Programming Xilinx XC9500 CPLDs on HP 3070 Testers

Revision 1.4 *September, 1998* *Printed in U.S.A.*

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 **svf2vc**l 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* `;`

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 SVF Files Using JTAG Programmer
- Generating an HP 3070 ISP Program

The **gen_hp** script translates Serial Vector Format (SVF) files to HP 3070 executable object files. This allows you to take SVF files created in JTAG Programmer and translate them for use in an HP 3070 test environment.

Instructions are included for both GUI and the command line. 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.

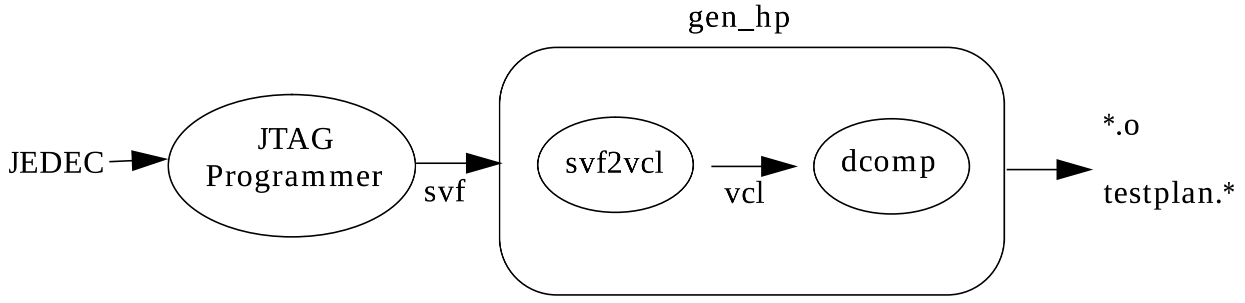


Figure 1-1 Program Flow

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 JTAG Programmer and the **`svf2vcl`** 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 **`svf2vcl`**, see Appendix A.

Chapter 2

Creating SVF Files

Creating an SVF File Using JTAG Programmer

This procedure describes how to create an SVF file; it assumes that you are using Xilinx Foundation or Alliance Series software, Version 1.3 or newer. These software packages include the Xilinx CPLD fitter and JTAG Programmer software. JTAG Programmer is available free of charge on the Xilinx World Wide Web site, www.xilinx.com.

JTAG Programmer is supplied with both graphical and batch user interfaces. The batch user interface executable name is **`jtagprog`**; and the graphical user interface is named **`jtagpgmr`**. The graphical tool can be launched from the Design Manager or Project Manager, but may also be launched by opening a shell and invoking **`jtag-pgmr`**. The batch tool is available by opening a shell and invoking **`jtagprog`** on the command line.

The goal of the following procedure is to create three separate SVF files for each device being programmed. We will show you how to do this using both the batch and the GUI tool. One SVF file contains erase information for the device, another the program information for the device, and the third contains verification information. On XC9500 devices the erase vectors should have a 2 ms TCK period.

Using the Batch Download Tool to Generate SVF Files

1. Run your design through the Xilinx fitter and create a JEDEC programming file. You may already have been provided with a JEDEC file; if so, proceed to the next step.
2. Invoke the batch JTAG Programmer tool from the command line in a new shell.

```
jtagprog -svf
```

The following messages will appear:

```
JTAGProgrammer: version <Version Number>
Copyright:1991-1998

Sizing system available memory...done.

***SVF GENERATION MODE***

[JTAGProgrammer::(1)]>
```

3. Set up the device types and assign design names by typing the following command sequence at the JTAG Programmer prompt:

   `part` *`deviceType1:designName1 deviceType2:designName2 ... deviceTypeN:designNameN`*

   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:ww133`

   The **`part`** command defines the composition and ordering of the boundary-scan chain. The devices are arranged with the first device specified being the first to receive TDI information and the last device specified being the one to provide the final TDO data.

Note: For any non-XC9500(XL) device in the boundary-scan chain, make certain that the BSDL file is available either in the XILINX variable data directory, or by specifying the complete path information in the *deviceType*. The *designName* in this case can be any arbitrary name.

4. Execute the required boundary-scan or ISP operation in JTAG Programmer.

   - **`erase`** [**`-fh`**] *designName* -- generates an SVF file to describe the boundary-scan sequence to erase the specified part. The **`-f`** flag generates an erase sequence that overrides write protection on devices. The **`-h`** flag indicates that all other parts (other than the specified *designName*) in the boundary-scan chain should be held in the HIGHZ state during the erase operation. Xilinx recommends **`erase -f -h`** *designName*.
   - **`verify`** [**`-h`**] *designName* [**`-j`** *jedecFileName*] -- generates an SVF file to describe the boundary-scan sequence to read back

the device contents and compare it against the contents of the specified JEDEC file. The JEDEC file defaults to be the *designName*.**`jed`** in the current directory, or may be alternatively specified using the **`-j`** flag. The **`-h`** flag is used to specify that all other parts (other than the specified *designName*) in the boundary-scan chain should be held in the HIGHZ state during the verify operation. Xilinx recommends **`verify -h`** *designName*.

- **`program`** [**`-bh`**] *designName* **`-j`** [*jedecFileName*] -- generates an SVF file to describe the boundary-scan sequence to program the device using the programming data in the specified JEDEC file. The JEDEC file defaults to be *designName*.**`jed`** in the current directory, or may be alternatively specified using the **`-j`** flag. The **`-h`** flag is used to specify that all other parts (other than the specified *designName*) in the boundary scan chain should be held in the HIGHZ state during the programming operation. The **`-b`** flag indicated the programming operations should erase the device. This is useful when programming devices shipped from the factory. Xilinx recommends **`program -b -h`** *designName*.
- **`partinfo`** [**`-h`**] **`-idcode`** *designName* -- generates an SVF file to describe the boundary-scan sequence to read back the 32 bit hard-coded device IDCODE. The **`-h`** flag is used to specify that all other parts (other than the specified *designName*) in the boundary scan chain should be held in the HIGHZ state during the IDCODE operation.This operation can be performed in any combination of the three SVF files.
- **`partinfo`** [**`-h`**] **`-signature`** *designName* -- generates an SVF file to describe the boundary-scan sequence to read back the 32 bit user-programmed device USERCODE. The **`-h`** flag is used to specify that all other parts (other than the specified *designName*) in the boundary scan chain should be held in the HIGHZ state during the USERCODE operation. This operation can be performed in any combination of the SVF files.

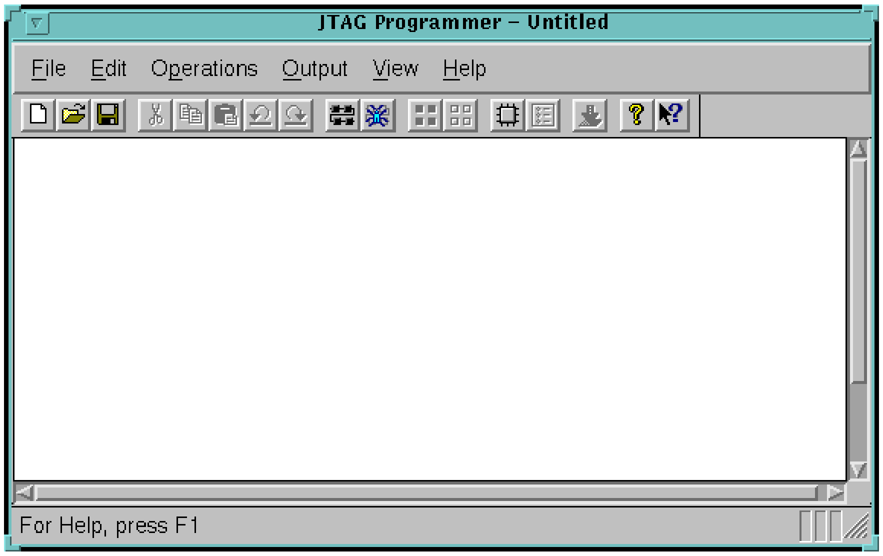
5. Exit JTAG Programmer by entering the following command:

```
quit
```

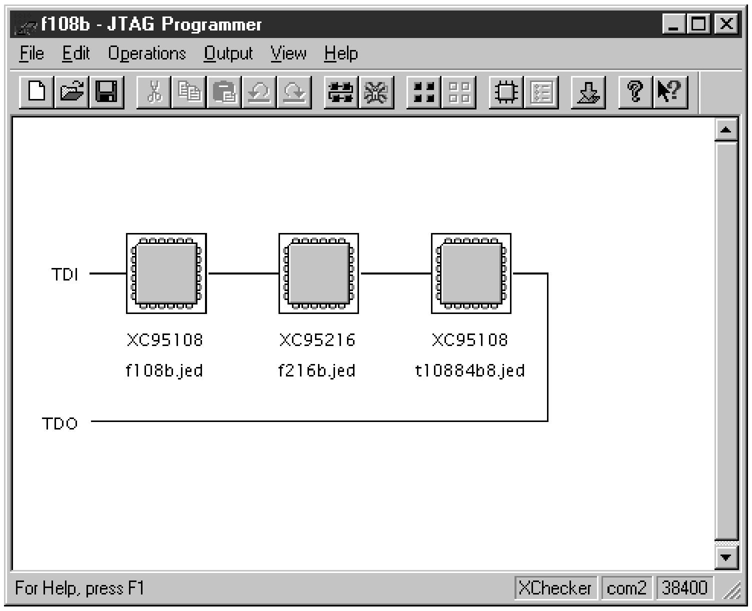
Note: The SVF file will be named *designName*.**`svf`** and will be created in the current working directory. Consecutive operations on the same *designName* will append to the SVF file. To create SVF files with separate operations in each, you will need to rename the SVF file after each operation by exiting to the system shell.

Using the Graphical User Interface to Generate SVF Files

1. Run your design through the Xilinx fitter and create a JEDEC file (you may already have been provided with one).
2. Double-click on the JTAG Programmer icon or open a shell and type **`jtagpgmr`**. The JTAG Programmer will appear.


JTAG Programmer – Untitled
File Edit Operations Output View Help
For Help, press F1

3. Instantiate your boundary-scan chain. There are two ways to do this. The first is to manually add each device in the correct boundary-scan order from system TDI to system TDO.
   a) Selecting **`Edit`** → **`Add Device`** for each device in the boundary-scan chain.
   b) Fill in the device properties dialog to identify the JEDEC (if it is an XC9500 device) or BSDL (if it is not an XC9500 device) file associated with the device you are adding.



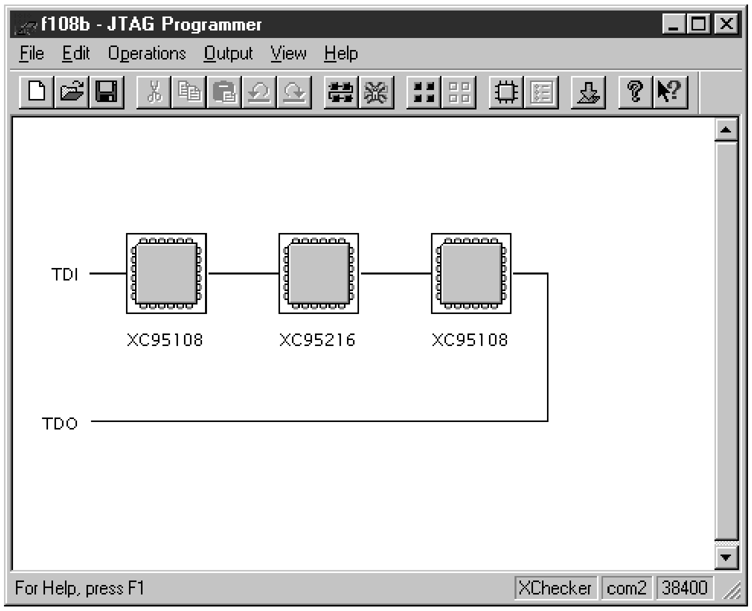
The device type and JEDEC file name will appear below the added device.

The second method is to allow JTAG Programmer to query the boundary-scan chain for devices, and then fill in the JEDEC and BSDL file information. This method will work only when you have the target system connected to your computer with a Xilinx serial or parallel cable. The cable must be powered up by the board under test. The steps are as follows:

a) Initialize the chain as follows:

File → Initialize Chain

JTAG Programmer will display the boundary-scan chain configuration as shown:



b) For each device in the resulting chain, double-click on the chip icon to bring up the device properties dialog, then select the JEDEC or BSDL file associated with that device.

4. Put the JTAG Programmer into SVF mode by selecting

   Output → Create SVF File...

   to create a new SVF file, or

   Output → Append to SVF File...

   to append to an existing SVF file. Fill in the SVF file dialog with the desired name of the target SVF file to be created.

Note: Once you enter SVF mode the composition of the boundary-scan chain cannot be edited in order to ensure consistency of the boundary-scan data in the SVF file.

5. Highlight one of the devices by clicking it once with the mouse. Then, select any of the enable operations from the Operations pull down menu to generate an SVF file to describe the boundary-scan sequence to accomplish the requested operation.

6. When you completed the required operations you may exit JTAG Programmer by selecting:

   File → Exit

Note: You may select **`Use HIGHZ instead of BYPASS`** from the **`File`** → **`Preferences...`** dialog to specify that all other parts (not the device selected) in the boundary-scan chain will be held in HIGHZ state during the requested operation.

Note: To generate separate SVF files for each operation you will have to perform the following steps between operations:

a) Select **`Output`** → **`Use Cable...`**

b) On the **`Cable Communications`** dialog box select **`Cancel`**

c) Select **`Output`** → **`Create SVF File..`**

d) Choose a new SVF file and proceed normally.

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 erase, 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 make use of the 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 **svf2vcl** 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***

   When accessing a device in a boundary-scan chain, the following options need to be specified:

   -NUMDEVICES *number* ***Specifies the total number of devices in the boundary-scan chain***

   -TARGETDEVICE *number* ***Specifies which device in the boundary-scan chain is being targeted***

   See Appendix A for a complete list of options.

4. Execute from the `board` directory:

   gen_hp *erase_vector_file***.svf** *prog_vector_file***.svf** *verify_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
filenamev.svf
#Fill out files variable for all the .svf files.
efile=$1
file=$2
vfile=$3
#
#  Process Erase
#
if [ -f svf/$efile ]
```

```
then
echo $efile
filename=`echo $efile | cut -f 1 -d .`
testplan="testplan."$filename
echo > $testplan
echo "\n\nsub ${filename}_svf2vcl" >> $testplan
echo "! " >> $testplan
echo "! APG Test Consultants, Inc." >> $testplan
echo "! " >> $testplan
echo "Time_s = msec" >> $testplan
echo "print \"Programing ${filename} into device.\"" >> $testplan
svf2vcl -trstnode "*" svf/$efile 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 \"Erase. Time = \"&val\$((Time_e - Time_s)/1000)" >>
$testplan
echo "subend" >> $testplan
fi
#
#  Process Program
#
if [ -f svf/$file ]
then
filename=`echo $file | cut -f 1 -d .`
echo $file
echo "\n\nsub ${filename}_svf2vcl" >> $testplan
echo "! " >> $testplan
echo "! APG Test Consultants, Inc." >> $testplan
echo "! " >> $testplan
echo "Time_s = msec" >> $testplan
echo "print \"Programing ${filename} into device.\"" >> $testplan
svf2vcl -trstnode "*" svf/$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
fi

#
#  Process Verify
#
if [ -f svf/$vfile ]
then
echo $vfile
filename=`echo $vfile | cut -f 1 -d .`
echo "\n\nsub ${filename}_svf2vcl" >> $testplan
echo "! " >> $testplan
echo "! APG Test Consultants, Inc." >> $testplan
echo "! " >> $testplan
echo "Time_s = msec" >> $testplan
echo "print \"Verifying ${filename} for device.\"" >> $testplan
svf2vcl -verify -trstnode "*" svf/$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

TCK Period

Erase vectors should be run with a TCK clock rate of 500 Hz. Program and verify can be run at 2 MHz.

Accessing Devices in a Multipart Boundary-scan Chain

In order to access a device situated in a multipart boundary-scan chain you must make certain that you follow these steps:

1. Generate a single SVF file that accesses the targeted device. For instance, if you are accessing a nine part chain do not concatenate an SVF file that programs device 3 with one that programs device 5. Keep these SVF files separate.
2. Run the SVF file through **`svf2vcl`**, specifying:
   a) the total number of devices in the boundary-scan chain using the **`-NUMDEVICES`** switch.
   b) the position of the targeted device in the boundary-scan chain using the **`-TARGETDEVICE`** switch. The target position is specified ordinally from the system TDI input.

For instance, in the system illustrated below,

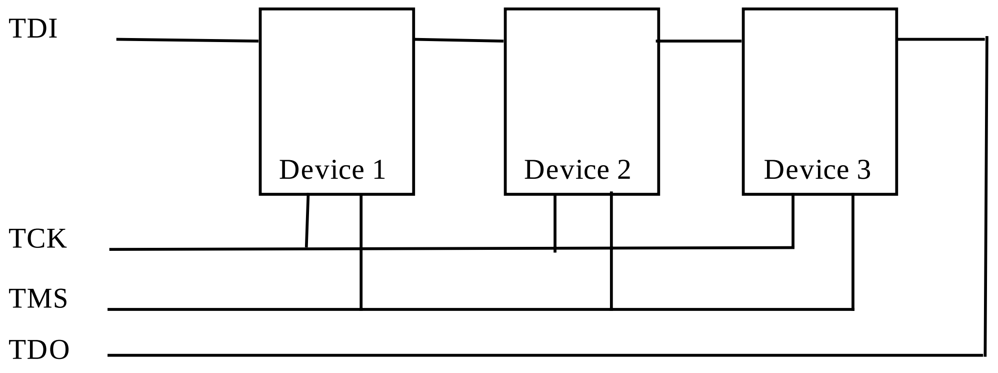


Figure 3-2 Three Device Boundary-Scan Chain

if you generated an SVF file to program Device 2 in the three device boundary-chain using EZTag, then you must use the switch values:

```
-NUMDEVICES 3 -TARGETDEVICE 2
```

for **`svf2vcl`**.

Similarly, if you have an SVF file to program Device 1, then you should choose **`-NUMDEVICES 3`** and **`-TARGETDEVICE 1`** as the svf2vcl switch value.

It is important that you select these values accurately and correctly as **`svf2vcl`** has no way to validate the veracity of the numbers entered.

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 **`svf2vcl`** 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

Abbr.	Option	Operation
-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
-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 #
	-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
-VL #	-VECTLIMIT	Maximum test vector count per VCL file
-ND	-NUMDEVICES #	Number of devices in the boundary-scan chain is #

Table A-1 svf2vcl Switches

Abbr.	Option	Operation
-TD	-TARGETDEVICE #	Device # in the boundary-scan being accessed
	-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 convert runtest to wait options allow for control of the conversion of runtest into wait commands; this option is required for the Xilinx erase and program algorithm. 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 to reduce the VCL source file size.

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 more vectors which will result in more files. Use this option if your digital compiler is having problems compiling large files. For more information on using this option see the File Splitting section below.

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. If you cannot increase the amount of vector program memory, you can increase the number of files and decrease the number of patterns in each file by increasing the value specified in the **-LOOPADJUST** switch to being some number greater than 1 (first try 10 and then scale up or down as required).

The value assigned to the LOOPADJUST switch indicates the relative memory requirement of the VCL statements associated with the homing loops. A value of 10 means that these VCL statements require 10 times more memory than ordinary (non-homing-loop) statements. Because the memory resources on each HP3070 vary from installation to installation, a trial and error method must be used to determine the correct value.

An alternative method is to use the -VECTLIMIT switch to specify the maximum number of test vectors per VCL file generated. This switch, however, can result in less efficient use of available ATE memory.

When the state machine goes to the Run-Test/Idle state as it is loading a new stimulus file, 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.