AN423: Configuring the MicroBlaster Passive Serial Software Driver

June 2008, version 1.1

Introduction

The MicroBlaster™ software driver configures Altera® programmable logic devices (PLDs) in passive serial (PS) mode for embedded configurations through the ByteBlaster™ II and ByteBlasterMVTM download cables. You can customize the modular source code's I/O control routines (provided as separate files) for your system. The MicroBlaster software driver is an embedded configuration driver that supports the Raw Binary File (.rbf) format generated by the Quartus® II software.

This application note describes how the MicroBlaster software driver works, the important parameters and functions of its source code, and how to port its source code to an embedded platform.

☞ The MicroBlaster software driver is developed and tested on the Windows NT platform. This Windows NT driver's binary file size is about 40 Kbytes.

Interface

The MicroBlaster software driver's source code has two modules:

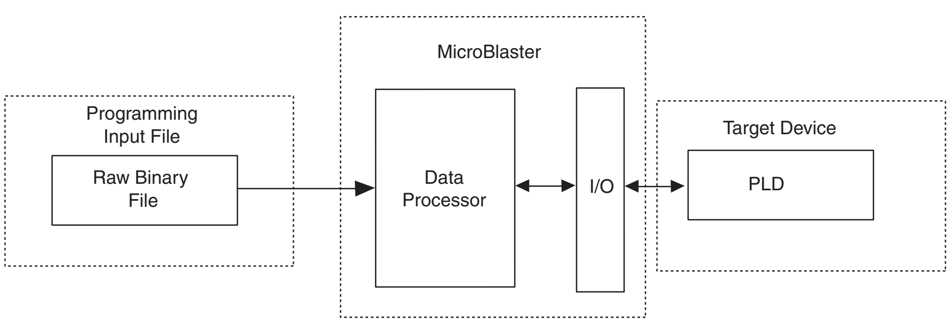
- Data processing
- I/O control

The data processing module reads the programming data from the Raw Binary File, rearranges it, and sends it to the I/O control module. The I/O control module sends that data to the target PLD. Periodically, the I/O control module senses certain configuration pins to determine if errors occurred during the configuration process. When an error occurs, the MicroBlaster source code re-initiates the configuration process.

Block Diagram

Figure 1 shows the MicroBlaster software block diagram and its interfaces to the programming input file and target PLD.

Figure 1. MicroBlaster Block Diagram & Interfaces



Source Files

Table 1 describes the MicroBlaster source files.

Table 1. Source Files

File	Description
mblaster.c	Contains the main ( ) function. It manages the processing of the programming input file, instantiates the configuration process, and handles any configuration errors. This file is platform independent.
mb_io.c **mb_io.h**	These files handle the I/O control functions and are platform dependent. They support the ByteBlaster II or ByteBlasterMV download cable for PCs running Windows NT only. You should modify these files to support other platforms.

The source files are available for download from the Altera website at **www.altera.com**.

Table 2 describes the directory structure of the source files.

Table 2. Directory Structure

Folders in MicroBlaster Driver	Files Available	Description
bin	MBlaster.exe, **MBlaster.txt**	Executable file for MicroBlaster driver
doc	an423.pdf, **readme.txt**	MicroBlaster documentation
source	**mblaster.c**, **mb_io.c**, **mb_io.h**	Source files

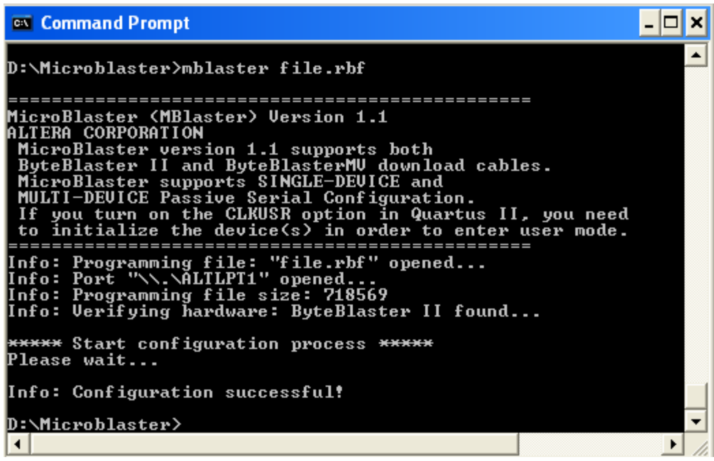
How To Use MicroBlaster

The MicroBlaster software drive supports the `Raw Binary File programming` source file (.rbf).You can generate the (.rbf) file from the Quartus II compilation or use the Quartus II Software Convert Programming File utility. After generating the .rbf, type the following command line at the Windows command prompt to configure the device:

```
mblaster <filename>.rbf
```

Figure 2 shows the screenshot of the execution of MicroBlaster software using `mblaster <filename>.rbf` command-line.

Figure 2. Configuring the Device with mblaster <filename>.rbf command



Parameters and Functions

Because the writing and reading of the data to and from the I/O ports on other platform maps to the parallel port architecture, this application note uses the pin assignments of the (PS) configuration signals to a parallel port. These pin assignments reduce the required source code modifications. Table 3 shows the assignment of the passive serial configuration signals to the parallel port.

Table 3. Pin Assignments of the Passive Serial Configuration Signals to the Parallel Port

Bit	7	6	5	4	3	2	1	0
Port 0 *(1)*	—	DATA0	—	—	—	—	nCONFIG	DCLK
Port 1 *(1)*	CONF_DONE	—	—	nSTATUS	—	—	—	—
Port 2 *(1)*	—	—	—	—	—	—	—	—

Note to Table 3:

(1) This port refers to the index from the base address of the parallel port; for example, 0x378.

Program and User-Defined Constants

The source code has program and user-defined constants. You should not change the program constants. You should set the values for user-defined constants. Table 4 summarizes the constants.

Table 4. Program and User-Defined Constants (Part 1 of 2)

Constant	Type	Description
WINDOWS_NT	Program	Designates Windows NT operating system.
EMBEDDED	Program	Designates embedded microprocessor system or other operating system.
PORT	Program	Determines the platform.
SIG_DCLK	Program	`DCLK` signal (port 0, bit 0)

Table 4. Program and User-Defined Constants (Part 2 of 2)

Constant	Type	Description
SIG_NCONFIG	Program	nCONFIG signal (port 0, bit 1)
SIG_DATA0	Program	DATA0 signal (port 0, bit 6)
SIG_NSTATUS	Program	nSTATUS signal (port 1, bit 4)
SIG_CONFDONE	Program	CONF_DONE signal (port 1, bit 7)
INIT_CYCLE	User-defined	The number of clock cycles to toggle after configuration is done to initialize the device. Each device family requires a specific number of clock cycles.
RECONF_COUNT_MAX	User-defined	The maximum number of auto-reconfiguration attempts allowed when the program detects an error.
CHECK_EVERY_X_BYTE	User-defined	Check nSTATUS pin for error every X number of bytes programmed. Do not use 0.
CLOCK_X_CYCLE (optional)	User-defined	The number of additional clock cycles to toggle after INIT_CYCLE. Use 0 if no additional clock cycles are required. The recommended value is 150 if this constant is used.

Global Variables

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

Table 5. Global Variables

Variable	Type	Description
sig_port_maskbit[W][X]	Two dimensional integer array	Variable that tells the port number of a signal and the bit position of the signal in the port register. *(1) (2)* W = 0 refers to SIG_DCLK W = 1 refers to SIG_NCONFIG W = 2 refers to SIG_DATA0 W = 3 refers to SIG_NSTATUS W = 4 refers to SIG_CONF_DONE X = 0 refers to the port number the signal falls into. For example, the signal SIG_DCLK falls into port number 0, and the signal SIG_NSTATUS falls into port number 1 (refer to Table 3). X = 1 refers to the bit position of thesignal.
port_mode_data[Y][Z]	Two dimensional integer array	The initial values of the registers in each port in different modes. The ports are in reset mode before and during configuration. The ports are in user mode after configuration. *(1)* Y = 0 refers to reset mode Y = 1 refers to user mode Z = port number

Table 5. Global Variables

Variable	Type	Description
`port_data[Z]`	Integer array	Holds the current value of each port. The value is updated each time a write is performed to the ports. *(1)* `Z` = port number.

Notes to Table 5:

(1) The port refers to the index from the base address of the parallel port; for example, 0×378.

(2) The signal refers to any of these signals: `SIG_DCLK`, `SIG_NCONFIG`, `SIG_DATA0`, `SIG_NSTATUS`, and `SIG_CONF_DONE`.

Functions

Table 6 describes the parameters and the return value of some of the functions in the source code. Only functions declared in the mb_io.c file are discussed because you need to customize these functions in order to work on platforms other than Windows NT. These functions contain the I/O control routines.

Table 6. Functions

Function	Parameters	Return Value	Description
`readbyteblaster`	`int port`	Integer	This function reads the value of the port and returns it. Only the least significant byte contains valid data. *(1)*

`writebyteblaster`	`int port`
	`int data`
	`int test`
		None	This function 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 3. *(1)*
			
			The functions in mblaster.c that call the `writebyteblaster` function have organized the bits. Only the value of specific bits are changed as needed before passing it to the `writebyteblaster` function as data.
			
			To reduce the number of dumps to the port, each time a signal other than `DCLK` is dumped to the port (typically the `DATA0` signal), the `DCLK` clock signal is toggled at the same time. The integer test determines if the `DCLK` signal needs to be toggled. *(1)*

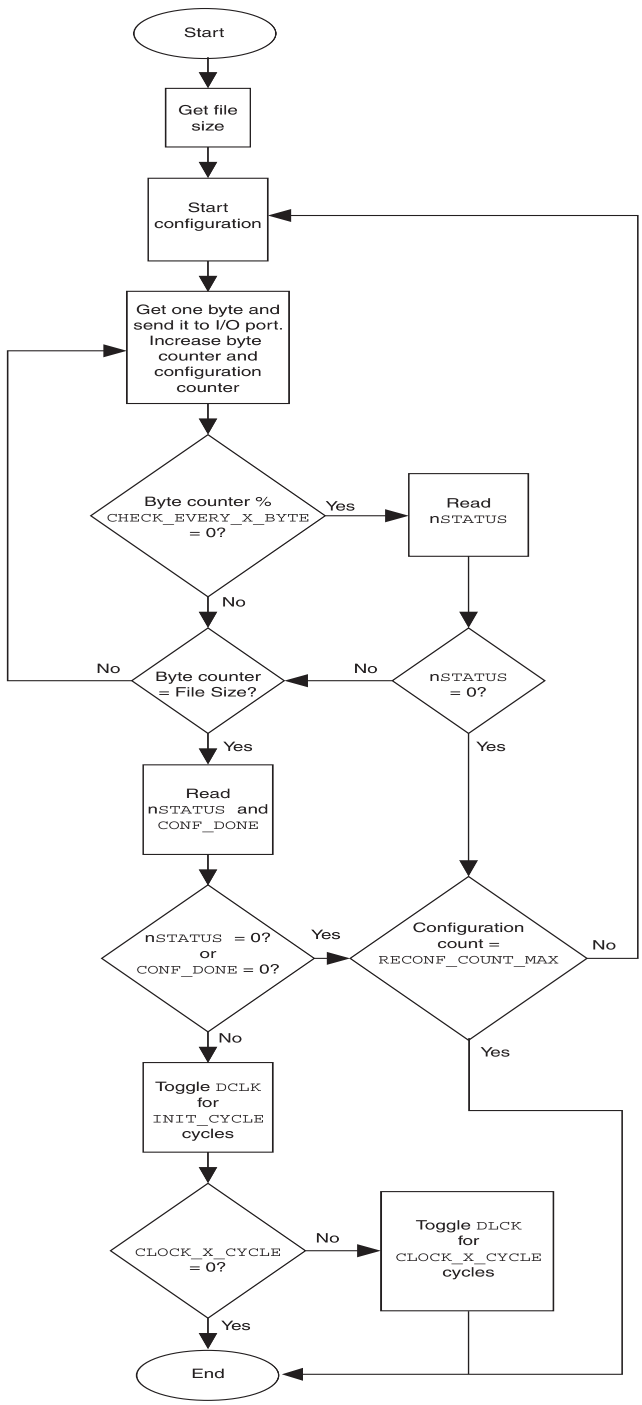
Note to Table 6:

(1) The port refers to the index from the base address of the parallel port; for example, 0×378.

Program Flow

Figure 3 illustrates the program flow of the MicroBlaster software driver. The CHECK_EVERY_X_BYTE, RECONF_COUNT_MAX, INIT_CYCLE, and CLOCK_X_CYCLE constants determine the flow of the configuration process. Refer to Table 4 for program and user-defined contants.

Figure 3. MicroBlaster Program Flow



Porting

Two separate platform-dependent routines handle the read and write operations in the I/O control module. The read operation reads the value of the required pin. The write operation writes data to 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, `readbyteblaster` and `writebyteblaster` (refer to 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

The `readbyteblaster` 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 (refer to 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` and `nSTATUS` signals from your system (defined in Table 3). Then the code should rearrange these signals within an integer variable so the values of `CONF_DONE` and `nSTATUS` are represented in bit positions 7 and 4 of the integer, respectively. This behaviorally maps your system's I/O ports to the pins in the pin assignments of the parallel port architecture. By adding these lines of translation code to the mb_io.c file, you can avoid modifying code in the mblaster.c file.

Writing

The `writebyteblaster` function accepts three integer parameters: `port`, `data`, and `test`. Modify the `writebyteblaster` function the same way as the `readbyteblaster` function. Your code maps or translates the port value that is defined in the parallel port architecture (refer to Table 3) to the I/O port definition of your system.

For example, when writing to port `0`, your source code should identify the `DATA0`, `nCONFIG`, and `DCLK` signals represented in each bit of the data parameter. The source code should mask the data variable with the `sig_port_maskbit` variable (refer to Table 5) to extract the value of the signal to write. To extract `DATA0` from "data" for example, mask "data" with `sig_port_maskbit[SIG_DATA0][1]`.

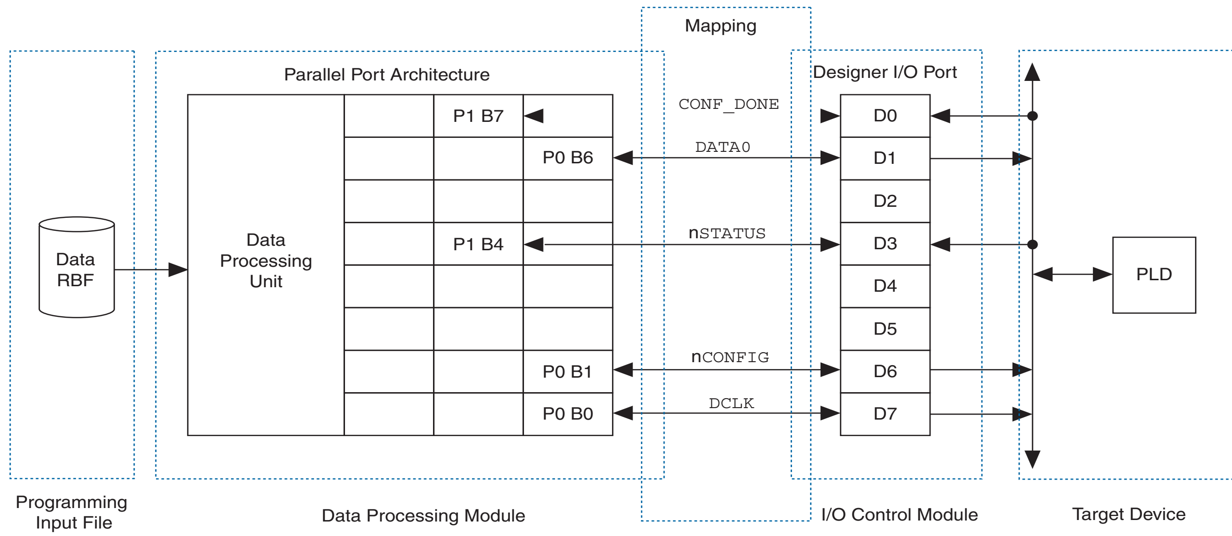
After extracting the values of the relevant signals, each signal is mapped to the I/O ports as defined in your system. By adding these translation code lines to the mb_io.c file, you can avoid modifying code in the mblaster.c file.

Example

Figure 4 shows an embedded system holding five configuration signals in the data registers `D0`, `D1`, `D3`, `D6`, and `D7` of an embedded microprocessor. When reading from the I/O ports, the I/O control routine reads the values of the data registers and maps them to the particular bits in the parallel port registers (`P0` to `P2`). These bits are later accessed and processed by the data processing module.

When writing, the values of the signals are stored in the parallel port registers (P0 to P2) by the data processing module. The I/O control routine then reads the data from the parallel port registers and sends it to the corresponding data registers (D0, D1, D3, D6, and D7).

Figure 4. Example of I/O Reading & Writing Mapping Process



Conclusion

The MicroBlaster passive serial embedded configuration source code is modular so you can easily port it to other platforms. It offers a simple and inexpensive embedded system to accomplish a PS configuration for Altera PLDs.

Document Revision History

Table 7 shows the revision history for this document.

Table 7. Document Revision History

Date and Chapter Version	Changes Made	Summary of Changes
June 2008 v1.1	■ Added new "How To Use MicroBlaster" and "Document Revision History" sections. ■ Added Figure 2.	Executable file for MicroBlaster driver
June 2006 v1.0	■ Initial Release.	Source Files