

# ***Programming Xilinx XC9500 CPLDs on HP 3070 Testers***

***Preface***

***Table of Contents***

***Introduction***

***Creating SVF Files***

***Creating Compiled Test Files***

***Appendix A: svf2vcl Options***

***Appendix B: Troubleshooting***

**XILINX**<sup>®</sup>, XACT, XC2064, XC3090, XC4005, and XC-DS501 are registered trademarks of Xilinx. All XC-prefix product designations, FastFLASH, FastCONNECT, EZTag, XACT-Floorplanner, XACT-Performance, XAPP, XAM, X-BLOX, X-BLOX plus, XChecker, XDM, XDS, XEPLD, XPP, XSI, BITA, Configurable Logic Cell, CLC, Dual Block, FastCLK, HardWire, LCA, Logic Cell, LogicProfessor, MicroVia, PLUSASM, SMARTswitch, UIM, VectorMaze, VersaBlock, VersaRing, and ZERO+ are trademarks of Xilinx. The Programmable Logic Company and The Programmable Gate Array Company are service marks of Xilinx.

IBM is a registered trademark and PC/AT, PC/XT, PS/2 and Micro Channel are trademarks of International Business Machines Corporation. DASH, Data I/O and FutureNet are registered trademarks and ABEL, ABEL-HDL and ABEL-PLA are trademarks of Data I/O Corporation. SimuCad and Silos are registered trademarks and P-Silos and P/C-Silos are trademarks of SimuCad Corporation. Microsoft is a registered trademark and MS-DOS is a trademark of Microsoft Corporation. Centronics is a registered trademark of Centronics Data Computer Corporation. PAL and PALASM are registered trademarks of Advanced Micro Devices, Inc. UNIX is a trademark of AT&T Technologies, Inc. CUPL, PROLINK, and MAKEPRG are trademarks of Logical Devices, Inc. Apollo and AEGIS are registered trademarks of Hewlett-Packard Corporation. Mentor and IDEA are registered trademarks and NETED, Design Architect, QuickSim, QuickSim II, and EXPAND are trademarks of Mentor Graphics, Inc. Sun is a registered trademark of Sun Microsystems, Inc. SCHEMA II+ and SCHEMA III are trademarks of Omaton Corporation. OrCAD is a registered trademark of OrCAD Systems Corporation. Viewlogic, Viewsim, and Viewdraw are registered trademarks of Viewlogic Systems, Inc. CASE Technology is a trademark of CASE Technology, a division of the Teradyne Electronic Design Automation Group. DECstation is a trademark of Digital Equipment Corporation. Synopsys is a registered trademark of Synopsys, Inc. Verilog is a registered trademark of Cadence Design Systems, Inc.

Xilinx does not assume any liability arising out of the application or use of any product described or shown herein; nor does it convey any license under its patents, copyrights, or maskwork rights or any rights of others. Xilinx reserves the right to make changes, at any time, in order to improve reliability, function or design and to supply the best product possible. Xilinx will not assume responsibility for the use of any circuitry described herein other than circuitry entirely embodied in its products. Xilinx devices and products are protected under one or more of the following U.S. Patents: 4,642,487; 4,695,740; 4,706,216; 4,713,557; 4,746,822; 4,750,155; 4,758,985; 4,820,937; 4,821,233; 4,835,418; 4,853,626; 4,855,619; 4,855,669; 4,902,910; 4,940,909; 4,967,107; 5,012,135; 5,023,606; 5,028,821; 5,047,710; 5,068,603; 5,140,193; 5,148,390; 5,155,432; 5,166,858; 5,224,056; 5,243,238; 5,245,277; 5,267,187; 5,291,079; 5,295,090; 5,302,866; 5,319,252; 5,319,254; 5,321,704; 5,329,174; 5,329,181; 5,331,220; 5,331,226; 5,332,929; 5,337,255; 5,343,406; 5,349,248; 5,349,249; 5,349,250; 5,349,691; 5,357,153; 5,360,747; 5,361,229; 5,362,999; 5,365,125; 5,367,207; 5,386,154; 5,394,104; 5,399,924; 5,399,925; 5,410,189; 5,410,194; 5,414,377; RE 34,363, RE 34,444, and RE 34,808. Other U.S. and foreign patents pending. Xilinx, Inc. does not represent that devices shown or products described herein are free from patent infringement or from any other third party right. Xilinx assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

Xilinx products are not intended for use in life support appliances, devices, or systems. Use of a Xilinx product in such applications without the written consent of the appropriate Xilinx officer is prohibited.

# Preface

---

## About This Manual

This manual describes how to program Xilinx XC9500 CPLDs on HP 3070 testers.

Before using this manual, you should be familiar with the operations that are common to all Xilinx's software tools: how to bring up the system, select a tool for use, specify operations, and manage design data.

## Manual Contents

This manual covers the following topics.

- Chapter 1, "Introduction," lays out the basic procedure for programming an XC9500 CPLD in an HP 3070 test environment.
- Chapter 2, "Creating SVF Files," discusses how to create an SVF files on PCs, and on Sun and HP workstations.
- Chapter 3, "Creating Compiled Test Files," discusses how to use **gen\_hp** to create compiled test files for use in the HP 3070 test environment.
- Appendix A, "**svf2vcl**," lists options for execution of the **svf2vcl** program. **svf2vcl** is the first part of the **gen\_hp** program.
- Appendix B, "Troubleshooting," contains troubleshooting options if programming fails.



## Conventions

---

In this manual the following conventions are used for syntax clarification and command line entries.

- **Courier font** indicates messages, prompts, and program files that the system displays, as shown in the following example.

```
speed grade: -100
```

- **Courier bold** indicates literal commands that you must enter in a syntax statement.

```
rpt_del_net=
```

- **Italic font** indicates variables in a syntax statement. See also, other conventions used on the following page.

```
xdelay design
```

- Square brackets “[ ]” indicate an optional entry or parameter. However, in bus specifications, such as bus [7:0], they are required.

```
xdelay [option] design
```

- Braces “{ }” enclose a list of items from which you choose one or more.

```
xnfprep designname ignore_rlocs={true|false}
```

- A vertical bar “|” separates items in a list of choices.

```
symbol editor [bus|pins]
```

Other conventions used in this manual include the following.

- *Italic font* indicates references to manuals, as shown in the following example.

See the *Development System Reference Guide* for more information.

- *Italic font* indicates emphasis in body text.

If a wire is drawn so that it overlaps the pin of a symbol, the two nets *are not* connected.

- A vertical ellipsis indicates repetitive material that has been omitted.

```
IOB #1: Name = QOUT'  
IOB #2: Name = CLKIN'  
.  
.  
.
```

- A horizontal ellipsis “...” indicates that the preceding can be repeated one or more times.

```
allow block blockname loc1 loc2 ... locn ;
```

## Table of Contents

Programming Xilinx XC9500 CPLDs on HP 3070 Testers	1
About This Manual	iii
Manual Contents	iii
Table of Contents	
Introduction	1-1
Hardware Considerations	1-2
Test Fixture Design Tips	1-2
Using PC or Sun	1-2
2-1	
Creating SVF Files	2-1
Creating an SVF File Using EZTag	2-1
On a Workstation	2-1
On a PC	2-3
Software Restrictions	2-5
3-1	
Creating Compiled Test Files	3-1
Using the gen_hp Script	3-1
TRST Optional Pin	3-3
svf2vcl	A-1
About svf2vcl	A-1
Running svf2vcl	A-1
File Splitting	A-3
Troubleshooting	B-1





# Chapter 1

## Introduction

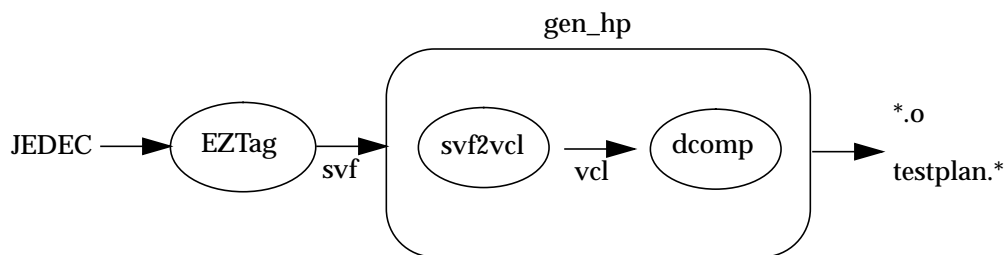
---

This document describes the procedures necessary to program Xilinx XC9500 CPLD designs in an HP 3070 test environment. The procedures described in this document lay out the necessary steps you need to perform; they are:

- Creating an SVF File Using EZTag
- Generating an HP 3070 ISP Program

The **gen\_hp** program translates Serial Vector Format (SVF) files to HP 3070 executable object files. This allows you to take SVF files created in EZTag and translate them for use in an HP 3070 test environment.

Instructions are included for both workstation and the PC environments. Chapter 2 describes the procedure for creating an SVF file. This procedure can be performed on a PC or on a Sun or HP workstation. Chapter 3 describes how to create HP 3070 test files using **gen\_hp** on an HP workstation. Appendix A describes details of the **svf2vcl** executable, including options. **svf2vcl** can be run on all platforms, while **dcomp** can only be run on an HP workstation.



**Figure 1-1 Program Flow**

## Hardware Considerations

This software and methodology works on the following HP testers:

- HP 3070
- HP 3072
- HP 3073
- HP 3074
- HP 3075
- HP 3079CT
- HP 3172
- HP 3173
- HP 3175

**Note:** Your HP 3070 configuration must include a “Control Plus” card.

## Test Fixture Design Tips

On the ATE loadboard you should add a 100 ohm pullup on TCK and a 100 ohm pulldown on TMS to ensure the stability of these signals between vector file loads. The resistors should be installed right at the corresponding probe points on the HP 3070.

Make certain that the 4 pins of the 1149.1 TAP (TCK, TMS, TDI and TDO) are marked as critical signals to the HP 3070. Always use as short leads as possible to connect these 4 pins from the load board to the tester.

## Using PC or Sun

Only EZTag and the **svf2vc1** translation can be run on these platforms, since the HP 3070 object compiler (**dcomp**) is supported only on the HP workstation. To use a PC or Sun for **svf2vc1**, see Appendix A.

## Chapter 2

### Creating SVF Files

---

#### Creating an SVF File Using EZTag

This procedure describes how to create an SVF file; it assumes that the Xilinx XACT-CPLD version 6.0.1 (or newer) or XABEL-CPLD version 6.1.1 (or newer) is being used, which includes the XC9500 fitter and the EZTag software.

**Note:** XABEL-CPLD runs on PCs only.

#### On a Workstation

If you have a **Sun or HP workstation**, from the XACT command line use the following procedure:

1. Fit the design and create a JEDEC output file.
2. Invoke the EZTag software

```
eztag -svf
```

The following message appears:

```
Xilinx (R) EZTAG XC9500-CPLD-6.0.3-JTAG Boundary-Scan  
Download
```

```
Copyright (C) Xilinx Inc. 1991-1996. All Rights  
Reserved.
```

```
-----  
SVF GENERATION MODE.  
EZTAG ?
```

3. Set up the device types and assign design names.

On a **Sun or HP workstation**, type the following command at the EZTAG ? prompt:

```
part deviceType1:designName1
    deviceType2:designName2
    deviceTypeN:designNameN <CR>
```

where **deviceType** is the name of the BSDL file for that device and **designName** is the name of the design to translate into SVF. Multiple *deviceType:designName* pairs are separated by spaces. For example:

```
part xc95108:abc12 xc95216:wxyz
```

This command defines the JTAG device chain, from one to any number of devices. The parts specified in the **part** command should be arranged in order beginning with the first device to receive TDI and ending with the last device to output TDO.

**Note:** For any non-XC9500 part in the JTAG chain make certain that the BSDL file for the specified part is available along the XACT path and is called *deviceType.bsd*. The design name in this case can be any arbitrary name.

4. Execute an EZTag operation.
  - **erase designName** — generates an SVF file for the bit sequence needed to erase the specified part.
  - **verify designName [-j jedecFileName]** — generates an SVF file that specifies the bit sequence to read back the device contents and compare it against the contents of the specified JEDEC file.
  - **program [-b] designName [-j jedecFileName]** — generates an SVF file that specifies the bit sequence to program the specified part from a JEDEC file named *designName.jed* (or alternately, the JEDEC file name specified after the “-j”). The program command option adds the following functionality:
    - b — Used to specify programming of a blank part. This is typically specified for parts delivered blank from the factory.
5. Exit EZTag

On a **Sun or HP workstation** you will exit EZTag by entering the following command:

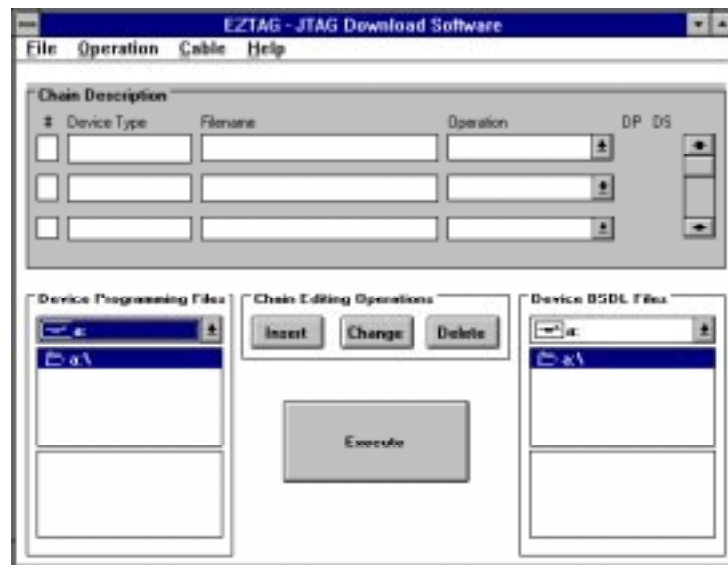
```
quit
```

**Note:** The SVF file will be named *designName.svf*, and will be created in the current working directory (the directory in which EZTAG is being run). Consecutive operations on the same *designName* file will overwrite the SVF file each time. The SVF file contains all data and commands necessary to perform the specified function.

## On a PC

If you are using a PC with Windows 3.1 or 95, use the following procedure.

1. Fit the design and create a JEDEC output file.
2. Double-click on the EZTag icon. The EZTag-JTAG Download Software will appear.



**Figure 2-1 JTAG Programmer**

3. Assign device types for each device in the chain. For each device:
  - a) Under **Device Programming Files** find the disk and directory containing the files you want to use.
  - b) Double-click on the file name. The **Device Type** and **File name** should appear in the **Chain Description**.

- c) Click once on the **Operation** down arrow to select the type of operation you want to select. Select **Program** (this is the default).
- 4. Specify the BSDL files for these parts as follows:
  - a) Under **Device BSDL Files**, find the disk and directory containing the files you want to use.
  - b) Double-click on the *filename* of the device. Type "OTHER." The selected *filename* should appear in the chain description.
  - c) The operation will be set to BYPASS and cannot be changed.

**Note:** For any non-XC9500 part in the JTAG chain make certain that the BSDL file for the specified part is available along the XACT path and is called *deviceType.bsd*. The design name in this case can be any arbitrary name.

- 5. Highlight one of the devices by clicking on it once with the mouse, then select one of the following operations:

**Operations > Erase**

generates an SVF file for the bit sequence needed to erase the specified part.

**Operations > Verify**

generates an SVF file that specifies the bit sequence to read back the device contents and compare it against the contents of the specified JEDEC file.

**Operations > Program**

generates an SVF file that specifies the bit sequence to program the specified part from a JEDEC file named *designName.jed*.

- 6. On a **PC** you will exit with:

**File > Exit**

**Note:** The SVF file will be named *designName.svf*, and will be created in the current working directory (the directory in which EZTAG is being run). Consecutive operations on the same *designName* file will overwrite the SVF file each time. The SVF file contains all data and commands necessary to perform the specified function.

## **Software Restrictions**

EZTAG can generate SVF files only for devices for which XC9500 JEDEC files can be created.

The current software can only generate SVF files for operations on one part at a time. If there are several parts to be programmed, additional program commands must be executed — one for each part, creating multiple SVF files. In each SVF file, one device will be programmed while the others are held in bypass mode.





## Chapter 3

### Creating Compiled Test Files

---

#### Using the `gen_hp` Script

If you are using the supplied HP workstation as the system controller for the HP 3070 ATE, this script will build all the necessary Vector Control Language (VCL) object and testplan files to generate a complete HP 3070 vector sequence to perform programming and, optionally, readback verification via the JTAG TAP.

Use the SVF file (or files if you have generated a verification file as well) that you generated above as input to the `gen_hp` tool. This tool takes SVF files and creates executable HP 3070 vector programs.

The `gen_hp` tool is run on the HP workstation that acts as the controller for the HP 3070. Create a directory called “svf” and another called “digital”. Copy all the SVF files to the “svf” directory. Before starting the `gen_hp` program you might want to modify it to suit your application. The `gen_hp` script is reproduced in Figure 3-1. The modifications that can be applied are generally to the call to `svf2vcl` which performs the SVF to VCL conversion. VCL (Vector Control Language) is the HP 3070 stimulus description language. The available switches for this program are listed in Table 1 of Appendix A.

To use make use of script for customizing your test procedure, follow these instructions:

1. Create an “svf” and a “digital” directory in the board directory.

```
mkdir svf
```

```
mkdir digital
```

2. Copy your svf files to the svf directory.

3. If necessary, edit **gen\_hp** to add extra options to **svf2vc1** command. An example of useful options:

```
-TCKNODE node_name Sets the TCK node name
-TMSNODE node_name Sets the TMS node name
-TDINODE node_name Sets the TDI node name
-TDONODE node_name Sets the TDO node name
-NOCOMMENTS No Comments
-COMMANDCOMMENTS Command Comments
```

See Appendix A for a complete list.

4. Execute from the board directory:

```
gen_hp prog_vector_file.svf verif_vector_file.svf
```

The script generates all the HP 3070 objects in the digital directory. It will also generate a **testplan.file** in the board directory that you can add to your existing testplan to call to the new subroutines to program and verify the part.

```
#!/bin/sh
#Create a directory called svf under the board dir.
#Files in the svf dir should have the .svf extension.
#If you have an verify file put it in the svf dir called filename.svf
#Fill out files variable for all the .svf files.
file=$1
vfile=$2
filename=`echo $file | cut -f 1 -d .`
testplan="testplan."$filename
echo > $testplan
echo $file
echo "\n\nsub ${filename}_svf2vc1" >> $testplan
echo "! " >> $testplan
echo "! APG Test Consultants, Inc." >> $testplan
echo "! " >> $testplan
echo "Time_s = msec" >> $testplan
echo "print \"Programming ${filename} into device.\"\" >> $testplan
svf2vc1 -trstnode "" svf/$file digital/$filename
for i in `ls digital/$filename.v[0-9][0-9] | sort` do
echo $i
echo " test \"$i\"\" >> $testplan
# to produce debug object change following line to dcomp -D $i
dcomp $i done
echo "Time_e = msec" >> $testplan
```

```
echo "print \"Prog. Time = \"&val\${(Time_e - Time_s)/1000}" >> $testplan
echo "subend" >> $testplan

if [ -f svf/$vfile ] then
echo $vfile filename=`echo $vfile | cut -f 1 -d .`
echo "\n\nsub ${filename}_svf2vc1" >> $testplan
echo "! " >> $testplan
echo "! APG Test Consultants, Inc." >> $testplan
echo "! " >> $testplan echo "Time_s = msec" >> $testplan
echo "print \"Verifying ${filename} for device.\" " >> $testplan
svf2vc1 -verify -trstnode "*" svf/$vfile digital/${filename}
for i in `ls digital/${filename}.v[0-9][0-9] | sort` do
echo $i
echo " test \"$i\" " >> $testplan
# to produce debug object change following line to dcomp -D $i
dcomp $i
done
echo "Time_e = msec" >> $testplan
echo "print \"Verifying Time = \"&val\${(Time_e - Time_s)/1000}" >>
$testplan
echo "subend" >> $testplan fi
```

Figure 3-1 gen\_hp Script

## TRST Optional Pin

The XC9500 parts do not have a TRST pin. If however, other parts on your system do have a TRST pin, you must modify the **svf2vc1** command line to specify the TRST pin name using the **-TRSTNODE** switch.



## Appendix A

### svf2vcl

---

#### About svf2vcl

The **svf2vcl** translation file can be run on all platforms; however, since the HP 3070 object compiler (**dcomp**) is supported only on the HP workstation, **gen\_hp** script can only be run on HP workstations.

If you do not want to use **gen\_hp**, or want to run **svf2vcl** separately, you can use a PC, or a Sun or HP workstation as follows:

1. Generate a **.svf** file as described in *Creating SVF Files*.
2. Run the **svf2vcl** program as described below.
3. Make a directory called **digital** in your HP workstation's file system. Copy all the **.vcl** files that were generated by running **svf2vcl** to this directory
4. Run **gen\_hp** to create your **\*.o** and **testplan.\*** files. Your HP 3070 object files will be created in the **digital** directory.

#### Running svf2vcl

**svf2vcl** can be run in a SunOS, HP-UX or Windows NT/95 environment. The command syntax for running the program is:

```
svf2vcl [options] input_filename output_filename
```

Remember not to add a file extension to the *output\_filename*.

To translate XC9500 SVF programming files, use:

```
svf2vcl -trstnode "" svf/file.svf digital/file
```

To translate XC9500 SVF verify files, use:

```
svf2vcl -verify -trstnode "" svf/filev.svf digital/  
file
```

To display available options, type:

**svf2vcl -?**

The **svf2vcl** switches are as follows:

**Table A-1 svf2vcl Switches**

<b>Abbr.</b>	<b>Option</b>	<b>Operation</b>
-NC	-NOCOMMENTS	No comments
-CC	-COMMANDCOMMENTS	Command comments
-FC	-FULLCOMMENTS	Full comments (default)
-Q	-QUIET	Quiet running
-V	-VERBOSE	Verbose output
-VV	-VERYVERBOSE	Very verbose output
-NOWC	-NOWAITCOMMENT	Don't convert wait comments to waits (default)
-WC	-WAITCOMMENT	Convert wait comments to waits
-NOCR	-NOCONVERTRUNTEST	Don't convert RUNTESTS to waits
-CR	-CONVERTRUNTEST	Convert RUNTESTS to waits (default)
	-NORETRY or -NOLOOP or -VERIFY	Disable retry mode
	-RETRY or -LOOP or -NOVERIFY	Enable retry mode (default)
-LC #	-LOOPCOUNT #	Loop count is #
-LA #	-LOOPADJUST #	Loop adjust is #
-CV	-CMPVECT #	Compare vector is #
	-TCKNODE "node name"	Set the TCK node name
	-TMSNODE "node name"	Set the TMS node name
	-TDINODE "node name"	Set the TDI node name
	-TDONODE "node name"	Set the TDO node name
	-TRSTNODE "node name"	Set the TRST node name
	-DD "device name"	Set the default device

Table A-1 svf2vcl Switches

Abbr.	Option	Operation
	-DEFAULTDEVICE "device name"	Set the default device
-?	-HELP	Display this screen

**Note:** If your boundary-scan Test Access Port pin names are non-standard do not forget to add the options to tell **svf2vcl** the node-names of TCK, TMS, TDO, TDI and TRST.

Comment options can be used to reduce the digital VCL source size. The quiet and verbose options will adjust how much information is displayed while running. The wait comments options allow for control of the conversion of a wait in a comment to a wait in the VCL code. The convert runtest to wait options allow for control of the conversion of runtest into wait commands to greatly reduce the size of the VCL source code. During programming of Xilinx devices a retry loop is enabled which allows for retrying of the programming of locations. If the SVF file is used for verifying the programmed device, use the **-verify** option to disable the retry loop. If the retry loop is enabled there are other options which can be used.

The LOOPCOUNT option can be used to change the default number of times that the retry loop will be executed. The LOOPADJUST option tells the file splitting routine how much to weight each loop. By increasing the value above one the loop will count move vectors which will result in more files. Use this option if your digital compiler is having problems compiling large files.

The CMPVECT option determines which bit of the output on TDO is used to verify that a location has been programmed. For Xilinx devices that are not in a chain do not change this option.

The node name options are used to set the actual node names for TCK, TDI, TDO, TMS and TRST.

## File Splitting

The HP 3070 tester has limitations to the size of the VCL test file. A large SVF file of the sort required for XC9500 programming or verification is likely to produce a VCL file too large for the HP 3070 tester RAM to handle. The **gen\_hp** program will automatically create multiple VCL files of appropriate size for the tester to handle.

To accomplish this the translator counts the number of lines translated to VCL and splits the file after a maximum line count. The current line count is compared to the maximum line count before each executable SVF command. If the current line is greater than or equal to the maximum line count then the state machine is forced to the Run-Test/Idle state, the current file is closed, the next file is opened and translation continues. The maximum line count value can be adjusted upwards on HP 3070s by installing large amounts of vector program memory. When the state machine goes to the Run-Test/Idle state, the HP 3070 will release the drivers on the JTAG control input pins (TCK, TMS, TDI and TRST). This is the reason for installing the 100 ohm pullup and 100 ohm pulldown resistors on TCK and TMS, respectively.



## Appendix B

### Troubleshooting

---

ATE environments tend to be very noisy. The presence of electrical noise can contribute to erratic ISP behavior. Consider the following tips if you suspect noise problems.

- Set ATE drive levels to 5V to minimize glitch effects in noisy environments.
- Experiment with the TCK/TMS pull/up/down values to provide more stable TCK and TMS signals during vector loads.
- Consider connection of a 1 nF capacitor in parallel with the TCK and TMS pullup/pulldown resistors if you experience erratic programming failures.
- Vary the vector cycle times and slew rates to increase noise immunity.
- Consider using twisted pair connections for TAP signals to reduce noise transmitted with signal values.

