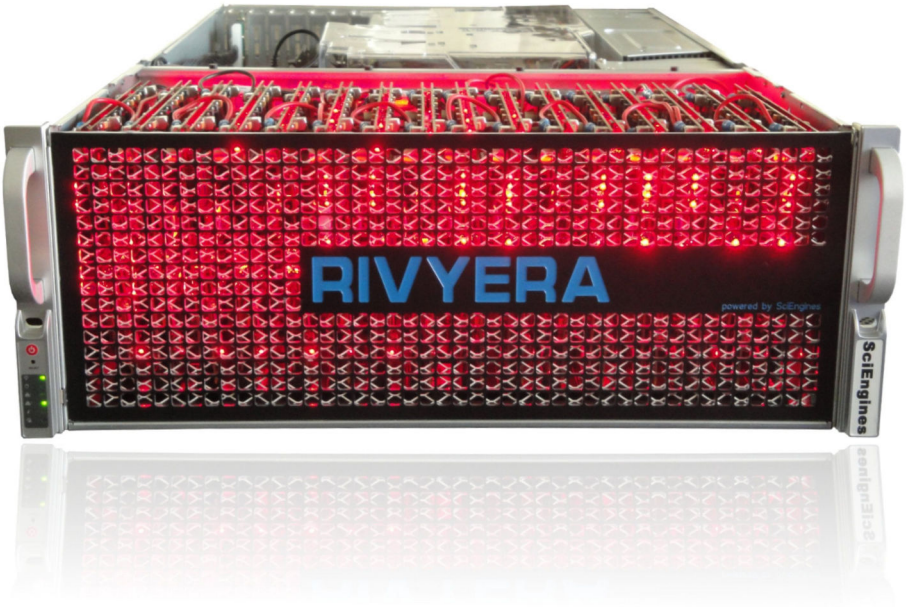
2023Q2/c94a45c0

SciEngines
massively parallel computing

SciEngines se_mon

Application User Guide

Version 1.95.10



May 16, 2023

Released version

SciEngines
Software Division

2023Q2/c94a45c0
May 16, 2023

SciEngines se_mon

Application User Guide
Version 1.95.10

SciEngines GmbH
Am Kiel-Kanal 2
24106 Kiel
Germany

Public

Released version

Contents

1 General Information

1.1 Overview

This guide introduces you to the basic features of the SciEngines monitoring tool *se_mon* and the tasks you need to perform using the command line user interface.
The primary task of *se_mon* is to enable the user to perform all SciEngines API calls in an interactive command shell. This allows a developer to test the FPGA design's behavior and communication between host and FPGAs.

1.2 Scope

The *se_mon User Documentation* steps you through the *se_mon* command line options, the shell functionality itself and the available commands.

2 Starting se_mon

2.1 Prerequisites

Before using *se_mon* the FPGA cluster should be ready for use. Please refer to the SciEngines RIVYERA User Guide to perform all steps needed to prepare the machine for operation.

2.2 Executing se_mon

Make sure you are logged in to your RIVYERA Computer. To execute *se_mon* just type `se_mon` into your favorite shell and hit <RETURN>. If your shell is unable to resolve the binary's correct location, the binary may be executed directly via typing `/opt/sciengines/current/bin/se_mon`.

When executing *se_mon* without any command line arguments, it is started in interactive mode. In interactive mode, the user is able to enter a sequence of commands at the prompt. Please refer to section 3 to learn more about the shell functionality. The *se_mon* command line prompt in the interactive mode is shown below:

```
se_mon version 1.95.10
Copyright (c) 2011-2023, SciEngines GmbH
All rights reserved.

SciEngines RIVYERA Host-API version 1.95.06 , build 1373

 -- Enter "help" to get some help. --

Machine=0 Contr=0 Slot=0 FPGA=0 >
```

2.3 Command line arguments

se_mon may be started with options that are set via command line arguments. For each option there may be either a short name or a long name used.

The possible options are as follows:

Option	Argument	Description
`-b` or `--batch`	`FILE`	Run the batch file `FILE` and exit.
`-c` or `--command`	`CMD`	Run the command `CMD` and exit. `CMD` may consist of multiple commands, separated by semicolon.
`-n` or `--no-color`		Disable colors otherwise used to highlight error and warning messages.
`-s` or `--force-stdout`		Force all warnings and errors to be printed to stdout instead of stderr.
`-i` or `--ignore-errors`		Ignore errors during batch file execution.
`-w` or `--warnings-as-errors`		Treat warnings as errors.
`-e` or `--ignore-eot`		Ignore EOT words when reading from FPGA.
`-t` or `--template`	`FILE`	Use the template file `FILE` for formatting the output. The template file's syntax is described in detail in section 4.
`--interactive`		Continue in interactive mode after processing a batch file or a command.
`-h` or `--help`		Display this help and exit.
`--version`		Display the product version and exit.

3 Shell prompt

3.1 Executing a command

There are several commands which may be executed. After starting *se_mon* in interactive mode (not using switch `-b` or `--batch`), you may enter a command by typing its name followed by optional command arguments. To execute that argument, press the <RETURN> key. Some commands produce outputs on the console, others do not. If there is no error printed the command was successfully executed. To alter a command, just use the <CURSOR LEFT> and <CURSOR RIGHT> keys to navigate within the line and standard edit keys to edit the line.

When executing, for example, the command `help`, type `help` followed by pressing the <RETURN> key. This command will print out a list of all possible commands including a short description.

3.2 Executing an API command

API commands are instructions, that use the SciEngines Host API. When running an API command it is executed based on the currently selected machine, controller, slot and FPGA. Depending on the the instruction, also the currently set timeout is used. To change the currently selected machine, controller, slot and FPGA indices, use the command `goto` (see section 5.26). The currently selected machine, controller, slot and FPGA indices are shown in the command prompt. To get or set the timeout that is used to execute an API command, use command `timeout` (see section 5.32).

This prompt, e.g., tells the user that the currently selected machine, controller, slot and FPGA indices are set to 0, 0, 1 and 2, respectively: "

```
Machine=0 Contr=0 Slot=1 FPGA=2 >
```

3.3 Command Completion

3.3.1 Completing a command name

When started interactively, command name prefixes may be completed to a command name using the <TABULATOR> key. For example, if one typed `he` and pressed the <tabulator> key, the command prefix `he` is completed to `help`. In case of ambiguities, i.e. prefixes possibly resolving to more than a single command, all possible completions are stepped through one after the other upon pressing the <TABULATOR> key. The prefix `fil`, for example, will be completed to `fileRead` at first. When pressing the <TABULATOR> key a second time `fileRead` will be replaced by `fileWrite`.

3.4 History

Each command that is interactively executed within *se_mon* is recorded in a history. This history allows the user to repeat a previous command or alter it before execution. To navigate within this history you may use the <CURSOR UP> key to get to older history entries. Use <CURSOR DOWN> to get to later entries. Also <PAGE UP> and <PAGE DOWN> may be used to get to the first and latest history entry. When quitting *se_mon*, the current history is saved into the file `.se_mon_history` located in the current user's home directory.

4 Templates

Usually, when executing the commands `readActive`, `readPassive`, `fileRead` and `waitForData -r`, all values are printed in decimal, hexadecimal and binary representation. This output format may be altered using a template file. The *se_mon* command line option `-t` or `--template` may be used to specify a default template. Additionally, the default template may be altered using the `template` command (see 5.29).

A template file is expected to be stored in ASCII format. Lines have to be separated by the newline character (\n). Each line is interpreted as one single instruction. Arguments for these instructions are separated by the tabulator character (\t). Instructions may not be split into several lines. Comments are indicated by the hash tag (#) as the line's first character. A template is applied for each single *64*-bit word separately.

The syntax for the template file (`<template_file>`) in extended Backus-Naur form is defined as follows:

```
<template_file>   := {<instr><newline>}*
<instr>           := <comment>|<value_instr>
                     |<enum_instr>|<header_instr>
                     |<line_instr>|<newline>
<comment>         := "#"<comment string>
<value_instr>     := "value"<tab>+<value_name><tab>+
                     <bit_pos><tab>+<bit_num>
                     [<tab>+<enum_identifier>]
<enum_instr>      := "enum"<tab>+<enum_identifier>
                     (<tab>+<enum_ordinal><tab>+<enum_str>)+
<header_instr>    := "header"<tab>+<format_string>
<line_instr>      := "line"<tab>+<format_string>
<bit_pos>         := 0..63
<bit_num>         := 0..63
<enum_identifier> := <string without tabulator key>
<enum_str>        := <string without tabulator key>
<enum_ordinal>    := 0..2^64-1
<value_name>      := <string without tabulator key>
<tab>             := <the tabulator character (\t)>
<newline>         := <the newline character (\n)>
<format_string>   := <string with format directives >
```

The instructions are interpreted as follows:

`value`

The first argument is used as *name string* for the `headline` instruction. The second argument specifies the word's bit offset *o*. The number of bits *n* is set by the third argument. Each current *64*-bit word is shifted by *o* bits to the right and the *n* rightmost bits are used to form a new *n*-bit value. This new value is used for the `line` instruction. In case the optional *enum identifier* is specified as fourth argument, this identifier is expected to be either defined using the `enum` instruction, or has to be a predefined *enum identifier*.

Predefined *enum identifiers* are `binary` and `index`. The `binary` identifier formats the *n*-bit value as a binary string with length *n*. When using `index` as *enum identifier*, the *n*-bit value is replaced by a *64*-bit value representing the word index within a sequence of words that is going to be printed.

`enum`

The first argument is used as *enum identifier*. All further arguments are interpreted as pairs where the first pair element is an *enum ordinal* and the second one an *enum string*. *Enum ordinals* have to be unique with respect to this enum instruction. If a value has been defined to be interpreted as enum value, the corresponding *n*-bit value is used to look up that *enum string* whose *enum ordinal* matches this *n*-bit value. The resulting *enum string* is then used as string value for the line instruction (see below).

`header`

The only argument for the header instruction is a format string similar to the one used for the commonly known printf function (see `man 3 printf`). Within this format string, all string directives are replaced by the *name strings* defined by the value instructions in the order of their definitions. In other words: the first string directive is replaced by the first value's *name string*, the second string directive is replaced by the second value's *name string* and so on. The format string may not have other directives than string directives. Furthermore, the number of string directives has to match the number of `value` definitions.

`line`

The line instruction is interpreted in a similar way as the `header` instruction. All format directives are respectively replaced by the *n*-bit value defined by each `value` instruction in the order of their definitions. A format directive may be a *long integer* directive like `%lu`, `%ld`, `%lx` or derivative directives. In case the corresponding `value` instruction refers to an *enum identifier* which is not `index`, a string directive has to be used for that value.

Within the whole template file the `header` and `line` instructions may be used once, only.

Example:

```
value No. 0 0 index
value dec 0 8
value hex 0 8
value bin 0 8 binary
value in words 0 8 enum_id_in_words

enum enum_id_in_words 0 zero 1 one 2 two
header %-7s some text    %-8s %-8s %-10s %-8s
line [%5lu]              %-8lu 0x%-06lx %-10s %-8s
```

For this example the words *0*, *1*, *2*, *3* are printed this way:

```
No.     some text    dec      hex      bin        in words
[    0]              0        0x0      00000000   zero
[    1]              1        0x1      00000001   one
[    2]              2        0x2      00000010   two
[    3]              3        0x3      00000011   UNDEF
```

The default template file used in *se_mon* is defined as follows:

```
value No. 0 0 index
value dec 0 64
value hex 0 64
value bin 56 56 8 binary
value 48 48 8 binary
value 40 40 8 binary
value 32 32 8 binary
value 24 24 8 binary
value 16 16 8 binary
value 8 8 8 binary
value 0 0 8 binary

header %-7s%21s %-18s %8s %8s %8s %8s %8s %8s %8s %8s
line [%5lu]%21lu %-18.16lX %8s %8s %8s %8s %8s %8s %8s %8s
```

For the same input words *0*, . . . , *3*, the output is:

```
No.                     dec  hex               bin   56       48
         40       32       24       16        8        0
[    0]                   0  0000000000000000  00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000000
[    1]                   1  0000000000000001  00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000001
[    2]                   2  0000000000000002  00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000010
[    3]                   3  0000000000000003  00000000 00000000
    00000000 00000000 00000000 00000000 00000000 00000011
```

5 Supported commands

Within this chapter all supported commands are described in detail. These commands may be used directly in the interactive mode of *se_mon* or within a batch file.

5.1 allocMachine

Usage:

```
allocMachine [MACHINE_INDEX]
```

Aliases: `alloc, am`
Description: Allocates the currently selected machine or the machine with optionally given machine index `MACHINE_INDEX`. The currently selected machine may be changed using the `goto` command (see section 5.26).

Example 1: This is a successful invocation for the currently selected machine with index 0:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > allocMachine
0m0.037s (allocMachine)
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

Example 2: This is an unsuccessful invocation for the currently selected machine with index 0:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > allocMachine
Machine 0 is locked for usage: user: johndoe, command: se_mon, pid: 856084
Error: SciEngines API returned "SeApiMachineInUse" (4)!
0m0.196s (allocMachine)
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

5.2 deprogram

Usage:

```
deprogram
```

Aliases: `dp`
Description: Deprograms FPGA(s) at currently selected address. See section 5.26 for changing the currently selected address.

Example: Deprogram all FPGAs on all cards.

```
Machine=0 Contr=0 Slot=* FPGA=* > deprogram
0m0.037s (deprogram)
Machine=0 Contr=0 Slot=* FPGA=* >
```

5.3 flush

Usage:

```
flush
```

Aliases: `f`
Description: Flushes currently buffered data for currently selected controller in the selected machine. `flush` waits at most as long as timeout was set, which is 1000ms by default (see section 5.32), until all buffered data has been completely written. See section 5.26 for changing the currently selected address.

Example:

```
Machine=0 Contr=0 Slot=0 FPGA=6 > flush
Machine=0 Contr=0 Slot=0 FPGA=6 >
```

5.4 freeMachine

Usage:

```
freeMachine [MACHINE_INDEX]
```

Aliases: `free, fm`
Description: Deallocates the currently selected machine or the machine with optionally given machine index `MACHINE_INDEX`. All unread words are discarded and all user FPGAs are deprogrammed. The currently selected machine may be changed using the `goto` command (see section 5.26).

Example:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > freeMachine
0m0.015s (freeMachine)
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

5.5 getControllerCount

Usage:

```
getControllerCount [MACHINE_INDEX]
```

Aliases: `ControllerCount, cc`
Description: Returns the number of controllers for the currently selected machine or the machine with optionally given machine index `MACHINE_INDEX`. The currently selected machine may be changed using the `goto` command (see section 5.26).

Example:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > getControllerCount
1
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

5.6 getControllerInfo

Usage:

```
getControllerInfo [CONTROLLER_INDEX]
```

Aliases: `ControllerInfo, ci`
Description: Returns information about the currently selected controller within the currently selected machine. Optionally, a controller index `CONTROLLER_INDEX` may be specified. The currently selected machine may be changed using the `goto` command (see section 5.26).

Example: Get first controller's information.

```
Machine=0 Contr=0 Slot=0 FPGA=0 > getControllerInfo
-- Driver name            : remote
-- Machine Slot           : 0
-- Serial                 : 0xC57
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

5.7 getFPGACount

Usage:

```
getFPGACount
```

Aliases: `FPGACount, fc`
Description: Returns the number of FPGAs for the currently selected machine and slot. See section 5.26 for changing the currently selected address.

Example:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > getFPGACount
8
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

5.8 getFPGAInfo

Usage:

```
getFPGAInfo
```

Aliases: `FPGAInfo, fi`
Description: Returns information about the FPGA at the currently selected address. See section 5.26 for changing the currently selected address.

Example:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > getFPGAInfo
-- Type                  : XC6SLX150-3FGG676
-- Programmed            : true
-- Firmware version      : 01.92.01
-- Firmware build        : 1129
0m0.008s (getFPGAInfo)
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

5.9 getMachineCount

Usage:

```
getMachineCount
```

Aliases: `MachineCount, mc`
Description: Returns the number of machines.

Example:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > getMachineCount
1
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

5.10 getSlotCount

Usage:

```
getSlotCount [MACHINE_INDEX]
```

Aliases: `SlotCount, sc`
Description: Returns the number of slots in the currently selected machine or the machine with optionally given machine index `MACHINE_INDEX`. The currently selected machine may be changed using the `goto` command (see section 5.26).

Example:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > getSlotCount
16
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

5.11 getSlotInfo

Usage:

```
getSlotInfo
```

Aliases: `SlotInfo, si`
Description: Returns information about the card at the currently selected slot and machine address. See section 5.26 for changing the currently selected address.

Example:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > getSlotInfo
-- Serial                 : 0xc57
-- FPGA Count             : 8
-- is Controller          : true
-- Controller index       : 0
-- Prev Controller index  : 0
-- Next Controller index  : 0
-- Firmware version       : 01.91.11
-- Firmware build         : 1091
-- Hardware revision      : 3.7
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

5.12 getProgInfo

Usage:

```
getProgInfo
```

Aliases: `ProgInfo, pi`
Description: Returns last programming information about the card at the currently selected slot and machine address. See section 5.26 for changing the currently selected address.

5.13 getTemperature

Usage:

```
getTemperature
```

Aliases: `temperature, temp`
Description: Returns a card's current temperature for the currently selected slot and machine address as well as the highest temperature ever measured. See section 5.26 for changing the currently selected address.

Example: Get the current and highest recorded temperatures for the currently selected machine index and slot address.

```
Machine=0 Contr=0 Slot=0 FPGA=0 > getTemperature
current: 20.0
max:     69.0
0m0.003s (getTemperature)
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

5.14 program

Usage:

```
program PROGRAM_FILE
```

Aliases: `p`
Description: Program FPGA(s) with `PROGRAM_FILE` at currently selected address. `PROGRAM_FILE` may be a .bit file, an .rbt or a .sim file. See section 5.26 for changing the currently selected address. Please keep in mind that all FPGAs on a slot have to be programmed since the FPGAs form a ring on a card and would be unable to communicate otherwise.

Example 1: Programming only one FPGA will result in an error because the card's ring is not set up completely.

```
Machine=0 Contr=0 Slot=0 FPGA=0 > program pingpong_top.bit
Programming a single FPGA.
[SciEngines RIVYERA API] FPGAs at slot 0 in machine 0 are programmed but do not react
     (after 2160392 bytes)!
Error: SciEngines API returned "SeApiFailed" (1)!
0m5.643s (program)
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

Example 2: Use goto (see section 5.26) to set all slots and all FPGAs as target.

```
Machine=0 Contr=0 Slot=0 FPGA=0 > goto * *
Machine=0 Contr=0 Slot=* FPGA=* > program pingpong_top.bit
0m0.568s (program)
Machine=0 Contr=0 Slot=* FPGA=* >
```

5.15 readActive

Usage:

```
readActive REG_ADDRESS NUM_WORDS [FILE [--overwrite|--append|--cancel]]
```

Aliases: `ra`
Description: Reads `NUM_WORDS` 64bit word(s) from given source register address `REG_ADDRESS`, from currently selected address using the active mode. See section 5.26 for changing the currently selected address. If no data is present, `readActive` waits at most as long as the timeout was set, which is 1000ms by default (see section 5.32). If `FILE` is provided, the read words are saved to file `FILE`. If `FILE` already exists, the user is interactively asked to append the data to the file, overwrite it or to cancel. If `FILE` is not provided, the read words are printed out in decimal, hexadecimal and binary form.

Example: Actively reading ten words from currently set address, register 0.

```
Machine=0 Contr=0 Slot=0 FPGA=6 > readActive 0 10
No.                      dec  hex               bin   56       48       40       32
     24       16        8        0
[    0]                    0  0000000000000000  00000000 00000000 00000000 00000000
     00000000 00000000 00000000 00000000
[    1]  1736515902335221724  181958034181FFDC  00011000 00011001 01011000 00000011
     01000001 10000001 11111111 11011100
[    2]  3473031804670443449  3032B0068303FFB9  00110000 00110010 10110000 00000110
     10000011 00000011 11111111 10111001
[    3]  6946063609340886898  6065600D0607FF72  01100000 01100101 01100000 00001101
     00000110 00000111 11111111 01110010
[    4] -4554616855027777820  C0CAC01A0C0FFEE4  11000000 11001010 11000000 00011010
     00001100 00001111 11111110 11100100
[    5] -9109233710055555639  81958034181FFDC9  10000001 10010101 10000000 00110100
     00011000 00011111 11111101 11001001
[    6]   228276653598440338  032B0068303FFB92  00000011 00101011 00000000 01101000
     00110000 00111111 11111011 10010010
[    7]   456553307196880677  065600D0607FF725  00000110 01010110 00000000 11010000
     01100000 01111111 11110111 00100101
[    8]   913106614393761355  0CAC01A0C0FFEE4B  00001100 10101100 00000001 10100000
     11000000 11111111 11101110 01001011
[    9]  1826213228787522710  1958034181FFDC96  00011001 01011000 00000011 01000001
     10000001 11111111 11011100 10010110
Read 10 words.
0m0.003s (readActive)
Machine=0 Contr=0 Slot=0 FPGA=6 >
```

5.16 readPassive

Usage:

```
readPassive REG_ADDRESS NUM_WORDS [FILE [--overwrite|--append|--cancel]]
```

Aliases: `read, r, rp`
Description: Reads `NUM_WORDS` 64bit word(s) from given source register address `REG_ADDRESS`, from currently selected address using the passive mode. See section 5.26 for changing the currently selected address. If no data is present, `readPassive` waits at most as long as the timeout was set, which is 1000ms by default (see section 5.32). If `FILE` is provided, the read words are saved to file `FILE`. If `FILE` already exists, the user is interactively asked to append the data to the file, overwrite it or to cancel. If `FILE` is not provided, the read words are printed out in decimal, hexadecimal and binary form.

Example: Passively reading five words from currently set address, register 2.

```
Machine=0 Contr=0 Slot=0 FPGA=6 > readPassive 2 5
No.                      dec  hex               bin   56       48       40       32
     24       16        8        0
[    0]                17544  0000000000004488  00000000 00000000 00000000 00000000
     00000000 00000000 01000100 10001000
[    1]                48815  000000000000BEAF  00000000 00000000 00000000 00000000
     00000000 00000000 10111110 10101111
[    2]                   42  000000000000002A  00000000 00000000 00000000 00000000
     00000000 00000000 00000000 00101010
[    3]                   47  000000000000002F  00000000 00000000 00000000 00000000
     00000000 00000000 00000000 00101111
[    4]                57050  0000000000000DEDA  00000000 00000000 00000000 00000000
     00000000 00000000 11011110 11011010
Read 5 words.
Machine=0 Contr=0 Slot=0 FPGA=6 >
```

5.17 readRequest

Usage:

```
readRequest REG_ADDRESS NUM_WORDS
```

Aliases: `rr`
Description: Requests to read `NUM_WORDS` 64bit word(s) from given register address `REG_ADDRESS` from currently selected address. See section 5.26 for changing the currently selected address. The FPGA is then instructed to send `NUM_WORDS` 64bit word(s) back to the host who may read the data via `readPassive` (see section 5.16). A `readRequest` in conjunction with `readPassive` is equal to `readActive`.

Example: Send two read requests each with three words to currently set address, register 0 and passively read the replies afterwards.

```
Machine=0 Contr=0 Slot=0 FPGA=6 > readRequest 0 5
Machine=0 Contr=0 Slot=0 FPGA=6 >
Machine=0 Contr=0 Slot=0 FPGA=6 > readRequest 0 5
Machine=0 Contr=0 Slot=0 FPGA=6 >
Machine=0 Contr=0 Slot=0 FPGA=6 > readPassive 0 10
No.                      dec  hex               bin   56       48       40       32
     24       16        8        0
[    0]                   0  0000000000000000  00000000 00000000 00000000 00000000
     00000000 00000000 00000000 00000000
[    1]  1736515902335221724  181958034181FFDC  00011000 00011001 01011000 00000011
     01000001 10000001 11111111 11011100
[    2]  3473031804670443449  3032B0068303FFB9  00110000 00110010 10110000 00000110
     10000011 00000011 11111111 10111001
[    3]  6946063609340886898  6065600D0607FF72  01100000 01100101 01100000 00001101
     00000110 00000111 11111111 01110010
[    4] -4554616855027777820  C0CAC01A0C0FFEE4  11000000 11001010 11000000 00011010
     00001100 00001111 11111110 11100100
[    5]                   0  0000000000000000  00000000 00000000 00000000 00000000
     00000000 00000000 00000000 00000000
[    6] -9109233710055555639  81958034181FFDC9  10000001 10010101 10000000 00110100
     00011000 00011111 11111101 11001001
[    7]   228276653598440338  032B0068303FFB92  00000011 00101011 00000000 01101000
     00110000 00111111 11111011 10010010
[    8]   456553307196880677  065600D0607FF725  00000110 01010110 00000000 11010000
     01100000 01111111 11110111 00100101
```

```
[    9]   9131066143937613 55  0CAC01A0C0FFEE4B  00001100 10101100 00000001 10100000
     11000000 11111111 11101110 01001011
Read 10 words.
0m0.003s (readPassive)
Machine=0 Contr=0 Slot=0 FPGA=6 >
```

5.18 waitForData

Usage:

```
waitForData [-g][-r [FILE] [-a] [-n MAX_NUM_WORDS] [--overwrite|--append|--cancel]]
```

Aliases: `wfd`
Description: Waits for incoming data at the currently selected controller and machine and returns the address the data originates from and the number of 64-bit words. If no data is present, `waitForData` waits at most as long as the timeout was set, which is 1000ms by default (see section 5.32). The currently selected machine may be changed using the `goto` command (see section 5.26). See section 5.26 for changing the currently selected address. Keep in mind there may have been more words available than indicated by `waitForData`, because more data may have arrived just after the command's completion. If the `-g` switch is set, `waitForData` switches the current address to the returned address. When providing the `-r` switch, all data is read from the returned address and written to the file `FILE` if provided or otherwise printed out. If provided `FILE` already exists, the user is interactively asked to append the data to the file, overwrite it or to cancel. If, additionally to `-r`, also `-a` is provided, `waitForData` is executed in a loop until there is no more data present within the timeout. Option `-n` makes `waitForData` read at most `MAX_NUM_WORDS` words if also `-r` is specified.

Example 1: In this example there are five 64-bit words at machine 0, controller 0, slot 0, FPGA 6 and data register 2.

```
Machine=0 Contr=0 Slot=0 FPGA=0 > waitForData
m0 c0 s0 f6 r2 with 5 words
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

Example 2: Use the `-g` parameter to instruct `waitForData` to change the currently selected address to the one with incoming data.

```
Machine=0 Contr=0 Slot=0 FPGA=0 > waitForData -g
m0 c0 s0 f6 r2 with 5 words
Machine=0 Contr=0 Slot=0 FPGA=6 >
```

5.19 write

Usage:

```
write REG_ADDRESS [VALUE [VALUE ...]]
```

```
write REG_ADDRESS {--file|-f} FILE [-o OFFSET] [-n NUM_WORDS]
```

Aliases: `w`
Description: Writes one or more values to given register address `REG_ADDRESS` at currently selected address using the `write` command (which is CMD_WR in the FPGA design). Alternatively, a file's content may be sent if `FILE` is provided. If the target is not ready to receive data or there is much data to be transferred, it might be necessary to set the timeout to a higher value (see section 5.32). If options `-o` or `-n` are specified, then `OFFSET` words are skipped from file and at most `NUM_WORDS` are written. See section 5.26 for changing the currently selected address.

Example 1:

```
Machine=0 Contr=0 Slot=0 FPGA=6 > write 2 0x4488
Wrote 1 words.
Machine=0 Contr=0 Slot=0 FPGA=6 >
```

Example 2:

```
Machine=0 Contr=0 Slot=0 FPGA=6 > write 2 -f dumpfile
Wrote 4 words.
Machine=0 Contr=0 Slot=0 FPGA=6 >
```

5.20 alias

Usage:

```
alias [COMMAND]
```

Aliases: `a`
Description: Most commands have aliases to make them shorter and easier to handle. When executing this command without argument, a list of all commands is printed out. Otherwise only the alias for `COMMAND` is printed. Aliases are predefined and may not be altered, created or deleted.

Example 1: This is an invocation with `allocMachine` as command argument:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > alias allocMachine
    allocMachine: alloc am
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

Example 2: This is an invocation without any command argument:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > alias
    allocMachine        : alloc am
    deprogram           : dp
    flush               : f
    freeMachine         : free fm
    getControllerCount  : ControllerCount cc
    getControllerInfo   : ControllerInfo ci
    getFPGACount        : FPGACount fc
    getFPGAInfo         : FPGAInfo fi
    getMachineCount     : MachineCount mc
    getSlotCount        : SlotCount sc
    getSlotInfo         : SlotInfo si
    getProgInfo         : ProgInfo pi
    getTemperature      : temperature temp
    program             : p
    readActive          : ra
    readPassive         : read r rp
    readRequest         : rr
    waitForData         : wfd
    write               : w
    alias               : a
    batch               : b
    batchLoop           : bl
    breakPoint          : break bp
    fileRead            : fr
    fileWrite           : fw
    goto                : g
    help                : h ?
    options             : o
    template            :
    quit                : q exit
    sleep               :
    timeout             : t
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

5.21 batch

Usage:

```
batch [BATCH_FILE]
```

Aliases: `b`
Description: Executes `BATCH_FILE` as batch file. Batch files are plaintext ASCII files containing a batch of commands that are all executed one by one. If an error occurs, the execution is interrupted unless *se_mon* was started with switch `--ignoreerrors`. Batch files may also be executed using the command line argument `--batch` at *se_mon* start time.

Example:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > batch test.batch
Machine=0 Contr=0 Slot=0 FPGA=0 > # select first machine, first controller, first slot,
     all FPGAs
Machine=0 Contr=0 Slot=0 FPGA=0 > goto 0 0 0 *
Machine=0 Contr=0 Slot=0 FPGA=* > # allocate currently selected machine
Machine=0 Contr=0 Slot=0 FPGA=* > allocMachine
0m0.034s (allocMachine)
Machine=0 Contr=0 Slot=0 FPGA=* > # program currently selected FPGAs
Machine=0 Contr=0 Slot=0 FPGA=* > # (see previous goto command)
Machine=0 Contr=0 Slot=0 FPGA=* > program pingpong_top.bit
0m0.575s (program)
Machine=0 Contr=0 Slot=0 FPGA=* > # select third programmed FPGAs
Machine=0 Contr=0 Slot=0 FPGA=* > goto 2
Machine=0 Contr=0 Slot=0 FPGA=2 > # write the word 0xdead to currently selected
Machine=0 Contr=0 Slot=0 FPGA=2 > # FPGA, first data register
Machine=0 Contr=0 Slot=0 FPGA=2 > write 0 0xdeda
Wrote 1 words.
Machine=0 Contr=0 Slot=0 FPGA=2 > # wait for incoming data at controller with
Machine=0 Contr=0 Slot=0 FPGA=2 > # index 0 at currently selected machine waitForData 0
Machine=0 Contr=0 Slot=0 FPGA=2 > # read one 64bit word from currently selected
Machine=0 Contr=0 Slot=0 FPGA=2 > # FPGA address, first data register
Machine=0 Contr=0 Slot=0 FPGA=2 > read 0 1
No.                      dec  hex               bin   56       48       40       32
     24       16        8        0
[    0]                57050  000000000000DEDA  00000000 00000000 00000000 00000000
     00000000 00000000 11011110 11011010
Read 1 words.
0m0.003s (readPassive)
Machine=0 Contr=0 Slot=0 FPGA=2 > # select first machine, first controller, first slot,
     all FPGAs
Machine=0 Contr=0 Slot=0 FPGA=2 > goto 0 0 0 *
Machine=0 Contr=0 Slot=0 FPGA=* > # deprogram currently selected FPGAs
Machine=0 Contr=0 Slot=0 FPGA=* > deprogram
0m0.035s (deprogram)
Machine=0 Contr=0 Slot=0 FPGA=* > # free currently selected  machine
Machine=0 Contr=0 Slot=0 FPGA=* > freeMachine
0m0.015s (freeMachine)
Machine=0 Contr=0 Slot=0 FPGA=* >
0m0.662s (batch)
Machine=0 Contr=0 Slot=0 FPGA=* >
```

5.22 batchLoop

Usage:

```
batchLoop BATCH_FILE [NUM_CALLS]
```

Aliases: `bl`
Description: Executes `BATCH_FILE` as batch file in a loop. If `NUM_CALLS` is provided, the loop is performed `NUM_CALLS` times, else the number of loops is infinite. If `NUM_CALLS` is 1, this call is equal to command `batch` (see section 5.21).

Example:

```
Machine=0 Contr=0 Slot=0 FPGA=* > batchLoop test.batch 2
0|Machine=0 Contr=0 Slot=0 FPGA=* > # select first machine, first controller, first slot,
     all FPGAs
0|Machine=0 Contr=0 Slot=0 FPGA=* > goto 0 0 0 *
0|Machine=0 Contr=0 Slot=0 FPGA=* > # allocate currently selected machine
0|Machine=0 Contr=0 Slot=0 FPGA=* > allocMachine
0m0.034s (allocMachine)
0|Machine=0 Contr=0 Slot=0 FPGA=* > # program currently selected FPGAs
0|Machine=0 Contr=0 Slot=0 FPGA=* > # (see previous goto command)
0|Machine=0 Contr=0 Slot=0 FPGA=* > program pingpong_top.bit
0m0.598s (program)
0|Machine=0 Contr=0 Slot=0 FPGA=* > # select third programmed FPGAs
0|Machine=0 Contr=0 Slot=0 FPGA=* > goto 2
0|Machine=0 Contr=0 Slot=0 FPGA=2 > # write the word 0xdead to currently selected
0|Machine=0 Contr=0 Slot=0 FPGA=2 > # FPGA, first data register
0|Machine=0 Contr=0 Slot=0 FPGA=2 > write 0 0xdeda
Wrote 1 words.
0|Machine=0 Contr=0 Slot=0 FPGA=2 > # wait for incoming data at controller with
0|Machine=0 Contr=0 Slot=0 FPGA=2 > # index 0 at currently selected machine waitForData 0
0|Machine=0 Contr=0 Slot=0 FPGA=2 > # read one 64bit word from currently selected
0|Machine=0 Contr=0 Slot=0 FPGA=2 > # FPGA address, first data register
0|Machine=0 Contr=0 Slot=0 FPGA=2 > read 0 1
No.                      dec  hex               bin   56       48       40       32
     24       16        8        0
[    0]                57050  000000000000DEDA  00000000 00000000 00000000 00000000
     00000000 00000000 11011110 11011010
Read 1 words.
0m0.004s (readPassive)
0|Machine=0 Contr=0 Slot=0 FPGA=2 > # select first machine, first controller, first slot,
     all FPGAs
0|Machine=0 Contr=0 Slot=0 FPGA=2 > goto 0 0 0 *
0|Machine=0 Contr=0 Slot=0 FPGA=* > # deprogram currently selected FPGAs
0|Machine=0 Contr=0 Slot=0 FPGA=* > deprogram
0m0.035s (deprogram)
0|Machine=0 Contr=0 Slot=0 FPGA=* > # free currently selected  machine
0|Machine=0 Contr=0 Slot=0 FPGA=* > freeMachine
0m0.004s (freeMachine)
0|Machine=0 Contr=0 Slot=0 FPGA=* >
1|Machine=0 Contr=0 Slot=0 FPGA=* > # select first machine, first controller, first slot,
     all FPGAs
1|Machine=0 Contr=0 Slot=0 FPGA=* > goto 0 0 0 *
1|Machine=0 Contr=0 Slot=0 FPGA=* > # allocate currently selected machine
1|Machine=0 Contr=0 Slot=0 FPGA=* > allocMachine
0m0.036s (allocMachine)
1|Machine=0 Contr=0 Slot=0 FPGA=* > # program currently selected FPGAs
1|Machine=0 Contr=0 Slot=0 FPGA=* > # (see previous goto command)
1|Machine=0 Contr=0 Slot=0 FPGA=* > program pingpong_top.bit
0m0.600s (program)
1|Machine=0 Contr=0 Slot=0 FPGA=* > # select third programmed FPGAs
1|Machine=0 Contr=0 Slot=0 FPGA=* > goto 2
1|Machine=0 Contr=0 Slot=0 FPGA=2 > # write the word 0xdead to currently selected
1|Machine=0 Contr=0 Slot=0 FPGA=2 > # FPGA, first data register
1|Machine=0 Contr=0 Slot=0 FPGA=2 > write 0 0xdeda
Wrote 1 words.
1|Machine=0 Contr=0 Slot=0 FPGA=2 > # wait for incoming data at controller with
1|Machine=0 Contr=0 Slot=0 FPGA=2 > # index 0 at currently selected machine waitForData 0
1|Machine=0 Contr=0 Slot=0 FPGA=2 > # read one 64bit word from currently selected
1|Machine=0 Contr=0 Slot=0 FPGA=2 > # FPGA address, first data register
1|Machine=0 Contr=0 Slot=0 FPGA=2 > read 0 1
No.                      dec  hex               bin   56       48       40       32
     24       16        8        0
[    0]                57050  000000000000DEDA  00000000 00000000 00000000 00000000
     00000000 00000000 11011110 11011010
Read 1 words.
0m0.003s (readPassive)
1|Machine=0 Contr=0 Slot=0 FPGA=2 > # select first machine, first controller, first slot,
     all FPGAs
1|Machine=0 Contr=0 Slot=0 FPGA=2 > goto 0 0 0 *
1|Machine=0 Contr=0 Slot=0 FPGA=* > # deprogram currently selected FPGAs
1|Machine=0 Contr=0 Slot=0 FPGA=* > deprogram
0m0.035s (deprogram)
1|Machine=0 Contr=0 Slot=0 FPGA=* > # free currently selected  machine
1|Machine=0 Contr=0 Slot=0 FPGA=* > freeMachine
0m0.004s (freeMachine)
1|Machine=0 Contr=0 Slot=0 FPGA=* >
0m1.353s (batchLoop)
Machine=0 Contr=0 Slot=0 FPGA=* >
```

5.23 breakPoint

Usage:

```
breakPoint [DISPLAY_MESSAGE]
```

Aliases: `break, bp`
Description: Sets a breakpoint that may be used within batch files. If a batch file execution reaches this `breakPoint` command, its `DISPLAY_MESSAGE` is printed out, if provided. The user is then interactively asked whether the execution should be continued or not. If the execution shall not be continued, only the currently executed batch file is stopped. If the currently executed batch file was executed by another batch file, then the other batch file is continued. Also, if the currently executed batch file was executed by `batchLoop` command (see section 5.22), the current iteration is stopped, only.

Example:

```
Machine=0 Contr=0 Slot=0 FPGA=* > breakPoint This is a sample text describing this break
    point
This is a sample text describing this break point
Continue batch execution? [y]es, [n]o: y
Machine=0 Contr=0 Slot=0 FPGA=* >
```

5.24 fileRead

Usage:

```
fileRead FILE [-o OFFSET] [-n NUM_WORDS]
```

Aliases: `fr`
Description: Reads 64-bit word(s) from file `FILE` and prints them out in decimal, hexadecimal and binary form. If options `-o` or `-n` are specified, then `OFFSET` words are skipped from file and at most `NUM_WORDS` are printed.

Example:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > fileRead dumpfile
No.                      dec  hex               bin   56       48       40       32
    24       16        8        0
[    0]                48815  000000000000BEAF  00000000 00000000 00000000 00000000
    00000000 00000000 10111110 10101111
[    1]                   42  000000000000002A  00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00101010
[    2]                   47  000000000000002F  00000000 00000000 00000000 00000000
    00000000 00000000 00000000 00101111
[    3]                57050  000000000000DEDA  00000000 00000000 00000000 00000000
    00000000 00000000 11011110 11011010
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

5.25 fileWrite

Usage:

```
fileWrite FILE [--overwrite|--append|--cancel] VALUE [VALUE [VALUE [VALUE ...]]]
```

Aliases: `fw`
Description: Writes 64-bit word(s) to file `FILE`. If `VALUE` starts with 0x, it is interpreted as hexadecimal, else it is interpreted as decimal 64-bit word. If `FILE` already exists, the user is interactively asked to append the data to the file, overwrite it or to cancel.

Example 1:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > fileWrite dumpfile 0xbeaf 42
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

Example 2:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > fileWrite dumpfile 47 0xdeda
File already present. [o]verwrite, [a]ppend, [c]ancel: a
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

5.26 goto

Usage:

```
goto MACHINE_INDEX CONTR_INDEX SLOT_ADDRESS FPGA_ADDRESS
```

```
goto CONTR_INDEX SLOT_ADDRESS FPGA_ADDRESS
```

```
goto SLOT_ADDRESS FPGA_ADDRESS
```

```
goto FPGA_ADDRESS
```

Aliases: `g`
Description: Sets the currently selected machine and FPGA address. If all four arguments are provided, the currently selected machine and controller indices as well as the slot and FPGA addresses are, respectively, set to `MACHINE_INDEX`, `CONTR_INDEX`, `SLOT_ADDRESS` and `FPGA_ADDRESS`. If three arguments are provided, the currently selected controller index, as well as the slot and FPGA addresses are set to `CONTR_INDEX`, `SLOT_ADDRESS` and `FPGA_ADDRESS`, respectively. If two arguments are provided, the currently selected slot and FPGA addresses are set to `SLOT_ADDRESS` and `FPGA_ADDRESS`, while the currently selected machine and controller indices are not changed. If one argument is provided, the currently selected FPGA address is set to `FPGA_ADDRESS`, while the currently selected machine and controller indices as well as the slot address remain unchanged. `CONTR_INDEX`, `SLOT_ADDRESS` and/or `FPGA_ADDRESS` may be an asterisk (*), representing a wildcard addressing all controllers, slots and/or FPGAs. Attention: The `goto` command does not check if an address is valid. When setting an invalid address an error will occur when first executing an API command which uses this invalid address.

Example 1: Set the current address to be machine index 1 (second machine), controller index 0 (first controller), slot address 2 (third card), FPGA address 3 (fourth FPGA).

```
Machine=0 Contr=0 Slot=* FPGA=* > goto 1 0 2 3
Machine=1 Contr=0 Slot=2 FPGA=3 >
```

Example 2: Set the current address to be slot address 5 (sixth card), FPGA address 0 (first FPGA) while leaving the machine and controller indices unchanged.

```
Machine=1 Contr=0 Slot=2 FPGA=3 > goto 5 0
Machine=1 Contr=0 Slot=5 FPGA=0 >
```

Example 3: Set the current address to be machine 0 (first machine), controller index 0 (first controller), all slots, all FPGAs.

```
Machine=1 Contr=0 Slot=5 FPGA=0 > goto 0 0 * *
Machine=0 Contr=0 Slot=* FPGA=* >
```

5.27 help

Usage:

```
help [COMMAND]
```

Aliases: `h, ?`
Description: If `COMMAND` is provided, usage information about `COMMAND` is printed out. If `COMMAND` is not provided, a list of all commands and a short description is printed out.

Example 1: This is an invocation with `allocMachine` as command argument:

```
Machine=0 Contr=0 Slot=0 FPGA=* > help allocMachine
usage:
allocMachine [MACHINE_INDEX]
aliases:
alloc, am
description:
Allocates the currently selected machine or the machine with optionally given machine
     index MACHINE_INDEX. The currently selected machine may be changed using the goto
     command (see "help goto").
Machine=0 Contr=0 Slot=0 FPGA=* >
```

Example 2: This is an invocation without any command argument:

```
Machine=0 Contr=0 Slot=0 FPGA=* > help
available commands:
    allocMachine        Allocates a machine.
    deprogram           Deprograms an FPGA.
    flush               Flushes buffered data
    freeMachine         Deallocates a machine.
    getControllerCount  Returns the number of controllers for a machine.
    getControllerInfo   Returns information about a controller.
    getFPGACount        Returns the number of FPGAs in a machine.
    getFPGAInfo         Returns information about an FPGA.
    getMachineCount     Returns the number of machines.
    getSlotCount        Returns the number of slots in a machine.
    getSlotInfo         Returns information about a slot.
    getProgInfo         Returns last programming information.
    getTemperature      Returns the temperature for the selected card.
    program             Programs FPGA(s) with program file.
    readActive          Reads from a register address using the active mode.
    readPassive         Reads from a register address using the passive mode.
    readRequest         Writes a read request.
    waitForData         Waits for incoming data.
    write               Writes to a data register using write command.
    alias               Prints the command aliases.
    batch               Executes a batch file.
    batchLoop           Executes a batch file in a loop.
    breakPoint          Sets a breakpoint.
    fileRead            Reads values from a file and displays them.
    fileWrite           Writes values to a file.
    goto                Sets current address.
    help                Prints help page.
    options             Gets or sets options used for allocMachine.
    template            Changes the template used for printing out values.
    quit                Quits se_mon.
    sleep               Sleeps for given time.
    timeout             Gets or sets timeout.
use help COMMAND to get usage information about COMMAND
Machine=0 Contr=0 Slot=0 FPGA=* >
```

5.28 options

Usage:

```
options [SYNC|ASYNC] [NORMAL]
```

Aliases: `o`
Description: Invoking options command without arguments, the currently set options are printed out. To change write behavior to be synchronous or asynchronous, the strings `"SYNC"` or `"ASYNC"` may be used as argument. To change the routing behavior, the string `"NORMAL"` may be used as argument. Please refer to the SciEngines Host API documentation to get to know more about the different write behaviors and routing methods.

These options will have an effect when allocating a machine, only. An already allocated machine needs to be freed and allocated again for changes to take effect.

Example 1: Setting the write behavior to be synchronous will take effect the next time `allocMachine` is used:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > options sync
```

Example 2: Without an argument, all currently set options are printed out:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > options
write behavior:     SYNC
routing method:     NORMAL
```

5.29 template

Usage:

```
template [TEMPLATE_FILE|--reset|-r]
```

Aliases:
Description: Sets or resets the template file. A template is used for printing values in a user defined format. If no argument is provided `template` returns the currently set template file. If otherwise `--reset` or `-r` is provided the template is reset to its default. When providing a valid template file `TEMPLATE_FILE` its template definition will be used within the commands `readActive`, `readPassive`, `fileRead` and `waitForData -r`. A template file is valid if it complies with the syntax described in the template section within the *se_mon* user documentation.

5.30 quit

Usage:

```
quit
```

Aliases: `q, exit`
Description: Quits *se_mon*. A shortcut to `quit` is <CTRL>-<d> on an empty line.

Example:

```
Machine=0 Contr=0 Slot=0 FPGA=* > quit
```

5.31 sleep

Usage:

```
sleep TIME_IN_MS
```

Aliases:
Description: Sleeps for `TIME_IN_MS` milliseconds and blocks any interaction while sleeping.

Example: Sleep for three seconds.

```
Machine=0 Contr=0 Slot=0 FPGA=6 > sleep 3000
0m3.000s (sleep)
Machine=0 Contr=0 Slot=0 FPGA=6 >
```

5.32 timeout

Usage:

```
timeout [TIME_IN_MS]
```

Aliases: `t`
Description: If `TIME_IN_MS` is provided, the new timeout value is set to `TIME_IN_MS` milliseconds. If `TIME_IN_MS` is not provided, the currently set timeout is printed out. To set timeout to be infinite, use an asterisk (*) for `TIME_IN_MS`. By default, timeout is set to 1000 milliseconds.

Example: Get the currently set timeout, set it to three seconds and get the currently set timeout again.

```
Machine=0 Contr=0 Slot=0 FPGA=0 > timeout
2000
Machine=0 Contr=0 Slot=0 FPGA=0 > timeout 3000
Machine=0 Contr=0 Slot=0 FPGA=0 > timeout
3000
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

6 Third Party Licenses

se_mon uses the linenoise library which has been published under this license:

```
Copyright (c) 2010-2014, Salvatore Sanfilippo <antirez at gmail dot com>
Copyright (c) 2010-2013, Pieter Noordhuis <pcnoordhuis at gmail dot com>

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice,
  this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

Imprint:

SciEngines GmbH
Am Kiel-Kanal 2
D-24106 Kiel Germany

Phone:	+49(0)431-9086-2000
Fax:	+49(0)431-9086-2009
E-Mail:	info@SciEngines.com
Internet:	www.SciEngines.com

CEO: Gerd Pfeiffer

Commercial Register:	Amtsgericht Kiel
Commercial Register No.:	HR B 9565 KI
VAT-Identification Number:	DE 814955925