XAPP500 (v1.1) January 17, 2001

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

Author: Randal Kuramoto

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: XC18V00, Virtex™, and Virtex-E.

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 data from the engine are separate)
- Single-step PLD configuration without intermediate translation or compilation steps using the PLD's 1532 BSDL and data files
- Independent updating of the device configuration algorithm or configuration data via the updated 1532 BSDL file or the 1532 data file, respectively
- Configuration of different or multiple device targets in a scan chain using a single 1532 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.

IEEE Standard 1532

The IEEE Standard 1532 is a formal extension to the IEEE Standard 1149.1b-1994 (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.

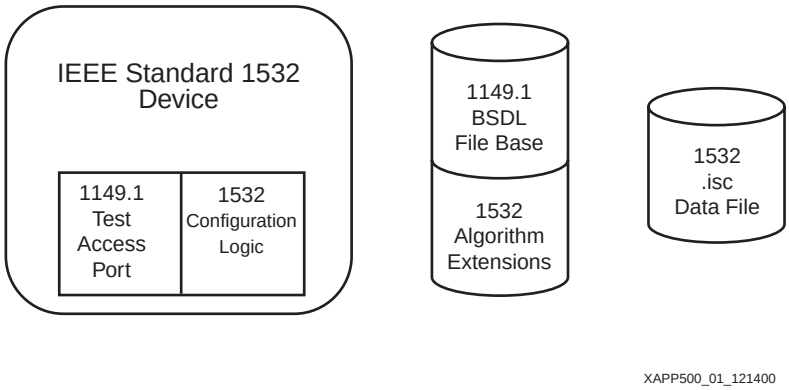

Figure 1: **IEEE Standard 1532 Components**

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. Refer to the IEEE Standard 1532 publication for further details at: **http://grouper.ieee.org/groups/1532/.**

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 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 1149.1b-1994 BSDL files. (Refer to the IEEE Standard 1149.1b-1994 publication for details on the original definition of the 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 1149.1b-1994 BSDL syntax.

An IEEE Standard 1532 BSDL file can be identified using the STD_1532_2000 package. The package is included in the BSDL file using the following syntax:

```
Use STD_1532_2000.all; -- 1532 BSDL Extension for ISC devices
```

The STD_1532_2000 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_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	Required	Specifies the security structure (if present).
ISC_Flow	Required	Specifies sequences of instructions and data to apply to the JTAG port of a device.

Table 1: **IEEE Standard 1532 BSDL Attributes *(Continued)***

Attribute Name	Required	Description
ISC_Procedure	Required	Specifies the sequence of flows that perform a procedure.
ISC_Action	Required	Specifies the common sequences of procedures that perform some action.
ISC_Illegal_Exit	Optional	Specifies the action to take in the event of an error during in-system programming.
ISC_Design_Warning	Optional	Specifies design warnings.

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 (Repeat)
- 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 JTAG 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 contents.
- Proc_program – Program contents.
- Proc_erase – Erase the device.
- Proc_blank_check – Check if device is blank.
- Proc_enable – Enable the ISC mode for the device.
- Proc_disable – Exit the ISC mode for the device.
- 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. Xilinx devices erase, program, and verify.
- Erase – Stand-alone erase.
- Blank_check – Stand-alone blank check.

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

IEEE Standard 1532 Data Files

The IEEE Standard 1532 data file is a text file format.

The first section in the file is the header, consisting of the following information:

- Header
- Version STD_1532_2000
- 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 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.

The predefined data sections include:

- Array
- Usercode
- Security
- Idcode

Generating IEEE Standard 1532 Data Files For Xilinx Devices

A design-specific 1532 data file must be generated for each PLD design in the system.

Xilinx 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, and Xilinx PROM data is stored in .**mcs** files.

The Xilinx xgen1532 utility generates an IEEE Standard 1532 data file from either the .**bit** file, .**jed** file, or .**mcs** file.

The syntax for the Xilinx xgen1532 command-line utility depends on the kind of input file.

For XC18V00 PROMs, the syntax is:

```
xgen1532 –m <mcs_file_name.mcs> -a xc1800 –p <device_pkg> –o
1532_data_file_name.isc
```

For Virtex/E FPGAs, the syntax is:

```
xgen1532 –b <bit_file_name.bit> -a virtex –o 1532_data_file_name.isc
```

The following examples are command-lines for different kinds of devices:

```
xgen1532 -m xc18v256.mcs -a xc1800 -p xc18v256_so20 -o xc18v256.isc
xgen1532 -b xcv50_pq240.bit -a virtex -o xcv50_pq240.isc
```

Figure 2 shows the xgen1532 utility generation of IEEE Standard 1532 data (.isc) files from FPGA (.bit), CPLD (.jed), or PROM (.mcs) files.

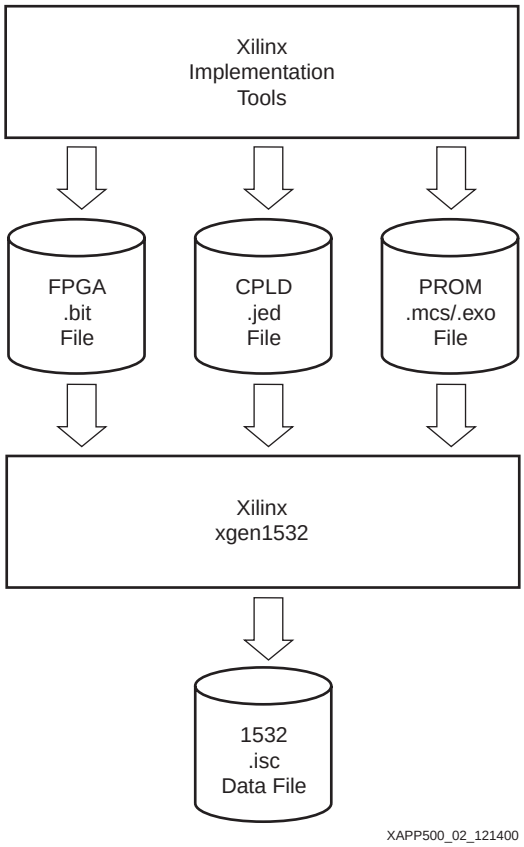

Figure 2: **Generation of IEEE Standard 1532 Data Files**

J Drive IEEE Standard 1532 Programming Engine Overview

The simple JTAG-based interface allows the J Drive programming engine to quickly program any of the 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 architecture 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 data files. See Figure 3.

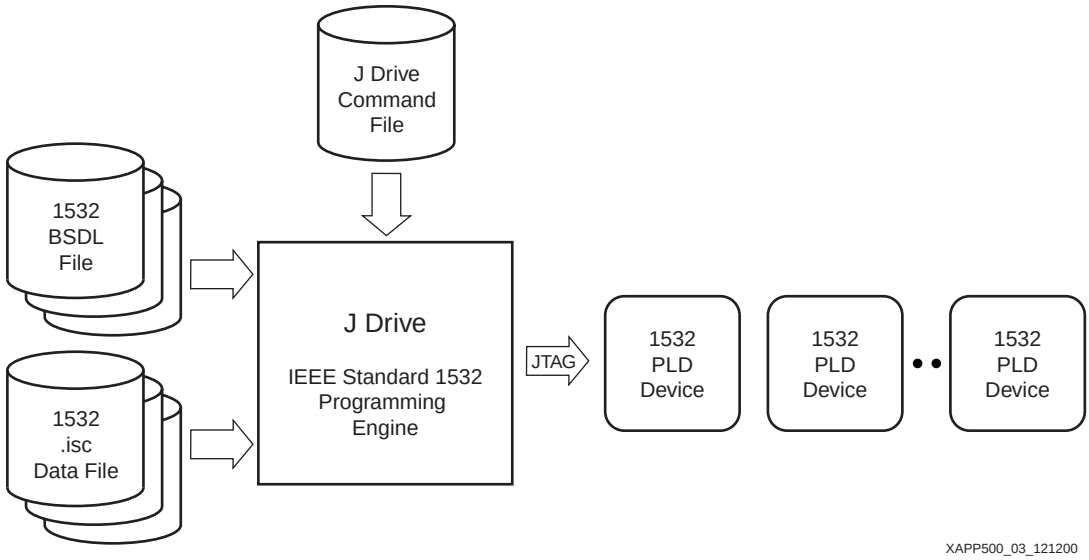

Figure 3: **J Drive Configures IEEE 1532 Devices In-System**

J Drive Command File

The J Drive command file declares the order of devices in the JTAG scan chain and should have a .**cmd** extension. The device kind 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 line in the command file corresponds to a device in the scan chain. Each line should contain the command for the device, and the command for each device must include the:

- Device name
- 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>]]
<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>
```

Example:

```
D3 -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 is the last device in the scan chain. D3 is programmed using the array data from the promdata.isc file.

Notes:

1. The version 1.00 implementation of the J Drive engine can apply only one action per command file.

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

The 1532 J Drive programmer can be started at the Windows MS-DOS command prompt with the command "JDrive." A prebuilt executable is provided in the standard J Drive download package that runs on the Win32 platforms and controls the JTAG signals of the Xilinx Parallel Cable III hardware.

Syntax

```
JDrive  [-adfhsv] [-l [log-file-name]] [--addclocks] [--debug]
        [--fail_ignore] [--help] [--logfile]
[log-file-name]] [--status] [--virtual] command_file.cmd
```

Table 2 lists the syntax parameter descriptions.

Table 2: **Parameter Descriptions**

Parameter	Description
-a/--addclocks	Sets the flag ADDCLOCKS. If set, a burst is always executed in a stable controller status (IRPAUSE, DRPAUSE, IRSHIFT, DRSHIFT, or RUN/IDLE). If the flag is not set, then the scan operations are executed according to the standard form.
-d/--debug	Detailed execution status information (e.g., vector synthesizer/interpreter debug information) is printed. This options runs J Drive much slower! For that, the compiler switch DEBUG is necessary!
-f/--fail_ignore	If a compare fails between the expected and captured bits, the execution is not aborted if parameter -f / --fail_ignore is set.
-h/--help	Print help information on the console.
-l/--logfile	Print all messages and information to the log file.
-s/--status	Print parsing and execution status information.
-v/--virtual	The programmer is executed without toggling the physical TAP port signals.
Command__file.cmd	The name of the command file. This is required.

Example command-lines are shown below:

```
JDrive  -a example.cmd
JDrive  -s -d -l example.log example.cmd (for log file output)
JDrive  -v example.cmd (for testing the files)
```

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 was encountered. Also, the contents of the line containing the error are displayed up to the point at which the error was found.

After the syntax of the BSDL file has been found to be correct, the semantics of the file are checked. Mistakes in semantics are listed together. Each error message contains the line in the BSDL file at which the error was found together with information about the kind of mistake. 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.

Troubleshooting J Drive Errors

Possible sources of error include:

- Problems initializing the hardware (e.g., missing drivers)
- Hardware problems during program execution
- Incorrect actions or data names in the command file, e.g., the corresponding action is not defined in the ISC extension of the BSDL file
- Operations on variables that were not yet initialized
- Incorrect data in the 1532 file, e.g., too few hex strings in the data block or hex strings with unused bits
- Incorrect scan chain specification in the command file
- A bad device
- A bad net on the board
- Comparison errors during the programming, e.g., captured values do not match the expected values. (If the procedure “proc_error_exit“ is defined, then it is carried out before program termination.)

The following courses of action can help identify 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. This sequence of actions can help to identify which operation is failing.
3. Use the Xilinx JTAG Programmer software with a Xilinx download cable to identify and program the devices on the board. If JTAG Programmer 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, then try the prebuilt version from a Windows PC to program the devices using a Xilinx Parallel Cable III connection.

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 4. 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 1149.1JTAG TAP controller module.

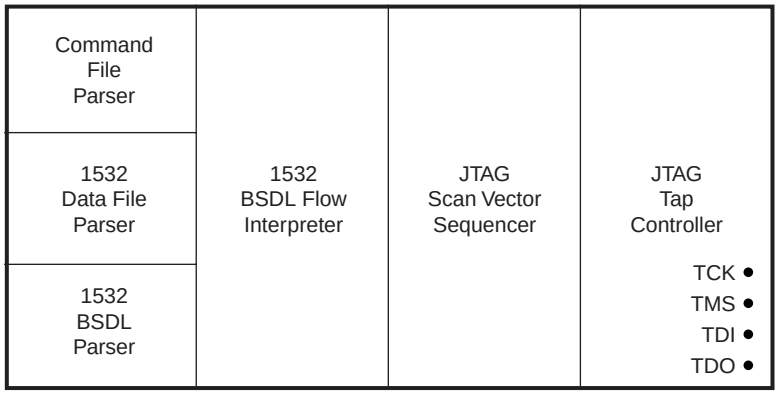

Figure 4: **J Drive High-Level Block Architecture**

When porting a J Drive to an embedded microprocessor or other environment, most of the code remains unchanged. Only the back-end JTAG TAP controller module needs careful modification. The given source code communicates using the Xilinx Parallel Cable III connection to control the JTAG TAP. A ported version of the J Drive needs to substitute these communication protocols for the appropriate protocols available 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 JTAG 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 JTAG TAP.
- The TCK, TMS, and TDI output signals must be controllable.
- The TDO input signal must be capturable.
- The ported implementation must accurately tune the timing procedures in the code for the target platform within ±25%. The more accurate the timing, the better the programming results.

A porting of the J Drive engine basically involves customizing the hardware module shown in Figure 5. The hardware module consists of the following source code files:

bscanio.h – Contains the interface function declarations for the hardware module.

bscanio.c – Contains the implementations of the interface functions for the hardware module.

Primarily, the **bscanio** functions control access to the values of the external TCK, TMS, TDI, and TDO device signals. Table 2 lists the functions in the bscanio.c file that must be customized to perform the described function on the target platform.

Table 3: **Hardware Interface Functions of the Bscanio Module**

Function	Description
BscanCtrlInit	This function is called once to initialize the hardware that controls the access to the TCK, TMS, TDI, and TDO signals.
BscanCtrlClose	This function is called once to release the hardware that controls the access to the TCK, TMS, TDI, and TDO signals.
BscanCtrlCheckCable	This function is called once after BscanCtrlInit to check the status of the hardware controlling the access to the TCK, TMS, TDI, and TDO signals.
BscanCtrlReadByte	This function reads a byte that contains the value of the TDO signal.
BscanCtrlWriteByte	This function writes a byte that contains the new values of the TCK, TMS, and TDI signals.
BscanCtrlBurstExecute	This function applies a burst of new values to TCK, TMS, and TDI. The function also reads and saves the values of TDO as the burst of TCK, TMS, and TDI values are being written.
WriteClocks	This function applies the given number of Low-High-Low pulses to the TCK signal.
InitDelay	This function is called once to calibrate the delay timer.
DelaySeconds	This function is called when the J Drive engine must wait for the specified number of seconds.
GotoNextState	This function is called when the J Drive engine must apply appropriate TMS and TCK signal values to transition the target device's JTAG state machine to the named state.

The original implementations of the **bscanio** functions in the given source code control Xilinx Parallel Cable III access to the TCK, TMS, TDI, and TDO signals through a Windows PC parallel port driver. The parallel port interface functions and implementation are contained in the following source code files:

bscandrv.h – Parallel port interface functions

bscandrv.c – Parallel port interface implementation

Notes:

1. The original J Drive implementation uses the Jungo WinDriver as the parallel port driver. More information about the Jungo WinDriver can be found at: **http://www.jungo.com**.

Additional Information

Additional information can be found at:

- 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**
- XAPP058: "Xilinx In-System Programming Using Embedded Microcontroller" contains related information about programming devices using JTAG from an embedded microcontroller and is available at: **http://www.xilinx.com/apps/xappsumm.htm#xapp058**

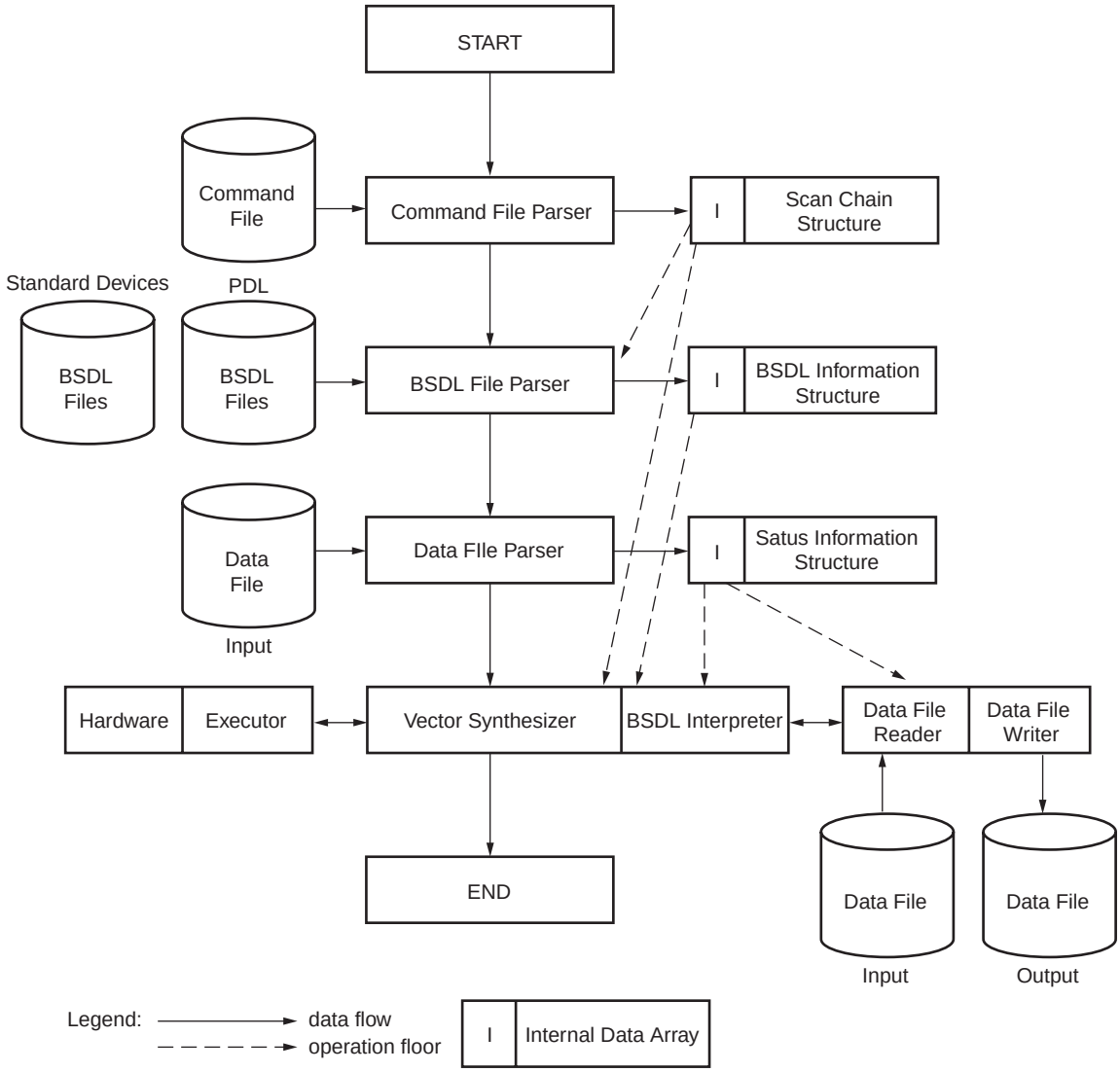

Figure 5: **J Drive High-Level Source Module and Flow Diagram**

Software Links

Two links are provided to the following required files:

1. J Drive software and source code

http://www.xilinx.com/isp/jdrivedownload.htm

2. Xilinx IEEE 1532 BSDL files

http://www.xilinx.com/isp/1532download.htm

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.