



# In-System Programming on an HP3070 through the MACH CPLD 1149.1 Test Access Port

*Application Note*

---

---

# In-System Programming on an HP3070 through the MACH CPLD 1149.1 Test Access Port

## **Application Note**

by Arthur Khu (AMD MACH/PLD Apps), APG Test Consultants

---

## **INTRODUCTION**

This application note describes in detail the procedures necessary to program AMD MACH CPLDs with IEEE 1149.1 (JTAG) compliant test access port circuitry using ATE equipment. The HP3070 In-Circuit test platform is used as an example, but, the techniques and practices outlined in this application note will apply across a broad range of ATE equipment and PC based IEEE-1149.1 tools.

### **Advantages of JTAG In-System Programming (JTAG-ISP)**

JTAG In-System Programming (JTAG-ISP) offers significant advantages over traditional stand alone device programmers. JTAG-ISP fits perfectly into today's highly automated manufacturing and test environments because it removes an entire and complex process step. JTAG-ISP eliminates the need to program parts off line away from the production flow and streamlines the manufacturing process by incorporating programming into the test process. With stand-alone part programmers, devices have to be unpacked from reels or tubes and loaded into programmers, programmed, and then re-packaged and sent to production or back into inventory. This labor intensive and expensive operation requires a considerable amount of planning and scheduling. JTAG-ISP eliminates this entire process. Parts are loaded directly onto the Printed Circuit Board (PCB) assemblies without any additional handling and programmed directly on the card during the test process. JTAG-ISP completely eliminates the possibility of a part with the wrong program being loaded onto the card and eliminates the need to track and separate parts with different programs. Costs associated with purchasing, maintaining, and updating part programmers are also removed.

The additional handling required when using part programmers exposes fine pitch components to potential lead damage and ESD exposure. Quantities of each individual programmed part have to be calculated, scheduled, and tracked. The manufacturing process has to treat each individual programmed part as a separate part number. With JTAG-ISP only the part type has to be considered just like non-programmable parts.

This simplifies inventory management. JTAG-ISP provides total flexibility when implementing design changes. With traditional device programming if a part undergoes a design change then the entire inventory of the old revision has to be purged and then re-programmed. This is a costly and time consuming process doubling the devices' handling exposure. With JTAG-ISP a simple re-compile of a new test program will have the design change on-line almost instantaneously. No inventory purging or additional handling required. The tester becomes a single focal point for engineering changes.

Another significant advantage of JTAG-ISP is the parallel programming capability. Designs that contain multiple MACH JTAG-ISP devices can be programmed in parallel in little more time than it takes to program one part. This eliminates bottlenecks associated with part programmers. JTAG-ISP also eliminates the need for costly and unreliable sockets. Sockets are a manufacturing, testing, and reliability nightmare. Detecting manufacturing defects on sockets such as open solder joints can be difficult. Signal integrity is reduced by sockets due to the unreliable mechanical interface. Having the CPLD soldered directly to a circuit board is far superior in every respect than having it placed in a socket. All connections to the board can be fully tested using the IEEE-1149.1 boundary scan and there is no need to remove the part if it needs re-programming, just re-test the card.

### **Additional Considerations are Required for JTAG-ISP**

Because in-system programming occurs "in the system" and not in a stand alone part programmer, certain additional design requirements must be taken into account. The interaction of other circuitry and the in-circuit test fixture must be carefully considered. However, these considerations are no more demanding than good PCB and design for test practices. If the simple requirements outlined below are followed throughout the PCB design cycle then the advantages of JTAG-ISP can be obtained with little effort.

## JTAG-ISP DESIGN FOR TEST AND DESIGN FOR PROGRAMMING REQUIREMENTS

Potential noise and signal integrity problems can be avoided by following a few simple design rules during the printed circuit board and fixture development stages. Noise and signal integrity problems are amplified when applying and receiving signals through a “bed of nails” fixture. Designers don’t normally have to consider the interactions of “probes” and “fixture wires” on their designs. A marginal signal on a PCB will become a nightmare when it is loaded with an in-circuit test fixture. Therefore, it is imperative that signal integrity on TCK and TMS traces be an important consideration when laying out and designing the printed circuit board.

If any of the requirements outlined below are not met then it is unlikely that in-circuit programming will be successful. It is imperative that in-circuit programming be considered during every phase of the circuit board design and test program design. Maintaining good signal integrity, reducing noise, and the ability to persistently disable upstream devices are key to successful in-circuit programming.

### Printed Circuit Board Lay Out Considerations for Boundary Scan Chains

The physical layout of the traces connecting the boundary scan chain together has a profound influence on signal integrity. Boundary scan chains should be connected on the PCB to minimize trace lengths and optimize signal quality. The signal traces for TCK and TMS must be treated with the same respect given to high speed clock lines during PCB layout. The traces should be as short as possible and the number of stubs and forked connections should be kept to a minimum. Boundary scan chain connections should be based on the physical position of the components on the PCB. Chains should not be connected to suit any logical connections on a schematic. It makes no difference to the program generation or test software where the target devices are logically in the chain. However, poor signal integrity caused by inferior PCB layout practices can cause significant problems during JTAG-ISP and boundary scan testing.

Ground access points (vias, component leads, etc.) should be numerous and distributed evenly across the entire PCB. The even distribution of ground access points across the surface of the PCB helps to reduce the wire length on the ground probes and reduces the effects of ground bounce. Large boundary scan chains must have a substantial number of ground access points distributed across the PCB. This will enable the fixture design software to generate short ground wires throughout the fixture. The number and distribution of ground access points will be the single most important

factor in determining the signal integrity of the in-circuit test fixture. Poor and limited distribution of ground access points will prevent reliable and repeatable in-system programming and boundary scan testing.

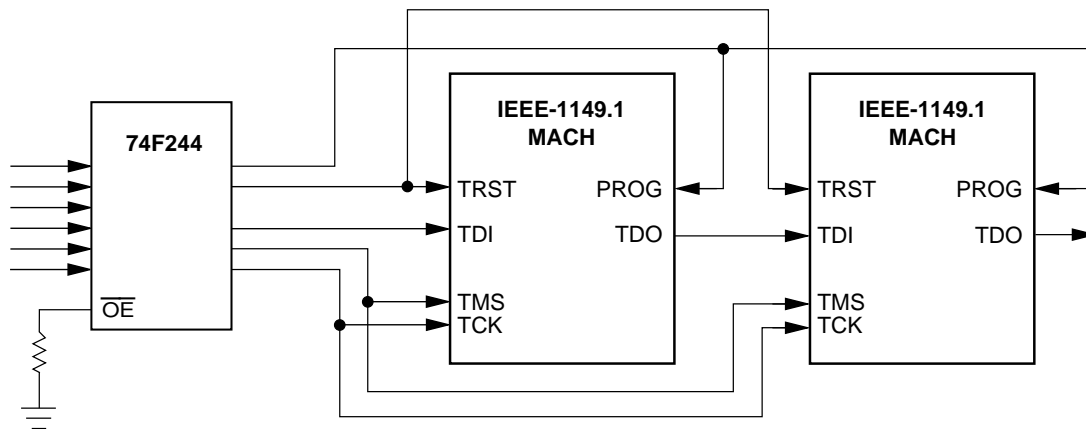
### Disabling Upstream Devices

All devices that can drive the nodes TCK, TMS, TDI, TDO, PROGRAM, and TRST must be disabled during in-circuit programming. In addition, the methods used to disable these devices must not backdrive the outputs of other components. The disabling methods used should also persist during periods where the tester drivers are inactive between tests.

Under most circumstances the IEEE-1149.1 bus signals TCK, TMS, TDI, TDO, and TRST are not normally driven by other components or bused with other devices. However, in some designs components do take control and drive the test bus. These devices could be buffers, scan controllers and ASICs used for embedded diagnostics or for dynamic self-configuration. Special consideration must be given to these unique topologies when attempting to implement JTAG-ISP using an in-circuit tester.

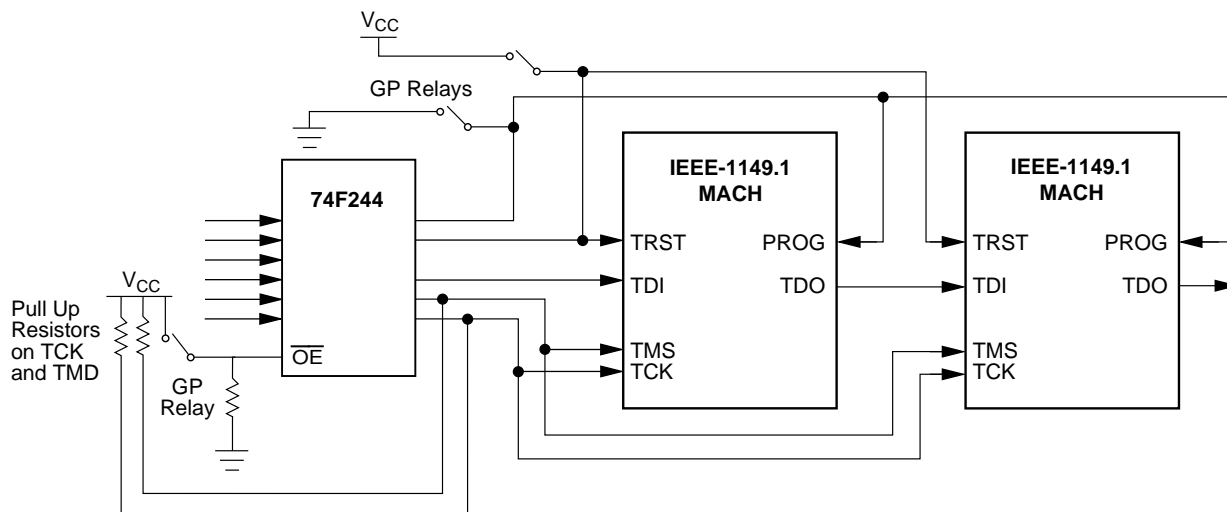
An example of a device that shares the test bus is shown in Figure 1. This device should be disabled from the bus at all times during JTAG-ISP. This is a simplistic example and more complex configurations could exist, but they must be designed in the same way to insure persistent and non-backdriving disables. Figure 1 is a good example of a design that allows an upstream device to be persistently disabled without backdriving. This component can be disabled persistently by connecting the Output Enable pin to VCC using a tester general purpose (GP) relay (see Figure 2).

File size and vector count for MACH programming are quite large when compared to regular in-circuit tests. Programming files for multiple devices are far too large for most testers to compile and apply in one single pass. If a programming file is too large to be compiled on its own then it must be partitioned (broken up) into a number of smaller tests that are applied sequentially. The key to successful partitioning is the ability to continuously hold critical signals in known states during transitions between tests. This can be achieved using a combination of pull up resistors and connecting signals directly to power or ground using GP relays. All TRST pins on every boundary scan device in the chain must be fully controlled along with all MACH program pins. TRST must remain high throughout the entire duration of JTAG-ISP. We recommend that all PROGRAM pins are fixed low throughout the duration of JTAG-ISP also. The programming vectors pulse the PROGRAM pin low when programming data has been loaded. However, holding the PROGRAM pin low for the entire duration of JTAG-ISP is acceptable and is recommended.



20985A-1

**Figure 1. The 74F244 Shares the Test Bus with Two 1149.1-Compliant MACH Parts**



20985A-2

**Figure 2. Persistence of Critical Signals and Disabling During Programming**

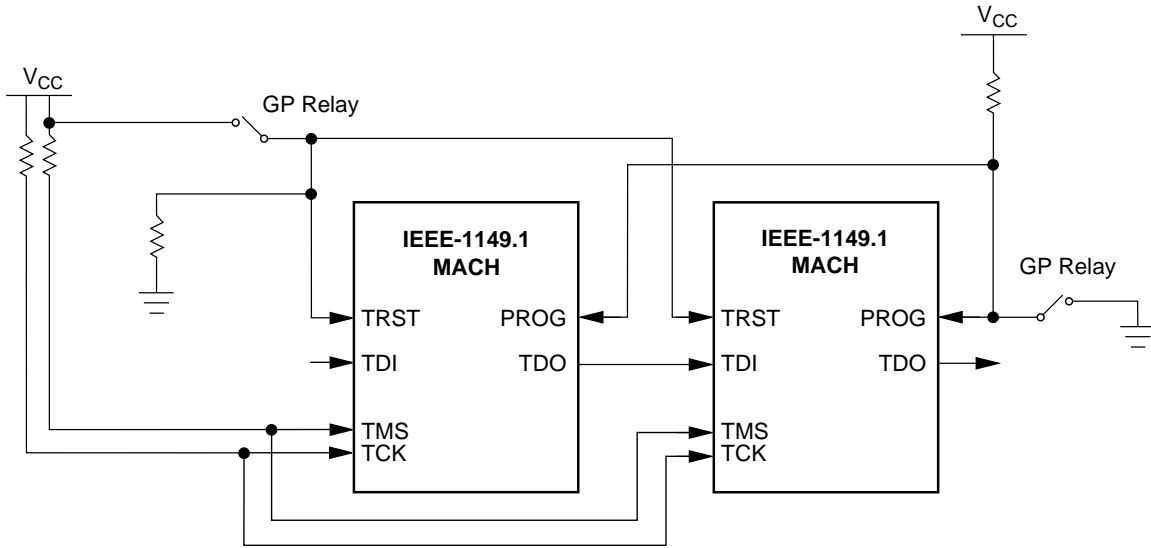
An example of how to achieve persistent signals and disables is shown in Figure 2. The test is strategically partitioned at a point where TCK and TMS are being driven high. At the end of the first test the drivers are turned off and TCK and TMS remain high due to the pull up resistors. TRST and PROGRAM are continuously held high and low, respectively, by the use of GP relays connecting them to the power and ground nodes. The next test in the sequence always starts out

driving the last vector of the previous test. In this case TCK and TMS will be driven high and the tester will again take control of the device.

Figure 2 shows a GP relay used to disable a bused device during programming. This is an ideal disabling situation for a device that shares the test bus. Figure 2 also shows the use of GP relays to hold the TRST and PROGRAM persistently during ISP.

Figure 3 shows an optimum scan chain design for JTAG-ISP. No other devices can drive the test bus except the tester. Pull up and pull down loads are designed onto the PCB (not wired into the fixture which adds additional wires on critical nodes). Only two devices are shown in Figure 3. However, any number of

IEEE-1149.1 devices could be bused together, occupying any position in the scan chain. If other devices in the chain have a TRST pin or "Compliance" pins (pins that must be asserted to place them into boundary scan mode) then these signals must be controlled persistently using GP relays or pull up/down resistors.



20985A-3

Figure 3. Optimum Scan Chain Design for JTAG-ISP

### Test Fixture Design Considerations for Boundary Scan Chains

Once the PCB has been designed and routed optimally for signal integrity, good ground access and distribution in the test fixture design must be considered. There are a number of things a test developer can do to reduce noise and increase signal integrity in the test fixture. The most important factor will be wire lengths. Long wires introduce noise and reduce signal quality. Nodes such as TCK and TMS must be marked "CRITICAL" in the HP3070 board test files so that during fixture design the shortest possible wires are assigned. For other ATE equipment similar instructions or precautions must be followed to insure that TCK and TMS nodes have short wires.

Twisted pair wiring can also be specified for critical nodes. Twisted pair wiring can be selected for these nodes using the HP3070 board consultant program. Twisted pair wiring in combination with a ground plane is strongly recommended for very large boundary scan chains and multiple part programming. All of the tester grounds should be wired to the plane with short low impedance wires or ground rakes. There must be an adequate number of ground resources assigned in the fixture. To increase the number of grounds specify a higher power supply current than required to power the

board. This will force the fixture design software to assign more ground resources. If good PCB design practices are followed and ground access points on the board are numerous and distributed adequately across the board then optimal short wire ground interfaces will exist.

Board placement can have an effect on signal integrity. When designing the fixture, carefully place the board over the tester resources paying close attention to the points where TCK, and TMS nets will be probed. Place the board so they are very near digital resources if possible. Also place the board so that only a minimum amount of ground resources are blocked by probes. In particular pay close attention to the ground resources on the cards that drive the TCK, TMS, and TDI nodes. Try not to block any ground resources on these cards.

Obtaining a reliable and solid probe contact with TCK, TMS, and TDI nodes is also crucial for reducing noise and maintaining signal quality. Try to design the PCB with pad sizes greater than 35 mil at least for TCK, TMS, TDI, TDO, TRST, and PROGRAM access points. Space the points at least 100 mil to 75 mil from other points so that 100 mil or 75 mil probes can be used. High force (10 oz.) and steel-tipped probes will help to obtain solid reliable probe contact.

## CREATING A HP3070 PCF FILE USING MACHPRO™

The first step in creating a PCF programming file is to create a chain description that describes the target boundary scan chain. An example chain file would look as follows:

```
'comment0' mach445 p 6 arbiter.jed / -o
Z -f arbiter.out;

'comment1' mach445 p 6 dramctrl.jed / -o
Z -f dramctrl.out;
```

This chain file is composed of two MACH445 devices. Both parts will be programmed ('p'); the first part listed in the chain (the one connected to TDI) will be programmed with the ARBITER.JED pattern, and the second part will be programmed with the DRAMCTRL.JED file. The second part will have its TDI fed by the TDO of the first part, and the TDO of the second part will be the TDO of the chain.

To process this file, use MACHPRO version 1.34 or later with the following command line at the DOS prompt:

```
C:\> machpro -i project.chn -z 3 -1 -2
projname.pcf
```

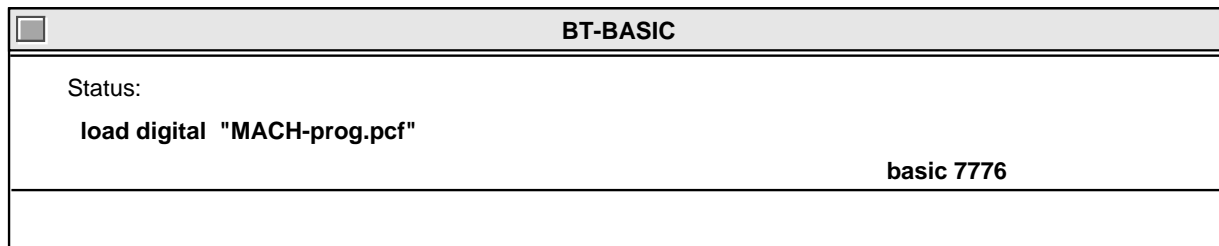
where:

- **-i PROJECT.CHN** is the option to specify the input/chain file
- **-z 3** instructs MACHPRO to display status messages while processing the input file
- **-1** turns on parallel programming mode
- **-2 PROJNAME.PCF** turns on PCF file generation and specifies the filename to write the vectors to

Refer to the AMD MACHPRO software manual for more information and details on creating a chain file, and on running and using the software.

The file size will vary depending on the number of MACH devices targeted for JTAG-ISP and the overall size of the boundary scan chain. Files size will always be large even for programming a single device.

After transferring the PCF file from the PC to the HP3070 a small amount of editing will need to take place. The file type will have to be changed from a text file to a digital file. This is done by loading the file in a BASIC window using the commands shown below and then re-saving it. Substitute your file name for "mach-prog.pcf".



20985A-4

It could take a while to load the file because of its size. For extremely large files that cannot be directly loaded into a BASIC window, proceed with File Partitioning first (see file partitioning section), and then change the file status and assign the node names in each file.

## ASSIGNING NODE NAMES IN THE PCF PROGRAMMING FILE

After the programming file has been loaded in a window, edit the node names to match those found in the board file. The following shows an example of a PCF programming file.

```
assign TCK      to nodes "TCK" ! Enter nodes
assign TMS      to nodes "TMS"
assign TDI      to nodes "TDI"
assign TDO      to nodes "TDO_TDI_2"
```

```
assign TRST     to nodes "N_BSCAN_RST"
assign PROGRAM  to nodes "N$3865"
```

In this example the actual node names on the board are: TCK, TMS, TDI, TDO\_TDI\_2, N\_BSCAN\_RST, N\$3865.

These node names will be unique on every board. Each programming file generated by MACHPRO must be edited by the test programmer to reflect the actual node names that are assigned in the target board file.

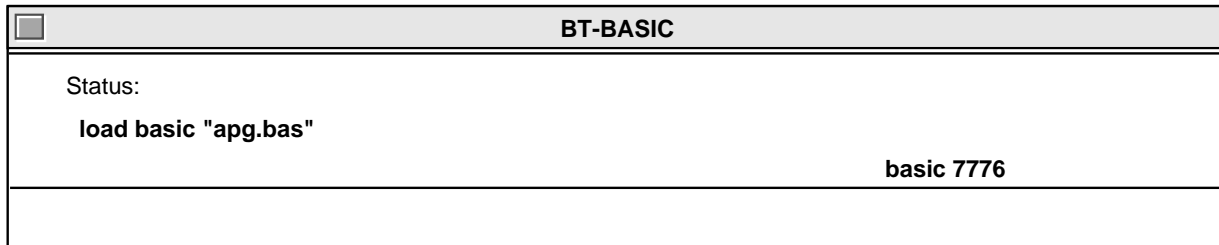
The MACHPRO program outputs dummy node names (place holders) called TCK\_NODE, TMS\_NODE, TDI\_NODE, TDO\_NODE, TRST\_NODE, PROGRAM\_NODE. Edit the PCF file and replace these dummy node names with the actual node names from the board file. If the PROGRAM node or the TRST node are to be controlled by GP relays or by other persistent methods, then these node names should be "starred

out" (not driven by the tester). Insert an "\*" where the dummy node name was assigned. This will prevent the tester from driving the node during the test causing over-power conditions. Only star out the TRST and PROGRAM nodes if they are being controlled by GP relays. Under no circumstances should these nodes float or glitch at any time during programming. We strongly recommend controlling these signals directly with GP relays at all times during JTAG-ISP.

```
assign TCK      to nodes "TCK" ! Enter nodes
assign TMS      to nodes "TMS"
assign TDI      to nodes "TDI"
assign TDO      to nodes "TDO_TDI_2"
assign TRST     to nodes *
assign PROGRAM  to nodes *
```

**FILE PARTITIONING**

Because of the large file sizes generated for programming CPLDs, it is necessary to break these files into smaller sizes. This process is called partitioning. AMD provides an "Automatic Partition Generator" (APG) program that runs on the HP3070 platform. APG will automatically partition files into sizes that the tester can handle. APG will break files at a point where TCK, and TMS are driven high. The last vector in one file is always the first vector in the next file. This helps to maintain a glitch-free program transition. If the test programmer has adequately taken care of the "persistence" of critical signals along with any disabling required, the tests should transition glitch-free. To use the APG program, simply copy it onto your HP3070, then open a BASIC window in the directory where your MACHPRO files reside. Load the APG BASIC program and run it by following the commands shown below.



20985A-5

You should observe the actual code load up in the window. When it is loaded type run.

```
run
```

The program will prompt you for the file name. Enter the MACHPRO filename and hit return.

```
-----
----- APG Test Consultants, Inc.-----
-----
----- MACH IEEE.1149.1 Programming-----
-----
----- PCF File Partitioning Utility-----
-----
Enter the PCF MACH file you would like to
Partition >
```

The actual number of partitioned test files generated by the APG utility depends upon the size of the original file. As a rule of thumb one file is created for each device programmed. For a single MACH4xx device not in a large chain it is possible that the test will compile without partitioning. Try to compile the test first. The APG

utility appends an underscore to the original file name to indicate the order in which to apply the tests. If the input file name was mach-prog.pcf then the files output by the APG utility will have the extension mach-prog.pcf\_1, mach-prog.pcf\_2, mach-prog.pcf\_3, etc. The following is an example of an HP3070 test plan showing how to execute the tests in sequential order. The GP relay connect statements are there to insure the persistence of the critical signals during programming.

```
gpconnect "disabling_nodes" to "VCC"
gpconnect "TRST"             to "VCC"
gpconnect "PROG"             to "GROUND"

test "digital/mach-prog.pcf_1"
test "digital/mach-prog.pcf_2"
test "digital/mach-prog.pcf_3"
```

After the files are partitioned they will have to be added to the HP3070 test order file and compiled. The files could take up to one hour to compile depending upon the system load. When the files have compiled they are ready for execution by the tester.

## VECTOR FORMATS

The vector format used in this example is Hewlett-Packard's "PCF" (Pattern Capture Format). Because PCF is an explicit binary vector format it is easy to convert into other ATE languages. AMD's MACHPRO also generates programming vectors in SVF (Serial Vector Format). The advantage of SVF is that it is a highly compressed vector format unlike PCF. SVF can be used directly by the ASSET<sup>®</sup> diagnostic system and some Teradyne<sup>®</sup> in-circuit testers.

### PATTERN CAPTURE FORMAT (PCF)

A detailed explanation of PCF vector format is provided here to facilitate conversions to other ATE vector formats. A comment is delimited by the exclamation point "!". Any text to the right of an exclamation point is a comment. The comments at the beginning of a MACHPRO-generated programming file are shown below. The `generate static test` and `dynamic` statements are valid HP3070 syntax but are commented out in the MACHPRO-generated programming files. These features are not used at this time.

```
! Wed Jan 24 15:29:56 1996
! HP PCF File generated by MACHPRO(tm)
! Version 1.32 (c) 1994-1996 Advanced
  Micro Devices, Inc.
! PCF header by APG Test Consultants
! Pattern Capture Format    subset of VCL
! Vector Control Language  digital test
  language
```

```
!generate static test ! 1 device
```

The vector cycle and receive delay time indicate the application rate of the vectors and when the receive strobe is activated.

```
vector cycle 500n
receive delay 400n
```

In the example above an individual PCF vector is applied every 500 nano seconds. The vector is driven by the tester for 500 nano seconds. If any responses are to be measured (out of TDO) by the tester it will measure the output 400 nano seconds after driving the inputs. The programming tests output by MACHPRO use a 50% duty cycle on TCK. Therefore, one cycle of TCK high and low will be represented by two PCF vectors lasting 500 ns, translating into an application rate of 1 MHz.

The assignment section maps the test variable names (TCK, TMS, TDO...) to the actual node names given in the board and fixture file.

```
assign TCK    to nodes "TCK"  ! Enter nodes
assign TMS    to nodes "TMS"
assign TDI    to nodes "TDI"
```

```
assign TDO    to nodes "TDO_TDI_2"
assign TRST   to nodes "N_BSCAN_RST"
assign PROGRAM to nodes "N$3865"
```

The next section allocates the tester resources. The nodes are assigned either a tester driver or receiver.

```
inputs  TCK,TMS,TDI,TRST,PROGRAM
outputs TDO
```

### The *pcf order* Statement

```
! dynamic TCK, TMS, TDI, TDO,
pcf order is
TCK, TMS, TDI, TRST, PROGRAM, TDO
```

This statement determines the vector order, MSB to the left and LSB to the right. In the following example, the tester is driving a low logic state on TCK, a high logic state on TMS, TDI, TRST, and PROGRAM, and receives a low logic state on TDO

The `unit` (and `end unit`) statements indicate the beginning and end of a vector set. The `pcf` (and `end pcf`) statement indicate that `pcf` format vectors are enclosed between the `pcf` and `end pcf` statements. A `pcf` vector contains ones, zeros, X's, L's and H's and are placed between quotes.

```
unit "Program_AMD_MACHS"
pcf
!                               ! Start of vectors
"00011L"
"00011X"
"01011H"
"11011X"    ! Logic Rst
"01011X"
"11011X"    ! Logic Rst
"01011X"
end pcf
end unit
```

Nodes defined as inputs can only be represented by one of three states in each vector. Input states (drive signals) use the convention "1" "0" "Z" (drive high, low, HIGH-Z)

Nodes defined as outputs can only be represented by one of three states in each vector. Output states (received signals) use the convention "H" "L" "X" (receive high, low, don't care).

Each PCF vector has two timing events. The drive event that occurs at time zero and lasts for the `vector cycle`, and a receive event that occurs at the time `receive delay` time. In this example the receive event occurs 400 nano seconds after the input drivers become active.

There are a number of comments embedded within the PCF code to make interpretation easier. Every time the TAP State machine enters a new state a comment appears to the right of the vector indicating the current new state. The number of vectors is also indicated along with information regarding the current activity.

```
"10011X"      ! Test Idle
"00010X"      ! Test Idle
end pcf
wait 100m
pcf
"00011X"      ! Test Idle
!== Parallel prog Init/shift row all 0s
!== Parallel shift in instruction 3
"01011X"
"11011X"      ! Select DR
"01011X"
"11011X"      ! Logic Rst
```

The "wait 100 m" statement is used to specify a wait state. When the tester executes this test statement, it will hold the drive states on the last pcf vector encountered for the time indicated in the wait period. In this case the drivers will drive/receive "00010x" for 100 ms and then execute the next vector ("00011X"). This wait period in specific JTAG TAP states allows the MACH device to program. If any conversion is attempted from PCF to other ATE vector formats the wait periods must be preserved.

**Trademarks**

Copyright © 1996 Advanced Micro Devices, Inc. All rights reserved.

AMD, the AMD logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc.

Product names used in this publication are for identification purposes only and may be trademarks of their respective companies.