RIVYERA

reconfigure versatilely your efficient raw architecture

ActiveRead

Version 2.0.1.0

SciEngines GmbH
Am Kiel-Kanal 2
24106 Kiel
Germany

www.sciengines.com

Revision: 1.92.02 December 10, 2015

Table of Contents

1 Introduction

SciEngines RIVYERA is a high performance reconfigurable computing platform. It is capable of massive performance gains compared to standard architectures. However, it does not follow the *von Neumann* architecture and therefore it requires a different style of programming, which is illustrated in this sample. ActiveRead is a very simple communication example to get in touch with the active read mechanism which is a special mode for the `se_read()` function. The active read is used for requesting a number of words from an FPGA, waiting for the reply and reading it. In this example, the FPGA may be asked to write an arbitrary number of words to the requester. These words initially start with 0 and are incremented word by word. It is also possible to change the starting number by writing it to an FPGA. It might be a good idea to have a look at the PingPong example first, before trying to understand this example.

2 RIVYERA Implementation

In contrast to the *PingPong* example, which uses the *passive read* mechanism, this example uses the *active read* mechanism.
The active read mechanism is defined by the SciEngines-API and allows the host or an FPGA to actively request and afterwards read an arbitrary number of 64 bit data words from a single FPGA. The active read is therefore internally devided into two parts:

1. Write a read request to an FPGA.
2. Wait for the answer and read it passively.

The read request is a special data packet that is sent to the target FPGA which is addressed by the active read. In contrast to regular data packet, the target command (`api_i_tgt_cmd`) is set to `CMD_RD` rather than `CMD_WR`. The data itself (`api_i_data`) is then an integer number which is the number of requested data packets. It is necessary for the FPGA design to explicitly handle incoming data packets which have the target command set to `CMD_RD`. Else the active read mechanism may not be used. Nevertheless, the active read is an optional feature and handling it in the FPGA design might be left away if there is no need for it.
`se_read()` implicitly does both parts -sending the read request and reading the requested data- when called with mode `SeReadActive`. But it may also be called with mode `SeReadReqest` to only write a read request to an FPGA. Afterwards `se_read()` may be called with mode `SeReadPassive` to wait for the requested data and read it passively. A typical active read communication flow is illustrated in figure 1.

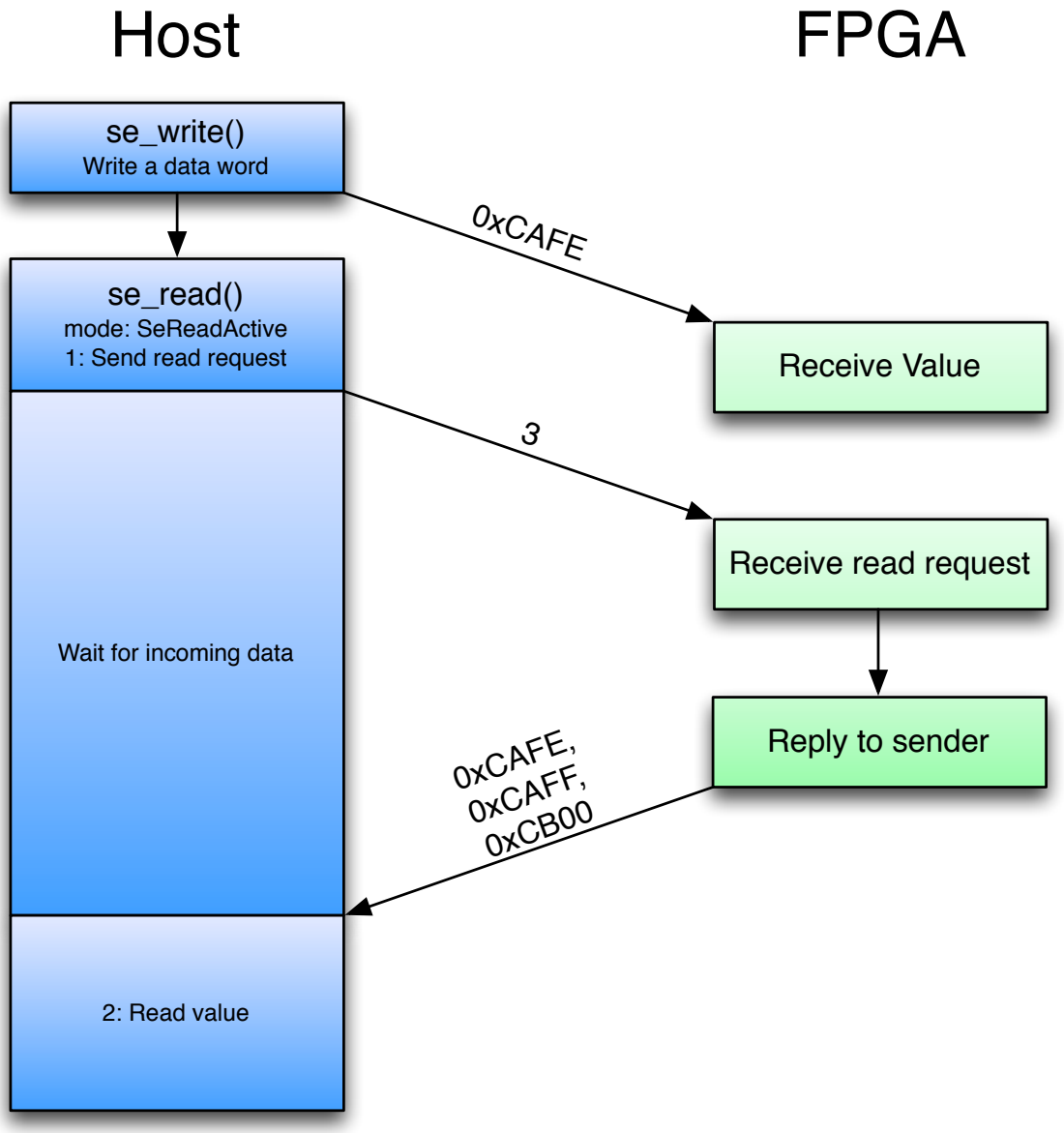

Figure 1: Communication flow: `ar_current_word` is set to `0xCAFE` and three data packets are read actively.

The ActiveRead example is intended to illustrate and help understanding the active read mechanism.

2.1 FPGA Design

The FPGA design's key features are (see activeread_main.vhd in chapter 3.1):

1. Incoming read requests are handled for any possible target register address $(0 \ldots 62)$. The FPGA design replies to a read request by writing values to the request's source address. These written values are taken from the internal register `ar_current_word`. Each time a value is sent, it is incremented and stored again in `ar_current_word`. There are as many data packets sent as there are requested.

2. `ar_current_word` is not reset between different read requests.

3. ar_current_word may be set by sending a new value to target register address 0. Initially, ar_current_word is set to value 0.

Initially ar_counter is set to 0. While there is no incoming data packet, api_i_empty_in remains '1' and the if statement in line 129 is false. That means that there is no change in ar_counter until there is an incoming read request data packet.
Lets assume there is an incoming read request (which is the active read's first part) targetting register address 0. Further lets assume the active read is about to read one data packet. The read request data packet requests one reply data packet by setting api_i_data_in to 1.

In the first clock cycle this will happen:

The if statement in line 129 is true (api_i_empty_in equals '0' and indicates an incoming data packet). Also, api_i_tgt_cmd_in is equal to CMD_RD indicating a read request. Thus the elsif statement in line 147 is true. Then the first case in line 154 is applicable since the request's target register address is 0. Due to the request that requests one data packet, the if statement in line 155 is false (api_i_data_in equals 1 but ar_counter is still 0). So the lines 159-162 are processed (ar_current_word is incremented and sent; ar_counter is incremented). Note that the read request data packet is not yet acknowledged by setting api_i_rd_en_out to '1'.

In the second clock cycle:

The read request is not yet acknowledged. So the first case in line 154 is still applicable. But ar_counter now equals api_i_data_in (both are 1) which means the incoming read request is now acknowledged by setting api_i_rd_en to '1'. ar_counter is reset to its default value 0.

Third clock cycle:

The read request is now being acknowleded. The if statement in line 129 is false due to the api_i_rd_en signal being '1'. So the default assignment in line 111 resets the api_i_rd_en to '0' again. In the next clock cycle the process will be ready to handle another data packet.

2.2 Host Design

The host design's key features are (see activeread.c in chapter 4.1 and ActiveRead.java in chapter 5.1):

1. Three words are read actively from the first FPGA on the first card by using the SciEngines-host-API function se_read() with mode SeReadActive.

2. Ten words are read actively from the first FPGA on the first card by using se_read() with mode ReadRequest and again se_read() but with mode SeReadPassive. These two function calls are identical to the se_read() with mode SeReadActive.

3. ar_current_word is set to value 0xcafe by writing it to the first FPGA on the first card using se_write().

4. Finally five words are read actively from the first FPGA on the first card.

```
# actively reading 3 words from FPGA 0 in slot 0, register 0
src [s0 fH r0 cWR]    >>> tgt [s0 f0 r0 cRD]: 0x0000000000000003
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x0000000000000000
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x0000000000000001
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x0000000000000002
# requesting 10 words from FPGA 0 in slot 0, register 0
src [s0 fH r0 cWR]    >>> tgt [s0 f0 r0 cRD]: 0x000000000000000a
# passively reading 10 words from FPGA 0 in slot 0, register 0
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x0000000000000003
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x0000000000000004
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x0000000000000005
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x0000000000000006
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x0000000000000007
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x0000000000000008
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x0000000000000009
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x000000000000000a
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x000000000000000b
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x000000000000000c
# writing the word 0xcafe to FPGA 0 in slot 0, register 0
# (for the ActiveRead example, this sets the start word for replies)
src [s0 fH r0 cWR]    >>> tgt [s0 f0 r0 cWR]: 0x000000000000cafe
# actively reading 5 words from FPGA 0 in slot 0, register 0
src [s0 fH r0 cWR]    >>> tgt [s0 f0 r0 cRD]: 0x0000000000000005
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x000000000000cafe
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x000000000000caff
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x000000000000cb00
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x000000000000cb01
tgt [s0 fH r0 cWR] <<<    src [s0 f0 r0 cRD]: 0x000000000000cb02
```

Figure 2: This is a communication log for the example execution. Within the squared brackets the *s* stands for *slot*, *f* for *fpga*, *r* for *register*, *c* for *command*, *WR* for *CMD_WR*, *RD* for *CMD_RD* and *h* for *host*.

The communication that occurs when executing this example is represented in detail in figure 2.
Note that in this very simple example the FPGA and Slot addresses are hardcoded. The SciEngines-host-API functions `se_getSlotCount()` and `se_getFPGACount()` might be useful for setting addresses dynamically.

3 FPGA VHDL Sources

3.1 activeread_main.vhd

```
1  ---
2  -- Project: ActiveRead
3  -- File:    activeread_main.vhd
4  -- Date:    Thu Nov 22 16:03:58 CET 2012
5  -- Author:  Daniel Siebert (SciEngines GmbH)
6  --
7  -- Description:
8  -- This is the ActiveRead example project.
9  --
10 -- Copyright (c) 2012-2015, SciEngines GmbH
11 -- All rights reserved.
12 --
13 -- Redistribution in source or binary forms, with or without modification,
14 -- is not permitted without specific prior written permission.
15 --
16 -- THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
17 -- "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
18 -- LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
19 -- A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
20 -- OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
21 -- SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
22 -- LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
23 -- DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
24 -- THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
25 -- (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
26 -- OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
27 ---
28 library ieee;
29 use work.sciengines_api_types.all;
30 use ieee.std_logic_1164.all;
31 use ieee.std_logic_unsigned.all;
32 use ieee.std_logic_arith.all;
33
34 entity activeread_main is
35    generic (
36       NUM_LEDS               : integer
37    );
38    port (
39       ------------------------------
40       --------- API PORTS ----------
41       ------------------------------
42       -- USER PORTS
43       api_clk_in             : in    std_logic;
44       api_rst_in             : in    std_logic;
45       api_led_out            : out   seBusFlag_type(NUM_LEDS-1 downto 0) := (others => '0');
46       api_hw_rev_in          : in    seHwRev_type;
47       -- ADDRESS PORTS
48       api_self_contr_in      : in    seFlag_type;
49       api_next_contr_in      : in    seSlotAddr_type;
50       api_prev_contr_in      : in    seSlotAddr_type;
51       api_self_slot_in       : in    seSlotAddr_type;
52       api_self_fpga_in       : in    seFpgaAddr_type;
53       -- OUTPUT REGISTER PORTS
54       api_o_clk_out          : out   std_logic;
55       api_o_rfd_in           : in    seFlag_type;
56       api_o_tgt_slot_out     : out   seSlotAddr_type := (others => '0');
57       api_o_tgt_fpga_out     : out   seFpgaAddr_type := (others => '0');
58       api_o_tgt_reg_out      : out   seRegAddr_type  := (others => '0');
59       api_o_tgt_cmd_out      : out   seCmd_type      := CMD_WR;
60       api_o_src_reg_out      : out   seRegAddr_type  := (others => '0');
61       api_o_src_cmd_out      : out   seCmd_type      := CMD_WR;
62       api_o_data_out         : out   seData_type     := (others => '0');
63       api_o_wr_en_out        : out   seFlag_type     := '0';
64       -- INPUT REGISTER PORTS
65       api_i_clk_out          : out   std_logic;
66       api_i_src_slot_in      : in    seSlotAddr_type;
67       api_i_src_fpga_in      : in    seFpgaAddr_type;
68       api_i_src_reg_in       : in    seRegAddr_type;
69       api_i_src_cmd_in       : in    seCmd_type;
70       api_i_tgt_reg_in       : in    seRegAddr_type;
71       api_i_tgt_cmd_in       : in    seCmd_type;
72       api_i_data_in          : in    seData_type;
73       api_i_empty_in         : in    seFlag_type;
74       api_i_am_empty_in      : in    seFlag_type;
75       api_i_rd_en_out        : out   seFlag_type     := '0'
76    );
77 end entity activeread_main;
78
79 architecture activeread_main_behave of activeread_main is
80
```

```vhdl
   -- Define some local signals because
   -- we can not read from the out-ports.
   signal api_i_rd_en      : seFlag_type       := '0';
   signal api_o_wr_en      : seFlag_type       := '0';
   signal ar_current_word  : seData_type       := (others => '0');
   signal ar_counter       : seData_type       := (others => '0');

begin

   -- When defining an own clock domain to
   -- run the user design with a different
   -- clock, the following two lines should
   -- probably be altered
   api_i_clk_out <= api_clk_in;
   api_o_clk_out <= api_clk_in;

   -- route the internal signals to the out-ports
   api_i_rd_en_out <= api_i_rd_en;
   api_o_wr_en_out <= api_o_wr_en;

   -- A Spartan 6 FPGA has two LEDs for debugging purposes.
   -- Set these LEDs disabled here. Comment following
   -- line and use this signal anywhere you want to!
   api_led_out <= (others => '0');

   activeread_process : process
   begin
      wait until rising_edge(api_clk_in);

      -- Do not read anything by default.
      api_i_rd_en <= '0';

      -- Do not write anything by default.
      api_o_wr_en <= '0';

      -- Set the answer's target slot-,
      -- fpga- and register-addresses.
      api_o_tgt_slot_out <= api_i_src_slot_in;
      api_o_tgt_fpga_out <= api_i_src_fpga_in;
      api_o_tgt_reg_out  <= api_i_src_reg_in;
      api_o_tgt_cmd_out  <= api_i_src_cmd_in;
      api_o_src_reg_out  <= api_i_tgt_reg_in;
      api_o_src_cmd_out  <= api_i_tgt_cmd_in;

      if (api_rst_in = '1') then
         -- While user_rst_in is high, the api
         -- is not ready for use.
      else
         if (api_i_empty_in = '0'
               and api_i_rd_en = '0') then
            -- There is a new incoming data packet
            -- and this data packet has not been read last clock cycle.

            if (api_i_tgt_cmd_in = CMD_WR) then
               -- this is an incoming regular data packet
               -- which has been written using se_write
               case api_i_tgt_reg_in is
                  when "000000" =>
                     -- this means, we set the incoming payload as start value
                     ar_current_word      <= api_i_data_in;
                     api_i_rd_en          <= '1';
                  when others =>
                     -- only handle register 0,
                     -- discard incoming data packet
                     api_i_rd_en          <= '1';
               end case;
            elsif (api_i_tgt_cmd_in = CMD_RD
               and api_o_rfd_in = '1') then
               -- this is an incoming read request data packet
               -- that has been written using se_read with mode
               -- SeReadActive or mode SeReadRequest.
               -- also the API is ready for data (rfd) to be written.
               case api_i_tgt_reg_in is
                  when "000000" =>
                     if (ar_counter = api_i_data_in) then
                        ar_counter        <= (others => '0');
                        api_i_rd_en       <= '1';
                     else
                        ar_current_word   <= ar_current_word + 1;
                        api_o_data_out    <= ar_current_word;
                        api_o_wr_en       <= '1';
                        ar_counter        <= ar_counter + 1;
                     end if;
                  when others =>
                     -- only handle register 0,
                     -- discard any other incoming data packet
```

```
                    api_i_rd_en          <= '1';
                end case;
             else
                -- This should never happen, but if it does,
                -- we just discard the incoming data packet.
                api_i_rd_en <= '1';
             end if;
          end if;
       end if;
   end process activeread_process;


end architecture activeread_main_behave;
```

4 Host C Sources

4.1 activeread.c

```
/*
 * Project: ActiveRead
 * File:    activeread.c
 * Date:    Thu Nov 22 16:25:08 CET 2012
 * Author:  Daniel Siebert (SciEngines GmbH)
 *
 * Description:
 * This is the ActiveRead example project.
 *
 * Copyright (c) 2012-2015, SciEngines GmbH
 * All rights reserved.
 *
 * Redistribution in source or binary forms, with or without modification,
 * is not permitted without specific prior written permission.
 *
 * 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.
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <getopt.h>
#include <SeHostAPI.h>

#define  API_ERROR_CHECK(x)   {SE_STATUS rc = x; if(rc != SeApiSuccess) { printf("ERROR! SciEngines API returned %d (= %s)! \n", rc, se_status2str(rc)); exit(EXIT_FAILURE);} else { printf("SUCCESS !\n"); }}

extern const char *__progname;
static const char *PROGRAM_NAME = "activeread";
static const char *PROGRAM_VERSION = "1.92.02";
static const char *COPYRIGHT_TXT = "Copyright (c) 2012-2015, SciEngines GmbH";

const char* BIT_FILE = "../../fpga/vhdl/xc6slx150-3fgg676/activeread_top.bit";
const char* SIM_FILE = "../../fpga/vhdl/xc6slx150-3fgg676/activeread.sim";

static void printUsage() {
   printf("Usage: %s [-options] [BIT_FILE]\n", __progname);
   printf("   \n");
   printf("where options include:\n");
   printf("   -h --help          print this help and exit\n");
   printf("   -v --version       print product version and exit\n");
   printf("   -m --machine       run your program on a specific machine\n");
}

static void printVersion() {
   printf("%s version %s\n", PROGRAM_NAME, PROGRAM_VERSION);
   printf("%s\n", COPYRIGHT_TXT);
}

static int run_program(se_machine_t machine, const char* bit_file) {
   SE_STATUS         retval = EXIT_SUCCESS;
   SE_ADDR           addr;         /* address structure  */
```

```
62 |    __uint64_t        pReceive[10]; /* received data words */
63 |    __uint64_t        send;         /* send data word     */
64 |    SE_CONTROLLERINFO controllerInfo;
65 |    size_t            tc;
66 |    size_t            i;
67 |
68 |    printf("Running␣ActiveRead␣with␣SciEngines␣RIVYERA␣Host-API␣v␣%02u.%02u.%02u␣(%s)...\n\n",
69 |          SE_API_VERSION_MAJOR, SE_API_VERSION_MINOR, SE_API_VERSION_SP,
70 |          SE_API_VERSION_REVISION);
71 |
72 |    if (machine >= se_getMachineCount()) {
73 |       printf("Error:␣No␣such␣index:␣%u\n", machine);
74 |       return EXIT_FAILURE;
75 |    }
76 |
77 |    printf("Allocating␣machine␣%u:␣", machine);
78 |    API_ERROR_CHECK(se_allocMachine(machine, NULL))
79 |
80 |    /* set address to all slots, all FPGAs. */
81 |    addr.contr = 0;
82 |    addr.slot  = SE_ADDR_SLOT_ALL;
83 |    addr.fpga  = SE_ADDR_FPGA_ALL;
84 |    addr.reg   = 0;
85 |
86 |    printf("Getting␣controller␣information␣for␣machine␣%u:␣", machine);
87 |    API_ERROR_CHECK(se_getControllerInfo(machine, addr.contr, &controllerInfo));
88 |    printf("Programming␣fpgas:␣");
89 |    if (strcmp(controllerInfo.driver_name, "isim") == 0) {
90 |       API_ERROR_CHECK(se_program(machine, &addr, SIM_FILE, 5000))
91 |    } else if (bit_file) {
92 |       API_ERROR_CHECK(se_program(machine, &addr, bit_file, 5000))
93 |    } else {
94 |       API_ERROR_CHECK(se_program(machine, &addr, BIT_FILE, 5000))
95 |    }
96 |
97 |    addr.slot = 0;
98 |    addr.fpga = 0;
99 |    printf("\nActively␣reading␣3␣words␣from␣[machine␣%u,␣contr␣%u,␣slot␣%u,␣fpga␣%u,␣reg␣%u]:␣",
   |         machine, addr.contr, addr.slot, addr.fpga, addr.reg);
100|    API_ERROR_CHECK(se_read(machine, &addr, pReceive, 3, SeReadActive,  &tc, 5000))
101|    for (i = 0; i < tc; i++) {
102|       printf("Received␣value␣at␣index␣%-2zu:␣0x%016lx.\n", i, pReceive[i]);
103|    }
104|
105|    printf("\nRequesting␣10␣words␣from␣[machine␣%u,␣contr␣%u,␣slot␣%u,␣fpga␣%u,␣reg␣%u]:␣", machine,
   |         addr.contr, addr.slot, addr.fpga, addr.reg);
106|    API_ERROR_CHECK(se_read(machine, &addr, NULL, 10, SeReadRequest,  &tc, 5000))
107|    printf("Passively␣reading␣3␣words␣from␣[machine␣%u,␣contr␣%u,␣slot␣%u,␣fpga␣%u,␣reg␣%u]:␣",
   |         machine, addr.contr, addr.slot, addr.fpga, addr.reg);
108|    API_ERROR_CHECK(se_read(machine, &addr, pReceive, 10, SeReadPassive,  &tc, 5000))
109|    for (i = 0; i < tc; i++) {
110|       printf("Received␣value␣at␣index␣%-2zu:␣0x%016lx.\n", i, pReceive[i]);
111|    }
112|
113|    send = 0xcafe;
114|    printf("\nWriting␣value␣0x%016lx␣to␣[machine␣%u,␣contr␣%u,␣slot␣%u,␣fpga␣%u,␣reg␣%u]:␣", send,
   |         machine, addr.contr, addr.slot, addr.fpga, addr.reg);
115|    API_ERROR_CHECK(se_write(machine, &addr, &send, 1, NULL, 5000))
116|
117|    printf("Actively␣reading␣5␣words␣from␣[machine␣%u,␣contr␣%u,␣slot␣%u,␣fpga␣%u,␣reg␣%u]:␣", machine
   |         , addr.contr, addr.slot, addr.fpga, addr.reg);
118|    API_ERROR_CHECK(se_read(machine, &addr, pReceive, 5, SeReadActive,  &tc, 5000))
119|    for (i = 0; i < tc; i++) {
120|       printf("Received␣value␣at␣index␣%-2zu:␣0x%016lx.\n", i, pReceive[i]);
121|    }
122|
123|    printf("Deprogramming␣fpgas:␣");
124|    addr.slot  = SE_ADDR_SLOT_ALL;
125|    addr.fpga  = SE_ADDR_FPGA_ALL;
126|    API_ERROR_CHECK(se_deprogram(machine, &addr))
127|
128|    printf("Freeing␣machine␣%u:␣", machine);
129|    API_ERROR_CHECK(se_freeMachine(machine))
130|
131|    return retval;
132|}
133|
134|int main(int argc, char* const argv[] ) {
135|    /* getopt_long stores the option index here. */
136|    int            option_index = 0;
137|    char           c;
138|    const char     *bit_file = NULL;
139|    se_machine_t   machine = 0;
140|
141|    static struct option long_options[] = {
142|       {"help",         no_argument,        NULL, 'h'},
143|       {"version",      no_argument,        NULL, 'v'},
```

```
144 |       {"machine",      required_argument,  NULL, 'm'},
145 |       {"timestamp",    no_argument,        NULL, 1},
146 |       {0, 0, 0, 0}
147 |    };
148 |
149 |    while ((c = getopt_long(argc, argv, "hvm:", long_options, &option_index)) != -1) {
150 |       switch (c) {
151 |       case 'h':
152 |          printUsage();
153 |          exit(EXIT_SUCCESS);
154 |          break;
155 |       case 'v':
156 |          printVersion();
157 |          exit(EXIT_SUCCESS);
158 |          break;
159 |       case 'm':
160 |          if (sscanf(optarg, "%u", &machine) != 1){
161 |             fprintf(stderr, "Could␣not␣read␣argument␣%s\n", optarg);
162 |          }
163 |          else{
164 |          }
165 |          break;
166 |       case 1:
167 |          printf("%s␣%s\n", __DATE__, __TIME__);
168 |          exit(EXIT_SUCCESS);
169 |          break;
170 |       default:
171 |          exit(EXIT_FAILURE);
172 |          break;
173 |       }
174 |    }
175 |
176 |    if (optind + 1 < argc) {
177 |       fprintf(stderr, "Unexpected␣argument:␣\"%s\"\n", argv[optind + 1]);
178 |       exit(EXIT_FAILURE);
179 |    } else {
180 |       if (optind < argc) {
181 |          bit_file = argv[optind];
182 |       }
183 |       exit(run_program(machine, bit_file));
184 |    }
185 |
186 |    return 0;
187 | }
```

5 Host Java Sources

5.1 ActiveRead.java

```
1  | import com.sciengines.rivyera.api.types.*;
2  | import com.sciengines.rivyera.api.types.exceptions.*;
3  | import static com.sciengines.rivyera.api.SciEngines_API.*;
4  | import static com.sciengines.rivyera.api.SciEngines_API_Const.*;
5  | import java.nio.ByteBuffer;
6  | import java.nio.ByteOrder;
7  |
8  | /**
9  |  * Project: ActiveRead
10 |  * Date:    Thu Nov 22 16:25:08 CET 2012
11 |  * @author  Daniel Siebert (SciEngines GmbH)
12 |  *
13 |  * Description:
14 |  * This is the ActiveRead example project.
15 |  *
16 |  * Copyright (c) 2012-2015, SciEngines GmbH
17 |  * All rights reserved.
18 |  *
19 |  * Redistribution in source or binary forms, with or without modification,
20 |  * is not permitted without specific prior written permission.
21 |  *
22 |  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
23 |  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
24 |  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
25 |  * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
26 |  * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
27 |  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
28 |  * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
29 |  * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
30 |  * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
31 |  * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
```

```
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
public class ActiveRead {

   private static final String PROGRAM_NAME = "activeread";
   private static final String PROGRAM_VERSION = "1.92.02";
   private static final String COPYRIGHT_TXT = "Copyright (c) 2012-2015, SciEngines GmbH";

   private static final String BIT_FILE = "../../../fpga/vhdl/xc6slx150-3fgg676/activeread_top.bit";
   private static final String SIM_FILE = "../../../fpga/vhdl/xc6slx150-3fgg676/activeread.sim";

   private static void printUsage() {
      System.out.println("Usage: java " + PROGRAM_NAME + " [-options] [BIT_FILE]");
      System.out.println("   ");
      System.out.println("where options include:");
      System.out.println("   -h --help           print this help and exit");
      System.out.println("   -v --version        print product version and exit");
      System.out.println("   -m --machine        run your program on a specific machine");
   }

   static void printVersion() {
      System.out.println(PROGRAM_NAME + " version " + PROGRAM_VERSION);
      System.out.println(COPYRIGHT_TXT);
   }

   public static int run_program(int machine, String bitFile) {
      int              retval = 0;
      SeOptions        options;
      SeAddress        addr;            /* address structure  */
      long             receive;         /* received data word */
      long             send;            /* send data word     */
      ByteBuffer       payload;
      SeControllerInfo controllerInfo;
      long             tc;

      options = new SeOptions(
            SeOptions.SeWriteBehavior.se_write_async,
            SeOptions.SeRoutingMethod.se_routing_normal);
      payload = ByteBuffer.allocateDirect(8 * 10)
            .order(ByteOrder.nativeOrder());
      System.out.printf("Running ActiveRead with SciEngines RIVYERA Host-API v %02d.%02d.%02d (%s)
          ...\n\n",
             SE_API_VERSION_MAJOR, SE_API_VERSION_MINOR, SE_API_VERSION_SP, SE_API_VERSION_REVISION)
                 ;

      if (machine >= se_getMachineCount()) {
         System.out.println("no such index: " + machine);
         System.exit(1);
      }

      try {
         System.out.print("Allocating machine " + machine + ": ");
         se_allocMachine(machine, options);
         System.out.println("SUCCESS!");

         addr = new SeAddress(0, SE_ADDR_SLOT_ALL, SE_ADDR_FPGA_ALL, 0);
         System.out.printf("Getting controller information for machine %d: ", machine);
         controllerInfo = se_getControllerInfo(machine, addr.contr);
         System.out.println("SUCCESS!");

         System.out.print("Programming fpgas: ");
         if ("isim".equals(controllerInfo.getDriverName())) {
            se_program(machine, addr, SIM_FILE, 5000);
         } else if (bitFile != null) {
            se_program(machine, addr, bitFile, 5000);
         } else {
            se_program(machine, addr, BIT_FILE, 5000);
         }
         System.out.println("SUCCESS!");

         addr.slot = 0;
         addr.fpga = 0;
         addr.reg = 0;

         System.out.printf("\nActively reading 3 words from [machine %d, contr %d, slot %d, fpga %d,
              reg %d]: ",
               machine, addr.contr, addr.slot, addr.fpga, addr.reg);
         tc = se_read(machine, addr, payload, 3, SE_READ_ACTIVE,
                 1000);
         System.out.println("Success!");
         for (int i = 0; i < tc; i++) {
            receive = payload.getLong(i * 8);
            System.out.printf("Received value at index %-2d: 0x%016x.\n", i, receive);
         }

         System.out.printf("\nRequesting 10 words from [machine %d, contr %d, slot %d, fpga %d, reg %
              d]: ",
```

```
115 |                machine, addr.contr, addr.slot, addr.fpga, addr.reg);
116 |          tc = se_read(machine, addr, payload, 10, SE_READ_REQUEST,
117 |                  1000);
118 |          System.out.println("Success!");
119 |
120 |          System.out.printf("Passively␣reading␣10␣words␣from␣[machine␣%d,␣contr␣%d,␣slot␣%d,␣fpga␣%d,␣
                reg␣%d]:␣",
121 |                machine, addr.contr, addr.slot, addr.fpga, addr.reg);
122 |          tc = se_read(machine, addr, payload, 10, SE_READ_PASSIVE,
123 |                  1000);
124 |          System.out.println("Success!");
125 |          for (int i = 0; i < tc; i++) {
126 |             receive = payload.getLong(i * 8);
127 |             System.out.printf("Received␣value␣at␣index␣%-2d:␣0x%016x.\n", i, receive);
128 |          }
129 |
130 |          send = 0xcafe;
131 |          System.out.printf("\nWriting␣value␣0x%016x␣to␣[machine␣%d,␣contr␣%d,␣slot␣%d,␣fpga␣%d,␣reg␣%
                d]:␣",
132 |                send, machine, addr.contr, addr.slot, addr.fpga, addr.reg);
133 |          payload.putLong(0, send);
134 |          se_write(machine, addr, payload, 1, 1000);
135 |          System.out.println("Success!");
136 |
137 |          System.out.printf("Actively␣reading␣5␣words␣from␣[machine␣%d,␣contr␣%d,␣slot␣%d,␣fpga␣%d,␣
                reg␣%d]:␣",
138 |                machine, addr.contr, addr.slot, addr.fpga, addr.reg);
139 |          tc = se_read(machine, addr, payload, 5, SE_READ_ACTIVE,
140 |                 1000);
141 |          System.out.println("Success!");
142 |          for (int i = 0; i < tc; i++) {
143 |             receive = payload.getLong(i * 8);
144 |             System.out.printf("Received␣value␣at␣index␣%-2d:␣0x%016x.\n", i, receive);
145 |          }
146 |
147 |          System.out.print("Deprogramming␣fpgas:␣");
148 |          addr.slot = SE_ADDR_SLOT_ALL;
149 |          addr.fpga = SE_ADDR_FPGA_ALL;
150 |          se_deprogram(machine, addr);
151 |          System.out.println("SUCCESS!");
152 |
153 |          System.out.print("Freeing␣machine␣" + machine + ":␣");
154 |          se_freeMachine(machine);
155 |          System.out.println("SUCCESS!");
156 |       } catch(SeApiException e) {
157 |          System.err.println("Failed:␣" + e.getMessage());
158 |       }
159 |
160 |       return retval;
161 |    }
162 |
163 |    public static void main(String[] args) {
164 |       String   bitFile = null;
165 |       int      machine = 0;
166 |
167 |       for (int optind = 0; optind < args.length; optind++) {
168 |          if (args[optind].equals("-h") || args[optind].equals("--help")) {
169 |             printUsage();
170 |             System.exit(0);
171 |          } else if (args[optind].equals("-v") || args[optind].equals("--version")) {
172 |             printVersion();
173 |             System.exit(0);
174 |          } else if (args[optind].equals("-m") || args[optind].equals("--machine")) {
175 |             optind++;
176 |             if (optind < args.length) {
177 |                machine = Integer.parseInt(args[optind]);
178 |             } else {
179 |                  System.err.println("Missing␣argument␣for␣option␣" + args[optind - 1]);
180 |                  System.exit(1);
181 |             }
182 |          } else if (bitFile == null) {
183 |             bitFile = args[optind];
184 |          } else {
185 |             System.err.println("Unexpected␣argument␣\"" + args[optind] + "\"");
186 |             System.exit(1);
187 |          }
188 |       }
189 |       System.exit(run_program(machine, bitFile));
190 |    }
191 |
192 | }
```

Thank you for choosing an original SciEngines product.

Imprint
Responsible for content:

Firm SciEngines GmbH

Street Am Kiel-Kanal 2

ZIP D-24106

City Kiel

Country Germany

Phone +49 431 9086200 0

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