

# Appendix F: Testbenches

## APPENDIX F: TESTBENCH INDEX

### **F.1 BUS MASTER**

Description: This testbench for the busmaster demonstrates how the weights are sent out on the data bus according to the signals asserted.

### **F.2 RAM**

Description: This testbench stores 256 words to separate addresses in the ram then reads them out in the same order. As it writes and reads in values, it the testbench stores these values to an associated file.

### **F.3 Bus Master Simulation**

### **F.4 RAM Simulation**

## F.1 Testbench for Bus Master

In the testbench for the busmaster we can see how according to the signal asserted the proper value of the weights is sent by the data out bus.

In page F.1.1 we can see how by asserting the signal “asser\_rand” the value coming from the random number generator is outputted to the data out bus. This value is stored in a register inside the bus, that is why it has some delay. We can also see that by incrementing the “incr\_sel” signal the Sel counter goes through all the weights of the specified neuron and through all the neurons. When all the weights of all neurons have been changed the “don” signal goes high.

In page F.1.2 we can see how by asserting the signal “ assert\_data” the values coming from RAM are outputted into the data out bus. Also by asserting “assert\_saved” the value previously stored that cam in the data in bus is outputted.

The testbench proved the component to work according to requirements.

## F.1 BUS MASTER TESTBENCH CODE

```
-----
-- Busmaster testbench
-- Author    : Guillermo Barreiro
-- Student ID : 1042071
-- Date      : March 24, 2002
-- File Name  : busmaster_test.vhd
-- Architecture : Structural
-----

library ieee;
use ieee.std_logic_1164.all;
library work;
use work.nn_pack.all;

-- textfixture
entity busmaster_test is
end busmaster_test;

architecture mixed of busmaster_test is
    signal reset, clock, done_wire, read_rand_wire, read_data_wire, incr_sel_wire, load_sel_wire: std_logic;
    signal assert_rand_wire, assert_data_wire, assert_saved_wire, load_addr_wire, do_nothing: std_logic;
    signal zero, one, notreset: std_logic;
    signal seed_ram, seed_rng : std_logic_vector ( 7 downto 0);
    signal stored_data_wires, dataout_wires, datain_wires, in_rng_wires, in_data_wires : std_logic_vector( 7 downto 0);
    signal addr_wires, sel_wires, in_sel_wires, in_addr_wires : std_logic_vector( 1 downto 0);

    constant T_halfclock : time := 25 ns;

begin

    -- clock generator
    clock_gen : process
    begin
        clock <= '0';
        wait for T_halfclock;
        clock <= '1';
        wait for T_halfclock;
    end process clock_gen;

    -----
    -- Instantiation of the NN datapath component --
    Component_busmaster : busmaster_2
    port map ( clock => clock,
        read_rand => read_rand_wire,
        read_data => read_data_wire,
        assert_rand => assert_rand_wire,
        assert_data => assert_data_wire,
        assert_saved => assert_saved_wire,
        load_addr => load_addr_wire,
        clear => reset,
        in_sel => in_sel_wires,
        incr_sel => incr_sel_wire,
        load_sel => load_sel_wire,
        datain => datain_wires,
        dataout => dataout_wires,
        addr => addr_wires,
        sel => sel_wires,
        in_rng => in_rng_wires,
        in_data => in_data_wires,
        in_addr => in_addr_wires,
        done => done_wire,
        stored_data => stored_data_wires,
        do_nothing => do_nothing);

    -----
    -- Instantiation of the PRNG
    seed_rng <= X"00";

    Random_numbers: LFSR_GENERIC generic map(Width => 8)
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
port map (clock => clock,
          reset => reset,
          load => one,
          enable => zero,
          parallel_in => seed_rng,
          parallel_out => in_rng_wires);

--          Values coming from RAM
seed_ram <= x"04";
datain_wires <= in_rng_wires;
zero <= '1';

values_RAM : LFSR_GENERIC generic map(Width => 8)
port map (clock => clock,
          reset => reset,
          load => one,
          enable => zero,
          parallel_in => seed_ram,
          parallel_out => in_data_wires);

-- process to initialize counters

in_sel_wires <= "00"; -- The trainer should start with first neuron
load_sel_wire <= '0';
in_addr_wires <= "00"; -- Trainer should start with first neuron
load_addr_wire <= '0';

ctrl_1 : process
begin
    -- pulse reset to avoid "don't cares" on input
    reset <= '0';
    wait for 50 ns;
    reset <= '1';
    wait for 50 ns;
    one <= '1';
    wait for 50 ns;
    one <= '0';
    wait for 50 ns;

    wait;
end process ctrl_1;

-- process to load random numbers in neurons
ctrl_2 : process
begin
    loop
        --Inputs random values into neurons
        read_rand_wire <= '1';
        wait for 100 ns;
        read_rand_wire <= '0';
        wait for 50 ns;
        assert_rand_wire <= '1';
        wait for 50 ns;
        assert_rand_wire <= '0';
        wait for 50 ns;
        incr_sel_wire <= '1';
        wait for 50 ns;
        incr_sel_wire <= '0';
        wait for 50 ns;
        exit when done_wire = '1';
    end loop;
    loop
        --Reads the values stored in RAM
        read_data_wire <= '1';
        wait for 100 ns;
        read_data_wire <= '0';
        wait for 50 ns;
        assert_data_wire <= '1';
        wait for 50 ns;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
        assert_data_wire <= '0';
        wait for 50 ns;
        incr_sel_wire <= '1';
        wait for 50 ns;
        incr_sel_wire <= '0';
        wait for 50 ns;
        exit when done_wire = '1';
    end loop;

    assert_saved_wire <= '1';
    wait for 50 ns;
    assert_saved_wire <= '0';
    wait for 50 ns;

    wait;
end process ctrl_2;

end mixed;
```

## F.2 Testbench for RAM

A 256-word ram is to be used in our project for storing previously loaded data. To ensure that the ram stores data correctly, a testbench is used to write values to the ram and read them out. The reason for a testbench is because it is impractical to exhaustively manually check if the ram stores 256 values correctly. What the testbench does is that it stores 256 words to separate addresses in the ram then reads them out in the same order. As it writes and reads in values, it the testbench stores these values to an associated file. Thus, to check if the ram is working, the output file of the read values should be exactly the same as the output file of the write values.

### F.2 RAM TESTBENCH CODE

```
-----
-- adder_pkg.vhd
--
-- Revised 2001/02/09
--

library ieee;
use ieee.std_logic_1164.all;
-- these packages allow math on std_logic
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

--LIBRARY lpm;
--USE lpm.lpm_components.ALL;

package adder_pkg is

constant data_width : positive := 16;

constant ram_data_width : positive := 16;          -- width of data in ram
constant ram_address_width : positive := 8;         -- width of address in ram
constant ram_data_size : positive := 256;          -- number of values in ram
subtype datapath is std_logic_vector(data_width-1 downto 0);

COMPONENT adder IS

    PORT( ram_data_in : in std_logic_vector(ram_data_width-1 downto 0);--ram_data_width - 1 downto 0);
        -- input data
            write : in std_logic;
            -- determines write
            read : in std_logic;
            -- determines read
            clock, reset : in std_logic;
            -- synchronized with system clock
            ram_data_out : out std_logic_vector(ram_data_width-1 downto 0);--ram_data_width - 1 downto 0)
-- output data
    r_add : out std_logic_vector(ram_address_width-1 downto 0);--ram_data_width - 1 downto 0) -- output data
    w_add : out std_logic_vector(ram_address_width-1 downto 0);
    add : out std_logic_vector(ram_address_width-1 downto 0)
        );
END COMPONENT adder;

end package adder_pkg;

-----
-- adder_test.vhd
--
-- Revised 2001/02/09
--
-- Authors: John Koob, Raymond Sung & Duncan Elliott
-- Date: Feb 2001
-- Course: EE552
-- Desc:
--
-- Testbench for adder.vhd. This file is to
-- be compiled using Mentor Graphics. The adder.vho
-- file should have been generated by MaxPlusII before
-- compiling this testbench.
--
-- This testbench serves as a template for testing
-- other designs that require pseudo-random input
-- data and signature analysis
--

library ieee;
```



## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
use ieee.std_logic_1164.all;

--LIBRARY lpm;
--USE lpm.lpm_components.ALL;

package test_pkg is

    -- this component is used to create the PRPG and Signature Compactor
    component LFSR_GENERIC is
        generic(Width: positive := 4);           -- length of pseudo-random sequence
        port (
            clock: in std_logic;
            reset: in std_logic;  -- active low reset
            load: in std_logic;   -- active high load (assert this to use as regular reg)
            enable: in std_logic; -- active high enable
            parallel_in: in std_logic_vector(Width-1 downto 0); -- parallel seed input
            parallel_out: out std_logic_vector(Width-1 downto 0); -- parallel data out
            serial_out: out std_logic -- serial data out (From last shift register)
        );
    end component LFSR_GENERIC;

end package test_pkg;

library ieee;
use ieee.std_logic_1164.all;
library work;
use work.test_pkg.all;
use work.adder_pkg.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;

-- textfixture
entity adder_test is
end adder_test;

architecture mixed of adder_test is

    constant T_halfclock : time := 100 ns;
    constant T_prop: time := 10 ns; -- avoid hold time violations

    signal a_internal, b_internal : datapath := (others => '0');
    signal a_internal_delayed, b_internal_delayed : datapath := (others => '0');
    signal seed_a, seed_b : datapath;
    --signal sum_internal_x : datapath;
    signal sum_internal_x : std_logic_vector(ram_data_width-1 downto 0);
    signal signature_x, signature_y : datapath := (others => '0');
    signal clock, reset : std_logic;
    signal load_prpg, load_compact : std_logic;
    signal serial_prpg_a, serial_prpg_b, serial_compact_x, serial_compact_y : std_logic;
    signal datain_request_internal, dataout_valid_internal : std_logic;
    signal dataout_request_internal, datain_valid_internal : std_logic;
    signal datain_valid_internal_delayed, datain_valid_internal_buf : std_logic;
    signal one : std_logic;
    signal zero : std_logic;

    signal ram_input : std_logic_vector(ram_data_width-1 downto 0);
    signal write_sig, read_sig, reading, writing : std_logic;
    signal write_address, read_address, address : std_logic_vector(ram_address_width -1 downto 0);

begin

    --insert delay elements - Do you know why this is necessary?
    a_internal_delayed <= a_internal after T_prop;
    b_internal_delayed <= b_internal after T_prop;
    datain_valid_internal_delayed <= datain_valid_internal_buf after T_prop;

    --initial seed value - DO NOT EDIT -
    seed_a <= X"AAAA";
    seed_b <= X"0014";
    one <= '1';
    zero <= '0';
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
-----
-- Your adder component port map here --
ram_input <= a_internal_delayed;
adder_part : component adder
    port map( ram_data_in => ram_input,
               write => write_sig,
               read => read_sig,
               clock => clock,

               reset => one,

               ram_data_out => sum_internal_x,

               r_add => read_address,
               w_add => write_address,
               add => address

    );
write_file:
process (clock) is -- write file_io.out (when done goes to '1')
    file my_output : TEXT open WRITE_MODE is "file_io.out";
    variable my_output_line : LINE;
    variable count : integer range 0 to 2 := 0;
begin
    if falling_edge(clock) then
        if write_sig = '1' and count = 2 then
            write(my_output_line, sum_internal_x); -- or any other stuff
            writeline(my_output, my_output_line);
            write(my_output_line, address); -- or any other stuff
            writeline(my_output, my_output_line);
            count := 0;

            writing <= '1';
        elsif write_sig = '1' and count < 2 then
            count := count + 1;
            writing <= '0';
        end if;
    end if;
end process write_file;

read_ram_file:
process (clock) is -- write file_io.out (when done goes to '1')
    file my_output : TEXT open WRITE_MODE is "file_read.out";
    variable my_output_line : LINE;
    variable count : integer range 0 to 2 := 1;
begin
    if falling_edge(clock) then
        if read_sig = '1' and count = 2 then
            write(my_output_line, sum_internal_x); -- or any other stuff
            writeline(my_output, my_output_line);
            write(my_output_line, address); -- or any other stuff
            writeline(my_output, my_output_line);
            count := 0;

            reading <= '1';
        elsif read_sig = '1' and count < 2 then
            count := count + 1;
            reading <= '0';
        end if;
    end if;
end process read_ram_file;

-----

datain_valid_input_ff: process (clock) is
begin
    if reset = '1' then -- if reset line low = active and clock edge received
        datain_valid_internal_buf <= '0';
    elsif clock = '1' and clock'event then -- if rising edge of clock
        if datain_request_internal = '1' then
            datain_valid_internal_buf <=
                datain_valid_internal; -- latch input and drive to outputs
        end if;
    end if;
end process;
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
end process datain_valid_input_ff;

-- instantiate a PRPG for input a
prpg_input_a : component LFSR_GENERIC
generic map(Width => data_width)
port map (clock => clock,
          reset => reset,
          load => load_prpg,
          enable => one, --datain_request_internal,
          parallel_in => seed_a,
          parallel_out => a_internal,
          serial_out => serial_prpg_a
        );

-- instantiate a PRPG for input b
prpg_input_b : component LFSR_GENERIC
generic map(Width => data_width)
port map (clock => clock,
          reset => reset,
          load => load_prpg,
          enable => one, --datain_request_internal,
          parallel_in => seed_b,
          parallel_out => b_internal,
          serial_out => serial_prpg_b
        );

-- instantiate a Signature compactor for output x
-- (output x can be used as the sum of an adder)
compactor_x : component LFSR_GENERIC
generic map(Width => data_width)
port map (clock => clock,
          reset => reset,
          load => load_compact,
          enable => dataout_valid_internal,
          parallel_in => sum_internal_x,
          parallel_out => signature_x,
          serial_out => serial_compact_x
        );

-- clock generator
clock_gen : process
begin
    clock <= '1';
    wait for T_halfclock;
    clock <= '0';
    wait for T_halfclock;
end process clock_gen;

-- process to control the load_compact and load_prpg
ctrl_1 : process
begin
    load_prpg <= '1';
    load_compact <= '1';

    -- pulse reset to avoid "don't cares" on input
    reset <= '0';
    reset <= '1';
    wait for 250 ns;
    reset <= '1';
    wait for 250 ns;
    reset <= '0';
    wait for 200 ns;

    -- start compacting
    load_compact <= '0';
    wait for 400 ns;

    -- start generating random data
```

## Hardware Implementation of a Neural Network Trainer and Associated Neural Network

---

```
        load_prpg <= '0';
        wait for 5000 ns;

        wait;
    end process ctrl_1;

    -- process to control handshaking signals
    ctrl_2 : process
    begin

        -- normal operation
        write_sig <= '0';
        read_sig <= '0';

        dataout_request_internal <= '1';
        datain_valid_internal <= '1';
        wait for 1300 ns;

        write_sig <= '1';
        wait for 153600 ns;
        write_sig <= '0';

        wait for 200 ns;

        read_sig <= '1';
        wait for 153600 ns;
        read_sig <= '0';

        wait;
    end process ctrl_2;

end mixed;
```