NAME

MSPDebug - debugging tool for MSP430 MCUs

SYNOPSIS

mspdebug [options] *driver* [*command* ...]

DESCRIPTION

MSPDebug is a command-line tool designed for debugging and programming the MSP430 family of MCUs. It supports the eZ430-F2013, eZ430-RF2500, Launchpad, Chronos, FET430UIF, GoodFET, Olimex MSP430-JTAG-TINY and MSP430-JTAG-ISO programming tools, as well as a simulation mode.

When started with appropriate options, MSPDebug will attempt to connect to the debugging tool specified and identify the device under test. Once connected, the user is presented with a command prompt which can be used to reflash the device memory, inspect memory and registers, set registers, and control the CPU (single step, run and run to breakpoint).

It supports a variety of file formats, described in the section **BINARY FORMATS** below. It can also be used as a remote stub for gdb(1).

On startup, MSPDebug will look for a file called .mspdebug first in the current directory, and then in the user's home directory. If either file exists, commands will be read and executed from this file before executing any other commands or starting the interactive reader.

Alternatively, a configuration file can be explicitly specified with the **-C** option.

COMMAND-LINE OPTIONS

Command-line options accepted by MSPDebug are described below. If commands are specified on the end of the command-line, then they are executed after connecting to the device, and the interactive prompt is not started. Please be aware that commands consisting of multiple words need to be enclosed in quotation marks, otherwise they are treated as single commands. Thus the common prog command would be used as "prog main.elf". See the section labelled **COMMANDS** for more information.

-q
: Start in quiet mode. See the "quiet" option described below.

-v *voltage*
: Set the programming voltage. The voltage should be specified as an integer in millivolts. It defaults to 3000 (3.0 V).

-j

Use JTAG instead of Spy-Bi-Wire to communicate with the MSP430. This option doesn't work with eZ430 or eZ430-RF2500 devices, which support Spy-Bi-Wire only.

-d *device*
Specify that the driver should connect via a tty device rather than USB. The supported connection methods vary depending on the driver. See the section **DRIVERS** below for details.

-U *bus*:*device*
Specify a particular USB device to connect to. Without this option, the first device of the appropriate type is opened.

-s *serial*
Specify a particular USB device serial number to connect to. Use this option to distinguish between multiple devices of the same type.

-n
Do not process the startup file (~/.mspdebug).

-C *file*
Specify an alternative configuration file (default is ~/.mspdebug). If -n is specified as well, no file will be read.

--long-password
When using the flash-bsl driver, send a 32-byte BSL password instead of the standard 16-byte password.

--help
Display a brief help message and exit.

--fet-list
Display a list of chips supported by the FET driver (the driver used for UIF, RF2500 and Olimex devices).

--fet-force-id *string*
When using a FET device, force the connected chip to be recognised by MSPDebug as one of the given type during initialization. This overrides the device ID returned by the FET. The given string should be a chip name in long form, for example "MSP430F2274".

--fet-skip-close
When using a FET device, skip the JTAG close procedure when disconnecting. With some boards, this removes the need to replug the debugger after use.

--usb-list
List available USB devices and exit.

--force-reset
When using a FET device, always send a reset during initialization. By default, an initialization without reset will be tried first.

--allow-fw-update
When using a V3 FET device via the TI library, allow the library to perform a firmware update if the FET firmware is incompatible with the library.

--require-fw-update *image.txt*
When using a V3 FET device, or certain Olimex devices, force a firmware update using the given firmware image. The firmware format depends on the driver.

--version
Show program version and copyright information.

--embedded
Start mspdebug as an embedded subprocess. See the documentation accompanying the source release for more information on embedded mode.

DRIVERS

A driver name must be specified on the command line for MSPDebug to connect to. Valid driver names are listed here.

rf2500 Connect to an eZ430-RF2500, Launchpad or Chronos device. Only USB connection is supported.

olimex Connect to an Olimex MSP430-JTAG-TINY device. Both USB and tty access are supported.

olimex-v1
Connect to an Olimex MSP430-JTAG-TINY (V1) device. Both USB and tty access are supported. This driver must be used instead of **olimex** if connecting to a V1 device via a tty interface.

olimex-iso
Connect to an Olimex MSP430-JTAG-ISO device. Both USB and tty access are supported.

olimex-iso-mk2
Connect to an Olimex MSP430-JTAG-ISO-MK2 device. Both USB and tty access are supported.

sim Do not connect to any hardware device, but instead start in simulation mode. A 64k buffer is allocated to simulate the device memory.

During simulation, addresses below 0x0200 are assumed to be IO memory. Programmed IO writes to and from IO memory are handled by the IO simulator, which can be configured and controlled with the **simio** command, described below.

This mode is intended for testing of changes to MSPDebug, and for aiding the disassembly of MSP430 binaries (as all binary and symbol table formats are still usable in this mode).

uif Connect to an eZ430-F2013 or a FET430UIF device. The device argument should be the filename of the appropriate tty device. The TI serial converter chips on these devices are supported by newer versions of the Linux kernel, and should appear as /dev/tty*XX* when attached.

USB connection is supported for this driver. The USB interface chip in these devices is a TI3410, which requires a firmware download on startup. MSPDebug will search for a file called ti_3410.fw.ihex in the configured library directory and the current directory. You can specify an alternate location for the file via the **MSPDEBUG_TI3410_FW** environment variable.

uif-bsl Connect to the bootloader on a FET430UIF device. These devices contain MSP430F1612 chips. By sending a special command sequence, you can obtain access to the bootloader and inspect memory on the MSP430F1612 in the programming device itself.

Currently, only memory read/write and erase are supported. CPU control via the bootloader is not possible.

flash-bsl

Connect to the built-in bootloader in MSP430 devices with flash bootloader memory. Devices with ROM bootloaders require another driver. Currently, this driver must mass-erase the device in order to gain access. Read, write, and erase operations are supported.

USB connection is not supported for this driver. Connection is via serial port, and bootloader entry is accomplished via the RTS and DTR lines. Connect RTS to the device's TEST pin and DTR to the device's RST pin. Use an appropriate serial level-shifter to make the connection, if necessary. If connecting to a device with non-multiplexed JTAG pins, connect RTS to the device's TCK pin via an inverter.

gdbc GDB client mode. Connect to a server which implements the GDB remote protocol and provide an interface to it. To use this driver, specify the remote address in *hostname:port* format using the **-d** option.

tilib Use the Texas Instruments MSP430.DLL to access the device. The library file (MSP430.DLL for Windows, libmsp430.so for Unix-like systems) must be present in the dynamic loader search path.

USB connection is not supported for this driver. This driver supports watchpoints. Note that the **-d** option for this driver passes its argument straight through to the library's **MSP430_Initialize** function. Any special argument supported by that function is therefore accessible via the **-d** option.

goodfet Connect to a GoodFET device. JTAG mode must be used, and only TTY access is supported. This device can be used for memory access (read, erase and program), but CPU control is limited. The CPU may be halted, run and reset, but register access and breakpoints aren't supported.

pif Connect to a parallel-port JTAG controller. Currently, this driver is only supported for Linux. A parallel port device must be specified via the **-d** option.

COMMANDS

MSPDebug can accept commands either through an interactive prompt, or non-interactively when specified on the command line. The supported commands are listed below.

Commands take arguments separated by spaces. Any text string enclosed in double-quotation marks is considered to be a single argument, even if it contains space characters. Within a quoted string, the usual C-style backslash substitutions can be used.

Commands can be specified by giving the first few characters of the command name, provided that the prefix is unambiguous. Some commands support automatic repeat. For these commands, pressing enter at the reader prompt without typing anything will cause repeat execution.

= *expression*

Evaluate an address expression and show both its value, and the result when the value is looked up in reverse in the current symbol table. This result is of the form *symbol*+*offset*, where *symbol* is the name of the nearest symbol not past the address in question.

See the section marked **ADDRESS EXPRESSIONS** for more information on the syntax of expressions.

alias Show a list of defined command aliases.

alias *name*

Remove a previously defined command alias.

alias *name command*

Define a command alias. The text *command* will be substituted for *name* when looking up commands. The given command text may contain a command plus arguments, if the entire text is wrapped in quotes when defining the alias. To avoid alias substitution when interpreting commands, prefix the command with \ (a backslash character).

break Show a list of active breakpoints. Breakpoints can be added and removed with the **setbreak** and **delbreak** commands. Each breakpoint is numbered with an integer index starting at 0.

cgraph *address length* [*address*]

Construct the call graph of all functions contained or referenced in the given range of memory. If a particular function is specified, then details for that node of the graph are displayed. Otherwise, a summary of all nodes is displayed.

Information from the symbol table is used for hinting at the possible locations of function starts. Any symbol which does not contain a "." is considered a possible function start.

Callers and callee names are shown prefixed by a "*" where the transition is a tail-call type transition.

delbreak [*index*]

Delete one or all breakpoints. If an index is given, the selected breakpoint is deleted. Otherwise, all breakpoints are cleared.

dis *address* [*length*]

Dissassemble a section of memory. Both arguments may be address expressions. If no length is specified, a section of the default length (64 bytes) is disassembled and shown.

If symbols are available, then all addresses used as operands are translated into *symbol*+*offset* form.

This command supports repeat execution. If repeated, it continues to disassemble another block of memory following that last printed.

erase [**all**|**segment**|**segrange**] [*address*] [*size*] [*segrange*]

Erase the device under test. With no arguments, all code memory is erased (but not information or boot memory). With the argument "all", a mass erase is performed (the results may depend on the state of the LOCKA bit in the flash memory controller).

Specify "segment" and a memory address to erase an individual flash segment. Specify "segrange", an address, size and segment size to erase an arbitrary set of contiguous segments.

exit Exit from MSPDebug.

fill *address length b0* [*b1 b2 ...*]
Fill the memory region of size *length* starting at *address* with the pattern of bytes given (specified in hexadecimal). The pattern will be repeated without padding as many times as necessary without exceeding the bounds of the specified region.

gdb [*port*]
Start a GDB remote stub, optionally specifying a TCP port to listen on. If no port is given, the default port is controlled by the option **gdb_default_port**.

MSPDebug will wait for a connection on this port, and then act as a GDB remote stub until GDB disconnects.

GDB's "monitor" command can be used to issue MSPDebug commands via the GDB interface. Supplied commands are executed non-interactively, and the output is sent back to be displayed in GDB.

help [*command*]
Show a brief listing of available commands. If an argument is specified, show the syntax for the given command. The help text shown when no argument is given is also shown when MSPDebug starts up.

hexout *address length filename*
Read the specified section of the device memory and save it to an Intel HEX file. The address and length arguments may both be address expressions.

If the specified file already exists, then it will be overwritten. If you need to dump memory from several disjoint memory regions, you can do this by saving each section to a separate file. The resulting files can then be concatenated together to form a single valid HEX file.

isearch *address length* [*options ...*]
Search over the given range for an instruction which matches the specified search criteria. The search may be narrowed by specifying one or more of the following terms:

opcode *opcode*
Match the specified opcode. Byte/word specifiers are not recognised, as they are specified with other options.

byte Match only byte operations.

word Match only word operations.

aword Match only address-word (20-bit) operations.

jump Match only jump instructions (conditional and unconditional jumps, but not instructions such as BR which load the program counter explicitly).

single Match only single-operand instructions.

double Match only double-operand instructions.

noarg Match only instructions with no arguments.

src *address*
Match instructions with the specified value in the source operand. The value may be given as an address expression. Specifying this option implies matching of only double-operand instructions.

dst *address*
Match instructions with the specified value in the destination operand. This option implies that no-argument instructions are not matched.

srcreg *register*
Match instructions using the specified register in the source operand. This option implies matching of only double-operand instructions.

dstreg *register*
Match instructions using the specified register in the destination operand. This option implies that no-argument instructions are not matched.

srcmode *mode*
Match instructions using the specified mode in the source operand. See below for a list of modes recognised. This option implies matching of only double-operand instructions.

dstmode *mode*
Match instructions using the specified mode in the destination operand. See below for a list of modes. This option implies that no-argument instructions are not matched.

For single-operand instructions, the operand is considered to be the destination operand.

The seven addressing modes used by the MSP430 are represented by single characters, and are listed here:

R Register mode.
I Indexed mode.
S Symbolic mode.
& Absolute mode.
@ Register-indirect mode.
+ Register-indirect mode with auto-increment.
Immediate mode.

load *filename*
Program the device under test using the binary file supplied. This command is like **prog**, but it does not load symbols or erase the device before programming.

The CPU is reset and halted before and after programming.

load_raw *filename address*
Write the data contained in a raw binary file to the given memory address.

The CPU is reset and halted before and after programming.

md *address* [*length*]
Read the specified section of device memory and display it as a canonical-style hexdump. Both arguments may be address expressions. If no length is specified, a section of the default length (64 bytes) is shown.

The output is split into three columns. The first column shows the starting address for the line. The second column lists the hexadecimal values of the bytes. The final column shows the ASCII characters corresponding to printable bytes, and . for non-printing characters.

This command supports repeat execution. If repeated, it continues to print another block of memory following that last printed.

mw *address bytes ...*
Write a sequence of bytes at the given memory address. The address given may be an address expression. Bytes values are two-digit hexadecimal numbers separated by spaces.

Unless used in the simulation mode, this command can only be used for programming flash memory.

opt [*name*] [*value*]
Query, set or list option variables. MSPDebug's behaviour can be configured using option variables, described below in the section **OPTIONS**.

Option variables may be of three types: boolean, numeric or text. Numeric values may be specified as address expressions.

With no arguments, this command displays all available option variables. With just an option name as its argument, it displays the current value of that option.

power info
Show basic power statistics gathered over the last few sessions. This includes total charge consumption, run time and average current.

power clear
Clear all recorded power statistics.

power all [*granularity*]
Show sample data gathered over all sessions. An optional granularity can be specified, in microseconds. For each time slice, relative session time, charge consumption, current consumption and approximate code location are shown.

power session *N* [*granularity*]
Same as **power all**, except that data is shown only for the *N*th session.

power export-csv *N filename*
Export raw sample data for the *N*th session to the given file in CSV format. For each line, the columns are, in order: relative time in microseconds, current consumption in microamps, memory address.

power profile
If a symbol table is loaded, compile and correlate all gathered power data against the symbol table. A single table is then shown listing, per

function, charge consumption, run time and average current. The functions are listed in order of charge consumption (biggest consumers first).

prog *filename*
Erase and reprogram the device under test using the binary file supplied. The file format will be auto-detected and may be any of the supported file formats.

In the case of a file containing symbols, symbols will be automatically loaded from the file into the symbol table (discarding any existing symbols), if they are present.

The CPU is reset and halted before and after programming.

read *filename*
Read commands from the given file, line by line and process each one. Any lines whose first non-space character is **#** are ignored. If an error occurs while processing a command, the rest of the file is not processed.

regs
Show the current value of all CPU registers in the device under test.

reset
Reset (and halt) the CPU of the device under test.

run
Start running the CPU. The interactive command prompt is blocked when the CPU is started and the prompt will not appear again until the CPU halts. The CPU will halt if it encounters a breakpoint, or if Ctrl-C is pressed by the user.

After the CPU halts, the current register values are shown as well as a disassembly of the first few instructions at the address selected by the program counter.

save_raw *address length filename*
Save a region of memory to a raw binary file. The address and length arguments may both be address expressions.

If the specified file already exists, then it will be overwritten.

set *register value*
Alter the value of a register. Registers are specified as numbers from 0 through 15. Any leading non-numeric characters are ignored (so a register may be specified as, for example, "R12"). The value argument is an address expression.

setbreak *address* [*index*]
Add a new breakpoint. The breakpoint location is an address expression. An optional index may be specified, indicating that this new breakpoint should overwrite an existing slot. If no index is specified, then the breakpoint will be stored in the next unused slot.

setwatch *address* [*index*]
Add a new watchpoint. The watchpoint location is an address expression, and an optional index may be specified. Watchpoints are considered to be a type of breakpoint and can be inspected or removed using the **break** and **delbreak** commands. Note that not all drivers support watchpoints.

setwatch_r *address* [*index*]
Add a watchpoint which is triggered only on read access.

setwatch_w *address* [*index*]

Add a watchpoint which is triggered only on write access.

simio add *class name* [*args ...*]

Add a new peripheral to the IO simulator. The *class* parameter may be any of the peripheral types named in the output of the **simio classes** command. The *name* parameter is a unique name assigned by the user to this peripheral instance, and is used with other commands to refer to this instance of the peripheral.

Some peripheral classes take arguments upon creation. These are documented in the output to the **simio help** command.

simio classes

List the names of the different types of peripherals which may be added to the simulator. You can use the **simio help** command to obtain more information about each peripheral type.

simio config *name param* [*args ...*]

Configure or perform some action on a peripheral instance. The *param* argument is specific to the peripheral type. A list of valid configuration commands can be obtained by using the **simio help** command.

simio del *name*

Remove a previously added peripheral instance. The *name* argument should be the name of the peripheral that was assigned with the **simio add** command.

simio devices

List all peripheral instances currently attached to the simulator, along with their types and interrupt status. You can obtain more detailed information for each instance with the **simio info** command.

simio help *class*

Obtain more information about a peripheral class. The documentation given will list constructor arguments and configuration parameters for the device type.

simio info *name*

Display detailed status information for a particular peripheral. The type of information displayed is specific to each type of peripheral.

step [*count*]

Step the CPU through one or more instructions. After stepping, the new register values are displayed, as well as a disassembly of the instructions at the address selected by the program counter.

An optional count can be specified to step multiple times. If no argument is given, the CPU steps once. This command supports repeat execution.

sym clear

Clear the symbol table, deleting all symbols.

sym set *name value*

Set or alter the value of a symbol. The value given may be an address expression.

sym del *name*

Delete the given symbol from the symbol table.

sym import *filename*

Load symbols from the specified file and add them to the symbol table. The file format will be auto-detected and may be either ELF32 or a BSD-style

symbol listing (like the output from nm(1)).

Symbols can be combined from many sources, as the syms command adds to the existing symbol table without discarding existing symbols.

sym import+ *filename*
This command is similar to **sym import**, except that the symbol table is not cleared first. By using this command, symbols from multiple sources can be combined.

sym export *filename*
Save all symbols currently defined to the given file. The symbols are saved as a BSD-style symbol table. Note that symbol types are not stored by MSPDebug, and all symbols are saved as type **t**.

sym find [*regex*]
Search for symbols. If a regular expression is given, then all symbols matching the expression are printed. If no expression is specified, then the entire symbol table is listed.

sym rename *regex string*
Rename symbols by searching for those matching the given regular expression and substituting the given string for the matched portion. The symbols renamed are displayed, as well as a total count of all symbols renamed.

verify *filename*
Compare the contents of the given binary file to the chip memory. If any differences are found, a message is printed for the first mismatched byte.

verify_raw *filename address*
Compare the contents of a raw binary file to the device memory at the given address. If any differences are found, a message is printed for the first mismatched byte.

BINARY FORMATS

The following binary/symbol formats are supported by MSPDebug:

ELF32
COFF
Intel HEX (program only)
BSD symbol table (symbols only)
TI Text (program only)
SREC (program only)

IO SIMULATOR

The IO simulator subsystem consists of a database of device classes, and a list of instances of those classes. Each device class has a different set of constructor arguments, configuration parameters and information which may be displayed. This section describes the operation of the available device classes in detail.

In the list below, each device class is listed, followed by its constructor arguments.

gpio Digital IO port simulator. This device simulates any of the digital ports with or without interrupt capability. It has the following configuration parameters:

base *address*
Set the base address for this port. Note that for ports without interrupt capability, the resistor enable port has a special address which is computable from the base address.

irq *vector*
Enable interrupt functionality for this port by specifying an interrupt vector number.

noirq Disable interrupt functionality for this port.

verbose Print a state change message every time the port output changes.

quiet Don't print anything when the port state changes (the default).

set *pin value*
Set the input pin state for the given pin on this port. The *pin* parameter should be an index between 0 and 7. The *value* should be either zero (for a low state) or non-zero (for a high state).

hwmult This peripheral simulates the hardware multiplier. It has no constructor or configuration parameters, and does not provide any extended information.

timer [*size*]
This peripheral simulators Timer_A modules, and can be used to simulate Timer_B modules, provided that the extended features aren't required.

The constructor takes a size argument specifying the number of capture/compare registers in this peripheral instance. The number of such registers may not be less than 2, or greater than 7.

The IO addresses and IRQs used are configurable. The default IO addresses used are those specified for Timer_A in the MSP430 hardware documentation.

base *address*
Alter the base IO address. By default, this is 0x0160. By setting this to 0x0180, a Timer_B module may be simulated.

irq0 *number*
Set the TACCR0 interrupt vector number. By default, this is interrupt vector 9. This interrupt is self-clearing, and higher priority than the TACCR1/TAIFG vector.

irq1 *number*
Set the TACCR1/TAIFG interrupt vector. By default, this is interrupt vector 8.

iv *address*
Alter the address of the interrupt vector register. By default, this is 0x012E. By setting this to 0x011E, a Timer_B module may be simulated.

set *channel value*
When Timer_A is used in capture mode, the CCI bit in each capture register reflects the state of the corresponding input pin, and can't be altered in software. This configuration command can be used to simulate changes in input pin state, and will trigger the corresponding interrupts if the peripheral is so configured.

tracer [*history-size*]
The tracer peripheral is a debugging device. It can be used to investigate and record the IO activity of a running program, to benchmark execution time, and to simulate interrupts.

The information displayed by the tracer gives a running count of clock cycles from each of the system clocks, and an instruction count. A list of the *N* most recent IO events is also displayed (this is configurable via the *history-size* argument of the constructor). Each IO event is timestamped by the number of MCLK cycles that have elapsed since the last reset of the device's counter.

The IO events that it records consist of programmed IO reads and writes, interrupt acceptance, and system resets. As well as keeping the IO events in a rotating buffer, the tracer can be configured to display the events as they occur.

Note that since clock cycles don't advance while the CPU isn't running, this peripheral can be used to calculate execution times for blocks of code. This can be achieved by setting a breakpoint at the end of the code block, setting the program counter to the start of the code block, clearing the tracer and running the code. After the breakpoint is reached, the information displayed by the tracer will contain a count of MCLK cycles elapsed during the last run.

The configuration parameters for this device class are:

verbose Start displaying IO events as they occur, as well as recording them in the rotating buffer.
quiet Stop displaying IO events as they occur, and just record them in the buffer.
trigger *irq*
Signal an interrupt request to the CPU. This request will remain raised until accepted by the CPU or cleared by the user.
untrigger
Clear a signalled interrupt request.
clear Reset the clock cycle and instruction counts to 0, and clear the IO event history.

wdt This peripheral simulates the Watchdog Timer+, which can be used in software either as a watchdog or as an interval timer. It has no constructor arguments.

The simulated state of the NMI/RST# pin can be controlled through a configuration parameter. Note that if this pin state is held low with the pin mode selected as a reset (the default), the CPU will not run.

The extended information for this peripheral shows all register states, including the hidden counter register. Configuration parameters are:

nmi *state*
: Set the NMI/RST# pin state. The argument should be zero to indicate a low state or non-zero for a high state.

irq *irq*
: Select the interrupt vector for interval timer mode. The default is to use interrupt vector 10.

ADDRESS EXPRESSIONS

Any command which accepts a memory address, length or register value as an argument may be given an address expression. An address expression consists of an algebraic combination of values.

An address value may be either a symbol name, a hex value preceded with the specifier "0x", a decimal value preceded with the specifier "0d", or a number in the default input radix (without a specifier). See the option **iradix** for more information.

The operators recognised are the usual algebraic operators: **+**, **-**, *****, **/**, **%**, **(** and **)**. Operator precedence is the same as in C-like languages, and the **-** operator may be used as a unary negation operator.

The following are all valid examples of address expressions:

```
2+2
table_start + (elem_size + elem_pad)*4
main+0x3f
__bss_end-__bss_start
```

OPTIONS

MSPDebug's behaviour can be configured via the following variables:

color (boolean)
: If true, MSPDebug will colorize debugging output.

fet_block_size (numeric)
: Change the size of the buffer used to transfer memory to and from the FET. Increasing the value from the default of 64 will improve transfer speed, but may cause problems with some chips.

enable_bsl_access (boolean)
: If set, some drivers will allow erase/program access to flash BSL memory. If in doubt, do not enable this.

enable_locked_flash_access (boolean)

If set, some drivers will allow erase/program access to the info A segment. If in doubt, do not enable this. Currently, the tilib and uif drivers are affected by this option.

gdb_default_port (numeric)
This option controls the default TCP port for the GDB server, if no argument is given to the "**gdb**" command.

gdb_loop (boolean)
Automatically restart the GDB server after disconnection. If this option is set, then the GDB server keeps running until an error occurs, or the user interrupts with Ctrl+C.

gdbc_xfer_size (numeric)
Maximum size of memory transfers for the GDB client. Increasing this value will result in faster transfers, but may cause problems with some servers.

iradix (numeric)
Default input radix for address expressions. For address values with no radix specifier, this value gives the input radix, which is 10 (decimal) by default.

quiet (boolean)
If set, MSPDebug will supress most of its debug-related output. This option defaults to false, but can be set true on start-up using the **-q** command-line option.

ENVIRONMENT

MSPDEBUG_TI3410_FW
Specifies the location of TI3410 firmware, for raw USB access to FET430UIF or eZ430 devices. This variable should contain the path to an Intel HEX file containing suitable firmware for the TI3410.

FILES

~/.mspdebug
File containing commands to be executed on startup.

ti_3410.fw.ihex
Firmware image for the TI3410 USB interface chip. This file is only required for raw USB access to FET430UIF or eZ430 devices.

SEE ALSO

nm(1), gdb(1), objcopy(1)

BUGS

If you find any bugs, you should report them to the author at dlbeer@gmail.com. It would help if you could include a transcript of an MSPDebug session illustrating the program, as well as any relevant binaries or other files.

COPYRIGHT

Copyright (C) 2009-2013 Daniel Beer <dlbeer@gmail.com>

MSPDebug is free software, distributed under the terms of the GNU General Public license (version 2 or later). See the file COPYING included with the source code for more details.

18 Jul 2013 Version 0.22