Programming Xilinx XC9500 on a Teradyne Z1800 or Spectrum

JTAG Programmer

Version 2.1i

June 1999

June 1999

Preface

About This Manual

This manual describes how to program Xilinx XC9500 CPLDs on Teradyne Z1800 or Spectrum 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 a Teradyne test environment.
- Chapter 2, "Creating SVF Files," discusses how to create SVF files using JTAG Programmer on PCs, and on Sun and HP workstations.
- Chapter 3, "Creating Teradyne Test Files," discusses how to generate a Teradyne Binary Vector File, and how to create the executable **`ptprog.exe`**, and integrate into your test environment.
- Appendix A, "Troubleshooting," contains information on troubleshooting a problem.

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/XL/XV CPLD designs in a Teradyne test environment. The procedures described in this document lay out the necessary steps you need to perform; they are:

- Creating an SVF File Using JTAG Programmer
- Generating a Teradyne Binary Vector File
- Creating the **`ptprog.exe`** executable
- Modifying the **`PT2.INI`** File
- Integrate the executable and **`.img`** file into your test program

JTAG Programmer translates JEDEC files into Serial Vector Format (SVF) files. The **`xilinx`** bat file translates SVF files to Teradyne Binary Vector Files.

The following is needed to run this software:

- Digital Function Processor option
- Windows95 or WindowsNT, a 32 bit environment
- Turbo C++ or Microsoft Visual C++ Compiler (the Microsoft Visual C++ must be able to produce DOS executables to produce the 16 bit executable **`ptprog.exe`**)
- Teradyne F1 Software

Note: The installation procedure is found in the README file on the disk. This is an ASCII text file and can be viewed from the DOS editor (**`edit readme.txt`**) or the Windows Notepad.

Chapter 2 describes the procedure for creating an SVF file from JTAG Programmer from both PC and workstation environments. Chapter 3 describes how to produce Teradyne compatible tester programming.

Hardware Considerations

This software and methodology applies to the following Teradyne testers running Teradyne software release F1 or greater.

- Z1800 with Digital Functional Processor
- Spectrum

Xilinx Device Support

- XC9500 Family (5 Volt devices)
- XC9500XL Family (3.3 Volt devices)
- XC9500XV Family (2.5 Volt devices)

XC9500	XC9500XL	XC9500XV
36	36XL	36XV
72	72XL	72XV
108		
144	144XL	144XV
216		
288	288XL	288XV

Table 1-1 Device Listing of Device Families

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/isp/toolbox.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 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.

Using the Batch Download Tool to Generate SVF Files

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. 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-XC9500XL / XV 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`**] **`-id`** *designName* **-j** [*jedecFileName*]-- 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* **-j** [*jedecFileName*]-- 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.

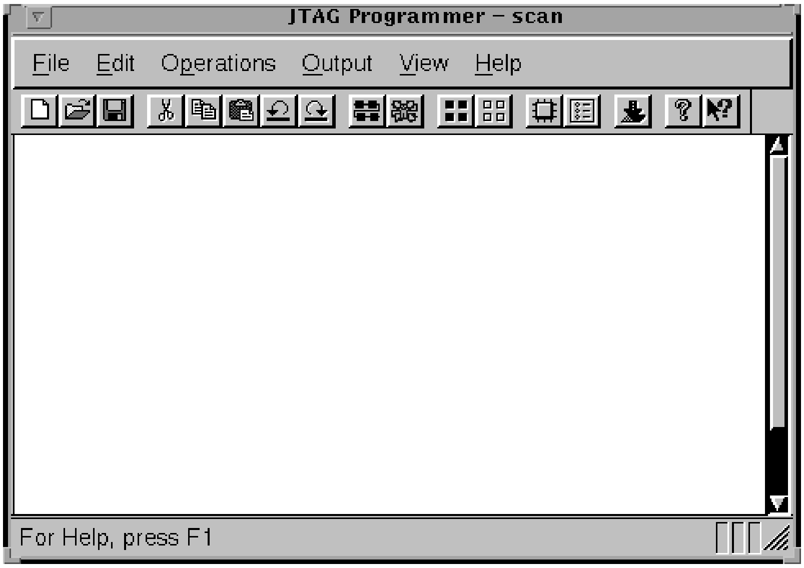
5. Exit JTAG Programmer by entering the following command:

```
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.

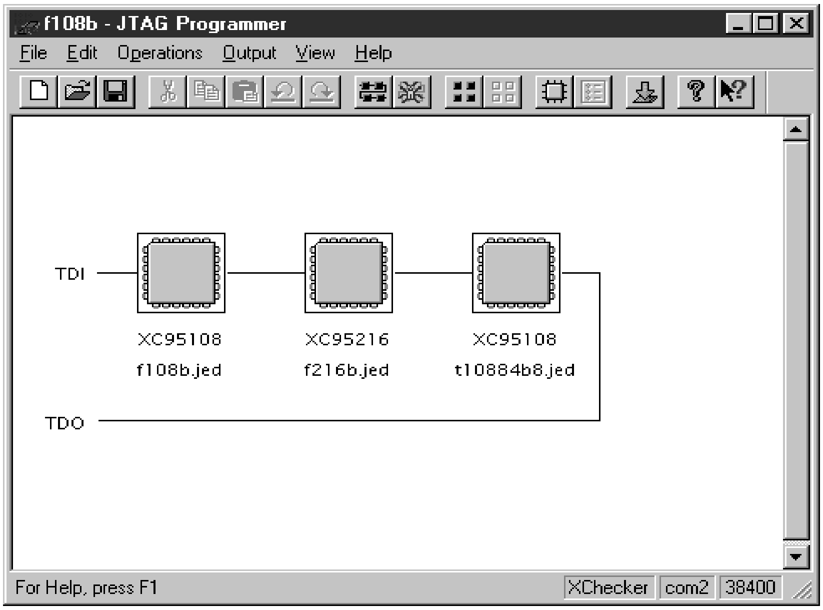
Using the Graphical User Interface to Generate SVF Files

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



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.



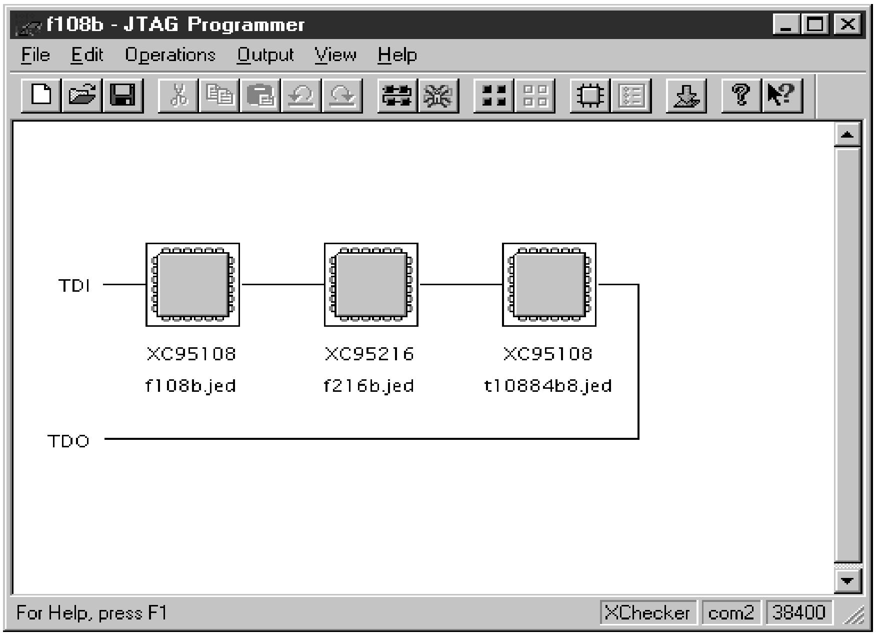
The device type and JEDEC file name will appear below the added device.

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

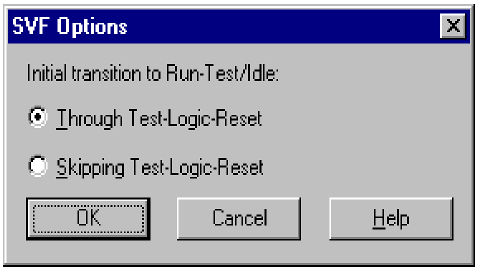
   `Output` → `Create SVF File...`

   to create a new SVF file, or

   `Output` → `Append to SVF File...`

   to append to an existing SVF file.

   Select the SVF Option **`Through Test-Logic-Reset`**, then click **`OK`**.


SVF Options

Initial transition to Run-Test/Idle:

[✓ Through Test-Logic-Reset]

Skipping Test-Logic-Reset

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 Teradyne Test Files

Introduction

Use the SVF file that you generated above as input for the creation of a Teradyne Binary Vector File (**`.img`**). This chapter explains how to create executable Teradyne programs from SVF files.

You first need to check that your PC is setup with the following files that were contained in the zip file (setup file). If you do not find these files, your unzip utility probably failed.

Table 3-1 Directories and Files

Directory	Files
dfp	pt2.ini xilinx.c
doc	teradyne.pdf readme.txt
xlate	svfp.exe xilinx.bat bsv.dll dfpbv.dll failrep.dll svfc.dll svfp.dll util.dll wrpsrv.dll

The **`xlate`** directory contains the executable and libraries you will need to parse the SVF file output of your design tool. Add the **`xlate`** directory to the PATH variable on your PC.

Generating a Binary Vector File

To create a binary vector file, run the translator by entering the **`xilinx`** command.

```
xilinx filename.svf
```

Note: **`xilinx`** is a DOS **`.bat`** file that will invoke the **`svfp`** program with the following parameters:

`-period 1000 -svf` *filename*.**`svf -target dfpbv`**.

The **`-period`** parameter is used by the **`svfp`** program to determine the real time clock value used. The **`-svf`** parameter refers to the **`.svf`** file that contains your programming information, and the **`-target`** parameter indicates that the binary vector file will be output such that it can be run on the Digital Functional Processor.

In general, the file size for the **`.img`** file produced by the **`svfp`** program will be smaller than the original **`.svf`** file.

After running **`xilinx.bat`** copy the resulting **`.img`** file to the directory containing your data files (on the DFP).

Creating the Executable ptprog.exe

Source code for the Digital Functional Processor is located in the dfp directory. Compile **`xilinx.c`** into an executable called **`ptprog.exe`**.

Note: Teradyne currently supports Turbo C++ Version 3.0 and Microsoft Visual C++. You must have a version of Visual C++ that produces DOS 16 bit executables.

`xilinx.c` has the channel assignments for the JTAG connections hardcoded. If your design has multiple devices in the JTAG scan chain, you will need to hook the TDI of the first device to channel 209 and TDO of the last device in the chain to channel 208. You should have also generated your SVF file with the same chain configuration. The table below shows the mapping that the **`xilinx.c`** expects. The TRST is not needed for Xilinx parts.

Table 3-2 Mapping

Signal	DFP Card	DFP node/s
TDO	Card 0	208 (Port C bit 0)
TDI	Card 0	209 (Port C bit 1)
TCLK	Card 0	210 (Port C bit 2)
TMS	Card 0	211 (Port C bit 3)
TRST	Card 0	212 (Port C bit 4)

TCLK is running at the fastest rate the Z1800 DFP will allow (600 kHz).

The **`ptprog.exe`** when executed will look for two input files: the **`.img`** binary vector file, and the **`pt2.ini`** file (described below).

Modifying the pt2.ini File

The **`pt2.ini`** file provides additional information to the **`ptprog.exe`** program at run time. A sample **`pt2.ini`** file is shown below.

Sample pt2.ini - XC95108 :

L,IC1,XC95108,pgm.svf,,,0,1

R,format = No translation

where:

`L`	= local device tag	
`IC1`	= board identifier	Modify the board identifier to your own board identifier.
`XC95108`	= device type	Make this be the device type for the device that is being programmed in the SVF file.
`pgm.svf`	= data source file	Make this the name of the original SVF file. This will generate a **`.img`** file with the same root name as your **`.svf`** file (**`pgm`** in this example)

Blank	= format of data source file (91 = Jedec fuse file)	Not needed for Xilinx devices.
Blank	= number of fuses	Not needed for Xilinx devices.
0	= chain position	Not needed for Xilinx devices.
1	= fill character	Not needed for Xilinx devices.
R	= remarks/comments	

Adding Xilinx Programming to your Z1800 Board Test Program

1. Edit the PRGMVARS section of your program. Enable the DFP and specify the Source File Directory path, usually the board directory.
2. Create a subdirectory of the source file directory and copy the **`*.img`**, **`pt2.ini`** and **`ptprog.exe`** files to it.
3. Create a DFP worksheet and specify the subdirectory name in the source directory field of the worksheet. No arguments are needed.
4. Press the update button and you are ready to program.

Appendix A

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.

If you are not able to program the Xilinx parts reliably, try turning down the TCK rate. You can do this by modifying the following **`xilinx.c`** statements as follows:

Uncomment `#define CCC_IO_MAP`

Comment out `#define CCC_MEM_MAP`

This will reduce the TCK clock frequency to about 300 kHz.