XAPP424 (v1.0.2) April 7, 2008

Embedded JTAG ACE Player

Author: Roy White, and Arthur Khu

Summary

This application note contains a reference design consisting of HDL IP and Xilinx® Advanced Configuration Environment (ACE) software utilities that give designers great flexibility in creating in-system programming (ISP) solutions. In-system programming support allows designers to revise existing designs, package the new bitstream programming files with the provided software utilities, and update the remote system through the JTAG interface using the Embedded JTAG ACE Player.

Included Design Files

Included with this application note are source files, software utilities, and additional documentation available for downloading as a .zip archive.

Introduction

Xilinx FPGA, CPLD, and PROM families incorporate industry standard IEEE 1149.1 JTAG support for in-system programming. Combining this support with IP and software utilities gives designers tools to design remote upgrade and management into their Xilinx-based systems. Features of this in-system upgrade capability include:

- HDL based solution not requiring a microprocessor
- Small footprint — under 250 FPGA slices
- Minimal buffering requirements for embedded system
- Industry-standard IEEE 1149.1 JTAG compliance
- Advanced features for passing user-defined (non-bitstream) information to remote systems
- Combinable with techniques described in other Xilinx application notes to create a complete field upgrade, monitoring, and revision control application

Table 1 lists the Xilinx product families supported by the ACE Player.

Table 1: **Xilinx Product Families Supported**

Xilinx Product Family	Devices Supported
FPGA	Virtex®, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Virtex-5, Spartan®-II, Spartan-IIE, Spartan-3, Spartan-3A, Spartan-3E FPGAs
CPLD	Coolrunner™-II CPLDs(1)
PROM	18Vxx, Platform Flash Configuration PROMs

Notes:

1. Refer to [Ref 1] for a solution for remote upgrade of 9500/XL/XV families.

In-system programming support allows designers to develop and test a revised bitstream in the lab, package the new file with the provided software utilities, and then perform the upgrade on the remote system through the JTAG interface using the ACE Player.

Using the ACE Player

iMPACT is used to convert the finished design to a standard serial vector format (SVF) ASCII file (Figure 1). The resulting SVF file is then converted to the more compact and efficient ACE file format by the utility **SVF2ACE**.

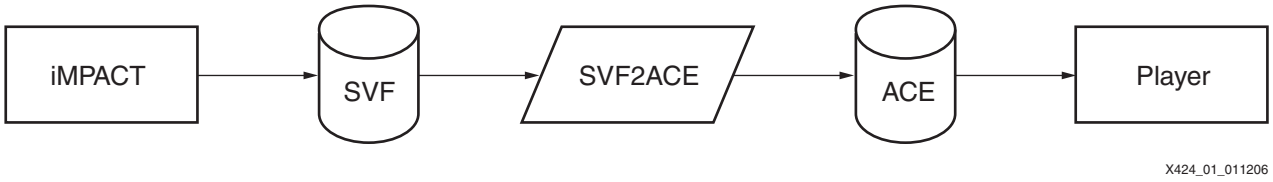

Figure 1: **Design Flow For Creating and Using ACE File**

The ACE file contents are then "played" to the JTAG port on the target system by the Player implemented in Xilinx programmable logic (Figure 2).

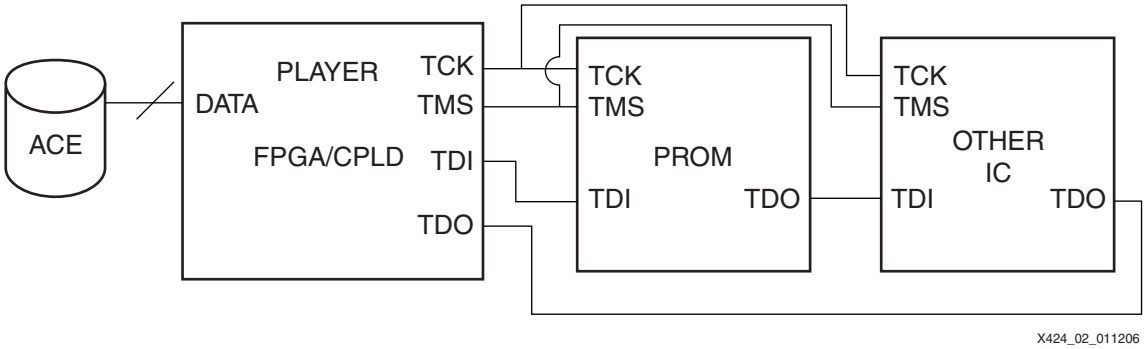

Figure 2: **Player Interface to JTAG Interface**

Components of Embedded JTAG ACE Player System

Serial Vector File Format

Use of the Embedded ACE Player begins with a serial vector format file, commonly referred to as SVF. SVF was jointly developed by Texas Instruments and Teradyne in 1991 as a standard ASCII format for expressing test patterns for IEEE STD 1149.1-based tests. SVF controls the IEEE 1149.1 (JTAG) test bus using commands that specify a transition of the JTAG TAP Controller from one steady state to another. SVF does not describe the explicit state of the JTAG system on every TCK cycle, but rather describes it in terms of transactions conducted between stable states. The target system is left to interpret the SVF instructions, control the operation of the JTAG TAP controller, and clock in the data.

For more information on the JTAG and SVF specification, refer to the Texas Instruments *IEEE JTAG 1149.1 Primer* at:

http://focus.ti.com/lit/an/ssya002c/ssya002c.pdf.

It is not necessary to understand the SVF specification to use the Embedded JTAG ACE IP and utilities. The ACE utilities provided by Xilinx process the SVF file written by iMPACT without any modification required by the designer.

Xilinx ACE File Format

While SVF is a powerful language for controlling a JTAG system in a test environment, it is not ideal for executing the operations used in configuring programmable logic devices in memory-constrained, embedded systems. For this reason, Xilinx developed the ACE file format and supporting software utilities to extend the value of SVF for use in embedded systems. Using the ACE File format preserves full compatibility with industry-standard SVF, while optimizing it for Xilinx programmable logic applications.

The ACE file format is defined in “Appendix B: Xilinx ACE File Format and Instruction Reference,” page 11.

Software Utilities for Creating the ACE File

The SVF2ACE utility contains sophisticated algorithms that process the SVF file and calculates the data and instruction bit shifts needed to scan in the bitstream data. This translation to the ACE format greatly reduces the processing and memory requirements of the embedded system, while preserving compatibility with industry standard SVF and IEEE 1149.1 JTAG standards.

The SVF2ACE utility also provides advanced options to pass user-defined (non-bitstream) data to a target system, to meet the customization needs of sophisticated field upgrade applications.

Detailed information on the usage of **SVF2ACE** for Embedded JTAG applications is provided in "Appendix A: ACE File Utilities for Embedded JTAG Applications," page 9.

HDL Player

The embedded JTAG ACE hardware player is HDL IP that executes the instructions and data shifts contained in the ACE file to control the JTAG TDI, TMS, TDO, and TCK pins — in effect, *playing* the data to the system's JTAG interface. The HDL code can be re-targeted to any supported Xilinx device (Table 1, page 1).

Example Implementations of the Embedded JTAG ACE Player

The ACE Player can be added to an existing FPGA design that has the ability to receive a file from a remote source such as a network or backplane, or interfaced to a microprocessor that can receive the ACE File.

Figure 3 illustrates typical systems utilizing the ACE Player.

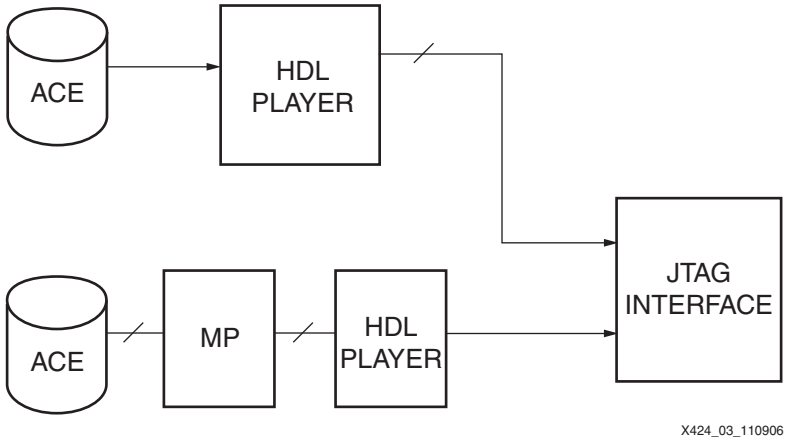

Figure 3: **Example Configurations for Implementing an ACE Player**

Creating and Using the Embedded JTAG SVF File

Generate SVF file using iMPACT

Once the FPGA design files are created, the iMPACT GUI guides the user through JTAG chain set-up and generation of an SVF file.

For more information, consult the iMPACT documentation located at:

http://www.xilinx.com/support/software_manuals.htm.

Generate the ACE Programming File

The SVF file created in the previous step is then processed into a Xilinx ACE file with the **SVF2ACE** utility provided in the download archive.

Command line usage:

```
svf2ace -i input_file.svf -o player_file.ace -header header_file.txt -tck
<cycles>
```

where:

-i: `Input_file.svf` is supplied by the Xilinx iMPACT Software program

-o: User-defined file for use by any ACE player implementation

-header: [OPTIONAL] input file of user-defined data.

-tck: [OPTIONAL] tck frequency (cycles) in Hz.

Refer "Appendix A: ACE File Utilities for Embedded JTAG Applications," page 9 for more details.

The resulting file with `.ace` extension is a binary file ready to be executed by an embedded JTAG ACE Player.

Using the HDL Player IP

The Player HDL IP (Figure 4) is written in a microprogram format. A spreadsheet of the player's operation is included with the download archive and can be used to understand its operation as well as to add user-defined extensions.

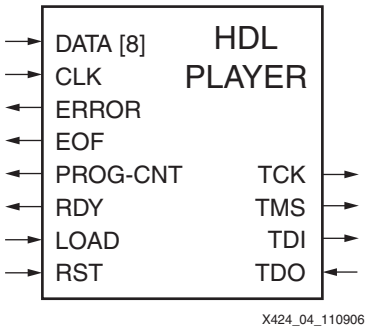

Figure 4: **HDL Player**

Each application has its own method of interfacing to a network or backplane to receive the upgrade ACE file, so it is left to the designer to create a byte-wide interface between the end application and the Player.

All Player interface signals are synchronous. The Player asserts a Ready signal and waits for the Load signal to go High. LOAD is captured on the following rising edge and propagates through metastability circuits. DATA is captured and RDY goes LOW four clock cycles after LOAD is captured. The Player only accepts one byte per asserted LOAD signal. LOAD must transition to Low for at least one Player clock cycle between LOADs. LOAD can be asserted before RDY goes High as long as it conforms to the transition rule (Figure 5).

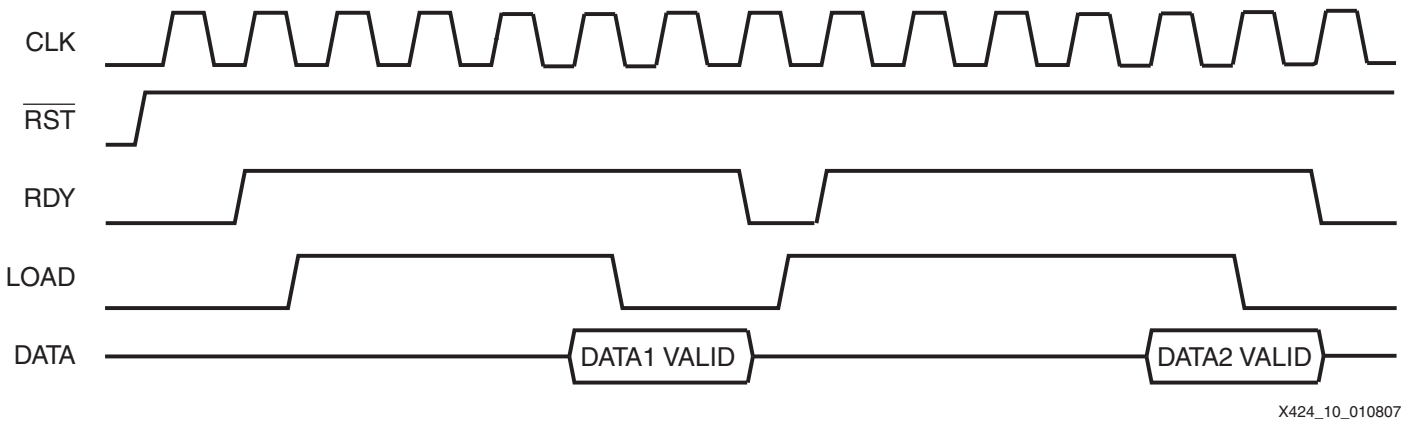

Figure 5: **Timing Diagram for Data Write Cycle to Player**

Data Flow Processing of ACE File

Figure 6, page 6, Figure 7, page 7 and Figure 8, page 8 illustrate the data flow for ACE file processing, showing how an each byte of an ACE file is evaluated by the ACE Player from beginning to end-of-file.

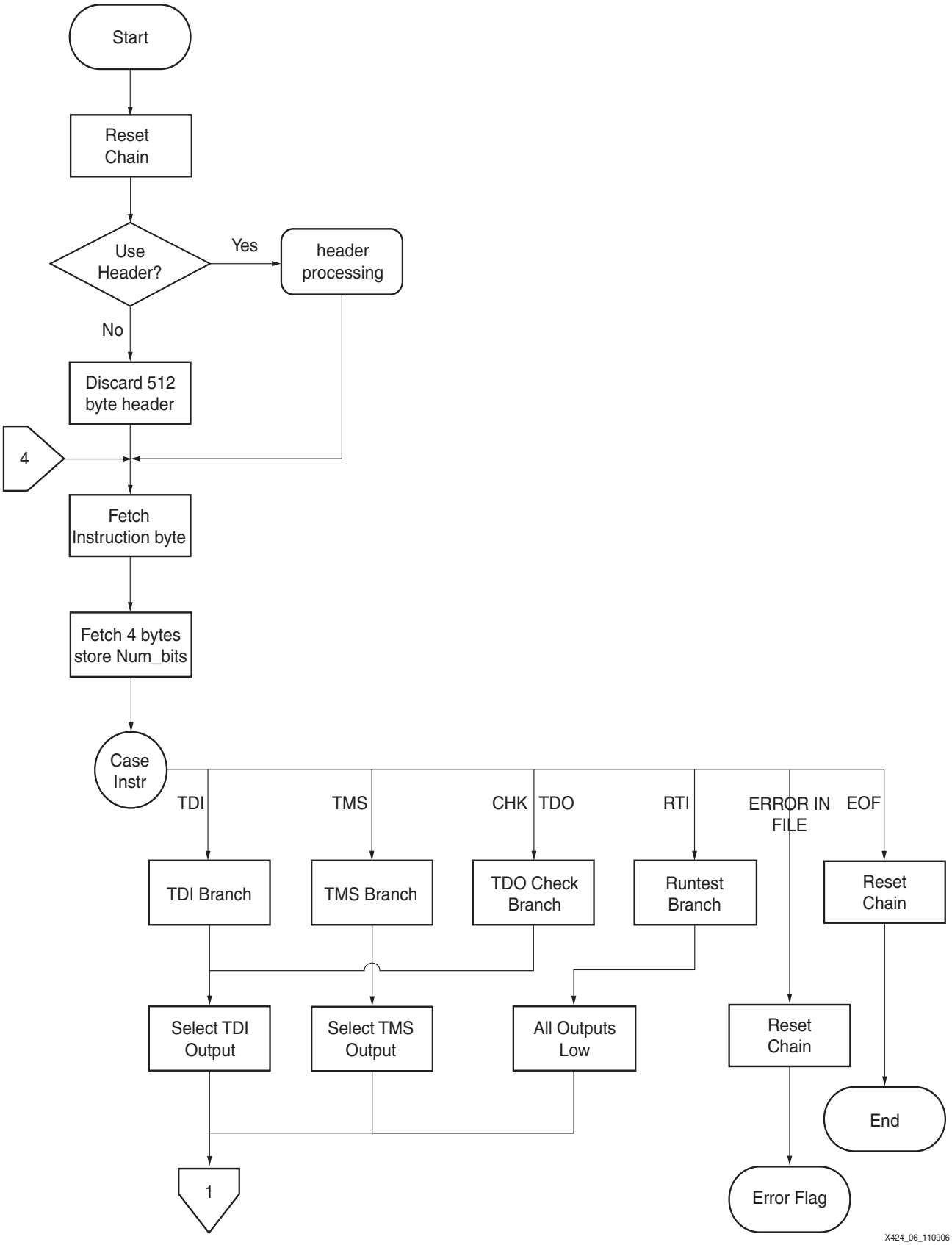

Figure 6: **Data Flow Processing of ACE File – Part 1**

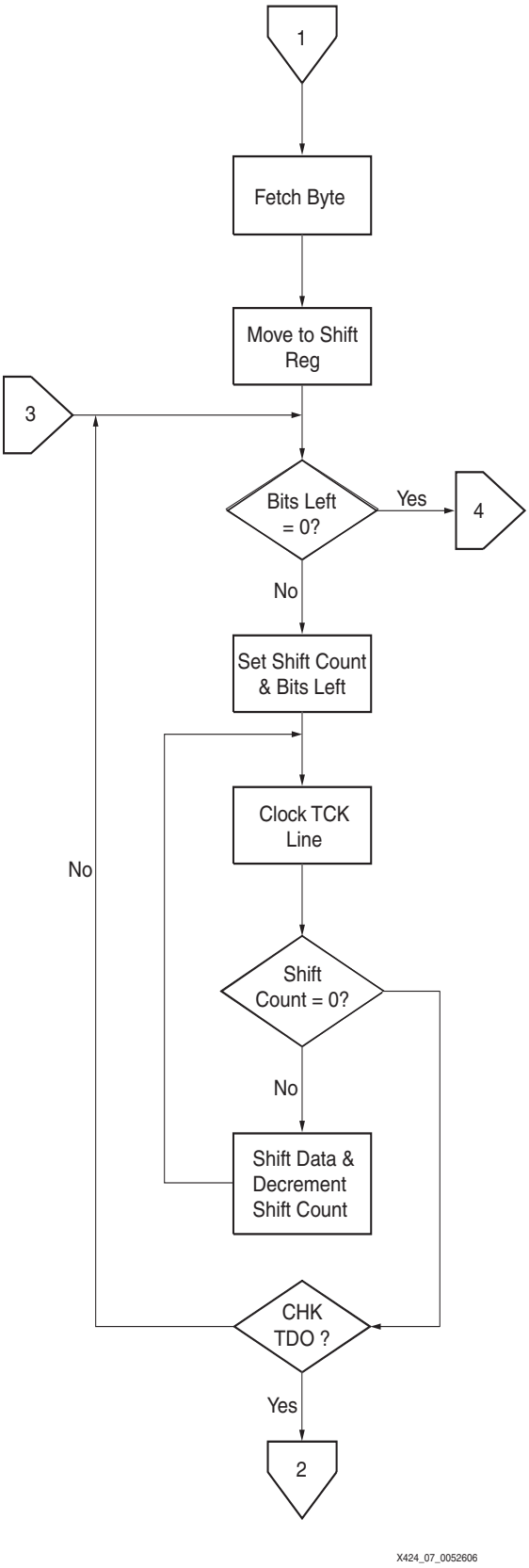

Figure 7: **Data Flow Processing of ACE File – Part 2**

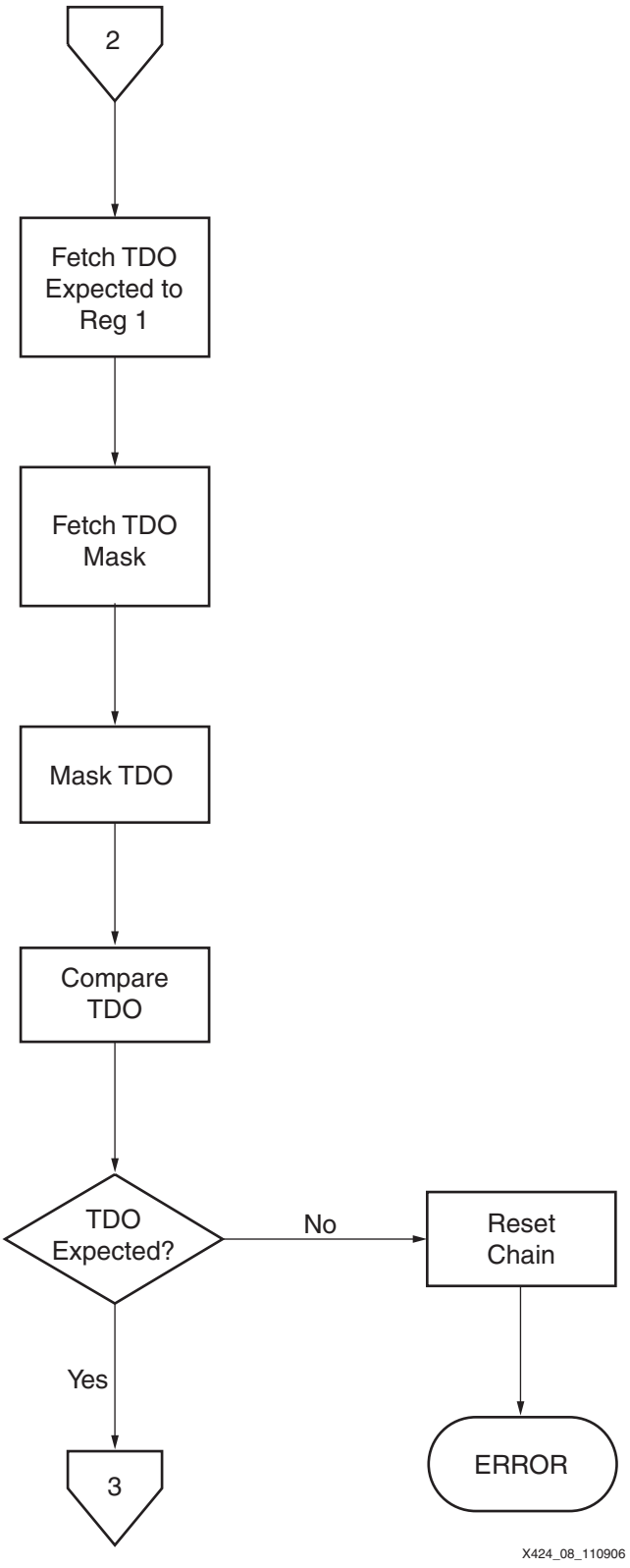

Figure 8: **Data Flow Processing of ACE File – Part 3**

Conclusion

The Xilinx Embedded JTAG ACE Player provides designers with flexible, efficient options for designing in-system programmability into embedded systems.

For More Information

1. XAPP058, *Xilinx In-System Programming Using an Embedded Microcontroller*
2. XAPP503, *SVF and XSVF File Formats for Xilinx Devices*
3. XAPP693, *A CPLD-Based Configuration and Revision Manager for Xilinx Platform Flash PROMs and FPGAs*
4. XAPP502, *Using a Microprocessor to Configure Xilinx FPGAs via Slave Serial or SelectMAP Mode*
5. XAPP500, *J Drive: In-System Programming of IEEE Standard 1532 Devices*
6. XAPP412, *Architecting Systems for Upgradability with IRL*

Appendix A: ACE File Utilities for Embedded JTAG Applications

Creating Embedded JTAG ACE applications requires the **SVF2ACE** utility version 1.19 or later, included in the download archive. Versions for Windows and Linux are provided.

> ***Caution!*** The **SVF2ACE** utility is also used to generate programming files for Xilinx System ACE™ CF applications, but these files are not interchangeable with those created for Embedded JTAG ACE Players. Xilinx iMPACT should always be used for any System ACE CF applications.

Command Line Options

These command line options are applicable when used with an Embedded JTAG ACE Player only:

```
svf2ace -i input_file.svf -o player_file.ace -header header_file.txt -tck <freq>
```

-i: `input_file.svf` is supplied by the Xilinx iMPACT Software program

-o: User-defined file for use by embedded player

-header: [OPTIONAL] input file of user-defined data to be included in the user section of the 512-byte header

-tck: [OPTIONAL] JTAG clock frequency in Hz

-wtck: [OPTIONAL, but required for FPGA configuration] generate ACE file with TCK driving during all RUNTEST wait cycles

Note: The **-wtck** option is required for FPGA configuration via JTAG. This option forces TCK to toggle during all JTAG RUNTEST wait cycles.

Header File Specification

It is often desirable in remote upgradeable systems to be able to pass user-defined information in addition to programming data. Examples include:

- Signaling the target to set a register selecting a new default configuration PROM in a multi-PROM system
- Sending security information to the target for controlling upgrade access
- Changing the frequency of the JTAG clock in systems with control over this variable
- Sending messages to a microprocessor-based system for other system control functions

The command line option `-header input_file.txt` allows a user-defined string of up to 127 bytes to be included in the ACE file header. The header input file contains an ASCII string enclosed in C-style comment brackets representing HEX values read by **SVF2ACE** and placed in the header beginning at byte 257. The length of the user-defined string is calculated and placed in byte 256. Use of the string-length field is optional. If no `-header` is specified at the command line, NULL bytes are placed in this section of the header. Comments can be placed in the file before or after the ASCII string and is ignored.

Example:

```
comments/*FEA09B347DEFFF*/more comments
```

Results in `7FEA09B347DEFFF` being placed starting at byte 256 of the header, indicating that seven user-defined bytes follow consisting of `FEA09B347DEFFF`. The header input file can be created with any standard text editor.

The following error conditions can occur when the `-header` command is executed:

- `error non-hex characters` – non-HEX (0–F) characters found between comment delimiters.
- `error exceeded max header length` – the length of the user-defined header exceeded.

-tck: Clock Scaling Specification

The `-tck` parameter is used to scale **RUNTEST** parameters for JTAG clock frequencies greater than 1 Mhz. The SVF instruction, **RUNTEST**, is used to insert wait states in the JTAG system and is necessary for certain PROM programming operations. iMPACT assumes a 1 MHz TCK frequency when calculating the necessary wait times for the target device when generating the SVF file. In systems with a JTAG TCK frequency slower than 1 MHz, this is not an issue, and the real wait times are slightly longer than calculated. However, in systems running faster than 1 MHz, the real wait times are shorter than calculated and might not meet the requirements of some devices.

If the actual JTAG clock frequency is faster than 1 Mhz, the **SVF2ACE** utility can scale the number of TCK wait cycles accordingly to correct for this discrepancy. For example, a frequency of 2 MHz specified with this parameter causes **SVF2ACE** to scale up the RUNTEST wait cycles by a factor of 2.

The frequency value is expressed according to the standard SVF format specification which allows the frequency to be expressed as either an integer value, or in scientific notation form.

Example:

```
-tck 2000000 : 2 million hz = 2 Mhz
-tck 2E6: scientific notation form of 2 Mhz
```

The possible range for this specification is 1 MHz to 500 MHz. If no `-tck` value is specified at the command line, no scaling is done and the **RUNTEST** times are based on the 1 MHz default TCK frequency assumed in the SVF file.

Appendix B: Xilinx ACE File Format and Instruction Reference

Although understanding the ACE file format and instruction set is not required to use the Xilinx Embedded JTAG ACE Player, they are documented here for those users wanting to customize applications or make use of advanced features.

ACE File Structure

The ACE file is a binary file consisting of two components: the device programming data, and a 512-byte header containing both Xilinx and user-defined information. Only the header is modifiable by the user.

Header Format

Bytes 0 through 255, and 384 to 511, contain Xilinx-defined information. Bytes 256 through 383 are user-defined.

Byte Location Number	Use
0…255	Xilinx use only
256..383	`<length field # 256><User defined data follows>`
384..511	Xilinx use only
512…End	Programming Data

Refer to the `-header` file specification in "Appendix A: ACE File Utilities for Embedded JTAG Applications," page 9 for user-defined usage of the header.

ACE Instruction Reference

An ACE file contains five possible instructions derived from the standard SVF specification and is read in LSB order to the end. All ACE instructions and data are on byte-wide boundaries. The instruction format is:

```
<ACE Instruction (8)> <number of bits (32)> < data (8)>< data(8) > <data(8)>
```

The 32-bit value representing the number of bits is written out as 4 bytes, with byte 0 containing the lower order 8 bits of the 32-bit number and byte 3 containing the 8 highest-order bits of the 32-bit number. There is no delimiter between instructions or data. All data shifts start at the LSB.

ACE Inst	Description	Format
`0x02`	Shift data to TMS pin	`<0x02> <#bits 4 bytes> <tms_data>`
`0x03`	Shift data to TDI pin	`<0x03> <#bits 4 bytes> <tdi_data>`
`0x04`	Shift data to TDI pin and check TDO shifted in	`<0x04> <#bits 4 bytes> <tdi_data><tdo expected><mask> <tdi_data><tdo expected><mask>...`
`0x05`	RUNTEST – wait TCK cycles w/o clocking TCK	`<0x05> <#TCK cycles -4 bytes>`
`0x07`	END	`<0x07>`

Example:

```
030C000000FF1F = <03> <0C><00><00><00> <FF><1F>
```

Results in the instruction "shift data to TDI pin", in this case, 13 bits of data (shift count goes from 12 to 0), are contained in the final two bytes of the instruction `FF 1F` means:

1. Shift first eight bits (`FF`) `1111 1111` starting at the LSB `1`
2. Shift five more bits `0001 1111` starting at the LSB `1`

The last three zeros are not used.

Appendix C: Example Usage of the User-Defined Header Option

Refer to "Appendix A: ACE File Utilities for Embedded JTAG Applications," page 9 for detailed usage instructions for including user-defined information in the ACE file header. Following is an example of modifications to the provided HDL IP in order to use the user-defined header.

This example demonstrates how to add user-defined code with minimum of changes to the provided HDL IP. Refer to the microprogram table provided in the download archive.

The method in this example adds a subroutine to the end of the existing HDL, and the microprogram jumps to this section to process the user-defined header and then returns to the previous branch point to continue playing the file. The modifications only requires that one value be added to the Jump table, and all additional operations can be added to the end of the existing HDL IP code.

Example design specifications:

- The ACE file is to contain a unique identifier string of AAFFFFFFFF.
- The entire string is evaluated by the user application during file transmission, but only the first byte AA needs to be evaluated by the Player.
- The Player should evaluate AA in the header, and if found, set the register called example_reg to the value of 3.

The user-defined file would contain the string:

```
"this is a comment - change register value based on AA in the header to
3*/AAFFFFFFFF*/".
```

The resulting ACE file header contains 05AAFFFFFFFF starting at byte 256, indicating that five user-defined bytes were added consisting of AAFFFFFFFF. HDL modifications are indicated in bold:

```
--Jump table:
..
..
constant reset_chain : integer := 60;
..
constant process_header : integer := 70;
..
..
case PC is
..
..  insert statement after number 4:
when 5 => branch_val <= process_header; sub_return <= PC + 1; ..
..
..

-- added program statements at bottom of CASE Statement
when 70 => branch_val <= get_byte; sub_return <= PC + 1;-- byte 256 of
header is string
length -- string length is thrown away in this example
when 71 => branch_val <= get_byte; sub_return <= PC + 1;
when 74 => if data_in = "10101010" then example_reg <= "11"; else
example_reg <= "00"; end if; branch_val <= sub_return;
```

Note: Each CASE statement after number 5 must be renumbered to correct for the inserted statement. The Jump Table must also be changed to match the renumbered statements.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/26/07	1.0	Initial Xilinx release.
11/16/07	1.0.1	Updated links and template.
04/07/08	1.0.2	• Corrected typo for **-wtck** under "Command Line Options," page 9. • Updated trademark notations.

Notice of Disclaimer