The JRunner Software Driver: An Embedded Solution for PLD JTAG Configuration

May 2006, version 1.0 Application Note 414

Introduction

The JRunner™ software driver is developed to configure Altera® FPGA devices in JTAG mode through the ByteBlaster II or ByteBlasterMV download cables for embedded configurations. Source code for the driver is provided to enable you to customize the I/O control routines for your system.

The JRunner driver supports the configuration file in a Raw Binary File (**.rbf**) format generated by the Altera Quartus® II software. The input file to the JRunner driver is in the Chain Description File (**.cdf**) format.

The JRunner software was developed and tested on the Windows platform (NT, 2000, and XP).

How JRunner Software Works

JRunner software utilizes JTAG interface to configure Altera devices in embedded configurations. This software consists of controller and parsers to process the information required from the input files such as device information and configuration data.

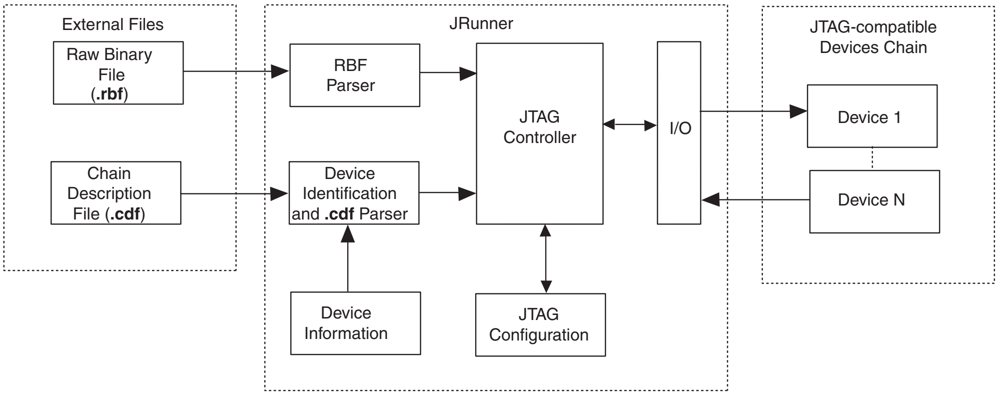
This software requires the following two external files:

- A Raw Binary File containing information on configuration images.
- A Chain Description File containing information on the Altera PLDs.

Block Diagram

Figure 1 shows the JRunner software block diagram, its interfaces to external files, and the device chain. The JTAG controller manages the data processing as well as the JTAG configuration process.

Figure 1. JRunner Block Diagram And Interfaces



Source Files

The JRunner source code is written in ANSI C and is divided into modules that reside in separate files. Table 1 describes the JRunner source code files.

Table 1. Source Files

File	Description	Platform Independent
jrunner.c	Contains the `main ()` function. It manages the processing of the programming input file, invokes the configuration process, and handles configuration errors.	Yes
jb_const.h	Contains program and user-defined variables and constants.	Yes
jb_device.h	Contains Altera device information and requires updates for new devices.	Yes
jb_jtag.c **jb_jtag.h**	Handles the JTAG instructions and keep track of the JTAG state machine (JSM).	Yes
jb_io.c **jb_io.h**	Handles the I/O control functions, file processing functions and string processing functions, and may be customized to work with your system.	No

☞ The source codes are available for download from the Altera website (**www.altera.com)**. For the complete list of supported Altera devices, refer to the source codes README file.

Directory Structure

The downloaded JRunner driver is stored in the structure as shown in Table 2.

Table 2. Directory Structure

Folders in JRunner	Files Available	Description
bin	**JRunner_exe**, **note.txt**	Executable file for JRunner driver
doc	**readme.txt**	JRunner documentation
source	**jrunner.c**, **jb_const.h**, **jb_device.h**, **jb_jtag.c**, **jb_jtag.h**, **jb_io.c**, **jb_io.h**	Source files

Input Files

The JRunner software driver supports the Raw Binary File programming source file. In addition, this driver requires a Chain Description File generated by the Quartus II software. The Chain Description File contains information on the devices in the JTAG chain. Modify the Chain Description File, which is generated by Quartus II software, before using it with the JRunner drivers. Modify the file as follows:

1. Open the Chain Description File in text format.

2. Replace the SRAM Object File (**.sof**) extension with the specified file name (<filename>.sof). Use the Raw Binary File (**.rbf)** extension. The file name is usually in the fifth or sixth line of the Chain Description File.

The JRunner software only supports configuration of Altera devices.

For more information on creating a Chain Description File, refer to the Quartus II online help.

The JRunner driver processes the action code for each device in the Chain Description File. The supported action codes are as follows:

- CFG and IGN for PROGRAM
- BYPASS JTAG instructions

How To Use JRunner

After modifying the Chain Description File, type the following command line at the Windows command prompt to configure the device:

```
JRunner <design file name>.cdf
```

Important Parameters & Functions

This software is developed to ease users in configuring Altera devices in embedded systems. JTAG configuration pins such as `TDI`, `TMS`, `TCK` and `TDO` are needed in order to use this software.

I/O Pin Assignment

Reading and writing of data to and from the I/O port registers on non-Windows NT platforms requires parallel port architecture mapping. This mapping reduces the number of required source code modifications. Table 3 shows the assignments of the JTAG configuration pins to the parallel port registers.

Table 3. Pin Assignment of the JTAG Configuration Signals to the Parallel Port Registers

Bit	7	6	5	4	3	2	1	0
Port 0 *(1)*	-	TDI	-	-	-	-	TMS	TCK
Port 1 *(1)*	TDO# *(2)*		-	-	-	-	-	-
Port 2 *(1)*	-	-	-	-	-	-	-	-

Notes to Table 3:

(1) The port refers to the index from the base address of the parallel port. For example, 0x378.

(2) Inverted signal.

Program and User-defined Constants

The source code has program and user-defined constants. You should set the values for the user-defined constants if the driver needs to be modified. Do not change the program constants. Table 4 summarizes the program and user-defined constants.

Table 4. Program and User-defined Constants

Constant	Type	File	Description
`WINDOWS`	Program	**jb_io.h**	Designates the Windows NT operating system.
`EMBEDDED`	Program	**jb_io.h**	Designates an embedded system or other operating system.
`SIG_TCK`	Program	**jb_const.h**	`TCK` signal (Port 0, Bit 0).
`SIG_TMS`	Program	**jb_io.h**	`TMS` signal (Port 0, Bit 1.)
`SIG_TDI`	Program	**jb_io.h**	`TDI` signal (Port 0, Bit 6).
`SIG_TDO`	Program	**jb_const.h**	`TDO` signal (Port 1, Bit 7).
`CDF_IDCODE_LEN`	Program	**jb_const.h**	The maximum characters allocated for the part name
`CDF_PNAME_LEN`	Program	**jb_const.h**	The maximum characters allocated for the Chain Description File path.
`CDF_PATH_LENGTH`	Program	**jb_const.h**	The maximum characters allocated for the Chain Description File name.
`MAX_DEVICE_ALLOW`	User-defined	**jb_const.h**	The maximum number of devices in the chain.
`MAX_CONFIG_COUNT`	User-defined	**jb_const.h**	The maximum number of auto-reconfiguration attempts allowed when the program detects an error.
`INIT_COUNT`	User-defined	**jb_const.h**	The number of clock cycles to toggle after the configuration is done to initialize the device. Each device family requires a specific number of clock cycles.

Global Variables

Table 5 summarizes the global variables used when reading from or writing to the I/O ports. Map the I/O ports of your system to these global variables.

Table 5. Global Variables

Global Variable	Type	Description
`sig_port_maskbit[W][X]`	2-dimensional integer array	Variable holding a signal's port number and bit position in the port registers. *(1) (2)* `W`= 0 refers to `SIG_TCK` `W` = 1 refers to SIG_TMS `W` = 2 refers to `SIG_TDI` `W` = 3 refers to `SIG_TDO`. `X`=0 refers to the signal's port number. For example, the signal `SIG_TCK` falls into port 0. `X`=1 refers to the signal's bit position. For example, the signal `SIG_TCK` is in bit 0 of port 0.
`port_data[Y]`	Integer array	The current value of each port. This value updates each time a write is done to the ports. *(1)*

Notes to Table 5:

(1) The port number refers to the index from the base address of the parallel port. For example, 0x378.

(2) The signal refers to these signals: `SIG_TCK`, `SIG_TMS`, `SIG_TDI`, and `SIG_TDO`.

I/O Routines

Table 6 describes the parameters and the return value of some of the functions in the source code. Only functions declared in the jb_io.c are discussed, because you must customize these functions in order to use the JRunner software on platforms other than Windows NT. These functions contain the I/O control routines.

Table 6. I/O Control Functions

Function	Parameters	Return Value	Description
`readport`	int port	integer	Reads the value of the port and returns it. Only the least significant byte contains valid data. (1)
`writeport`	int port int data int buffer_enable	none	Writes the data to the port. Data of the integer type is passed to the function. Only the least significant byte contains valid data. Each bit of the least significant byte represents the signal in the port, as discussed in Table 1. *(1)* The functions in **jrunner.c** that call the `writeport` function organize the bits prior to sending them to the `writeport` function. Only the specific bits are changed as needed before passing them to the `writeport` function as data. The data is written to the parallel port immediately if `buffer_enable`=0. If `buffer_enable`=1, the operations are stored in the buffer and flushed once the number of operations reaches 256.

Note to Table 6:

(1) The port refers to the index from the base address of the parallel port. For example, 0x378.

JTAG Configuration Flow Using JRunner

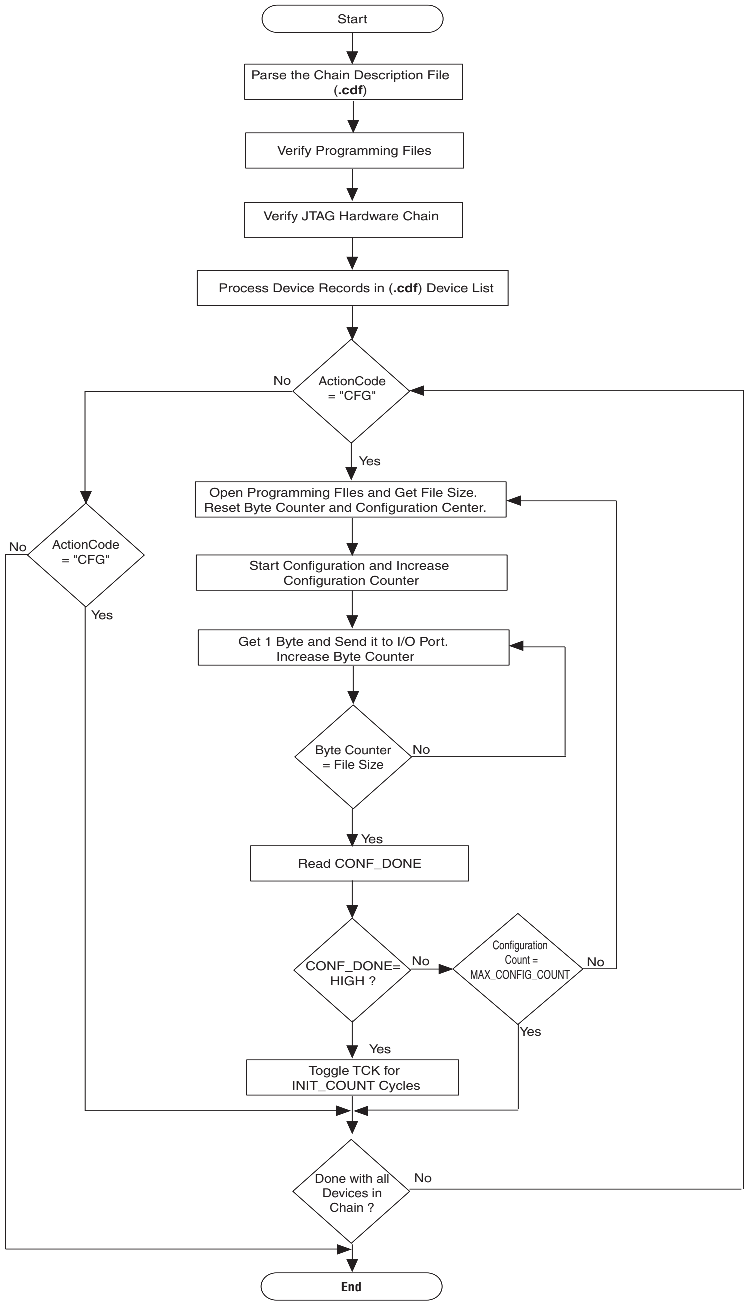
JRunner driver is developed using JTAG configuration state machines with the addition of several parameters and constants to represent the number of devices in JTAG-compatible devices chain and related information.

Program Flow

Figure 2 illustrates the program flow of the JRunner software driver. The `MAX_DEVICE_ALLOW`, `MAX_CONFIG_COUNT`, and `INIT_COUNT` constants determine the flow of the configuration process.

Refer to Table 4 for more information on these constants.

Figure 2. JRunner Program Flow


Start
Parse the Chain Description File (.cdf)
Verify Programming Files
Verify JTAG Hardware Chain
Process Device Records in (.cdf) Device List
ActionCode = "CFG"
No
Yes
Open Programming FIles and Get File Size. Reset Byte Counter and Configuration Center.
ActionCode = "CFG"
No
Yes
Start Configuration and Increase Configuration Counter
Get 1 Byte and Send it to I/O Port. Increase Byte Counter
Byte Counter = File Size
No
Yes
Read CONF_DONE
CONF_DONE= HIGH ?
No
Configuration Count = MAX_CONFIG_COUNT
No
Yes
Yes
Toggle TCK for INIT_COUNT Cycles
Done with all Devices in Chain ?
No
End

How To Port JRunner Driver To An Embedded Platform

As JRunner is developed to enable configuration in an embedded platform, the source codes have been written in ways to users can easily port them to platforms of their preference.

Porting the Source Code to Other Platforms or Embedded Systems

Two separate platform-dependent routines handle read and write operations in the I/O control module. The read operation reads the value of the required pin. To port the source code to other platforms or embedded systems, you must implement your I/O control routines in the existing I/O control functions, `readport` and `writeport` (see Table 6). You can implement your I/O control routines between the following compiler directives:

```
#if PORT == WINDOWS_NT
/* original source code */
#else if PORT == EMBEDDED
/* put your I/O control routines source code here */
#endif
```

Reading Data from the I/O Ports

The `readport` function accepts port as an integer parameter and returns an integer value. Your code should map or translate the port value defined in the parallel port architecture (see Table 3) to the I/O port definition of your system.

For example, when reading from port 1, your source code should read the `CONF_DONE` signal from the bit defined in Table 3. Then your code should rearrange the signal within an integer variable so that the value of `CONF_DONE` is represented in bit position 7 of the integer. This maps your system's I/O ports to the pin in the pin assignments of the parallel port architecture. By adding these lines of translation code to the **jb_io.c** file, you can avoid modifying code in the **jrunner.c** file.

Writing Data to the I/O Ports

The `writeport` function accepts three integer parameters: `port`, `data`, and `buffer_enable`. Modify the `writeport` function in the same way as you did for the `readport` function. Your code maps or translates the port value defined in the parallel port architecture (see Table 3) to the I/O port definition of your system.

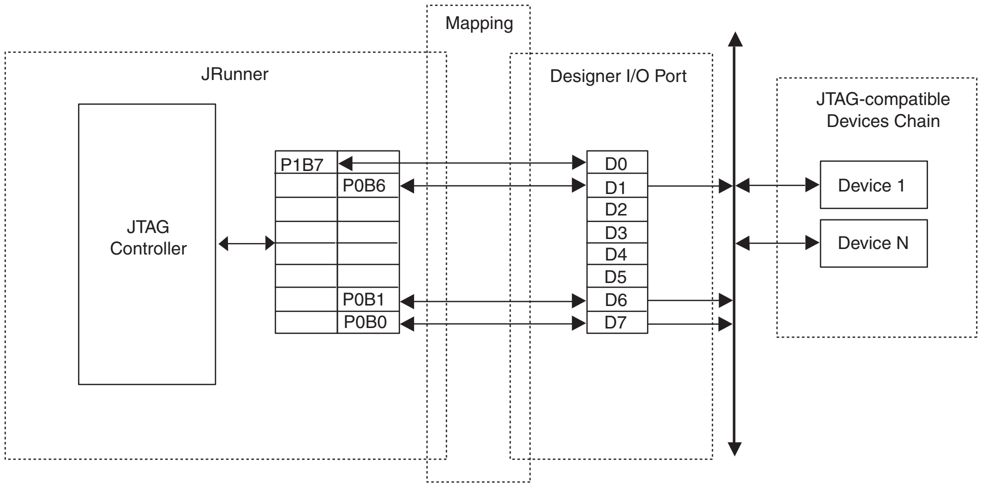
For example, when writing to port 0, your source code should identify the `TDI`, `TMS`, and `TCK` signals represented in each bit of the `data` parameter. The source code should mask the `data` variable with the `sig_port_maskbit` variable (see Table 5) to extract the value of the signal to write. To extract TDI from `data`, for instance, mask `data` with `sig_port_maskbit` [SIG_TDI][1].

Example

Figure 3 shows an embedded system holding four configuration signals in the port registers `D0`, `D1`, `D6`, and `D7` of an embedded microprocessor. When reading from the I/O port, the I/O control routine reads the values of the port registers and maps them to the particular bits in the parallel port registers (`P0` to `P2`).

When writing, the values of the signals are stored in the parallel port registers and sent to the corresponding data registers (`D0`, `D1`, `D6`, and `D7`).

Figure 3. Example of I/O Reading and Writing Mapping



Conclusion

You can easily port the JRunner JTAG embedded source code to other platforms. The JRunner software is a simple, inexpensive embedded system for JTAG configuration of Altera FPGAs.

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
literature@altera.com

I.S. EN ISO 9001