Programming Xilinx CPLDs on GENRAD Testers

Revision 1.8.6

Preface

Introduction

Creating SVF Files

Creating GenRad Test Files

DTS Example and Explanation

Optimizations

Using the Batch Download Tool to Generate SVF Files

May 4, 2000 *Printed in U.S.A.*

Preface

About This Manual

This manual describes how to program Xilinx CPLDs on GenRad testers.

Table 0-1 Supported Device Families

XC9500 Series	XC9500 (5V) XC9500XL (3.3V)
CoolRunner	**XPLA Original** XCR3128 (3.3V) XCR5128 (5V) **XPLA Enhanced** XCR5000C (5V) XCR3000A (3.3V) XCR3000C (3.3V) **XPLA3** XCR3000XL (3.3V)

Manual Contents

This manual covers the following topics.

- Chapter 1, “Introduction,” lays out the basic procedure for programming a Xilinx CPLD in a GenRad test environment.
- Chapter 2, “Creating SVF Files,” discusses how to create SVF files using JTAG Programmer or XPLA Programmer on a PC.

- Chapter 3, “Creating GenRad Test Files,” discusses how to use **`svf2dts`** to create compiled test files for use in the GenRad test environment.
- Appendix A, “DTS Example and Explanations,” provides a programming example.
- Appendix B, “Optimization,” contains some optimization hints.
- Appendix C, “Using the Batch Download Tool to Generate SVF Files Using JTAG Programmer,” describes using the batch programming tool.

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 assumes that you have obtained a JEDEC file for a given design, and describes the procedure necessary to program Xilinx CPLDs with a GenRad in-circuit test system. This document presents the steps you need to perform, which includes:

- Creating an serial vector format (SVF) file using Xilinx's Programmer software
- Generating a GenRad digital test source (**.dts**) using Xilinx's **svf2dts** version 1.8.6 software
- Merging a digital test source and other GenRad files into the test program

Xilinx programmer software translates a JEDEC file (**.jed**) into a Serial Vector Format (**.svf**) file. The Xilinx **svf2dts** v1.8.6 tool translates a **.svf** file to a GenRad digital test source, and is maintained by GenRad. Xilinx **svf2dts** v1.8.6 has been designed specifically to provide support for XC9500 and XC9500XL **.svf** file constructs created by JTAG Programmer using the program flow illustrated in Figure 1-1.

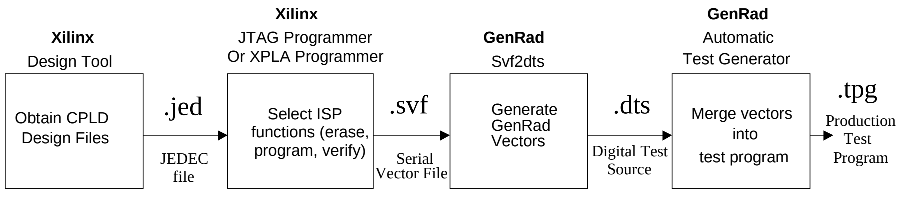


Figure 1-1 Program Flow

Note: GenRad and Xilinx have successfully verified the flow with all members of the XC9500 and XC9500XL families. Certain Xilinx Cool-Runner devices can be programmed with the GenRad tester by replacing JTAG Programmer with XPLA Programmer (available free of charge at www.xilinx.com/sxpresso/webpack.htm. GenRad's ISP support page contains more information related to GenRad's support of Xilinx's ISP-capable PLDs. See www.genrad.com/CSC/isp/default.htm)

The Xilinx **svf2dts** v1.8.6 program runs under Windows NT, 95, or 98. JTAG Programmer runs under Windows 95/98 or NT on a PC.

Chapter 2 describes the procedure for creating an SVF file from JTAG Programmer or XPLA Programmer on a PC.

Chapter 3 describes how to produce the GenRad digital test source file (**.dts**) and merge it into the a test program for production programming.

Hardware Requirements

This software and ISP methodology applies to the following Genrad in-circuit tester families running GenRad software release 3.2 or greater.

- GR2287L/LX
- GR228X *i*-Series
- TS121
- TS8X

Software Requirements

Xilinx Programmer Software

Software modules can be downloaded for free from http://www.xilinx.com/sxpresso/webpack.htm

- JTAG Programmer (for XC9500/XL)
- XPLA Programmer (for CoolRunner)

Genrad Translator svf2dts

Xilinx svf2dts v1.8.6 program requires a GenRad ISP Tools license. **svf2dts** version 1.8.6 is a standalone program complete with a graphical user interface and runs under Windows NT, 95, or 98. The program uses only 164 kilobytes of memory, but requires at least 24 megabytes of hard disk for output files.

- http://www.xilinx.com/isp/ate.htm

Genrad Software

- http://www.genrad.com

Chapter 2

Creating SVF Files

Introduction

Xilinx SVF files are created with the JTAG Programmer for XC9500/XL devices, or the XPLA Programmer for Xilinx CoolRunner Devices. This chapter is divided into two sections:

"Using the JTAG Programmer to Generate SVF Files for XC9500/XL Devices" section

"Using the XPLA Programmer to Generate SVF Files for CoolRunner Devices" section

In addition, there is a batch tool for generating SVF files for XC9500/XL devices. For information on using the JTAG Programmer batch tool, see Appendix C.

Using the JTAG Programmer to Generate SVF Files for XC9500/XL Devices

This procedure describes how to create an SVF file for XC9500/XL devices; it assumes that you have JTAG Programmer, Version 2.1i or newer. JTAG Programmer is included with the Xilinx Foundation or Alliance Series software. JTAG Programmer is also available free of charge with the *Device Programming Tools* on the Xilinx World Wide Web site, http://www.xilinx.com/sxpresso/webpack.htm.

JTAG Programmer is supplied with both graphical and batch user interfaces. The batch user interface executable name is **jtagprog**; and the graphical user interface executable 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 **jtagpgmr**. The batch tool is available by opening a shell and invoking

jtagprog on the command line. For information on using the JTAG Programmer batch tool, see Appendix C.

In order to program XC9500(XL) devices, the procedure is to **create a separate SVF file for each device being programmed**.

1. Run your design through the Xilinx fitter and create a JEDEC file.
2. Double-click on the JTAG Programmer icon or open a shell and type **`jtagpgmr`**. The JTAG Programmer interface will appear.

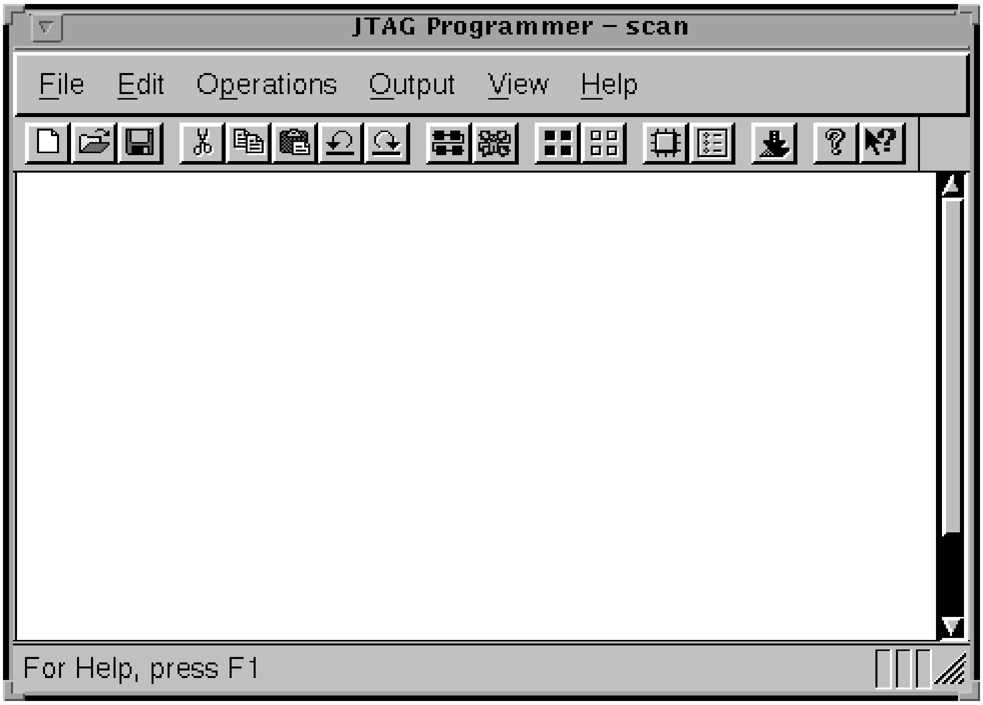


Figure 2-1 JTAG Programmer Interface

3. Instantiate your boundary-scan chain: Manually add each device in the correct boundary-scan order from system TDI to system TDO.
   a) Select **`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(XL) device) or BSDL (if it is not an XC9500(XL) device) file associated with the device you are adding.

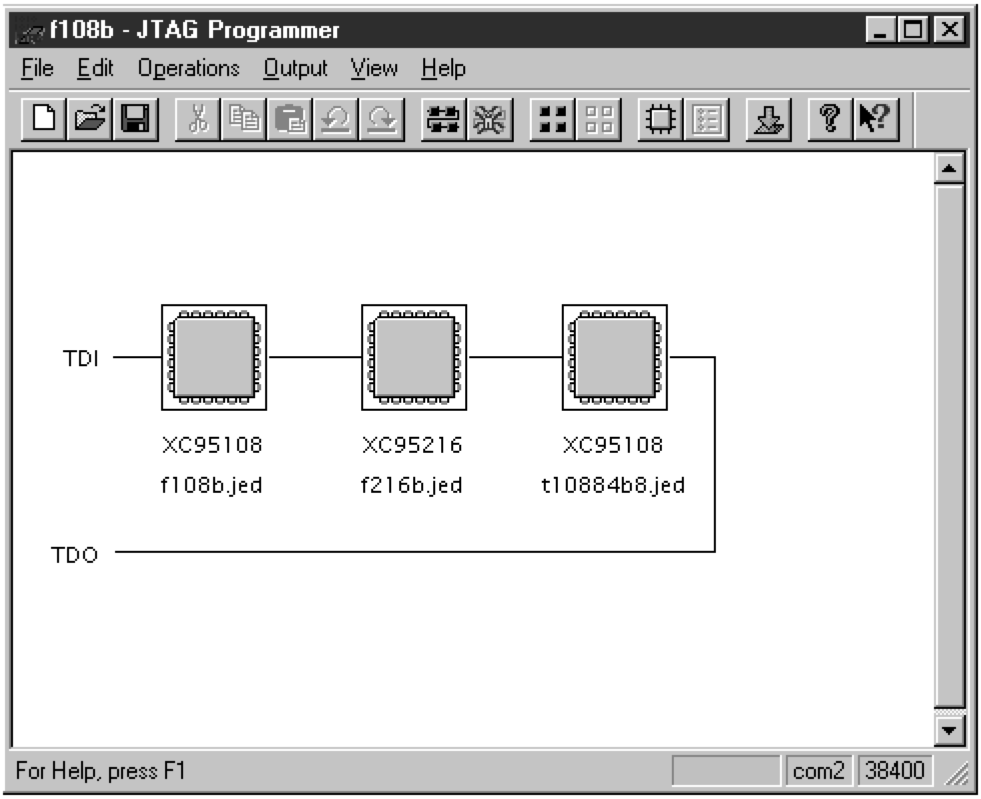


Figure 2-2 Device Chain

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

4. Put the JTAG Programmer into SVF mode by selecting

   `Output → Create SVF File...`

   to create a new 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 Xilinx XC9500/XL devices by clicking it once with the mouse. Then, select the **`Operation -> Program`**

menu item and check the **Erase Before Programming** and **Verify** options in the ensuing Program dialog box.

6. When you completed the required operations, you may close the current SVF output file by selecting the **`Output -> Use Cable`** menu item and pressing **Cancel** in the ensuing **Use Cable** dialog box.
7. Repeat steps 4 through 6 for each Xilinx XC9500/XL CPLD.
8. When you are done with JTAG Programmer, exit by selecting:

   `File->Exit`

Using the XPLA Programmer to Generate SVF Files for CoolRunner Devices

This section describes how to create an SVF file for the CoolRunner devices; it assumes that you have the XPLA Programmer, Version 4.08 or newer. XPLA Programmer software is available free of charge with the Device Programming Tools on the Xilinx World Wide Web site, http://www.xilinx.com/sxpresso/webpack.htm. Simply download the single XPLA Programmer module.

In order to program CoolRunner devices, the procedure is to create a separate SVF file for each device being programmed. We will show you how to do this using the XPLA Programmer software.

Note: GenRad does not support programming of XCR (CoolRunner) parts configuration in multiple device JTAG chains

Generating SVF Files

1. Run your design through the Xilinx CoolRunner fitter and create a JEDEC file.

2. Double-click on the “WebPACK XPLA Programmer” icon or invoke the Programmer from the start menu.

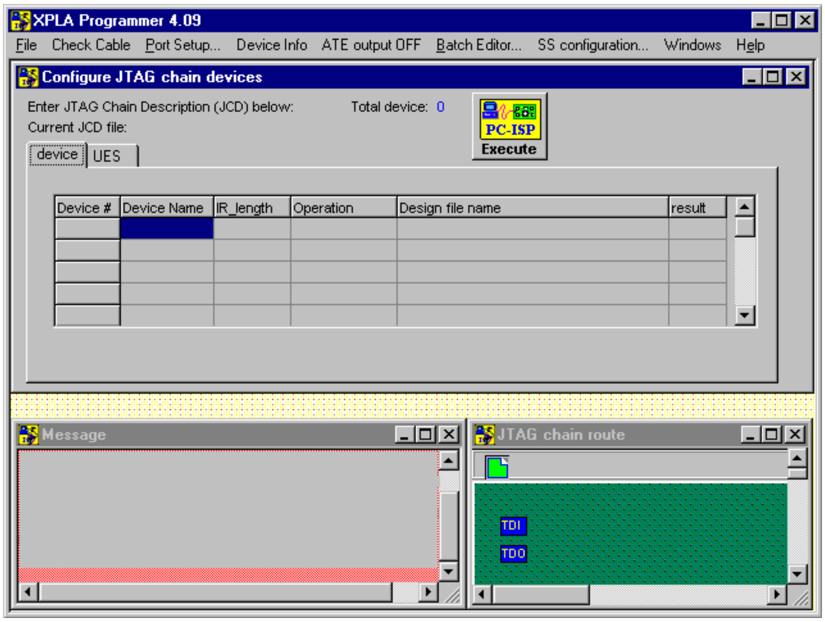
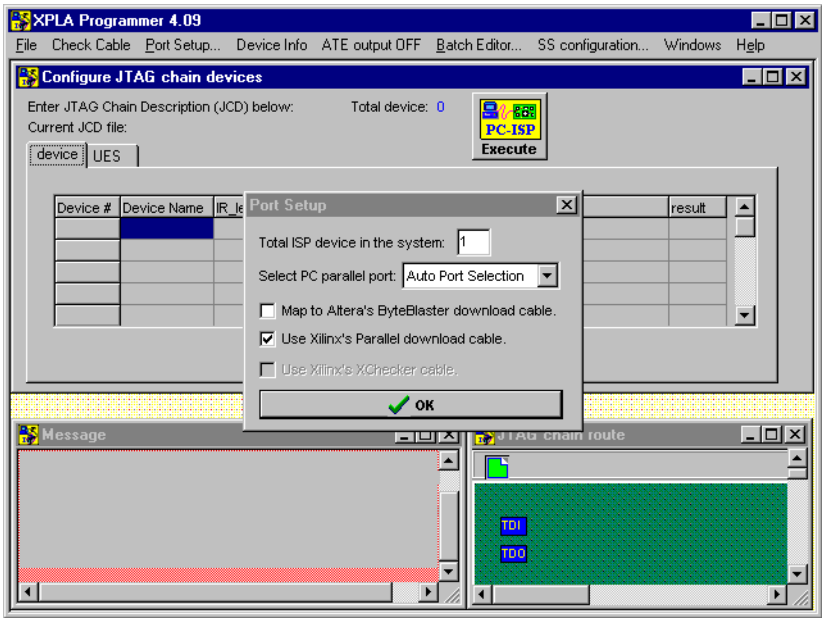

XPLA Programmer 4.09
File Check Cable Port Setup... Device Info ATE output OFF Batch Editor... SS configuration... Windows Help
Configure JTAG chain devices
Enter JTAG Chain Description (JCD) below:
Total device: 0
Current JCD file:
PC-ISP
Execute
device
UES
Device #
Device Name
IR_length
Operation
Design file name
result
Message
JTAG chain route
TDI
TDO

Figure 2-3 XPLA Programmer Interface

3. Put the XPLA Programmer into the SVF mode by selecting the following options:

   `Port Setup -> Total Device in the system`

Set this value to 1 and click **OK**.


XPLA Programmer 4.09
File Check Cable Port Setup... Device Info ATE output OFF Batch Editor... SS configuration... Windows Help
Configure JTAG chain devices
Enter JTAG Chain Description (JCD) below:
Total device: 0
Current JCD file:
PC-ISP
Execute
device
UES
Device #
Device Name
IR_le
result
Message
JTAG chain route
TDI
TDO

Port Setup

Total ISP device in the system: 1

Select PC parallel port: Auto Port Selection

☐ Map to Altera's ByteBlaster download cable.

☑ Use Xilinx's Parallel download cable.

☐ Use Xilinx's XChecker cable.

OK

4. Select from the menu bar **`ATE output OFF -> Generate svf vectors`** (.svf) (**Note**: When this option is selected the menu option will change to **ATE output ON**)

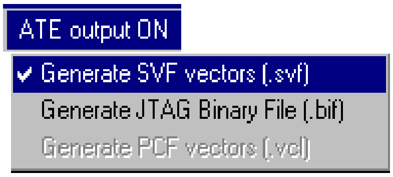

ATE output ON
✓ Generate SVF vectors (.svf)
Generate JTAG Binary File (.bif)
Generate PCF vectors (.vcl)

Figure 2-4 ATE SVF Output Generation Selection

5. Select Off-line programming as follows:

`Help -> Output/Debug Options -> Off-line programming (no I/O port access)`

then click **OK**.

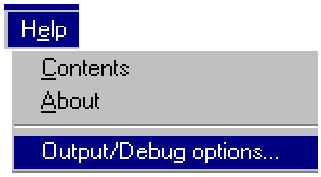
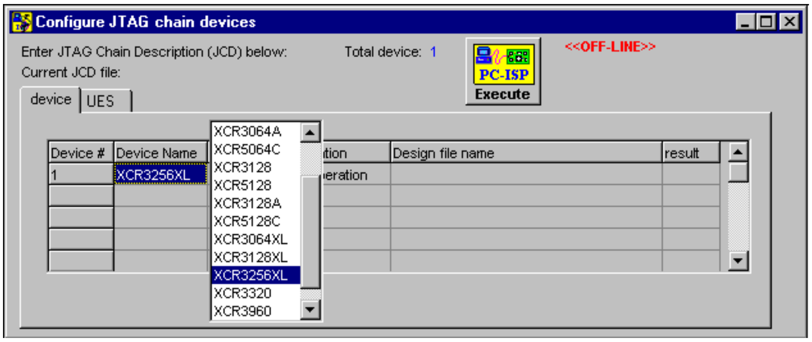


Figure 2-5 Output/Debug Option to write to a file

Note: When this option is selected a flag <<OFF-LINE>> is high-lighted in red next to the **Execute** option.

6. Once you are in SVF mode, you can then select the CoolRunner device and programming file. Double-click the **Device Name** field and select from the menu the CoolRunner device targeted.


Configure JTAG chain devices

Enter JTAG Chain Description (JCD) below: Total device: 1   <<OFF-LINE>>

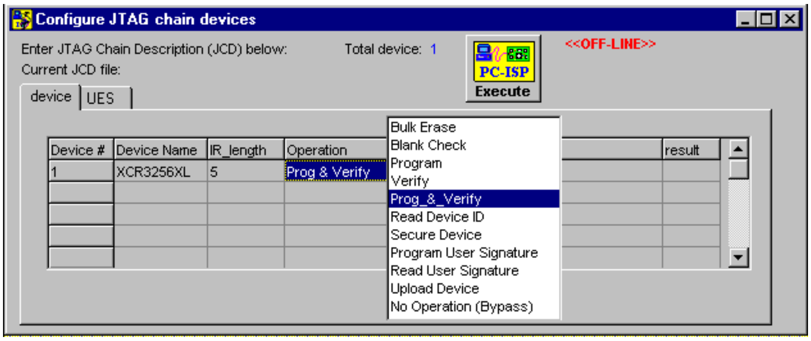
Current JCD file:

Device #	Device Name	...tion	Design file name	result
1	XCR3256XL	...eration		

XCR3064A
XCR5064C
XCR3128
XCR5128
XCR3128A
XCR5128C
XCR3064XL
XCR3128XL
XCR3256XL
XCR3320
XCR3960

Figure 2-6 Device Selection

7. Double-click the **Operation** field and select from the menu **Prog & Verify** or some other desired operation.


Configure JTAG chain devices

Enter JTAG Chain Description (JCD) below: Total device: 1

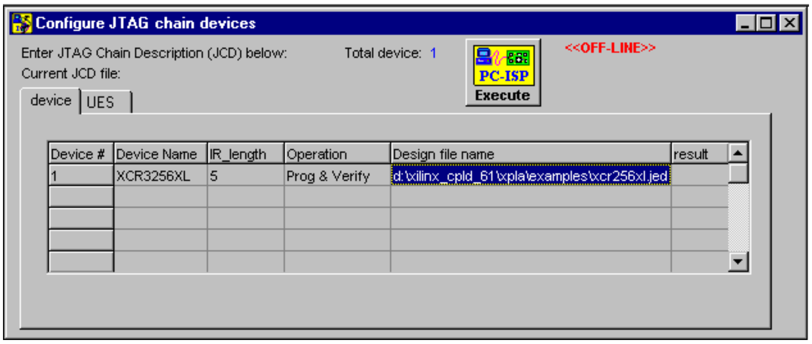
Current JCD file:

<<OFF-LINE>>

Device #	Device Name	IR_length	Operation	result
1	XCR3256XL	5	Prog & Verify	

Figure 2-7 Operation Selection

8. Double-click the **Design file name** field and browse for the programming file.


Configure JTAG chain devices

Enter JTAG Chain Description (JCD) below: Total device: 1

Current JCD file:

<<OFF-LINE>>

Device #	Device Name	IR_length	Operation	Design file name	result
1	XCR3256XL	5	Prog & Verify	d:\xilinx_cpld_61\xpla\examples\xcr256xl.jed	

Figure 2-8 Programming File Selection

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.

9. To save the SVF file:

   ```
   Select File -> Save JCD file
   ```

   and select a directory.

   (**Note**: This is the directory in which the SVF file will be saved.)

   Click the **Execute** button to write the SVF file to the location selected.

10. When you are done with the XPLA Programmer, exit by selecting:

    `File -> Exit`

Chapter 3

Creating GenRad Test Files

Using svf2dts

Use the SVF file that you generated as input to the **svf2dts** Conversion Utility. This tool takes an SVF file as input and creates a digital test source (**.dts**) file and report file (**.rpt**) which relays errors or warnings encountered during the conversion process. This chapter provides an overview of the **svf2dts** v1.8.6 user interface, file conversion, and integration program prep steps. While the program prep flow is rather straight forward, the user should be aware of different device configurations and programming techniques. The following sections also discuss different board configurations and GenRad programming options (standard driver/sensor pin memory or Deep Serial Memory) to enhance file management and minimize programming times.

This chapter is divided into the following sections:

- Review of Program Prep
- **svf2dts** Interface Overview
- Programming an Isolated JTAG Device
- Programming a Device in JTAG Chain Configuration
- Using the Deep Serial Memory Option
- Troubleshooting

Review of Program Prep Flow

Recall that converting the JEDEC design file into an SVF file with a Xilinx Programmer is only one step in the program prep process. The SVF file must be converted into GenRad vectors (**.dts** file) and integrated into the GenRad test program (**.tpg** file). The **svf2dts** tool creates a digital test source (**.dts**). GenRad's automatic test generator then creates an abbreviated **.tpg** file which contains the programming instructions and correlates the Xilinx CPLD's signal pins to the target UUT's nodes and test nails. Then the user cuts and pastes the single component **.tpg** file into the target UUT's test program file. Finally, GenRad's software translates the new **.tpg** into object code.

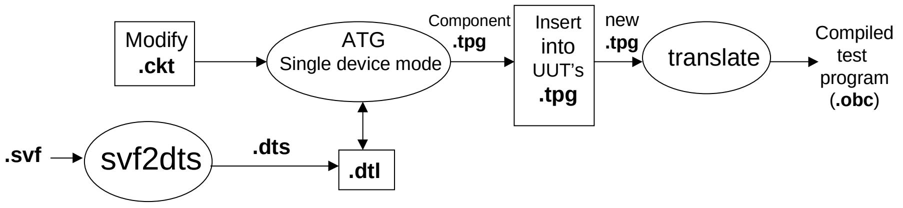
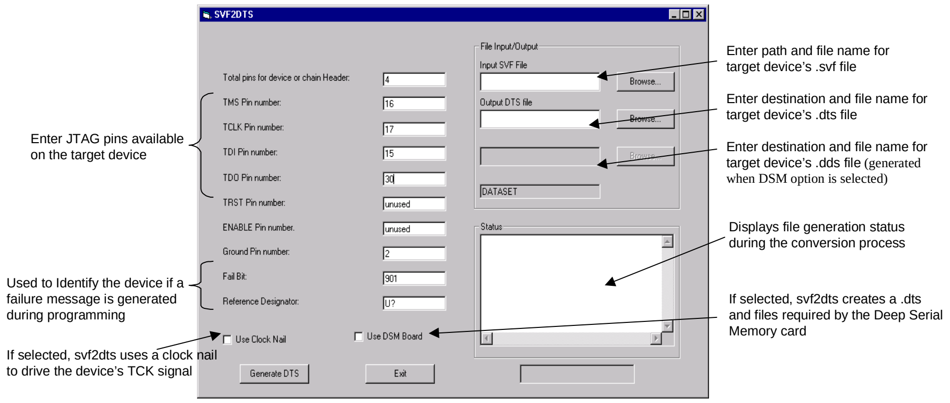


Figure 3-1 GenRad Program Prep Flow for Xilinx ISP

Svf2dts Interface Overview

The conversion tool creates a digital test source, complete with header section, and then incorporates the **.svf** file's programming vectors into commands that will instruct the test systems hardware to drive the Xilinx CPLD's JTAG pins and accomplish various supported functions. In order to successfully generate the GenRad program file(s), the conversion software must obtain information specific to the CPLD device, circuit description and GenRad programming technique. All required information is entered via the **svf2dts** interface shown below.



Total pins for device or chain Header: 4
TMS Pin number: 16
TCLK Pin number: 17
TDI Pin number: 15
TDO Pin number: 30
TRST Pin number: unused
ENABLE Pin number: unused
Ground Pin number: 2
Fail Bit: 901
Reference Designator: U?
Use Clock Nail
Use DSM Board
Input SVF File
Output DTS file
DATASET

Enter JTAG pins available on the target device

Used to Identify the device if a failure message is generated during programming

If selected, svf2dts uses a clock nail to drive the device's TCK signal

Enter path and file name for target device's .svf file

Enter destination and file name for target device's .dts file

Enter destination and file name for target device's .dds file (generated when DSM option is selected)

Displays file generation status during the conversion process

If selected, svf2dts creates a .dts and files required by the Deep Serial Memory card

Figure 3-2 SVF2DTS V1.8.6 Interface

Programming an Isolated JTAG Device

If the target PCB contains an isolated JTAG device or single device chain, all JTAG pins are free and not connected to any other components (see Figure 3-3). The JTAG pins (TDI, TDO, TMS, TCK, TRST) are likely connected to a header or the gold fingers on the PCB. If the target Xilinx CPLD's JTAG pins are connected to another JTAG-compliant device(s), please skip to the next section (*Programming a Device in a JTAG Chain Configuration*).

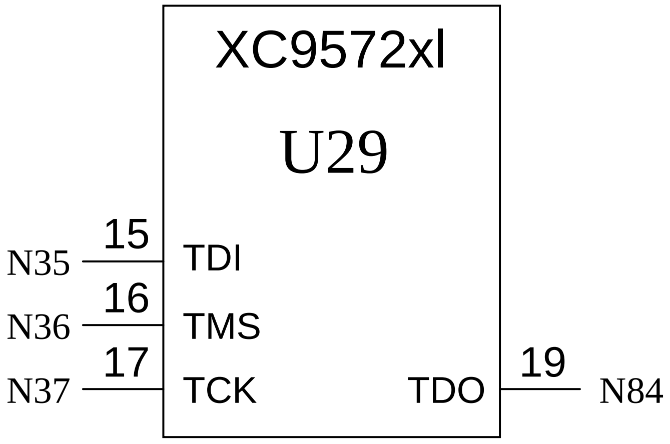


Figure 3-3 Single ISP-capable CPLD

```
U29     xc9572xl   1=N24,  2=N103, 3=N199, 4=N222,
                      :           :        :          :
                      :           :        :          :
                   13=N13, 14=N43, 15=N35, 16=N36,
                   17=N37, 18=N92, 19=N84, 20=N233,

U29_P   U29PV      1=N24,  2=N103, 3=N199, 4=N222,
                      :         :          :         :
                      :         :          :         :
                   13=N13, 14=N43, 15=N35, 16=N36,
                   17=N37, 18=N92, 19=N84, 20=N233,
```

Figure 3-4 Adding a CPLD Programming Entry in the .ckt File

Programming Example for a Single Device Chain

This section presents a step-by-step example for generating a **.dts** file using the **svf2dts** v1.8.6 for a Xilinx ISP-capable CPLD.

1. Start the **svf2dts**.
2. When the user interface appears, carefully fill in all fields in the left column with information related to the target CPLD. (It may be useful to have a schematic available to answer some of the specific device related questions).

   Total number of pins for device or chain Header

   This field refers to the number of JTAG-related pins present on the target CPLD and will dictate the argument of the SIZE keyword statement in the resultant **.dts** file. After reviewing the CPLD's pin-out in your schematic document, enter the total number of ISP-related pins. ISP-capable devices should employ at a minimum 4 JTAG-related pins (TDI, TDI, TMS, and TCK).

   TMS Pin Number

   This field refers to the CPLD's TMS signal pin. **svf2dts** will enter this number as the argument/value for TMS in the resultant **.dts** file's INPUT keyword statement located in the Header section.

   TCK Pin Number

   This field refers to the CPLD's TCK signal pin. **svf2dts** will enter this number as the argument/value for TCK in the resultant **.dts** file's INPUT keyword statement.

   TDI Pin Number

   This field refers to the CPLD's TDI signal pin. **svf2dts** will enter this number as the argument/value for TDI in the resultant **.dts** file's INPUT keyword statement located in the Header section.

   TDO Pin Number

   This field refers to the CPLD's TDO signal pin. **svf2dts** will enter this number as the argument/value for TDO in the resultant **.dts** file's OUTPUT keyword statement located in the Header section.

TRST Pin Number

XC9500/XL and CoolRunner devices do not have a TRST JTAG pin. If you are using a device from either of these Xilinx families, leave this space blank when queried by the **svf2dts** program

ENABLE Pin Number

XC9500(XL) and CoolRunner devices do not have an ENABLE JTAG pin. If you are using a device from one of these Xilinx families, leave this space blank when queried by the **svf2dts** program. If another device in the JTAG chain employs a ENABLE pin, enter the pin designator.

If you used an alpha-numeric for the signal names above, you must edit the DTS file to add or change all other pins on the device to be alpha-numeric. This will maintain consistency across library modules.

Ground Pin Number

Enter on the pin number for one of the CPLD's ground pins.

Fail Bit

This field refers to the fail bit to be set in the event that a programming function fails. The default is fail bit 901. Any number over 900 is acceptable.

Reference Designator

Enter the reference designator shown in the schematic to identify the target CPLD. In the event that a programming operation fails, the resultant test program will generate an error message which indicts the name denoted in this field (i.e., U29_P).

Designating input files and output files. In the upper right of the interface, fill in the fields beneath *File Input/Output*

Input SVF File

Provides the conversion tool with the path and name of the target CPLDs SVF file. Enter the path and name of the target SVF file (e.g., c:\temp\design1.svf).

Output DTS File

Provides the conversion tool with the target location and name for the resultant **.dts** file. Enter the path and name of the new **.dts** file (e.g., c:\temp\design1.dts). This feature allows you to use a different name for the **.dts** and **.rpt** files than you had on the **.svf** files. This allows easier integration into the ATG process.

3. Select clock driver nail to drive the CPLD’s TCK signal by clicking the option box next to *Use Clock Nail* located in the lower left of the interface. Using a Clock Driver Nail gives a faster slew rate and, therefore, better noise immunity. Noise can be a significant problem for some applications when programming parts in-system. GenRad recommends selecting this option if one of the test systems clock driver nails is available.
4. Selecting the *Use DSM Board* Option (**XC9500/XL only**). If you have a Deep Serial Memory card, GenRad recommends selecting this option which will help minimize programming time and data vector file sizes. However, since a few more program prep steps are required, please proceed to the section entitled Programming Xilinx CPLDs with Deep Serial Memory. If you're not using a DSM continue to step 6.
5. Once all of the option selections and fields have been completed, press *Generate DTS*. The resulting **.dts** file may contain multiple bursts and require a lot of disk space. A resulting **.rpt** file contains a summary of the data used to generate the **.dts** and the number of bursts generated. Actual execution time for file conversion varies depending on the size of the **.svf** file and the speed of your computer.
6. Insert the new **.dts** file into a user **.dtl**. If a user **.dtl** does not exist create one with GenRad’s library tool interface.
7. Run GenRad's Automatic Test Generator in single component mode for the new **.dts** (in the previous example the single component is U29_P). The software creates a file named U29_P.tpg. Copy the file's content and paste in the UUT's **.tpg** file beneath the section of code for U29_P in the digital test section.
8. Translate the target **.tpg** file.
9. Debug the test program

Programming a Device in a JTAG Chain Configuration

Note: The CoolRunner family is supported in single device chains only.

When the target XC9500/XL CPLD lies in a multiple device or JTAG chain configuration, the user must make a couple additional modifications to the **.ckt** file and generate a **.dts** for each Xilinx ISP-capable CPLD. In a multiple-device chain, all JTAG-compliant devices in the chain share TCK and TMS signals. However, the first device's TDO signal is connected to the TDI of the second device and so forth (see figure below). From a program prep perspective, each Xilinx ISP-capable CPLD has an **.svf** file which contains vectors to erase, program, and/or verify the device.

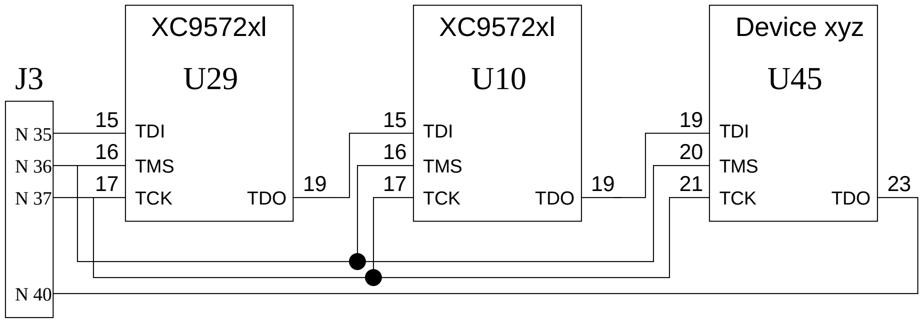


Figure 3-5 Multiple JTAG Device Chain Configuration

```
U29_P   U29PV     :          :          :         :
                13=N13, 14=N43, 15=N35, 16=N36,
                17=N37, 18=N92, 19=N40, 20=N233,

U10_P   U10PV     :          :          :         :
                13=N13, 14=N43, 15=N35, 16=N36,
                17=N37, 18=N92, 19=N40, 20=N233,

U45_P   U45PV     :          :          :         :
                17=N37, 18=N92, 19=N35, 20=N36,
                21=N37, 22=N55, 23=N40, 24=N422
```

Figure 3-6 Modifying the .ckt to Program CPLDs in the Same JTAG Chain

The above example illustrates a three device chain consisting of U29, U5 and U40. The three entries have the same node numbers since the chain must always be driven from the beginning and there are three devices to program and verify.

Programming Example for a Multiple Device Chain

Note: The CoolRunner family is supported in single device chains only.

Remember, for each Xilinx XC9500/XL CPLD in the JTAG chain that you want to program, the **svf2dts** must generate a separate **.dts** file. In the example, there are three devices in the JTAG chain. Two are Xilinx XC9572XL devices (which can be programmed) and the third is an arbitrary JTAG-compliant device (e.g., a DSP). In order to program the XC9572XL parts, two **.svf** files (at minimum) should be available. One **.svf** contains instructions to put U10 and U45 into bypass mode and program U29. The other contains instructions to put U29 and U45 into bypass mode and program U10. This section walks through a step-by-step example for generating a **.dts** for U29. Use the same steps to create a **.dts** file for U10.

1. Start the **svf2dts** tool.
2. When the user interface appears, carefully fill in all of the required fields with information related to the target XC9500/XL CPLD. (It may be useful to have a schematic available to answer some of the specific device related questions).

   Total number of pins for device or chain Header

   This field refers to the JTAG-related pins employed on the target CPLD. In the example, there are 4 such pins (TDI, TDO, TMS, and TCK)

   TMS Pin Number

   This field refers to the JTAG chain's TMS signal pin. In the multiple-device chain example, TMS for all devices should be referred to entered as 16.

   TCK Pin Number

   This field refers to the JTAG chain's TCK signal pin. In the example, TCK for all devices should be referred to entered as 17.

   TDI Pin Number

   This field refers to the JTAG chain's TDI signal pin. **svf2dts** will enter this number as the argument/value for TDI in the resultant **.dts** files INPUT keyword statement located in the Header section.

TDO Pin Number

This field refers to the JTAG chain's TDO signal pin. Since the last device in the chain is U40, it provides the TDO reference for the chain. Since the **.ckt** file has been modified to note that U29_P's pin 19 connects to N40 (the node connected to U40's TDO signal), simply enter 19 in this field.

TRST Pin Number

XC9500(XL) devices do not have a TRST JTAG pin. If you are using XC9500(XL) devices, leave this space blank when queried by the **svf2dts** program. If another device in the JTAG chain employs a TRST pin, enter the pin designator.

ENABLE Pin Number

XC9500(XL) devices do not have an ENABLE JTAG pin. If you are using XC9500(XL) devices, leave this space blank when queried by the **svf2dts** program. If another device in the JTAG chain employs a ENABLE pin, enter the pin designator.

If you used an alpha-numeric for the signal names above, you must edit the DTS file to add or change all other pins on the device to be alpha-numeric. This will maintain consistency across library modules.

Ground Pin Number

Enter on the pin number for one of the XC9500/XL CPLD's ground pins.

Fail Bit

This field refers to the fail bit to be set in the event that a programming function fails. The default is fail bit 901 and could be used for the first device in the chain (any number over 900 is acceptable). However, subsequent devices require a different fail bit to differentiate one from the other when a failure is encountered. GenRad recommends incrementing the failure bit by one for each additional device (i.e., in the example, use 902 for U10).

Reference Designator

This field denotes reference designator for the target XC9500/XL XC9500/XL Series CPLD. In the event that a programming operation fails, the resultant test program will generate an error message which indicts the name denoted in this field (i.e., U29_P). This assists in properly identifying a failed device.

3) Designate input files and output files. In the upper right of the interface, fill in the fields beneath *File Input/Output*

Input SVF File

Provides the conversion tool with the path and name of the target XC9500/XL CPLD's SVF file. Enter the path and name of the target SVF file (e.g., c:\temp\design1.svf).

Output DTS File

Provides the conversion tool with the target location and name for the resultant **.dts** file. Enter the path and name of the new **.dts** file (e.g., c:\temp\design1.dts). This feature allows you to use a different name for the **.dts** and **.rpt** files than you had on the **.svf** files. This allows easier integration into the ATG process. The DTS file will be in the kilobyte range.

3. Select clock driver nail to drive the XC9500/XL CPLD's TCK signal by clicking the option box next to *Use Clock Nail* located in the lower left of the interface. Using a Clock Driver Nail gives a faster slew rate and, therefore, better noise immunity. Noise can be a significant problem for some applications when programming parts in- system. GenRad recommends selecting this option if one of the test systems clock driver nails is available.
4. Selecting the *Use DSM Board* Option. If you have a Deep Serial Memory card, GenRad recommends selecting this option which will help minimize programming time and data vector file sizes. However, since a few more program steps are required please proceed to the section entitled Programming Xilinx XC9500/XL CPLDs with Deep Serial Memory.
5. Once all of the option selections and fields have been completed, press *Generate DTS*. The resulting **.dts** file will contain multiple bursts and may require a lot of disk space. There will also be a **.rpt** file that contains a summary of the data used to generate the **.dts** and the number of bursts generated. Actual execution time for the file conversion varies depending on the size of the **.svf** file and the speed of your computer.
6. Repeat steps 1-5 for each device and/or **.svf** file.

7. Insert the new **.dts** file into a user **.dtl**. If a user **.dtl** does not exist create one with GenRad’s library tool interface.
8. Run GenRad's Automatic Test Generator in single component mode for the new **.dts** (in the previous example the single component is U29_P). The software creates a file named U29_P.tpg. Copy the file's content and paste in the UUT's **.tpg** file beneath the section of code for U29_P in the digital test section.
9. Translate the target **.tpg** file.
10. Debug the test program
11. Repeat steps 6-10 for each new **.dts** file.

Using the Deep Serial Memory Option

Note: The Deep Serial Memory Option is supported for XC9500/XL devices only. CoolRunner devices do not support this option.

If you have a Deep Serial Memory (DSM) board on your tester you should use it. Using Deep Serial Memory will result in significantly smaller file sizes. The DDS file will be in the 1 to 4 megabyte range. The DTS file will be in kilobyte range.

1. Enter the appropriate information into all **svf2dts** interface fields.
2. Click the option box next to *Use DSM Board* located in the lower left of the interface. Notice that a new field is now present within the File Input/Output section.
3. Enter a filename and destination path for the resultant **.dds** file required for the DSM option.
4. Press *Generate DTS.*
5. Insert the new **.dts** component into **.dtl**
6. Deep Serial Memory provisions
   a) Create an automatic test options file (**.ato**). This file will load the DSM binary file for the ISP component
   b) Run DSM Translate on the DSM source (**.dds**) file to create the **.ddb** file. Use the DSM Translate Page in GenRad’ ISP

Tool or 228X monitor pages DSM Translate page to create the **.ddb** file.

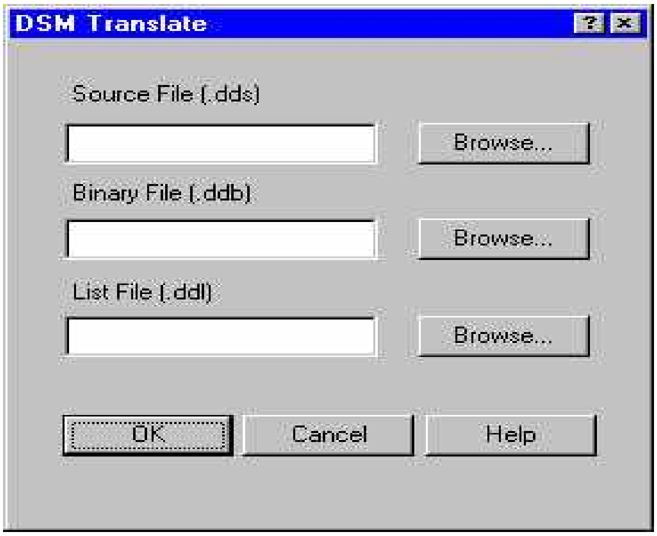


Figure 3-7 DSM Translate Interface Available in GenRad's ISP Tool V1.0

c) Run GenRad's ATG with the newly created **.ato** file

7. Translate/compile test program (**.tpg**)
8. Debug test program

Troubleshooting

- There MUST be accurate BSDL files for all devices in the chain.
- Unique FAIL bits MUST be assigned to each Xilinx device being tested.
- Branch statements may need to be added at label DDONE: to include the FAIL bits used in testing the devices.
- For erase vectors on XC9500 devices, you may need to increase the erase "pulse" times. In fact, this is **highly recommended** for a robust program. To increase the erase "pulse" times, you must manually edit the DTS or SVF file. It is easiest to edit the SVF file before running it through the **svf2dts** translator. In the SVF, replace every instance of "RUNTEST 1300000 TCK;" with

"RUNTEST 2600000 TCK;". This will double the erase "pulse" time from 1.3 seconds to 2.6 seconds per erase "pulse". There are 2 to 32 erase "pulses" per erase operation depending on the device. This is not usually required for XC9500XL devices.

Appendix A

DTS Example and Explanation

```
/* U5PV.DTS created by SVF2DTS version 1.8.6 utility 04-2-2000 09:30:33 */
.HEAD;
.SIZE 4;
.INPUT(1=TMS,2=TCLK,3=TDI);
.OUTPUT(4=TDO);
.PERIOD 1U; /* TCK is set to 1 MHz by the conversion tool
.END HEAD;
```

Any routine used more than once is put in a subroutine to decrease the number of bursts required.

```
.FASTSUB TCLK;
$ IC(TCLK) IH(TCLK);
$ IL(TCLK);
$ RETURN;
.END FASTSUB;
```

Additional subroutines deleted for readability.

```
.MAIN USING(F=TDO);
BURST ACTIVE NOPRINT NOFAULT NODIAG FAIL() MAXTIME=60;
FAST;
```

Test steps deleted for readability.

The output .DTS is fully commented and includes any comments that were in the input .SVF file. Test steps that check data set the assigned FAIL bit if they fail.

```
/* Shift in FF masked with FF */
/* Shift out 01 masked with FF */
OS(TDO) OH(TDO) ~FAIL(196)~;
$ IH(TDI);
$ GOSUB TCLK;
OL(TDO) ~FAIL(196)~;
```

Test steps deleted for readability.

```
/* Xilinx programming loop */
/* Shift in 00BFFFFE masked with 07FFFFFF */
/* Shift out 00000003 masked with 00000003 */
```

The Xilinx programming algorithm requires the location be tried up to 32 times if it fails to program. To accomplish this we loop on the programming command 31 times. The two status bits are tested each pass through the loop and set the local FLAGFAIL Status bit.

```
FLOOP = D'31';
OS(TDO) OH(TDO) FLAGFAIL;
$ IL(TDI);
$ GOSUB TCLK;
OH(TDO) FLAGFAIL;
$ IH(TDI);
$ GOSUB TCLK;
OI(TDO);
```

Test steps deleted for readability.

```
$ GOSUB TCLK;
/* State is now DREXIT1 */
```

If FLAGFAIL is not set then both status checks pass. Branch around retry and continue programming. If either status check fails, attempt to program the location up to 31 times.

```
GOTO P1 FLAG PASSES;

/* Operation failure retry */

$ GOSUB RETRY;

$ GOSUB W1800000;

/* Set state to DRSHIFT from IDLE */

$ GOSUB IDL2DRS;

IL(TMS);

/* State is now DRSHIFT */

END FLOOP;
```

If after 31 attempts the location still fails, try one more time. This time if it fails set the status FAIL bits 193 or 194.

```
OS(TDO) OH(TDO) FLAGFAIL ~FAIL(193)~;

$ IL(TDI);

$ GOSUB TCLK;

OH(TDO) FLAGFAIL ~FAIL(194)~;

$ IH(TDI);
```

Test steps deleted for readability.

```
/* Branch to burst end if fails after 32 attempts */
```

Check the fail flag and branch to the end of the burst if the location failed.

```
GOTO BE1 FLAG FAILS;

P1:
```

Test steps deleted for readability.

```
BE1: $ ;

END FAST;
```

The END BURST statement clears FAIL bits 193 and 194. Then it checks the FAIL bit assigned to the device. If the FAIL bit is set, a failure message is printed and any remaining bursts are branched. DTG will add the correct branch statement before the "]".

```
END BURST[IF FAIL(193) THEN BITCLR(FAIL,193);
IF FAIL(194) THEN BITCLR(FAIL,194);
IF FAIL(196) THEN WRITE ID=MESFILE 'Device U5 failed%NL%';
IF FAIL(196) THEN ];
```

Additional bursts deleted.

The last burst is done only to clear the active state.

```
BURST;
/* This burst is done only to clear the active state!! */
IC(TDI,TMS,TCLK);
ID(#);
END BURST;
.END MAIN;
```

Appendix B

Optimizations

You can use several simple techniques to optimize and reduce the overall programming time.

- If you have Deep Serial Memory, use the Deep Serial Memory option. Deep Serial Memory more efficiently handles the large ISP vector sets (not applicable to CoolRunner devices).
- If you always program factory fresh (i.e. blank) XC9500 devices, eliminate the erase operation by unchecking the Erase Before Programming option in the Program dialog box when generating the SVF file. The erase operation consumes the majority of the overall programming time for XC9500 devices.

  The erasure time for CoolRunner devices is only 100 ms; the removal of the Erase procedure will not significantly improve the programming times for CoolRunner CPLDs.

Additionally, several advanced techniques may be used to optimize and reduce the overall programming time:

- If most of the devices you program are blank (erased), you can conditionally erase an XC9500 device. You must generate separate SVF files that contain the following operations per SVF file: blank check, erase, and program+verify. To create the blank check SVF, you must obtain the **.jed** file for a blank (erased) device. Blank **.jed** files can be obtained via ftp://ftp.xilinx.com/pub/swhelp/cpld/blank.zip. Use JTAG Programmer to generate an SVF that contains a verify operation using the blank **.jed** file. In your GenRad test program, you must first execute the blank verify operation. Then, based on the result, you can conditionally execute the erase vectors from the SVF with the erase operation. Finally, execute the program+verify vectors on the device. The verify (blank check) operation executes very quickly in comparison to the regular, sector-based erase operation.

- You can use the faster *bulk erase* method for later revisions of the XC9500 devices. By default, JTAG Programmer uses the sector-based erase operation for XC9500 devices because the sector-based erase is supported in all revisions of the XC9500 devices.
    - All XC9500 devices with a 0000 bit value in the revision field (4 most-significant-bits) of the IDCODE only support the sector-based erase. The later revisions of the XC9500 devices support a faster *bulk erase* operation.
    - To force JTAG Programmer to use the bulk erase operation, you must temporarily replace the base BSDL file in the $XILINX/xc9500/data/ directory with the higher revision BSDL file while you create the SVF files. For example, you can replace the xc9536.bsd file with the xc9536_v2.bsd file. Now, you can generate an SVF that contains the *bulk erase* operation. (Remember to restore the original BSDL file so that JTAG Programmer will work correctly with all revisions of the XC9500 devices.)
    - You may want to implement your GenRad test program such that it conditionally performs either the sector-based erase or the bulk erase operation based on the device IDCODE. To create vectors that check a device IDCODE, create an SVF that contains the Get IDCODE operation. By default, JTAG Programmer generates the IDCODE check (SDR command) in the SVF with a MASK value that ignores the revision field (4 most-significant-bits) of the IDCODE. Manually edit the SVF to check all bits of the 32-bit IDCODE. Make sure the TDO is expecting all zeroes for the revision field of the IDCODE. Then, you can use the IDCODE SVF to check if a device is a revision 0000 device. If it is a revision 0000 device, you perform the regular sector-based erase. Otherwise, you can perform the *bulk erase.*

Appendix C

Using the Batch Download Tool to Generate SVF Files Using JTAG Programmer

1. Run your design through the Xilinx fitter and create a JEDEC programming file. If you have already been provided with a JEDEC file, proceed to the next step.
2. Set the Xilinx environment variable to the Xilinx installation directory:

   SET XILINX=C:\XILINX_CPLD
3. 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> Copy-
   right:1991-2000

   Sizing system available memory...done.

   ***SVF GENERATION MODE***

   [JTAGProgrammer::(1)]>
   ```
4. 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.

5. Enter the following command to create SVF files for each Xilinx XC9500/XL CPLD:

```
program -h -v designName -j jedecName.jed
```

You may enter any of the following alternate commands:

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 [**-vbh**] *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 **–v** option specifies that a verify operation should be performed immediately after the program operation has completed. The **-b** flag indicates the erase operation should be skipped. The erase operation is performed, by default, prior to the program operation. This is useful when programming devices shipped from the factory. Xilinx recommends **program –h -v** *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.

6. Exit JTAG Programmer by entering the following command:

   ```
   quit
   ```

   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.