XAPP500 (v2.1.1) November 27, 2006

# J Drive: In-System Programming of IEEE Standard 1532 Devices

Author: Arthur Khu

## Summary

The J Drive programming engine provides immediate and direct in-system configuration (ISC) support for IEEE Standard 1532 programmable logic devices (PLDs). To configure an in-system device, the programming engine uses the configuration algorithm information from a 1532 Boundary Scan Description Language (BSDL) file to apply configuration data from the 1532 data file through the IEEE Standard 1149.1 test access port (TAP). The J Drive executable, source code, and a programming example are available in a download package from the Xilinx website. The J Drive programming engine can be used for the following Xilinx families: CoolRunner-II CPLDs, XC9500/XL/XV CPLDs, Spartan™-3 Generation FPGAs, and Virtex™-II (or later) FPGAs. For an alternative configuration solution appropriate for more advance systems of Xilinx devices, refer to [Ref 2].

## J Drive Programming Engine Advantages

The J Drive programming engine provides typical in-system configuration advantages, such as:

- Reduced device handling costs
- Reduced time to market
- Remote upgradability and testing
- Extended product life span

Moreover, because of the standardization of many aspects of programmable logic configuration, the J Drive IEEE Standard 1532 programming engine offers the following additional advantages:

- Immediate support for new generations of IEEE Standard 1532-compliant families, regardless of the device vendor
- Elimination of the need to update software implementations to support new generations (because the algorithm and configuration data are separate)
- Single-step PLD configuration without intermediate translation or compilation steps using the PLD's 1532 BSDL and ISC data files
- Independent updating of the device configuration algorithm or configuration data via the updated 1532 BSDL file or the 1532 ISC data file, respectively
- Configuration of different or multiple device targets in a scan chain using a single 1532 ISC data file
- Potential for reduced multi-device configuration times through concurrent programming techniques allowed by the IEEE Standard 1532 (thus, reducing manufacturing costs)

The J Drive programming engine makes a highly desirable base for programming 1532-compliant devices in a wide range of applications.

Previous PLDs implemented proprietary configuration architectures using proprietary combinations of algorithms and data files. Frequently, each proprietary configuration variant required a proprietary configuration implementation.

## IEEE Standard 1532

The IEEE Standard 1532 is a formal extension to the IEEE Standard 1149.1 (also known as JTAG) for PLDs. This standard defines the three items required to configure in-system programmable logic devices. The three essential items are:

- Device architectural components for configuration
- Algorithm description framework
- Configuration data file

Figure 1 illustrates configuration-specific architectural extensions to the device, algorithm extensions to the BSDL file, and a standard data file.

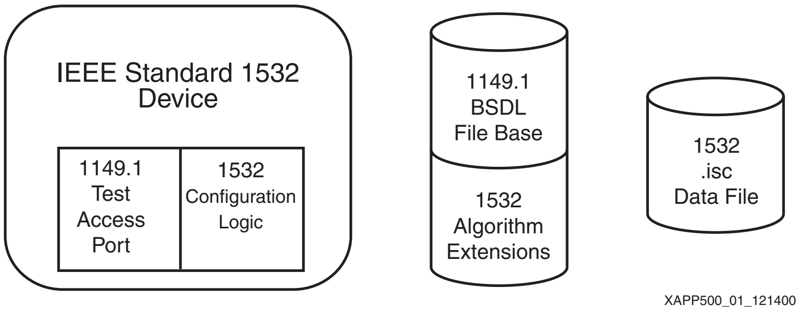

*Figure 1:* **IEEE Standard 1532 Components**

The IEEE 1149.1 TAP provides access to the device configuration logic, while the IEEE 1532 BSDL file provides the device configuration algorithm. The IEEE 1532 data file provides the configuration data. For further details, refer to the IEEE Standard 1532 publication (available for purchase) at:

http://ieee.org/

### IEEE Standard 1532 BSDL Files

The PLD manufacturer generates an IEEE Standard 1532 BSDL file for each conforming device.

The IEEE Standard 1532 defines extensions to the IEEE Standard 1149.1 BSDL files. (Refer to the IEEE Standard 1149.1 publication for details on the definition of the basic BSDL file contents.) The IEEE Standard 1532 extensions define a 1532-specific package and framework of attributes that describe the programming algorithms for the corresponding PLD. These extensions are compatible with the original IEEE Standard 1149.1 BSDL syntax.

An IEEE Standard 1532 BSDL file can be identified using the STD_1532_2002 package. The package is included in the BSDL file using the following syntax:

```
Use STD_1532_2002.all; -- 1532 BSDL Extension for ISC devices
```

The STD_1532_2002 package contains attribute definitions used to supply in-system programming information to the J Drive programming engine. Each attribute supplies the specific information given in Table 1.

*Table 1:* **IEEE Standard 1532 BSDL Attributes**

| Attribute Name | Required | Description |
|---|---|---|
| ISC_Conformance | Required | Indicates the conformance of the BSDL to a specific IEEE Std 1532 version. |
| ISC_Pin_Behavior | Required | Defines the behavior of the I/O pins during configuration: HIGHZ or CLAMP. |
| ISC_Status | Required | Specifies whether or not the standard status scheme is implemented. |
| ISC_Blank_Usercode | Required | Specifies the value of a blank (erased) USERCODE. |
| ISC_Security | Optional | Specifies the security structure (if present). |
| ISC_Flow | Required | Specifies sequences of instructions and data to apply to the TAP of a device for a reusable ISC flow. |
| ISC_Procedure | Required | Specifies the sequence of flows that perform a procedure. |
| ISC_Action | Required | Specifies the common sequences of procedures that perform each action. The user can call these actions to be performed. |
| ISC_Fixed_System_Pins | Optional | Documents the system pins whose behavior is not described by the ISC_Pin_Behavior attribute (when they exist). |
| ISC_Illegal_Exit | Optional | Specifies any instruction that clears the ISC_Enabled signal as a side effect of its operation. |
| ISC_Design_Warning | Optional | Specifies special ISC functions or options implemented in this device. |

## ISC_Flow Attribute

The ISC_Flow attribute consists of named sets of basic operations. Each named set is a flow. Each set consists of the name of the corresponding data section from the 1532 data file and the following phases:

- Initialization
- Body (repeatable sequence)
- Termination

Each phase consists of zero or more four-part tuples. The four tuple parts are as follows:

- ISC instruction to be loaded
- Update field – bit values to be shifted into TDI during the data shift
- Wait time – time to wait in the Run-Test/Idle state after the data shift
- Capture field – bit values to expect out of TDO during the following data shift

The bit values in the update field can be specified as constants, variables, or as values that are extracted from the corresponding data section in the 1532 data file.

The capture field can specify expected bit values as constants, variables, or as values to be extracted from the corresponding data section in the 1532 data file. In addition, the captured bit values can be written to a file or used in the calculation of a cyclic redundancy check (CRC). The algorithm for calculating the CRC is defined in the IEEE Standard 1532, *Data CRC Algorithm*.

### ISC_Procedure Attribute

The ISC_Procedure attribute contains definitions of procedures. The procedures are defined by a sequence of flows to be performed from the ISC_Flow attribute. The IEEE Standard 1532 predefines the functionality of the following procedures:

- Proc_read – Read device contents and write to a data file.
- Proc_verify – Verify device contents.
- Proc_program – Program device contents.
- Proc_erase – Erase the device.
- Proc_blank_check – Check if device is blank.
- Proc_enable – Enter ISC mode for the device.
- Proc_disable – Exit ISC mode for the device.
- Proc_preload - Preload the boundary-scan register with clamp values.
- Proc_error_exit – Perform when an error occurs.
- Proc_program_done – Set the DONE bit.

### ISC_Action Attribute

The named actions are common sequences of procedures, with the IEEE Standard 1532 predefined actions being the following:

- Read – Read the device contents and write them to a file.
- Verify – Stand-alone verify the device contents.
- Program – Recommended, full programming algorithm. For Xilinx devices this consists of erase, program, and verify.
- Erase – Stand-alone erase of device contents.
- Blank_check – Stand-alone blank check of device contents.

By default, procedures comprising the action are required. However, procedures can also be specified as optional or recommended.

## IEEE Standard 1532 ISC Data Files

The IEEE Standard 1532 data file is in ASCII format. The first section in the file is the header, consisting of the following information:

- Header
- Version STD_1532_2001 or STD_1532_2002
- Creation date
- Creator

The remainder of the data file contains data for the various named data sections, as described in the device's 1532 BSDL file. The flows in the BSDL file name the associated data section. Thus, each data section contains a subsection that corresponds to the initialize, body (repeat), and termination sections of the flow. Each data section can also end with a cyclic redundancy check (CRC) value.

## Generating IEEE Standard 1532 Data Files For Xilinx Devices

A design-specific 1532 ISC data file must be generated for each PLD or PROM design in the system. By default, Xilinx ISE software tools generate different kinds of implementation files for PLD designs, depending on the kind of PLD that is targeted:

- Xilinx FPGA implementations are stored in `.bit` files.
- Xilinx CPLD implementations are stored in `.jed` files.
- Xilinx PROM data is stored in `.mcs` files.

Xilinx ISE 6.1i (or later) tools can optionally directly generate 1532 (`.ISC`) data files (for example, when the "Create IEEE 1532 Data File" option in the Project Navigator Generate Programming File properties dialog box is selected). For detailed instructions regarding the generation of IEEE 1532 data files, see the Xilinx ISE software manual (http://www.xilinx.com/support/software_manuals.htm).

## J Drive IEEE Standard 1532 Programming Engine Overview

The boundary-scan interface allows the J Drive programming engine to quickly program any IEEE Standard 1532 devices in a given scan chain with a minimum set of support files.

The IEEE Standard 1532 defines the configuration architecture, BSDL algorithm extensions, and configuration data file format for individual devices. The J Drive command file defines the composition of the JTAG boundary-scan chain and the IEEE 1532 actions to be performed on target devices within the scan chain. The J Drive IEEE Standard 1532 programming engine configures 1532 devices in-system directly using the 1532 BSDL and 1532 ISC data files. See Figure 2.

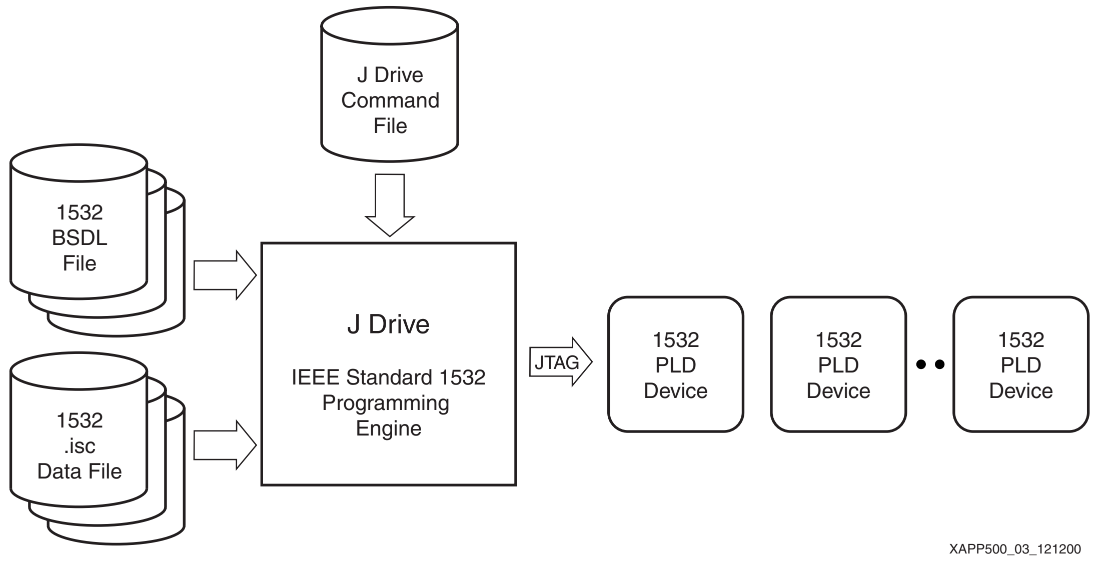

*Figure 2:* **J Drive Configures IEEE 1532 Devices In-System**

### J Drive Command File

The J Drive command file declares the order of devices in the boundary-scan chain and should have a `.cmd` extension. The device type is specified via the association of the IEEE Standard 1532 BSDL file to the device in the scan chain. The action to be performed is also specified for target devices in the scan chain. Non-target (BYPASSED) devices can simply be declared with the length of the device's instruction register rather than the BSDL file.

Each device declaration in the command file corresponds to a device in the scan chain. A device declaration can span multiple lines and must contain the following:

- Device name (a user-defined character string)
- Corresponding BSDL file name or the length of the instruction register
- Action to be carried out (optional) and corresponding data name (optional)
- Name of corresponding 1532 data file (optional)
- Name of data output file (optional)
- Ending semi-colon

The first device in the command file is the device closest to the final TDO output. The last device in the command file is the device closest to the initial TDI input. There must be an action defined for at least one device.

### BNF for Command File

```
<command file format> ::= <command list>
<command list> ::= <command> {<command>}
<command> ::= <device name> <bsdl stmt>; | <device name> <instruction
register stmt>;
<bsdl stmt> ::= -b"<bsdl file name>" [<action stmt> [<data input file stmt>
[<data output file stmt> [<data output format>]]]
<action stmt> ::= -a"<action name>" | -a"<action name> ( <data name> )"
<data input file stmt> ::= -d"<data input file name>"
<data output file stmt> ::= -o"<data output file name>"
<instruction register stmt> ::= -i <instruction length>
<data output format> ::= -data_isc
```

Example:

```
D3_CFGMEM -b"xc18v04_pc44_1532.bsd" -a"program(array)" -d"promdata.isc";
D2 -i8;
D1 -i5;
```

This command file reflects a scan chain containing three devices. D1 is the first device in the scan chain to receive the initial TDI input. D1 has an instruction register length of five bits. D2 is the second device in the scan chain and has an instruction register length of eight bits. D3_CFGMEM is the last device in the scan chain. D3_CFGMEM is programmed using the array data from the promdata.isc file.

## Using the J Drive Engine to Program an IEEE Standard 1532 Device

The J Drive engine is provided as reference C code for implementations of a IEEE Std 1532 programming solution in systems with embedded microprocessors. Given the appropriate 1532 BSDL files, 1532 data files, and command file, an embedded J Drive solution can perform in-system programming of devices through their TAPs.

To demonstrate the operation of the J Drive engine with the 1532 BSDL, 1532 data, and command files, a limited, pre-compiled example of the J Drive engine is provided with the reference C code. The example executable works only on Windows 2000/XP systems and only with the Xilinx Parallel Cables.

***Note:*** For the example J Drive executable to run correctly, Xilinx ISE software 6.1i [or later] must be installed with the Parallel Cable driver on the Windows PC.

To run the example J Drive executable, open a command window and type "JDrive" with the required and optional fields shown in Table 2.

After the program starts, the syntax of the command file is checked. (If an error is found, the program terminates with an appropriate message). All BSDL files are then checked for syntax and semantics. Should a syntax error be encountered, the program terminates with a corresponding message. The error message contains the file name and the line number where the error occurred.

After the syntax of the BSDL file has been found to be correct, the semantics of the file are checked. Errors in semantics are listed together. Each error message contains the line in the BSDL file at which the error was found together with a more complete description of the problem. The 1532 data files are similarly checked for syntax and semantics.

After all files have been successfully checked, the execution proper can begin. Should the J Drive engine encounter an error at this point, the program terminates with a corresponding message. The J Drive engine reports a “successful” message when the program completes successfully.

***Note:*** Xilinx recommends using iMPACT 6.1i (or later) to configure IEEE Std 1532 devices in-system, using an IEEE Std 1532 flow from operating system platforms and cables supported by the Xilinx ISE software. iMPACT can be used in either standard or interactive mode, or in a scripted batch mode. See the Xilinx ISE software manuals for further details.

*Table 2:* **Required and Optional Fields**

| Parameter | Description |
|---|---|
| **Required Fields** | |
| `-i <command file>` | The command file describes the devices in the boundary scan chain, the operations to perform on these devices, and the associated files for the specified operation. |
| **Optional Fields** | |
| `-bsdlwarnings` | Copy DESIGN_WARNINGs and ISC_DESIGN_WARNINGs from BSDL file to display. |
| `-checkinpfiles` | Check command file syntax, BSDL files, and associated ISC data files, if specified, then exit. |
| `-l <log file>` | Log or print any messages displayed by the program to the log file. |
| `-m [sequential \| concurrent]` | Specify sequential or concurrent processing mode. Defaults to sequential if not specified. |
| `-maxbiterr N` | Specify the maximum number of bit errors to display (default = 50). |
| `-o <output file>` | The output file is used for any error or debug messages. |
| `-skip_error_proc` | Use BYPASS instead of PROC_ERROR_EXIT procedure when errors occur. |
| `-skip_exit_on_error` | Continue processing flows, procedures, or actions for other devices, even if an error occurs on a device with a flow when the EXIT_ON_ERROR attribute is defined. JDrive puts the device that produced the error in BYPASS mode until all other devices have been processed, after which JDrive runs the PROC_ERROR_EXIT procedure of the failing device (unless -skip_error_proc is specified) before terminating. |

## Troubleshooting J Drive Errors

Possible sources of error and course of action that can help identify problem the source:

- **Problems initializing the cable hardware**, for example, missing drivers.

  Check that iMPACT is installed properly on PC.
- **Incorrect actions or data names in the command file**, for example, the corresponding action is not defined in the ISC extension of the BSDL file.

  Check BSDL file for the correct action name, match it with command file action name.
- **Incorrect scan chain specification in the command file**.

  In iMPACT run "Automatically connect to a cable and identify Boundary-Scan chain" then check that the command file matches the scan chain generated by iMPACT.
- **A bad net on the board.**

  To validate a broken JTAG signal connection on the board use iMPACT to perform an operation on JTAG chain.
- **Comparison errors during the programming,** for example, captured values do not match the expected values.

  Run iMPACT to check device ID, perform ID code looping and program the device.

The following courses of action can help identify additional problem sources:

1. Execute just the verify (IDCODE) action. The IDCODE test is the simplest test that can be performed. A failed IDCODE indicates a gross problem in either the scan chain description or the physical connections between the J Drive and the device.
2. Separately, execute the erase, blank_check, program, and verify actions (if available) in order. Confirm each operation using a known-good programming source, such as iMPACT with a Xilinx cable. This sequence of actions can help to identify which operation is failing.
3. Use the Xilinx iMPACT software with a Xilinx download cable to identify and program the devices on the board. If iMPACT reports an error, then there is a gross problem with the connectivity on the board.
4. If a custom implementation of J Drive fails to work, try to perform the action with a Xilinx cable, using the IEEE Std 1532 flow supported in the Xilinx iMPACT software. iMPACT automatically uses its IEEE Std 1532 flow when a 1532 data (.ISC) file is assigned to a device in the iMPACT Boundary-Scan window. See the Xilinx ISE software manuals for details regarding the iMPACT 1532 flow. Use iMPACT and cable signals as a comparative reference for the applied 1532 sequence.

## Porting J Drive to an Embedded Microprocessor

Xilinx provides the source code for the J Drive programming engine. The high-level block architecture of the J Drive programming engine is shown in Figure 3. The J Drive programming engine consists of several parsers for the input files, an interpreter for the BSDL algorithm information, a vector composer/sequencer, and a back-end 1149. TAP controller module.

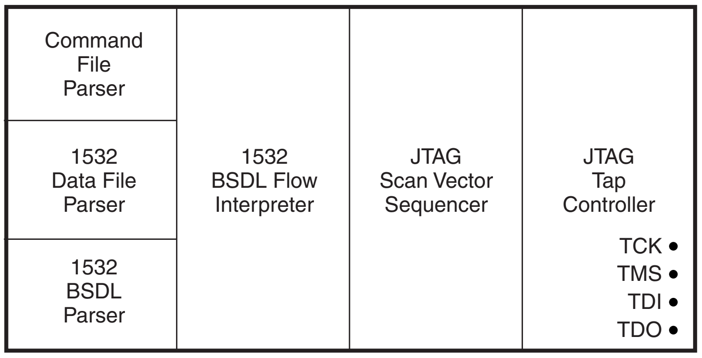

*Figure 3:* **J Drive High-Level Block Architecture**

When porting a J Drive to an embedded microprocessor or other environment, most of the code remains unchanged. Only the back-end TAP controller module needs careful modification. The given source code communicates using the Xilinx Parallel Cable connection to control the TAP. A ported version of the J Drive needs to substitute these communication protocols for the appropriate protocols used on the target platform.

When porting the J Drive programming engine to another target environment, the following issues must be taken into consideration:

- The speed at which the TAP signals can be toggled significantly affects the programming times for the XC18V00 PROMs and Virtex/Virtex-E FPGAs, because both require a significant amount of data to be transmitted through the serial TAP.
- The TCK, TMS, and TDI output signals must be controllable.
- The TDO input signal must be readable.
- The ported implementation must accurately tune the timing procedures in the code for the target platform such that the implementation waits for at least the requested time (see the waitTime function implementation in the `ports.cpp` file). More accurate timing results in better programming performance.

A porting of the J Drive engine basically involves customizing the "hardware" module shown in Figure 4, page 9. This hardware module must realize the following functions:

- Control TCK, TMS, and TDI signals going to devices in the boundary-scan chain, and read the TDO signal from the boundary-scan chain.
- Control wait times (waitTime()).

Changes can also be made to a higher-level function called jtagShift() in `xchkfunc.cpp`, which shifts data to the TAP port.

***Note:*** The example J Drive executable uses the Jungo WinDriver as the parallel port driver. More information about the Jungo WinDriver can be found at:

http://www.jungo.com.

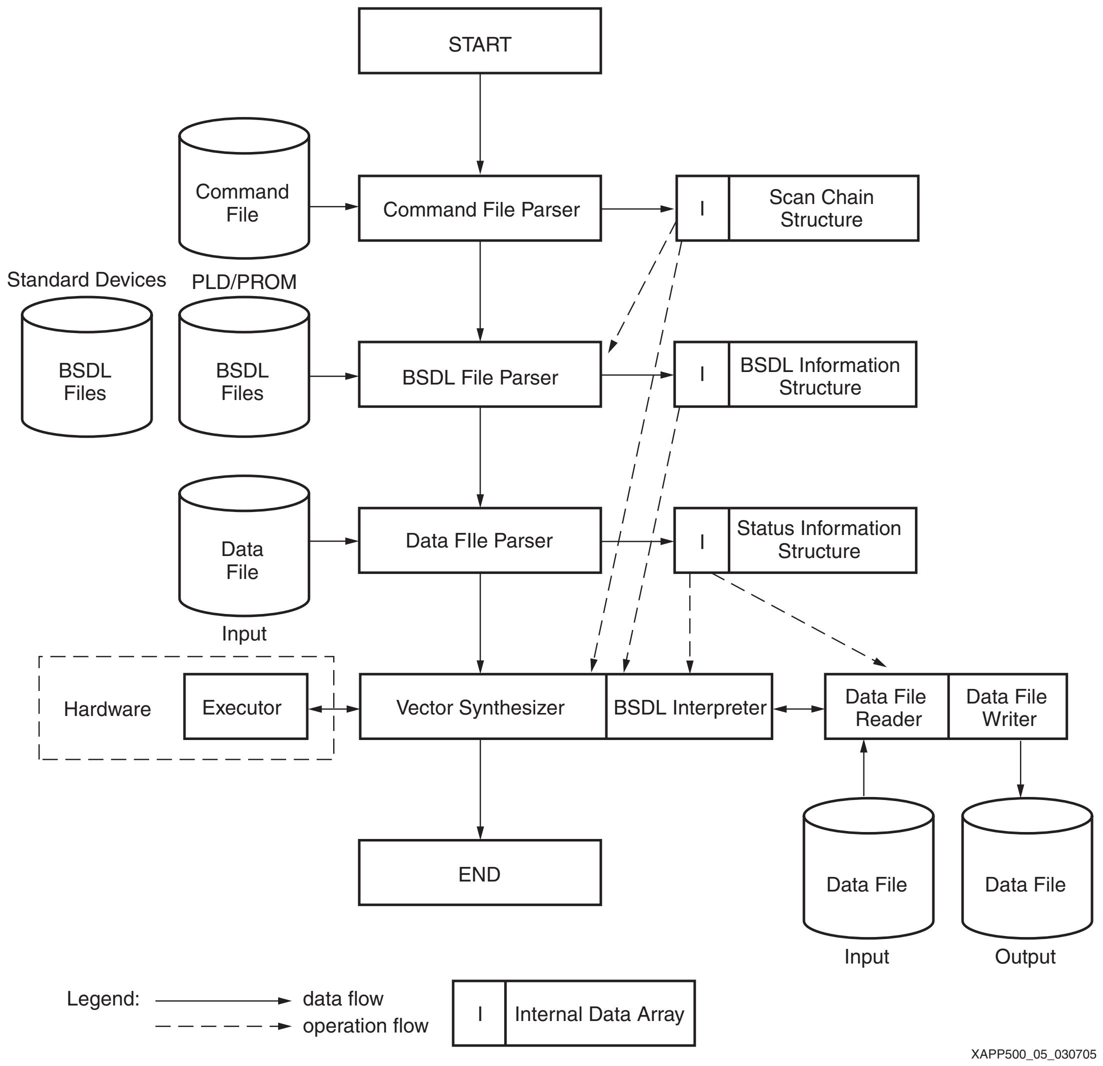

*Figure 4:* **J Drive High-Level Source Module and Flow Diagram**

## Links

The latest version of the XAPP500 associated files are available for downloading at the following:

- J Drive software and source code:

  http://www.xilinx.com/isp/jdrivedownload.htm
- Xilinx IEEE 1532 BSDL files:

  http://www.xilinx.com/xlnx/xil_sw_updates_home.jsp?update=bsdl

## References

The following reference material exists:

1. Several resources describing IEEE Standard 1149.1 (JTAG) are available from the Xilinx Configuration Solutions “JTAG and ISP” web page:

   http://www.xilinx.com/xlnx/xil_prodcat_product.jsp?title=isp_standards_specs
2. XAPP058, *Xilinx In-System Programming Using Embedded Microcontrolle*r, contains compact reference C code for programming Xilinx devices using JTAG from an embedded microcontroller.
3. *IEEE Std 1532-2002*, The Institute of Electrical and Electronics Engineers, Inc., New York, NY, 2003.
4. Neil G. Jacobson. 2004. *The In-System Configuration Handbook: A Designer's Guide to ISC.* Hingham, MA: Kluwer Academic Publishers.

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 01/15/01 | 1.0 | Initial Xilinx release. |
| 01/17/01 | 1.1 | Added software link. |
| 03/07/05 | 2.0 | Updated text and reference code for IEEE Std 1532-2002 revision. |
| 08/23/06 | 2.1 | • Updated document format.<br>• Updated “Summary,” page 1.<br>• Updated links on page 2 and page 5.<br>• Updated “Troubleshooting J Drive Errors,” page 7.<br>• Updated “Links,” page 9. |
| 11/27/06 | 2.1.1 | Updated trademark status of J Drive. |