The JBlaster Software Driver: An Embedded Solution to the JTAG Configuration

Introduction

The JBlaster™ software driver is developed to configure Altera® SRAM-based programmable logic devices (PLDs) in JTAG mode for embedded configurations. You can customize the modular source code's I/O control routines (provided as separate files) for your system. The JBlaster driver supports the raw binary file (**.rbf**) format generated by the Altera Quartus® II software. The input file to the JBlaster driver is in chain description file (**.cdf**) format. The JBlaster software was developed and tested on the Windows NT platform. The Windows NT driver's binary file size is about 48 Kbytes.

This document explains how the JBlaster software works, the important parameters and functions, the JTAG configuration flow, and how to port the driver to an embedded platform.

I/O Pin Assignment

The reading and writing of the data to and from the I/O port registers on non-Windows NT platforms requires the parallel port architecture mapping. This mapping reduces the required source code modifications. Table 1 shows the assignments of the JTAG configuration pins to the parallel port registers.

Table 1. 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 1:

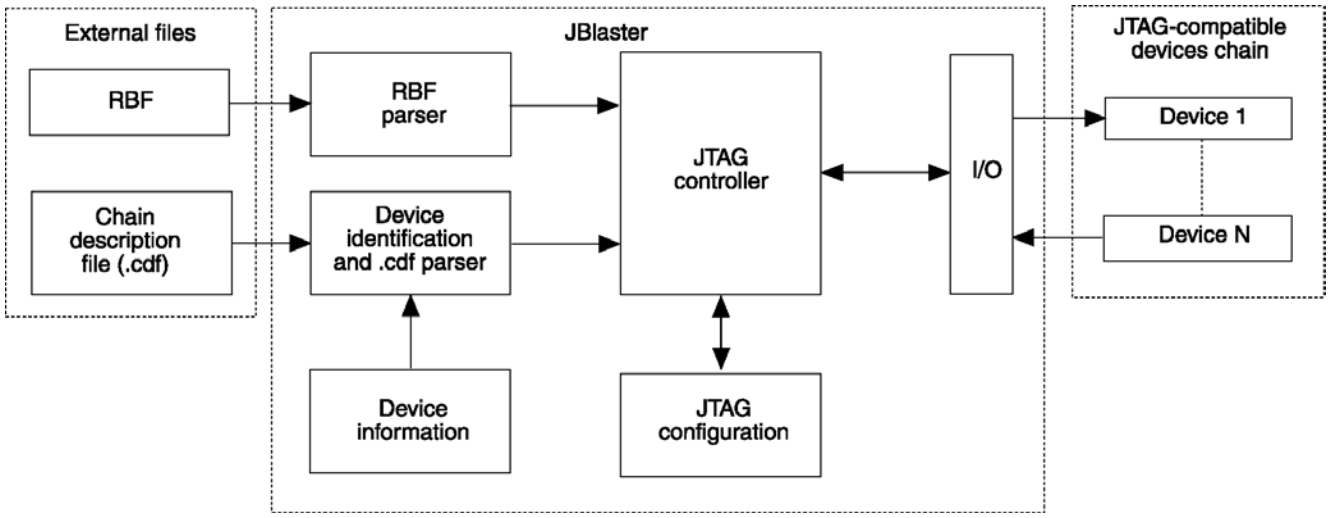
(1) The port refers to the index from the base address of the parallel port, e.g., 0x378.

(2) Inverted signal.

Block Diagram

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

Figure 1. JBlaster Block Diagram and Interfaces



Source Files

The JBlaster source code is written in the ANSI C Language. It is divided into modules that reside in separate files. Table 2 describes the JBlaster source code files.

Table 2. Source Files

File	Description	Platform Independent
jblaster.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 the 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**	Handle the JTAG instructions and keep track of the JTAG state machine (JSM)	Yes
jb_io.c **jb_io.h**	Handle the I/O control functions, file processing functions and string processing functions, and may be customized to work with your system	No

Input Files

The JBlaster software driver supports the raw binary file (RBF) format programming source file. The JBlaster driver also requires a chain description file (**.cdf**) generated by the Quartus II software. The chain description file (CDF) contains information on the devices in the JTAG chain.

The JBlaster driver processes the action code for each device in the CDF. The supported action codes are `CFG` and `IGN` for `PROGRAM` and `BYPASS JTAG` instructions, respectively. The JBlaster software only configures Altera devices.

Program and User-defined Constants

The source code has program and user-defined constants. Do not change the program constants. You should set the values for the user-defined constants. Table 3 summarizes the program and user-defined constants.

Table 3. Program and User-defined Constants

Constant	Type	File	Description
WINDOWS_NT	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 CDF path.
CDF_PATH_LENGTH	Program	**jb_const.h**	The maximum characters allocated for the CDF 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 4 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 4. Global Variables

Global Variables	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) Y is the port number.

Notes to Table 4:

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

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

I/O Routines

Table 5 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 JBlaster software on platforms other than Windows NT. These functions contain the I/O control routines.

Table 5. 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 test`
		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 **jblaster.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 `test`=1. If `test`=0, the operations are stored in the buffer and flushed once the number of operations reaches 256.

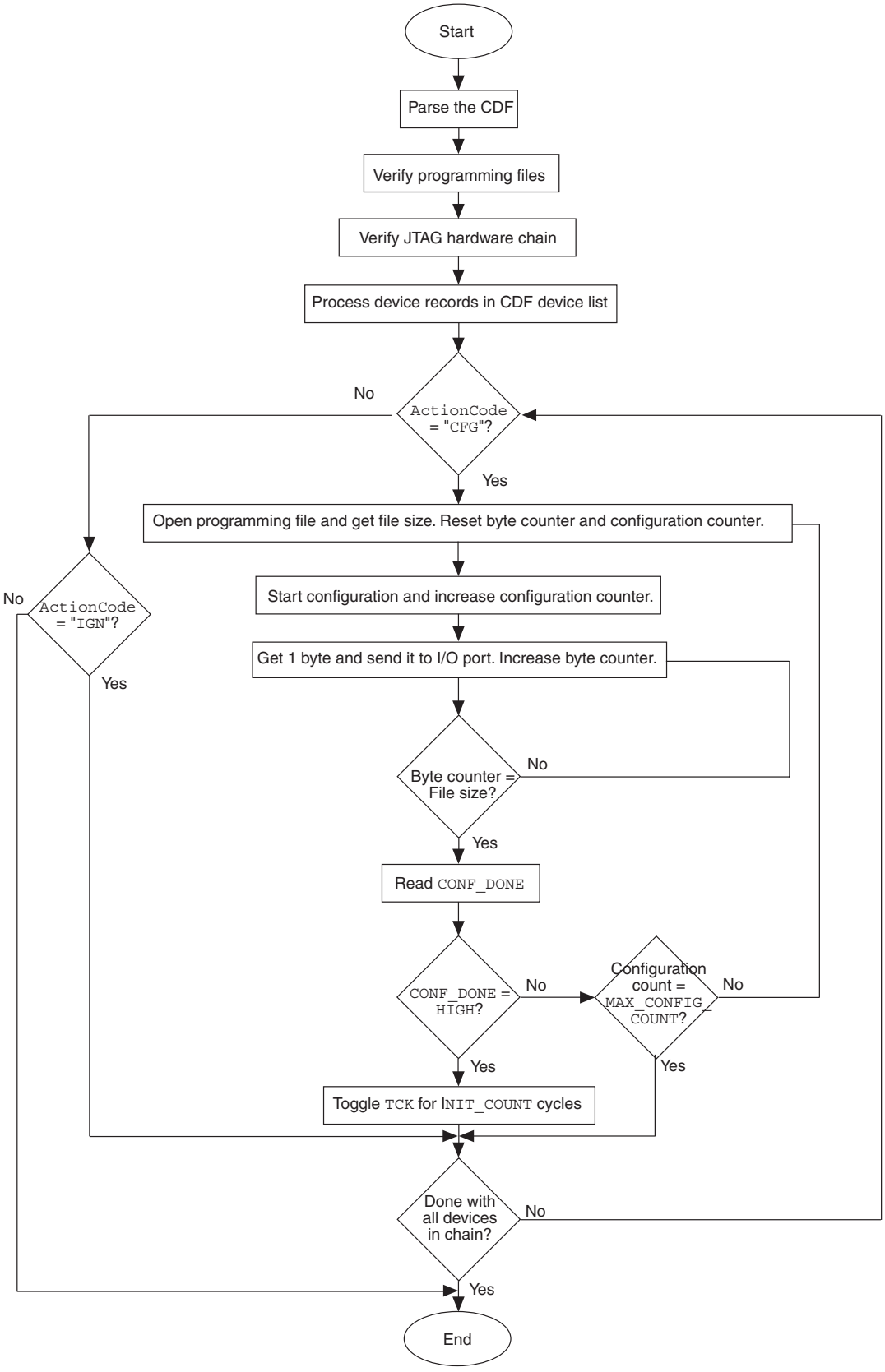
Note to Table 5:

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

Program Flow

Figure 2 illustrates the program flow of the JBlaster software driver. The `MAX_DEVICE_ALLOW`, `MAX_CONFIG_COUNT`, and `INIT_COUNT` constants determine the flow of the configuration process. See Table 3.

Figure 2. JBlaster Program Flow



Porting the Source Code to Other Platforms or Embedded Systems

Two separate platform-dependent routines handle the read and the 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 5). 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 1) 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 1. 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 **jblaster.c** file.

Writing Data to the I/O Ports

The `writeport` function accepts three integer parameters, `port`, `data`, and `test`. 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 1) to the I/O port definition of your system.

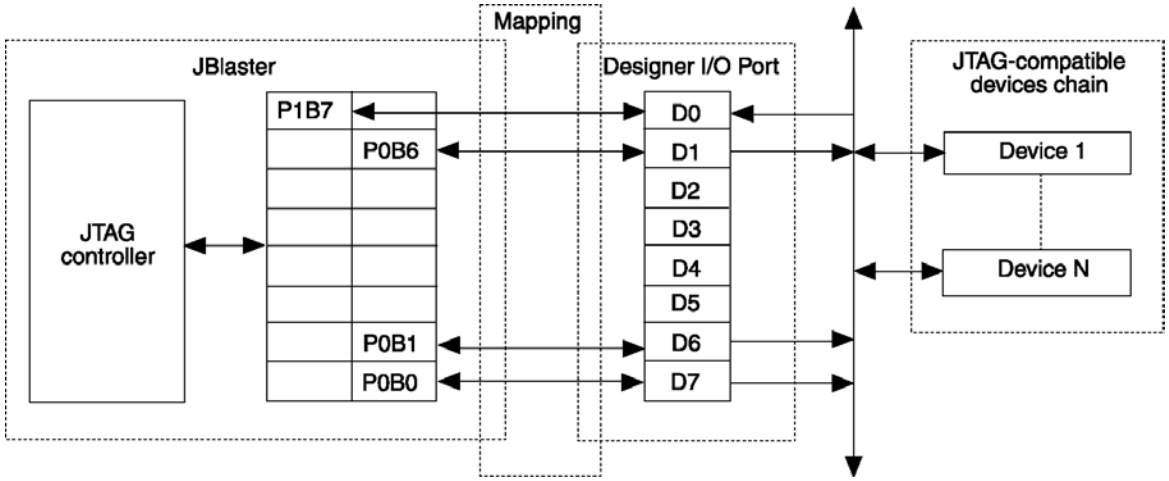
For example, when writing to port 0, your source code should identify the `TDI`, `TMS`, and `TCK` signals represented in each bit of the data parameter. The source code should mask the data variable with the `sig_port_maskbit` variable (see Table 4) to extract the value of the signal to write. To extract `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 Processes



Conclusion

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

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com

Copyright © 2002 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries.* All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.