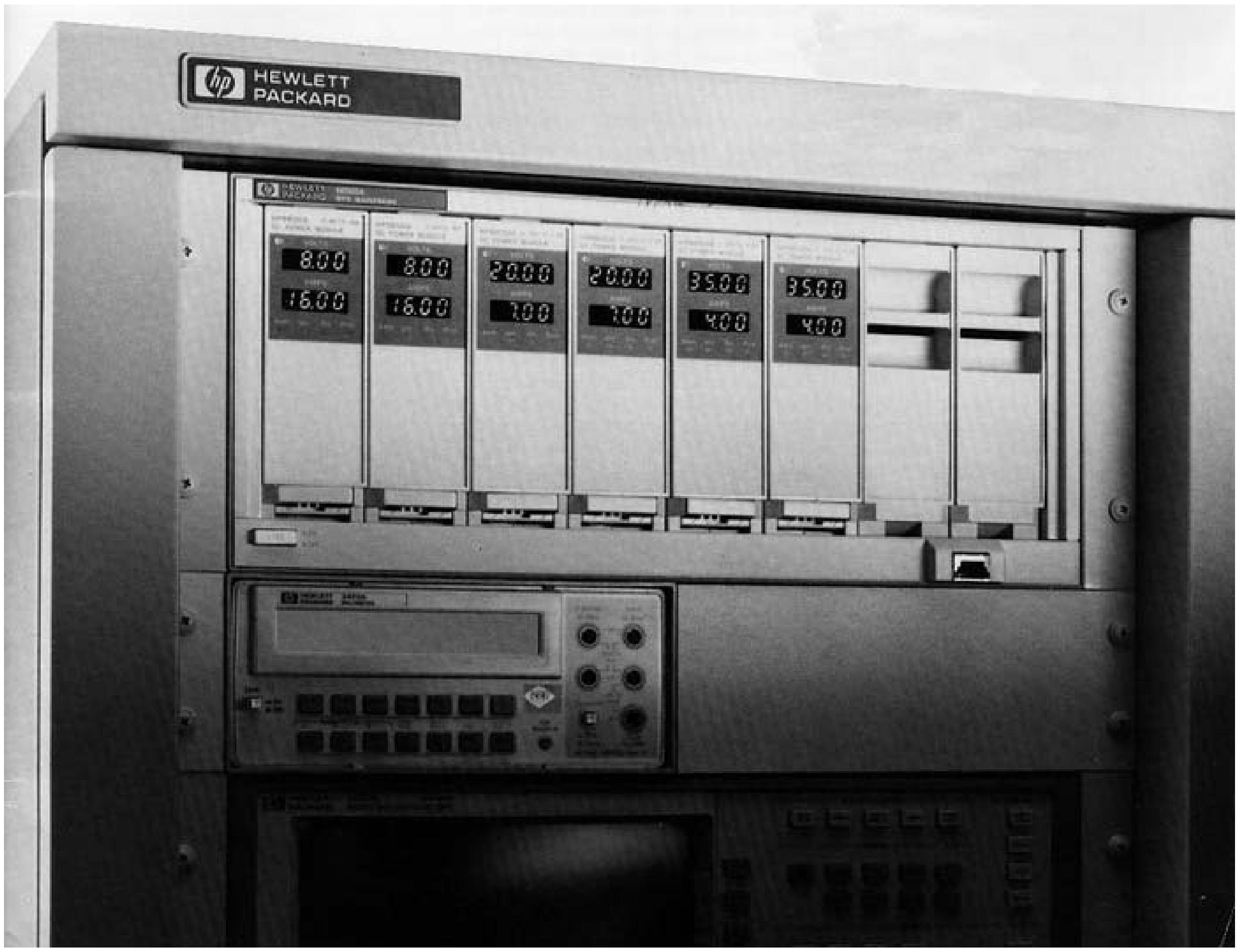
HP 66000 Modular Power System

Product Note

Using the New List and Trigger Capabilities to Sequence Outputs in These Applications:

- Sequencing Multiple Modules During Power Up
- Sequencing Multiple Modules to Power Down on Event
- Controlling Output Voltage Ramp Up at Turn On
- Providing Time-Varying Voltages
- Providing Time-Varying Current Limiting
- Output Sequencing Paced by the Computer
- Output Sequencing Without Computer Intervention



Introduction

This note provides information on how you can use the advanced programmable features of the HP 66000 Modular Power System to address a variety of applications. Although your exact application may not be described here, the capabilities described can be generalized and applied to your specific needs. The programming examples are given in HP BASIC. However, the appendix has listings translated into languages available on DOS-based computers.

These are the capabilities that are discussed and a description of how they can be applied:

Introduction to the HP 66000 Modular Power System

The HP 66000 Modular Power System (MPS) is a member of the Hewlett-Packard "One-Box" Solution family of power supplies. The "One-Box" family is the next generation of power supplies and offers many advantages over other power supply solutions.

The MPS consists of the HP 66000A Modular Power Mainframe and a number of different HP 66100A series dc power modules. Each MPS dc power module contains a dc power supply with automatic CV/CC crossover, voltage and current programmers, a DMM with precision current shunt, and a digital interface. This allows you to directly program voltage and current, read back the actual measured voltage and current, and monitor the status conditions of the power module. Each module is optimized for ATE applications and offers all the built-in features needed for easy integration into your test system.

The MPS dc power modules feature a SCPI (Standard Commands for Programmable Instruments) command set. This industry standard command set simplifies test system development by offering command set commonality between all types of instrumentation so that all instruments performing the same function use the same self-documenting SCPI instructions. For example, measuring the actual output voltage from a dc power module and reading the voltage from an external multimeter use the same SCPI commands. Because you spend less time learning device commands, you can get your application up and running faster.

The following sections of this note explain how to get the most from the advanced programming features of the MPS dc power modules. These explanations are followed by seven practical example applications that show how to apply these advanced features.

HP 66000 MPS Triggering

How Can I benefit from Triggers?

With most system power supplies, the only way to make it perform some action is to send it a command over HP-IB. Upon receipt of that command, the power supply processed the command and executes the action. This can cause two problems:

1. When many commands must be processed, the combined command processing time can slow down test execution.
2. When multiple power supplies must be programmed to act in unison, the command processing time can cause the resultant outputs to occur out of synchronization.

The MPS dc power modules offer a solution to these problems by providing the ability to send and receive triggers. With triggers, the new voltage and current levels are sent to the power module before they are needed so that they can be preprocessed. When you want to change the output, you send a trigger and the module immediately changes its output to the new programmed settings, without command processing delays. You no longer need to explicitly program the voltage and current each time you needed to change the output setting.

Triggers can be used when you require multiple bias modules to act in unison (for example, to apply power from three modules to the DUT simultaneously). Instead of separately programming the voltage and current settings for each of the three modules (and having them reach their voltage settings at different times), you can preprogram the modules to the required correct output. You can then trigger all three at once and they will reach their desired outputs together.

In addition to responding to triggers, the power module can provide a Trigger Out signal. This signal can be used to notify other hardware in your test system that the power module has changed its output. For example, Trigger Out can be connected to the Trigger In of a digital oscilloscope to request it to begin making a measurement on the DUT. With this method of synchronization, a measurement is valid because it is triggered as soon as the power module reaches its programmed voltage but not before it has had time to do so. This method also maximizes throughput because the measurement begins as soon as the module reaches its required output voltage.

What Actions Can I Trigger?

The MPS dc power modules support triggering of any of the following, called "trigger actions":

- a change in output voltage
- a change in output current
- a start of a List sequence (see next section)
- the pace of a List sequence (see next section)

The MPS dc power modules also have a programmable trigger delay. This allows you to insert a time interval between the time a trigger is detected by the module and the time the trigger action occurs.

From Where Do I Receive Triggers?

The MPS dc power modules can receive triggers from the following trigger sources:

The HP-IB. The computer can send trigger commands to the modules. There is a short command processing time associated with this source.

External Trigger In. This is the MPS mainframe TRIGGER IN connector. It is a BNC connector wired to a backplane bus that provides a trigger input common to all modules. TRIGGER IN accepts TTL levels, with the falling edge detected as the trigger.

Status Conditions. A change in a module's status can be used to generate a trigger. For example, you can use the constant current status condition to trigger the module to change its output voltage.

TTL Trigger. The TTL Trigger bus is a backplane bus that can be used as a trigger input by any module. (See next section for how to drive the backplane TTL Trigger bus.)

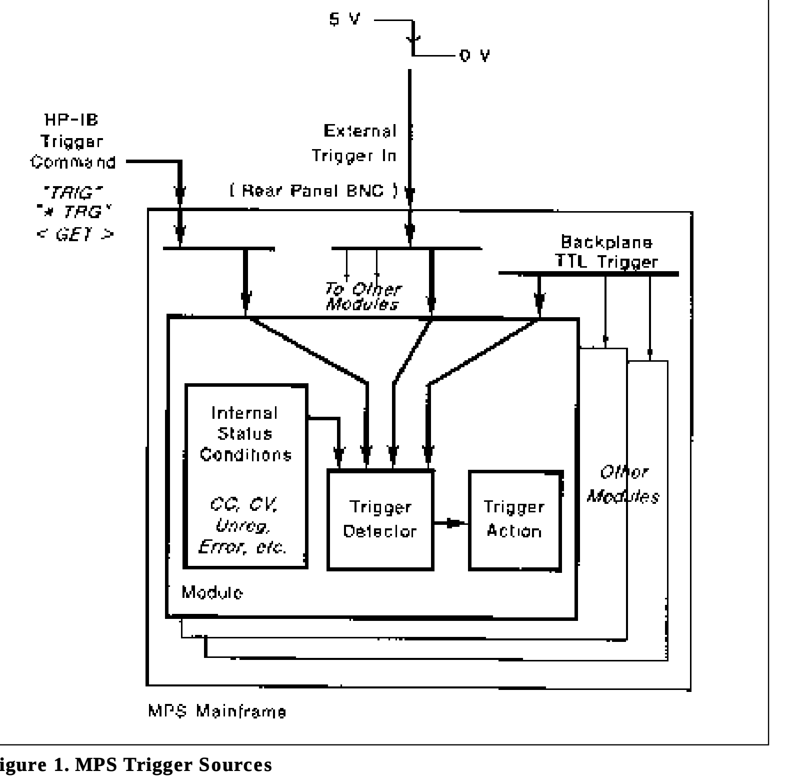


Figure 1. MPS Trigger Sources

How Do I Generate Triggers?

Any module can be programmed to drive the backplane TTL Trigger bus. By means of this bus, a module can trigger all other modules that have been programmed to respond to the TTL Trigger bus as their trigger source (see the preceding paragraph).

The TTL Trigger bus is wired to the MPS mainframe TRIGGER OUT BNC connector. The TRIGGER OUT signal is a 20-microsecond, negative-true TTL pulse. Whenever a module drives the backplane TTL Trigger bus, it also drives the TRIGGER OUT connector on the MPS mainframe.

The MPS dc power modules can drive the TTL Trigger line in response to:

The HP-IB. When the computer sends a trigger command to a module, it can drive the TTL Trigger bus. This permits you to trigger one module from the computer and have that module "echo" the trigger command over the TTL Trigger bus, simultaneously triggering all other modules.

External Trigger In. A module can drive the TTL Trigger bus in response to a trigger received from the mainframe TRIGGER IN connector.

Status Conditions. A change in a module's status can be used to drive the TTL Trigger bus. This allows a change in status in one module to trigger other modules. For example, you can use a current limit condition in one module to trigger the other modules to perform a trigger action.

How Do I Enable the MPS to Respond to Triggers?

The default state of the dc power module is the idle state, where trigger detection is disabled. Before it can respond to a trigger, the module must be placed in the "initiated" state (by a computer command). Once initiated, the module can detect a trigger from the selected source. This is the only time when the module responds to a trigger; no trigger detection occurs in the idle state (above) or in the "triggered" state (next paragraph).

Once the trigger is detected, the module moves into a triggered state and performs the trigger action (after waiting any programmed trigger delay time). Upon completion of the trigger action, the module returns to the idle state.

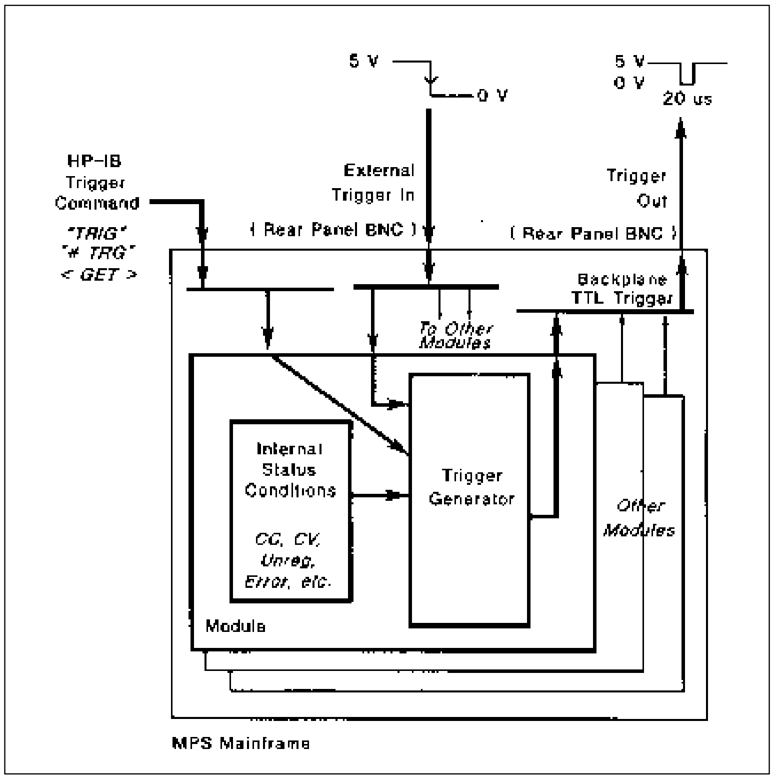


Figure 2. Generating TTL Triggers

HP 66000 MPS Lists

What is a List and How Can I Benefit from its Use?

Normally, an MPS dc power module operates in a mode called "Fixed", where the output settings stay fixed at the programmed value until you send another command to change them. Another available mode is called "List" mode, where no computer commands are needed to alter the output settings.

Lists allow you to download a sequence of voltage and/or current values that will be generated at specific intervals or on the occurrence of a trigger. Up to 20 voltage and current values (points), with 20 associated dwell times, can be programmed. Once downloaded, the dc power module can execute the sequence of output settings with minimal interaction between the computer and the dc power module. By pacing a List using the programmable dwell times, you can time output changes more precisely. By pacing a List using triggers, you can better synchronize changes in the output with asynchronous events.

Can a List be Repeated?

Although each List can contain up to 20 points, a List can be programmed to repeat. The number of repetitions, called the List count, is programmable from 1 to 65534 loops. It can also be set to repeat continuously by making the count infinite.

How Do I Execute a List?

If you want the voltage to change per a downloaded List, set the voltage mode to List. Similarly, if you want the current to change per a downloaded List, set the current mode to List. If you want both to change, set both to List mode.

A module begins executing a List in response to a trigger. All of the trigger sources mentioned under "HP 66000 MPS Triggering" are valid. Once triggered, Lists can be paced by dwell times or paced by triggers.

Dwell-Paced Lists

When the List is dwell paced, each List point is programmed for the dwell time associated with that point. When the dwell time expires, the output changes to the next point in the List.

Valid dwell times for the MPS dc power modules are from 10 milliseconds to 65 seconds per point. The resolution is in 2-millisecond increments. The command to select dwell-paced List mode is LIST:STEP AUTO. This commands the List to step automatically to the next point after expiration of the dwell time. If the List count is greater than 1, the List will automatically repeat until the count has been satisfied. With dwell pacing, the execution of the entire List is the triggered action, so only one trigger is required.

The following diagram shows the timing for the start trigger, status conditions, and output voltage changes for a dwell paced List. The programmed parameters are:

List points:	5 V for 50 ms
	10 V for 150 ms
	0 V for 30 ms
Trigger delay:	20 ms

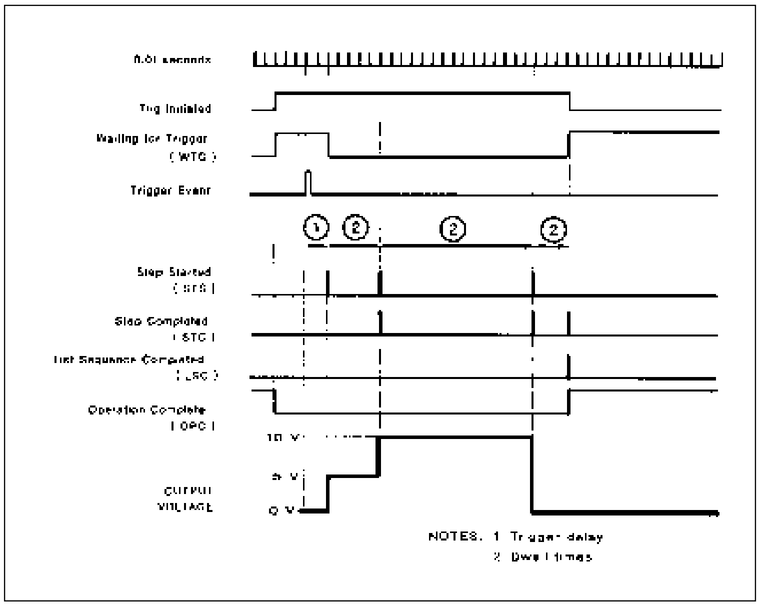


Figure 3. Timing Diagram for Dwell-Paced Lists

Triggering Each Point in a List

When the List is paced by triggers, each List point is generated when a trigger is detected. When the dwell time expires, the output stays at the same value until the next trigger is detected. Thus, one trigger is required to sequence the List from each point to the next. Any triggers received during a point's dwell time are ignored. The command to select trigger-paced List mode is LIST:STEP ONCE. This commands the List to execute only one List point (step) upon receipt of each trigger.

Note that since the module ignores triggers during the dwell time, you must be careful not to program a dwell time that is so long that you will miss the next trigger. If you are unsure of when triggers will be sent and you want to guarantee that you get all of them, set the dwell time to its minimum value.

The following diagram shows the timing for triggers, status conditions, and output voltage changes for a trigger paced List. The programmed parameters are:

List points:	5 V for 50 ms
	10 V for 150 ms
	0 V for 30 ms
Trigger delay:	20 ms

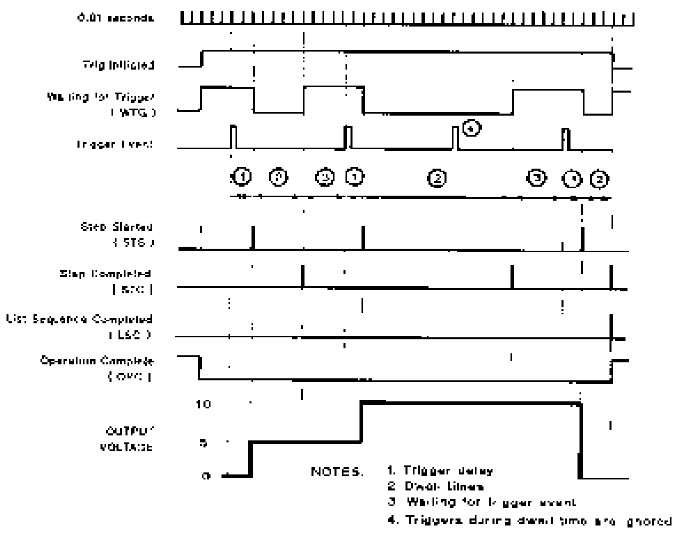


Figure 4. Timing Diagram for Trigger-Paced Lists

Using Triggers with Lists to Implement Output Sequences

This section describes how to make several modules respond synchronously. The general concepts described here provide the basis for many synchronized multiple-output applications.

Each MPS dc power module functions independently. To get them to operate in unison, a common synchronizer must be established. This can be done by having a single trigger source or timebase for all modules.

For example, to sequence 3 modules to power up 50 ms apart, you could use the three simple Lists in the table below:

As you can see, although each module functions independently from the other, the result is a sequenced output. This simple method has two problems if tight synchronization is needed:

- How do you get three Lists to start simultaneously?
- How do you guarantee that the dwell time clocks in all modules stay synchronized?

The following sections describe how the MPS solves the problems of synchronization.

HP-IB Triggering Between Modules

Sometimes a common trigger signal is not available. For example, when it sends a trigger command, the computer cannot talk to all devices on the HP-IB simultaneously. The time required to individually trigger each dc power module could cause an undesirable "skewing" of the trigger actions.

You can use the backplane TTL Trigger bus of the mainframe to solve the lack of a common trigger. The procedure is as follows: Program the first module to generate a backplane TTL Trigger when it receives a trigger command over HP-IB. Then, command all other modules to use the backplane TTL Trigger bus as a source. The computer then triggers only the first module, which "echos" that trigger over the backplane to all other modules simultaneously.

Status Triggering Between Modules

Some applications require a trigger to occur in response to a change in status within a module. Since change-of-status detection is internal to each module, one module is unaware of, and therefore cannot react to, status changes within another module. Using the mainframe backplane TTL Trigger bus solves this. The module that detects a status change can be programmed to not only trigger itself to perform some trigger action, but also to send a TTL Trigger out on the backplane bus. This triggers all the other modules.

	module #1		module #2		module #3	
step no.	voltage list	dwell time	voltage list	dwell time	voltage list	dwell time
1	5 V	50 ms	0 V	50 ms	5 V	50 ms
2	5 V	50 ms	5 V	50 ms	0 V	50 ms
3	5 V	50 ms	5 V	50 ms	5 V	50 ms

A Generalized Synchronizing Scheme

It is desirable to simultaneously start and then maintain synchronization between Lists running in multiple modules . You can use a combination of HP-IB and status triggering to synchronize several Lists. Use the following approach:

For Module 1...

- Trigger the module from a computer trigger command over HP-IB
- Generate the output voltage from a List
- Use a dwell-paced List. The computer trigger command causes the execution of the entire List.
- Have the module generate a backplane TTL Trigger at the start of each point (step) in the List. This can be done by triggering on the STS (Step Started) status condition.

For Modules 2 and 3...

- Trigger the module from the backplane TTL Trigger bus
- Generate the output voltage from a List
- Use a trigger-paced List. Each backplane TTL Triggers causes the execution of one point of the List.
- Program the dwell times to minimum (so you don't miss any triggers)

When the computer sends the trigger command to module 1, it will begin executing its List. Module 1 will step through its List of points, generating a TTL Trigger each time the Step Started (STS) condition occurs at the start of each new point. Modules 2, 3... will wait for these triggers before proceeding to their next point. The result is an "unskewed" synchronized output, started by a single computer command.

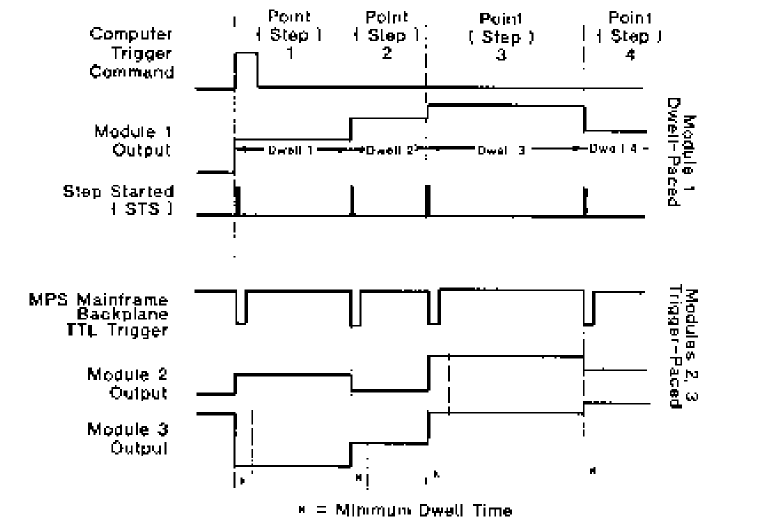


Figure 5. Timing Diagram for the Generalized Synchronizing Scheme

Applications

This section contains seven example applications. For each application, there is:

- An overview of the application
- Which MPS features are used to implement the application
- The advantages and benefits of the MPS solution
- The details of the implementation of the solution
- A block diagram of the setup
- A sample program listing in HP BASIC
- A description of variations on the application

The following table lists what MPS features are used in each of the applications. It can be used as an index into this section.

Application 1. Sequencing Multiple Modules During Power Up
Application 2. Sequencing Multiple Modules to Power Down on Event
Application 3. Controlling Output Voltage Ramp Up at Turn On
Application 4. Providing Time-Varying Voltages
Application 5. Providing Time-Varying Current Limiting
Application 6. Output Sequencing Paced by the Computer
Application 7. Output Sequencing Without Computer Intervention

	Application						
	1	2	3	4	5	6	7
Lists							
20-point current List					•		
20-point voltage List			•	•		•	•
Repetitive Lists				•			
Dwell time			•	•	•		•
List Pacing							
Dwell-paced Lists			•	•	•		
Trigger-paced Lists						•	•
Actions due to a change in status							
Generate an SRQ				•			•
Generate a trigger		•					•
Disable the output				•	•		
Stop the List				•	•		
Triggers							
Change the voltage on trigger	•	•			•		
Trigger in/out from MPS backplane TTL Trigger	•	•				•	•
Trigger on an HP-IB trigger command	•			•	•	•	•
Trigger delay	•	•					
Other Features							
Active downprogramming		•		•	•		
Overcurrent protection				•	•		

Application 1.
Sequencing Multiple Modules during Power Up

Overview of application

When testing mixed signal devices, ± bias supply voltages are typically applied before logic bias supply voltages. For a device that is sensitive to when bias voltages are applied, the order of power up of multiple power modules can be controlled.

For this example, the device requires three bias supplies, + 5 V for the logic circuits and ± 15 V for amplifier circuits. To properly power up the device, the supplies must be sequenced so that the ± 15 V are applied first and the + 5 V is applied 50 ms later.

The MPS can easily address this application through the use of triggers. The trigger will cause the modules to change from 0 V, where they are not powering the DUT, to their final voltage. By delaying the response to the trigger, you can control when the module's output voltage changes. This means you can control the sequence of the modules during power up.

MPS features used

- Change the voltage on trigger
- Trigger in/out from MPS mainframe backplane TTL Trigger
- Trigger on an HP-IB trigger command Trigger delay
- Trigger delay

Advantages/benefits of the MPS solution

- By using trigger delay, the timing is accurate and repeatable.
- The sequence is simpler to program (no timing loops).
- The computer is not devoted to sequencing power modules.
- The computer does not provide timing for the sequence.
- One command initiates the sequence.

Implementation details

How the MPS implements the sequence

- The computer sends a trigger command to the first module.
- The first module simultaneously sends a backplane trigger to other two modules and goes to + 15 V.
- The second module receives the backplane TTL Trigger and immediately goes to -15 V.
- The third module receives the backplane TTL Trigger, delays 50 ms, and then goes to + 5 V.

MPS set up

Module in slot 0:

- The module is connected to + 15 V on the DUT.
- The initial voltage setting is 0 V.
- The module listens for the computer to send a trigger command.
- Upon receipt of the trigger command, the module goes to 15 V.
- Also upon receipt of the trigger command, the module generates a backplane TTL Trigger.

Module in slot 1:

- The module is connected to -15 V on the DUT.
- The initial voltage setting is 0 V.
- The module listens for a backplane TTL Trigger.
- Upon receipt of the trigger, the module goes to 15 V.

Module in slot 2:

- The module is connected to + 5 V on the DUT.
- The initial voltage setting is 0V.
- The module listens for a backplane TTL Trigger.
- The trigger delay is programmed to 50 ms.
- Upon receipt of the trigger, the module waits the trigger delay time and then goes to 5 V.

Variations on this implementation

1. The modules could be set to generate SRQ when the last module (+ 5 V) reaches its final output value. This would notify the computer that power has been applied to the DUT and the testing can begin.
2. To provide a delay between the application of the + 15 V and the -15 V bias, you can program different trigger delays into modules 2 and 3. The delay time will be relative to module in slot 0.
3. To get all three modules to apply power to the DUT at the same time, simply eliminate the trigger delay on the + 5 V module.
4. When modules need to be connected in parallel to increase current, they will also need to be synchronized so that they all apply power simultaneously. To get modules in parallel to apply power at the same time, use the approach described in this example, but eliminate any trigger delays.

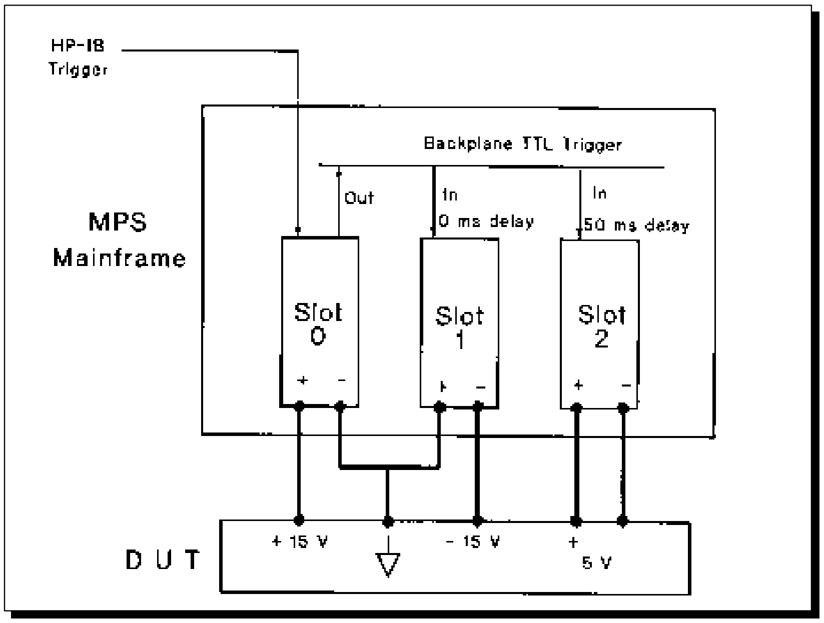


Figure 1-1. Block Diagram of Application #1

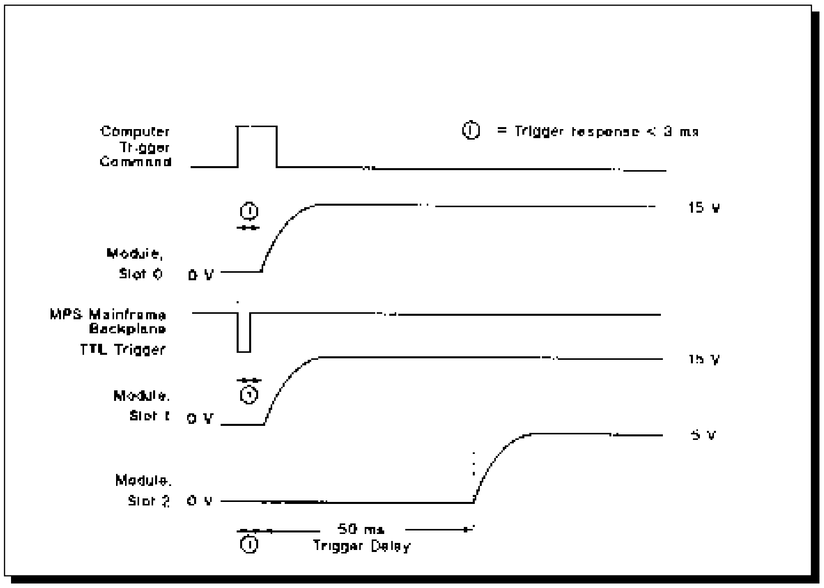


Figure 1-2. Timing Diagram of Application #1

```
10    ! APPLICATION #1: SEQUENCING MULTIPLE MODULES DURING POWER UP
20    ! PROGRAM: APP_1
30    !
40    ASSIGN @Slot0 TO 70500                        ! SELECT CODE 7, MAINFRAME ADDRESS 05, SLOT 00
50    ASSIGN @Slot1 TO 70501                        ! SELECT CODE 7, MAINFRAME ADDRESS 05,     SLOT 01
60    ASSIGN @Slot2 TO 70502                        ! SELECT CODE 7,   MAINFRAME ADDRESS 05, SLOT 02
70    !
80    ! SET UP MODULE IN SLOT 0 AS +15 V BIAS SUPPLY ---------------------
90    !
100   OUTPUT @Slot0;"*RST;*CLS;STATUS:PRESET" ! RESET AND CLEAR MODULE
110   OUTPUT @Slot0;"VOLT 0”                        ! START AT 0 V
120   OUTPUT @Slot0;"VOLT:TRIGGERED 15"             ! GO TO 15 V ON TRIGGER
130   OUTPUT @Slot0;"TRIGGER:SOURCE BUS"            ! TRIGGER SOURCE IS HP-IB 'BUS'
140   OUTPUT @Slot0;OUTPUT:TTLTRG:SOURCE BUS" ! GENERATE BACKPLANE TTL TRIGGER WHEN HP-IB 'BUS'
      TRIGGER IS RECEIVED
150   OUTPUT @Slot0;"OUTPUT:TTLTRG:STATE ON"      ! ENABLE BACKPLANE TTL TRIGGER DRIVE
160   OUTPUT @Slot0;"OUTPUT ON"                     ! ENABLE OUTPUT
170   OUTPUT @Slot0;"INITIATE"                      ! ENABLE RESPONSE TO TRIGGER
180   !
190   ! SET UP MODULE IN SLOT 1 AS -15 V BIAS SUPPLY ---------------------
200   !
210   OUTPUT @Slot1;"*RST;*CLS;STATUS:PRESET”       ! RESET AND CLEAR MODULE
220   OUTPUT @Slot1;"VOLT 0"                        ! START AT 0 V
230   OUTPUT @Slot1;"VOLT:TRIGGERED 15"             ! GO TO 15 V ON TRIGGER
240   OUTPUT @Slot1;"TRIGGER:SOURCE TTLTRG"         ! TRIGGER SOURCE IS BACKPLANE TTL TRIGGER
250   OUTPUT @Slot1;"OUTPUT ON"                     ! ENABLE OUTPUT
260   OUTPUT @Slot1;"INITIATE"                      ! ENABLE RESPONSE TO TRIGGER
270   !
280   ! SET UP MODULE IN SLOT 2 AS +5 V BIAS SUPPLY --------------------
290   !
300   OUTPUT @Slot2;"*RST;*CLS;STATUS:PRESET.       ! RESET AND CLEAR MODULE
310   OUTPUT @Slot2;"VOLT 0"                        ! START AT 0 V
320   OUTPUT @Slot2;"VOLT:TRIGGERED 5"              ! GO TO 5 V ON TRIGGER
330   OUTPUT @Slot2;"TRIGGER:SOURCE TTLTRG”         ! TRIGGER SOURCE IS BACKPLANE TTL TRIGGER
340   OUTPUT @Slot2; "TRIGGER :DELAY 0.050"         ! 50 ms TRIGGER DELAY
350   OUTPUT @Slot2;"OUTPUT ON"                     ! ENABLE OUTPUT
360   OUTPUT @Slot2;"INITIATE"                      ! ENABLE RESPONSE TO TRIGGER
370   !
380   ! BEFORE TRIGGERING THE NODULES, DETERMINE IF THE MODULES ARE READY BY CHECKING FOR
390   ! 'WAITING FOR TRIGGER' (BIT 5 OF THE OPERATION STATUS REGISTER). IF THE LAST NODULE PROGRAMMED
400   ! IS READY THEN SO ARE THE OTHERS, SO JUST CHECK SLOT 2.
410   !
420   ! YOU COULD ELIMINATE THIS STEP BY SIMPLY INSERTING A PAUSE IN THE PROGRAM. HOWEVER, BY
430   ! CHECKING THE INSTRUMENT STATUS, YOU CAN AVOID TIMING PROBLEMS. ALSO, ANY OTHER OPERATIONS
440   ! THAT TAKE TIME WILL GIVE THE MODULES A CHANCE TO COMPLETE PROCESSING.
450   !
460   REPEAT
470   OUTPUT @Slot2; "STATUS :OPERATION: CONDITION?"
480   ENTER @Slot2;Condition_data
490   UNTIL BIT(Condition_data,5)                   ! TEST FOR BIT 5 = TRUE
500   !
510   ! TRIGGER MODULE IN SLOT 0 TO BEGIN SEQUENCING THE 3 NODULES TO POWER UP
520   !
530   OUTPUT @Slot0;"*TRG"                          ! SEND HP-IB 'BUS TRIGGER
540   !
550   END
```

Figure 1-3. HP BASIC Program Listing for Application #1

Application 2.
Sequencing Multiple Modules to Power Down on Event

Overview of application

When testing devices, such as some GaAs and ECL devices that are sensitive to when bias voltages are removed, the order of power-down of multiple power modules can be controlled. The power-down sequence can be initiated by an event, such as a change in power module status, fault condition, detection of a TTL signal, etc.

For this example, there are three supplies + 5 V and ±15 V. (See previous application for how to generate a power up sequence.) Once the power has been applied to the DUT, the modules can be reprogrammed to perform the power down sequence. The power down sequence is initiated when a fault in the DUT draws excessive current from the power module, causing the module to change from CV to CC. To prevent damage to the DUT, it is necessary to remove the + 5 V first, then the ±15 V modules 15 ms later.

Once again, MPS triggering can solve the application. In this scenario, the CV-to-CC crossover event will be used as the trigger source. The trigger will cause the modules, in the correct order, to change from their programmed voltages down to 0 V.

MPS features used

- Generate a trigger on a change in internal status
- Change the voltage on trigger
- Trigger in/out from MPS mainframe backplane TTL Trigger
- Trigger delay
- Active downprogramming

Advantages/benefits of the MPS solution

- By using the modules' change in status to automatically generate a trigger, the computer is not devoted to polling the modules to detect a change in state.
- By letting each module monitor its status, the CC condition will generate a response faster than if the computer was polling the module to detect a change in state.
- The sequence is simpler to program (no timing loops).
- By using trigger delay, the timing is accurate and repeatable because the computer does not provide timing for the sequence.
- The active downprogrammers in the module output can quickly discharge the module's output capacitors and any capacitance in the DUT.

Implementation details

How the MPS implements the solution

- All modules are set to listen for a backplane TTL Trigger.
- When any module detects a change in status from CV to CC, it sends out a backplane TTL Trigger.
- When the + 5 V module receives the trigger, it immediately goes to 0 V.
- When the + 15 V and -15 V modules receive the trigger, they wait the trigger delay time and then go to 0 V.

(Note that any module can both generate the backplane TTL Trigger signal and be triggered by that same signal.)

MPS set up

Module in slot 0:

- The module is connected to + 15 V on the DUT.
- The initial voltage setting is 15 V.
- The module monitors its status.
- The module will generate a backplane TTL Trigger on CV-to-CC crossover.
- The module listens for a backplane TTL Trigger.
- The trigger delay is programmed to 15 ms.
- Upon receipt of the trigger, the module waits the trigger delay time and then goes to 0 V.

Module in slot 1:

- The module is connected to supply -15 V to the DUT.
- The initial voltage setting is 15 V.
- The module monitors its status.
- The module will generate a backplane TTL Trigger on CV-to-CC crossover.
- The module listens for backplane TTL Trigger.
- The trigger delay is programmed to 15 ms.
- Upon receipt of the trigger, the module waits the trigger delay time and then goes to 0 V.

Module in slot 2:

- The module is connected to supply + 5 V to the DUT.
- The initial voltage setting is 5 V.
- The module monitors its status.
- The module will generate a backplane TTL Trigger on CV-to-CC crossover.
- The module listens for backplane TTL Trigger.
- Upon receipt of the trigger, the module immediately goes to 0 V.

Variations on this implementation

1. The modules could be set to generate SRQ when the last module reaches 0 V. This could notify the computer that power has been removed from the DUT.
2. The modules could be set to generate a DFI (Discrete Fault Indicator) signal on the MPS rear panel on a change in status. This signal could be used to shut down other power modules, to flash an alarm light, or to sound a buzzer. This could also be routed to other instruments to signal them to stop making measurements.
3. To get all three modules to remove power from the DUT at the same time, simply eliminate the trigger delay on the ±15 V modules.
4. To provide a delay between the removal of the three bias voltages, you can program a different trigger delay into each module.

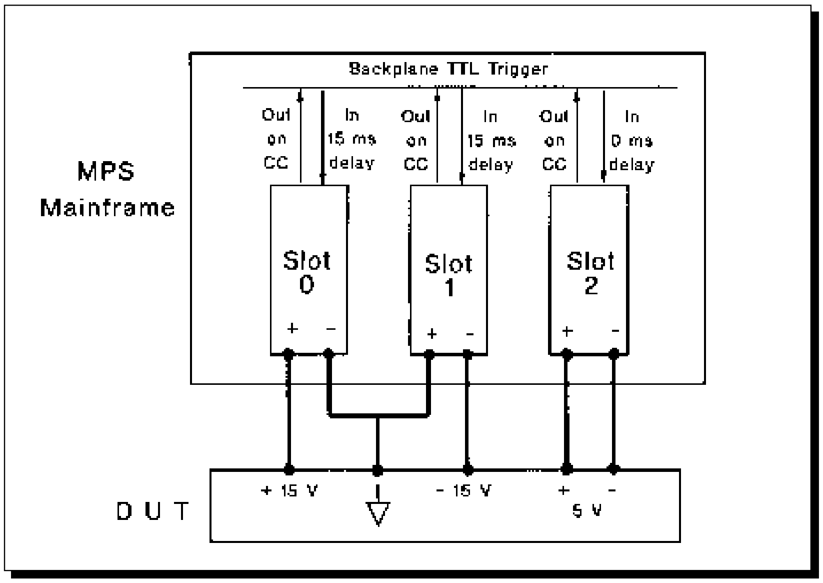


Figure 2-1. Block Diagram of Application #2

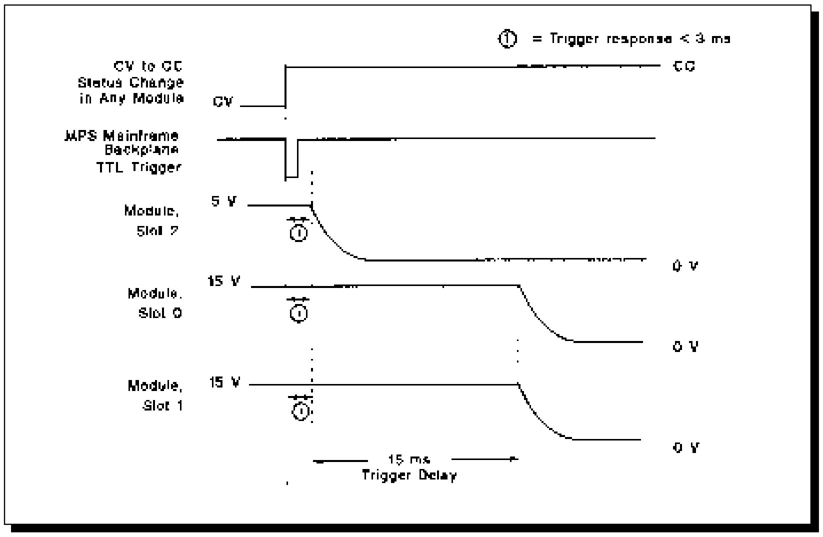


Figure 2-2. Timing Diagram of Application #2

```
10     ! APPLICATION #2: SEQUENCING MULTIPLE MODULES TO POWER DOWN ON EVENT
20     ! PROGRAM: APP_2
30     !
40     ASSIGN @Slot0 TO 70500                              ! SELECT CODE 7, MAINFRAME ADDRESS 05,   SLOT 00
50     ASSIGN @Slot1 TO 70501                              ! SELECT CODE 7, MAINFRAME ADDRESS 05, SLOT 01
60     ASSIGN @Slot2 TO 70502                              ! SELECT CODE 7, MAINFRAME ADDRESS 05 SLOT 02
70     !
80     ! SET UP MODULE IN SLOT 0 AS +15 V BIAS SUPPLY ____________________
90     !
100    OUTPUT @Slot0;"*RST;*CLS;STATUS:PRESET"             ! RESET AND CLEAR MODULE
110    OUTPUT @Slot0;"CURR .5"
120    OUTPUT @Slot0;"VOLT 15"                             ! START AT 15 V
130    OUTPUT @Slot0;"VOLT:TRIGGERED 0"                    ! GO T0 0 V ON TRIGGER
140    OUTPUT @Slot0;"TRIGGER:SOURCE TTLTRG"               ! TRIGGER SOURCE IS TTL TRIGGER
150    OUTPUT @Slot0;"TRIGGER:DELAY .015"                  ! 15 ms TRIGGER DELAY
160    OUTPUT @Slot0;"INITIATE"                            ! ENABLE RESPONSE TO TRIGGER
170    OUTPUT @Slot0;"OUTPUT:TTLTRG:SOURCE LINK"           ! GENERATE A BACKPLANE TTL TRIGGER
180    OUTPUT @Slot0;"OUTPUT:TTLTRG:LINK 'CC'.             ! WHEN A CV_TO_CC TRANSITION OCCURS
190    OUTPUT @Slot0;"OUTPUT:TTLTRG:STATE ON"              ! ENABLE TTL TRIGGER DRIVE
200    OUTPUT @Slot0;"OUTPUT ON"                           ! ENABLE OUTPUT
210    !
220    ! SET UP MODULE IN SLOT 1 AS _15 V BIAS SUPPLY ____________________
230    !
240    OUTPUT @Slot1;"*RST;*CLS;STATUS:PRESET"             ! RESET AND CLEAR MODULE
250    OUTPUT @Slot,"CURR .5"
260    OUTPUT @Slot1;"VOLT 15"                             ! START AT 15 V
270    OUTPUT @lot1;"VOLT:TRIGGERED 0"                     ! GO TO 0 V ON TRIGGER
280    OUTPUT @Slot1;"TRIGGER:SOURCE TTLTRG”               ! TRIGGER SOURCE IS BACKPLANE TTL TRIGGER
240    OUTPUT @Slot1;"TRIGGER:DELAY .015"                  ! 15 ms TRIGGER DELAY
300    OUTPUT @Slot1;"INITIATE"                            ! ENABLE RESPONSE TO TRIGGER
310    OUTPUT @Slot1;"OUTPUT:TTLTRG:SOURCE LINK"           ! GENERATE A BACKPLANE TTL TRIGGER
320    OUTPUT @Slot1;"OUTPUT:TTLTRG:LINK 'CC'"             ! WHEN A CV_TO_CC TRANSITION OCCURS
330    OUTPUT @Slot1;"OUTPUT:TTLTRG:STATE ON”              ! ENABLE TTL TRIGGER DRIVE
340    OUTPUT @Slot1;”OUTPUT ON"                           ! ENABLE OUTPUT
350    !
360    ! SET UP MODULE IN SLOT 2 AS +5 V BIAS SUPPLY _____________________
370    !
380    OUTPUT @Slot2;"*RST;*CLS;STATUS:PRESET"             ! RESET AND CLEAR MODULE
340    OUTPUT @Slot2;"CURR .5"
400    OUTPUT @Slot2;"VOLT 5"                              ! START AT 5 V
410    OUTPUT @Slot2;"VOLT:TRIGGERED 0"                    ! GO TO 0 V ON TRIGGER
420    OUTPUT @Slot2;"TRIGGER:SOURCE TTLTRGH               ! TRIGGER SOURCE IS BACKPLANE TTL TRIGGER
430    OUTPUT @Slot2;"INITIATE"                            ! ENABLE RESPONSE TO TTL TRIGGER
440    OUTPUT @Slot2;"OUTPUT:TTLTRG:SOURCE LINK”           ! GENERATE A BACKPLANE TTL TRIGGER
450    OUTPUT @Slot2;"OUTPUT:TTLTRG:LINK 'CC'"             ! WHEN A CV_TO_CC TRANSITION OCCURS
460    OUTPUT @Slot2;"OUTPUT:TTLTRG:STATE ON”              ! ENABLE TTL TRIGGER DRIVE
470    OUTPUT @Slot2;”OUTPUT ON"                           ! ENABLE OUTPUT
480    !
490    ! THE POWER NODULES ARE NOW SET UP TO IMPLEMENT THE POWER DOWN ON EVENT.
500    ! ANY TIRE ANY MODULE GOES INTO CC, THE SEQUENCE WILL OCCUR.
510    !
520    END
```

Figure 2-3. HP BASIC Program Listing for Application #2

Application 3.
Controlling Output Voltage Ramp Up at Turn On

Overview of application

When control over the rate of voltage ramp up at turn-on of the power module output is required, the desired shape can be approximated by downloading and executing a series of voltage and dwell time points.

For this example, you need to program the power module to change its output from 2 volts to 10 volts, slewing through the 8 volt transition in 0.5 seconds. This results in a turn-on ramp-up of 16 V per second.

The MPS can create this voltage versus time characteristic using Lists. The desired characteristic (in this case, linear) is simulated using the 20 available voltage points. To determine the value of each point in the transition, simply divide the change in voltage by 20. To determine the dwell time of each voltage point, divide the total transition time by 19. After the List has been executed, the module will continue to output the final value (in this case, 10 volts) until the output has been reprogrammed to another value. Note that the dwell-time of the last point is not part of the transition time.

To determine the slowest ramp up (longest transition time) that can be generated, you must consider how smooth you need the voltage versus time characteristic to be. As the dwell time associated with each point gets longer, the output voltage will become more like a "stair step" and less like a linear transition. (See Figure 3-1)

To determine the fastest ramp up (shortest transition time) that can be generated, you must consider the minimum dwell time specification (10 ms) and the maximum rise-time of specification the power module (20 ms). If you program 10 ms dwell times, the power module will not be able to reach its output voltage before the next voltage point is output.
(See Figure 3-2)

MPS features used

- 20-point voltage List
- Dwell time
- Dwell-paced Lists

Advantages/benefits of the MPS solution

- By using Lists, the module changes its output voltage automatically, so that the computer is not devoted to reprogramming the output voltage.
- The outputs can change faster when dwell paced than when the computer must explicitly reprogram each change.
- The sequence is simpler to program (no timing loops).
- By using dwell times, the timing of each point is accurate and repeatable.
- The computer does not provide timing for the sequence.
For negative-going ramps, the active downprogrammers in the module output can quickly discharge the module's output capacitors and any capacitance in the DUT when negative going ramps are required.

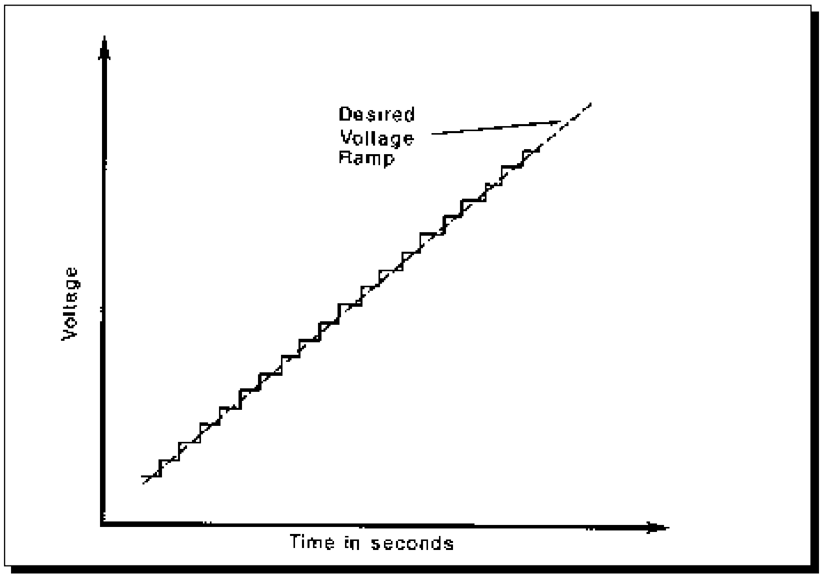


Figure 3-1. Simulating a Slow Voltage Ramp

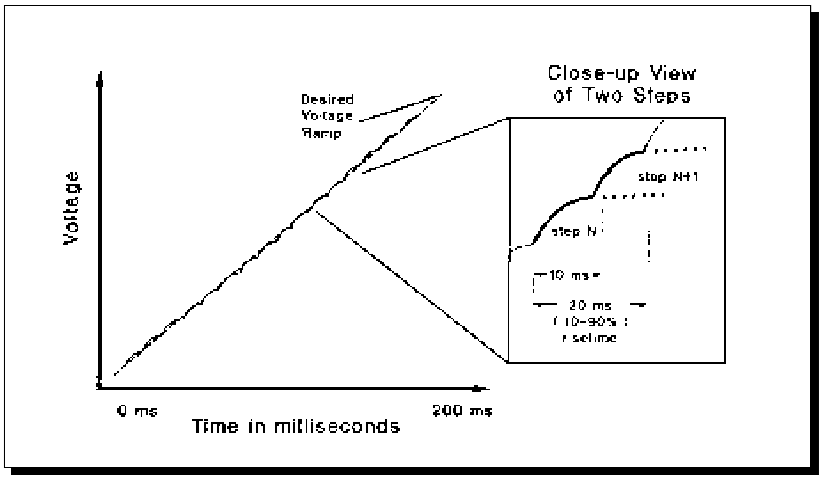


Figure 3-2. Simulating a Fast Voltage Ramp

Implementation details

How the MPS implements the sequence

- The module is programmed to List mode.
- The module will execute a dwell-paced List.
- The 20 voltage points are downloaded to the module.
- The 20 dwell times are downloaded to the module.
- When the transition must occur, the module is triggered by the computer.
- The module output ramps under its own control.

1. The module could be set to begin ramping in response to an external or backplane TTL Trigger.
2. The module could be set to generate SRQ when it has finished its transition. This would notify the computer that the voltage is at the proper level.
3. The module could be set to generate an external trigger when it has finished its transition. This trigger could be routed to other instruments as a signal to start making measurements.
4. Multiple modules could be programmed to slew together in response to the computer trigger command.
5. The module could be set to generate an external trigger for each point in the transition. This trigger could be routed to other instruments as a signal to take a measurement at various supply voltages. (See application #7)
6. Many voltage versus time characteristics can be generated by varying the voltage values and the dwell times in the List.

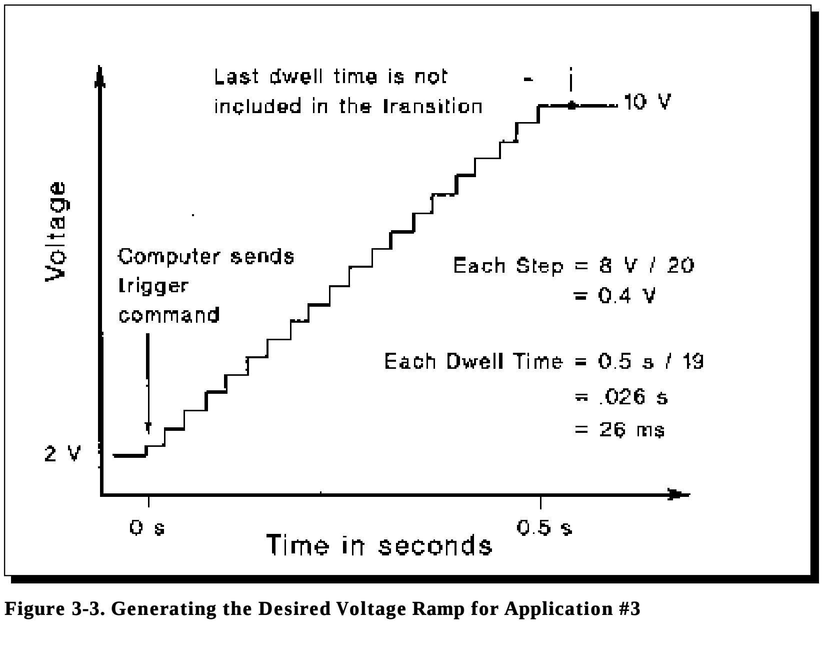


Figure 3-3. Generating the Desired Voltage Ramp for Application #3

```
10     ! APPLICATION #3:                                     CONTROLLING VOLTAGE RAMP UP AT TURN ON
20     ! PROGRAM: APP 3
30     !
40     ASSIGN @Slot0 TO 70500                                ! SELECT CODE 7, MAINFRAME ADDRESS 05, SLOT 00
50     !
60     OPTION BASE 1
70     DIM V_step(20)                                        ! ARRAY TO HOLD THE VOLTAGE RAMP STEPS
80     Vstart=2                                              ! START VOLTAGE FOR RAMP
90     Vstop=10                                              ! STOP VOLTAGE FOR RAMP
100    Runp_time=.5                                          ! SECONDS TO CHANGE FROM Vstart TO Vstop
110    Dwell=Ramp_time/19                                    ! IN SECONDS
120    !
130    ! SINCE THE OUTPUT STAYS AT THE LAST VOLTAGE POINT AFTER ITS DWELL TIME EXPIRES, THE DWELL TIME OF THE
140    ! LAST POINT IS NOT PART OF THE TRANSITION TIME. THEREFORE, DIVIDE THE TOTAL TIME BY 19 POINTS, NOT 20.
150    ! ALSO, YOU ONLY NEED TO DOWNLOAD 1 DWELL TIME. IF THE MODULE RECEIVES ONLY 1 DWELL TIME, IT ASSUMES
160    ! YOU WANT THE SAME DWELL TIME FOR EVERY POINT IN THE LIST.
170    !
180    FOR I=1 TO 20
190    V_step(I)=Vstart+(((Vstop-Vstart)/20)*1)              ! CALCULATES VOLTAGE LIST POINTS
200    NEXT I
210    !
220    OUTPUT @Slot0;"*RST;*CLS;STATUS:PRESET”               ! RESET AND CLEAR MODULE
230    OUTPUT @Slot0;”VOLT ";Vstart                          ! START RAMP AT Vstart
240    OUTPUT @Slot0;"CURR .1"
250    OUTPUT @Slot0;"OUTPUT ON"                             ! ENABLE OUTPUT
260    OUTPUT @Slot0;"VOLT:MODE LIST”                        ! SET TO GET VOLTAGE FROM LIST
270    OUTPUT @Slot0;”LIST:VOLT ";V step(*)                  ! DOWNLOAD VOLTAGE POINTS
280    OUTPUT @Slot0;"LIST:DWELL ";Dwell                     ! DOWNLOAD 1 DWELL TIME
290    OUTPUT @Slot0; "LIST :STEP AUTO"                      ! DWELL-PACED LIST
300    OUTPUT @Slot0;”INITIATE"                              ! ENABLE TRIGGER TO START LIST
310    !
320    ! BEFORE TRIGGERING THE MODULE, DETERMINE IF IT IS READY BY CHECKING FOR
330    ! 'WAITING FOR TRIGGER' (BIT 5 OF THE OPERATION STATUS REGISTER).
340    !
350    ! YOU COULD ELIMINATE THIS STEP BY SIMPLY INSERTING A PAUSE IN THE PROGRAM. HOWEVER, BY
360    ! CHECKING THE INSTRUMENT STATUS, YOU CAN AVOID TIMING PROBLEMS. ALSO, ANY OTHER OPERATIONS
370    ! THAT TAKE TIME WILL GIVE THE MODULE A CHANCE TO COMPLETE PROCESSING.
380    !
390    REPEAT
400    OUTPUT @Slot0;"STATUS:OPERATION:CONDITION?"
410    ENTER @Slot0;Cordition_data
420    UNTIL BIT(Condition_data,5)                           ! TEST FOR BIT 5 = TRUE
430    !
440    ! SEND TRIGGER COMMAND TO START LIST AND GENERATE THE VOLTAGE RAMP
450    !
460    OUTPUT @Slot0;”TRIGGER:IMMEDIATE”                     ! THIS IS AN IMMEDIATE TRIGGER, WHICH IS ALWAYS ACTIVE.
470                                                          ! THEREFORE, IT DOES NOT NEED TO BE SELECTED AS A
TRIGGER SOURCE.
480    !
490    END
```

Figure 3-4. HP BASIC Program Listing for Application #3

Application 4.
Providing Time-Varying Voltages

Overview of application

To burn-in devices using thermal or mechanical cycling/stress, cyclical time-varying voltage is provided by programming a set of voltage and dwell time points that repetitively sequence over time.

For this example, the power module must provide the repetitive waveform shown in Figure 4-1. This time-varying voltage will be applied to a hybrid IC. By continually cycling the voltage from 0 to 7 volts over a 33 second interval, the hybrid is given time to heat up and undergo thermal and mechanical stress as the welds inside the hybrid expand and contract, and then subsequently cool down.

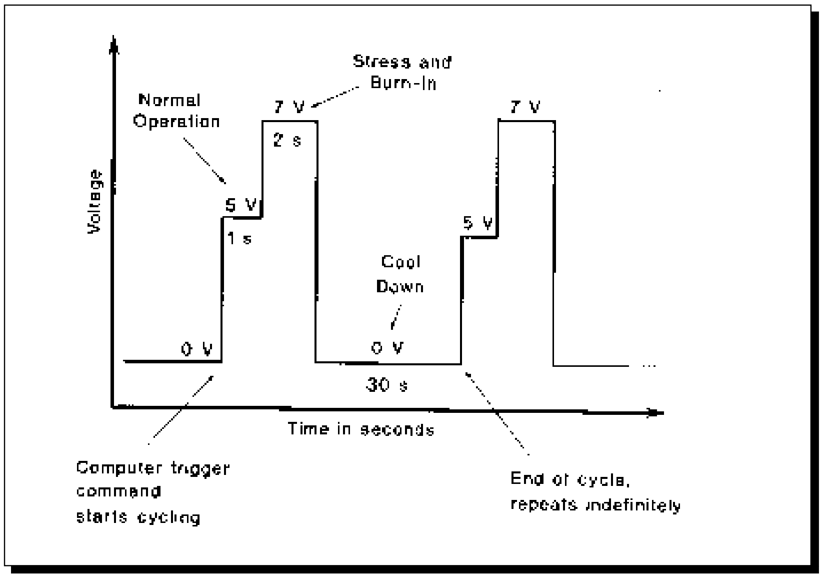


Figure 4-1. Voltage Waveform for Application #4

In addition to generating the cyclical voltage, it is desirable to have the power module notify the computer should the device fail and stop the cycling. Since the module is monitoring test status, the computer is free to perform other tests.

The MPS can address this application using dwell-paced repetitive Lists. This application could be thought of as a simple power arbitrary waveform generator. To get the desired time-varying voltage, you must be able to describe the waveform in 20 discrete voltage points, with each point ranging from 10 ms to 65 seconds. This range of dwell times determines the range of frequencies (or time rate of change) of the voltage waveform to be generated.

Once the waveform has been described, it is downloaded to the module. Upon being triggered, it will repetitively generate the waveform without computer intervention.

The module will also be set up to generate an SRQ and stop the voltage cycling of the hybrid should fail. If the hybrid fails by shorting, the module will go into CC. This change in status will cause the module to protect the DUT by disabling the output, which will stop the test and generate an SRQ. (Open circuit failures will not be detected. Since failures of this type are less likely to have destructive consequences, detection is not required.)

MPS features used

- 20-point voltage List
- Repetitive Lists
- Dwell time
- Dwell-paced Lists
- Generate an SRQ on a change in internal status
- Disable the output on a change in internal status
- Stop the List on a change in internal status
- Trigger on an HP-IB trigger command
- Overcurrent protection
- Active downprogramming

Advantages/benefits of the MPS solution

- By using Lists, the module changes its output voltage automatically, so that the computer is not devoted to reprogramming the output voltage.
- The output can change faster when dwell paced than when the computer must explicitly reprogram each change.
- Overcurrent protection can disable the output before the DUT is damaged.
- By letting each module monitor its status, the CC condition will be responded to faster than if the computer was responsible for stopping the test.
- The sequence is simpler to program (no timing loops).
- By using dwell times, the timing of each point is accurate and repeatable because the computer does not provide timing for the sequence.
- When the output is disabled, the active downprogrammers in the module output can quickly discharge the module's output capacitors and any capacitance in the DUT.

Implementation details

How the MPS implements the sequence

- The module is programmed to List mode.
- The module will execute a dwell-paced List.
- The 3 voltage points are downloaded to the module.
- The 3 dwell times are downloaded to the module.
- To begin the cycling, the module is triggered by the computer.
- The module continuously generates the voltage waveform.
- The module continuously monitors its status.
- If the module goes into CC, the overcurrent protection disables the output.
- The module generates an SRQ when the overcurrent protection occurs.

Module set up

- Set voltage mode to List.
- Download voltage List.
- Download dwell times.
- Set Lists to dwell paced.
- Set Lists to infinitely repeat.
- Enable status monitoring of overcurrent condition.
- Enable overcurrent protection.
- Enable SRQ generation on overcurrent protection occurrence.

Variations on this implementation

1. The module could be set to begin generating the waveform in response to an external or backplane TTL Trigger.
2. The module could be set to generate external triggers for each point in the List. This trigger could be routed to other instruments to synchronize external measurements to the change in voltage. (See application #7) Using this technique, parametric measurements could be made on the device during the thermal cycling.
3. Multiple modules could be programmed to cycle together in response to the computer trigger command.
4. To determine how many times the hybrid was cycled before it failed, you can use the SRQ (that was generated when the hybrid failed and the module went into CC) to timestamp the failure. The elapsed time will give the number of cycles executed.

```
10    ! APPLICATION #4: PROVIDING TIME-VARYING VOLTAGES
20    ! PROGRAM: APP_4
30    !
40    ASSIGN @Slot0 TO 70500                            ! SELECT CODE 7, MAINFRAME ADDRESS 05, SLOT 00
50    !
60    ! INITIALIZE THE MODULE
70    !
80    OUTPUT @Slot0;"*RST;*CLS;STATUS:PRESET"           ! RESET AND CLEAR MODULE
90    OUTPUT @Slot0;"VOLT 0"                            ! START TEST AT 0 V
100   OUTPUT @Slot0;"CURR .1"                           ! SET CURRENT LIMIT
110   OUTPUT @Slot0;"OUTPUT ON"                         ! ENABLE OUTPUT
120   !
130   ! SET UP OVERCURRENT PROTECTION (OCP) AND GENERATE SRQ ON OCP TRIP
140   !
150   OUTPUT @Slot0;"CURRENT:PROTECTION:STATE ON"       ! ENABLE OCP
160   OUTPUT @Slot0;"OUTPUT:PROTECTION:DELAY 0"         ! NO DELAY BEFORE PROTECTION OCCURS
170   OUTPUT @Slot0;"STATUS:QUESTIONABLE:ENABLE 2"      ! ENABLE DETECTION OF OC CONDITION IN THE
180   ! QUESTIONABLE REGISTER, WHERE OC = BIT 1 = VALUE 2.
190   OUTPUT @Slot0;"STATUS:QUESTIONABLE:PTRANSITION 2"          ! ENABLES DETECTION ON POSITIVE TRANSITION,
                                                        I.E., GOING INTO OC.
200   OUTPUT @Slot0;"*SRE 8"                            ! ENABLES THE SERVICE REQUEST REGISTER TO GENERATE AN
210                                                     1 SRQ WHEN ANY EVENT IN THE QUESTIONABLE REGISTER IS
220                                                     ! ASSERTED. THE QUESTIONABLE REGISTER = BIT 3 = VALUE 8.
230   !
240   ! SET UP THE VOLTAGE LIST
250   !
260   OUTPUT @Slot0;"VOLT:MODE LIST"                    ! SET TO GET VOLTAGE FROM LIST
270   OUTPUT @Slot0;"LIST:VOLT 5,7,0"                   ! DOWNLOAD VOLTAGE POINTS
280   OUTPUT @Slot0;"LIST:DWELL 1,2,30"                 ! DOWNLOAD DWELL TIMES
290   OUTPUT @Slot0;"LIST:STEP AUTO"                    ! DWELL-PACED LIST
300   OUTPUT @Slot0;"LIST:COUNT INF"                    ! CONTINUOUSLY REPEAT LIST (INF = INFINITE)
310   OUTPUT @Slot0;"INITIATE"                          ! ENABLE TRIGGER TO START LIST
320   !
330   !
340   ! BEFORE TRIGGERING THE MODULE, DETERMINE IF IT IS READY BY CHECKING FOR
350   ! 'WAITING FOR TRIGGER' (BIT 5 OF THE OPERATION STATUS REGISTER).
360   !
370   ! YOU COULD ELIMINATE THIS STEP BY SIMPLY INSERTING A PAUSE IN THE PROGRAM. HOWEVER, BY
380   ! CHECKING THE INSTRUMENT STATUS, YOU CAN AVOID TIMING PROBLEMS. ALSO, ANY OTHER OPERATIONS
390   ! THAT TAKE TIME WILL GIVE THE MODULE A CHANCE TO COMPLETE PROCESSING.
400   !
410   REPEAT
420   OUTPUT @Slot0;"STATUS:OPERATION:CONDITION?"
430   ENTER @Slot0;Condition_data
440   UNTIL BIT(Condition_data,5)                       ! TEST FOR BIT 5 = TRUE
450   !
460   ! SEND HP-IB TRIGGER COMMAND TO START LIST
470   !
480   OUTPUT @Slot0;"TRIGGER:IMMEDIATE"                 ! THIS IS AN IMMEDIATE TRIGGER, WHICH IS ALWAYS ACTIVE.
490   ! THEREFORE, IT DOES NOT NEED TO BE SELECTED AS A TRIGGER SOURCE.
500   !
510   END
```

Figure 4-2. HP BASIC Programming Listing for Application #4

Application 5.
Providing Time-Varying Current Limiting

Overview of application

To provide current limit protection which varies as a function of time, multiple thresholds on current limit are required. Having multiple thresholds can provide a high limit to protect the DUT during its power-up in-rush with automatic switchover to a lower limit to protect the DUT during its steady state operation.

For this example, the DUT is a printed circuit assembly. This assembly is being tested prior to installation in the end product. The module provides power to the assembly, which will undergo a functional test. The assembly has capacitors on-board, and when power is applied, the in-rush current approaches 4 A. After the capacitors charge, which takes about 500 milliseconds, the steady state current settles to 600 mA. See Figure 5-1.

The MPS can address this application using dwell-paced Lists. In this case, the List will consist of a set of current limits and dwell times, because the voltage will remain constant throughout the test.

Once power has been applied, the first current limit, which provides protection to a shorted DUT while still allowing high current in-rush to occur, will remain in effect for the dwell time. Then the current limit will switch to its next setting in the List. The result is a current limit which changes with time and provides protection as the DUT current requirements drop off to their steady state value. When the dwell time expires for the last current limit in the List, the current limit stays at this value until reprogrammed. Thus, the actual value of the last dwell time is not important. The last current List point would be the current limit for the steady state operation during the test of the DUT. See Figure 5-2 for how the MPS implements this protection.

Throughout List execution, overcurrent protection will be enabled. If at any time the module goes into CC, the output will be disabled, the test stopped, and the DUT protected.

MPS features used

- 20-point current List
- Dwell time
- Dwell-paced Lists
- Disable the output on a change in internal status
- Stop the List on a change in internal status
- Change the voltage on trigger
- Trigger on an HP-IB trigger command
- Overcurrent protection
- Active downprogramming

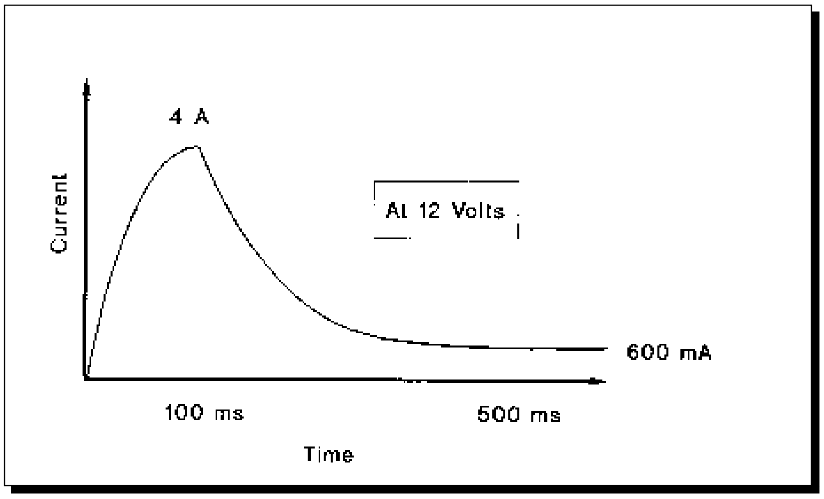


Figure 5-1. Typical DUT Current vs Time

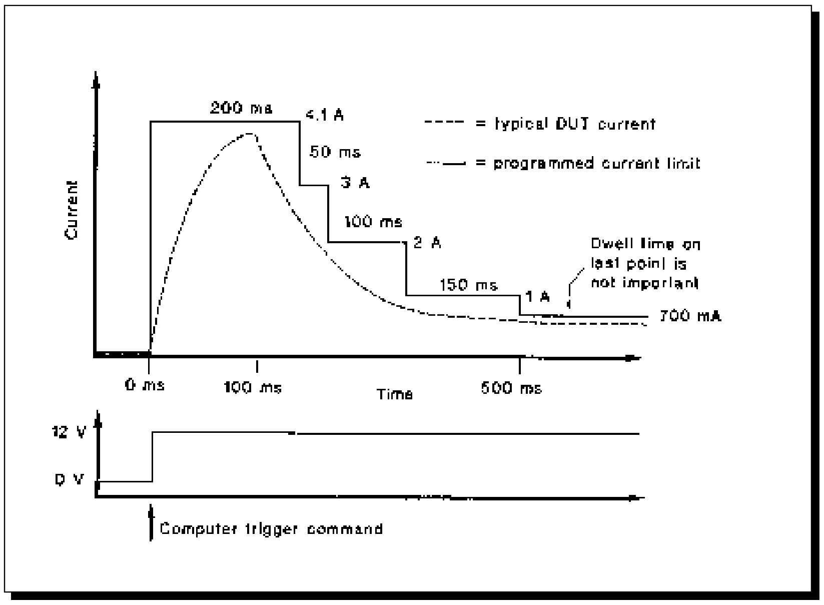


Figure 5-2. Desired Current Limit vs Time

Advantages/benefits of the MPS solution

- By using Lists, the module changes its current limit automatically, so that the computer is not devoted to reprogramming the current limit.
- The output can change faster when dwell paced than when the computer must explicitly reprogram each change.
- Overcurrent protection can disable the output before the DUT is damaged.
- By letting the modules monitor status, the CC condition will be responded to faster than if the computer was responsible for stopping the test.
- The sequence is simpler to program (no timing loops).
- By using dwell times, the timing of each point is accurate and repeatable because the computer does not provide timing for the sequence.
- When the output is disabled, the active downprogrammers in the module output can quickly discharge the module's output capacitors and any capacitance in the DUT.

Implementation details

How the MPS implements the sequence.

- The module is programmed to current List mode.
- The module will execute a dwell-paced current List.
- The current limit List points are downloaded to the module.
- The dwell times are downloaded to the module.
- To begin powering the DUT, the module is triggered by the computer.
- This one trigger causes the current List to begin executing and the voltage to go to its programmed value.
- The module steps through the current limit List.
- The module continuously monitors its status.
- If the modules goes into CC, the overcurrent protection disables the output.

Module set up

- Set the current mode to List.
- Download current List. Download the dwell times.
- Set the List to be dwell paced. Enable overcurrent protection.
- The initial voltage setting is 0 V.
- The module listens for the computer to send a trigger command.
- Upon receipt of the trigger command, the module goes to 12 V.
- Also upon receipt of the trigger command, the module begins executing its current limit List.

Variations on this implementation

1. The module could be set to begin applying power in response to an external or backplane TTL Trigger.
2. Multiple modules could be programmed to cycle together in response to the computer trigger command. Each module could have unique current limits, voltage settings, and dwell times.
3. The module could be set to generate an SRQ if the overcurrent protection disables the output.

4. The module could be set to generate an external or backplane TTL Trigger if the overcurrent protection disables the output.
   5. The in-rush current can be controlled using the current limit settings of the current List. Instead of setting the current limit slightly above 4 A, it could be set at a much lower value. This would limit the in-rush current to the value in the List. It would take longer to charge the capacitors on the assembly, but the inrush condition would be controlled. In this variation, the overcurrent protection could not be used, because you want the module to be in CC.

```
10    ! APPLICATION #5:                                   PROVIDING TIME-VARYING CURRENT LIMITING
20    ! PROGRAM: APP 5
30    !
40    DIM C_limit$[50],Dwell$[50]
50    !
60    C_limit$="4.1, 3.0, 2.0, 1.0, 0.7"                  ! CURRENT LIMIT DATA
70    Dwell$="0.2, 0.05, 0.1, 0.15, 0.01"                 ! DWELL TIME DATA
80    !
90    ASSIGN @Slot0 TO 70500                              ! SELECT CODE 7, MAINFRAME ADDRESS 05, SLOT 00
100   !
110   OUTPUT @Slot0;"*RST;*CLS;STATUS:PRESET"             ! RESET AND CLEAR MODULE
120   OUTPUT @Slot0;"VOLT 0"                              ! START TEST AT 0 V
130   OUTPUT @Slot0;"OUTPUT ON"                           ! ENABLE OUTPUT
140   OUTPUT @Slot0; "CURRENT :PROTECTION : STATE ON"     ! ENABLE OCP
150   OUTPUT @Slot0;"OUTPUT:PROTECTION:DELAY 0"           ! NO DELAY BEFORE PROTECTION OCCURS
160   OUTPUT @Slot0;"CURRENT:MODE LIST"                   ! SET TO GET CURRENT FROM LIST
170   OUTPUT @Slot0;"LIST:CURRENT ";C_limit$              ! DOWNLOAD CURRENT POINTS
180   OUTPUT @Slot0;"LIST:DWELL ";Dwell$                  ! DOWDLOAD DWELL TIMES
190   OUTPUT @Slot0;"LIST:STEP AUTO"                      ! DWELL-PACED LIST
200   OUTPUT @Slot0;"VOLT:TRIGGERED 12"                   ! GO TO 12 V WHEN TRIGGERED
210   OUTPUT @Slot0;"INITIATE"                            ! ENABLE TRIGGER TO START LIST AND APPLY 12 V
220   !
230   ! BEFORE TRIGGERING THE MODULE, DETERMINE IF IT IS READY BY CHECKING FOR
240   ! 'WAITING FOR TRIGGER' (BIT 5 OF THE OPERATION STATUS REGISTER).
250   !
260   ! YOU COULD ELIMINATE THIS STEP By SIMPLY INSERTING A PAUSE IN THE PROGRAM. HOWEVER, BY
270   ! CHECKING THE INSTRUMENT STATUS, YOU CAN AVOID TIMING PROBLEMS. ALSO, ANY OTHER OPERATIONS
280   ! THAT TAKE TIME WILL GIVE THE MODULE A CHANCE TO COMPLETE PROCESSING.
290   !
300   REPEAT
310   OUTPUT @Slot0;"STATUS:OPERATION:CONDITION?"
320   ENTER @Slot0;Condition_data
330   UNTIL BIT(Condition_data,5)                         ! TEST FOR BIT 5 = TRUE
340   !
350   ! SEND HP-IB TRIGGER COMMAND TO START LIST AND APPLY 12 V
360   !
370   OUTPUT @Slot0;"TRIGGER:IMMEDIATE"                   ! THIS IS AN IMMEDIATE TRIGGER, WHICH IS
380                                                       ! ALWAYS ACTIVE. THEREFORE, IT DOES NOT NEED
390   END                                                 ! TO BE SELECTED AS A TRIGGER SOURCE.
```

Figure 5-3. HP BASIC Program Listing for Application #5

Application 6.
Output Sequencing Paced by the Computer

Overview of application

When performing bias supply margin testing, throughput can be maximized by eliminating the command processing time associated with reprogramming all outputs for each set of limit conditions. Instead, multiple sets of bias limit conditions can be downloaded to the power modules during test system initialization. During the testing, the computer can use a single command to simultaneously signal all power modules to step through each test condition.

In this example, the DUT requires + 5 V and + 12 V. The DUT is tested to ensure proper operation at marginal supply voltages. The margin specified is ± 5% of nominal voltage. At each of the combinations given below, the computer first sets up the three modules and makes a measurement on the DUT. The combinations to be tested are:

Nominal 6 V	Nominal + 12 V	Nominal -12 V
4.75 V	12 V	- 12 V
5 V	12 V	- 12 V
5.25 V	12 V	- 12 V
5V	11.4 V	- 12 V
5V	12.6 V	-12 V
5V	12V	-11.4V
5 V	12 V	- 12.6 V

When conducting this test, the modules will need to be reprogrammed 21 times and seven measurements made. The command processing time could slow down this test.

The MPS can be used to increase throughput. By downloading all of the combinations into the three modules, each setting can be quickly stepped through by triggering all modules to change to their next voltage setting and then taking a measurement from the DUT. This permits testing without command processing overhead.

MPS features used

- 20-point voltage List
- Trigger-paced Lists
- Trigger in/out from MPS mainframe backplane TTL Trigger
- Trigger on an HP-IB trigger command

Advantages/benefits of the MPS solution

- By using Lists, the module changes its voltage without delays due to processing the command to change the output voltage.
- By using triggers, all three outputs can be changed with one command.
- The computer loop to change the settings and take a measurement is simplified, because you do not have to explicitly reprogram each module output. Instead, the loop becomes "Trigger" and "Measure".

Implementation details

How the MPS implements the sequence

- The following steps are performed for each point in the List:
- The computer sends a trigger command to the first module.
- The first module simultaneously sends a backplane TTL Trigger to the other two modules and goes to its next List point.
- The second module receives the backplane TTL Trigger and immediately goes to its next List point.
- The third module receives the backplane TTL Trigger, immediately goes to its next List point.
- The computer gets a measurement from the measurement instrument.

MPS set up

Module in slot 0:

- The module is connected to + 5 V on the DUT.
- The initial voltage setting is 0 V.
- Set the voltage mode to List.
- Download the voltage List.
- Set the List to be trigger paced.
- The module listens for the computer to send a trigger command.
- Upon receipt of the trigger command, the module outputs its next List point.
- Also upon receipt of the trigger command, the module generates a backplane TTL Trigger.

Module in slot 1:

- The module is connected to + 12 V on the DUT.
- The initial voltage setting is 0 V.
- Set the voltage mode to List.
- Download the voltage List.
- Set the List to be trigger paced.
- The module listens for a backplane TTL Trigger.
- Upon receipt of a trigger, the module goes to its next List point.

Module in slot 2:

- The module is connected to -12 V on the DUT.
- The initial voltage setting is 0 V.
- Set the voltage mode to List.
- Download the voltage List.
- Set the List to be trigger paced.
- The module listens for a backplane TTL Trigger.
- Upon receipt of a trigger, the module goes to its next List point.

Variations on this implementation

1. A current List could also have been executed by the module so that for each voltage point, a corresponding current limit could be programmed.
2. Overcurrent protection could be enabled to protect a faulty DUT.
3. The module could generate an SRQ when it finishes changing voltage for each point in the List based on the STC (Step Completed) status bit, which indicates when the module has completed executing the next point in the List. The SRQ could tell the computer to get a measurement from the measurement instrument.
4. The module could be told to output its next List point in response to an external or backplane TTL Trigger. (See next application.)

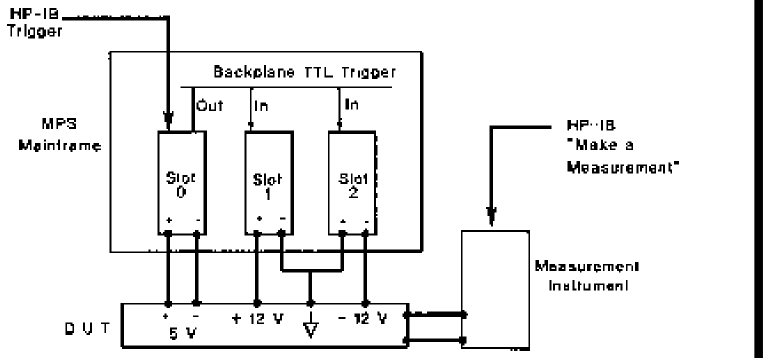


Figure 6-1. Block Diagram Application #6

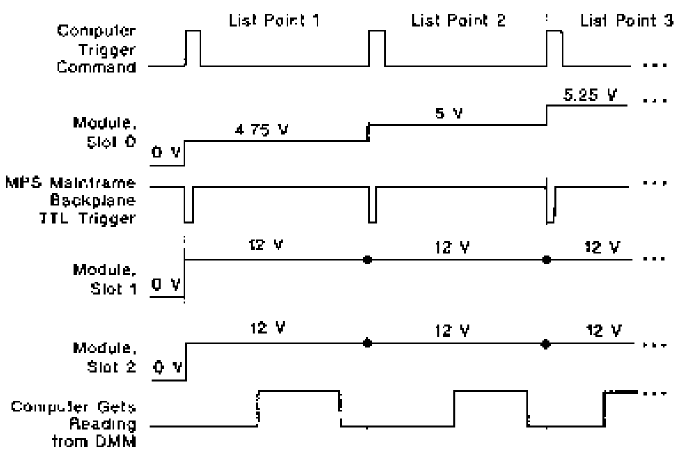


Figure 6-2. Diagram of Application #6

```
10     ! APPLICATION #6: OUTPUT SEQUENCING PACED BY THE COMPUTER
20     ! PROGRAM: APP_6
30     !
40     DIM Plus_5v$[50],Plus_12v$[50],Minus_12v$[50]
50     !
60     Plus_5v$="4.75, 5, 5.25, 5, 5, 5, 5"                    ! THESE ARE THE BIAS
70     Plus_12v$="12, 12, 12, 11.4, 12.6, 12, 12"              ! SUPPLY LIMIT CONDITIONS
80     Minus 12v$="12, 12, 12, 12, 12, 11.4, 12.6"             ! TO BE TESTED
90     !
100    Num_test_steps=7                                        ! NUMBER OF BIAS SUPPLY LIMIT COMBINATIONS
110    Dwell=.010                                              ! SECONDS OF DWELL TIME
120    !
130    ASSIGN @Slot0 TO 70500                                  ! SELECT CODE 7, MAINFRAME ADDRESS 05, SLOT 00
140    ASSIGN @Slot1 TO 70501                                  ! SELECT CODE 7, MAINFRAME ADDRESS 05, SLOT 01
150    ASSIGN @Slot2 TO 70502                                  ! SELECT CODE 7, MAINFRAME ADDRESS 05, SLOT 02
160    !
170    ! SET UP MODULE IN SLOT 0 AS +5 V BIAS SUPPLY ---------------------
180    !
190    OUTPUT @Slot0;"*RST;*CLS;STATUS:PRESET"                 ! RESET AND CLEAR MODULE
200    OUTPUT @Slot0;"VOLT 0"                                  ! START AT 0 V
210    OUTPUT @Slot0;"OUTPUT ON"                               ! ENABLE OUTPUT
220    OUTPUT @Slot0;"VOLTAGE:MODE LIST"                       ! SET TO GET VOLTAGE FROM LIST
230    OUTPUT @Slot0;"LIST:VOLTAGE ";Plus_5v$                  ! DOWNLOAD VOLTAGE LIST POINTS
240    OUTPUT @Slot0;"LIST:DWELL";Dwell                        ! DOWNLOAD 1 DWELL TIME (ASSUMES SAME FOR ALL POINTS)
250    OUTPUT @Slot0;"LIST:STEP ONCE"                          ! EXECUTE 1 LIST POINT PER TRIGGER
260    OUTPUT @Slot0;"TRIGGER:SOURCE BUS"                      ! TRIGGER SOURCE IS HP-IB 'BUS'
270    OUTPUT @Slot0;"OUTPUT:TTLTRG:SOURCE BUS"                ! GENERATE BACKPLANE TTL TRIGGER WHEN HP-IB 'BUS' TRIGGER
IS RECEIVED
280    OUTPUT @Slot0;"OUTPUT:TTLTRG:STATE ON"                  ! ENABLE TTL TRIGGER DRIVE
290    OUTPUT @Slot0;"INITIATE"                                ! ENABLE RESPONSE TO TRIGGER
300    !
310    ! SET UP MODULE IN SLOT 1 AS +12 V BIAS SUPPLY ----------------------
320    !
330    OUTPUT @Slot1;"*RST;*CLS;STATUS:PRESET"                 ! RESET AND CLEAR MODULE
340    OUTPUT @Slot1;"VOLT 0"                                  ! START AT 0 V
350    OUTPUT @Slot1;"OUTPUT ON"                               ! ENABLE OUTPUT
360    OUTPUT @Slot1;"VOLT:MODE LIST"                          ! SET TO GET VOLTAGE FROM LIST
370    OUTPUT @Slot1;"LIST:VOLTAGE ";Plus_12v$                 ! DOWNLOAD VOLTAGE LIST POINTS
380    OUTPUT @Slot1;"LIST:DWELL";Dwell                        ! DOWNLOAD 1 DWELL TIME (ASSUMES SAME FOR ALL POINTS)
390    OUTPUT @Slot1;"LIST:STEP ONCE"                          ! EXECUTE 1 LIST POINT PER TRIGGER
400    OUTPUT @Slot1;"TRIGGER:SOURCE TTLTRG"                   ! TRIGGER SOURCE IS BACKPLANE TTL TRIGGER
410    OUTPUT @Slot1;"INITIATE"                                ! ENABLE RESPONSE TO TRIGGER
420    !
430    ! SET UP MODULE IN SLOT 2 AS -12 V BIAS SUPPLY ---------------------
440    !
450    OUTPUT aSlot2;"*RST;*CLS;STATUS:PRESET"                 ! RESET AND CLEAR MODULE
460    OUTPUT @Slot2;"VOLT 0"                                  ! START AT 0 V
470    OUTPUT @Slot2;"OUTPUT ON"                               ! ENABLE OUTPUT
480    OUTPUT @Slot2;"VOLT:MODE LIST"                          ! SET TO GET VOLTAGE FROM LIST
490    OUTPUT @Slot2;"LIST:VOLTAGE ";Minus_12v$                ! DOWNLOAD VOLTAGE LIST POINTS
500    OUTPUT @Slot2;"LIST:DWELL";Dwell                        ! DOWNLOAD 1 DWELL TIME (ASSUMES SAME FOR ALL POINTS)
510    OUTPUT @Slot2;"LIST:STEP ONCE"                          ! EXECUTE 1 LIST POINT PER TRIGGER
520    OUTPUT @Slot2;"TRIGGER:SOURCE TTLTRG"                   ! TRIGGER SOURCE IS BACKPLANE TTL TRIGGER
530    OUTPUT @Slot2;"INITIATE"                                ! ENABLE RESPONSE TO TTL TRIGGER
540    !
```

```
550    ! BEFORE TRIGGERING THE MODULES, DETERMINE IF THE MODULES ARE READY BY CHECKING FOR
560    ! 'WAITING FOR TRIGGER' (BIT 5 OF THE OPERATION STATUS REGISTER). IF THE LAST MODULE PROGRAMMED
570    ! IS READY THEN SO ARE THE OTHERS, SO JUST CHECK SLOT 2.
580    !
590    ! YOU COULD ELIMINATE THIS STEP BY SIMPLY INSERTING A PAUSE IN THE PROGRAM. HOWEVER, BY
600    ! CHECKING THE INSTRUMENT STATUS, YOU CAN AVOID TIMING PROBLEMS. ALSO, ANY OTHER OPERATIONS
610    ! THAT TAKE TIME WILL GIVE THE MODULES A CHANCE TO COMPLETE PROCESSING.
620    !
630    REPEAT
640    OUTPUT @Slot2;"STATUS:OPERATION:CONDITION? "
650    ENTER @Slot2;Condition_data
660    UNTIL BIT(Condition_data,5)                        ! TEST FOR BIT 5 = TRUE
670    !
680    ! GENERATE A TRIGGER AND MAKE A MEASUREMENT FOR EACH TEST CONDITION
690    !
700    FOR Loop_count=1 TO Num_test_steps
710    OUTPUT @Slot0;"*TRG"                               ! SEND HP-IB BUS TRIGGER
720    GOSUB Get_measurement
730    NEXT Loop_count
740    !
750    STOP
760    !
770    Get_measurement:                                   !
780    !
790    ! THIS IS JUST TO SHOW YOU WHERE YOU WOULD ADD CODE TO GET DATA FROM THE MEASUREMENT INSTRUMENT
800    ! THE MEASUREMENT MUST TAKE LONGER THAN THE PROGRAMMED DWELL TIME OR YOU WILL MISS TRIGGERS.
810    !
820    WAIT .1
830    !
840    RETURN
850    !
860    END
```

Figure 6-3. HP BASIC Program Listing for Application #6

Application 7. Output Sequencing without Computer Intervention

Overview of application

When characterizing devices, the DUT's performance is measured over a range of power supply voltages. This test can be performed without computer intervention by using hardware signals from the measurement instrument to cause the power module to sequence to the next voltage in a preprogrammed List. By buffering these readings in the measurement instrument, the entire test can be executed without computer involvement. For characterizations that require long measurement times, the computer is free to do other tasks. For characterizations that must execute at hardware speeds, the computer is not involved and will not slow down the test loop.

In this example, the power module must apply 8 to 14 volts (in 13 0.5-volt increments) to an automotive engine sensor. The module varies the bias voltage to the engine sensor and the sensor's output is measured to characterize its performance over the range of possible “battery voltages”. The sensor output is measured by a DMM that has an internal buffer and stores each reading.

By combining Lists and trigger capabilities, the MPS can be used to address this application. The module can be programmed to use its triggering capabilities to the fullest extent. Each time the module executes the next step in its List and changes voltage, the module will generate an external trigger. The external trigger will cause the DMM, equipped with an external trigger input, to take and store a reading. The DMM, also equipped with a “Measurement Complete” output, sends its output trigger signal to the module to tell the module to go to its next List point. Effectively, the module and the DMM “handshake”, so that the two function at hardware speeds without computer intervention.

When the test is complete, either device can signal the computer to get the data from the DMM. For the purpose of this example, the module will generate an SRQ when the last List point has been executed. This is indicated by the OPC (Operation Complete) bit in the status register.

Another detail that needs attention is timing. The DUT may require some settling time before the DMM is told to take a reading. The module's dwell time can be used to do this. The STC (Step Complete) status signal indicates when the point has been executed and its dwell time has expired. The dwell time is programmed to be the engine sensor's settling time. The external trigger is generated when STC is asserted. Thus, the DMM will not be triggered until the dwell time has expired and the sensor's output has settled.

This type of self-paced test execution is useful in two situations. When the test must execute very fast, there is no time for the computer to be involved in each iteration of the test loop. Therefore, the test must execute without computer intervention. The second situation is when the test is very long. For example, if the measurement instrument took 1 minute to make each measurement, the test would take 13 minutes to execute. The computer is not used efficiently if it is idle while waiting for each measurement loop, so it would be best to have the computer executing another task. Without self-pacing, you would need to develop interrupt driven software that stops every 1 minute to take a reading. By letting the module and the DMM run on their own, code development is much simpler and computer resources are used more efficiently.

MPS features used

- 20-point voltage List
- Dwell time
- Trigger-paced Lists
- Generate an SRQ on a change in internal status
- Generate a trigger on a change in internal status
- Trigger in/out from MPS mainframe backplane TTL Trigger
- Trigger on an HP-IB trigger command

Advantages/benefits of the MPS solution

- The entire test executes without computer involvement, the command processing time is eliminated from the test loop.
- The entire test executes without computer involvement, so the computer can perform other tasks while the test executes.
- Software development is simplified; you do not need to write a test loop because the module and the DMM are running on their own.
- By using dwell times, the trigger out signal can be sent at the correct time, which permits the DUT to settle before a reading is taken.

Implementation details

How the MPS implements the sequence

- The module listens for the computer to send a trigger command.
- Upon receipt of the trigger command, the module outputs its first List point.
- After the dwell time expires, the STC is asserted and the module generates an external trigger.
- The DMM receives the external trigger, takes and stores a reading.
- The DMM generates a “Measurement Complete” output signal when it's done.
- The module receives the DMM output signal as an external trigger in.
- Also upon receipt of the trigger in, the module outputs its next List point.
- The process repeats for each List point.
- After the last List point has executed, the module generates SRQ, telling the computer the test has completed.

MPS set up

Set the voltage mode to List.

- Download the voltage List.
- Download the dwell time List.
- Set the List to be trigger paced.
- Set the trigger source to external trigger. (Note that the computer trigger command "TRIGGER:IMMEDIATE" is always active, even if the external trigger is the selected source.)
- Set the module to generate a backplane TTL Trigger on STC. This backplane TTL Trigger drives external trigger out. Set the module to generate SRQ on OPC.

Variations on this implementation

1. A current List could also have been executed by the module so that for each voltage point, a corresponding current limit could be programmed.
2. Overcurrent protection could be enabled to protect a faulty engine sensor.
3. If the DMM does not have an internal buffer, the computer could take a reading on each iteration of the test loop (see previous application).

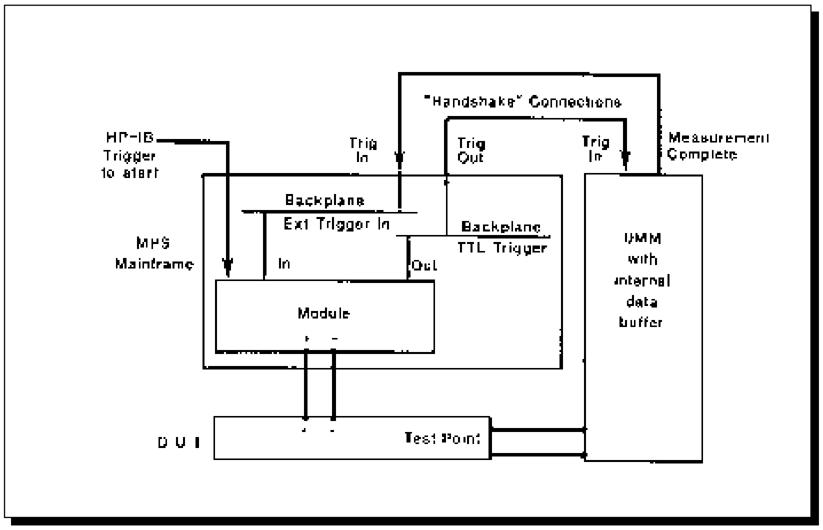


Figure 7-1. Block Diagram of Application #7

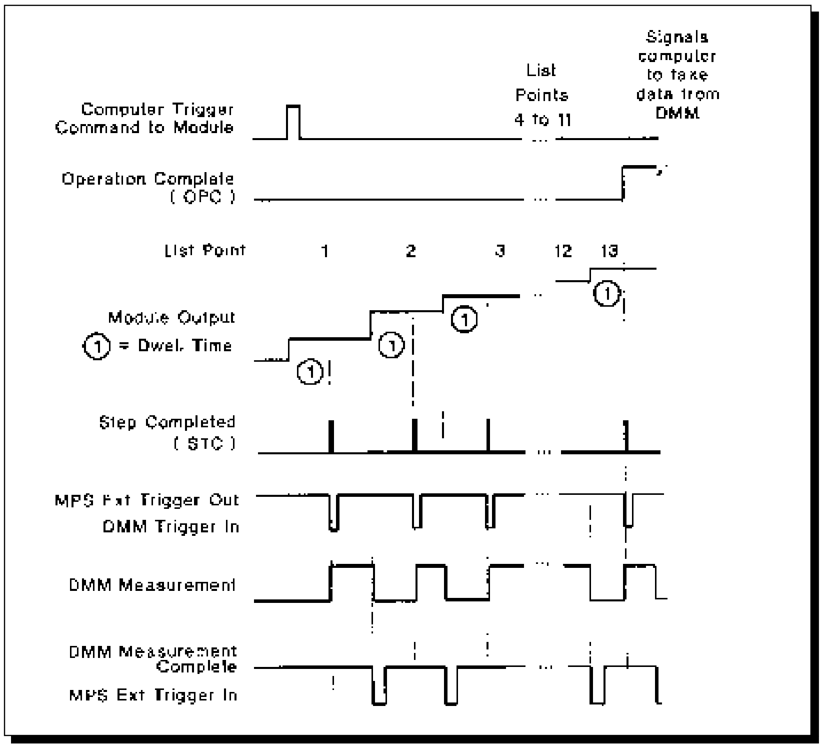


Figure 7-2. Timing Diagram of Application #7

```
10   ! APPLICATION #7: OUTPUT SEQUENCING WITHOUT COMPUTER INTERVENTION
20   ! PROGRAM: APP_7
30   !
40   ASSIGN @Slot0 TO 70500                        ! SELECT CODE 7, MAINFRAME ADDRESS 05, SLOT 00
50   !
60   DIM Vlist$[80]
70   Vlist$="8, 8.5, 9, 9.5, 10, 10.5, 11, 11.5, 12, 12.5, 13, 13.5, 14"       ! VOLTAGE LIST POINTS
80   !
90   OUTPUT @Slot0;"*RST;*CLS;STATUS:PRESET"        ! RESET AND CLEAR MODULE
100  OUTPUT @Slot0;"VOLT 0"                        ! START AT 0 V
110  OUTPUT @Slot0;"CURR 1"                        ! SET CURRENT LIMIT
120  OUTPUT @Slot0;"OUTPUT ON"                     ! ENABLE OUTPUT
130  OUTPUT @Slot0;"VOLT:MODE LIST"                ! SET TO GET VOLTAGE FROM LIST
140  OUTPUT @Slot0;"LIST:VOLT ";Vlist$             ! DOWNLOAD VOLTAGE LIST POINTS
150  OUTPUT @Slot0;"LIST:DWELL .050"               ! DOWNLOAD 1 DWELL POINT (ASSUMES SAME FOR ALL POINTS)
160                                                ! USE A 50 ms SETTLING TIME AS THE DWELL TIME
170  OUTPUT @Slot0;"LIST:STEP ONCE"                ! EXECUTE 1 POINT PER TRIGGER
180  !
190  OUTPUT @Slot0;n*ESE 1"                        ! ENABLES DETECTION OF OPC IN THE STANDARD EVENT REGISTER.
200                                                ! OPC = BIT 0 = VALUE 1 OF THE STANDARD EVENT REGISTER.
210  OUTPUT @Slot0;"*SRE 32"                       ! ENABLES THE SERVICE REQUEST REGISTER TO GENERATE AN SRQ
WHEN
220                                                ! ANY EVENT IN THE STANDARD EVENT REGISTER IS ASSERTED.
230                                                ! THE STANDARD EVENT REGISTER = BIT 5 = VALUE 32.
240  !
250  OUTPUT @Slot0;"OUTPUT:TTLTRG:STATE ON"        ! ENABLE BACKPLANE TTL TRIGGER DRIVE
260  OUTPUT @Slot0;"OUTPUT:TTLTRG:SOURCE LINK"     ! WHEN THE MODULE INDICATES STC (STEP COMPLETED),
270  OUTPUT @Slot0;"OUTPUT:TTLTRG:LINK 'STC'"      ! GENERATE A BACKPLANE TTL TRIGGER
280  OUTPUT @Slot0;"TRIGGER:SOURCE EXTERNAL"       ! USE EXTERNAL TRIGGER IN BNC AS TRIGGER SOURCE
290  OUTPUT @Slot0;"INITIATE"                      ! ENABLE RESPONSE TO TRIGGER
300  OUTPUT @Slot0;"*OPC"                          ! TELLS MODULE TO ASSERT OPC (OPERATION COMPLETE)
310                                                ! WHEN IT COMPLETES THE LIST. OPC GENERATES SRQ.
320  !
330  ON INTR 7 GOSUB Srq_handler                   ! ENABLE SRQ INTERRUPT AND
340  ENABLE INTR 7;2                               ! IDENTIFY HANDLER SUBROUTINE
350  !
360  ! BEFORE TRIGGERING THE MODULE, DETERMINE IF IT IS ARE READY BY CHECKING FOR
370  ! 'WAITING FOR TRIGGER' (BIT 5 OF THE OPERATION STATUS REGISTER).
380  !
390  ! YOU COULD ELIMINATE THIS STEP BY SIMPLY INSERTING A PAUSE IN THE PROGRAM. HOWEVER, BY
400  ! CHECKING THE INSTRUMENT STATUS, YOU CAN AVOID TIMING PROBLEMS. ALSO, ANY OTHER OPERATIONS
410  ! THAT TAKE TIME WILL GIVE THE MODULE A CHANCE TO COMPLETE PROCESSING.
420  !
430  REPEAT
440  OUTPUT @Slot0; "STATUS :OPERATION :CONDITION?"
450  ENTER @Slot0;Condition_data
460  UNTIL BIT(Condition_data,5)                   ! TEST FOR BIT 5 = TRUE
470  !
480  ! BEGIN THE SELF-PACED TEST LOOP BY TRIGGERING THE MODULE TO START THE LIST
440  !
500  OUTPUT @Slot0;"TRIGGER:IMMEDIATE"             ! THIS IS AN IMMEDIATE TRIGGER, WHICH IS ALWAYS ACTIVE.
510  ! IT DOES NOT NEED TO BE SELECTED AS TRIGGER SOURCE.
520  !
530  GOTO 530                                      ! IDLE IN LOOP WAITING FOR SRQ OR GO DO OTHER TASKS
540  !
550 Srq_handler:                                   !
560  !
570  ! ADD LINES HERE TO READ THE DATA BUFFER FROM THE DMM
580  !
590  END
```
Figure 7-3. HP BASIC Program Listing of Application #7

Appendix

This appendix contains program listings translated into the following DOS-compatible languages and HP-IB interfaces:

- GWBASIC and the HP 61062/82990/82335A HP-IB Command Library for MS-DOS
- GWBASIC and the National Instruments GPIB-PC Interface Card
- Microsoft C and the HP 61062/82990/82335A HP-IB Command Library for MS-DOS
- Microsoft C and the National Instruments GPIB-PC Interface Card

Each program is translated from the HP BASIC listing found in application #3. This example program was chosen as representative of all application programs because it shows how to:

- Configure the interface card
- Address the power module
- Write strings to the power module
- Write real arrays to the power module
- Receive real numbers from the power module

The six other application programs all use a subset of the above functions.

```
1         ´ MERGE "SETUP.BAS" AS DESCRIBED IN YOUR HP-IB COMMAND LIBRARY MANUAL
2         ´
1000    ´ APPLICATION #3: CONTROLLING VOLTAGE RAMP UP AT TURN ON
1010    ´ FOR GWBASIC AND THE HP 61062/82990/82335A HP-IB COMMAND LIBRARY
1020    ´ PROGRAM: HP3.BAS
1030    ´
1040    OPTION BASE 1
1050    INTERFACE = 7                                    ´SELECT CODE OF THE HP-IB CARD
1060    SLOT0 = 705001!                                  ´SELECT CODE 7, MAINFRAME ADDRESS 05, SLOT 00
1070    CR.LF$ = CHR$(13) + CHR$(10) '                   ´ CARRIAGE RETURN + LINE FEED = END OF LINE
TERMINATION
1080    DIM VSTEP(20)                                    ´ ARRAY TO HOLD THE VOLTAGE RAMP STEPS
1090    NUM.POINTS = 20                                  ´ NUMBER OF POINTS IN THE VOLTAGE RAMP ARRAY
1100    VSTART = 2                                       ´ START VOLTAGE FOR RAMP
1110    VSTOP = 10                                       ´ STOP VOLTAGE FOR RAMP
1120    RAMPTIME = .5                                    ´ TIME IN SECONDS TO CHANGE FROM VSTART TO VSTOP
1130    DWELL = RAMPTIME / 19                            ´ DWELL TIME FOR EACH POINT
1140    ´
1150    ´ SINCE THE OUTPUT STAYS AT THE LAST VOLTAGE POINT AFTER ITS DWELL TIME EXPIRES, THE DWELL TIME OF THE
1160    ´ LAST POINT IS NOT PART OF THE TRANSITION TIME. THEREFORE, DIVIDE THE TOTAL TIME BY 19 POINTS, NOT 20.
1170    ´ YOU WANT THE SAME DWELL TIME FOR EVERY POINT IN THE LIST, SO YOU NEED TO DOWNLOAD ONLY 1 DWELL
TIME.
1180    ´
1190    FOR I=1 TO 20
1200    VSTEP(I) = VSTART + ((( VSTOP - VSTART ) / 20 ) * I )     ´ CALCULATES VOLTAGE LIST POINTS
1210    NEXT I
1220    ´
1230    ´ NOTE REGARDING HP-IB READ/WRITE TERMINATIONS:
1240    ´
1250    ´                                                THE DEFAULT MODE OF THE INTERFACE CARD IS THAT EOI IS
ENABLED AND THE READ/WRITES TERMINATE
1260    ´                                                ON CARRIAGE RETURN/LINE FEED. THE MODULE
TERMINATES ON EITHER EOI OR LINE FEED, SO THE
1270    ´                                                DEFAULT SETTINGS OF THE CARD ARE SUFFICIENT.
1280    ´
1290    CMD$ = "*RST;*CLS;STATUS:PRESET"                 ´ RESET AND CLEAR MODULE
1300    L = LEN( CMD$ )
1310    CALL IOOUTPUTS( SLOT0, CMD$, L )
1320    IF PCIB.ERR<>0 THEN ERROR PCIB.BASERR
1330    ´
1340    CMD$ = "VOLT " + STR$( VSTART )                  ´ START RAMP AT VSTART. USE NUMBER TO STRING
1350    L = LEN( CMD$ )                                  ´ CONVERSION TO SEND REAL NUMBERS OVER THE BUS
1360    CALL IOOUTPUTS( SLOT0, CMD$, L )                 ´ AS PART OF THE COMMAND STRING.
1370    IF PCIB.ERR<>0 THEN ERROR PCIB.BASERR
1380    ´
1390    CMD$ = "CURR .1"
1400    L = LEN( CMD$ )
1410    CALL IOOUTPUTS( SLOT0, CMD$, L )
1420    IF PCIB.ERR<>0 THEN ERROR PCIB.BASERR
1430    ´
1440    CMD$ = "OUTPUT ON"                               ´ ENABLE OUTPUT
1450    L = LEN( CMD$ )
1460    CALL IOOUTPUTS( SLOT0, CMD$, L )
1470    IF PCIB.ERR<>0 THEN ERROR PCIB.BASERR
1480    ´
1490    CMD$ = "VOLT:HODE LIST"                          ´ SET TO GET VOLTAGE FROM LIST
1500    L = LEN( CMD$ )
1510    CALL IOOUTPUTS( SLOT0, CMD$, L )
1520    IF PCIB.ERR<>0 THEN ERROR PCIB.BASERR
1530    ´
1540    ´ SENDING THE VOLTAGE DATA POINTS REQUIRES TWO STEPS USING THE HP-IB COMMAND LIBRARY. THE
INSTRUCTION CONTAINS BOTH
1550    ´ STRING DATA AND A REAL ARRAY. FIRST, SEND THE STRING DATA COMMAND HEADER "LIST:VOLT" TO THE MODULE
USING IOOUTPUTS.
1560    ´ THEN, SEND THE REAL ARRAY USING IOOUTPUTA. HOWEVER, YOU MUST INHIBIT THE EOI AND END-OF-LINE
TERMINATOR AFTER THE
1570    ´ IOOUTPUTS COMMAND OR THE MODULE WILL STOP TAKING DATA. THEN RE-ENABLE THEM TO TERMINATE THE
IOOUTPUTA.
1580    ´
```

```
1590   EOI.STATE = 0                                    ´ TURN OFF EOI
1600   CALL IOEOI( INTERFACE, EOI.STATE )
1610   IF PCIB.ERR<>0 THEN ERROR PCIB.BASERR
1620   ´
1630   END.OF.LINE = 0                                  ´ TURN OFF END-OF-LINE TERMINATION
```
41

```
1640   CALL IOEOL( INTERFACE, CR.LF$, END.OF.LINE )
1650   IF PCIB.ERR <>0 THEN ERROR PCIB.BASERR
1660   ´
1670   CMD$ = "LIST:VOLT "
1680   L = LEN( CMD$ )
1690   CALL IOOUTPUTS( SLOT0, CMD$, L )                 ´ SEND THE VOLTAGE HEADER (STRING) ---
1700   IF PCIB.ERR<>0 THEN ERROR PCIB.BASERR
1710   ´
1720   EOI.STATE = 1                                    ´ TURN ON EOI
1730   CALL IOEOI( INTERFACE, EOI.STATE )
1740   IF PCIB.ERR<>0 THEN ERROR PCIB.BASERR
1750   ´
1760   END.OF.LINE = LEN (CR.LF$ )                      ´ TURN ON END-OF-LINE TERMINATION
1770   CALL IOEOL( INTERFACE, CR.LFS , END.OF.LINE )
1780   IF PCIB.ERR<>0 THEN ERROR PCIB.BASERR
1790   ´
1800   CALL IOOUTPUTA( SLOT0, VSTEP(1), NUM.POINTS )     ´ DOWNLOAD THE VOLTAGE POINTS (ARRAY)
1810   IF PCIB.ERR <> 0 THEN ERROR PCIB.BASERR
1820   ´
1830   CMD$ = “LIST:DWELL " + STR$( DWELL )             ´ DOWNLOAD 1 DWELL TIME. USE NUMBER TO STRING
1840   L = LEN( CMD$ )                                  ´ CONVERSION TO SEND REAL NUMBERS OVER THE BUS
1850   CALL IOOUTPUTS( SLOT0, CMD$, L )                 ´ AS PART OF THE COMMAND STRING.
1860   IF PCIB.ERR<>0 THEN ERROR PCIB.BASERR
1870   ´
1880 CMD$ = "LIST:STEP AUTO"                            ´ DWELL-PACED LIST
1890   L = LEN( CMD$ )
1900   CALL IOOUTPUTS( SLOT0, CMD$, L )
1910   IF PCIB.ERR<>0 THEN ERROR PCIB.BASERR
1920   ´
1930   CMD$ = "INITIATE"                                ´ ENABLE TRIGGER TO START LIST
1940   L = LEN( CMD$ )
1950   CALL IOOUTPUTS( SLOT0, CMD$, L )
1960   IF PCIB.ERR<>0 THEN ERROR PCIB.BASERR
1970   ´
1980   ´ BEFORE TRIGGERING THE MODULE, DETERMINE IF IT IS READY BY CHECKING FOR
1990   ´ ´WAITING FOR TRIGGER´ (BIT 5 OF THE OPERATION STATUS REGISTER).
2000   ´
2010   ´ YOU COULD ELIMINATE THIS STEP BY SIMPLY INSERTING A PAUSE IN THE PROGRAM. HOWEVER, BY
2020   ´ CHECKING THE INSTRUMENT STATUS, YOU CAN AVOID TIMING PROBLEMS. ALSO, ANY OTHER OPERATIONS
2030   ´ THAT TAKE TIME WILL GIVE THE MODULE A CHANCE TO COMPLETE PROCESSING.
2040   ´
2050   CONDITION . DATA = 0
2060   ´
2070   WHILE ( ( CONDITION.DATA AND 32 ) <> 32 )        ' CONTINUE TO LOOP UNTIL BIT 5 (VALUE 32) = TRUE
2080   CMD$ = “STATUS:OPERATION:CONDITION?”
2090   L = LEN( CMD$ )
2100   CALL IOOUTPUTS( SLOT0, CMD$, L )
2110   IF PCIB.ERR<>0 THEN ERROR PCIB.BASERR
2120   CALL IOENTER( SLOT0, CONDITION.DATA )
2130   IF PCIB.ERR<>0 THEN ERROR PCIB.BASERR
2140   WEND
2150   ´
2160   ´ SEND TRIGGER COMMAND TO START LIST AND GENERATE THE VOLTAGE RAMP
2170   ´
2180   CMD$ = "TRIGGER:IMMEDIATE"                       ´ THIS IS AN IMMEDIATE TRIGGER, WHICH IS ALWAYS
2190   L s LEN( CMD$ )                                  ´ ACTIVE. THEREFORE, IT DOES NOT NEED TO BE
2200   CALL IOOUTPUTS( SLOT0, CMD$, L )                 ´ SELECTED AS A TRIGGER SOURCE.
2210   IF PCIB.ERR<>0 THEN ERROR PCIB.BASERR
2220   ´
2230   END
```
42

```
1       ´ MERGE "DECL.BAS" AS INSTRUCTED IN YOUR NATIONAL INSTRUMENTS GPIB-PC MANUAL
2       ´
1000    ´ APPLICATION #3: CONTROLLING VOLTAGE RAMP UP AT TURN ON
1010    ´ FOR GWBASIC AND THE NATIONAL INSTRUMENTS GPIB-PC INTERFACE CARD
1020    ´ PROGRAM: N3.BAS
1030    ´
1040    ´ CONFIGURE THE GPIB.COM HANDLER FOR THE FOLLOWING:
1050    ´
1060    ´ EOI ENABLED FOR BOTH READ AND WRITE
1070    ´ DISABLE AUTO SERIAL POLL
1080    ´
1090    INSTRUMENT.NAME$ = "SLOT0"
1100    CALL IBFIND( INSTRUMENT.NAME$, SLOT0% )
1110    IF SLOT0% < 0 THEN PRINT "COULDN'T FIND MODULE": STOP
1120    ´
1130    OPTION BASE 1
1140    VSTEP$ = ""                                        ´ STRING TO HOLD THE VOLTAGE RAMP STEPS
1150    VSTART = 2                                         ´ START VOLTAGE FOR RAMP
1160    VSTOP = 10                                         ´ STOP VOLTAGE FOR RAMP
1170    RAMPTIME = .5                                      ´ TIME IN SECONDS TO CHANGE FROM VSTART TO VSTOP
1180    DWELL = RAMPTIME / 19                              ´ DWELL TIME FOR EACH POINT
1190    ´
1200    ´ SINCE THE OUTPUT STAYS AT THE LAST VOLTAGE POINT AFTER ITS DWELL TIME EXPIRES, THE DWELL TIME OF THE
1210    ´ LAST POINT IS NOT PART OF THE TRANSITION TIME. THEREFORE, DIVIDE THE TOTAL TIME BY 19 POINTS, NOT 20.
1220    ´ YOU WANT THE SAME DWELL TIME FOR EVERY POINT IN THE LIST, SO YOU NEED TO DOWNLOAD ONLY 1 DWELL
TIME.
1230    ´
1240    ´ SINCE THE NATIONAL INSTRUMENTS GPIB-PC WORKS WITH STRINGS, THE RAMP DATA MUST BE CONSOLIDATED
INTO A SINGLE
1250    ´ STRING WHICH CONTAINS ALL THE POINTS, SEPARATED BY COMMAS
1260    ´
1270    FOR I=1 TO 20                                      ´ MAKES THE STRING EQUIVALENTS OF THE
1280    VSTEP$ = VSTEP$ + STR$( VSTART + ((( VSTOP - VSTART ) / 20 ) * 1))          ´ VOLTAGE POINTS AND
                                                           CONCATENATES THEM ONLY
1290    IF I <> 20 THEN VSTEP$=VSTEP$+","                  ´ FOR THE FIRST 19 POINTS, EACH FOLLOWED
1300    NEXT I                                             ´ BY A COMMA. THE LAST POINT IS NOT
1310    ´´ FOLLOWED BY A COMMA.
1320    ´
1330    CMD$ = "*RST;*CLS;STATUS:PRESET"                   ´ RESET AND CLEAR MODULE
1340    CALL IBWRT( SLOT0%, CMD$ )
1350    IF IBSTA% < 0 THEN GOTO 1960
1360    ´
1370    CMD$ = "VOLT " + STR$( VSTART )                    ´ START RAMP AT VSTART. USE NUMBER TO STRING
1380    CALL IBWRT( SLOT0%, CMD$ )                         ´ CONVERSION TO SEND REAL NUMBERS OVER THE BUS
1390    IF IBSTA$ < 0 THEN GOTO 1960                       ´ AS PART OF THE COMMAND STRING.
1400    ´
1410    CMD$ = "CURR .1"
1420    CALL IBWRT( SLOT0%, CMD$ )
1430    IF IBSTA% < 0 THEN GOTO 1960
1440    ´
1450    CMD$ = "OUTPUT ON"                                 ´ ENABLE OUTPUT
1460    CALL IBWRT( SLOT0%, CMD$ )
1470    IF IBSTA% < 0 THEN GOTO 1960
1480    ´
1490    CMD$ = "VOLT:MODE LIST"                            ´ SET TO GET VOLTAGE FROM LIST
1500    CALL IBWRT( SLOT0%, CMD$ )
1510    IF IBSTA% < 0 THEN GOTO 1960
1520    ´
1530    CMD$ = "LIST:VOLT " + VSTEP$                       ´ DOWNLOAD VOLTAGE LIST POINTS
1540    CALL IBWRT( SLOT0%, CMD$ )
1550    IF IBSTA% < 0 THEN GOTO 1960
1560    ´
1570    CMD$ = "LIST:DWELL " + STR$( DWELL )               ´ DOWNLOAD 1 DWELL TIME. USE NUMBER TO STRING
1580    CALL IBWRT( SLOT0%, CMD$ )                         ´ CONVERSION TO SEND REAL NUMBERS OVER THE BUS
1590    IF IBSTA% < 0 THEN GOTO 1960                       ´ AS PART OF THE COMMAND STRING.
1600    ´
1610    CMD$ = "LIST:STEP AUTO"                            ´ DWELL-PACED LIST
1620    CALL IBWRT( SLOT0%, CMD$ )
1630    IF IBSTA% < 0 THEN GOTO 1960
```

```
1640    ´
1650    CMD$ = "INITIATE"                                   ´ ENABLE TRIGGER TO START LIST
1660    CALL IBWRT( SLOT0%, CMD$ )
1670    IF IBSTA% < 0 THEN GOTO 1960
1680    ´
1690    ´ BEFORE TRIGGERING THE MODULE, DETERMINE IF IT IS READY BY CHECKING FOR
1700    ´ ´WAITING FOR TRIGGER´ (BIT 5 OF THE OPERATION STATUS REGISTER).
1710    ´
1720    ´ YOU COULD ELIMINATE THIS STEP BY SIMPLY INSERTING A PAUSE IN THE PROGRAM. HOWEVER, BY
1730    ´ CHECKING THE INSTRUMENT STATUS, YOU CAN AVOID TIMING PROBLEMS. ALSO, ANY OTHER OPERATIONS
1740    ´ THAT TAKE TIME WILL GIVE THE MODULE A CHANCE TO COMPLETE PROCESSING.
1750    ´
1760    CONDITION.DATA$ = SPACE$(20)                        ´ RESERVE SPACE FOR READING IN STRING
1770    ´
1780    WHILE ( ( VAL( CONDITION.DATA$ ) AND 32 ) <> 32 )   ´ CONTINUE TO LOOP UNTIL BIT 5 (VALUE 32) = TRUE
1790    CMD$ = "STATUS:OPERATION:CONDITION?"
1800    CALL IBWRT( SLOT0%, CMD$ )
1810    IF IBSTA% < 0 THEN GOTO 1960
1820    CALL IBRD( SLOT0%, CONDITION.DATA$ )
1830    IF IBSTA% < 0 THEN GOTO 1960
1840    WEND
1850    ´
1860    ´ SEND TRIGGER COMMAND TO START LIST AND GENERATE THE VOLTAGE RAMP
1870    ´
1880    CMD$ = "TRIGGER:IMMEDIATE"                          ´ THIS IS AN IMMEDIATE TRIGGER, WHICH IS ALWAYS
1890    CALL IBWRT( SLOT0%, CMD$ )                          ´ ACTIVE THEREFORE, IT DOES NOT NEED TO BE
1900    IF IBSTA% < 0 THEN GOTO 1960                        ´ SELECTED AS A TRIGGER SOURCE.
1910    ´
1920    STOP
1930    ´
1940    ´ GENERAL ERROR HANDLER
1950    ´
1960    PRINT "GPIB function call error: "
1970    PRINT "IBSTA%="; IBSTA%, "IBERR% = "; IBERR% ,"IBCNT% = ";IBCNT%
1980    ´
1990    END
```

```
/* APPLICATION #3: CONTROLLING VOLTAGE RAMP UP AT TURN ON
        FOR MICROSOFT C AND THE HP 61062/82990/82335A HP-IB COMMAND LIBRARY FOR MS-DOS
        PROGRAM: HP3. C                                         */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "chpib.h"
#include "cfunc.h"
#define INTERFACE 7L                                    /* Select code 7 for the HP-IB interface card.  */
#define SLOT0     70500L                                /* Select code 7, mainframe address 05, slot 00.         */
#define WTG       32                                    /* Waiting for Trigger (WTG) = bit 5 = value 32.         */
#define NUM_PTS                                         20          /* 20 points in the voltage List.  */
int error;
mains()
{
        char *cmd;   /* Used to hold command strings sent to the module.          */
        char cmd_buff[255];                             /* Used to hold command strings during string manipulations.
*/
        static char cr_lf[13] = { 13, 10, 0 };          /* Carriage return+line feed = end of line.      */
        int i /* Loop counter.                          */
        float condition_data;                           /* Used to hold data from read back of status conditions.*/
        float vstart = 2.0;                             /* Start voltage for the ramp.        */
        float vstop = 10.0;                             /* Stop voltage for the ramp.         */
        float vstep[NUM_PTS];                           /* Used to hold voltage List points for the ramp.          */
        float ramptime = 0.5;                           /* Transition time (in seconds) for the ramp.   */
        float dwell;   /* Dwell time (in seconds) for each ramp step.         */
        dwell = ramptime / 19.0                         /* Since the output stays at the last voltage point after its dwell
             time expires, the dwell time of the last point is not part of the
             transition time. Therefore, divide the total time by 19 points,
             not 20. You want the same dwell time for every point in the List,
             so only download 1 dwell time.             */
for (i = 1; i <= NUM_PTS; i++)                          /* Calculate the voltage List points  */
        vstep[i] = vstart + (((vstop - vstart) / NUM_PTS) * i);
        error = ioreset(INTERFACE);                     /* To get the interface to its defaults.          */
        error handler(error, "Reacting the interface");
        error = iotimeout(INTERFACE, (double)2.0);      /* Enables timeout of 2 seconds.    */
        error_handler(error, "Setting the timeout");
/* Note regarding HP-IB read/write terminations:
The default of the interface card is that EOI is enabled and the read/writes terminate on carriage
return/line feed. The module terminates on either EOI or line feed, so the default settings of the card
are sufficient.     */
cmd = "*RST;*CLS;STATUS:PRESET";                        /* Reset and clear module.          */
error = iooutputs(SLOT0, cmd, strlen(cmd));
error handler(error, cmd);
sprintf(cmd_buff , "VOLT %f", vstart);                  /* Start ramp at vstart. Use number to string   */
error = iooutputs(SLOT0, cmd_buff, strlen(cmd_buff));   /* conversion to send real numbers over the   */
        error_handler(error, cmd_buff);                 /* bus as part of the command string.         */
cmd = "CURR .1";
error = iooutputs(SLOT0, cmd, strlen(cmd));
        error_handler(error, cmd);
cmd = "OUTPUT ON";                                      /* Enable output          */
error = iooutputs(SLOT0, cmd, strlen(cmd));
        error_handler(error, cmd);
cmd = "VOLT:MODE LIST";                                 /* Set to get voltage from List       */
error = iooutputs(SLOT0, cmd, strlen( cmd) );
error_handler(error, cmd);
/* Sending voltage data points requires two steps using the HP-IB Command Library. The instruction
contains both string data and a real array. First, send the string data command header "LIST:VOLT " to
the module using iooutputs. Then, send the real array using iooutputa. However, you must inhibit the EOI
and End-of-Line terminator after the iooutputs or the module will stop taking data. Then, re-enable them
to terminate the iooutputa.                                          */
error = ioeoi(INTERFACE, 0);                                         /* Turn off EOI         */
        error_handler/error, "Disabling EOI");
error_ioeol(INTERFACE, "", 0),                                       /* Turn off End-of-Line termination  */
        error_handler(error, "Disabling EOL"),
cmd = "LIST:VOLT ";                                                  /* First send the voltage header (string) ---      */
error = iooutputs(SLOT0, cmd, strlen(cmd));
        error_handler(error, cmd);
error = ioeoi(INTERFACE, 1);                                         /* Turn on EOI          */
        error_handler(error, "Enabling EOI");
error = ioeol(INTERFACE, cr_lf, strlen(cr_lf));                      /* Turn on End-of-Line termination  */
        error_handler(error, "Enabling EOL");
error = iooutputa(SLOT0, &vstep[1], NUM_PTS);                        /* Download voltage points (array), starting      */
        error_handler(error, "Voltage List Array");                  /* with the element 1, not 0.
sprintf(cmd buff, "LIST:DWELL %f", dwell);                           /* Download 1 dwell time. Use number to        */
error = iooutputs(SLOT0, cmd_buff, strlen(cmd_buff));                /* string conversion to send the real*/
        error_handler(error, cmd_buff);                              /* number over the bus as part of the          */
             /* command string.                                      */
cmd = "LIST:STEP AUTO";                                              /* Dwell-paced List      */
error = iooutputs(SLOT0, cmd, strlen(cmd));
        error_handler(error, cmd);
cmd = "INITIATE";                                                    /* Enable trigger to start List        */
error = iooutputs(SLOT0, cmd, strlen(cmd));
        error_handler(error, cmd);
/* Before triggering the module, determine if it is ready by checking for
´Waiting for Trigger´ (bit 5 of the Operation Status Register).
You could eliminate this step by simply inserting a pause in the program. However, by
checking the instrument status, you can avoid timing problems. Also, any other operations
that take time will give the module a chance to complete processing.    */
do {
        cmd = "STATUS:OPERATION:CONDITION?";
        error = iooutputs(SLOT0, cmd, strlen(cmd));
             error_handler(error, cmd);
        error = ioenter(SLOT0, &condition data);                     /* You must convert float to integer  */
        error_handler(error, "Read back of status");                 /* to do an integer bit test.         */
        } while (((int)condition_data && WTG) == 0) ;                /* Loop until bit 5 (value 32) is true. */
/* Send trigger command to start List and generate the voltage ramp. */
cmd = "TRIGGER:IMMEDIATE";                                           /* This is an immediate trigger, which is always             */
error = iooutputs(SLOT0, cmd, strlen(cmd));                          /* active. Therefore, it does not need to be      */
        error_handler(error, cmd);                                   /* selected as a trigger source.       */
}
error handler(error,bad_string)                                      /* This is a generalized error checking routine.*/
int error;
char *bad_string;
{
        if (error != D) {
             printf("HP-IB error while sending or receiving ´%s´.\n", bad_string);
        printf("Error #: %d -- %s\n", error, errstr(error));
        }
}
```

```
/* APPLICATION #3: CONTROLLING VOLTAGE RAMP UP AT TURN ON
       FOR MICROSOFT C AND THE NATIONAL INSTRUMENTS GPIB-PC INTERFACE CARD
       PROGRAM: N3.C
       Configure the GPIB.CCM handler for the following:        EOI enabled for both read and write
            Disable auto serial poll                              */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "decl.h"
#define ERR (1<<15)                                          /* Error detected as bit 15 of ibsta.  */
#define NUM_PTS 20                                           /* The number of points in the voltage List.      */
#define MAX_LEN 255                                          /* Maximum length of a string = 255 characters.             */
#define SMALL_STRING 15                                      /* When you need a small string of 15 characters.           */
#define WTG 32   /* Waiting for Trigger (WTG) = bit 5 = value 32       */
int slot0;  /* Device number of module in slot 0. slot0 is configured in GPIB.COM as
            GPIB address 5, secondary address 96.             */
main()
{
       char *cmd;    /* Used to hold command strings sent to the module.                 */
       char cmd_buff[MAX_LEN];                               /* Used to hold co_and strings during string manipulations.
*/
       char vpoint[SMALL_STRING];                            /* Used to hold the string equivalent of one voltage ramp step.
*/
       char vlist[MAX LEN];                                  /* Used to hold the entire voltage List command header and points.
*/
       char condition_data[SMALL_STRING];                    /* Reserve space for reading back status conditions.       */
       int i;      /* Loop counter.                          */
       float vstart = 2.0;                                   /* Start voltage for the ramp.           */
       float vstop = 10.0;                                   /* Stop voltage for the ramp.            */
       float ramptime = 0.5;                                 /* Transition time for the ramp.         */
       float dwell;    /* Dwell time for each ramp step.     */
       dwell = ramptime / 19.0;                              /* Since the output stays at the last voltage point after its dwell
            dwell expires, the dwell time of the last point is not part of the
            transition time. Therefore, divide the total time by 19 points, not 20.
            You want the same dwell time for every point in the List, so only
            download 1 dwell time.                            */
       if ((slot0 = ibfind("SLOT0")) < 0)                    /* Assign unique identifier to the device slot0 and store in
*/
       finderr();       /* variable slot0. Error = negative value returned.     */
       cmd = "*RST;*CLS;STATUS:PRESET";                      /* Reset and clear module.             */
       ibwrt(slot0, cmd, strlen(cad));
       if (ibsta & ERR)
            error(cmd);
       sprintf(cmd_buff, "VOLT %f", vstart);                 /* Start ramp at vstart. Use number to string conversion to send
*/
            /* real numbers over the bus as part of the command string.  */
       ibwrt(slot0, cmd_buff, strlen(cmd buff));
       if (ibsta & ERR)
            error(cmd_buff);
       cmd = "CURR .1";
       iburt(slot0, cmd, strlen(cmd));
       if (ibsta & ERR)
       error(cmd);
       cmd = "OUTPUT ON";                                    /* Enable output             */
       ibwrt(slot0, cmd, strlen(cmd));
       if (ibsta & ERR)
       error(cmd);
       cmd = "VOLT:MODE LIST";                               /* Set to get voltage from List          */
       ibwrt(slot0, cmd, strlen(cmd));
       if (ibsta & ERR)
            error(cmd);
       strcpy(vlist, "LIST:VOLT ");                          /* Start with the command header for the voltage List.      */
for (i = 1; i < NUM_PTS; i++) {                                          /* The loop calculates the string     */
        sprintf(vpoint, " %f, ", vstart+(((vstop - vstart) / NUM_PTS) * I));  /* equivalents of the voltage List      */
        strcat(vlist, vpoint);                                           /* points and concatenates them for                 */
}            /* only the first 19 because there                          */
             /* should not be comma after the                            */
sprintf(vpoint , "%f" , vstop);                                          /* last point. Do the last point           */
strcat(vlist, vpoint);                                                   /* separately with no comma.           */
iburt(slot0, vlist, strlen(vlist));                                      /* Download voltage List points       */
if (ibsta & ERR)
        error(viist);
sprintf(cmd_buff, "LIST:DUELL %f", dwell);                               /* Download 1 dwell time. Use number to        */
ibwrt(slot0, cmd_buff, strlen(cmd_buff));                               /* string conversion to send the real*/
if (ibsta & ERR)    /* number over the bus as part of the                */
        error(cmd_buff);                                                 /* command string.         */
cmd = "LIST:STEP AUTO";                                                  /* Dwell-paced List         */
ibwrt(slot0, cmd, strlen(cmd));
if (ibsta & ERR)
        error(cmd);
cmd = "INITIATE";                                                        /* Enable trigger to start List           */
ibwrt(slot0, cmd, strlen(cmd));
if (ibsta & ERR)
        error(cmd);
/* Before triggering the module, determine if it is ready by checking for
        ´waiting for Trigger´ (bit 5 of the Operation Status Register).
        You could eliminate this step by simply inserting a pause in the program. However, by
        checking the instrument status, you can avoid timing problems. Also, any other operations
        that take time will give the nodule a chance to complete processing.                    */
do {
        cmd = ~STATUS:OPERATION:CONDITION? ";
        iburt(slot0, cmd, strlen(cmd));
if (ibsta & ERR)
        error(cmd);
ibrd(slot0, condition_data, SMALL_STRING);                               /* Allow to read SHALL_STRING bytes, which is more    */
if (ibsta & ERR)    /* than enough. Note that first byte will be a + sign,  */
        error(condition_data);                                           /* so you must convert the string to float, then to int,        */
             /* to do an integer bit test.                               */
} while (((int)(atof(condition_data)) && WTG) == 0);                     /* Loop until WTG = bit 5 (value 32) is true.    */
/* Send trigger command to start List and generate the voltage ramp.   */
cmd = "TRIGGER:IMMEDIATE";                                               /* This is an immediate trigger, which is always              */
iburt(slot0, cmd, strlen(cmd));                                          /* active. Therefore, it does not need to be       */
if (ibsta & ERR)    /* selected as a trigger source.                     */
error(cmd);
}
finderr()  /* Indicates that ibfind failed                               */
{
printf(.lbfind error: Does device name given match configuration name?\n");
}
error(bad_string)   /* This is a generalized error checking routine.       */
char *bad_string;
{
printf("GPIB error while sending or receiving ´%s´.\n´´, bad_string);
printf("GPIB status : ibsta = 0x%x, iberr = 0x%x, ibcnt = 0x%x\n", ibsta, iberr, ibcnt);
}
```

For more information about Hewlett-Packard Test and Measurement products, applications, services, and for a current sales office listing, visit our web site, http://www.hp.com/go/tmdir. You can also contact one of the following centers and ask for a test and measurement sales representative.

United States:
Hewlett-Packard Company
Test and Measurement Call Center
P.O. Box 4026
Englewood, CO 80155-4026
1 800 452 4844

Canada:
Hewlett-Packard Canada Ltd.
5150 Spectrum Way
Mississauga, Ontario
L4W 5G1
(905) 206 4725

Europe:
Hewlett-Packard
European Marketing Centre
P.O. Box 999
1180 AZ Amstelveen
The Netherlands
(31 20) 547 9900

Japan:
Hewlett-Packard Japan Ltd.
Measurement Assistance Center
9-1, Takakura-Cho, Hachioji-Shi,
Tokyo 192, Japan
Tel: (81-426) 56-7832
Fax: (81-426) 56-7840

Latin America:
Hewlett-Packard
Latin American Region Headquarters
5200 Blue Lagoon Drive
9th Floor
Miami, Florida 33126
U.S.A.
(305) 267 4245/4220

Australia/New Zealand:
Hewlett-Packard Australia Ltd.
31-41 Joseph Street
Blackburn, Victoria 3130
Australia
1 800 629 485

Asia Pacific:
Hewlett-Packard Asia Pacific Ltd
17-21/F Shell Tower, Times Square,
1 Matheson Street, Causeway Bay,
Hong Kong
Tel: (852) 2599 7777
Fax: (852) 2506 9285

Data subject to change
Copyright © 1991
Hewlett-Packard Company
Printed in U.S.A. April, 16, 2001
5988-2800EN