RIVYERA

se_mon User Documentation

Version 1.91.59

SciEngines GmbH
Fraunhoferstrasse 13
24118 Kiel
Germany

www.sciengines.com

Revision: 1.91.59, 22. Januar 2014

Inhaltsverzeichnis

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 Comand 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 behaviour 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 to be logged in onto 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 commandline arguments, it is started in the interactive mode. In the interactive mode, the user is able to enter several commands into a promt. Please refer to section 3 to learn more about the shell functionality. The *se_mon* command line prompt in the interactive mode looks like this:

```
se_mon version 1.91.59
Copyright (c) 2011-2014, SciEngines GmbH

SciEngines Rivyera Host-API version 1.91.11 B12, build 1052

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

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`	execute a batch file `FILE` and exit.
`-c` or `--command`	`CMD`	execute command `CMD` and exit.
`-n` or `--nocolor`		disable colors used to highlight error and warning messages.
`-s` or `--force-stdout`		forces all warnings and errors to be printed to stdout instead of stderr.
`-i` or `--ignoreerrors`		ignore errors during batch file execution.
`-t` or `--template`	`FILE`	use template file `FILE` for formatting the output.
`-h` or `--help`		print this help and exit.
`-v` or `--version`		print product version and exit.

3 Shell prompt

3.1 Executing a command

There are several commands, that may be executed. After starting *se_mon* in interactive mode (not using switch `-b` or `--batch`), you may type a command by typing its name possibly followed by command arguments. To execute this argument, press <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 <CURSOR LEFT> and <CURSOR RIGHT> keys to navigate within the line and common known 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 commands, that uses the SciEngines Host API. If executing an API command, the command is executed based on the currently selected machine, slot and FPGA. Also the currently set timeout is used for this command. To change the currently selected machine, slot and FPGA indices, use command `goto` (see section 4.25). The currently selected machine slot and FPGA indices are shown in the command prompt. To get/-set the timeout, that is used to execute an API command, use command `timeout` (see section 4.31).
This prompt for example tells the user, that currently slected machine, controller, slot and FPGA are set to 0, 0, 1 and 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` . There are prefixes, that have no distinct command to complete to. Such a command prefix will be completed to another prefix that is prefix of all possible commands where the original prefix was prefix of. When pressing the <tabulator> key a second time, a list representing all possible completions is printed. For example the prefix `fil` will be completed to `file`, which is prefix to the commands `fileRead` and `fileWrite`. When pressing <tabulator> key a second time these two commands are printed out: `fileRead fileWrite` .

3.3.2 Completing a filesystem path

Arguments to a command will always be completed to file paths. If there is not one distinct file to complete to, the path is completed until the

first difference for all possible completions. If there is for example only one file `testfile` in current working directory the prefix `fileRead t` will be completed to `fileRead testfile`. If there are two files `testfile_one` and `testfile_two`, the prefix `fileRead t` will be completed to `fileRead testfile_`. When pressing <tabulator> key a second time, the list of all possible completions for current argument will be printed. In our exmample this is `testfile_one testfile_two`.

3.4 History

Each command that is interactively executed within *se_mon* is saved 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 <CURSOR UP> key to get to older history entries. Use <CURSOR DOWN> key to get to later entries. Also <page up> and <page down> keys may be used to get to the first and latest history entry. When quitting *se_mon* the current history is saved into file `.se_mon_history` located in the current user's home directory.

4 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.

4.1 allocMachine

usage: `allocMachine [MACHINE_INDEX]`
aliases: `alloc, am`
description: Allocates currently selected machine or machine with optional given machine index `MACHINE_INDEX` The currently selected machine may be changed using the goto command (see section 4.25).

Example 1: Here a successful invocation for currently selected machine with index 0:

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

Example 2: Here an unsuccessful invocation for currently selected machine with index 0:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > allocMachine
Machine 0 is locked for usage by user "johndoe" with command "se_mon" (PID: 16032).
ERROR! SciEngines API returned "SeApiMachineInUse" (4)!
0m0.099s (allocMachine)
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

4.2 deprogram

usage: `deprogram`
aliases: `dp`
description: Deprograms FPGA(s) at currently selected address. See section 4.25 for changing the currently selected address.

Example: Deprogram all FPGAs on all cards.

```
Machine=0 Contr=0 Slot=* FPGA=* > deprogram
0m0.018s (deprogram)
Machine=0 Contr=0 Slot=* FPGA=* >
```

4.3 flush

usage: `flush`
aliases: `f`
description: Flushes currently buffered data to currently selected controller and in selected machine.The currently selected machine may be changed using the goto command (see section 4.25). See section 4.25 for changing

the currently selected address.

Example:

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

4.4 freeMachine

usage: `freeMachine [MACHINE_INDEX]`
aliases: `free, fm`
description: Deallocates currently selected machine or machine with optional given machine index `MACHINE_INDEX`. All unread words are discarded.The currently selected machine may be changed using the goto command (see section 4.25).

Example:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > freeMachine
0m0.001s (freeMachine)
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

4.5 getControllerCount

usage: `getControllerCount [MACHINE_INDEX]`
aliases: `ControllerCount, cc`
description: Returns number of controllers for currently selected machine or machine with optional given machine index `MACHINE_INDEX`. The currently selected machine may be changed using the goto command (see section 4.25).

Example:

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

4.6 getControllerInfo

usage: `getControllerInfo [CONTROLLER_INDEX]`
aliases: `ControllerInfo, ci`
description: Returns information about specified controller `CONTROLLER_INDEX` at currently selected machine. The currently selected machine may be changed using the goto command (see section 4.25).

Example: Get first controller's information.

```
Machine=0 Contr=0 Slot=0 FPGA=0 > getControllerInfo
-- Driver name           : pcie
-- Machine Slot          : 0
-- Serial                : 0x27B
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

4.7 getFPGACount

usage: `getFPGACount`
aliases: `FPGACount, fc`
description: Returns number of FPGAs for currently selected machine and slot. See section 4.25 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 >
```

4.8 getFPGAInfo

usage: `getFPGAInfo`
aliases: `FPGAInfo, fi`
description: Returns information about FPGA at currently selected address. See section 4.25 for changing the currently selected address.

Example:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > getFPGAInfo
-- Type                  : XC6SLX150-3fg676
-- Programmed            : true
-- Firmware version      : 01.91.08
-- Firmware build        : 1035
0m0.003s (getFPGAInfo)
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

4.9 getMachineCount

usage: `getMachineCount`
aliases: `MachineCount, mc`
description: Returns number of connected machines.

Example:

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

4.10 getSlotCount

usage: `getSlotCount [MACHINE_INDEX]`
aliases: `SlotCount, sc`
description: Returns number of slots in currently selected machine or machine with optional given machine index `MACHINE_INDEX`. The currently selected machine may be changed using the goto command (see section 4.25).

Example:

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

4.11 getSlotInfo

usage: `getSlotInfo`
aliases: `SlotInfo, si`
description: Returns information about card at currently selected slot and machine address. See section 4.25 for changing the currently selected address.

Example:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > getSlotInfo
-- Serial                : 0x27b
-- FPGA Count            : 8
-- is Controller         : true
-- Controller index      : 0
-- Prev Controller index : 0
-- Next Controller index : 0
-- Firmware version      : 01.91.08
-- Firmware build        : 1035
-- Hardware revision     : 3
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

4.12 getTemperature

usage: `getTemperature`
aliases: `temperature, temp`
description: Returns a card's current and max temperature for currently selected slot and machine address. See section 4.25 for changing the currently selected address.

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

```
Machine=0 Contr=0 Slot=0 FPGA=0 > getTemperature
current: 24.0
max:     38.0
0m0.002s (getTemperature)
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

4.13 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 4.25 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 are 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.
ERROR: FPGAs at slot 0 are programmed but do not react (after 2085428 bytes)!
ERROR! SciEngines API returned "SeApiFailed" (1)!
0m3.789s (program)
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

Example 2: Use goto (see section 4.25) 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
0m0.382s (program)
Machine=0 Contr=0 Slot=* FPGA=* >
```

4.14 readActive

usage: `readActive REG_ADDR NUM_WORDS [FILE]`
aliases: `ra`
description: Reads `NUM_WORDS` 64bit word(s) from given source register address REG_ADDR, from currently selected address using the active mode. See section 4.25 for changing the currently selected address. If no data is present, readActive waits at most as long as timeout was set, which is 1000ms by default (see section 4.31). If `FILE` is provided, the read words are saved to file `FILE`. If `FILE` already exists, the user is interactively asked to append, overwrite the file or to cancel. If `FILE` is not provided, the read words are printed out in decimal, hexadecimal and binary form.

Example: Reading actively 10 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]   9131066143937613 55  0CAC01A0C0FFEE4B  00001100 10101100 00000001 10100000 11000000 11111111
      11101110 01001011
[    9]  1826213228787522710  1958034181FFDC96  00011001 01011000 00000011 01000001 10000001 11111111
       11011100 10010110
0m0.002s (readActive)
Machine=0 Contr=0 Slot=0 FPGA=6 >
```

4.15 readPassive

usage: `readActive REG_ADDR NUM_WORDS [FILE]`
aliases: `read, r, rp`
description: Reads `NUM_WORDS` 64bit word(s) from given source register address REG_ADDR, from currently selected address using the passive mode. See section 4.25 for changing the currently selected address. If no data is present, readPassive waits at most as long as timeout was set, which is 1000ms by default (see section 4.31). If `FILE` is provided, the read words are saved to file `FILE`. If `FILE` already exists, the user is interactively asked to append, overwrite the file or to cancel. If `FILE` is not provided, the read words are printed out in decimal, hexadecimal and binary form.

Example: Reading passively 10 words from currently set address, register 2

```
Machine=0 Contr=0 Slot=0 FPGA=6 > readPassive 2 10
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  000000000000DEDA  00000000 00000000 00000000 00000000 00000000 00000000
       11011110 11011010
ERROR! SciEngines API returned "SeApiReadTimeout" (7)! 5 of 10 words transferred.
0m3.000s (readPassive)
Machine=0 Contr=0 Slot=0 FPGA=6 >
```

4.16 readRequest

usage: `readRequest REG_ADDR NUM_WORDS`
aliases: `rr`
description: Requests to read `NUM_WORDS` 64bit word(s) from given register address REG_ADDR from currently selected address. See section 4.25 for changing the currently selected address. The FPGA is then advised to send `NUM_WORDS` 64bit word(s) back to the host who may read the data via readPassive (see section 4.15). A "readRequest" in conjunction with "readPassive" is equal to "readActive".

Example: Send two read requests each with 3 words to currently set address, register 0 and afterwards read passivly the replies.

```
Machine=0 Contr=0 Slot=0 FPGA=6 > readRequest 0 5
0m0.001s (readRequest)
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]   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
Machine=0 Contr=0 Slot=0 FPGA=6 >
```

4.17 waitForData

usage: `waitForData [-g][-r [FILE_NAME]]`
aliases: `wfd`
description: Waits for incoming data at currently selected controller and machine and returns the address where the data occurred and the number of 64bit words. The currently selected machine may be changed using the goto command (see section 4.25). See section 4.25 for changing the currently selected address. Keep in mind there may be more words than waitForData says to be, because there may be more data arrived just after waitForData returned. 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 returned address and written to file `FILE_NAME` if provided or printed out, else.If no data is present, waitForData waits at most as long as timeout was set, which is 1000ms by default (see section 4.31).

Example 1: In this example there are 5 64bit 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 advice 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 >
```

4.18 write

usage: `write DATA_REGISTER [VALUE [VALUE ...]] or write REG_ADDRESS {-file|-f} FILE`
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 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 4.31). See section 4.25 for changing the currently selected address.

Example 1:

```
Machine=0 Contr=0 Slot=0 FPGA=6 > write 2 0x4488
0m0.001s (write)
Machine=0 Contr=0 Slot=0 FPGA=6 >
```

Example 2:

```
Machine=0 Contr=0 Slot=0 FPGA=6 > write 2 -f dumpfile
0m0.001s (write)
Machine=0 Contr=0 Slot=0 FPGA=6 >
```

4.19 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. Else only the alias for `COMMAND` is printed. Aliases are predefined and may not be altered, created or deleted.

Example 1: Here 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: Here 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
    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 >
```

4.20 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 commandline 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
0m0.014s (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
0m0.418s (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
Machine=0 Contr=0 Slot=0 FPGA=2 > # wait for incoming data at controller with # index 0 at currently
      selected machine waitForData 0
Machine=0 Contr=0 Slot=0 FPGA=2 > # read 1 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
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
0m0.018s (deprogram)
Machine=0 Contr=0 Slot=0 FPGA=* > # free currently selected  machine
Machine=0 Contr=0 Slot=0 FPGA=* > freeMachine
0m0.001s (freeMachine)
Machine=0 Contr=0 Slot=0 FPGA=* >
0m0.455s (batch)
Machine=0 Contr=0 Slot=0 FPGA=* >
```

4.21 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 times is infinite. If `NUM_CALLS` is 1, this call is equal to command batch (see section 4.20).

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
0m0.014s (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
0m0.420s (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
0|Machine=0 Contr=0 Slot=0 FPGA=2 > # wait for incoming data at controller with # index 0 at currently
      selected machine waitForData 0
```

```
0|Machine=0 Contr=0 Slot=0 FPGA=2 > # read 1 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
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
0m0.017s (deprogram)
0|Machine=0 Contr=0 Slot=0 FPGA=* > # free currently selected  machine
0|Machine=0 Contr=0 Slot=0 FPGA=* > freeMachine
0m0.001s (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
0m0.014s (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
0m0.420s (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
1|Machine=0 Contr=0 Slot=0 FPGA=2 > # wait for incoming data at controller with # index 0 at currently
      selected machine waitForData 0
1|Machine=0 Contr=0 Slot=0 FPGA=2 > # read 1 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
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
0m0.018s (deprogram)
1|Machine=0 Contr=0 Slot=0 FPGA=* > # free currently selected  machine
1|Machine=0 Contr=0 Slot=0 FPGA=* > freeMachine
0m0.001s (freeMachine)
1|Machine=0 Contr=0 Slot=0 FPGA=* >
0m0.912s (batchLoop)
Machine=0 Contr=0 Slot=0 FPGA=* >
```

4.22 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 if the execution should be continued or not. If the execution shall not be continued then 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 not stopped. Also if the currently executed batch file was executed by batchLoop command (see section 4.21), 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=* >
```

4.23 fileRead

usage: `fileRead FILE`
aliases: `fr`
description: Reads 64bit word(s) from file `FILE` and prints them out in

decimal, hexadecimal and binary form.

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 >
```

4.24 fileWrite

usage: `fileWrite FILE VALUE [VALUE [VALUE [VALUE ...]]]`
aliases: `fw`
description: Writes 64bit word(s) to file `FILE`. If `VALUE` starts with 0x, it is interpreted as hexadecimal, else it is interpreted as decimal 64bit word. If `FILE` already exists, the user is interactively asked to append, overwrite the file or to cancel.

Example 1:

```
Machine=0 Contr=0 Slot=0 FPGA=0 > fileWrite dumpfile 0xbeaf 42
0m0.009s (fileWrite)
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
0m0.010s (fileWrite)
Machine=0 Contr=0 Slot=0 FPGA=0 >
```

4.25 goto

usage: `goto MACHINE_INDEX CONTR_INDEX SLOT_ADDRESS FPGA_ADDRESS or goto CONTR_INDEX SLOT_ADDRESS FPGA_ADDRESS or goto SLOT_ADDRESS FPGA_ADDRESS or 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 address are set to `MACHINE_INDEX`, `CONTR_INDEX`, `SLOT_ADDRESS` and `FPGA_ADDRESS`. If three arguments are provided, the currently selected machine index, as well as the slot and FPGA addresses are set to `MACHINE_INDEX`, `SLOT_ADDRESS` and `FPGA_ADDRESS`. 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 index is not changed. If one argument is provided, the currently selected FPGA address is set to `FPGA_ADDRESS`, while currently selected machine index and slot address remain unchanged. `SLOT_ADDRESS` and/or `FPGA_ADDRESS` may be an asterisk (*) representing a wildcard addressing all 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 index unchanged at index 1.

```
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=* >
```

4.26 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: Here 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 currently selected machine or machine with optional 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: Here 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 number of controllers for a machine.
    getControllerInfo   Returns information about a controller.
    getFPGACount        Returns number of FPGAs in a machine.
    getFPGAInfo         Returns information about an FPGA.
    getMachineCount     Returns number of connected machines.
    getSlotCount        Returns number of slots in a machine.
    getSlotInfo         Returns information about a slot.
    getTemperature      Returns temperature for a card.
    program             Programs FPGA(s) with program file.
    readActive          Reads from a data register using the active mode.
    readPassive         Reads from a data register 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=* >
```

4.27 options

usage: `options [sync|async] [normal|systolic]`
aliases: o
description: Invoking options command without arguments, the currently set options are printed out.To change write behavior to be synchronous or asynchronous, SYNC or ASYNC may be used as argument. To change routing behavior, NORMAL or SYSTOLIC may be used as argument. Please refer to API documentation to get to know more about the different write behavior and routing methods. These options are used only when allocating a machine. An already allocated machine needs to be freed and allocated again to make 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
```

4.28 template

usage: `template [TEMPLATE_FILE|-reset|-r]`
aliases:
description:

4.29 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
```

4.30 sleep

usage: `sleep TIME_IN_MS`
aliases:
description: Sleeps for `TIME_IN_MS` milliseconds and blocks any interaction while sleeping.

Example: Sleep for 3 seconds.

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

4.31 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`. 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.

Example: Get the currently set timeout, set it to 3 seconds and get the currently set timeout again.

```
Machine=0 Contr=0 Slot=0 FPGA=0 > timeout
1000
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 >
```

5 Third Party Licenses

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

```
Copyright (c) 2010-2013, 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
HOLDER 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.
```

Thank you for choosing an original SciEngines product.

Imprint
Responsible for content:

Firm SciEngines GmbH

Street Fraunhoferstr 13

ZIP D-24118

City Kiel

Country Germany

Phone +49 431 5302 482

Email info@sciengines.com

WWW http://www.sciengines.com

CEO Gerd Pfeiffer

Commercial Register Amtsgericht Kiel

Commercial Register No. HR B 9565 KI

VAT- Identification Number DE 814955925