

Summary

This application note describes how to program XC9500 devices in-system, using standard automatic test equipment.

Xilinx Family

XC9500

Introduction

XC9500 devices use a standard 4-wire Test Access Port (TAP) for both In-System Programming (ISP) and IEEE 1149.1 boundary scan (JTAG) testing. Therefore, manufacturers can reduce their overall system cost and reduce device damage due to unnecessary handling by using automatic test equipment (ATE) to both program and test these devices. The XC9500 Boundary-scan architecture is shown in Figure 1.

The Xilinx EZTag™ software helps make this possible by automatically generating a Serial Vector Format (SVF) file describing the programming and test algorithms required by the XC9500 devices. Most ATE platforms accept SVF as a test vector input format. This application note describes the steps required to generate an SVF file and how the ATE uses this file to program and test a device.

SVF Overview

The original Serial Vector Format was developed jointly by Texas Instruments and Teradyne in response to a need for the exchange of boundary-scan test vectors between such tools as test generation software and ATE. At that time, usage of the IEEE standard 1149.1 was increasing but no common format or language existed to satisfy the need for a common data exchange.

The developers of SVF chose a format that did not use test vectors solely to provide TCK (clock) and TMS (mode control) signals to the IEEE 1149.1 TAP. Instead, the underlying models of the SVF format assume that all operations begin and end in stable states. This results in a much simpler and more concise description of the stimulus vectors.

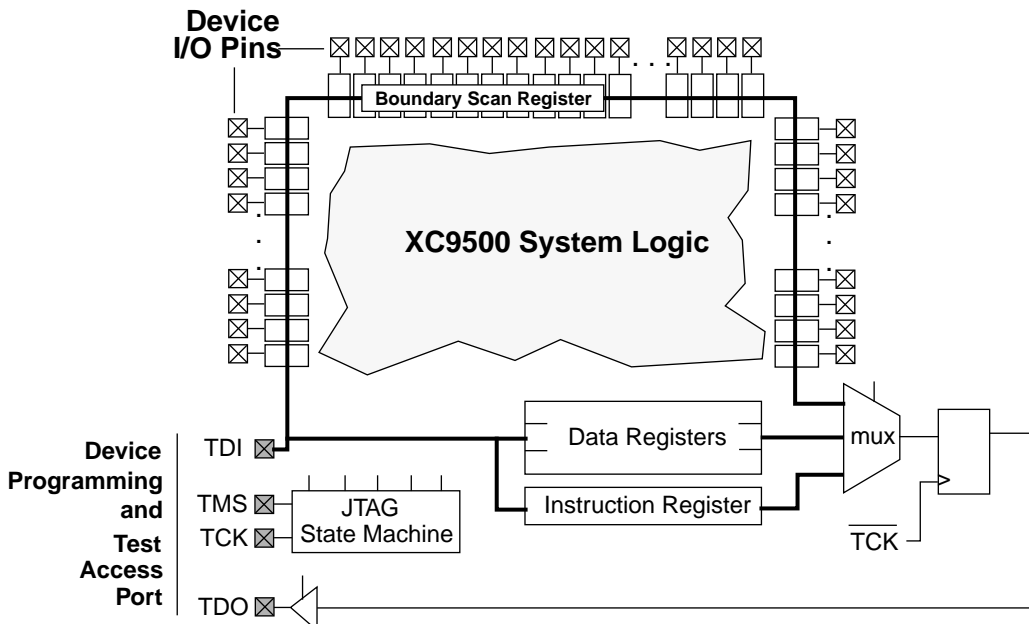


Figure 1: XC9500 Boundary Scan Architecture

Between mid-1991 and the autumn of 1994 three revisions of SVF were developed, with the goal of creating a format that was independent of the test application vehicle. By late 1994 over 100 companies had developed SVF-based tools and at least ten vendors of CAE tools and ATE were supporting SVF.

SVF has proven itself to be a useful and reliable format for exchanging data between ATE and the software that drives it.

SVF Specification

For the purposes of XC9500 ISP, only three records of the thirteen SVF records that describe the standard are needed. Those three records are discussed in this section.

An SVF file contains a set of ASCII statements. The maximum number of characters allowed on a line is 256, however one SVF statement can span more than one line. Each statement consists of a command and its associated parameters, terminated by a semicolon. SVF is not case sensitive and comments are indicated by an exclamation point (!) or a pair of slashes (//) at the beginning of a line, terminated by a carriage return.

Scan data within a statement is expressed in hexadecimal and is always enclosed in parenthesis. The scan data cannot specify a data string that is larger than the specified bit length; the Most Significant Bit (MSB) zeros in the hex string are not considered when determining the string length. The bit order for scan data defines the LSB (right-most bit) as the first bit scanned into the device for scan data specified by the TDI and SMASK keywords, and is the first bit scanned out for data specified by the TDO and MASK keywords.

The following SVF Commands are supported by the XC9500 EZTag software:

- SDR (Scan Data Register).
- SIR (Scan Instruction Register).
- RUNTEST.

In each of the following command descriptions the parameters are mandatory. Optional parameters are enclosed in brackets ([]). Variables are shown in *italics*. Parenthesis "(") are used to indicate hexadecimal values.

A scan operation is defined as the execution of an SIR or SDR command and any associated header or trailer commands.

SDR, SIR

```
SDR length [TDI (tdi)] [TDO (TDO)] [MASK (msk)] [SMASK (smask)] [PIO (pio)];
```

```
SIR length [TDI (tdi)] [TDO (TDO)] [MASK (msk)] [SMASK (smask)] [PIO (pio)];
```

These commands specify a scan pattern to be applied

to the target scan registers. The SDR command (Scan Data Register) specifies a data pattern to be scanned into the target device Data Register. The SIR command (Scan Instruction Register) specifies a data pattern to be scanned into the target device Instruction Register.

Parameters:

length — A 32-bit decimal integer specifying the number of bits to be scanned.

[TDI (*tdi*)] — (optional) This specifies the value to be scanned into the target, expressed as a hex value. If this parameter is not present, the value of TDI to be scanned into the target device will be the TDI value specified in the previous SDR/SIR statement. If a new scan command is specified, which changes the length of the data pattern with respect to a previous scan, the TDI parameter must be specified, otherwise the default TDI pattern is undetermined and is an error.

[TDO (*tdo*)] — (optional) This specifies the test values to be compared against the actual values scanned out of the target device, expressed as a hex string. If this parameter is not present, no comparison will be performed. If no TDO parameter is present, the MASK will not be used.

[MASK (*mask*)] — (optional) This specifies the mask to be used when comparing TDO values against the actual values scanned out of the target device, expressed as a hex string. A "0" in a specific bit position indicates a "don't care" for that position. If this parameter is not present, the mask will equal the previously specified MASK value specified for the SIR/SDR statement. If a new scan command is specified which changes the length of the data pattern with respect to a previous scan, the MASK parameter must be specified, otherwise the default MASK pattern is undefined and is an error. If no TDO parameter is present, the MASK will not be used.

[SMASK (*smask*)] — (optional) This specifies which TDI data is "don't care", expressed as a hex string. A "0" in a specific bit position indicates that the TDI data in that bit position is a "don't care". If this parameter is not present, the mask will equal the previously specified SMASK value specified for the SDR/SIR statement. If a new scan command is specified which changes the length of the data pattern with respect to a previous scan, the SMASK parameter must be specified, otherwise the default SMASK pattern used is undefined and is an error. The SMASK will be used even if the TDI parameter is not present.

Example:

```
SDR 24 TDI (000010) TDO (818181) MASK (FFFFFF) SMASK (0);
```

```
SIR 16 TDO (ABCD);
```

RUNTEST

RUNTEST *run_count* TCK;

This command forces the target IEEE 1149.1 bus to the Run- Test/Idle state for a specific number of TCK clock periods. This can be used to specify latency periods when operating the TAP.

Parameters:

run_count — The number of TCK clock periods that the 1149.1 bus will remain in the Run Test/Idle state, expressed as a 32 bit unsigned number.

Example:

RUNTEST 1000 TCK;

A Sample XVF File is shown as follows:

```
! Begin Test Program
TRST OFF;          !disable test reset line
ENDIR IDLE;        !End IR scan in IDLE
SIR 8 TDI (FE) MASK (FF)
SDR 14 TDI (3afe) MASK (3ff) TDO (0003)
                   SMASK (3ff)
RUNTEST 100 TCK
!End test program
```

Using EZTag to generate an SVF file

This procedure shows how to create an SVF file; it assumes that the Xilinx XACT version 6.0.0 software, or newer, which includes the XC9500 fitter and the EZTag software, is being used.

1. Create the design using XABEL-CPLD or any compatible third-party design entry tool.
2. Fit the design and save it to a JEDEC output file.
3. Invoke the EZTag software from the XACT command line using the following command:

eztag -svf

The following message appears:

```
Xilinx (R) EZTAG XC9500-CPLD-6.0.0 - JTAG
Boundary-Scan Download
Copyright (C) Xilinx Inc. 1991-1995. All
Rights Reserved.
```

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

4. At the **EZTAG?** Prompt type the following command:

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

where **designName** is the name of the design to translate into SVF. Multiple *deviceType:designName* pairs are separated by spaces.

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

Note: For any non-XC9500 part in the JTAG chain make certain that the BSDL file for the specified part is available along the XACT path and is called *device-Type.bsd* (e.g., 4003pc84.bsd for a XC4003 in the PC84 package).

5. Enter any one of the following commands:

erase designName — generates an SVF file that specifies the bit sequence to erase the specified part.

verify designName [-j jedecFileName] — generates an SVF file that specifies the bit sequence to read back the device contents and compares it against the contents of the specified JEDEC file.

program [-b] designName [-j jedecFileName] — generates an SVF file that specifies the bit sequence to program the specified part from a JEDEC file named *designName.jed* (or alternately, the JEDEC file name specified after the “-j”). The program command options add the following functionality:

-b — When using new devices shipped from the factory, this switch skips the erasure process that usually precedes programming. Erasure is not necessary for a device that has not been previously programmed.

6. Exit EZTag by entering the following command:

quit

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

SVF Interpretation

The simplicity of SVF is also one of its major weaknesses. Much of the behavior of SVF, while running, is left unspecified by the standard. In order to optimize SVF stimulus for an application, the interpretations of some operations must be defined more precisely.

RUNTEST TCK

Many ATE manufacturers prefer not to generate bursts of TCK activity because this results in significantly increased test vector file sizes. This increases the overall test cost and can cause the vector set to run inefficiently. Because the RUNTEST record is used to wait for something to happen, most ATE manufacturers interpret the TCK burst specified as a time value. The favored interpretation is that the

number represents a wait time in microseconds. This is how the EZTag-generated SVF files should be interpreted.

SDR predicted TDO values

The SVF specification describes a method for specifying predicted TDO values. It does not, however, specify actions to be taken when the predicted TDO value does not equal the expected values.

When using Xilinx XC9500 parts, the TDO values predicted reflect the status of the just completed operation (which could be an erase or a program operation). If the status is not the success status (which is the value predicted as the TDO value in the generated SVF file) then the following 1149.1 TAP controller state transition sequence should be followed (assuming the TDO validation failure is detected in the EXIT1-DR state):

1. EXIT1-DR
2. PAUSE-DR
3. EXIT2-DR
4. SHIFT-DR
5. EXIT1-DR
6. UPDATE-DR
7. RUN-TEST/IDLE

The above state transition sequence is illustrated in the 1149.1 TAP state diagram in [Figure 2](#).

The net effect of the state transition sequence is to nullify the just-shifted-in programming or erase data and re-apply the previous program or erase data. Note that the ATE application interpreting the SVF must acknowledge this by not advancing beyond the current SVF record.

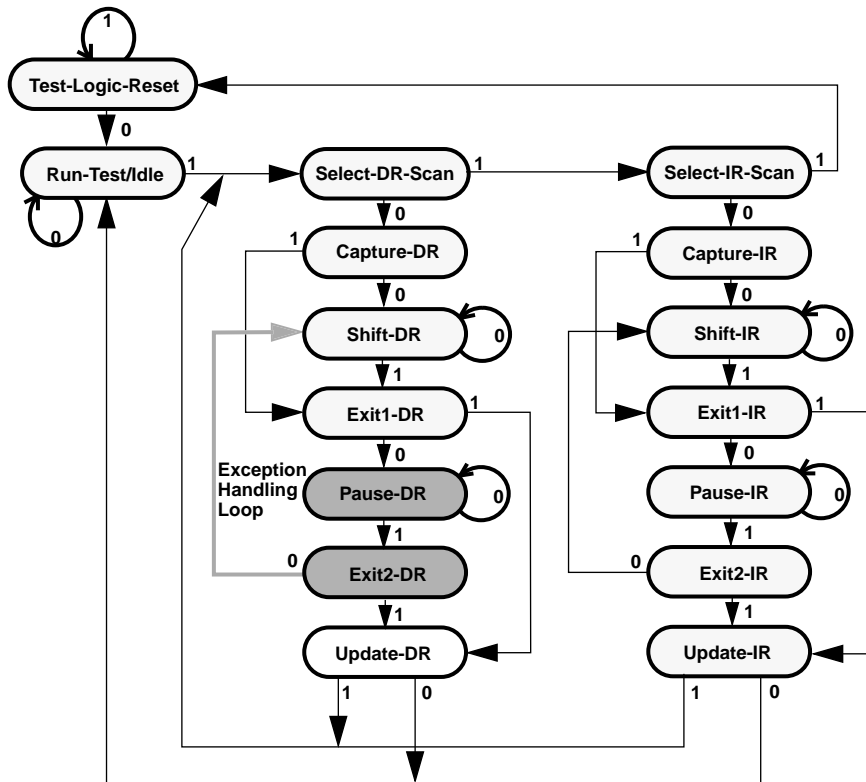


Figure 2: Test Access Port State Diagram

```
// First SDR record
SDR 27 TDI (000003fe) SMASK (07ffffff); // Just apply the value - no test for
TDO
RUNTEST 160000 TCK; // Wait for 160 msec.
// Second SDR record
SDR 27 TDI (008003fe) SMASK (07ffffff) TDO (00000003) MASK (00000003);
// Apply value to TDI read TDO test for concurrence
// if not as expected do "failure recovery loop" - hold
// at this SDR instruction.
RUNTEST 160000 TCK; // Wait for 160 msec
// Third SDR record
SDR 27 TDI (010003fe) SMASK (07ffffff) TDO (00000003) MASK (00000003);
RUNTEST 160000 TCK; // Wait for 160 msec
// Fourth SDR record
SDR 27 TDI (018003fe) SMASK (07ffffff) TDO (00000003) MASK (00000003);
RUNTEST 160000 TCK; // Wait for 160 msec
```

Figure 3: SVF File Fragment Illustrating ATE Flow

Using the SVF file as an example, as shown in [Figure 3](#), the required ATE operation should then be as follows:

1. When reading an SDR instruction with a TDO specified (like the second one in [Figure 3](#)), the predicted TDO value must match the value output from the device on the tester. If it does not match, then the failure recovery loop is executed. In the RUN-TEST/IDLE state a pause is inserted for the amount of time specified in the previously applied RUNTEST instruction.
2. On exit from the RUNTEST instruction, re-apply that same SDR record (in this case the second one in the file) and test the TDO value again.
3. If the TDO matches the expected value, the TAP state machine is transitioned back to RUN-TEST/IDLE the normal way (via EXIT1-DR and UPDATE-DR) and is applied to the next SDR record.
4. This "recovery loop" is to be attempted no more than 32 times. If the TDO value does not match after 32 times, the part is considered defective and the process should abort with some failure indication supplied to the user.

Normally, less than 1% of the addresses fail the TDO check and require the additional erase or program time associated with execution of the failure recovery loop.

Pseudo-code ATE Algorithm

The following pseudo-code describes the sequence of operations that should be used in interpreting the SVF file on a generic ATE.

1. Go to Test-Logic-Reset state
2. Go to Run-Test Idle state
3. Read SVF record

4. if SIR record then
go to Shift-IR state
Scan in <TDI value>
5. else if SDR record then
set <repeat count> to 0
store <TDI value> as <current TDI value>
store <TDO value> as <current TDO value>
6. go to Shift-DR state
scan in <current TDI value>
if <current TDO value> is specified then
if <current TDO value> does not equal <actual TDO value> then
if <repeat count> > 32 then
LOG ERROR
go to Run-Test Idle state
go to Step 3
end if
go to Pause-DR
go to Exit2-DR
go to Shift-DR
go to Exit1-DR
go to Update-DR
go to Run-Test/Idle
increment <repeat count> by 1
pause <current pause time> microseconds
go to Step 6)
end if
else
go to Run-Test Idle state
go to Step 3
endif
7. else if RUNTEST record then
pause tester for <TCK value> microseconds
store <TCK value> as <current pause time>
end if

Predicted ATE Programming Time

Two components make up the overall programming time of the XC9500 part on ATE. The first component is the number of TAP vectors required to perform the specified operations. This is the number of TCK pulses that must be applied to the TAP to execute the specified operations (with TMS and TDI set appropriately). In the SVF file, this is described by the SDR and SIR records. The second component is the latency time associated with program and erase operations. During this period of time, TCK does not need to be pulsed (although it could be). Rather, the tester should simply pause. In the SVF file, this is described by the RUNTEST TCK record.

To determine the overall performance, the speed at which the ATE can stream the TAP stimulus vectors to the part must be considered. The equation below describes the typical programming time for a single XC9500 part.

Definition of Terms:

- FB = the number of function blocks = (the number of macrocells) / 18.
- Tvec = rate of application of ISP vectors = sec/vector, typically @ 10MHz = 0.1 usec/vector.
- Terase = flash erase time for each sector, typically 160 milliseconds.
- Tpgm = flash program time for each address, typically 160 usec.

Note: XC95144 and larger parts have auto-increment mode which significantly reduces number of ATE vectors that must be applied.

Operations:

Parts arriving from the factory are already erased. For these parts, the erase step may be skipped and therefore programming is described by the following equation:

1. Program Only

$$(2880*(FB)**2 + 34561*FB + 60)*Tvec +$$

$$(90*(FB)**2 + 1080*(FB))*Tpgm$$

2. Program (auto-inc.) Only

$$(1350*(FB)**2 + 16201*FB + 91)*Tvec + (90*(FB)**2 + 1080*(FB))*Tpgm$$

Parts being re-programmed must first be erased and then programmed. The EZTag software defaults to using this approach:

3. Erase/Program

$$(2880*(FB)**2 + 34625*(FB) + 74)*Tvec + (2*FB)*Terase + (90*(FB)**2 + 1080*(FB))*Tpgm$$

4. Erase/Program (auto-inc.)

$$(1350*(FB)**2 + 16265*(FB) + 105)*Tvec + (2*FB)*Terase + (90*(FB)**2 + 1080*(FB))*Tpgm$$

Conclusion

By using the EZTag-generated SVF files it is possible to streamline manufacturing flows by programming XC9500 parts on automatic test equipment. This allows integration of the program and test steps of the system manufacturing process. This integration will result in higher system yields, better manufacturability, and simpler part inventory management.

References

Serial Vector Format Specification, Rev C., Texas Instruments.

The Boundary-Scan Handbook, Kenneth Parker, Kiewer Academic Publishers, 1994.

IEEE Standard Test Access Port and Boundary-Scan Architecture, IEEE Std 1149.1-1990 (including IEEE Std 1149.1a-1993)