# The Serial-to-JTAG Board for MAXQ Processors

*This application note discusses the commands accepted by the Serial-to-JTAG board. This board is used to interface with MAXQ microcontrollers. The commands described here allow developers to read and write to the MAXQ's memory (code and data), read and write registers, and utilize the in-circuit debugger.*

## Introduction

The MAXQ microcontroller incorporates a Test Access Port (TAP) for communication with a host device across a 4-wire synchronous serial interface. This TAP is used to support in-system programming and in-circuit debugging. The TAP is compatible to the JTAG IEEE standard 1149. To connect to the TAP, Dallas Semiconductor developed a Serial-to-JTAG board and firmware that accept commands over a standard RS-232 serial port and concert these commands to the appropriate JTAG signals. This application note describes the command protocol implemented by the firmware. If you need more details on the Serial-To-JTAG board itself, send your questions to micro.software@dalsemi.com.

Note: This application note assumes familiarity with the MAXQ microcontroller's TAP and basic JTAG communications. For detailed information regarding these topics, refer to the MAXQ Family User's Guide.

## Interfacing with the Firmware

To establish communication with the Serial-to-JTAG board, connect to the board's serial port at 115200 baud, using 8 data bits, no parity, and 1 stop bit. Once connected, you can interface with the firmware in either of two modes: ASCII or binary. The firmware defaults to ASCII mode where human readable text commands are sent to the board and the results are returned as text strings. In binary mode, explained later, all transmissions are a series of 8-bit bytes. There are commands that allow switching between the two modes at any time. When in ASCII mode, the commands listed in **Table 1** are always available. All commands are case sensitive. Groups of commands can be entered on a single line, or each command can be entered one at a time.

**Table 1. Commands Accepted in Any Mode**

| Command | Description |
|---|---|
| h | Halts the MAXQ by holding it in reset. |
| H | Releases RESET allowing the MAXQ to run. |
| I | Puts the system into bypass mode and resets the TAP, returning it to the Run-Test-Idle state. |

| | |
|---|---|
| JB | Instructs the firmware to begin accepting background mode commands. This command does not switch modes on the target MAXQ nor does it send any JTAG commands to the target device. It is intended only to instruct the firmware that the MAXQ has changed modes by some other means. |
| JD | Instructs the firmware to begin accepting debug mode commands. This command does not switch modes on the target MAXQ nor does it send any JTAG commands to the target device. It is intended only to instruct the firmware that the MAXQ has changed modes by some other means. |
| JL | Instructs the firmware to begin accepting bootstrap loader commands. This command does not switch modes on the target MAXQ nor does it send any JTAG commands to the target device. It is intended only to instruct the firmware that the MAXQ has changed modes by some other means. |
| JX | Instructs the firmware to begin accepting bypass mode commands. This command does not switch modes on the target MAXQ nor does it send any JTAG commands to the target device. It is intended only to instruct the firmware that the MAXQ has changed modes by some other means. |
| Q | Queries the interface version number of the JTAG board. The version number will be output as two hexadecimal characters. This version will change anytime the format of any command or its output changes. The interface version at the time this document was written, was 01. |
| q | Queries the firmware version number of the JTAG board. The version number will be output as two hexadecimal characters. This version will change anytime the firmware changes. The firmware version at the time this document was written, was 02. |
| Vtxxyy | Sets up Timer0 of the JTAG board. As the JTAG clock must be less than 1/8 of the target's clock, the firmware uses Timer0 to control the speed of the JTAG clock. The firmware waits for the timer to overflow before generating each edge of the JTAG clock. Replace 't' with the value to use for the timer's T0M bit, 'xx' with the value to use for TH0, and 'yy' with the value to use for TL0. All values should be entered in hexadecimal format. For more information about the purpose of these values, see the Ultra-High-Speed Flash Microcontroller User's Guide. |
| Yrbbdd | Sends a value directly to the TAP. Replace 'r' with the TAP register to which you want to write to: 0 for DR and 1 for IR. 'bb' is the number of bits to write (not including the status bits) and 'dd' is the data to send. All values should be entered in hexadecimal format. |
| Z | Switches the firmware to binary transfers. |
| z | Executes a single pulse of the JTAG clock. |
| + | Performs a simple hardware test of the JTAG board. The CLK, TMS, and TDI pins are all asserted, and the state of TDO is read and output as a '0' or '1'. The voltage at the pins can then be measured to ensure that they are functioning properly. |
| - | Performs a simple hardware test of the JTAG board. The CLK, TMS, and TDI pins are all set to a logic low, and the state of TDO is read and output as a '0' or '1'. The voltage at the pins can then be measured to ensure that they are functioning properly. |

As indicated by the commands above, there are a few different modes for the MAXQ JTAG engine: bypass mode, bootstrap loader mode, background mode, and debug mode. The JTAG engine functions differently in each mode. As a result, in addition to the commands listed above, other commands become available when the JTAG engine enters these different modes.

## Bypass Mode

The TAP initializes to bypass mode during a power-on-reset. In this mode, the TAP is disabled and does not interact with the rest of the MAXQ microcontroller. To activate the TAP, enter one of the two additional commands available in this mode: 'D' and 'L'. The 'D' command activates the in-circuit debugger and the 'L' command activates the bootstrap loader.

## Bootstrap Loader Mode

When the bootstrap loader has been activated using the 'L' command, you can send bytes directly to the MAXQ's utility ROM. Enter each value as two hexadecimal characters. (For more information on the bytes accepted by the utility ROM, contact micro.software@dalsemi.com.) For each byte entered, the firmware outputs the byte returned by the loader and the status bits received from the TAP. The format for the output will be "00xx:ss", where 'xx' is the output byte and 'ss' are the status bits. Once you enter the "Exit Loader" command (0x01), you must use one of the 'J' commands listed in Table 1 to instruct the JTAG board that the MAXQ is no longer in bootstrap loader mode.

## Background Mode

When in the background mode of the JTAG engine, you can read and write the JTAG breakpoint registers (BP0-BP5), read and write the in-circuit debug registers (ICDC, ICDF, ICDA, and ICDD), determine when a breakpoint match has occurred, and invoke debug mode manually. The commands to support these operations are listed in **Table 2**. For all the commands in this mode that have output, the format will be "xxyy:ss", where 'xx' is the MSB of the output data, 'yy' is the LSB, and 'ss' are the status bits returned by the TAP.

**Table 2. Commands Available in Background Mode**

| Command | Description |
|---|---|
| A | Read the ICDA register. |
| axxyy | Write the ICDA register, where 'xx' is the MSB of the new value and 'yy' is the new LSB. Values should be entered as two hexadecimal characters. |
| Bi | Read any of the 6 breakpoint registers, where 'i' is the index of the breakpoint register to read (0 through 5). |
| bixxyy | Write any of the 6 breakpoint registers, where 'i' is the index of the breakpoint register to write (0 through 5), 'xx' is the MSB of the new value, and 'yy' is the LSB. The MSB and LSB values should be entered as two hexadecimal characters. |
| C | Read the ICDC register. |
| cxx | Write the ICDC register, where 'xx' is the new value. Values should be entered as two hexadecimal characters. |

| | |
|---|---|
| D | Read the ICDD register. |
| dxxyy | Write the ICDD register, where 'xx' is the MSB of the new value and 'yy' is the new LSB. Values should be entered as two hexadecimal characters. |
| E | Enter debug mode. |
| F | Read the ICDF register. |
| N | No operation. |

## Debug Mode

There are two ways the JTAG engine can enter debug mode. The first way is to enter the "Enter debug mode" command ("E") while in background mode. The second way debug is activated occurs when a breakpoint match occurs. In this case, you should enter the "JD" command to inform the firmware that the mode has changed. Once in debug mode, you can read and write the MAXQ registers, read the program stack, read and write data memory, single step the MAXQ CPU, return to background mode, and perform a password match to unlock certain commands. **Table 3** lists the commands supporting this functionality.

**Table 3. Commands Available in Debug Mode**

| Command | Description |
|---|---|
| E | Exit debug mode, return to background mode. |
| G | Gets all registers. The order of the registers returned depends on the type of MAXQ device. |
| Mxxyyiijj | Read data memory, where 'xx' is the MSB of the word address to read, 'yy' is the LSB of the address, 'ii' is the MSB of the number of words to read, and 'jj' is the LSB of the length. All values should be entered as two hexadecimal characters. |
| mxxyyiijj | Write a word to data memory, where 'xx' is the MSB of the word address, 'yy' is the LSB of the address, 'ii' is the MSB of the word to be written, and 'jj' is the LSB of the word to be written. All values should be entered as two hexadecimal characters. |
| n | No operation. |
| Pxx1...xx32 | Attempt a password match with the given data. All 32 values should be entered as two hexadecimal characters. |
| R0iim | Read a register, where 'ii' is the register's index and 'm' is the register's module. The index should be entered as two hexadecimal characters, and the module should be entered as a single hexadecimal character. |
| r0iimxxyy | Write a register, where 'ii' is the register's index, 'm' is the register's module, 'xx' is the MSB of the new value, and 'yy' is the LSB. The index and each byte of the new value should be entered as two hexadecimal characters. The module should be entered as a single hexadecimal character. |
| Sxxyyiijj | Read the program stack, where 'xx' is the MSB of the word address to read, 'yy' is the LSB of the address, 'ii' is the MSB of the number of words to read, and 'jj' is the LSB of the length. All values should be entered as two hexadecimal characters. |

| | |
|---|---|
| T | Execute the instruction at the current instruction pointer. |

Note: All background mode commands listed in Table 2 (except 'E') can also be used while in Debug Mode.

## Binary Transfers

All the commands described in Tables 1, 2, and 3 are easily entered manually and their outputs are easily understood. In many cases, however there will be software controlling the JTAG board. Because the ASCII commands are not conveniently used by software and there is unnecessary processing required in converting the results back into binary data, the JTAG firmware also supports binary transfers. When binary transfer mode, data is sent to the TAP by first sending a byte that indicates the number of bytes being sent. The data should then follow. For every byte of data sent, the firmware outputs two bytes. The first byte returned is the status bits that were read during the transfer. The second byte will hold the value read from the TAP during the transfer. There are also special commands that can be sent while in binary mode. These commands are described in **Table 4**. To send one of these special commands, send a 0 for the length byte. This instructs the firmware to treat the next byte received as a special command. For these special commands, a single byte is returned. Typically this will just be an echo of the command.

**Table 4. Special Commands Used in Binary Transmissions**

| Command | Description |
|---|---|
| 0x00 | Exit binary mode transfers and return to accepting ASCII commands. |
| 0x01 | Set the TAP's IR register as the destination for data transfers. |
| 0x02 | Set the TAP's DR register as the destination for data transfers. |
| 0x03 | Send only the lowest 3 bits of each byte when transferring data. |
| 0x04 | Set the RESET pin to a logic high. |
| 0x05 | Clear the RESET pin to a logic low. |
| 0x06 | Execute a single pulse of the JTAG clock. |
| 0x07 | Read the state of the TDO pin. |
| 0x08 | Send all 8 bits of each byte when transferring data. |
| 0x09 | Set the TMS pin to a logic high. |
| 0x0A | Clear the TMS pin to a logic low. |
| 0x0B | Set the TDI pin to a logic high. |
| 0x0C | Clear the TDI pin to a logic low. |
| 0x0D | Set the T0M bit. For more information on this bit, refer to the 'V' command described in Table 1. |
| 0x0E | Clear the T0M bit. For more information on this bit, refer to the 'V' command described in Table 1. |

| | |
|---|---|
| 0x11 | Use the next byte received as the value for TL0. This "next" byte does not require a length byte or the '0' used as the special-command escape character. For more information of the TL0 register, refer to the 'V" command described in Table 1. |
| 0x12 | Use the next byte received as the value for TH0. This "next" byte does not require a length byte or the '0' used as the special command escape character. For more information of the TH0 register, refer to the 'V" command described in Table 1. |

## Detecting Errors

In both the ASCII transfer mode and the binary transfer mode, any errors that occur are indicated by the output from the commands. In ASCII mode errors will be output as "*ERR=xx*", where xx indicates the type of error that occurred. In binary mode the error code is output instead of the command echo. Refer to **Table 5** for the description of the possible error codes.

**Table 5. Error Codes**

| Error Code | Description |
|---|---|
| 0x80 | Command not recognized or invalid command. |
| 0x90 | Received an invalid hexadecimal character. |
| 0xA0 | Not enough input received. |
| 0xB0 | Bad breakpoint register index. |
| 0xC? | Received unexpected status, where ? represents the status bits received. |

## Conclusion

Using the commands described in this document, the Serial-To-JTAG board can be used to load code into MAXQ processors, read and write system registers, read and write memory, and utilize the in-circuit debugger. This process can either be automated through host software using the binary protocol or entered interactively with a terminal program. Building blocks for all the commands needed to fully control a MAXQ system have been provided.