform data is downloaded into RAM from the Input Data Register. When bit 6 is not set (0), the output tracks the Input Data Register as the data is loaded.
4. All AFG clock sources and modes (Table C-2) are applicable for arbitrary waveforms.

C and QuickBASIC Programs

The C program ARBWAVE.C is in directory "CPROG", and the QuickBASIC program ARBWAVE.BAS is in directory "QBPROG" on the example program disk HP P/N E1340-10036.

Fast Frequency Changes

The following program demonstrates the 'fast frequency change' mode. The program outputs 20 frequencies with durations between one and two seconds. The frequencies and durations are computed with the program 'FREQ_GEN' and stored in the file Fdata. Program 'FASTFREQ' reads the file and writes the frequencies (4 bytes at a time) to Frequency Register 1.

HP IBASIC (FASTFREQ)

```
1  !RE-SAVE"FASTFREQ"
2  !Program which demonstrates how to use the AFG's fast frequency
3  !change mode. The program outputs 20 frequencies between 10 kHz
4  !and 20 kHz, for periods between 1 and 2 seconds. The frequencies are
5  !generated by the program 'FREQ_GEN'.
6  !
10 !Compute the E1340 base address in E1300 A16 address space
20 COM Base_addr
30 Base_addr=DVAL("1FC000",16)+(80*64) !base address
40 !
50 !Reset the AFG
60 Afg_reset
70 !Set the function and load RAM
80 Function(4,0) !square wave in RAM 0
90 !Set amplitude, offset, impedance, correction
100 Amplitude(5,0,50,1) !5 Vpk, 0V, 50 ohms, correction disabled
110 !Set the Aux Out level, Aux In level, clock source, Aux Out signal,
120 !RAM bank, frequency register, and output mode.
130 Mode(0,0,3,1,0,0,7) !norm,norm,internal,pulse/cycle,bank 0,xxx,ffreq
140 !
150 !Write frequency data to the AFG
160 ASSIGN @F TO "Fdata" !path to frequency file
170 DIM S$[100]
180 ON ERROR GOTO Done
190 LOOP
200 ENTER @F;S$ !enter frequency
210 IF POS(S$,"!") THEN S$=S$[POS(S$,"")+1] !set string not to include !
220 ENTER S$;Freq,Duration !enter frequency and duration
230 Frequency1(Freq) !write frequency byte
240 WAIT Duration
250 END LOOP
260 Check_done
270 Done:DISP "END OF FREQUENCY FILE REACHED"
280 END
```

```

290 !
      •
1059 SUB Frequency1(Freq1)
1069 Frequency1: !Subprogram which sets the output frequency using frequency
1079      !register 1.
1089 COM Base_addr
1099 Freq=(Freq1*100)
1109 C$=DVAL$(Freq,16)
1119 !
1129 !Load the frequency value
1139 ! Write_reg(8,1)
1149 Write_reg(10,IVAL(C$[1;2],16))
1159 Write_reg(10,IVAL(C$[3;2],16))
1169 Write_reg(10,IVAL(C$[5;2],16))
1179 Write_reg(10,IVAL(C$[7;2],16))
1189 ! Check_done

```

The following program generates the frequencies used by the program "FASTFREQ".

HP IBASIC (FREQ_GEN)

```

1  !RE-SAVE "FREQ_GEN"
2  !This program generates the frequencies used by example program 'FASTFREQ'.
3  !
10 DIM Line$[100]
20 CREATE ASCII "Fdata",10
30 ASSIGN @F TO "Fdata"
40 FOR I=1 TO 20
50   Freq=10000.+RND*10000.
60   Duration=1+RND
70   OUTPUT @F;VAL$(Freq)&","&VAL$(Duration)
80 NEXT I
90 !
100 ASSIGN @F TO *
110 ASSIGN @F TO "Fdata"
120 FOR I=1 TO 20
130  ENTER @F;Line$
140  PRInt Line$
150 NEXT I
160 END

```

Comments

1. Before running this program, type ' GET "SUBS",500 '. This adds the subprograms to program "FASTFREQ" starting at line 500.
2. Fast frequency changes are enabled by setting the 'Fast frequency change' mode. The frequency changes after the fourth frequency byte is written to the Frequency Register (1 or 2).
3. Once the mode is set, frequencies (bytes) are written to the Frequency Register without writing the command opcode (line 1139 is commented '!'), and without checking for the 'done' condition (line 1189 is commented '!'). The frequency will continue to be changed after every four bytes until the mode is changed. Amplitude correction is not available with this mode.
4. This mode uses the 4k RAM size. The Frequency Register and RAM size bits (Table C-2) are ignored.

C and QuickBASIC Programs

The C program FASTFREQ.C is in directory "CPROG", and the QuickBASIC program FASTFREQ.BAS is in directory "QBPROG" on the example program disk HP P/N E1340-10036.

Sending Data Directly to the DAC

The following program downloads data directly to the DAC from the Input Data Register (base +0C16). The data are a series of random numbers (amplitudes). Since the data is not stored in RAM, it is output as it is received by the DAC.

HP IBASIC (DACDRIVE)

```
1 !RE-SAVE"DACDRIVE"
2 GET "SUBS",500,3 !place subprograms at line 500, continue w/line 3
3 !Program which demonstrates how to download amplitude data directly
4 !to the DAC. The data sent is a series of random amplitudes.
5 !
10 !Compute the E1340 base address in E1300 A16 address space
20 COM Base_addr
30 Base_addr=DVAL("1FC000",16)+(80*64) !base address
40 !
50 !Compute data to be downloaded to the DAC.
60 INTEGER Voltage(1:10000)
70 FOR I=1 TO 10000
80 Voltage(I)=1024*RND+2048
90 NEXT I
100 !
110 !Reset the AFG
120 Afg_reset
130 !Set amplitude, offset, impedance, correction
140 Amplitude(10,0,50,0) !10 Vpk, 0V, 50 ohms, correction enabled
150 !Set the Aux Out level, Aux In level, clock source, Aux Out signal,
160 !RAM bank, frequency register, and output mode.
170 Mode(0,0,4,1,0,0,6) !norm,norm,in data reg,pulse/cycle,xxx,xxx,dirDAC
180 !
190 !Write data to the DAC
200 FOR I=1 TO 10000
210 WRITEIO -9826,Base_addr+12;Voltage(I)
220 NEXT I
230 END
240 !
```

Comments

1. In the 'Direct DAC access' mode, the frequency and length of the arbitrary waveform are limited by the controller (computer). The AFG frequency is set to DC. The attenuation and offset can be programmed as required. The output remains at the last level written to the DAC. The 'Direct DAC access' mode is useful for setting the output to a given level before a burst occurs.

2. In this mode, the Frequency Register, RAM bank, and RAM size bits (Table C-2) are not used.

C and QuickBASIC Programs

The C program DACDRIVE.C is in directory "CPROG", and the QuickBASIC program DACDRIVE.BAS is in directory "QBPROG" on the example program disk HP P/N E1340-10036.

Example Program Subprograms

The following file contains the subprograms used by the preceding example programs.

```
1  !Subprograms used for register-based programming (file name 'SUBS').
2  !
10  SUB Function(Function,Ram)
20  Function: !Subprogram which selects the output waveform and places it
30      !in the specified RAM.
40  COM Base_addr
50  SELECT Function !128 + waveform selection
60  CASE 0      !Sine wave
70  Eeprom=128
80  CASE 1      !Triangle wave
90  Eeprom=132
100 CASE 2      !Sin(x)/x
110 Eeprom=136
120 CASE 3      !Haversine
130 Eeprom=140
140 CASE 4      !Square wave
150 Eeprom=144
160 CASE 5      !First 10 terms of square wave
170 Eeprom=148
180 CASE 6      !First 4 terms of square wave
190 Eeprom=152
200 CASE 7      !Falling ramp
210 Eeprom=156
220 CASE 8      !First 20 terms of falling ramp
230 Eeprom=160
240 CASE 9      !Rising ramp
250 Eeprom=164
260 CASE 10     !First 20 terms of rising ramp
270 Eeprom=168
280 CASE 11     !White noise
290 Eeprom=172
300 CASE 12     !Modulated white noise
310 Eeprom=176
```



```

320 CASE 13      !3rd, 4th, 5th harmonic chord
330   Eeprom=180
340 CASE 14      !8 cycles linear rising sine wave
350   Eeprom=184
360 CASE 15      !Positive 1/2 cycle sine wave
370   Eeprom=188
380 CASE ELSE    !Input Data Register (arbitrary waveform)
390   Eeprom=64  !Turn off output as waveform is downloaded
400 END SELECT
410 !
420 SELECT Ram
430 CASE 1       !RAM 1
440   Eeprom=Eeprom+1
450 CASE 2       !RAM 2
460   Eeprom=Eeprom+2
470 CASE 3       !RAM 3
480   Eeprom=Eeprom+3
490 END SELECT
500 !
510 Write_reg(8,5)
520 Write_reg(10,Eeprom)
530 Check_done
540 SUBEND
550 !
560 SUB Frequency1(Freq1)
570 Frequency1: !Subprogram which sets the output frequency using frequency
580             !register 1.
590 COM Base_addr
600 Freq=(Freq1*100)
610 C$=DVAL$(Freq,16)
620 !
630 !Load the frequency value
640 Write_reg(8,1)
650 Write_reg(10,IVAL(C$[1;2],16))
660 Write_reg(10,IVAL(C$[3;2],16))
670 Write_reg(10,IVAL(C$[5;2],16))
680 Write_reg(10,IVAL(C$[7;2],16))
690 Check_done
700 SUBEND

```

```

710 !
720 SUB Frequency2(Freq2)
730 Frequency2: !Subprogram which loads frequency register 2 with the
740         !second FSK frequency or the stop sweep frequency.
750 COM Base_addr
760 Freq=(Freq2*100)
770 C$=DVAL$(Freq,16)
780 !
790 !Load the frequency value
800 Write_reg(8,2)
810 Write_reg(10,IVAL(C$[1;2],16))
820 Write_reg(10,IVAL(C$[3;2],16))
830 Write_reg(10,IVAL(C$[5;2],16))
840 Write_reg(10,IVAL(C$[7;2],16))
850 Check_done
860 SUBEND
870 !
880 SUB Amplitude(Ampl,Offset,Imp,Corr)
890 Amplitude: !Subprogram which sets the necessary attenuation for the
900         !specified amplitude. Also sets the DC offset and specifies
910         !the output impedance and sets frequency correction.
920 COM Base_addr
930 !
940 Atten=16384*(Corr<>0) !turn off amplitude correction if Corr = 1
950 SELECT Imp
960 CASE 50
970 IF Ampl>.51175 OR Offset>.51175 THEN
980     Atten=Atten+(Ampl/5.1175)*2047
990     C$=DVAL$(Atten+32768,16) !32768 = no attenuation
1000     Offset=(Offset/.0025)+2048
1010 ELSE
1020     Atten=Atten+(Ampl/.51175)*2047
1030     C$=DVAL$(Atten,16)
1040     Offset=((Offset/.0025)*10)+2048
1050 END IF
1060 CASE ELSE
1070 IF Ampl>1.0235 OR Offset>1.0235 THEN
1080     Atten=Atten+(Ampl/10.235)*2047
1090     C$=DVAL$(Atten+32768,16) !32768 = no attenuation
1100     Offset=(Offset/.005)+2048
1110 ELSE

```

```

1120   Atten=Atten+(Ampl/1.0235)*2047
1130   C$=DVAL$(Atten,16)
1140   Offset=((Offset/.005)*10)+2048
1150   END IF
1160 END SELECT
1170   !
1180 !Load the attenuation value
1190 Write_reg(8,4)
1200 Write_reg(10,IVAL(C$[5;2],16))
1210 Write_reg(10,IVAL(C$[7;2],16))
1220 Check_done
1230   !
1240 !Load the offset value
1250 C$=DVAL$(Offset,16)
1260 Write_reg(8,3)
1270 Write_reg(10,IVAL(C$[5;2],16))
1280 Write_reg(10,IVAL(C$[7;2],16))
1290 Check_done
1300 SUBEND
1310   !
1320 SUB Linsweep(Start_freq,Stop_freq,Rate)
1330 Linsweep: !Subprogram which sets up linear sweeps
1340 COM Base_addr
1350 !Set start and stop frequencies
1360 Frequency1(Start_freq)
1370 Frequency2(Stop_freq)
1380 !Calculate phase (frequency) increment
1390 Tic_cnt=256
1400 REPEAT
1410   Phase_inc=(Stop_freq-Start_freq)*(.00025+.0001*Tic_cnt)/Rate
1420   Tic_cnt=Tic_cnt-1
1430 UNTIL Phase_inc>=.01 AND Phase_inc<42949672.955 OR Tic_cnt=0
1440 C$=DVAL$(INT(100.*Phase_inc+.5),16)
1450   !
1460 !Load the phase increment, tic count, and sweep type value
1470 Write_reg(8,7)
1480 Write_reg(10,IVAL(C$[1;2],16)) !phase increment
1490 Write_reg(10,IVAL(C$[3;2],16)) !phase increment
1500 Write_reg(10,IVAL(C$[5;2],16)) !phase increment
1510 Write_reg(10,IVAL(C$[7;2],16)) !phase increment
1520 Write_reg(10,BINAND(Tic_cnt+1,255)) !tic count
1530 Write_reg(10,0) !linear sweep
1540 Check_done

```

```

1550 SUBEND
1560 !
1570 SUB Burst(Coun)
1580 Burst: !Subprogram which sets the burst (cycle) count
1590 COM Base_addr
1600 C$=DVAL$(Coun,16)
1610 !
1620 !Load burst count
1630 Write_reg(8,8)
1640 Write_reg(10,IVAL(C$[5;2],16)) !burst count
1650 Write_reg(10,IVAL(C$[7;2],16)) !burst count
1660 Check_done
1670 SUBEND
1680 !
1690 SUB Mode(Out_M,In_IV,Clock_source,Aux_out,Ram_bank,Freq_reg,Mode)
1700 Mode: !Subprogram which sets the output mode
1710 COM Base_addr
1720 Clock=0
1730 Out_mode=0
1740 SELECT Out_IV
1750 CASE 1          !Aux out signal inverted
1760   Clock=Clock+128
1770 END SELECT
1780 !
1790 SELECT In_IV
1800 CASE 1          !Invert signal in to Aux In
1810   Clock=Clock+64
1820 END SELECT
1830 !
1840 SELECT Clock_source
1850 CASE 0          !External
1860   Clock=Clock+1
1870 CASE 1          !Gated
1880   Clock=Clock+9
1890 CASE 2          !Burst
1900   Clock=Clock+16
1910 CASE 3          !Internal (default)
1920   Clock=Clock+24
1930 CASE 4          !Input Data Register
1940   Clock=Clock+32
1950 END SELECT

```

```

1960 !
1970 SELECT Aux_out
1980 CASE 1      !Pulse/cycle
1990   Clock=Clock+2
2000 CASE 2      !Zero cross
2010   Clock=Clock+4
2020 CASE 3      !Sweep sync
2030   Clock=Clock+6
2040 END SELECT
2050 !
2060 SELECT Ram_bank
2070 CASE 1      !RAM 1
2080   Out_mode=Out_mode+64
2090 CASE 2      !RAM 2
2100   Out_mode=Out_Mode+128
2110 CASE 3      !RAM 3
2120   Out_mode=Out_mode+192
2130 CASE 8      !RAM size = 8k, RAM bank not used
2140   Out_mode=Out_mode+1
2150 CASE 16     !RAM size = 16k, RAM bank not used
2160   Out_mode=Out_mode+2
2170 END SELECT
2180 !
2190 SELECT Freq_reg
2200 CASE 1      !Frequency Register 2
2210   Out_mode=Out_mode+32
2220 END SELECT
2230 !
2240 SELECT Mode
2250 CASE 1      !FSK
2260   Out_mode=Out_mode+4
2270 CASE 2      !Wave hop
2280   Out_mode=Out_mode+8
2290 CASE 3      !Single burst
2300   Out_mode=Out_mode+12
2310 CASE 4      !Triggered burst
2320   Out_mode=Out_mode+16
2330 CASE 5      !Sweep
2340   Out_mode=Out_mode+20
2350 CASE 6      !Direct dac access
2360   Out_mode=Out_mode+24
2370 CASE 7      !Fast frequency change
2380   Out_mode=Out_mode+28

```

```

2390 END SELECT
2400 !
2410 !Enable waveform output
2420 Write_reg(8,9)
2430 Write_reg(10,Clock)
2440 Write_reg(10,Out_mode)
2450 Check_done
2460 SUBEND
2470 !
2480 SUB Afg_reset
2490 Afg_reset: !Subprogram which resets the AFG
2500 COM Base_addr
2510 WRITEIO -9826,Base_addr+4;2 !inhibit sysfail during reset
2520 WRITEIO -9826,Base_addr+4;3 !reset AFG (sysfail inhibited)
2530 WRITEIO -9826,Base_addr+4;2 !turn reset off (sysfail inhibited)
2540 Check_pass
2550 WRITEIO -9826,Base_addr+4;0 !enable sysfail
2560 SUBEND
2570 !
2580 SUB Write_reg(Reg,Cp_data)
2590 Write_reg: !Subprogram which writes data to command and parameter
2600 !registers
2610 COM Base_addr
2620 Check_ready
2630 WRITEIO -9826,Base_addr+Reg;Cp_data
2640 SUBEND
2650 !
2660 SUB Check_pass
2670 Check_pass: !Monitors completion of reset
2680 REPEAT
2690 CALL Read_status(Pass_fail,Ready,Done)
2700 UNTIL Pass_fail
2710 SUBEND
2720 !
2730 SUB Check_done
2740 Check_done: !Monitors command completion
2750 REPEAT
2760 CALL Read_status(Pass_fail,Ready,Done)
2770 UNTIL Done
2780 SUBEND
2790 !
2800 SUB Check_ready

```

```
2810 Check_ready: !Monitors command and parameter execution
2820 REPEAT
2830 CALL Read_status(Pass_fail,Ready,Done)
2840 UNTIL Ready
2850 SUBEND
2860 !
2870 SUB Read_status(Pass_fail,Ready,Done)
2880 Read_status:!Subprogram which reads the arb's Status Register
2890 COM Base_addr
2900 Status=READIO(-9826,Base_addr+4)
2910 Ready=BIT(Status,0)
2920 Pass_fail=BIT(Status,2)
2930 Done=BIT(Status,7) AND Ready
2940 Qryrdy=BIT(Status,1) AND Done
2950 SUBEND
```

Querying AFG Parameters

This program is used to query the following AFG parameters:

- function
- frequency (Registers 1 and 2)
- attenuation
- offset
- burst count
- sweep rate
- mode

The query opcodes used are shown in Table C-4 below.

Table C-4. AFG Query Opcodes

Parameter	Query Opcode	Description
Function	60	Returns the byte used to set the function and select RAM.
Frequency Register 1	48 49 50 51	Returns the most significant byte (MSB) used to set the output frequency. Returns the second significant byte (2SB) used to set the output frequency. Returns the third significant byte (3SB) used to set the output frequency. Returns the least significant byte (LSB) used to set the output frequency.
Frequency Register 2	52 53 54 55	Returns the most significant byte (MSB) used to set the output frequency. Returns the second significant byte (2SB) used to set the output frequency. Returns the third significant byte (3SB) used to set the output frequency. Returns the least significant byte (LSB) used to set the output frequency.
Attenuation	58 59	Returns the most significant byte (MSB) used to set the attenuation. Returns the least significant byte (LSB) used to set the attenuation.
Offset	56 57	Returns the most significant byte (MSB) used to set the DC offset. Returns the least significant byte (LSB) used to set the DC offset.
Burst Count	17 18	Returns the most significant byte (MSB) used to set the burst count. Returns the least significant byte (LSB) used to set the burst count.
Sweep Rate	40 41 62 63 16	Returns the most significant byte (MSB) used to set the phase increment. Returns the second significant byte (2SB) used to set the phase increment. Returns the third significant byte (3SB) used to set the phase increment. Returns the least significant byte (LSB) used to set the phase increment. Returns the tic count (sent as a one byte parameter query).
Mode	61 23	Returns 'Byte 1' (Table C-2) of the clock source and output mode. Returns 'Byte 2' (Table C-2) of the clock source and output mode.

HP IBASIC (AFGQUERY)

```
1  !RE-SAVE"AFGQUERY"
2  !Program which queries the AFG parameters.
3  !
10 !Compute the E1340 base address in E1300 A16 address space
20 COM Base_addr
30 Base_addr=DVAL("1FC000",16)+(80*64) !base address
40 !
50 Parm1(60) !one byte parameter query
60 Parm2(58,59) !two byte parameter query
70 Parm4(48,49,50,51) !four byte parameter query
80 END
90 !
100 SUB Parm1(Byte)
110 Parm1: !Subprogram which queries one byte parameters
120 COM Base_addr
130 Write_reg(8,13)
140 Write_reg(10,Byte)
150 Queryrdy
160 Byte=READIO(-9826,Base_addr+8)
170 Byte=BINAND(Byte,255)
180 PRInt Byte
190 SUBEND
200 !
210 SUB Parm2(Byte1,Byte2)
220 Parm2: !Subprogram which queries two byte parameters
230 COM Base_addr
240 Write_reg(8,13)
250 Write_reg(10,Byte1)
260 Queryrdy
270 Byte1=READIO(-9826,Base_addr+8)
280 Byte1=BINAND(Byte1,255)*2^8
290 !
300 Write_reg(8,13)
310 Write_reg(10,Byte2)
320 Queryrdy
330 Byte2=READIO(-9826,Base_addr+8)
340 Byte2=BINAND(Byte2,255)
350 Value=Byte1+Byte2
360 PRInt Value
370 SUBEND
380 !
```

```

390 SUB Parm4(Byte1,Byte2,Byte3,Byte4)
400 Parm4: !Subprogram which queries four byte parameters
410     COM Base_addr
420     Write_reg(8,13)
430     Write_reg(10,Byte1)
440     Queryrdy
450     Byte1=READIO(-9826,Base_addr+8)
460     Byte1=BINAND(Byte1,255)*2^24
470     !
480     Write_reg(8,13)
490     Write_reg(10,Byte2)
500     Queryrdy
510     Byte2=READIO(-9826,Base_addr+8)
520     Byte2=BINAND(Byte2,255)*2^16
530     !
540     Write_reg(8,13)
550     Write_reg(10,Byte3)
560     Queryrdy
570     Byte3=READIO(-9826,Base_addr+8)
580     Byte3=BINAND(Byte3,255)*2^8
590     !
600     Write_reg(8,13)
610     Write_reg(10,Byte4)
620     Queryrdy
630     Byte4=READIO(-9826,Base_addr+8)
640     Byte4=BINAND(Byte4,255)
650     !
660     Value=(Byte1+Byte2+Byte3+Byte4)/100
670     PRInt Value
680     SUBEND
690     !
700 SUB Write_reg(Reg,Cp_data)
710 Write_reg: !Subprogram which writes data to command and parameter
720     !registers
730     COM Base_addr
740     Check_ready
750     WRITEIO -9826,Base_addr+Reg;Cp_data
760     SUBEND
770     !
780 SUB Check_ready
790 Check_ready: !Monitors command and parameter execution
800     REPEAT

```

```

810  CALL Read_status(Pass_fail,Ready,Done,Qryrdy)
820  UNTIL Ready
830  SUBEND
840  !
850  SUB Queryrdy
860  REPEAT
870  CALL Read_status(Pass_fail,Ready,Done,Qryrdy)
880  UNTIL Qryrdy
890  SUBEND
900  !
910  SUB Read_status(Pass_fail,Ready,Done,Qryrdy)
920  Read_status: !Subprogram which reads the arb's Status Register
930  COM Base_addr
940  Status=READIO(-9826,Base_addr+4)
950  Ready=BIT(Status,0)
960  Pass_fail=BIT(Status,2)
970  Done=BIT(Status,7) AND Ready
980  Qryrdy=BIT(Status,1) AND Done
990  SUBEND

```

Comments

1. The parameters shown in the program listing query the function (60), the attenuation (58,59), and the frequency in Frequency Register 1 (48,49,50,51). Other parameters can be queried by sending other opcodes to the subprograms. To query only one or two parameters, comment out (!) lines 50 through 70 as appropriate.
2. Query responses (bytes) are returned to the Query Response Register (base + 08₁₆). Each time command opcode 13 and a query opcode are sent, one byte is returned. Thus, for multiple byte parameters, command opcode 13 must be sent for each byte queried.

- *CLS, 188 - 189
- *DMC, 188 - 189
- *EMC, 188, 190
- *EMC?, 188, 190
- *ESE, 188, 190
- *ESE?, 188, 190
- *ESR?, 188, 190
- *GMC?, 188, 191
- *IDN?, 188, 191
- *LMC?, 188, 192
- *LRN?, 188, 192
- *OPC, 188, 192
- *OPC?, 188, 193
- *PMC, 188, 193
- *RCL, 188, 193
- *RMC, 188, 194
- *RST, 188, 194
- *SAV, 188, 194
- *SRE, 188, 195
- *SRE?, 188, 195
- *STB?, 188, 195
- *TST?, 188, 196
- *WAI, 188, 196

A

- A16 Address, 229 - 231
- Abbreviated SCPI Commands, 133
- ABORt Subsystem, 138
- Aborting Waveforms, 102, 138, 252
- Absolute Addressing, 232 - 233
- Addressing Registers, 229
- Addressing the AFG, 18
- AFG
 - addressing, 18
 - arm count, 99, 107, 139
 - arming, 95 - 108
 - arming sources, 107
 - assigning to a commander, 18
 - configuration sequence, 254
 - connectors, 214
 - description, 211

- drivers, downloading, 15 - 16
- frequency modes, 94, 107, 156 - 157
- frequency-shift keying, 83, 92, 156
- high speed operation, 109 - 130
- installation, 15
- logical address, 17
- operating specifications, 215 - 218
- output mode, 248, 250
- reset sequence, 253
- resetting and clearing, 37
- SCPI commands, 131 - 132
- self-test, 36
- soft reset, 252
- status, 203 - 210
- sweeping, 83
- triggering, 95, 186
- AFGQUERY Example Program, 281 - 283
- Amplitude
 - control, 177
 - correction, 246
 - effects on DAC codes, 129
 - effects on voltage list, 82
 - levels, selecting, 50
 - offset, setting, 246 - 247
- ARB_GEN Example Program, 61
- ARB_HOP Example Program, 70
- Arbitrary Block Program Data, 135
- Arbitrary Waveform
 - built-in, 73 - 74
 - damped sine wave, 76 - 77
 - description, 211
 - download mode, 243
 - downloading, 243, 266 - 267
 - executing several segments, 65 - 66
 - exponential charge/discharge, 78
 - flowchart, 55, 58
 - frequency, 244
 - generating, 55 - 82, 212
 - half-rectified sine wave, 80
 - hopping, 69 - 70, 263
 - sine wave with spikes, 79
- ARBWAVE Example Program, 266
- ARM
 - configuration, 95
 - count, 99, 107, 139

- source, 97, 107
- states, 95
- ARM Subsystem, 139 - 141
- ARM[:START]:LAYer2:COUNT, 140
- ARM[:START]:LAYer2:SLOPe, 140
- ARM[:START]:LAYer2:SOURce, 141
- ARM[:START][:LAYer[1]]:COUNT, 139
- Arming
 - AFG, 95 - 108
 - commands, 96, 139
 - sources, 107
- Assigning the AFG to a Commander, 18
- Attenuator
 - description, 214
 - setting, 245 - 246
 - turning off, 246
- Aux In
 - connector, 214
 - level, setting, 248
 - port, 249
- Aux Out
 - connector, 214
 - level, setting, 248
 - port, 249
- Available Marker Sources, 103

B

- Backplane
 - See* VXIbus Backplane
- Base Address, 229 - 231
- BASIC Language Programming, 19
- Block Diagram Description, 211 - 214
- Boolean Command Parameters, 134
- Built-In Arbitrary Waveforms, 73 - 74
- Burst Count, 99
 - format, 247
 - setting, 247
- BURST Example Program, 100

C

- C Language Example Programs
 - program structure, 25 - 26
 - system configuration, 25
- C Language Programming, 19, 25
 - transferring data, 62 - 63, 116
- CALibration Subsystem, 142 - 145
- CALibration:AC:BEGIn, 142
- CALibration:AC:POINt, 143
- CALibration:DATA:AC, 143
- CALibration:DATA[:DC], 143
- CALibration:STATe:AC, 145

- CALibration[:DC]:BEGIn, 144
- CALibration[:DC]:POINt, 144
- Certification, 9
- CHARGE Example Program, 78
- Checking for Errors, 38
- Clock Source, 248
- *CLS, 188 - 189
- Command Descriptions and Formats, 239
- Command Register, 238
- Commander, E1445A AFG, 18
- Commands
 - arming, 96, 139
 - coupled, 20 - 21, 136, 153, 222
 - initiating the waveform, 96, 146
 - linking multiple, 21, 137
 - marker, 103
 - opcodes, 240 - 242, 256 - 257
 - optional SCPI, 20, 134 - 135
 - SCPI command reference, 137
 - SCPI structure, 19 - 20
 - types of, 132
 - un-coupled, 222
- Comment sheet, reader, 13
- Common (*) Command
 - *CLS, 188 - 189
 - *DMC, 188 - 189
 - *EMC, 188, 190
 - *EMC?, 188, 190
 - *ESE, 188, 190
 - *ESE?, 188, 190
 - *ESR?, 188, 190
 - *GMC?, 188, 191
 - *IDN?, 188, 191
 - *LMC?, 188, 192
 - *LRN?, 188, 192
 - *OPC, 188, 192
 - *OPC?, 188, 193
 - *PMC, 188, 193
 - *RCL, 188, 193
 - *RMC, 188, 194
 - *RST, 188, 194
 - *SAV, 188, 194
 - *SRE, 188, 195
 - *SRE?, 188, 195
 - *STB?, 188, 195
 - *TST?, 188, 196
 - *WAI, 188, 196
 - format, 132
 - list of, 188
- Computer Configurations, 18, 231 - 232
- Condition Register, 182, 205
 - reading, 205
- Configuring the AFG, 254

Conformance Information, SCPI, 200 - 201
Conformity, declaration, 11
Connectors, 214
Control Register, 237
Coupled Commands
 list of, 222
Coupled Commands, executing, 21
Coupling Commands, 20, 136, 153
Cycle count, waveform, 99, 107

D

DAC Data, 109
 amplitude effects on codes, 129
 codes, 109
 downloading into, 120, 126
 generating waveforms, 109
 incorrect codes, 129
 output, 212
 sending data to, 271
 source, 149
 transferring, 109, 113
DACBLOK Example Program, 115
DACDRIVE Example Program, 271
Damped Sine Wave, generating, 76 - 77
Data
 DAC, 109
 definite length block, 113
 swapping data bytes, 119
 transferring as DAC codes, 109, 113
 transferring in C, 62 - 63, 116
 transferring in QuickBASIC, 63 - 64, 118
 transferring using definite length blocks, 113
 transferring using VXIbus backplane, 120
 waveform, 150 - 151
DC Voltages, generating, 43
DCVOLTS Example Program, 43
Declaration of Conformity, 11
Definite Length Block
 See Definite Length Block Data
Definite Length Block Data, 113
 byte size, 113
 format, 113
Device Type Register, 235
DIAG:PEEK?, 233
DIAG:POKE, 233
Discrete Command Parameters, 134
*DMC, 188 - 189
Documentation history, 10
Downloading
 AFG device drivers, 15 - 16
 arbitrary waveforms, 243, 266 - 267
 data using VXIbus backplane, 120

 into DAC, 120, 126
 segment data, 120, 126
 waveform segments in memory, 120 - 122
Drivers, 15 - 16

E

Embedded Computer Programming, 232
*EMC, 188, 190
*EMC?, 188, 190
Enable Register, 206
 presetting, 209
Enabling the Gate, 108, 171
End-Of-Line (EOL) Terminator
 suppressing, 24
EPROM Waveforms, 82, 150
Error
 messages, 185, 224 - 228
 numbers, 224 - 226
 queue, 185
ERRORCHK Example Program, 39
Errors
 checking for, 38
 setting conflict error messages, 227 - 228
*ESE, 188, 190
*ESE?, 188, 190
*ESR?, 188, 190
Event Register, 182 - 183, 206
Example Programs
 addressing the AFG, 18
 AFGQUERY, 281 - 283
 ARB_GEN, 61
 ARB_HOP, 70
 ARBWAVE, 266
 BURST, 100
 CHARGE, 78
 DACBLOK, 115
 DACDRIVE, 271
 DCVOLTS, 43
 ERRORCHK, 39
 EXT_ARM, 98
 EXT_BRST, 261
 FREQ_GEN, 269 - 270
 FSK, 262
 FSK1, 93
 GATE, 265
 GATE_SIG, 101
 gating the output, 265
 IBASIC program structure, 22
 INT_BRST, 260
 introductory, 36
 LINSWEEP, 264
 LRN, 38

- MARK_OUT, 105
- MULSEG, 67
- MULTFUNC, 259
- multiple waveforms, 259
- OUTPUNIT, 51
- RAMPWAVE, 48
- register-based, 256
- RGBSINE, 258
- ROM_DOWN, 74
- RSTCLS, 37
- RSTSINE, 40
- SIN_D, 76
- SIN_R, 81
- SINEWAVE, 46
- SLFTST, 36
- SMPLSWP1, 86
- SMPLSWP2, 88
- SPIKES, 79
- subprograms, 272
- SWP_PVST, 90
- UNS_DAT, 110
- VXIDOWN, 123
- VXISRCE, 127
- WAVE_HOP, 263
- Executing Several Waveform Segments, 65 - 66
- Exponential Charge/Discharge Waveform, 78
- EXT_ARM Example Program, 98
- EXT_BRST Example Program, 261
- External Computer Programming, 233
- External Triggered Bursts, 247, 261

F

- Fast Frequency Changes, 268 - 270
- FASTFREQ Program Example, 268
- Flowchart
 - arbitrary waveforms, 55, 58
 - standard waveforms, 41 - 42, 84
 - sweeping, 83
- FREQ_GEN Example Program, 269 - 270
- Frequency
 - arbitrary waveform, 244
 - changes, fast, 268 - 270
 - format, 244
 - generators, description, 213
 - modes, 94, 107, 156 - 157
 - multiple segment waveforms, 245
 - non-swept waveform, 155
 - parameters, 244
 - points, 94
 - register, 250
 - setting signal, 244
 - shift keying, 83, 92, 156, 262

- start and span, 88, 157
- start and stop, 85, 158
- sweeping, 85
- Frequency-Shift Keying, 83 - 94
 - using the FSK port, 92
- FSK Example Program, 262
- FSK1 Example Program, 93
- FSweeping, 83 - 94

G

- GATE Example Program, 265
- GATE_SIG Example Program, 101
- Gating
 - enabling, 108, 171
 - output, 265
 - waveforms, 101, 107 - 108, 171
- Generating
 - arbitrary waveforms, 55 - 82, 212
 - built-in arbitrary waveforms, 73 - 74
 - damped sine wave, 76 - 77
 - DC voltages, 43
 - exponential charge/discharge wave, 78
 - half-rectified sine wave, 80
 - ramp waves, 47
 - several waveform segments, 65 - 66
 - simple arbitrary waveforms, 59 - 60
 - sine wave, 40, 258
 - sine wave with spikes, 79
 - sine waves, 45
 - square waves, 45
 - standard waveforms, 41 - 54
 - triangle waves, 45
 - waveforms, 212
 - waveforms using DAC data, 109
 - zero crossing marker pulses, 104
- Getting Started, 15 - 40
- *GMC?, 188, 191

H

- Half-Rectified Sine Wave, generating, 80
- High Speed Operation, 109 - 130
- HP IBASIC Example Programs
 - addressing the AFG, 18
 - AFGQUERY, 281 - 283
 - ARB_GEN, 61
 - ARB_HOP, 70
 - ARBWAVE, 266
 - BURST, 100
 - CHARGE, 78
 - DACBLOK, 115
 - DACDRIVE, 271

DCVOLTS, 43
 declaring IBASIC variables, 24
 ERRORCHK, 39
 EXT_ARM, 98
 EXT_BRST, 261
 FREQ_GEN, 269 - 270
 FSK, 262
 FSK1, 93
 GATE, 265
 GATE_SIG, 101
 getting, 24
 INT_BRST, 260
 LINSWEEP, 264
 LRN, 38
 MARK_OUT, 105
 MULSEG, 67
 MULTFUNC, 259
 OUTPUNIT, 51
 program structure, 22
 RAMPWAVE, 48
 register-based, 256
 RGBSINE, 258
 ROM_DOWN, 74
 RSTCLS, 37
 RSTSINE, 40
 SIN_D, 76
 SIN_R, 81
 SINEWAVE, 46
 SLFTST, 36
 SMPLSWP1, 86
 SMPLSWP2, 88
 SPIKES, 79
 subprograms, 272
 SWP_PVST, 90
 system configuration, 22
 UNS_DAT, 110
 VXIDOWN, 123
 VXISRCE, 127
 WAVE_HOP, 263

HP-IB

- primary address, 18
- secondary address, 18

I

IBASIC Array Variables, 24
 IBASIC Program Example
 FASTFREQ, 268
 IBASIC Programming, 232 - 233
 absolute addressing, 232 - 233
 declaring variables in COM memory, 234
 select code 8, 232 - 233
 subprograms, 256

ID Register, 234
 *IDN?, 188, 191
 IEEE-488.2 Common Commands
 See Common (*) Command
 Impedance, output, 52, 147
 Implied Key words, 134 - 135
 INITiate Subsystem, 146
 INITiate[:IMMediate], 96, 146
 Initiating Waveforms, 96, 146
 Input Data Register, 238
 Installation, mainframe, 15
 Instrument Specifications, 215 - 217
 INT_BRST Example Program, 260
 Interface Select Code, 18
 Internal Triggered Bursts, 247, 260
 Introductory Programs, 36

L

Languages
 C Language, 25
 HP BASIC, 19
 QuickBASIC, 31 - 32
 SCPI, 19
Languages Used, 19
 Linear Sweep, generating, 264
 Linking Commands, 21, 137
 LINSWEEP Example Program, 264
 *LMC?, 188, 192
 Loading RAM, 243
 Loads, output, 52, 148
 Logical Address, 17
 LRN Example Program, 38
 *LRN?, 188, 192

M

Mainframe Installation, 15
MARK_OUT Example Program, 105
Marker
 commands, 103
 outputs, 95 - 108, 167
 pulses, generating zero crossing, 104
 signal polarity, 168
 sources, 103
Memory
 declaring IBASIC variables in, 234
 description of, 213
 downloading waveform into, 120 - 122
Microprocessor, purpose of, 214
MINimum and MAXimum Parameters
 in coupling groups, 136
 using, 53

MULSEG Example Program, 67
MULTFUNC Example Program, 259
Multiple
 segment waveforms, frequency, 245
 waveforms in single output sequence, 243, 259

N

Numeric Command Parameters, 134

O

Offset
 amplitude, setting, 246 - 247
 circuitry description, 214
 format, 247
 register, 233
Offset Voltage, 177
*OPC, 188, 192
*OPC?, 188, 193
Opcode/Parameter Quick Reference, 256 - 257
Operating Specifications, 215 - 218
Operation Status Group, 205
 condition register, 205
 enable register, 206
 event register, 206
 transition filter, 205
Operation Status Register, 181
Operation, high speed, 109 - 130
Optional Keywords, 134 - 135
Oscillator Sources, reference, 52, 82, 94, 107, 170 - 172
OUTPUNIT Example Program, 51
Output
 amplifier description, 214
 attenuation, setting, 245 - 246
 circuitry description, 214
 DAC, description, 212
 function, setting, 243
 gating, 265
 impedance, 52, 147
 loads, 52, 148
 marker, 95 - 108, 167
 mode, 248, 250
 offset voltage, 180
 signal, 243
 units, 52
 units, selecting, 50
 voltage points, 163 - 164
 waveform, 56
OUTPut Subsystem, 147 - 148
OUTPut:IMPedance, 147
OUTPut:LOAD, 148

P

Parameter Register, 238
Parameters
 arbitrary block program data, 135
 boolean, 134
 discrete, 134
 MINimum and MAXimum, 53, 136
 numeric, 134
 optional, 134 - 135
 query settings, 136
 querying, 251 - 252, 255, 280
 signal frequency, 244
*PMC, 188, 193
Polarity
 marker signal, 168
 ramp waveform, 169
Power-on Configuration, 37
Preparation for Use, 15
Presetting
 enable register, 209
 transition filter, 209
Primary HP-IB Address, 18
Program Example
 FASTFREQ, 268
Program Structure
 C language, 25 - 26
 HP IBASIC language, 22
 QuickBASIC language, 31 - 32
Program Timing and Execution, 252
Programming Languages Used, 19
Programs, register-based examples, 256

Q

Query
 AFG parameters, 251 - 252, 255, 280
 condition register, 182
 error queue, 185
 event register, 183
 operation event register, 183
 parameter settings, 136
 SCPI revision number, 185
 segment sequence, 166
 segment sequence length, 165 - 166
 segment sequence names, 165
 size of waveform segment, 162
 voltage point list, 164
 waveform segment names, 161
Query Response Register, 237
Questionable Signal Status Register, 181
QuickBASIC Language Example Programs

- program structure, 31 - 32
- system configuration, 31
- QuickBASIC Language Programming, 31 - 32
- transferring data, 63 - 64, 118

R

RAM

- bank, 249
- loading, 243
- size, 251

Ramp Waves, generating, 47

RAMPWAVE Example Program, 48

*RCL, 188, 193

READ registers, 234 - 237

- device type register, 235
- ID register, 234
- query response register, 237
- status register, 236

Reader comment sheet, 13

Reading

- condition register, 205
- standard event status register, 207
- status byte register, 209

Reference Oscillator

- description, 213
- sources, 52, 82, 94, 107, 170 - 172

Register-based Programming, 229 - 284

- aborting waveforms, 252
- AFG configuration sequence, 254
- AFG Output Mode, 248, 250
- AFG reset sequence, 253
- amplitude offset, 246
- arbitrary waveform download mode, 243
- attenuation, 245 - 246
- Aux In level, setting, 248
- Aux In port, 249
- Aux Out level, setting, 248
- Aux Out port, 249
- base address, 229 - 231
- burst count, setting, 247
- clock source, setting, 248
- command descriptions and formats, 239
- command opcodes, 240 - 242, 256 - 257
- computer configurations, 231 - 232
- DIAG:PEEK?/DIAG:POKE, 233
- downloading arbitrary waveforms, 266 - 267
- embedded computers, 232
- example programs, 256
- external computers, 233
- external triggered bursts, 247, 261
- fast frequency changes, 268 - 270
- frequency format, 244

- frequency register, 250
- frequency-shift keying, 262
- gating the output, 265
- generating a sine wave, 258
- guidelines, 234
- IBASIC, 232 - 233
- IBASIC subprograms, 256
- internal triggered bursts, 247, 260
- loading RAM, 243
- multiple waveforms, 243, 259
- offset format, 247
- output attenuation, setting, 245 - 246
- output function, setting, 243
- querying parameters, 251 - 252, 255, 280
- RAM bank, 249
- RAM size, 251
- register number, 233
- register offset, 233
- sending data to the DAC, 271
- setting amplitude offset, 247
- signal frequency, setting, 244
- sine wave correction, 246
- soft reset, 252
- starting the waveform, 251
- sweep rate, setting, 247
- sweeping, 264
- throughput speed, 232
- timing and execution, 252
- turning attenuation off, 246
- VXI:READ?/VXI:WRITE, 233
- waveform hopping, 263

Registers

- addressing, 229
- command, 238
- condition, 182, 205
- control, 237
- descriptions, 234
- device type, 235
- enable, 206
- event, 182 - 183, 206
- frequency, 250
- ID, 234
- input data, 238
- number, 233
- offset, 233
- operation status, 181
- parameter, 238
- query response, 237
- questionable signal status, 181
- READ, 234 - 237
- service request enable, 209
- standard event status, 207
- standard event status enable, 208

- status, 203 - 210, 236
- status byte, 208
- WRITE, 237 - 238
- Repetition Count, 107
- Reset Configuration, 37
- Resetting
 - AFG, 253
- Resetting and Clearing, AFG, 37
- RGBSINE Example Program, 258
- *RMC, 188, 194
- ROM_DOWN Example Program, 74
- *RST, 188, 194
- RSTCLS Example Program, 37
- RSTSINE Example Program, 40

S

- Safety warnings, 10
- Sample Programs, 76
- *SAV, 188, 194
- SCPI Command
 - abbreviated, 133
 - ABORt subsystem, 138
 - arbitrary block program data, 135
 - ARM subsystem, 139 - 141
 - boolean parameters, 134
 - CALibration subsystem, 142 - 145
 - confirmed commands, 200
 - conformance information, 200 - 201
 - coupling, 20 - 21, 136, 153
 - discrete parameters, 134
 - execution of, 136
 - format, 133
 - format used, 20
 - INITiate subsystem, 146
 - linking multiple, 21, 137
 - list of, 131 - 132
 - non-confirmed commands, 201
 - numeric parameters, 134
 - optional, 20, 134 - 135
 - OUTPut subsystem, 147 - 148
 - query parameter settings, 136
 - reference, 137
 - separator, 133
 - SOURce:ARBitrary subsystem, 149 - 154
 - SOURce:FREQuency subsystem, 153 - 158
 - SOURce:FUNcTION subsystem, 159 - 160
 - SOURce:LIST subsystem, 161 - 166
 - SOURce:MARKer subsystem, 167 - 168
 - SOURce:RAMP subsystem, 169
 - SOURce:ROSCillator subsystem, 170 - 173
 - SOURce:SWEep subsystem, 173 - 176
 - SOURce:VOLTage subsystem, 177 - 181

- STATus subsystem, 181 - 184
- structure, 19 - 20
- SYSTem subsystem, 185
- TRIGger subsystem, 186 - 187
- SCPI Conformance Information, 200 - 201
- SCPI Programming, 19
- Secondary HP-IB Address, 18
- Segment Data, downloading, 120, 126
- Select Code 8, 232 - 233
- Selecting
 - amplitude levels, 50
 - output units, 50
- Self-test, AFG, 36
- Sending Data to DAC, 271
- Separator, SCPI commands, 133
- Service Request Enable Register, 209
- Setting
 - amplitude offset, 247
 - amplitude offset, setting, 246
 - arm count, 99, 107, 139
 - arm source, 97, 107
 - Aux In/Aux Out levels, 248
 - burst count, 247
 - clock source, 248
 - output attenuation, 245 - 246
 - output function, 243
 - signal frequency, 244
 - sweep rate, 247
- Signal
 - frequency, setting, 244
 - output, 243
- Signal Output Connector, 214
- Simple Arbitrary Waveforms, 59 - 60
- SIN_D Example Program, 76
- SIN_R Example Program, 81
- Sine Wave
 - correction, 246
 - generating, 40, 45, 258
 - generating damped, 76 - 77
 - generating half-rectified, 80
 - generating spikes, 79
- SINEWAVE Example Program, 46
- SINusoid Waves
 - See* Sine Wave
- SLFTST Example Program, 36
- SMPLSWP1 Example Program, 86
- SMPLSWP2 Example Program, 88
- Soft Reset, 252
- SOURce:ARBitrary Subsystem, 149 - 154
- SOURce:ARBitrary:DAC:SOURce, 149
- SOURce:ARBitrary:DOWNload, 150 - 151
- SOURce:ARBitrary:DOWNload:COMPLete, 152 - 154
- SOURce:FREQuency Subsystem, 153 - 158

SOURce:FREQuency:CENTer, 155
 SOURce:FREQuency:FSKey, 156
 SOURce:FREQuency:MODE, 156 - 157
 SOURce:FREQuency:SPAN, 157
 SOURce:FREQuency:START, 158
 SOURce:FREQuency:STOP, 158
 SOURce:FREQuency[:CW]:FIXed], 155
 SOURce:FUNcTION Subsystem, 159 - 160
 SOURce:FUNcTION:USER, 160
 SOURce:FUNcTION[:SHApe], 159
 SOURce:LIST Subsystem, 161 - 166
 SOURce:LIST:SSEquence:CATalog?, 165
 SOURce:LIST:SSEquence:DEFine?, 165
 SOURce:LIST:SSEquence:SElect, 165
 SOURce:LIST:SSEquence:SEquence:SEGments?, 166
 SOURce:LIST:SSEquence:SEquence?, 166
 SOURce:LIST[:SEGment]:CATalog?, 161
 SOURce:LIST[:SEGment]:DEFine?, 162
 SOURce:LIST[:SEGment]:SElect, 162
 SOURce:LIST[:SEGment]:VOLtage, 163
 SOURce:LIST[:SEGment]:VOLtage:DAC, 164
 SOURce:LIST[:SEGment]:VOLtage:POINts?, 164
 SOURce:MARKer Subsystem, 167 - 168
 SOURce:MARKer:FEED, 167
 SOURce:MARKer:POLarity, 168
 SOURce:RAMP Subsystem, 169
 SOURce:RAMP:POLarity, 169
 SOURce:ROSCillator Subsystem, 170 - 173
 SOURce:ROSCillator:FREQuency:EXTernal, 170
 SOURce:ROSCillator:GATE:STATe, 171
 SOURce:ROSCillator:SOURce, 171 - 173
 SOURce:SWEep Subsystem, 173 - 176
 SOURce:SWEep:COUNt, 174
 SOURce:SWEep:POINts, 175
 SOURce:SWEep:TIME, 175 - 176
 SOURce:VOLtage Subsystem, 177 - 181
 SOURce:VOLtage[:LEVel][:IMMEDIATE]:OFFSet, 180 - 181
 SOURce:VOLtage[:LEVel][:IMMEDIATE][:AMPLitude], 177 - 178
 SOURce:VOLtage[:LEVel][:IMMEDIATE][:AMPLitude]: UNIT[:VOLtage], 179
 Span and Start Frequencies, 88, 157
 Specifications, 215 - 218
 Specifying a Sweep Time, 94, 175
 Spiked Sine Wave, generating, 79
 SPIKES Example Program, 79
 Square Wave, generating, 45
 *SRE, 188, 195
 *SRE?, 188, 195
 Standard Commands for Programmable Instruments
 See SCPI Command
 Standard Event Status Enable Register, 208
 Standard Event Status Group, 207
 standard event status enable register, 208
 standard event status register, 207
 using, 210
 Standard Event Status Register, 207
 reading, 207
 Standard Waveforms Flowchart, 41 - 42, 84
 Standard Waveforms, generating, 41 - 54
 Start and Span Frequencies, 88, 157
 Start and Stop Frequencies, 85, 158
 Starting the waveform, 251
 Status Bit Precedence, 236
 Status Byte Register, 208
 reading, 209
 Status Byte Status Group, 208
 service request enable register, 209
 status byte register, 208
 Status Register, 236
 Status Registers, 203 - 210
 STATus Subsystem, 181 - 184
 Status System Registers, 203
 STATus:OPERation:CONDition?, 182
 STATus:OPERation:ENABle, 182
 STATus:OPERation:NTRansition, 183
 STATus:OPERation:PTRansition, 184
 STATus:OPERation[:EVENT]?, 183
 STATus:PRESet, 184
 STATus:QUEStionable:CONDition?, 182
 STATus:QUEStionable:ENABle, 182
 STATus:QUEStionable:NTRansition, 183
 STATus:QUEStionable:PTRansition, 184
 STATus:QUEStionable[:EVENT]?, 183
 *STB?, 188, 195
 Stop and Start Frequencies, 85, 158
 Subprograms used by Example Program, 272
 Swapping Data Bytes, 119
 Sweep
 linear, 264
 points, 175
 points vs. time, 90
 rate, setting, 247
 time, specifying, 94, 175
 Sweeping
 flowchart, 83
 frequency, 85
 points vs. time, 90
 using start and span frequencies, 88, 157
 using start and stop frequencies, 85, 158
 SWP_PVST Example Program, 90
 SYSTem Subsystem, 185
 SYSTem:ERRor?, 185
 SYSTem:VERSion?, 185

T

Throughput Speed, 232
Transferring Data
 as DAC codes, 109, 113
 in C language, 62 - 63, 116
 in QuickBASIC language, 63 - 64, 118
 using definite length blocks, 113
 using VXIbus backplane, 120
Transition Filter, 205
 presetting, 209
Triangle Wave, generating, 45
TRIGger Subsystem, 186 - 187
TRIGger[:START]:COUNT, 186
TRIGger[:START]:SOURCE, 187
Triggering Configuration, 95
*TST?, 188, 196
Turning Attenuation Off, 246

U

Un-coupled Commands, 222
Units, output, 52
UNS_DAT Example Program, 110
Using the VXIbus Backplane, 120

V

Variables, declaring in IBASIC, 24
Voltage
 offset, 177
 output offset, 180
VXI:READ?, 233
VXI:WRITE, 233
VXIbus Backplane, using, 120
VXIDOWN Example Program, 123
VXISRCE Example Program, 127

W

*WAI, 188, 196
WARNINGS, 10
Warranty, 9
WAVE_HOP Example Program, 263
Waveform
 aborting, 102, 138, 252
 built-in, 73 - 74
 cycle count, 99, 107
 damped sine waves, 76 - 77
 data, 150 - 151
 downloading in memory, 120 - 122
 executing several segments, 65 - 66

 exponential charge/discharge waves, 78
 frequency, 244
 gating, 101, 107 - 108, 171
 generating, 212
 generating arbitrary waves, 55 - 82
 generating built-in arbitrary waves, 73 - 74
 generating DC voltages, 43
 generating ramp waves, 47
 generating simple arbitrary waves, 59 - 60
 generating sine waves, 45
 generating square waves, 45
 generating standard waves, 41 - 42
 generating triangle waves, 45
 generating using DAC data, 109
 half-rectified sine wave, 80
 hopping, 69 - 70, 263
 in EPROM, 82, 150
 initiating, 96, 146
 multiple in single output sequence, 243, 259
 multiple segment frequency, 245
 non-swept frequency, 155
 outputting, 56
 polarity, 169
 sine wave with spikes, 79
 starting, 251
WRITE registers, 237 - 238
 command register, 238
 control register, 237
 input data register, 238
 parameter register, 238

Z

Zero Crossing Marker Pulses, 104

