1 NIOS 1

The required data path is depicted in Figure 1.

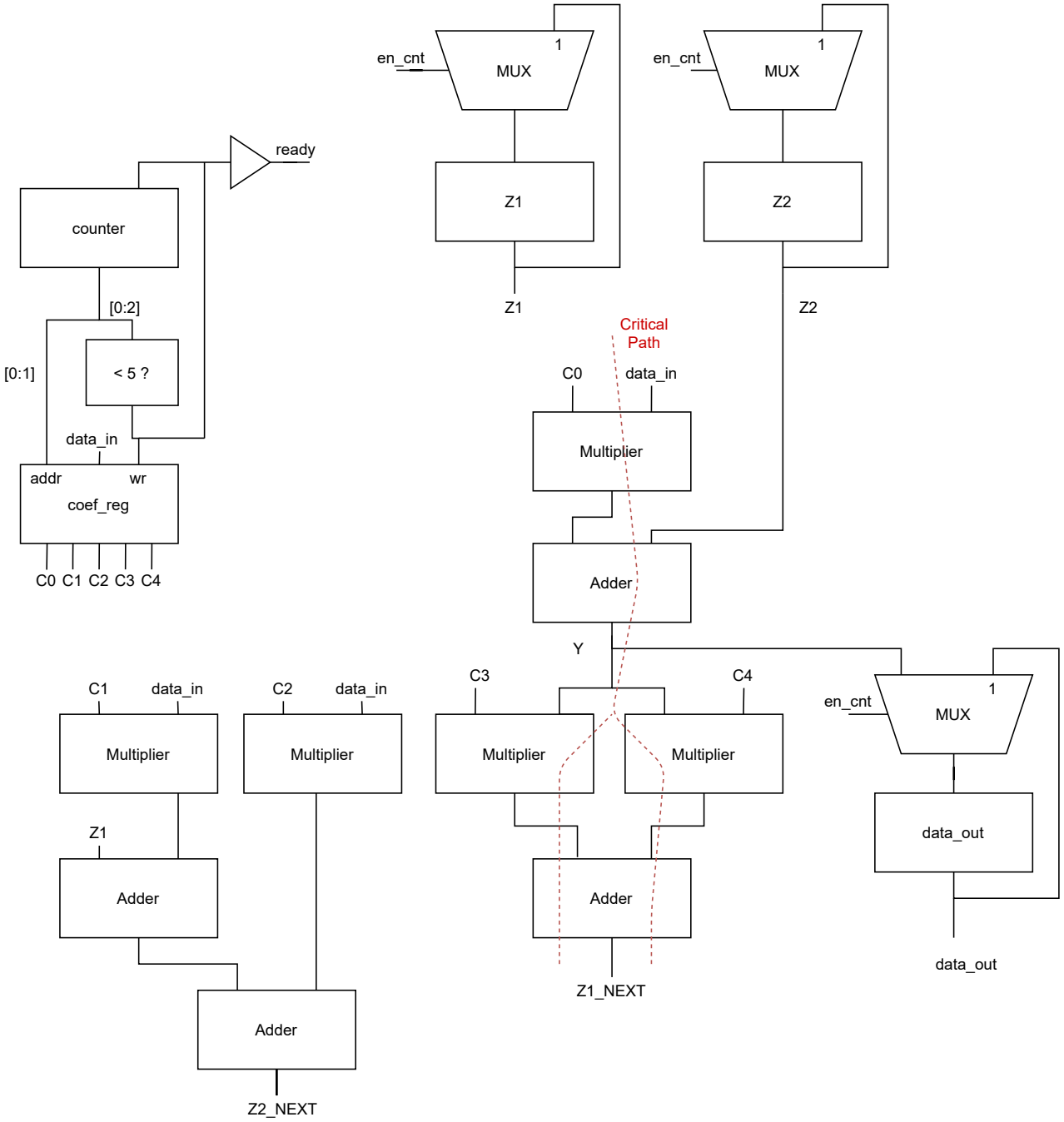
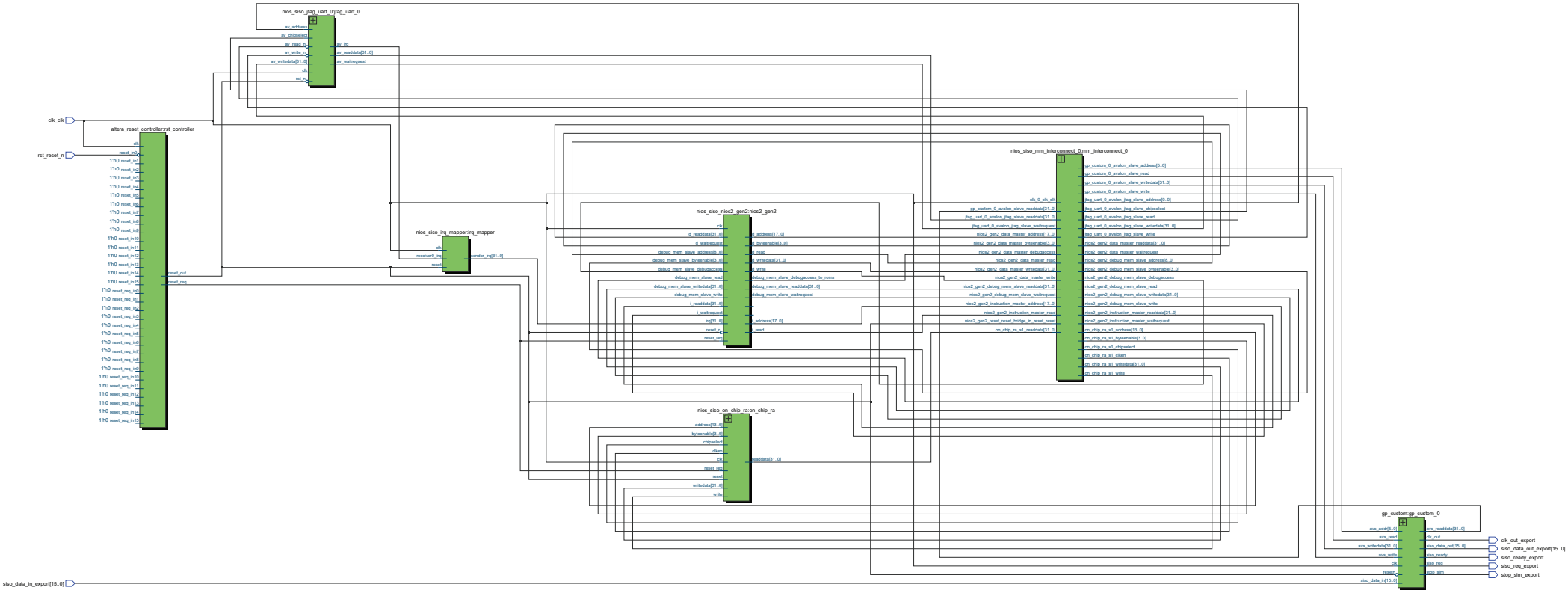
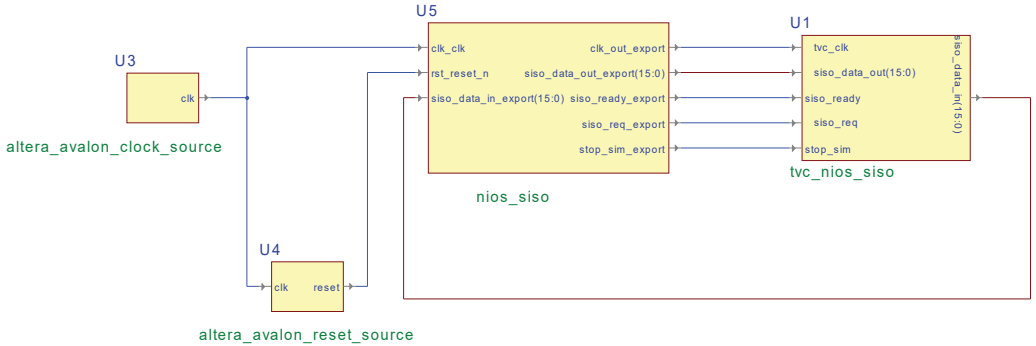


Figure 1: Data path of `sec_par` architecture

The critical path has a length of 4.



Block Diagram of the NIOS-II System



Block Diagram of the testbench

NIOS - 5

To evaluate the performance of the `sec_soft.c` the `simplePerformace`[1] architecture for `gp_custom` has been introduced. This architecture allows the user to trigger a start and stop event from `C` codes by means of functions `tic()` and `toc()`. Once the stop event has been triggered the number of clock cycles required to execute the code between this two events is reported in the `Modelsim` log as well as in a plain text file (`performance.txt`). The number of clock cycles needed for the execution of each of the 4 blocks without any optimization is depicted in Figure 2. From the analysis of the `performance.txt` file, when no compiler optimizations are turned on, we deduce that: the **maximum clock cycles for one multiplication** is 1736, the **minimum clock cycles for one multiplication** is 635 and the **average clock cycles for one multiplication** is 1083.

```
# --> Start of sec_soft <--
# ** Note: Clock Cycles: 86449
#    Time: 9326550 ns  Iteration: 1  Instance: /tb_nios_siso/nios_system/gp_custom_0
# Block nr. = 1
# ** Note: Clock Cycles: 86484
#    Time: 19037450 ns  Iteration: 1  Instance: /tb_nios_siso/nios_system/gp_custom_0
# Block nr. = 2
# ** Note: Clock Cycles: 86484
#    Time: 28748350 ns  Iteration: 1  Instance: /tb_nios_siso/nios_system/gp_custom_0
# Block nr. = 3
# ** Note: Clock Cycles: 86477
#    Time: 38458550 ns  Iteration: 1  Instance: /tb_nios_siso/nios_system/gp_custom_0
# Block nr. = 4
# ** Failure: OK! Simulation stopped at end of input file.
```

Block	Clock Cycles
1	86449
2	86484
3	86484
4	86477

Figure 2: Number of clock cycles needed for the execution of each of the 4 blocks. Compiler configuration: `-save-temps -O0`

The number of clock cycles needed for the execution of each of the 4 blocks with `-O3` optimization is depicted in Figure 3. From the analysis of the `performance.txt` file, when `-O3` optimization is turned on, we deduce that: the **maximum clock cycles for one multiplication** is 1417, the **minimum clock cycles for one multiplication** is 9 and the **average clock cycles for one multiplication** is 147. The compiler optimization has reduced the average multiplication time by almost a factor of 10.

```
# --> Start of sec_soft <--
# ** Note: Clock Cycles: 35845
#    Time: 4260350 ns  Iteration: 1  Instance: /tb_nios_siso/nios_system/gp_custom_0
# Block nr. = 1
# ** Note: Clock Cycles: 61246
#    Time: 11434950 ns  Iteration: 1  Instance: /tb_nios_siso/nios_system/gp_custom_0
# Block nr. = 2
# ** Note: Clock Cycles: 61176
#    Time: 18602550 ns  Iteration: 1  Instance: /tb_nios_siso/nios_system/gp_custom_0
# Block nr. = 3
# ** Note: Clock Cycles: 56913
#    Time: 25343850 ns  Iteration: 1  Instance: /tb_nios_siso/nios_system/gp_custom_0
# Block nr. = 4
# ** Failure: OK! Simulation stopped at end of input file.
```

Block	Clock Cycles
1	35845
2	61246
3	61176
4	56913

Figure 3: Number of clock cycles needed for the execution of each of the 4 blocks. Compiler configuration: `-save-temps -O3`

The four blocks have slightly different execution time because the multiplication function execution time depend from the input.

[1] This architecture cannot be synthesized.

NIOS 6

Data Path & Memory Organization

The proposed `simple6` architecture is an extension of the `simple` architecture with one multiplier. This multiplier accepts as inputs numbers in the `Q2.8` notation and returns an output in the same notation. The `out_buf` and `in_buf` have been extended to 14 locations. In particular the user now can store data from `C` code in 5 locations of the `out_buf`. The new multiplication function `hdMult2()` has as input two integers,the first one is the address of one of the 5 constants in `out_buf` and the second one (interpreted in the `simple6` in `Q2.8` notation) is the multiplicand.Data path and memory organization for `siso6` are shown in Figure 4 and 5.

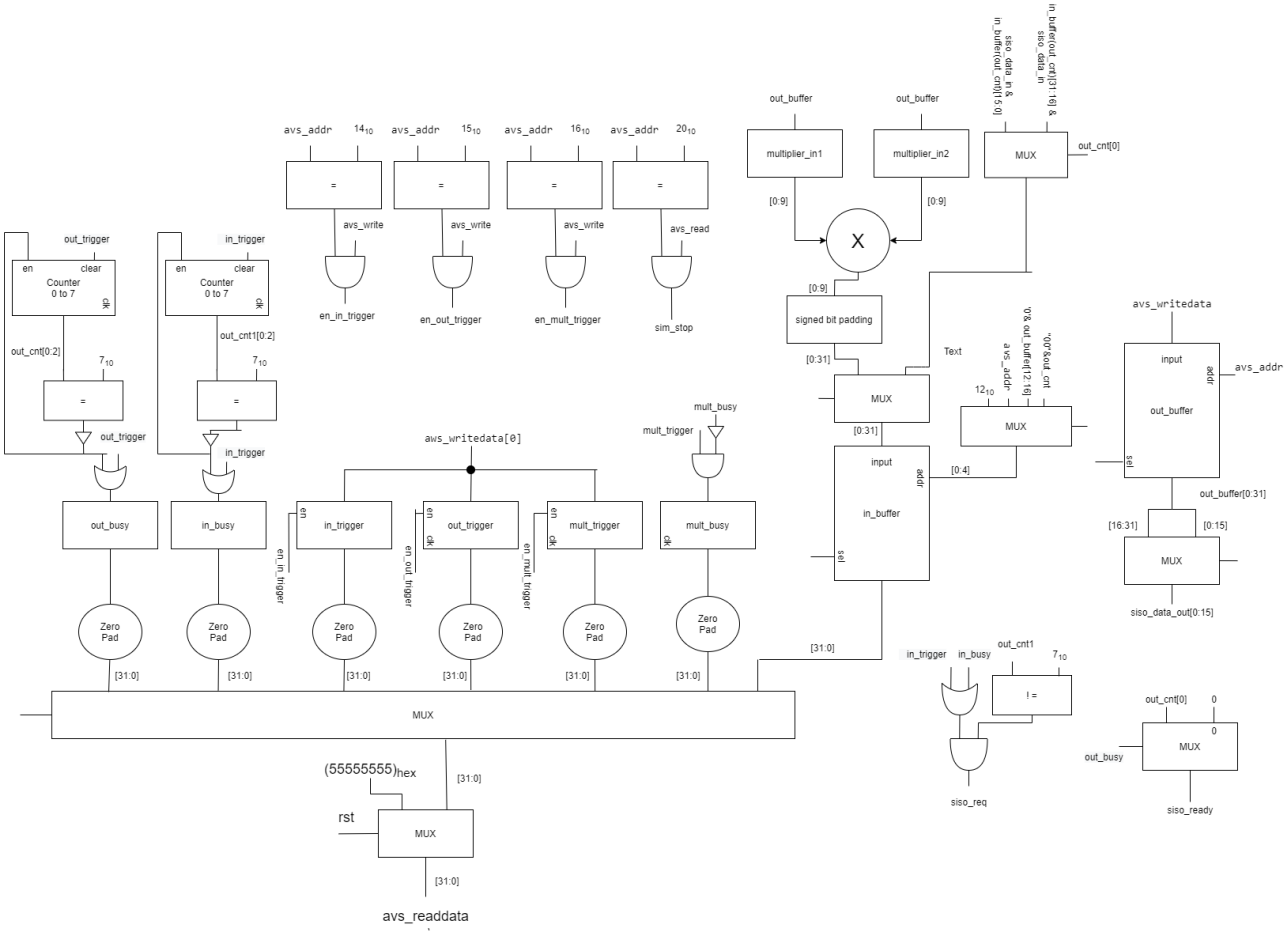


Figure 4: Data Path

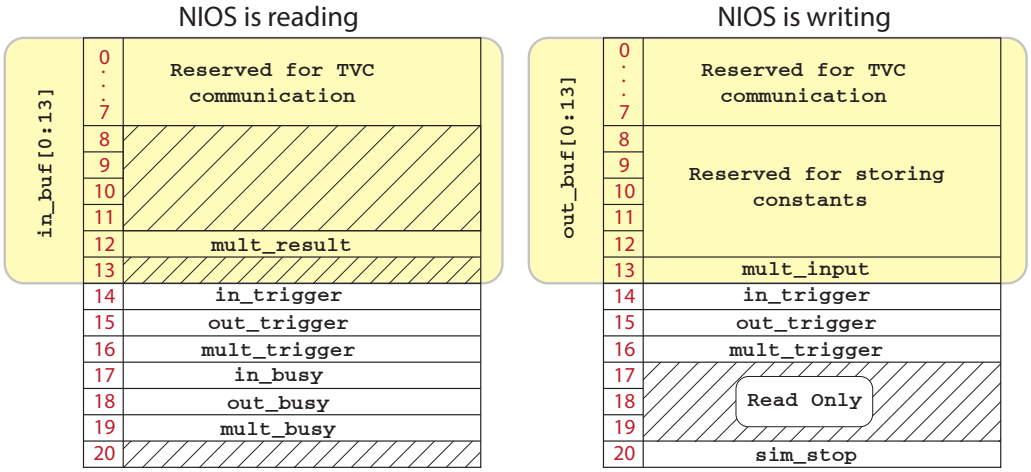


Figure 5: Memory Organization

Pre - Synthesis Simulation

The pre - synthesis simulation results are summarized in Figures 6, 7 and 8. All the compiler optimization have been deactivated. The result of the simulation was checked with the `diff` command.

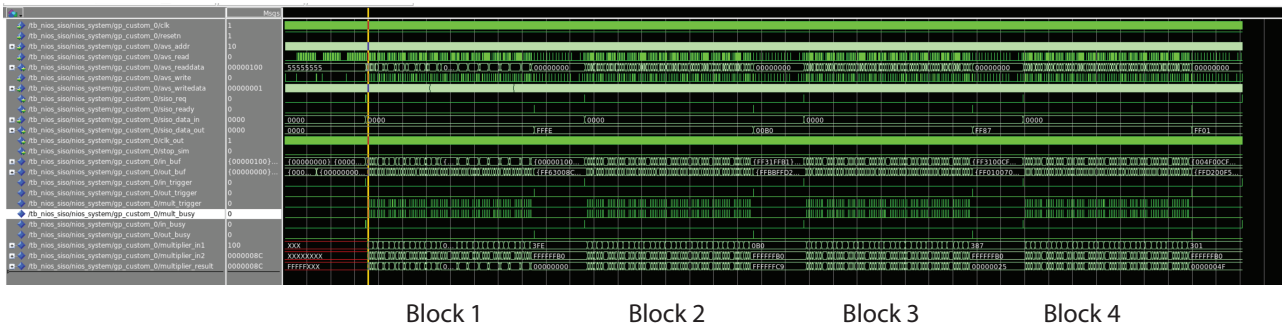


Figure 6: Pre - Synthesis Simulation simulation of `siso6` architecture (zoom full view).

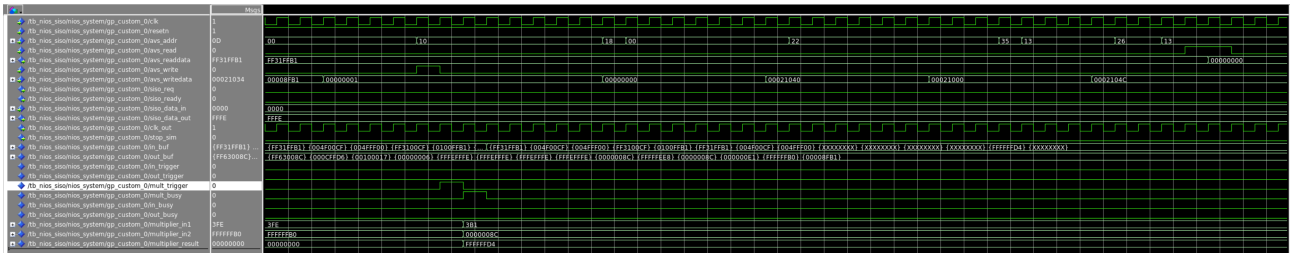


Figure 7: Example of multiplication event in `siso6` architecture

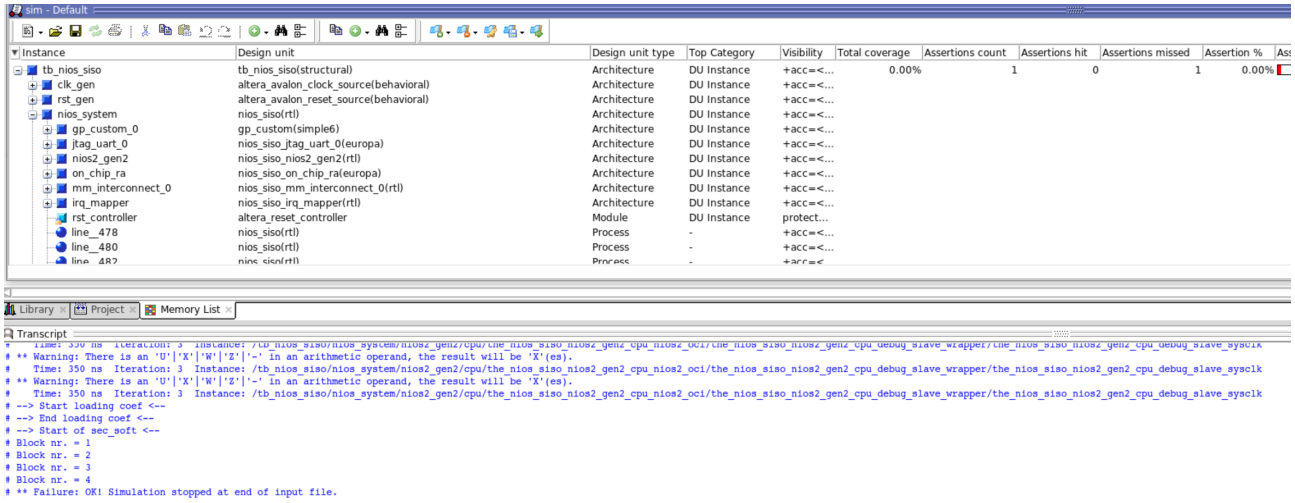


Figure 8: Modelsim Log

Post - Synthesis Simulation

The synthesized design was successfully simulated. The result of the simulation was checked with the `diff` command. Figure 9 depicts post simulation wave forms for the `simple6` architecture. All the compiler optimization have been deactivated.

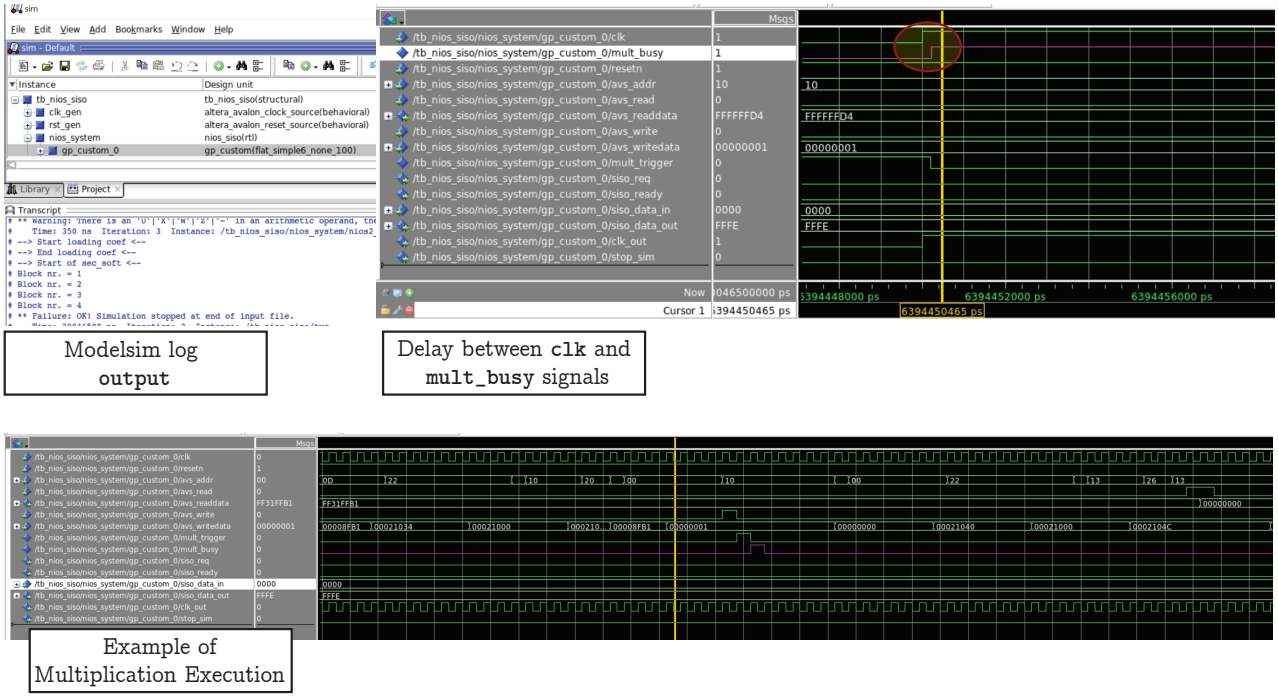


Figure 9: Post Simulation Waveform

Synthesis results are summarized in Figure 10.

D-Flip Flops Summary (Proposed Design)	
Cell Name	Count
DFEPQ1	66
DFERPQ1	580
DFESPQ1	1
DFFRPQ1	31
DFFSPQ1	15
Total	693

	Proposed Design	Given Design
Area [nm^2]	89567	68577
Slack [ns]	48.17	48.58

Figure 10: Synthesis summary

Performance Evaluation

To evaluate the improvements introduced by `simple6` architecture, clock cycles needed to perform the multiplication function and each of the four block have been computed. The computation has been carried out similarly as for `NIOS5` exercise. The result of the simulations are depicted respectively in Figures 11 and 12.

```
# ** Note: Clock Cycles: 35345
#    Time: 5266950 ns  Iteration: 1  Instance: /tb_nios_siso/nios_system/gp_custom_0
# Block nr. = 1
# ** Note: Clock Cycles: 35380
#    Time: 9867450 ns  Iteration: 1  Instance: /tb_nios_siso/nios_system/gp_custom_0
# Block nr. = 2
# ** Note: Clock Cycles: 35380
#    Time: 14467950 ns  Iteration: 1  Instance: /tb_nios_siso/nios_system/gp_custom_0
# Block nr. = 3
# ** Note: Clock Cycles: 35373
#    Time: 19067750 ns  Iteration: 1  Instance: /tb_nios_siso/nios_system/gp_custom_0
# Block nr. = 4
# ** Failure: OK! Simulation stopped at end of input file.
```

Block	Clock Cycles
1	35345
2	35380
3	35380
4	35373

Figure 11: Clock Cycles needed to execute the four different blocks. No optimization used in the compiler.

```
# --> Start loading coef <--
# --> End loading coef <--
# --> Start of sec_soft <--
# ** Note: Clock Cycles: 7138
#    Time: 2427850 ns  Iteration: 1  Instance: /tb_nios_siso/nios_system/gp_custom_0
# Block nr. = 1
# ** Note: Clock Cycles: 7103
#    Time: 4188150 ns  Iteration: 1  Instance: /tb_nios_siso/nios_system/gp_custom_0
# Block nr. = 2
# ** Note: Clock Cycles: 7103
#    Time: 5948450 ns  Iteration: 1  Instance: /tb_nios_siso/nios_system/gp_custom_0
# Block nr. = 3
# ** Note: Clock Cycles: 7110
#    Time: 7709450 ns  Iteration: 1  Instance: /tb_nios_siso/nios_system/gp_custom_0
# Block nr. = 4
# ** Failure: OK! Simulation stopped at end of input file.
```

Block	Clock Cycles
1	7138
2	7103
3	7103
4	7110

Figure 12: Clock Cycles needed to execute the four different blocks.Optimization `-O3` used in the compiler.

The comparison in terms of the number of clock cycles between the `simple6` and `simple` architecture is summarized in table of Figure 13.

	-O0 (No Optimization)		-O3 Optimization	
	Proposed Design	Given Design	Proposed Design	Given Design
Max. Multiply C.C.	465	1736	75	1417
Min. Multiply C.C.	413	635	61	9
Average Multiply C.C.	444	1083	68	147
Block 1 Clock Cycles	35345	86449	7138	35845
Block 2 Clock Cycles	35380	86484	7103	61246
Block 3 Clock Cycles	35380	86484	7103	61176
Block 4 Clock Cycles	35373	86477	7110	56913

Figure 13: Clock Cycle Comparison

C Code

The code reported below is `sec_soft6.c`. This new version of `sec_soft.c` makes use of the multiplier introduction in `simple6`.

```
 #include "sys/alt_stdio.h"
#include "system.h"


// start address of memory space (GP_CUSTOM_0_BASE is in system.h)
volatile unsigned int *IO_CUSTOM=(unsigned int *)GP_CUSTOM_0_BASE;

// declare some mnemonics

#define B0ADDR 8
#define B1ADDR 9
#define B2ADDR 10
#define A1ADDR 11
#define A2ADDR 12
#define IN_TRIGGER 14
#define OUT_TRIGGER 15
#define MULT_TRIGGER 16
#define STOP_SIM 20

#define IN_BUSY 17
#define OUT_BUSY 18
#define MULT_BUSY 19


// define hd multiply functions
static inline int hdMult2(int constantAddr, int mult_in)
{  int result;

   // write input
   IO_CUSTOM[13] = (( constantAddr << 12) & 0x0000F000) | (mult_in & 0x00000FFF);
   // init multiplication
   IO_CUSTOM[MULT_TRIGGER]=1;
   while (IO_CUSTOM[MULT_BUSY]) ;
   result = IO_CUSTOM[12];
   return result;
}

// top-level function name is always called "main"
int main()
{
  // state variables
  int z1 = 0;
  int z2 = 0;

  // constants
  const int b0 = 140;
  const int b1 = -280;
  const int b2 = 140;
  const int a1 = 225;
  const int a2 = -80;

    // write coef in buffer
  alt_putstr("--> Start loading coef <--\n");
  IO_CUSTOM[B0ADDR] = b0;
  IO_CUSTOM[B1ADDR] = b1;
  IO_CUSTOM[B2ADDR] = b2;
  IO_CUSTOM[A1ADDR] = a1;
  IO_CUSTOM[A2ADDR] = a2;
  alt_putstr("--> End loading coef <--\n");


  // temporary variables
  int m1, m2, m3, m4, m5;
  int m1_debug,  m2_debug, m3_debug, m4_debug, m5_debug;
  int z1_next, z2_temp, z2_next, y;

  // print to UART
  alt_putstr("--> Start of sec_soft <--\n");

  // repeat until end of input file is reached
  int block_count = 0;
  while (1) {
```

```
    // initiate data transfer from testbench to ioport
    IO_CUSTOM[IN_TRIGGER] = 1;
    // wait until transfer is ready
    while (IO_CUSTOM[IN_BUSY])
      ;

    // iteration indices
    int i,j;

    for (i=0; i <= 7; i++) {
      // each data transfer gives two signal samples
      int current_pair = IO_CUSTOM[i];

      // arrays of length two, to hold split input and output
      int in_pair[2], out_pair[2];

      // split pair; think of sign extension
      if (current_pair & 0x00008000)  // negative?
         in_pair[0] = current_pair | 0xFFFF0000;
      else
         in_pair[0] = current_pair & 0x0000FFFF;
      in_pair[1] = current_pair >> 16; // shift automatially extends sign

      // process both samples

      for (j=0; j <= 1; j++) {
        // multiplications and additions

        m1 = hdMult2( B0ADDR, in_pair[j] );
        m2 = hdMult2(B1ADDR, in_pair[j]);
        m4 = hdMult2(B2ADDR, in_pair[j]);
        y  = z2 + m1;
        m3 = hdMult2(A1ADDR, y);
        m5 = hdMult2(A2ADDR, y);

        z1_next = m4 + m5;
        z2_temp = z1 + m2;
        z2_next = z2_temp + m3;
        // state uptdate
        z1 = z1_next;
        z2 = z2_next;
        // result sample
        out_pair[j] = y;
      }
      // write output by combining output pair
      IO_CUSTOM[i] = ((out_pair[1] << 16) & 0xFFFF0000) |
                     (out_pair[0] & 0x0000FFFF);
    }

    // initiate data transfer from ioport to testbench
    IO_CUSTOM[OUT_TRIGGER] = 1;
    while (IO_CUSTOM[OUT_BUSY]) ;

    // print sign of life
    // alt_printf only supports %x, %s, %c, and %%
    alt_printf("Block nr. = %x\n", ++block_count);
  }
}
```

2 Delivered Files

- `gp_custom_simple6_arch.vhd`: Proposed architecture to replace `simple`.
- `gp_custom_simple6Performance_arch.vhd`: This architecture is used to evaluate the performance of the `sec_soft6.c`. It has the same content as `gp_custom_simple6_arch.vhd`, but with and additional process which allows the user to trigger start and stop event. The architecture is not designed to be synthesized (simulation only).
- `gp_custom_simplePerformance_arch.vhd`:This architecture is used to evaluate the performance of the `sec_soft.c`. It has the same content as `gp_custom_simple6_arch.vhd`, but with and additional process which allows the user to trigger start and stop event. The architecture is not designed to be synthesized (simulation only).

- `conf_tb_nios_siso_sec6_soft.vhd`: Configuration for the pre simulation of `simple6` architecture.
- `conf_tb_nios_siso_sec6_soft_post.vhd`:Configuration for the post simulation of `simple6` architecture.
- `conf_tb_nios_siso_sec6Performance_soft.vhd`: Configuration to simulate the architecture in `gp_custom_simple6Performance_arch.vhd`
- `conf_tb_nios_siso_secPerformance_soft.vhd`: Configuration to simulate the architecture in `gp_custom_simplePerformance_arch.vhd`
- `sec_soft6.c`: Proposed `C` code to make use of the multiplier introduced in `simple6`.
- `sec_soft6Perf.c`: Same as `sec_soft6.c`, but with new functions `tic()` and `toc()`. These functions are used to trigger start and stop event introduced in `gp_custom_simple6Performance_arch.vhd`.
- `sec_softPerf.c`:Same as `sec_soft.c`, but with new functions `tic()` and `toc()`. These functions are used to trigger start and stop event introduced in `gp_custom_simplePerformance_arch.vhd`.

```vhdl
library on_chip_ra;

configuration conf_tb_nios_siso_sec6_soft of tb_nios_siso is
  for structural
    for nios_system: nios_siso use entity work.nios_siso(rtl);
      for rtl
        for gp_custom_0: gp_custom use entity work.gp_custom(simple6);
        end for;
        for on_chip_ra: nios_siso_on_chip_ra
          use entity on_chip_ra.nios_siso_on_chip_ra(europa)
            generic map (INIT_FILE => "my_software/sec_soft6.hex");
        end for;
      end for;
    end for;
    for tvc: tvc_nios_siso use entity work.tvc_nios_siso(file_io)
          generic map (in_file_name =>  "sec_soft.in",
                       out_file_name => "sec_soft.out");
    end for;

    for clk_gen : altera_avalon_clock_source use entity
work.altera_avalon_clock_source(behavioral)
        generic map (
            -- default clock rate 10 MHz (100ns)
            CLOCK_RATE => 10,  -- clock rate
            CLOCK_UNIT => 1000000  -- clock rate unit MHz / kHz / Hz
        );
    end for;
  end for;
end conf_tb_nios_siso_sec6_soft;

```

```
1  library on_chip_ra;
2
3  configuration conf_tb_nios_siso_sec6_soft_post of tb_nios_siso is
4    for structural
5      for nios_system: nios_siso use entity work.nios_siso(rtl);
6        for rtl
7          for gp_custom_0: gp_custom use entity work.gp_custom(flat_simple6_none_100);
8          end for;
9          for on_chip_ra: nios_siso_on_chip_ra
10           use entity on_chip_ra.nios_siso_on_chip_ra(europa)
11             generic map (INIT_FILE => "my_software/sec_soft6.hex");
12         end for;
13       end for;
14     end for;
15     for tvc: tvc_nios_siso use entity work.tvc_nios_siso(file_io)
16           generic map (in_file_name =>  "sec_soft.in",
17                        out_file_name => "sec_soft.out");
18     end for;
19
20     for clk_gen : altera_avalon_clock_source use entity
work.altera_avalon_clock_source(behavioral)
21         generic map (
22             -- default clock rate 10 MHz (100ns)
23             CLOCK_RATE => 10,  -- clock rate
24             CLOCK_UNIT => 1000000  -- clock rate unit MHz / kHz / Hz
25         );
26     end for;
27   end for;
28 end conf_tb_nios_siso_sec6_soft_post;
29
```

```
library on_chip_ra;

configuration conf_tb_nios_siso_sec6Performance_soft of tb_nios_siso is
  for structural
    for nios_system: nios_siso use entity work.nios_siso(rtl);
      for rtl
        for gp_custom_0: gp_custom use entity work.gp_custom(simple6Performance);
        end for;
        for on_chip_ra: nios_siso_on_chip_ra
          use entity on_chip_ra.nios_siso_on_chip_ra(europa)
            generic map (INIT_FILE => "my_software/sec_soft6Perf.hex");
        end for;
      end for;
    end for;
    for tvc: tvc_nios_siso use entity work.tvc_nios_siso(file_io)
          generic map (in_file_name =>  "sec_soft.in",
                       out_file_name => "sec_soft.out");
    end for;

    for clk_gen : altera_avalon_clock_source use entity
work.altera_avalon_clock_source(behavioral)
        generic map (
            -- default clock rate 10 MHz (100ns)
            CLOCK_RATE => 10,  -- clock rate
            CLOCK_UNIT => 1000000  -- clock rate unit MHz / kHz / Hz
        );
    end for;
  end for;
end conf_tb_nios_siso_sec6Performance_soft;

```

```
1  library on_chip_ra;
2
3  configuration conf_tb_nios_siso_sec_softPerformance of tb_nios_siso is
4    for structural
5      for nios_system: nios_siso use entity work.nios_siso(rtl);
6        for rtl
7          for gp_custom_0: gp_custom use entity work.gp_custom(simplePerformace);
8          end for;
9          for on_chip_ra: nios_siso_on_chip_ra
10           use entity on_chip_ra.nios_siso_on_chip_ra(europa)
11             generic map (INIT_FILE => "my_software/sec_softPerf.hex");
12         end for;
13       end for;
14     end for;
15     for tvc: tvc_nios_siso use entity work.tvc_nios_siso(file_io)
16           generic map (in_file_name =>  "sec_soft.in",
17                        out_file_name => "sec_soft.out");
18     end for;
19
20     for clk_gen : altera_avalon_clock_source use entity
   work.altera_avalon_clock_source(behavioral)
21     generic map (
22         -- default clock rate 10 MHz (100ns)
23         CLOCK_RATE => 10,  -- clock rate
24         CLOCK_UNIT => 1000000  -- clock rate unit MHz / kHz / Hz
25     );
26     end for;
27
28   end for;
29 end conf_tb_nios_siso_sec_softPerformance;
30
```

```vhdl
-- This is the proposed architecture to replace the 'simple' architecture.
architecture simple6 of gp_custom is

    type buf_memory is array (0 to 13) of std_logic_vector (31 downto 0);
    --type buf_coef is array (8 to 12) of std_logic_vector (31 downto 0);

    -- define multiplication function
    function mult (a, b: signed(9 downto 0)) return std_logic_vector is
        variable signed_mult: signed(19 downto 0);
        variable temp : std_logic_vector(9 downto 0);
        variable result : std_logic_vector(31 downto 0);
        constant onePad : std_logic_vector(21 downto 0) := (others=>'1');
        constant zeroPad : std_logic_vector(21 downto 0) := (others=>'0');
    begin
        signed_mult := a * b;
        temp := std_logic_vector(signed_mult(17 downto 8)); -- throw away 2 MSBs and reduce to 'word_length=10'
        if temp(9)='0' then
            result:=  zeroPad & temp;
        else
            result:=  onePad & temp;
        end if ;
        return result;
    end mult;


    signal in_buf, out_buf: buf_memory;
    --signal filter_coef : buf_coef;

    signal in_trigger, out_trigger: std_logic;
    signal mult_trigger : std_logic;

    signal in_busy, out_busy, mult_busy: std_logic;
    signal multiplier_in1: signed(9 downto 0);
    signal multiplier_in2: signed(31 downto 0);
    signal multiplier_result : std_logic_vector(31 downto 0);

  begin

    -- bus interface
    bus_if:  process(resetn, clk)
      variable i: integer range 0 to 7;
    begin
      if resetn = '0'
      then
        for i in 0 to 12 loop
          out_buf(i) <= (others => '0');
        end loop;

        in_trigger <= '0';
        out_trigger <= '0';
        mult_trigger <= '0';
        stop_sim <= '0';
        avs_readdata <= X"55555555"; -- alternating 0/1 pattern
      elsif rising_edge(clk)
      then
        if avs_write = '1' -- Nios is writing
        then
          case to_integer(unsigned(avs_addr)) is
            when 0 to 13 => -- write memory
```

```
60              out_buf(to_integer(unsigned(avs_addr))) <= avs_writedata;
61            when 14 => -- request new external data
62              in_trigger <= avs_writedata(0); -- only LSB matters!
63            when 15 => -- request data flush to external
64              out_trigger <= avs_writedata(0); -- only LSB matters!
65            when 16 => -- request multiplication function
66              mult_trigger <= avs_writedata(0); -- only LSB matters!
67            when 20 => -- request to stop simulation
68              stop_sim <= '1'; -- ignore data value
69            when others => null;
70          end case;
71        else
72          -- clear triggers; they should be cleared within 16 clock
73          -- cycles after setting them; it is pretty safe to use
74          -- the absence of "avs_write" for this purpose.
75          in_trigger  <= '0';
76          out_trigger <= '0';
77          mult_trigger <= '0';
78          if avs_read = '1'  -- Nios is reading
79          then
80            case to_integer(unsigned(avs_addr)) is
81              when 0 to 13 => -- read memory
82                avs_readdata <= in_buf(to_integer(unsigned(avs_addr)));
83              when 14 => -- request new external data
84                avs_readdata <= (31 downto 1 => '0', 0 => in_trigger);
85              when 15 => -- request data flush to external
86                avs_readdata <= (31 downto 1 => '0', 0 => out_trigger);
87              when 16 =>
88                avs_readdata <= (31 downto 1 => '0', 0 => mult_trigger);
89              when 17 => -- input from external ready?
90                avs_readdata <= (31 downto 1 => '0', 0 => in_busy);
91              when 18 => -- output to external ready?
92                avs_readdata <= (31 downto 1 => '0', 0 => out_busy);
93              when 19 => -- is mult ready ?
94                avs_readdata <= (31 downto 1 => '0', 0 => mult_busy);
95              when others =>
96                avs_readdata <= X"55555555"; -- alternating 0/1 pattern
97            end case;
98          end if; -- avs_read
99        end if; -- avs_write
100     end if; -- rising edge
101   end process bus_if;
102
103   -- input buffer
104   inputs: process(resetn, clk)
105     variable i: integer range 0 to 7;
106     variable in_counter: integer range 0 to 7;
107     variable odd: std_logic;
108   begin
109     if resetn = '0'
110     then
111       for i in 0 to 7 loop
112         in_buf(i) <= (others => '0');
113       end loop;
114       in_busy <= '0';
115       siso_req <= '0';
116       in_counter := 0;
117       odd := '0';
118     elsif rising_edge(clk)
119     then
```

```
120         in_buf(12) <= multiplier_result ;
121         if in_busy = '1'
122         then
123           if odd = '0'
124           then
125             in_buf(in_counter)(15 downto 0) <= siso_data_in;
126             odd := '1';
127           else -- odd = '1'
128             in_buf(in_counter)(31 downto 16) <= siso_data_in;
129             odd := '0';
130             if in_counter = 7
131             then
132               siso_req <= '0';
133               in_busy <= '0';
134             else
135               in_counter := in_counter + 1;
136             end if;
137           end if;
138         elsif in_trigger = '1'
139         then
140           in_busy <= '1';
141           siso_req <= '1';
142           in_counter := 0;
143         end if;
144       end if;
145     end process inputs;
146
147     -- output buffer
148     outputs: process(resetn, clk)
149       variable out_counter: integer range 0 to 7;
150       variable odd: std_logic;
151     begin
152       if resetn = '0'
153       then
154         siso_data_out <= (others => '0');
155         out_busy <= '0';
156         siso_ready <= '0';
157         out_counter := 0;
158         odd := '0';
159       elsif rising_edge(clk)
160       then
161         if out_busy = '1'
162         then
163           if odd = '0'
164           then
165             siso_data_out <= out_buf(out_counter)(15 downto 0);
166             odd := '1';
167             siso_ready <= '1';
168           else
169             siso_data_out <= out_buf(out_counter)(31 downto 16);
170             odd := '0';
171             siso_ready <= '1';
172             if out_counter = 7
173             then
174               out_busy <= '0';
175             else
176               out_counter := out_counter + 1;
177             end if;
178           end if;
179         else
```

```
180           siso_ready <= '0';
181           if out_trigger = '1'
182           then
183             out_busy <= '1';
184             out_counter := 0;
185           end if;
186         end if;
187       end if;
188     end process outputs;
189
190
191     process_mult : process(resetn, clk)
192       variable mult_addr: integer;
193       variable tmp_data : std_logic_vector(31 downto 0);
194     begin
195       if resetn = '0' then
196         -- reset action
197         mult_addr := 7;
198         mult_busy <= '0';
199       elsif rising_edge(clk) then
200           if mult_busy = '1' then
201             mult_busy <= '0';
202           else
203               if mult_trigger = '1' then
204                   mult_busy <= '1';
205
206                   -- if (mult_addr = 12) then
207                   --   mult_addr := 8;
208                   -- else
209                   --   mult_addr :=  mult_addr + 1;
210                   -- end if;
211                   tmp_data := out_buf(13);
212                   mult_addr := to_integer(unsigned( tmp_data(16 downto 12) )) ;
213                   multiplier_in1 <= signed(tmp_data(9 downto 0));
214                   multiplier_in2 <= signed(out_buf(mult_addr));
215               end if ; -- mult_trigger
216           end if; -- end mult busy
217       end if ;
218    end process ; -- process_mult
219
220     -- connect clock for SISO
221     clk_out <= clk;
222     multiplier_result <= mult(multiplier_in1, multiplier_in2(9 downto 0));
223   end architecture simple6; -- of gp_custom
224
```

```vhdl
-- This NOT the proposed architecture
-- This architecture is for debug of the C code only
-- Do not attempt to synthesize it.
library ieee, std;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_textio.all;
use std.textio.all;

architecture simple6Performance of gp_custom is

    type buf_memory is array (0 to 13) of std_logic_vector (31 downto 0);
    --type buf_coef is array (8 to 12) of std_logic_vector (31 downto 0);

    -- define multiplication function
    function mult (a, b: signed(9 downto 0)) return std_logic_vector is
        variable signed_mult: signed(19 downto 0);
        variable temp : std_logic_vector(9 downto 0);
        variable result : std_logic_vector(31 downto 0);
        constant onePad : std_logic_vector(21 downto 0) := (others=>'1');
        constant zeroPad : std_logic_vector(21 downto 0) := (others=>'0');
    begin
        signed_mult := a * b;
        temp := std_logic_vector(signed_mult(17 downto 8)); -- throw away 2 MSBs and reduce to 'word_length=10'
        if temp(9)='0' then
            result:=  zeroPad & temp;
        else
            result:=  onePad & temp;
        end if ;
        return result;
    end mult;


    signal in_buf, out_buf: buf_memory;
    --signal filter_coef : buf_coef;

    signal in_trigger, out_trigger: std_logic;
    signal mult_trigger : std_logic;

    signal in_busy, out_busy, mult_busy: std_logic;
    signal multiplier_in1: signed(9 downto 0);
    signal multiplier_in2: signed(31 downto 0);
    signal multiplier_result : std_logic_vector(31 downto 0);

    -- Performance
    signal start_performance, stop_performance : std_logic;
    file file_RESULTS: text;

  begin

    -- bus interface
    bus_if:  process(resetn, clk)
      variable i: integer range 0 to 7;
    begin
      if resetn = '0'
      then
        for i in 0 to 12 loop
          out_buf(i) <= (others => '0');
        end loop;
```

```
60
61          in_trigger <= '0';
62          out_trigger <= '0';
63          mult_trigger <= '0';
64          stop_sim <= '0';
65          start_performance <= '0';
66          stop_performance <= '0';
67          avs_readdata <= X"55555555"; -- alternating 0/1 pattern
68        elsif rising_edge(clk)
69        then
70          if avs_write = '1' -- Nios is writing
71          then
72            case to_integer(unsigned(avs_addr)) is
73              when 0 to 13 => -- write memory
74                out_buf(to_integer(unsigned(avs_addr))) <= avs_writedata;
75              when 14 => -- request new external data
76                in_trigger <= avs_writedata(0); -- only LSB matters!
77              when 15 => -- request data flush to external
78                out_trigger <= avs_writedata(0); -- only LSB matters!
79              when 16 => -- request multiplication function
80                mult_trigger <= avs_writedata(0); -- only LSB matters!
81              when 20 => -- request to stop simulation
82                stop_sim <= '1'; -- ignore data value
83              when 30 =>
84                start_performance <= avs_writedata(0);
85              when 31 =>
86                stop_performance <= avs_writedata(0);
87              when others => null;
88            end case;
89          else
90            -- clear triggers; they should be cleared within 16 clock
91            -- cycles after setting them; it is pretty safe to use
92            -- the absence of "avs_write" for this purpose.
93            in_trigger  <= '0';
94            out_trigger <= '0';
95            mult_trigger <= '0';
96            start_performance <= '0';
97            stop_performance <= '0';
98            if avs_read = '1'  -- Nios is reading
99            then
100             case to_integer(unsigned(avs_addr)) is
101               when 0 to 13 => -- read memory
102                 avs_readdata <= in_buf(to_integer(unsigned(avs_addr)));
103               when 14 => -- request new external data
104                 avs_readdata <= (31 downto 1 => '0', 0 => in_trigger);
105               when 15 => -- request data flush to external
106                 avs_readdata <= (31 downto 1 => '0', 0 => out_trigger);
107               when 16 =>
108                 avs_readdata <= (31 downto 1 => '0', 0 => mult_trigger);
109               when 17 => -- input from external ready?
110                 avs_readdata <= (31 downto 1 => '0', 0 => in_busy);
111               when 18 => -- output to external ready?
112                 avs_readdata <= (31 downto 1 => '0', 0 => out_busy);
113               when 19 => -- is mult ready ?
114                 avs_readdata <= (31 downto 1 => '0', 0 => mult_busy);
115               when others =>
116                 avs_readdata <= X"55555555"; -- alternating 0/1 pattern
117             end case;
118           end if; -- avs_read
119         end if; -- avs_write
```

```
120       end if; -- rising edge
121     end process bus_if;
122
123     -- input buffer
124     inputs: process(resetn, clk)
125       variable i: integer range 0 to 7;
126       variable in_counter: integer range 0 to 7;
127       variable odd: std_logic;
128     begin
129       if resetn = '0'
130       then
131         for i in 0 to 7 loop
132           in_buf(i) <= (others => '0');
133         end loop;
134         in_busy <= '0';
135         siso_req <= '0';
136         in_counter := 0;
137         odd := '0';
138       elsif rising_edge(clk)
139       then
140         in_buf(12) <= multiplier_result ;
141         if in_busy = '1'
142         then
143           if odd = '0'
144           then
145             in_buf(in_counter)(15 downto 0) <= siso_data_in;
146             odd := '1';
147           else -- odd = '1'
148             in_buf(in_counter)(31 downto 16) <= siso_data_in;
149             odd := '0';
150             if in_counter = 7
151             then
152               siso_req <= '0';
153               in_busy <= '0';
154             else
155               in_counter := in_counter + 1;
156             end if;
157           end if;
158         elsif in_trigger = '1'
159         then
160           in_busy <= '1';
161           siso_req <= '1';
162           in_counter := 0;
163         end if;
164       end if;
165     end process inputs;
166
167     -- output buffer
168     outputs: process(resetn, clk)
169       variable out_counter: integer range 0 to 7;
170       variable odd: std_logic;
171     begin
172       if resetn = '0'
173       then
174         siso_data_out <= (others => '0');
175         out_busy <= '0';
176         siso_ready <= '0';
177         out_counter := 0;
178         odd := '0';
179       elsif rising_edge(clk)
```

```
180       then
181         if out_busy = '1'
182         then
183           if odd = '0'
184           then
185             siso_data_out <= out_buf(out_counter)(15 downto 0);
186             odd := '1';
187             siso_ready <= '1';
188           else
189             siso_data_out <= out_buf(out_counter)(31 downto 16);
190             odd := '0';
191             siso_ready <= '1';
192             if out_counter = 7
193             then
194               out_busy <= '0';
195             else
196               out_counter := out_counter + 1;
197             end if;
198           end if;
199         else
200           siso_ready <= '0';
201           if out_trigger = '1'
202           then
203             out_busy <= '1';
204             out_counter := 0;
205           end if;
206         end if;
207       end if;
208     end process outputs;
209
210
211     process_mult : process(resetn, clk)
212       variable mult_addr: integer;
213       variable tmp_data : std_logic_vector(31 downto 0);
214     begin
215       if resetn = '0' then
216         -- reset action
217         mult_addr := 7;
218         mult_busy <= '0';
219       elsif rising_edge(clk) then
220           if mult_busy = '1' then
221             mult_busy <= '0';
222           else
223               if mult_trigger = '1' then
224                   mult_busy <= '1';
225
226                   -- if (mult_addr = 12) then
227                   --   mult_addr := 8;
228                   -- else
229                   --   mult_addr :=  mult_addr + 1;
230                   -- end if;
231                   tmp_data := out_buf(13);
232                   mult_addr := to_integer(unsigned( tmp_data(16 downto 12) )) ;
233                   multiplier_in1 <= signed(tmp_data(9 downto 0));
234                   multiplier_in2 <= signed(out_buf(mult_addr));
235               end if ; -- mult_trigger
236           end if; -- end mult busy
237       end if ;
238     end process ; -- process_mult
239
```

```
240   process_performance : process(resetn, clk)
241     variable timer : integer;
242     variable can_cnt : std_logic;
243     variable line2write : line; -- line to write
244   begin
245    if resetn='0' then
246      timer := 0;
247      -- clear log file
248      file_open(file_RESULTS, "performance6.txt", write_mode);
249      write(line2write, string'("Performance6 Log"));
250      writeline(file_RESULTS, line2write);
251      file_close(file_RESULTS);
252
253    elsif (rising_edge(clk)) then
254      if start_performance='1' then
255        can_cnt := '1';
256        timer := 0;
257        file_open(file_RESULTS, "performance6.txt", append_mode );
258      end if ;
259
260      if stop_performance='1' then
261        can_cnt := '0';
262        write(line2write,timer);
263        writeline(file_RESULTS, line2write);
264        file_close(file_RESULTS);
265        report "Clock Cycles: " & integer'image(timer) severity note;
266      end if ;
267
268      if can_cnt = '1'  then
269        timer := timer +1;
270      end if ;
271
272    end if ;
273  end process ; -- process performance
274
275     -- connect clock for SISO
276     clk_out <= clk;
277     multiplier_result <= mult(multiplier_in1, multiplier_in2(9 downto 0));
278   end architecture simple6Performance; -- of gp_custom
279
```

```vhdl
-- This NOT the proposed architecture
-- This architecture is for debug of the C code only
-- Do not attempt to synthesize it.
library ieee, std;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_textio.all;
use std.textio.all;

architecture simplePerformace of gp_custom is

  type buf_memory is array (0 to 7) of std_logic_vector (31 downto 0);

  signal in_buf, out_buf: buf_memory;
  signal in_trigger, out_trigger: std_logic;

  signal in_busy, out_busy: std_logic;
  signal start_performance, stop_performance : std_logic;
  file file_RESULTS: text;
begin
  -- INPUT MAPPING --
  -- add <= avs_addr(5 downto 2);

  -- bus interface
  bus_if:  process(resetn, clk)
    variable i: integer range 0 to 7;
  begin
    if resetn = '0'
    then
      for i in 0 to 7 loop
        out_buf(i) <= (others => '0');
      end loop;
      in_trigger <= '0';
      out_trigger <= '0';
      stop_sim <= '0';
      start_performance <= '0';
      stop_performance <= '0';
      avs_readdata <= X"55555555"; -- alternating 0/1 pattern
    elsif rising_edge(clk)
    then
      if avs_write = '1' -- Nios is writing
      then
        case to_integer(unsigned(avs_addr)) is
          when 0 to 7 => -- write memory
            out_buf(to_integer(unsigned(avs_addr))) <= avs_writedata;
          when 8 => -- request new external data
            in_trigger <= avs_writedata(0); -- only LSB matters!
          when 9 => -- request data flush to external
            out_trigger <= avs_writedata(0); -- only LSB matters!
          when 12 => -- request to stop simulation
            stop_sim <= '1'; -- ignore data value
          when 30 =>
            start_performance <= avs_writedata(0);
          when 31 =>
            stop_performance <= avs_writedata(0);
          when others => null;
        end case;
      else
        -- clear triggers; they should be cleared within 16 clock
        -- cycles after setting them; it is pretty safe to use
```

```
61          -- the absence of "avs_write" for this purpose.
62          in_trigger  <= '0';
63          out_trigger <= '0';
64          start_performance <= '0';
65          stop_performance <= '0';
66          if avs_read = '1'  -- Nios is reading
67          then
68            case to_integer(unsigned(avs_addr)) is
69              when 0 to 7 => -- read memory
70                avs_readdata <= in_buf(to_integer(unsigned(avs_addr)));
71              when 8 => -- request new external data
72                avs_readdata <= (31 downto 1 => '0', 0 => in_trigger);
73              when 9 => -- request data flush to external
74                avs_readdata <= (31 downto 1 => '0', 0 => out_trigger);
75              when 10 => -- input from external ready?
76                avs_readdata <= (31 downto 1 => '0', 0 => in_busy);
77              when 11 => -- output to external ready?
78                avs_readdata <= (31 downto 1 => '0', 0 => out_busy);
79              when others =>
80                avs_readdata <= X"55555555"; -- alternating 0/1 pattern
81            end case;
82          end if; -- avs_read
83        end if; -- avs_write
84      end if; -- rising edge
85    end process bus_if;
86
87    -- input buffer
88    inputs: process(resetn, clk)
89      variable i: integer range 0 to 7;
90      variable in_counter: integer range 0 to 7;
91      variable odd: std_logic;
92    begin
93      if resetn = '0'
94      then
95        for i in 0 to 7 loop
96          in_buf(i) <= (others => '0');
97        end loop;
98        in_busy <= '0';
99        siso_req <= '0';
100       in_counter := 0;
101       odd := '0';
102     elsif rising_edge(clk)
103     then
104       if in_busy = '1'
105       then
106         if odd = '0'
107         then
108           in_buf(in_counter)(15 downto 0) <= siso_data_in;
109           odd := '1';
110         else -- odd = '1'
111           in_buf(in_counter)(31 downto 16) <= siso_data_in;
112           odd := '0';
113           if in_counter = 7
114           then
115             siso_req <= '0';
116             in_busy <= '0';
117           else
118             in_counter := in_counter + 1;
119           end if;
120         end if;
```

```
121       elsif in_trigger = '1'
122       then
123         in_busy <= '1';
124         siso_req <= '1';
125         in_counter := 0;
126       end if;
127     end if;
128   end process inputs;
129
130   -- output buffer
131   outputs: process(resetn, clk)
132     variable out_counter: integer range 0 to 7;
133     variable odd: std_logic;
134   begin
135     if resetn = '0'
136     then
137       siso_data_out <= (others => '0');
138       out_busy <= '0';
139       siso_ready <= '0';
140       out_counter := 0;
141       odd := '0';
142     elsif rising_edge(clk)
143     then
144       if out_busy = '1'
145       then
146         if odd = '0'
147         then
148           siso_data_out <= out_buf(out_counter)(15 downto 0);
149           odd := '1';
150           siso_ready <= '1';
151         else
152           siso_data_out <= out_buf(out_counter)(31 downto 16);
153           odd := '0';
154           siso_ready <= '1';
155           if out_counter = 7
156           then
157             out_busy <= '0';
158           else
159             out_counter := out_counter + 1;
160           end if;
161         end if;
162       else
163         siso_ready <= '0';
164         if out_trigger = '1'
165         then
166           out_busy <= '1';
167           out_counter := 0;
168         end if;
169       end if;
170     end if;
171   end process outputs;
172
173   process_performance : process(resetn, clk)
174     variable timer : integer;
175     variable can_cnt : std_logic;
176     variable line2write : line; -- line to write
177   begin
178     if resetn='0' then
179       timer := 0;
180       -- clear log file
```

```
181       file_open(file_RESULTS, "performance.txt", write_mode);
182       write(line2write, string'("Performance Log"));
183       writeline(file_RESULTS, line2write);
184       file_close(file_RESULTS);
185 
186     elsif (rising_edge(clk)) then
187       if start_performance='1' then
188         can_cnt := '1';
189         timer := 0;
190         file_open(file_RESULTS, "performance.txt", append_mode );
191       end if ;
192 
193       if stop_performance='1' then
194         can_cnt := '0';
195         write(line2write,timer);
196         writeline(file_RESULTS, line2write);
197         file_close(file_RESULTS);
198         report "Clock Cycles: " & integer'image(timer) severity note;
199       end if ;
200 
201       if can_cnt = '1'  then
202         timer := timer +1;
203       end if ;
204 
205     end if ;
206   end process ; -- process performance
207 
208   -- connect clock for SISO
209   clk_out <= clk;
210 
211 end architecture simplePerformace; -- of gp_custom
212 
```

```c
#include "sys/alt_stdio.h"
#include "system.h"


// start address of memory space (GP_CUSTOM_0_BASE is in system.h)
volatile unsigned int *IO_CUSTOM=(unsigned int *)GP_CUSTOM_0_BASE;

// declare some mnemonics

#define B0ADDR 8
#define B1ADDR 9
#define B2ADDR 10
#define A1ADDR 11
#define A2ADDR 12
#define IN_TRIGGER 14
#define OUT_TRIGGER 15
#define MULT_TRIGGER 16
#define STOP_SIM 20

#define IN_BUSY 17
#define OUT_BUSY 18
#define MULT_BUSY 19


// define hd multiply functions
static inline int hdMult2(int constantAddr, int mult_in)
{  int result;

   // write input
   IO_CUSTOM[13] = (( constantAddr << 12) & 0x0000F000) | (mult_in & 0x00000FFF);
   // init multiplication
   IO_CUSTOM[MULT_TRIGGER]=1;
   while (IO_CUSTOM[MULT_BUSY]) ;
   result = IO_CUSTOM[12];
   return result;
}

// top-level function name is always called "main"
int main()
{
  // state variables
  int z1 = 0;
  int z2 = 0;

  // constants
  const int b0 = 140;
  const int b1 = -280;
  const int b2 = 140;
  const int a1 = 225;
  const int a2 = -80;

    // write coef in buffer
  alt_putstr("--> Start loading coef <--\n");
  IO_CUSTOM[B0ADDR] = b0;
  IO_CUSTOM[B1ADDR] = b1;
  IO_CUSTOM[B2ADDR] = b2;
  IO_CUSTOM[A1ADDR] = a1;
  IO_CUSTOM[A2ADDR] = a2;
  alt_putstr("--> End loading coef <--\n");

```

```
61
62  // temporary variables
63  int m1, m2, m3, m4, m5;
64  int m1_debug,  m2_debug, m3_debug, m4_debug, m5_debug;
65  int z1_next, z2_temp, z2_next, y;
66
67  // print to UART
68  alt_putstr("--> Start of sec_soft <--\n");
69
70  // repeat until end of input file is reached
71  int block_count = 0;
72  while (1) {
73
74    // initiate data transfer from testbench to ioport
75    IO_CUSTOM[IN_TRIGGER] = 1;
76    // wait until transfer is ready
77    while (IO_CUSTOM[IN_BUSY])
78      ;
79
80    // iteration indices
81    int i,j;
82
83    for (i=0; i <= 7; i++) {
84      // each data transfer gives two signal samples
85      int current_pair = IO_CUSTOM[i];
86
87      // arrays of length two, to hold split input and output
88      int in_pair[2], out_pair[2];
89
90      // split pair; think of sign extension
91      if (current_pair & 0x00008000)  // negative?
92         in_pair[0] = current_pair | 0xFFFF0000;
93      else
94         in_pair[0] = current_pair & 0x0000FFFF;
95      in_pair[1] = current_pair >> 16; // shift automatially extends sign
96
97      // process both samples
98
99      for (j=0; j <= 1; j++) {
100       // multiplications and additions
101       //m1 = hdMult( in_pair[j] );
102       //m1 = fxmult_2_8(b0, in_pair[j]);
103       m1 = hdMult2( B0ADDR, in_pair[j] );
104
105       //alt_printf("--> m1= %x <--\n", m1);
106       //alt_printf("--> m1_debug = %x <--\n", m1_debug);
107
108       //m2 = fxmult_2_8(b1, in_pair[j]);
109       //m2 = hdMult( in_pair[j] );
110       m2 = hdMult2(B1ADDR, in_pair[j]);
111
112       //m4 = fxmult_2_8(b2, in_pair[j]);
113       //m4 = hdMult( in_pair[j] );
114       m4 = hdMult2(B2ADDR, in_pair[j]);
115
116       y  = z2 + m1;
117
118       //m3 = fxmult_2_8(a1, y);
119       //m3 = hdMult(y);
120       m3 = hdMult2(A1ADDR, y);
```

```
121
122         //m5 = fxmult_2_8(a2, y);
123         //m5 = hdMult(y);
124         m5 = hdMult2(A2ADDR, y);
125
126         z1_next = m4 + m5;
127         z2_temp = z1 + m2;
128         z2_next = z2_temp + m3;
129         // state uptdate
130         z1 = z1_next;
131         z2 = z2_next;
132         // result sample
133         out_pair[j] = y;
134       }
135       // write output by combining output pair
136       IO_CUSTOM[i] = ((out_pair[1] << 16) & 0xFFFF0000) |
137                      (out_pair[0] & 0x0000FFFF);
138     }
139
140     // initiate data transfer from ioport to testbench
141     IO_CUSTOM[OUT_TRIGGER] = 1;
142     while (IO_CUSTOM[OUT_BUSY])
143       ;
144
145     // print sign of life
146     // alt_printf only supports %x, %s, %c, and %%
147     alt_printf("Block nr. = %x\n", ++block_count);
148   }
149 }
150
```

```c
// This code is used for debug only
#include "sys/alt_stdio.h"
#include "system.h"


// start address of memory space (GP_CUSTOM_0_BASE is in system.h)
volatile unsigned int *IO_CUSTOM=(unsigned int *)GP_CUSTOM_0_BASE;

// declare some mnemonics

#define B0ADDR 8
#define B1ADDR 9
#define B2ADDR 10
#define A1ADDR 11
#define A2ADDR 12
#define IN_TRIGGER 14
#define OUT_TRIGGER 15
#define MULT_TRIGGER 16
#define STOP_SIM 20

#define IN_BUSY 17
#define OUT_BUSY 18
#define MULT_BUSY 19

#define TIC 30
#define TOC 31

// define hd multiply functions
static inline int hdMult2(int constantAddr, int mult_in)
{  int result;

   // write input
   IO_CUSTOM[13] = (( constantAddr << 12) & 0x0000F000) | (mult_in & 0x00000FFF);
   // init multiplication
   IO_CUSTOM[MULT_TRIGGER]=1;
   while (IO_CUSTOM[MULT_BUSY]) ;
   result = IO_CUSTOM[12];
   return result;
}

void tic(){
  IO_CUSTOM[TIC]=1;
}

void toc(){
  IO_CUSTOM[TOC]=1;
}


// top-level function name is always called "main"
int main()
{
  // state variables
  int z1 = 0;
  int z2 = 0;

  // constants
  const int b0 = 140;
  const int b1 = -280;
  const int b2 = 140;
```

```
61  const int a1 = 225;
62  const int a2 = -80;
63
64    // write coef in buffer
65  alt_putstr("--> Start loading coef <--\n");
66  IO_CUSTOM[B0ADDR] = b0;
67  IO_CUSTOM[B1ADDR] = b1;
68  IO_CUSTOM[B2ADDR] = b2;
69  IO_CUSTOM[A1ADDR] = a1;
70  IO_CUSTOM[A2ADDR] = a2;
71  alt_putstr("--> End loading coef <--\n");
72
73
74  // temporary variables
75  int m1, m2, m3, m4, m5;
76  int m1_debug,  m2_debug, m3_debug, m4_debug, m5_debug;
77  int z1_next, z2_temp, z2_next, y;
78
79  // print to UART
80  alt_putstr("--> Start of sec_soft <--\n");
81
82  // repeat until end of input file is reached
83  int block_count = 0;
84  while (1) {
85    tic();
86    // initiate data transfer from testbench to ioport
87    IO_CUSTOM[IN_TRIGGER] = 1;
88    // wait until transfer is ready
89    while (IO_CUSTOM[IN_BUSY])
90      ;
91
92    // iteration indices
93    int i,j;
94
95    for (i=0; i <= 7; i++) {
96      // each data transfer gives two signal samples
97      int current_pair = IO_CUSTOM[i];
98
99      // arrays of length two, to hold split input and output
100     int in_pair[2], out_pair[2];
101
102     // split pair; think of sign extension
103     if (current_pair & 0x00008000)  // negative?
104        in_pair[0] = current_pair | 0xFFFF0000;
105     else
106        in_pair[0] = current_pair & 0x0000FFFF;
107     in_pair[1] = current_pair >> 16; // shift automatially extends sign
108
109     // process both samples
110
111     for (j=0; j <= 1; j++) {
112       // multiplications and additions
113
114
115       m1 = hdMult2( B0ADDR, in_pair[j] );
116       m2 = hdMult2(B1ADDR, in_pair[j]);
117       m4 = hdMult2(B2ADDR, in_pair[j]);
118       y  = z2 + m1;
119       m3 = hdMult2(A1ADDR, y);
120       m5 = hdMult2(A2ADDR, y);
```

```
121
122         z1_next = m4 + m5;
123         z2_temp = z1 + m2;
124         z2_next = z2_temp + m3;
125         // state uptdate
126         z1 = z1_next;
127         z2 = z2_next;
128         // result sample
129         out_pair[j] = y;
130       }
131       // write output by combining output pair
132       IO_CUSTOM[i] = ((out_pair[1] << 16) & 0xFFFF0000) |
133                      (out_pair[0] & 0x0000FFFF);
134     }
135
136     // initiate data transfer from ioport to testbench
137     IO_CUSTOM[OUT_TRIGGER] = 1;
138     while (IO_CUSTOM[OUT_BUSY])
139       ;
140     toc();
141     // print sign of life
142     // alt_printf only supports %x, %s, %c, and %%
143     alt_printf("Block nr. = %x\n", ++block_count);
144   }
145 }
146
```

```
1  // --------------------------------------------------------------------------
2  // File         : sec_soft.c
3  // Description  : Full-software implementation of second-order filter
4  //                on nios_siso system
5  // Author       : Sabih Gerez, University of Twente
6  // Creation date: August 26, 2015
7  // --------------------------------------------------------------------------
8  // $Rev: 306 $
9  // $Author: gerezsh $
10 // $Date: 2019-09-26 01:17:22 +0200 (Thu, 26 Sep 2019) $
11 // $Log$
12 // --------------------------------------------------------------------------
13
14 #include "sys/alt_stdio.h"
15 #include "system.h"
16
17
18 // start address of memory space (GP_CUSTOM_0_BASE is in system.h)
19 volatile unsigned int *IO_CUSTOM=(unsigned int *)GP_CUSTOM_0_BASE;
20
21 // declare some mnemonics
22 #define IN_TRIGGER 8
23 #define OUT_TRIGGER 9
24 #define IN_BUSY 10
25 #define OUT_BUSY 11
26 #define STOP_SIM 12
27 #define TIC 30
28 #define TOC 31
29
30
31 // auxiliary function for fixed-point multiply (2 integer, 8
32 // fractional bits)
33
34 // inline int fxmult_2_8(int left, int right)
35 static inline int fxmult_2_8(int left, int right)
36 {
37   int tmp = left * right;
38   return (tmp >> 8);
39 }
40
41 void tic(){
42   IO_CUSTOM[TIC]=1;
43 }
44
45 void toc(){
46   IO_CUSTOM[TOC]=1;
47 }
48
49 // top-level function name is always called "main"
50 int main()
51 {
52
53   // state variables
54   int z1 = 0;
55   int z2 = 0;
56
57   // constants
58   const int b0 = 140;
59   const int b1 = -280;
60   const int b2 = 140;
```

```
61  const int a1 = 225;
62  const int a2 = -80;
63
64  // temporary variables
65  int m1, m2, m3, m4, m5;
66  int z1_next, z2_temp, z2_next, y;
67
68  // print to UART
69  alt_putstr("--> Start of sec_soft <--\n");
70
71  // repeat until end of input file is reached
72  int block_count = 0;
73  while (1) {
74
75
76    // initiate data transfer from testbench to ioport
77    IO_CUSTOM[IN_TRIGGER] = 1;
78    // wait until transfer is ready
79    while (IO_CUSTOM[IN_BUSY])
80      ;
81
82    // iteration indices
83    int i,j;
84
85    for (i=0; i <= 7; i++) {
86      // each data transfer gives two signal samples
87      int current_pair = IO_CUSTOM[i];
88
89      // arrays of length two, to hold split input and output
90      int in_pair[2], out_pair[2];
91
92      // split pair; think of sign extension
93      if (current_pair & 0x00008000)  // negative?
94         in_pair[0] = current_pair | 0xFFFF0000;
95      else
96         in_pair[0] = current_pair & 0x0000FFFF;
97      in_pair[1] = current_pair >> 16; // shift automatially extends sign
98
99      // process both samples
100
101     for (j=0; j <= 1; j++) {
102       // multiplications and additions
103
104       tic();
105       m1 = fxmult_2_8(b0, in_pair[j]);
106       toc();
107
108       tic();
109       m2 = fxmult_2_8(b1, in_pair[j]);
110       toc();
111
112       tic();
113       m4 = fxmult_2_8(b2, in_pair[j]);
114       toc();
115
116       y  = z2 + m1;
117       tic();
118       m3 = fxmult_2_8(a1, y);
119       toc();
120
```

```
121         tic();
122         m5 = fxmult_2_8(a2, y);
123         toc();
124 
125         z1_next = m4 + m5;
126         z2_temp = z1 + m2;
127         z2_next = z2_temp + m3;
128         // state uptdate
129         z1 = z1_next;
130         z2 = z2_next;
131         // result sample
132         out_pair[j] = y;
133       }
134       // write output by combining output pair
135       IO_CUSTOM[i] = ((out_pair[1] << 16) & 0xFFFF0000) |
136                      (out_pair[0] & 0x0000FFFF);
137     }
138 
139     // initiate data transfer from ioport to testbench
140     IO_CUSTOM[OUT_TRIGGER] = 1;
141     while (IO_CUSTOM[OUT_BUSY])
142       ;
143 
144     // print sign of life
145     // alt_printf only supports %x, %s, %c, and %%
146     alt_printf("Block nr. = %x\n", ++block_count);
147   }
148 
149 }
150 
```