XAPP503 (v2.0) August 23, 2007

SVF and XSVF File Formats for Xilinx Devices

Authors: Brendan Bridgford and Justin Cammon.

Summary

This application note provides users with a general understanding of the SVF and XSVF file formats as they apply to Xilinx devices. Some familiarity with IEEE STD 1149.1 (JTAG) is assumed. For information on using Serial Vector Format (SVF) and Xilinx Serial Vector Format (XSVF) files in embedded programming applications, refer to [Ref 2].

Introduction

SVF is an industry standard file format that is used to describe JTAG chain operations in a compact, portable fashion. SVF files are portable because complicated vendor-specific programming algorithms can be conveyed by generic SVF instructions, requiring no special knowledge of the target device. Xilinx provides software that can directly generate SVF files for in-system programming Xilinx devices. Xilinx also provides SVF-based embedded solutions for remotely programming Xilinx devices in-system.

File formats are given in "Appendix A: SVF File Format for Xilinx Devices" and "Appendix B: XSVF File Format".

SVF – General

SVF files are used to record JTAG operations by describing the information that needs to be shifted into the device chain. The JTAG operations are recorded in the SVF file with iMPACT or JTAG Programmer. SVF files are written as ASCII text and, therefore, can be read, modified, or written manually in any text editor.

Many third-party programming utilities use an SVF file as an input and can program Xilinx devices in a JTAG chain with the information contained in the SVF file.

XSVF – General

To provide the functionality of an SVF file in a compact, binary format, Xilinx has defined the XSVF format. XSVF files are optimized for performing JTAG operations on Xilinx devices and are intended for use in embedded applications.

Creating SVF and XSVF Files

Creating an SVF or XSVF File With iMPACT Software

The Xilinx iMPACT software can directly generate SVF and XSVF files that apply supported operations to Xilinx devices in a JTAG chain.

iMPACT software is available in the ISE™ Foundation™ software or ISE WebPACK™ software. See http://www.xilinx.com/products/design_resources/design_tool/index.htm. For instructions on how to create an SVF or XSVF file with iMPACT, refer to the iMPACT help section of [Ref 1].

Creating an SVF or XSVF File With iMPACT Command Line

The iMPACT command line interface can be used to generate SVF or XSVF files from the DOS or UNIX command line. This is useful for situations where an automated SVF or XFVF generation flow is required. See the Command Line and Batch Mode section in the iMPACT help of [Ref 1].

Testing SVF and XSVF Files

The iMPACT software can generate SVF or XSVF files for in-system programming Xilinx devices on various platforms and in embedded solutions. In addition, the iMPACT software can execute an SVF or XSVF file in order to test the functionality of the file. When iMPACT executes the SVF or XSVF file, iMPACT applies the corresponding JTAG sequences through a Xilinx cable to the JTAG chain on a target board.

The basic steps to execute a SVF or XSVF file in iMPACT follow:

1. Connect the Xilinx cable to the JTAG chain on the target board and power the board.
2. Start the iMPACT software
3. Create a new project in iMPACT
4. Prepare iMPACT to configure devices in a boundary-scan chain and to allow manual entry of devices in the chain.
5. When iMPACT asks which devices to add to the boundary-scan chain, select one SVF or XSVF file to be added in the boundary-scan window.
6. Click on the SVF or XSVF file displayed in the boundary-scan window to select it.
7. Invoke the **Operation** → **Execute XSVF/SVF** to execute the selected file.

Understanding SVF and XSVF File Formats

BSDL Files and JTAG

The capabilities of any JTAG compliant device is defined in its Boundary Scan Description Language (BSDL) file. BSDL files are written in VHDL and describe a device's pinout and all its Boundary Scan registers. All Xilinx BSDL files have a file extension of `.bsd`, although other manufacturers may use different file extensions. Xilinx BSDL files are available through the Xilinx download webpage at:

http://www.xilinx.com/xlnx/xil_sw_updates_home.jsp

To understand SVF files, users need only be concerned with the following few sections of the BSDL file:

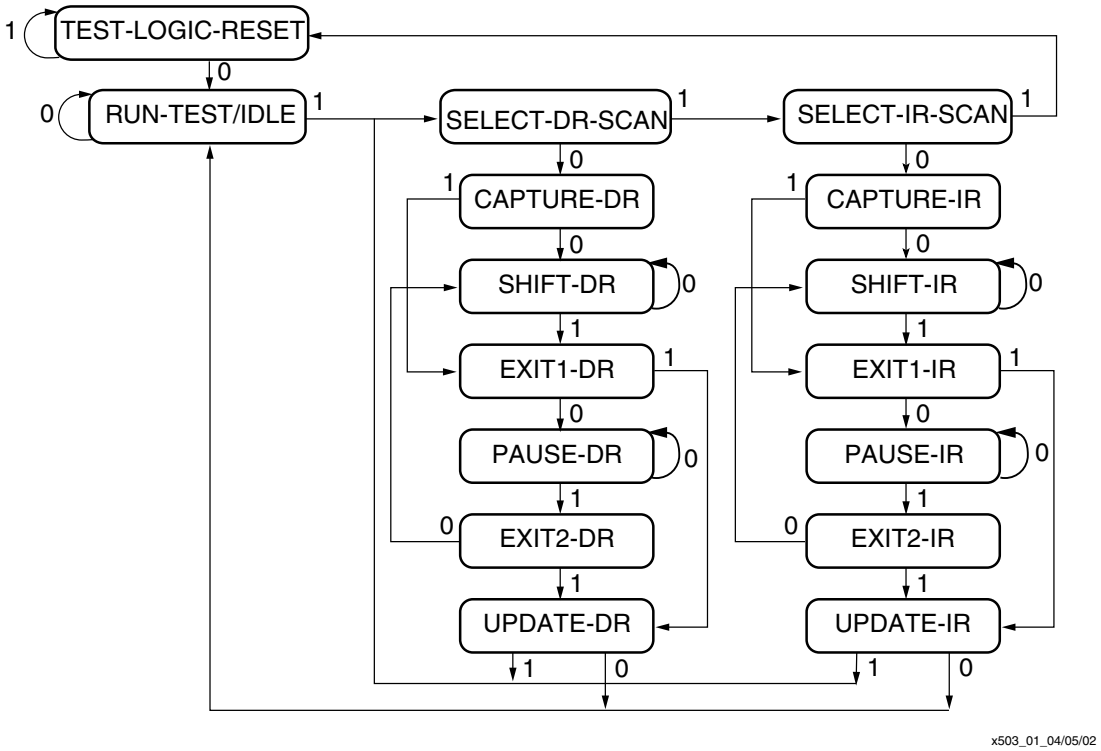
- The Instruction Length Attribute:

  This attribute defines the length of a device's Instruction Register (IR). The IR length is chosen by the device manufacturer and is of arbitrary size greater than 2 bits.
- The Instruction Opcode Attribute:

  This attribute defines the available JTAG instructions. Each JTAG instruction has its own opcode, such as BYPASS, IDCODE, EXTEST, INTEST, etc. Some opcodes, such as the opcode for the BYPASS instruction, are defined by IEEE 1149.1. Other opcodes are defined by the manufacturer.
- The IDCODE Register Attribute:

  Many JTAG compliant devices have a 32-bit IDCODE, which is stored in a special Device ID register. The IDCODE can be used to identify the device manufacturer and part number. To scan the Device ID register, shift the IDCODE instruction into the device, then shift the IDCODE through the device's Data Register (Table 1). All Xilinx devices implement this optional register.

All JTAG operations are controlled through a device's Test Access Port (TAP). The TAP consists of four signals: TMS, TDI, TDO, and TCK. These signals interact with the device through the TAP Controller, a 16-state finite state machine (Figure 1).



Notes:

1. The TAP state transitions occur on the rising edge of TCK. The TMS input value, shown on the state transition arcs, determines the next TAP state.

Figure 1: **JTAG TAP Controller State Diagram**

The IEEE 1149.1 standard defines behavior of the TAP state machine and its output pins according to specified activities on the TAP input pins. TAP state transitions occur on the rising edge of TCK. The TMS input value determines the next state of a TAP state transition. The TDI input value is sampled at the rising edge of TCK during the Shift-DR and Shift-IR TAP states. The TDO output value is updated on the falling-edge of TCK. The TDO output is driven only when the TAP is shifting data or instructions. The TDO output begins to drive when the TAP transitions into the Shift-DR or Shift-IR state, the TDO output continues to drive while the TAP remains in the Shift-DR or Shift-IR state, and the TAP returns to high-impedance at the falling-edge of TCK when the TAP completes the shift operation in the Exit1-DR or Exit1-IR state. At other times, the TDO output is maintained in a high-impedance condition.

One special TAP state sequence to note is the sequence that guarantees the TAP state machine is put into the Test-Logic-Reset state. From any start state, holding TMS High for at least five TCK cycles (five state transitions) leaves the TAP in the Test-Logic-Reset state.All JTAG operations shift data into or out of JTAG instruction and data registers. The TAP Controller provides direct access to all of these registers. There are two classes of JTAG registers: the Instruction register (only one) and Data registers (many). Access to the Instruction Register is provided through the Shift-IR state, while access to the Data Register is provided through the Shift-DR state.

To shift data through these registers, the TAP Controller of the target device must be moved to the corresponding state. For example, to shift data into the Instruction Register, the TAP Controller must be moved to the Shift-IR state, and the data shifted in, LSB first (Figure 2).

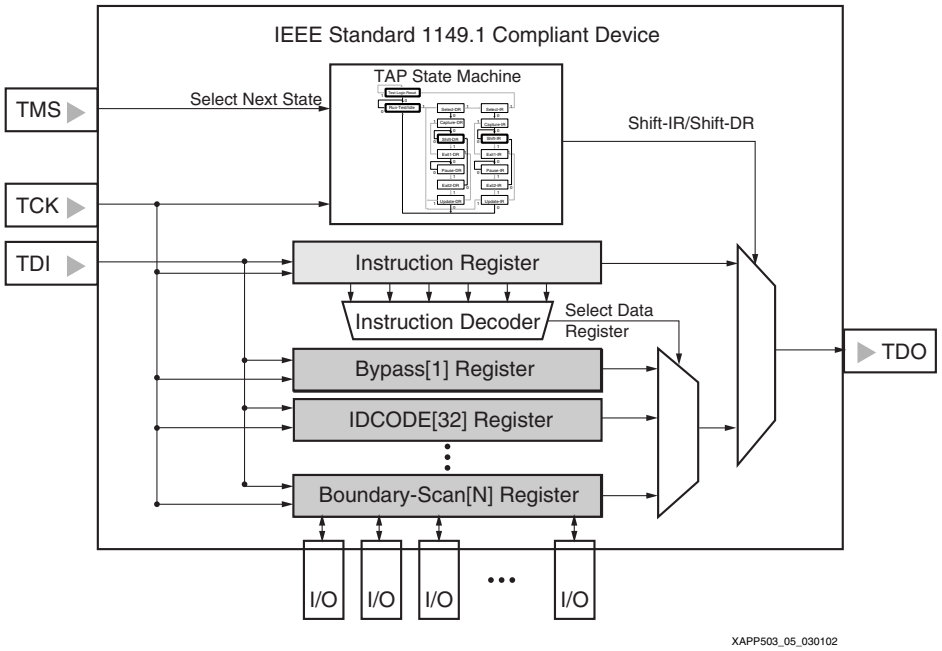


Figure 2: **Typical JTAG Architecture**

Basic SVF Commands

The SVF standard specifies several commands (refer to [Ref 5]) Most of the JTAG operations on Xilinx devices can be performed with a few basic SVF commands.

Scan Instruction Register (SIR)

```
SIR length TDI (tdi) SMASK (smask) [TDO (tdo) MASK (mask)];
```

where:

length – specifies the number of bits to be shifted into the Shift-IR state.

TDI – specifies the scan pattern to be applied to the Shift-IR state.

SMASK – specifies “don’t care” bits in the scan pattern (1 = care, 0 = don’t care).

TDO – specifies the expected pattern on TDO while shifting through the Shift-DR state.

MASK – specifies “don’t care” bits in the expected TDO pattern (1 = care, 0 = don’t care).

Scan Data Register (SDR)

```
SDR length TDI (tdi) SMASK (smask) [TDO (tdo) MASK (mask)];
```

where:

length – specifies the number of bits to be shifted into the Shift-DR state.

TDI – specifies the scan pattern to be applied to the Shift-DR state.

SMASK – specifies “don’t care” bits in the scan pattern (1 = care, 0 = don’t care).

TDO – specifies the expected pattern on TDO while shifting through the Shift-DR state.

MASK – specifies “don’t care” bits in the expected TDO pattern (1 = care, 0 = don’t care).

The third SVF instruction of importance to Xilinx users is the RUNTEST instruction. The RUNTEST instruction specifies an amount of time for the TAP Controller to wait in the Run-Test-Idle state. This wait time is a required part of the programming algorithm for certain Xilinx devices.

RUNTEST

```
RUNTEST run_count TCK;
```

where

run_count – specifies the number of TCK cycles or number of microseconds to wait while the TAP state machine is maintained within the TAP Run-Test state. A general rule can be used to determine the proper interpretation of the run_count value: when the SVF contains JTAG operations for FPGAs, the run_count is interpreted as a minimum number of TCK cycles; or, when the SVF contains JTAG operations for CPLDs or PROMs, the run_count is interpreted as a minimum number of microseconds to wait as if the TCK is running for the specified number of TCK cycles at a frequency of 1 MHz.

When iMPACT preference is set to use absolute time in the SVF file, iMPACT can generate an alternate form of the RUNTEST command in the SVF.

```
RUNTEST min_time SEC;
```

where

min_time – the minimum time to wait while maintaining the TAP in the Run-Test state.

STATE

```
STATE tap_state;
```

where:

tap_state – specifies a TAP state to move the TAP state machine to. Multiple states can be specified in order to specify an explicit path through the TAP state machine.

Note: When RESET is specified as the tap_state, Xilinx tools interpret the state transition as requiring the guaranteed TAP transition to the Test-Logic-Reset state, i.e. hold TMS High for a minimum of five TCK cycles.

Specifying JTAG Shift Operations in SVF for Single-Device Chains

Table 1 shows the necessary activity on TDI and TMS to scan the IDCODE register of an XC9572XL CPLD. The TAP states in Table 1 correspond to the diagram in Figure 1.

Table 1: **JTAG Activity Required to Scan the IDCODE Register of an XC9572XL Device**

	Current TAP State	Next TAP State(1)	TDI	TMS	Notes
1.1	TLR	RTI	X(2)	0	TAP Reset State
1.2	RTI	Select-DR-Scan	X	1	
1.3	Select-DR-Scan	Select-IR-Scan	X	1	
1.4	Select-IR-Scan	Capture-IR	X	0	
1.5	Capture-IR	Shift-IR	X	0	
1.6	Shift-IR	Shift-IR	0	0	Shift the least significant bit (d0) first.
1.7	Shift-IR	Shift-IR	1	0	Shift d1
1.8	Shift-IR	Shift-IR	1	0	Shift d2
1.9	Shift-IR	Shift-IR	1	0	Shift d3
1.10	Shift-IR	Shift-IR	1	0	Shift d4
1.11	Shift-IR	Shift-IR	1	0	Shift d5
1.12	Shift-IR	Shift-IR	1	0	Shift d6

Table 1: **JTAG Activity Required to Scan the IDCODE Register of an XC9572XL Device *(Continued)***

1.13	Shift-IR	Exit1-IR	1[3]	1	Shift d7 while moving to Exit1-IR
1.14	Exit1-IR	Update-IR	X	1	IDCODE instruction (0xFE) has now been passed to the device.
1.15	Update-IR	Select-DR-Scan	X	1	The IDCODE register is now connected through the TAP Shift-DR state.
1.16	Select-DR-Scan	Capture-DR	X	0	
1.17	Capture-DR	Shift-DR	X	0	
1.18	Shift-DR	Shift-DR	0	0	Shift first bit of the device IDCODE out on TDO
1.19	Shift-DR	Shift-DR	0	0	Shift second bit of the device IDCODE out on TDO
1.20	Shift-DR	Shift-DR	0	0	…repeat 29 times…
1.21	Shift-DR	Exit1-DR	0	1	Shift last bit of the device IDCODE out on TDO while moving to Exit1-DR
1.22	Exit1-DR	Update-DR	X	1	
1.23	Update-DR	RTI	X	0	Return to Run-Test-Idle; Operation complete.

Notes:

1. All activity on TDI and TMS is synchronous to TCK.
2. 'X' indicates that TDI is ignored in this state.
3. The IR length of an XC9572XL is 8 bits; its IDCODE instruction is 0xFE .

As Table 1 demonstrates, it is difficult to express JTAG operations in terms of the explicit activity on TMS and TDI. SVF was created to address this problem. Table 2 gives the equivalent SVF syntax to get an IDCODE from an XC9572XL device.

Table 2: **SVF Instructions to Scan the IDCODE Register of an XC9572XL Device**

	SVF Syntax	**Notes**
2.1	`SIR 8 TDI (fe) SMASK (ff);`	Shift the IDCODE Instruction to the Instruction Register
2.2	`SDR 32 TDI (00000000) TDO (f9604093) SMASK (ffffffff) TDO (f9604093) MASK (0fffffff) ;`	Shift the device IDCODE through the Data Register

Explanation of Table 2

2.1 `SIR 8 TDI (fe);`

Shift `11111110` into the target Instruction Register (IDCODE Instruction)

2.2 `SDR 32 TDI (00000000);`

Shift 32 zeros through the Data Register to displace the 32-bit IDCODE. The expected TDO values are `0xf9604093`. This is the IDCODE for the XC9572XL.

Expected TDO output specified by the SDR instruction in Table 2 (item 2.2):

```
TDO -  f9604093: 1111 1001 0110 0000 0100 0000 1001 0011
Mask - 0fffffff: 0000 1111 1111 1111 1111 1111 1111 1111
Expected output: xxxx 1001 0110 0000 0100 0000 1001 0011
9572XL IDCODE  : xxxx 1001 0110 0000 0100 0000 1001 0011
```

Note: The SVF SIR and SDR instructions do not say how to move the TAP controller to the Shift-IR and Shift-DR states. This transition is implied in the SVF standard and must be understood by the program that reads the SVF file.

Specifying JTAG Shift Operations in SVF for Multiple Device Chains

When a JTAG chain contains more than one device, operations are typically performed on one device at a time. Since the TMS and TCK signals are connected to all devices in parallel, it is not possible to move the TAP Controller of one device independently of the TAP Controller of another device. If an instruction is being shifted into one device in the chain, *some* instruction must be shifted into each device in the chain (because each TAP Controller is in the Shift-IR state simultaneously).

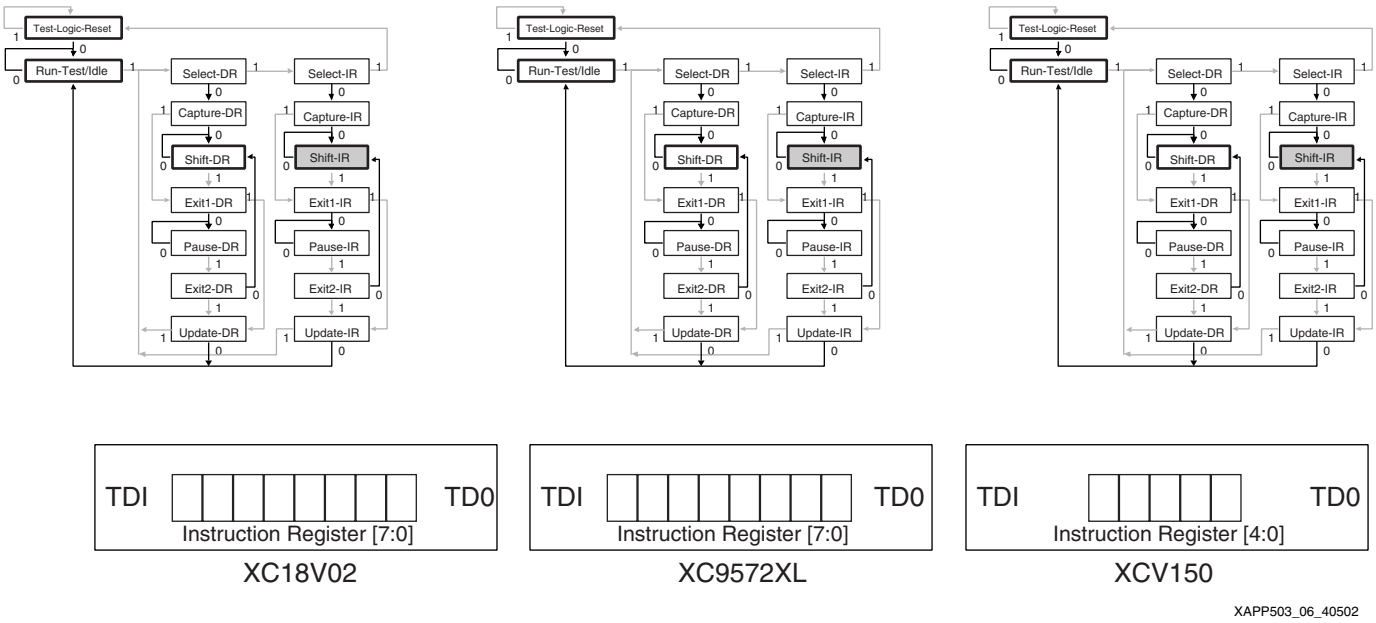


Figure 3: **TAP Controller State Diagrams and IR Contents Prior to IR Shift**

When an operation is going to be performed on a device in the chain, the Bypass instruction is issued to all other devices. IEEE 1149.1 requires that an IR value of all 1s be interpreted as the Bypass instruction for any JTAG device (i.e., if a device's IR is 5 bits long, its Bypass instruction is `11111`; if a device's IR is 8 bits long, its Bypass instruction is `11111111`). To issue the IDCODE instruction to the XC9572XL device in this example, the Bypass instruction is given to the XC18V02 and the XCV150 devices.

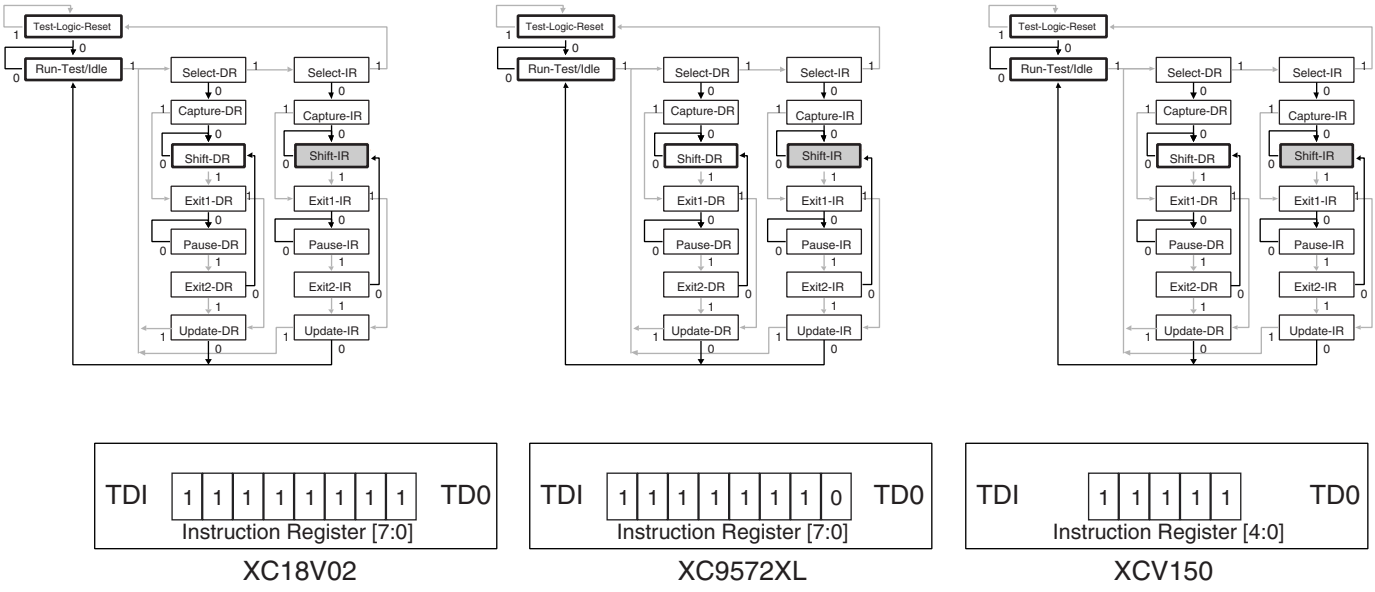


Figure 4: **TAP Controller State Diagrams and IR Contents After Shifting the IDCODE Instruction**

Placing a device in Bypass mode connects its 1-bit Bypass register to the Data Register (Figure 2).

After shifting in the IDCODE operation, the device Device ID register is scanned through the Shift-DR TAP Controller state (Table 2, item 2.2). When the TAP Controllers in this example are moved to the Shift-DR state, the data path becomes a 34-bit pipeline: one bit for the XC18V04 Bypass register, 32 bits for the XC9572XL Device ID register, and one bit for the XCV150 Bypass register.

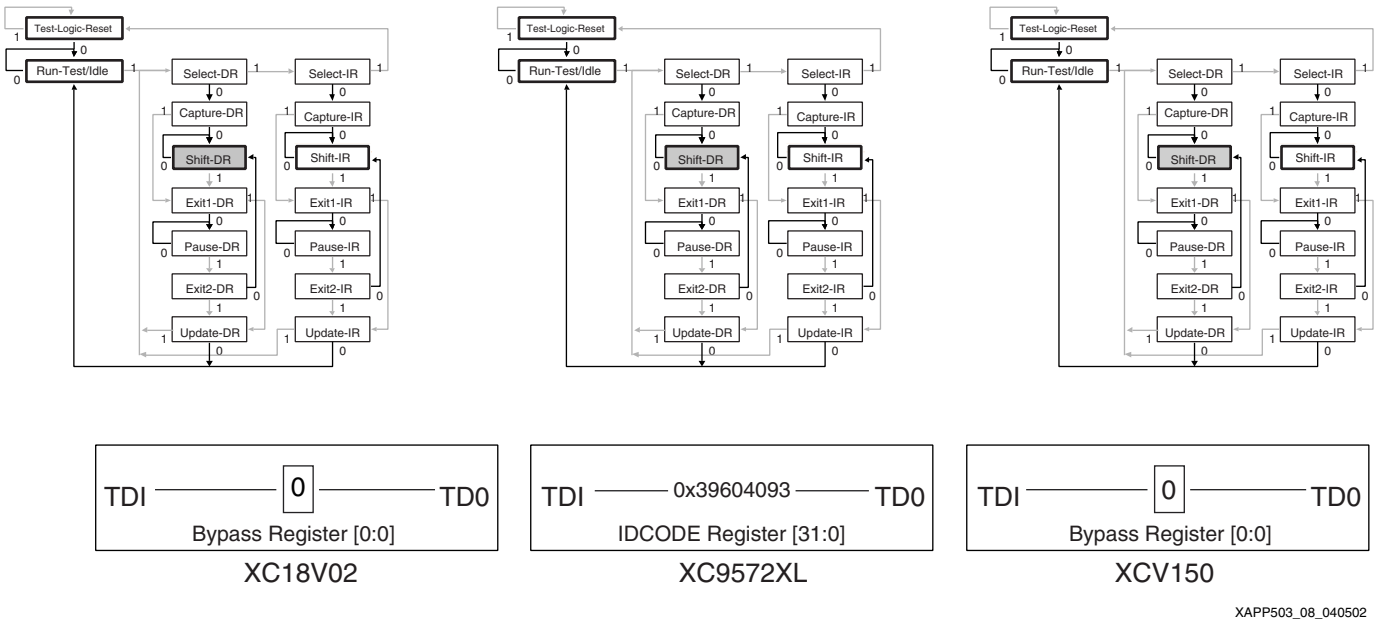


Figure 5: **TAP Controller State Diagrams and DR Contents After Shifting the IDCODE and BYPASS Instructions**

Table 3 gives the equivalent SVF instructions to scan the IDCODE Register of the XC9572XL device in this three-device chain.

Table 3: **SVF Instructions to Scan the Device ID Register of an XC9572XL Device in a Three-Device Chain (XC18V02 → XC9572 →XCV150)**

	SVF Syntax	Notes
3.1	`SIR 21 TDI (1fffdf) SMASK (1fffff);`	Shift the IDCODE Instruction to the Instruction Register chain
3.2	`SDR 34 TDI (01fffffffe) SMASK (03ffffffff) TDO (0012c08126) MASK (001ffffffe) ;`	Shift the device IDCODE through the Data Register chain

Explanation of Table 3

3.1. Shift `0x1fffdf` (`1_1111_1111_1111_1101_1111`) into the Instruction Register chain:

XC18V02 IR <= `11111111` (Bypass)

XC9572XL IR <= `11111110` (IDCODE)

V150 IR <= `11111` (Bypass)

Note that the IR length of the XCV150 is 5 bits, and the IR lengths of the XC18V02 and XC9572XL devices are both 8 bits (to learn the IR length of a particular device, refer to its BSDL file). The SMASK value indicates that all twenty-one TDI shift bits are relevant.

3.2. Move to the Data Register TAP state, and clock it 34 times to displace the contents of the two Bypass registers and the 32-bit Device ID register. The expected output is the 34 Data Register bits plus six additional bits to account for TAP controller state transitions, for a total of 40 expected TDO bits.

```
TDO -  0012c08126 : 0000 0000 0001 0010 1100 0000 1000 0001 0010 0110
Mask - 001ffffffe : 0000 0000 0001 1111 1111 1111 1111 1111 1111 1110
Expected output   : xxxx xxxx xxx1 0010 1100 0000 1000 0001 0010 011x
9572XL IDCODE     :    x xxx1 0010 1100 0000 1000 0001 0010 011
```

SVF Header and Trailer Instructions

Whenever an SVF file is used to perform operations on a chain of devices, several bits must be accounted for that are not of interest. In Table 3, the SIR instruction had to shift the Bypass instruction into the two devices that were not being operated on, and the SDR instruction had to account for two Bypass registers and six other padding bits.

The number of “don’t care” bits in an SVF file increases with the size of the device chain, which can dramatically increase the size of the SVF file. In many cases, several consecutive operations are performed on the same device, each requiring that several “don’t care” bits be specified.

To reduce the size of an SVF file, the SVF specification provides four global padding instructions: Header Instruction Register (HIR), Trailer Instruction Register (TIR), Header Data Register (HDR), and Trailer Data Register (TDR).

`HIR length TDI(tdi) SMASK(smask) [TDO(tdo) MASK(mask]`

Specifies bits to follow subsequent Shift-IR instructions.

`TIR length TDI(tdi) SMASK(smask) [TDO(tdo) MASK(mask]`

Specifies bits to precede subsequent Shift-IR instructions.

`HDR length TDI(tdi) SMASK(smask) [TDO(tdo) MASK(mask]`

Specifies bits to follow subsequent Shift-DR instructions

`TDR length TDI(tdi) SMASK(smask) [TDO(tdo) MASK(mask]`

Specifies bits to precede subsequent Shift-DR instructions.

Note: SVF “Header” instructions specify padding bits at the end of a shift pattern, while “Trailer” instructions specify padding bits at the beginning of a shift pattern. This is a common point of confusion, and may initially seem counterintuitive.

These global commands specify the number of bits to pad the beginning and end of a shift operation, to account for bypassed devices, and provide a simple method of SVF file compression. Once specified, these bits lead or follow every set of bits shifted for the SIR or SDR commands.

Table 4: **Comparison of the IDCODE Operation from Table 3 With and Without Global Padding Instructions**

	SVF Syntax from Table 3 (without global padding instructions)	SVF Syntax from Table 3 (with global padding instructions)
4.1		`TIR 8 TDI (ff) SMASK (ff) ;`
4.2		`HIR 5 TDI (1f) SMASK (1f) ;`
4.3		`HDR 1 TDI (00) SMASK (00) ;`
4.4		`TDR 1 TDI (00) SMASK (00) ;`
4.5	`SIR 21 TDI (1fffdf) SMASK (1fffff);`	`SIR 8 TDI (fe) SMASK (ff) ;`
4.6	`SDR 34 TDI (01fffffffe) SMASK (03ffffffff) TDO (0012c08126) MASK (001ffffffe) ;`	`SDR 32 TDI (00000000) SMASK (ffffffff) TDO (f9604093) MASK (0fffffff) ;`

Explanation of Table 4

4.5 SIR Instruction Without TIR and HIR Instructions

```
SIR 21 TDI 1fffdf: 1 1111 1111 1111 1101 1111
  SMASK 1fffff:  1 1111 1111 1111 1111 1111
Resulting IR Shift:1 1111 1111 1111 1101 1111
```

4.5 SIR Instruction With TIR and HIR Instructions

```
TIR 8 TDI ff:       1 1111 111
    SMASK ff:       1 1111 111
SIR 8 TDI fe                  1 1111 110
    SMASK ff                  1 1111 111
HIR 5 TDI 1f:                           1 1111
    SMASK 1f:                           1 1111
Resulting IR Shift:1 1111 1111 1111 1101 1111
```

Note: Each set of SVF instructions accomplishes the same Instruction Register Shift.

4.6 SDR Instruction Without TDR and HDR Instructions

```
TDO  0012c08126:      00 0001 0010 1100 0000 1000 0001 0010 0110
MASK 001ffffffe:      00 0001 1111 1111 1111 1111 1111 1111 1110
Expected Output:      xx xxx1 0010 1100 0000 1000 0001 0010 011x
```

4.6 SDR Instruction With TDR and HDR Instructions

```
TDR 1 TDI 00:         0
    SMASK 00:         0


SDR TDO f9604093:      1 1111 0010 1100 0000 1000 0001 0010 011
MASK 0fffffff:         0 0001 1111 1111 1111 1111 1111 1111 111


HDR 1 TDI 00:                                                 0
    SMASK 00:                                                 0
Expected Output:      xx xxx1 0010 1100 0000 1000 0001 0010 011x


9572XL IDCODE:         x xxx1 0010 1100 0000 1000 0001 0010 011
```

Note: Each set of SVF instructions expects the same output on the TDO pin.

Special SVF Commands and their TAP State Sequences

The typical JTAG shift operation begins with the TAP in the Run-Test/Idle state and ends with the TAP in the Run-Test/Idle state. The typical shift operation loads an instruction or data into the device and enables the device to run the instructed internal operation upon returning to the Run-Test/Idle state. A sample set of SVF commands comprised of typical shift operations is shown below:

```
SIR 8 TDI(E8);  // Shift the 0xE8 instruction value
SDR 8 TDI(34);  // Shift the 0x34 data value
```

The corresponding TAP state sequence for these typical shift operations is:

1. Run-Test/Idle (start state)
2. Select-DR (beginning state transition for the SIR command)
3. Select-IR
4. Capture-IR
5. Shift-IR and repeat to shift the `0xE8` instruction value
6. Exit1-IR
7. Update-IR
8. Run-Test/Idle (end state for the SIR command)
9. Select-DR (beginning state transition for SDR command)
10. Capture-DR
11. Shift-DR and repeat to shift the 0x34 data value
12. Exit1-DR
13. Update-DR
14. Run-Test/Idle (end state for the SDR command)

Theoretically, the device can run the internal operation that corresponds to the loaded `0xE8` instruction whenever the instruction is active and when the TAP is in the Run-Test/Idle state. In the above sequence, the internal operation for the `0xE8` instruction can run at step 8 and step 14.

In some cases, the device should not run the internal operation for the instruction until after the data is loaded. In the above sequence, step 8 should not be performed. Instead, the TAP sequence should go from the Update-IR state in step 7 directly to the Select-DR state in step 9. The TAP state machine shown in Figure 1 permits the alternate transition from Update-IR to Select-DR instead of the Update-IR to Run-Test/Idle transition shown in the above sequence.

A pair of SVF commands can indirectly affect the TAP state path that are taken between shift operations: ENDIR and ENDDR.

ENDIR

```
ENDIR tap_state;
```

where

tap_state – specifies the state in which to finish any following SIR command. The tap_state persists for all following SIR commands until another ENDIR command changes the tap_state.

By default, the ENDIR tap_state is IDLE (Run-Test/Idle). When the ENDIR tap_state is IDLE, the SIR command leaves the TAP in the Run-Test/Idle state after the instruction is shifted into the device. The default ENDIR tap_state of IDLE means that typical SIR instructions end with a transition through the Run-Test/Idle state.

When the ENDIR tap_state is IRPAUSE (Pause-IR), the TAP state machine is left in the Pause-IR state after the instruction is shifted into the device. When the ENDIR tap_state is IRPAUSE, the TAP sequence for the SIR command is:

1. Start state
2. Select-DR
3. Select-IR
4. Capture-IR
5. Shift-IR and repeat to shift the specified instruction
6. Exit1-IR
7. Pause-IR

The indirect effect of the ENDIR IRPAUSE condition is that the TAP sequence for a shift instruction that follows the SIR instruction skips the Run-Test/Idle state. When an SVF interpreter encounters a shift command and when the starting TAP state is a Pause state, the SVF interpreter follows a TAP state path toward the Shift state that skips the Run-Test/Idle state. For example, the ENDIR command can change the TAP sequence when the original sample set of SVF commands is changed to the following:

```
ENDIR IRPAUSE;// Following SIR commands end in the Pause-IR state
SIR 8 TDI(E8);// Shift the 0xE8 instruction value
SDR 8 TDI(34);// Shift the 0x34 data value
```

The corresponding TAP state sequence for the above set of SVF commands is:

1. Run-Test/Idle (start state)
2. Select-DR (beginning state transition for the SIR command)
3. Select-IR
4. Capture-IR
5. Shift-IR and repeat to shift the `0xE8` instruction value
6. Exit1-IR
7. Pause-IR (end state for the SIR command, per the ENDIR command)
8. Exit2-IR (beginning state transition for the SDR command)
9. Update-IR
10. Select-DR
11. Capture-DR
12. Shift-DR and repeat to shift the 0x34 data value
13. Exit1-DR
14. Update-DR
15. Run-Test/Idle (end state for the SDR command)

The above TAP sequence shows that the Run-Test/Idle state is skipped between the SIR and SDR commands due to the indirect effect of the ENDIR command. This prevents the internal operation for the loaded instruction from being performed until after the data is shifted. The internal operation can be performed in step 15.

ENDDR

```
ENDDR tap_state;
```

where

tap_state – specifies the state in which to finish any following SDR command. The tap_state persists for all following SDR commands until another ENDDR command changes the tap_state.

Note: The default tap_state for ENDDR is IDLE.

Specifying ENDDR DRPAUSE has the same indirect effect on the TAP state transitions following an SDR command as ENDIR IRPAUSE has for the TAP state transitions following an SIR command. Effectively, the Run-Test/Idle state is skipped between an SDR command and any following Shift command. See the ENDIR section for examples of this effect.

XSVF Files

XSVF format is similar in form and function to the SVF format, but without the use of global padding instructions. XSVF files are binary, making them far more compact than ASCII SVF files. There are equivalent XSVF instructions for most SVF instructions:

Table 5: **Equivalent XSVF Instructions for Some SVF Instructions**

SVF	XSVF
HIR, TIR	Accounted for in XSIR instruction
SIR	XSIR
HDR, TDR	Accounted for in XSDR instruction
SDR	XSDR, XSDRB, XSDRC, XSDRE, XSDRTDO, XSDRTDOB, XSDRTDOC
RUNTEST	XRUNTEST

Each XSVF instruction is 1 byte in length and is followed by an argument of variable length. A detailed description of all XSVF instructions is provided in "Appendix B: XSVF File Format".

Table 6 gives a side-by-side comparison of the SVF and XSVF instructions to scan the IDCODE register of an XC9572XL device in a three-device chain.

Table 6: **SVF Instructions to Scan the IDCODE Register of an XC9572XL Device in a Three-Device Chain (XC18V02 → XC9572 → XCV150)**

	SVF File	SVF File
6.1	`TIR 0 ;`	
6.2	`HIR 0 ;`	
6.3	`TDR 0 ;`	
6.4	`HDR 0 ;`	
6.5	`// Validating chain...`	
6.6	`TIR 0 ;`	
6.7	`HIR 0 ;`	
6.8	`TDR 0 ;`	`XREPEAT 0x08`
6.9	`HDR 0 ;`	`XRUNTEST 0x00000000`
6.10	`SIR 21 TDI (1fffff) SMASK (1fffff) TDO (002021) MASK (1c7c63) ;`	`XSIR 0x1D 0x1fffffff`
6.11	`TIR 8 TDI (ff) SMASK (FF) ;`	
6.12	`HIR 5 TDI (1f) SMASK (1F) ;`	
6.13	`HDR 1 TDI (00) SMASK (01) ;`	
6.14	`TDR 1 TDI (00) SMASK (01) ;`	
6.15	`//Loading device with 'idcode' instruction.`	
6.16	`SIR 8 TDI (fe) SMASK (ff) ;`	`XSIR 0x15 0x1fffdf`
6.17		`XSDRSIZE 0x00000022`
6.18		`XTDOMASK 0x001ffffffe`

Table 6: **SVF Instructions to Scan the IDCODE Register of an XC9572XL Device in a Three-Device Chain (XC18V02 → XC9572 → XCV150) *(Continued)***

	SVF File	SVF File
6.19	SDR 32 TDI (00000000) SMASK (00000000) TDO (f9604093) MASK (0fffffff) ;	XSDRTDO 0x0000000000 0x01f2c08126
6.20	//Check for Read/Write Protect.	
6.21		
6.22	SIR 8 TDI (ff) TDO (01) MASK (e3) ;	XSIR 0x15 0x1fffff
6.23	//Loading device with 'idcode' instruction.	
6.24	SIR 8 TDI (fe) ;	XSIR 0x15 0x1fffdf
6.25	SDR 32 TDI (00000000) TDO (f9604093) ;	XSDRTDO 0x0000000000 0x01f2c08126

Explanation of Table 6

6.8: XREPEAT 0x08 – Some devices (especially flash-based devices like the XC18V02 and the XC9572XL) can require more than one attempt at a given operation. The XREPEAT instruction specifies the number of times that an instruction should be retried before exiting with a failure.

6.9: XRUNTEST 0x00000000 – Denotes the amount of time in microseconds to remain in the Run-Test/Idle TAP Controller state after scanning the Data Register.

6.10: XSIR 0x1D 0x1fff ffff – Specifies a shift of 0x1D (29) bits into the Instruction Register. The scan pattern is 1_1111_1111_1111_1111_1111_1111_1111, ensuring that all devices are in Bypass mode.

6.16: XSIR 0x15 0x1fffdf – Specifies a shift of 0x15 (21) bits into the Instruction Register.

```
XSVF: XSIR 0x15 0x1fffdf       : 1 1111 1111 1111 1101 1111
SVF: SIR 8 TDI(fe) SMASK(ff)   :           1 1111 110
TIR/HIR bypass padding         : 1 1111 111          1 1111
Resulting SVF IR shift         : 1 1111 1111 1111 1101 1111
```

XC18V02 IR <= 11111111 (Bypass)

XC9572XL IR <= 11111110 (IDCODE)

V150 IR <= 11111 (Bypass)

6.17: XSDRSIZE 0x00000022 – Indicates that the length of the next XTDO shift instruction is 0x22 (34) bits.

6.18: XTDOMASK 0x001ffffffe – Indicates which two bits are "don't cares".

6.25: XSDRTDO 0x0000000000 0x01f2c08126 – TDO shift out.

```
XSVF: XSDRTDO 01f2c08126 : 01 1111 0010 1100 0000 1000 0001 0010 0110
XTDOMASK                 : 00 0001 1111 1111 1111 1111 1111 1111 1110
XSVF Expected Output     : xx xxx1 0010 1100 0000 1000 0001 0010 011x

SVF: SDR f9604093        : 1 1111 0010 1100 0000 1000 0001 0010 011
TDR/HDR padding          : 0                                      0
SVF Expected Output      : 01 1111 0010 1100 0000 1000 0001 0010 0110

9572XL IDCODE            : x xxx1 0010 1100 0000 1000 0001 0010 011
```

Note: There is a minor difference between the expected TDO output specified in the XSVF file and the expected output given in the SVF file. The first and last bits expected by the XSVF file are "don't cares," while the first and last bits expected by the SVF file are 0s.

When a device is placed in Bypass mode, its Bypass register is always initialized to 0. The XSVF file ignores these bits; the SVF file expects to see 0s. Either method works.

Conclusion

SVF files for Xilinx devices can be generated with Xilinx programming software — either iMPACT or JTAG Programmer. XSVF files are created with the SVF2XSVF file translator, which is available for download with [Ref 2].

SVF files are well suited for programming FPGA, EEPROM, and CPLD devices in-system, because they shield the user from potentially complicated programming algorithms. They are understood by many third-party SVF players and device programmers and have become a *de facto* industry standard.

XSVF files are especially useful for embedded programming solutions, where in-system configuration data can be stored in on-board memory. XSVF files can be read and played back on a microprocessor using the source code provided in [Ref 2].

References

1. Xilinx Software Manuals
2. XAPP058, *Xilinx In-System Programming Using an Embedded Microcontroller*
3. XAPP067, *Using Serial Vector Format Files to Program XC9500/XL/XV Devices In-System*
4. *Texas Instruments IEEE Std 1149.1 (JTAG) Testability Primer*: http://focus.ti.com/lit/an/ssya002c/ssya002c.pdf
5. *ASSET-Intertech Serial Vector Format Specification*: http://www.asset-intertech.com/support/svf.pdf
6. *IEEE 1149.1-2001 Standard Test Access Port and Boundary-Scan Architecture*: http://www.ieee.org

Appendix A: SVF File Format for Xilinx Devices

SVF Overview

This appendix describes the Serial Vector Format syntax, as it applies to Xilinx devices; only those commands and command options that apply to Xilinx devices are described.

An SVF file is the medium for exchanging descriptions of high-level IEEE 1149.1 bus operations which consist of scan operations and movements between different stable states on the 1149.1 state diagram (as shown in Figure 2). SVF does not explicitly describe the state of the 1149.1 bus at every Test Clock (TCK).

An SVF file contains a set of ASCII statements. Each statement consists of a command and its associated parameters, terminated by a semicolon. SVF is case sensitive and comments are indicated by an exclamation point (!) or double slashes (//).

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 Least Significant Bit (LSB) (right most bit) as the first bit scanned into the device for TDI and SMASK scan data, and it is the first bit scanned out for TDO and MASK data.

For the complete Serial Vector Format specification, see [Ref 5].

SVF Commands

The following SVF Commands are supported by the Xilinx devices:

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

For 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) SMASK (smask), [TDO (tdo) MASK (mask)]; SIR length TDI (tdi) TDO SMASK (smask);

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

Prior to scanning the values specified in these commands, the last defined header command (HDR or HIR) is added to the beginning of the SDR or SIR data pattern and the last defined trailer command (TDR or TIR) is appended to the end of the SDR or SIR data pattern.

Parameters:

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

[TDI (tdi)] — (optional) 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 is 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) 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 is performed. If no TDO parameter is present, the MASK is not used.

[MASK (mask)] — (optional) 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 equals 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 is not used.

[SMASK (smask)] — (optional) 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 equals 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 is used even if the TDI parameter is not present.

Example:

```
SDR 27 TDI (008003fe) SMASK (07ffffff) TDO (00000003) MASK (00000003);
SIR 16 TDO (ABCD);
HDR, HIR, TDR, TIR
HDR length TDI(tdi) SMASK(smask) [TDO(tdo) MASK(mask]
HIR length TDI(tdi) SMASK(smask) [TDO(tdo) MASK(mask]
TDR length TDI(tdi) SMASK(smask) [TDO(tdo) MASK(mask]
TIR length TDI(tdi) SMASK(smask) [TDO(tdo) MASK(mask]
```

These commands specify header and trailer bits for data and instruction shifts. Once specified, these bits lead or follow every set of bits shifted for the SIR or SDR commands. These commands are used to specify bits for non-target (bypassed) devices in the scan chain.

The parameters are the same as the SIR and SDR commands.

Example:

```
HDR 1 TDI (0);
TDR 3 TDI (0);
HIR 8 TDI (ff);
TIR 24 TDI (ffffff);
```

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 microseconds, then moves the target device bus to the IDLE state.

The RUNTEST command is typically used to control RUNBIST operations in the target device. Some Xilinx devices require a pause between programming operations; Xilinx uses the RUNTEST operation for this purpose in SVF files. To calculate the number of TCK cycles required for a pause, Xilinx software assumes a TCK frequency of 1 MHz.

Parameters:

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

Example:

```
RUNTEST 1000 TCK;
```

Appendix B: XSVF File Format

This appendix includes the XSVF commands, supported instructions, their arguments, and definitions.

XSVF Commands

The following commands describe the IEEE 1149.1 operations in a way that is similar to the SVF syntax. The key difference between SVF and XSVF is that the XSVF file format affords better data compression and, therefore, produces smaller files.

The format of the XSVF file is a 1-byte instruction followed by a variable number of arguments (as described in the command descriptions below). The binary (hex) value for each instruction is shown in Table 7.

Table 7: **Binary Encoding of XSVF Instructions**

XSVF Instruction	Binary Encoding (hex)
XCOMPLETE	0x00
XTDOMASK	0x01
XSIR	0x02
XSDR	0x03
XRUNTEST	0x04
XREPEAT	0x07
XSDRSIZE	0x08
XSDRTDO	0x09
XSETSDRMASKS	0x0a
XSDRINC	0x0b

Table 7: **Binary Encoding of XSVF Instructions *(Continued)***

XSVF Instruction	Binary Encoding (hex)
XSDRB	0x0c
XSDRC	0x0d
XSDRE	0x0e
XSDRTDOB	0x0f
XSDRTDOC	0x10
XSDRTDOE	0x11
XSTATE	0x12
XENDIR	0x13
XENDDR	0x14
XSIR2	0x15
XCOMMENT	0x16
XWAIT	0x17

XTDOMASK

```
XTDOMASK value<"length" bits>
```

XTDOMASK sets the TDO mask which masks the value of all TDO values from the SDR instructions. Length is defined by the last XSDRSIZE instruction. XTDOMASK can be used multiple times in the XSVF file if the TDO mask changes for various SDR instructions.

Example:

```
XTDOMASK 0x00000003
```

This example defines that TDOMask is 32 bits long and equals `0x00000003`.

XREPEAT

```
XREPEAT times<1 byte>
```

Defines the number of times that TDO is tested against the expected value before the ISP operation is considered a failure. By default, a device can fail an XSDR instruction 32 times before the ISP operation is terminated as a failure. This instruction is optional.

The recommended times value for XSVF containing operations for XC9500/XL/XV CPLDs is 16. For operations on other Xilinx devices, the XREPEAT has no benefit and the times value should be set to zero. The See the SDR Predicted TDO Values section in [Ref 3] for details regarding the repeat loop for XC9500/XL/XV CPLDs.

Example:

```
XREPEAT 0x0f
```

This example sets the command repeat value to 15.

XRUNTEST

```
XRUNTEST time<4 bytes>
```

Defines the amount of time (in microseconds) the device should sit in the Run-Test/Idle state after each visit to the SDR state when the current XENDDR state is IDLE (see the XENDDR command below). The initial XRUNTEST time is zero microseconds.

> ***Note:*** For XSVF containing XRUNTEST commands applied to Xilinx FPGAs, the time parameter must be interpreted as the minimum number of TCK pulses issued within the Run-Test/Idle state after each visit to the SDR state.

Example:

```
XRUNTEST 0x00000fa0
```

This example specifies an idle time of 4000 microseconds.

XSIR

```
XSIR length<1 byte> TDIValue<"length" bits>
```

Go to the Shift-IR state and shift in the TDIValue. If the last XRUNTEST time is non-zero, go to the Run-Test/Idle state and wait for the last specified XRUNTEST time. Otherwise, go to the last specified XENDIR state.

Example:

```
XSIR 0x08 0xec
```

XSDR

```
XSDR TDIValue<"length" bits>
```

Go to the Shift-DR state and shift in TDIValue; compare the TDOExpected value from the last XSDRTDO instruction against the TDO value that was shifted out (use the TDOMask which was generated by the last XTDOMASK instruction). Length comes from the XSDRSIZE instruction.

If the TDO value does not match TDOExpected, perform the exception handling sequence described in the XC9500 programming algorithm section. If TDO is wrong more than the maximum number of times specified by the XREPEAT instruction, then the ISP operation is determined to have failed.

If the last XRUNTEST time is zero, then go to the XENDDR state. Otherwise, go to the Run_Test/Idle state and wait for the XRUNTEST time.

Example:

```
XSDR 02c003fe
```

XSDRSIZE

```
XSDRSIZE length<4 bytes>
```

Specifies the length of all XSDR/XSDRTDO records that follow.

Example:

```
XSDRSIZE 0x0000001b
```

This example defines the length of the following XSDR/XSDRTDO arguments to be 27 bits (4 bytes) in length.

XSDRTDO

```
TDIValue<"length" bits>
TDOExpected<"length" bits>
```

Go to the Shift-DR state and shift in TDIValue; compare the TDOExpected value against the TDO value that was shifted out (use the TDOMask which was generated by the last XTDOMASK instruction). Length comes from the XSDRSIZE instruction.

If the TDO value does not match TDOExpected, perform the exception-handling sequence described in the XC9500 programming algorithm section. If TDO is wrong more than the maximum number of times specified by the XREPEAT instruction, then the ISP operation is determined to have failed.

If the last XRUNTEST time is zero, then go to XENDDR state. Otherwise, go to the Run_Test/Idle state and wait for the XRUNTEST time.

The TDOExpected Value is used in all successive XSDR instructions until the next XSDR instruction is given.

Example:

```
XSDRTDO 0x000007fe 0x00000003
```

For this example, go to the Shift-DR state and shift in `0x000007fe`. Perform a logical AND on the TDO shifted out and the TDOMASK from the last XTDOMASK instruction and compare this value to `0x00000003`.

XSDRB

```
XSDRB TDIValue<"length" bits>
```

Go to the shift-DR state and shift in the TDI value. Continue to stay in the shift-DR state at the end of the operation. No comparison of TDO value with the last specified TDOExpected is performed.

XSDRC

```
XSDRC TDIValue<"length" bits>
```

Shift in the TDI value. Continue to stay in the shift-DR state at the end of the operation. No comparison of TDO value with the last specified TDOExpected is performed.

XSDRE

```
XSDRE TDIValue<"length" bits>
```

Shift in the TDI value. At the end of the operation, go to the XENDDR state. No comparison of TDO value with the last specified TDOExpected is performed.

XSDRTDOB

```
XSDRTDOB TDIValue<"length" bits> TDOExpected<"length" bits>
```

Go to the shift-DR state and shift in TDI value; Compare the TDOExpected value against the TDO value that was shifted out. TDOMask is not applied. All bits of TDO are compared with the TDOExpected. Length comes from the XSDRSIZE instruction.

Because this instruction is primarily meant for FPGAs, if the TDO value does not match TDOExpected, the programming is stopped with an error message. At the end of the operations, continue to stay in the SHIFT-DR state.

XSDRTDOC

```
XSDRTDOC TDIValue<"length" bits>
TDOExpected<"length" bits>
```

Shift in the TDI value; compare the TDOExpected value against the TDO value that was shifted out. Length comes from the XSDRSIZE instruction. TDOMask is not applied. All bits of TDO are compared with the TDOExpected.

If the TDO value does not match TDOExpected, stop the programming operation with an error message. At the end of the operation continue to stay in the SHIFT-DR state.

XSDRTDOE

```
XSDRTDOE TDIValue<"length" bits>
TDOExpected<"length" bits>
```

Shift in the TDI value; compare the TDOExpected value against the TDO value that was shifted out. Length comes from the last XSDSIZE instruction. TDOMask is not applied. All bits of TDO are compared with the TDOExpected.

If the TDO value does not match the TDOExpected, stop the programming operations with an error message. At the end of the operation, go to the XENDDR state.

XSETSDRMASKS

```
XSETSDRMASKS addressMask<"length" bits> dataMask<"length" bits>
```

Set SDR Address and Data Masks. The address and data mask of future XSDRINC instructions are indicated using the XSETSDRMASKS instructions. The bits that are 1 in addressMask indicate the address bits of the XSDR instruction; those that are 1 in dataMask indicate the data bits of the XSDR instruction. "Length" comes from the value of the last XSDRSize instruction.

Example:

```
XSETSDRMASKS 00800000 000003fc
```

Notes:

1. XSETSDRMASKS is no longer used in the XSVF from the iMPACT software. XSETSDRMASKS is described here for systems that are compatible with older XSVF files.

XSDRINC

```
XSDRINC startAddress<"length" bits> numTimes<1 byte>
data[1]<"length2" bits> ...data[numTimes]<"length2" bits>
```

Do successive XSDR instructions. Length is specified by the last XSDRSIZE instruction. Length2 is specified as the number of 1 bits in the dataMask section of the last XSETSDRMASKS instruction.

The startAddress is the first XSDR to be read in. For numTimes iterations, increment the address portion (indicated by the addressMask section of the last XSETSDRMASKS instruction) by 1, and load in the next data portion into the dataMask section.

Notes:

1. An `XSDRINC <start> 255 data0 data1 ... data255` actually does 256 SDR instruction since the start address also represents an S instruction.
2. XSDRINC is no longer used in the XSVF from the iMPACT software. XSDRINC is described here for systems that are compatible with older XSVF files.

Example:

```
XSDRINC 004003fe 05 ff ff ff ff ff
```

XCOMPLETE

```
XCOMPLETE
```

End of XSVF file reached.

Example:

```
XCOMPLETE
```

XSTATE

```
XSTATE state<1 byte>
```

If the state is `0x00` (Test-Logic-Reset), then force the TAP to the Test-Logic-Reset state via the guaranteed TAP reset sequence: hold TMS High and apply a minimum of five TCK cycles. For non-zero state values, if the TAP is already in the specified state, then do nothing. Otherwise, transition the TAP to the next specified state.

Table 8: **Valid State Values for the XSTATE Command**

State Value	TAP State Description
0x00	Test-Logic-Reset
0x01	Run-Test/Idle
0x02	Select-DR
0x03	Capture-DR
0x04	Shift-DR
0x05	Exit1-DR
0x06	Pause-DR
0x07	Exit2-DR
0x08	Update-DR
0x09	Select-IR
0x0A	Capture-IR
0x0B	Shift-IR
0x0C	Exit1-IR
0x0D	Pause-IR
0x0E	Exit2-IR
0x0F	Update-IR

For special states known as stable states (Test-Logic-Reset, Run-Test/Idle, Pause-DR, Pause-IR), an XSVF interpreter follows predefined TAP state paths when the starting state is a stable state and when the XSTATE specifies a new stable state (see the STATE command in the [Ref 5] for the TAP state paths between stable states). For non-stable states, XSTATE should specify a state that is only one TAP state transition distance from the current TAP state to avoid undefined TAP state paths. A sequence of multiple XSTATE commands can be issued to transition the TAP through a specific state path.

XENDIR

```
XENDIR state<1 byte>
```

Set the XSIR end state to Run-Test/Idle (0) or Pause-IR (1). The default is Run-Test/Idle.

Table 9: **Valid State Values for the XENDIR Command**

State Value	TAP State Description
0x00	Run-Test/Idle
0x01	Pause-IR

XENDDR

```
XENDDR state<1 byte>
```

Set the XSDR and XSDRTDO end state to Run-Test/Idle (0) or Pause-DR (1). The default is Run-Test/Idle.

Table 10: **Valid State Values for the XENDDR Command**

State Value	TAP State Description
0x00	Run-Test/Idle
0x01	Pause-DR

XSIR2

```
XSIR2 length<2 bytes> TDIValue<"length" bits>
```

Go to the Shift-IR state and shift in the TDIValue. If the last XRUNTEST time is non-zero, go to the Run-Test/Idle state and wait for the last specified XRUNTEST time. Otherwise, go to the last specified XENDIR state. The XSIR2 command is used for instruction shifts longer than 255 bits. Otherwise, the XSIR command is used to specify instruction shifts of 255 bits or less in length.

Example:

```
XSIR2 0x0008 0xec
```

XCOMMENT

```
XCOMMENT char-string-ending-in-zero
```

The XCOMMENT command specifies an arbitrary length character string that ends with a zero byte.

See Figure 6 for an example of a binary XSVF file, as viewed with a HEX editor (`.xsvf` files are binary and cannot be viewed with a text editor).

XWAIT

```
XWAIT wait_state<1 byte> end_state<1 byte> wait_time<4 bytes>
```

Go to the TAP wait_state, stay in the TAP wait_state for a minimum of wait_time (microseconds), and finally go to the TAP end_state to complete the command.

07	08	04	00	00	00	00	02	1D	1F	FF	FF	FF	02	15	1F
FF	DF	08	00	00	00	22	01	00	1F	FF	FF	FF	09	00	00
00	00	00	01	F2	C0	81	26	02	15	1F	FF	FF	02	15	1F
FF	DF	09	00	00	00	00	00	01	F2	C0	81	26	02	0D	1F
FF	D2	15	1F	FF	FF	08	00	00	00	03	01	00	09	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

XAPP503_03_030102

Figure 6: **A Binary XSVF File, as Viewed With a HEX Editor**

Appendix C: Creating an SVF with JTAG Programmer 3.1i or XPLA Programmer for Legacy Applications

A few legacy applications can require the SVF from the classic JTAG Programmer 3.1i or XPLA Programmer software.

JTAG Programmer 3.1i and XPLA Programmer software are available for download from the ISE Classics software webpage at http://www.xilinx.com/ise/logic_design_prod/classics.htm. The software is available with the ISE WebPACK 3.3WP8.1 release. Choose the programmer module that corresponds to the target device.

See the JTAG Programmer Guide in the 3.x Software Manuals for instructions on generating SVF files.

Appendix D: Using the Stand-Alone SVF2XSVF Utility to Translate SVF to XSVF

For most applications, the iMPACT software directly generates the necessary XSVF file. For a few applications that require customized XSVF files, a translation tool (SVF2XSVF) can be used to convert custom SVF files to the XSVF format.

The SVF2XSVF tool is provided with the downloadable `xapp058.zip` associated with [Ref 2]. Both are available at:

http://www.xilinx.com/xlnx/xweb/xil_publications_display.jsp?category=Application+Notes/Device+Configuration+and+Programming/Embedded+In_System+Configuration&show=xapp058.pdf

The SVF2XSVF tool can take custom-created SVF or the modified SVF, that originated from the iMPACT software, as input. The SVF2XSVF output is the XSVF formatted equivalent of the SVF input file.

Note: The SVF2XSVF tool cannot translate all forms of SVF input. Only SVF commands that can be represented by the supported XSVF commands can be translated correctly.

To create the XSVF file, first download the translator and unzip it to a directory. Use this syntax to translate an SVF file to an equivalent XSVF file:

```
svf2xsvf -i  <input_file.svf>  -o <output_file.xsvf>
-a <text_xsvf.txt>
```

where:

-i designates the input file

-o designates the output file

-a designates an ASCII version of the XSVF file (optional)

The optional -a command-line switch provides a human-readable version of the XSVF that is useful for interpreting the binary XSVF code.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/17/02	1.0	Xilinx initial release.
08/23/07	2.0	• Added the XSVF commands XSIR2, XCOMMENT, and XWAIT to "XSVF Commands," page 17. • Added state values (Table 8, page 22) for the XSTATE command. • Added "Special SVF Commands and their TAP State Sequences," page 11. • Updated the software flows in "Creating SVF and XSVF Files," page 1 to ISE iMPACT 9.2i. • Added "Testing SVF and XSVF Files," page 2.

Notice of Disclaimer