McGill University

ECSE 325 - Lab2

Fixed-point Representation and Modelsim Simulation

Group 2
Demo on Tuesday

Yi Zhu 260716006

Mai Zeng 260782174

1. Introduction

In this lab, the main goal is to learn the basics of fixed-point representations, VHDL testbench creation and functional verification using ModelSim. We will write VHDL code to represent a fixed-point multiply-accumulation unit which consists of multiplier, adder and register. We will also write a testbench code in ModelSIm to testify the correction of our fixed-point multiply-accumulation unit design.

2. VHDL Code

2.1 Multiply-Accumulation Unit

The VHDL code for the multiply-accumulation unit is shown in *Figure 2.1* below:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  use IEEE.std_logic_unsigned.ALL;
5
6  entity g02_MAC is
7  port(
8          x   : in std_logic_vector(9 downto 0); --input x
9          y   : in std_logic_vector(9 downto 0); --input y
10         N   : in std_logic_vector(9 downto 0); --total number of inputs
11         clk : in std_logic; --clock
12         rst : in std_logic; --asynchronous reset
13         mac : out std_logic_vector(20 downto 0); --output of MAC unit
14         ready   : out std_logic); --denotes the validity of mac signal
15 end g02_MAC;
16
17 architecture MAC of g02_MAC is
18     signal  macsum      : std_logic_vector(20 downto 0);
19     signal  counter     : std_logic_vector(9 downto 0);
20     signal ready_temp   : std_logic;
21     begin
22         process(clk)
23         begin
24             if rising_edge(clk) then -- After reset all bits for counter and macsum should set to 0
25                 if rst = '1' then -- asyn reset should not be shown in sensitivity list
26                     counter <= (others => '0'); -- all bits set to 0
27                     macsum <= (others => '0'); -- all bits set to 0
28                     ready_temp  <= '0'; -- set ready to 0 again
29                 else
30                     macsum <= std_logic_vector(signed(macsum) + signed(x)*signed(y));
31                     counter <= counter + 1; -- after macsum operation, temporary counter add by 1
32                     if (counter = N) then -- whenever temporary counter reaches N (input from testbench) should write to txt file
33                         ready_temp <= '1';
34                     end if;
35                 end if;
36             end if;
37         end process;
38         mac <= macsum; -- output the mac
39         ready <= ready_temp; -- output ready (ready_out in testbench)
40
41     end MAC;
```

Figure 2.1-1 VHDL Code for Multiply-Accumulation Unit

In this code we will use IEEE as our main library and three sub-libraries, which are STD_LOGIC_1164, NUMERIC_STD and STD_LOGIC_UNSIGNED. The STD_LOGIC_UNSIGNED is used in this lab since we will deal with the signed number in the multiplication process.

In the entity declaration section, we have our inputs to be x, y, N, clk and rst while our outputs to be mac and ready. For x and y, they are 10 bits since each two's complement numbers of our output files will have 10 bits. N, which is considered to be the total number of inputs also has 10 bits in length since we have a total of 1000 numbers. Since we will implement a time-sensitive multiply-accumulation unit, we

also have clk as input. Thus, the multiplication will only be processed at the rising edge of the clock signal. Besides on, we will also have rst as our input since we will include the reset function. For outputs, we have mac and ready. For the former one, it is the 21-bit-output of the multiply-accumulation unit. The reason for 21-bit-length is that since we are multiplying and accumulating two 10-bit-numbers, the result will be in 20 bits while we still need an additional bit for the sign. The latter one, ready, is a signal to demonstrate whether the computation is completed.

In the architecture section, we will implement the multiply-accumulation unit logic as shown in the below:

```
mac = 0; ready = 0;
for i = 1:N
        mac = mac + x(i) * y(i);
end
ready = 1;
```

Figure 2.1-2 Multiply-Accumulation Unit Logic Code

We added three signals in the architecture: macsum, counter and ready_temp. The signal macsum and ready_temp are two buffers that temporarily store the 21-bit-output value of the multiply-accumulation unit and 1-bit-signal of ready respectively. The counter, as its names indicated, is used to count the number of multiplication and accumulation. It has 10 bits in length since we will have 1000 numbers to calculate in total.

Since the multiply-accumulation unit is time-sensitive that the output will be changed only at the rising edge, we will have a process block with clk in the sensitivity list. For the reset function, we have one if statement that when the rst signal is high, we will set the counter, output and ready signals all to be ‘0’. Otherwise, we will perform the multiplication and addition calculation as the logic illustrated above. After so, the counter will count up by one for each calculation until it reaches 1000. At that point, it will set the ready signal to be ‘1’, which means the summation is ready to output. Finally, we will output the macsum and ready_temp signal to the output variables mac and ready.

The complication results and flow summary are shown in *Figure 2.1-3* and *Figure 2.1-4* below:

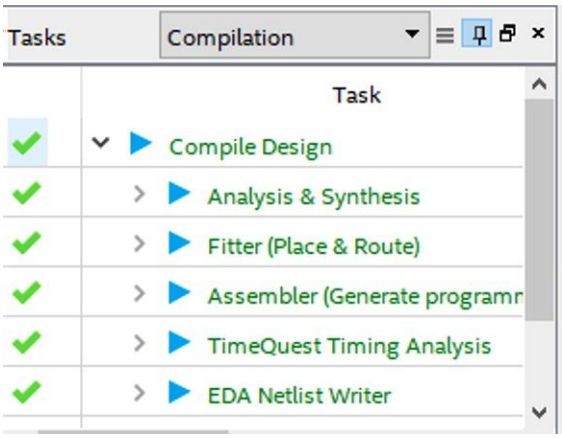

Figure 2.1-3 Complication Result

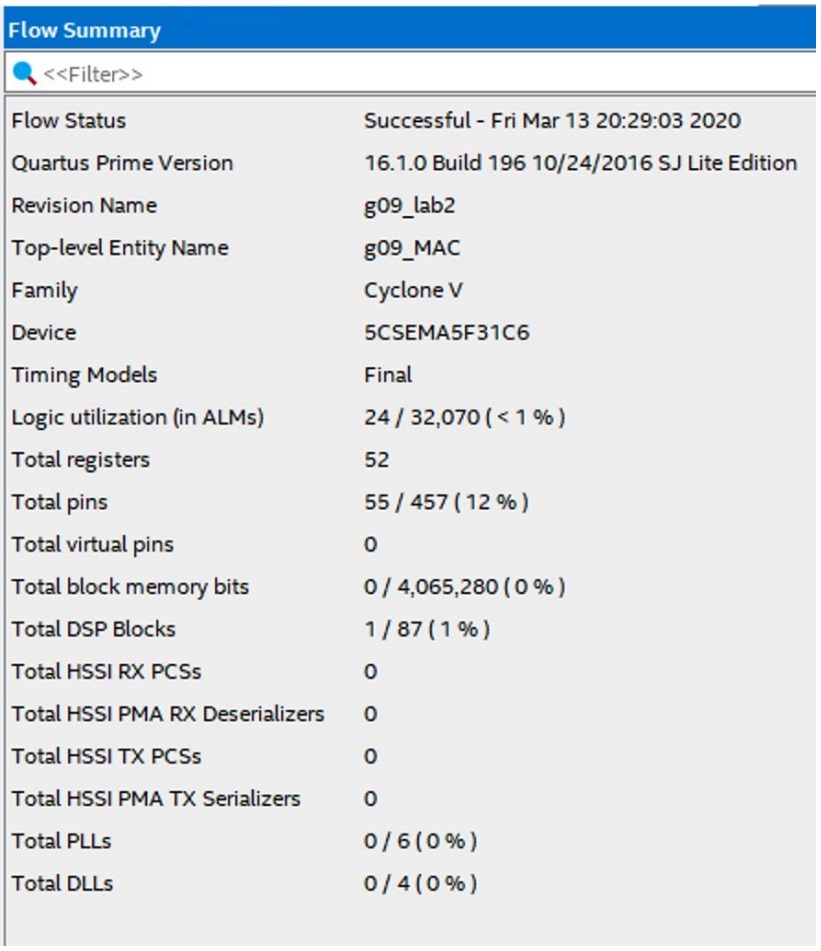

Figure 2.1-4 Flow Summary

From the flow summary we can see that the logic utilization (in ALMs) is 24/32070, which is less than 1%. There are 52 registers used for our 10-bit-multiply-accumulation unit.

The RTL viewer of our design in shown in *Figure 2.1-5* below:

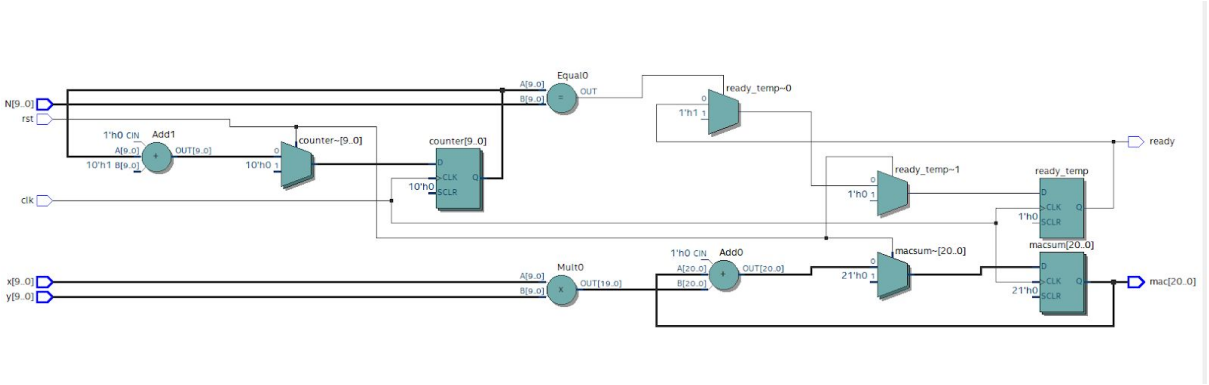

Figure 2.1-5 RTL Viewer

The RTL viewer shows that there are four 2:1 multiplexers, three D flip-op, two adders, one multiplier and one equal operator used. In the left of the figure, one adder, one multiplexer and one register are used for the counter. The equal operator is used for comparing the counter and number of inputs N. The multiplier and adders at the bottom are used for our multiply-accumulation unit. Finally, two registers in the right are used for storing the outputs.

2.2 Testbench

The VHDL code for our testbench is shown in *Figure 2.2-1* and *Figure 2.2-2* below:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use STD.textio.all;
use ieee.std_logic_textio.all;
use IEEE.std_logic_unsigned.ALL;

entity g02_MAC_tb is
end g02_MAC_tb;

architecture test of g02_MAC_tb is

------------------------------------------------------------
-- Declare the Component Under Test
------------------------------------------------------------

component g02_MAC is
  port (
      x     : in std_logic_vector(9 downto 0);
    y     : in std_logic_vector(9 downto 0);
    N     : in std_logic_vector(9 downto 0);
    clk   : in std_logic;
    rst   : in std_logic;
    mac   : out std_logic_vector (20 downto 0);
    ready : out std_logic);
end component g02_MAC;

------------------------------------------------------------
-- Testbench Internal Signals
------------------------------------------------------------

file file_VECTORS_X : text;
file file_VECTORS_Y : text;
file file_RESULTS   : text;

constant clk_PERIOD : time := 100 ns;

signal x_in       :  std_logic_vector(9 downto 0);
signal y_in       :  std_logic_vector(9 downto 0);
signal N_in       :  std_logic_vector(9 downto 0);
signal clk_in     :  std_logic;
signal rst_in     :  std_logic;
signal mac_out    :  std_logic_vector (20 downto 0);
signal ready_out  :  std_logic;

begin -- Instantiate MAC
 g02_MAC_INST : g02_MAC
  port map (
    x => x_in,
    y => y_in,
    N => N_in,
    clk => clk_in,
    rst => rst_in,
    mac => mac_out,
    ready => ready_out
  );

```

Figure 2.2-1 VHDL Code for Testbench

```

------------------------------------------------------------
-- Clock Generation
------------------------------------------------------------

clk_generation : process
begin
    clk_in <= '1';
    wait for clk_PERIOD / 2;
    clk_in <= '0';
    wait for clk_PERIOD / 2;
end process clk_generation;

------------------------------------------------------------
-- Providing Inputs
------------------------------------------------------------

feeding_instr : process is
  variable v_Iline1 : line;
  variable v_Iline2 : line;
  variable v_Oline  : line;
  variable v_x_in   : std_logic_vector(9 downto 0);
  variable v_y_in   : std_logic_vector(9 downto 0);
 begin
  --reset the circuit
  N_in <= "1111101000"; -- N = 1000
  rst_in <= '1';
  wait until rising_edge(clk_in);
  wait until rising_edge(clk_in);
  rst_in <= '0';
  file_open(file_VECTORS_X, "lab2-x-fixed-point.txt", read_mode);
  file_open(file_VECTORS_Y, "lab2-y-fixed-point.txt", read_mode);
  file_open(file_RESULTS, "lab2-out.txt", write_mode);

  while not endfile(file_VECTORS_X) loop
   readline(file_VECTORS_X, v_Iline1);
   read(v_Iline1, v_x_in);
   readline(file_VECTORS_Y, v_Iline2);
   read(v_Iline2, v_y_in);

   x_in <= v_x_in;
   y_in <= v_y_in;

   wait until rising_edge(clk_in);
   end loop;
  wait until rising_edge(clk_in);
  wait until rising_edge(clk_in);
  if ready_out = '1' then

   write(v_Oline, mac_out);
   writeline(file_RESULTS, v_Oline);
   --wait until rising_edge(clk_in);
  end if;
   wait;
end process;
end architecture test;
```

Figure 2.2-2 VHDL Code for Testbench

A testbench is a special VHDL entity that generates inputs applied to our circuit, to automate the simulation of your circuit and compare the outputs to respond to different inputs. In this VHDL code, we followed the step-by-step instructions in the lab manual. To validate our multiply-accumulation unit design, we need to include the libraries containing various data types in our testbench code. After that, to verify the component of our multiply-accumulation unit, we will declare our design and wire it to the testbench by instantiating it to realize the I/O mapping. Port mapping is used to enable the interaction. A clock signal is also created in order to synchronize stimulus signals inside the testbench. Finally, the multiply-accumulation unit is instantiated and wired into the testbench. Then we can start reading test vectors into the circuits and simulating. Once the ready signal goes high, the output is written into an output file.

3. Simulation Verification

The wave simulation result of our multiply-accumulation unit in ModelSim is shown in *Figure 3.1* below:

Figure 3.1 Wave Simulation Result in ModelSim

The result is shown in 2's complement binary is 01111111110101001011 00.

4. Exercises

For the exercises, we are asked to split the 1000 input data into 5 batches, each containing 200 values. Since we split the inputs the precision of each input file may vary. This means that we need to customize the number of bits for input of x, input of y and the output. Also, in the testbench we have to change the number of input from 1000 to 200.

We rewrite our python scripts and get the precisions for each patch and we got that for the X and Y the precisions for W are 2,3,2,2,2 (include the sign) and for the precisions of F are 3,5,6,4,7. It makes sense because what we got the precision for the whole input W and F are 3 and 7 add them up is 10 (1 bit for the sign) and right now what we got the max precision for X is 3 and max precision for Y is 7.

Take the first batch for example to show what we should change the number of bits for x input y input and output and also the number of input.*Figure 4.1* shows that we change N_in from 1000 to 200 and now the binary number becomes 11001000. Also, we change the v_x_in and v_y_in to 5 bits. Because for batch 0 the W precision is 2 (include the bit for the sign) and F the precision is 3 and together they should be 5 bits.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use STD.textio.all;
use ieee.std_logic_textio.all;
use IEEE.std_logic_unsigned.ALL;

entity g02_MAC_tb is
end g02_MAC_tb;

architecture test of g02_MAC_tb is

--------------------------------------------------------------------------------
-- Declare the Component Under Test
--------------------------------------------------------------------------------

component g02_MAC is
  port (
       x     : in std_logic_vector(4 downto 0);
    y     : in std_logic_vector(4 downto 0);
    N     : in std_logic_vector(7 downto 0);
    clk   : in std_logic;
    rst   : in std_logic;
    mac   : out std_logic_vector (9 downto 0);
    ready : out std_logic);
end component g02_MAC;

--------------------------------------------------------------------------------
-- Testbench Internal Signals
--------------------------------------------------------------------------------

file file_VECTORS_X : text;
file file_VECTORS_Y : text;
file file_RESULTS   : text;

constant clk_PERIOD : time := 100 ns;

signal x_in       :  std_logic_vector(4 downto 0);
signal y_in       :  std_logic_vector(4 downto 0);
signal N_in       :  std_logic_vector(7 downto 0);
signal clk_in     :  std_logic;
signal rst_in     :  std_logic;
signal mac_out    :  std_logic_vector (10 downto 0);
signal ready_out  :  std_logic;

begin -- Instantiate MAC
 g02_MAC_INST : g02_MAC
  port map (
    x => x_in,
    y => y_in,
    N => N_in,
    clk => clk_in,
    rst => rst_in,
    mac => mac_out,
    ready => ready_out
  );
```

Figure 4.1 Change for Testbench for batch 0

With the same idea, we have to change the bit number component *Figure 4.2* shows that

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_unsigned.ALL;

entity g02_MAC is
port(
        x   : in std_logic_vector(4 downto 0); --input x
        y   : in std_logic_vector(4 downto 0); --input y
        N   : in std_logic_vector(7 downto 0); --total number of inputs
        clk : in std_logic; --clock
        rst : in std_logic; --asynchronous reset
        mac : out std_logic_vector(9 downto 0); --output of MAC unit
        ready   : out std_logic); --denotes the validity of mac signal
end g02_MAC;

architecture MAC of g02_MAC is
    signal  macsum      : std_logic_vector(9 downto 0);
    signal  counter     : std_logic_vector(9 downto 0);
    signal ready_temp   : std_logic;
    begin
        process(clk)
        begin
            if rising_edge(clk) then -- After reset all bits for counter and macsum should set to 0
                if rst = '1' then -- asyn reset should not be shown in sensitivity list
                    counter <= (others => '0'); -- all bits set to 0
                    macsum <= (others => '0'); -- all bits set to 0
                    ready_temp  <= '0'; -- set ready to 0 again
                else
                    macsum <= std_logic_vector(signed(macsum) + signed(x)*signed(y));
                    counter <= counter + 1; -- after macsum operation, temporary counter add by 1
                    if (counter = N) then -- whenever temporary counter reaches N (input from testbench) should write to txt file
                        ready_temp <= '1';
                    end if;
                end if;
            end if;
        end process;
        mac <= macsum; -- output the mac
        ready <= ready_temp; -- output ready (ready_out in testbench)

    end MAC;
```

Figure 4.2 Change for VHDL code for batch 0

And *Figure 4.3* shows that the result and the result is in 10 bits because we get 5 bits for each input.

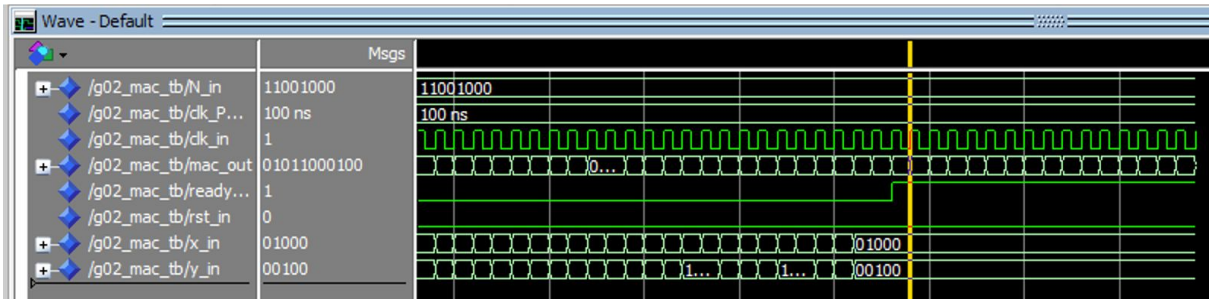

Figure 4.3 Result for Batch 0

The table and figures below are the result for the exercises.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_unsigned.ALL;

entity g02_MAC is
port(
        x   : in std_logic_vector(7 downto 0); --input x
        y   : in std_logic_vector(7 downto 0); --input y
        N   : in std_logic_vector(7 downto 0); --total number of inputs
        clk : in std_logic; --clock
        rst : in std_logic; --asynchronous reset
        mac : out std_logic_vector(16 downto 0); --output of MAC unit
        ready   : out std_logic); --denotes the validity of mac signal
end g02_MAC;

architecture MAC of g02_MAC is
    signal  macsum      : std_logic_vector(16 downto 0);
    signal  counter     : std_logic_vector(16 downto 0);
    signal ready_temp   : std_logic;
    begin
        process(clk)
        begin
            if rising_edge(clk) then -- After reset all bits for counter and macsum should set to 0
                if rst = '1' then -- asyn reset should not be shown in sensitivity list
                    counter <= (others => '0'); -- all bits set to 0
                    macsum <= (others => '0'); -- all bits set to 0
                    ready_temp  <= '0'; -- set ready to 0 again
                else
                    macsum <= std_logic_vector(signed(macsum) + signed(x)*signed(y));
                    counter <= counter + 1; -- after macsum operation, temporary counter add by 1
                    if (counter = N) then -- whenever temporary counter reaches N (input from testbench) should write to txt file
                        ready_temp <= '1';
                    end if;
                end if;
            end if;
        end process;
        mac <= macsum; -- output the mac
        ready <= ready_temp; -- output ready (ready_out in testbench)

    end MAC;
```

Figure 4.4 Change for VHDL code batch 1

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4   use STD.textio.all;
5   use ieee.std_logic_textio.all;
6   use IEEE.std_logic_unsigned.ALL;
7
8   entity g02_MAC_tb is
9   end g02_MAC_tb;
10
11  architecture test of g02_MAC_tb is
12
13  ------------------------------------------------------------------------------
14  -- Declare the Component Under Test
15  ------------------------------------------------------------------------------
16
17  component g02_MAC is
18    port (
19          x     : in std_logic_vector(7 downto 0);
20      y     : in std_logic_vector(7 downto 0);
21      N     : in std_logic_vector(7 downto 0);
22      clk   : in std_logic;
23      rst   : in std_logic;
24      mac   : out std_logic_vector (16 downto 0);
25      ready : out std_logic);
26  end component g02_MAC;
27
28  ------------------------------------------------------------------------------
29  -- Testbench Internal Signals
30  ------------------------------------------------------------------------------
31
32  file file_VECTORS_X : text;
33  file file_VECTORS_Y : text;
34  file file_RESULTS   : text;
35
36  constant clk_PERIOD : time := 100 ns;
37
38  signal x_in       :  std_logic_vector(7 downto 0);
39  signal y_in       :  std_logic_vector(7 downto 0);
40  signal N_in       :  std_logic_vector(7 downto 0);
41  signal clk_in     :  std_logic;
42  signal rst_in     :  std_logic;
43  signal mac_out    :  std_logic_vector (16 downto 0);
44  signal ready_out  :  std_logic;
45
46  begin -- Instantiate MAC
47   g02_MAC_INST : g02_MAC
48    port map (
49      x => x_in,
50      y => y_in,
51      N => N_in,
52      clk => clk_in,
53      rst => rst_in,
54      mac => mac_out,
55      ready => ready_out
56    );
57
```

Figure 4.5 Change for testbench batch 1

Figure 4.6 Result for batch 1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_unsigned.ALL;

entity g02_MAC is
port(
        x   : in std_logic_vector(7 downto 0); --input x
        y   : in std_logic_vector(7 downto 0); --input y
        N   : in std_logic_vector(7 downto 0); --total number of inputs
        clk : in std_logic; --clock
        rst : in std_logic; --asynchronous reset
        mac : out std_logic_vector(16 downto 0); --output of MAC unit
        ready   : out std_logic); --denotes the validity of mac signal
end g02_MAC;

architecture MAC of g02_MAC is
    signal  macsum      : std_logic_vector(16 downto 0);
    signal  counter     : std_logic_vector(16 downto 0);
    signal ready_temp   : std_logic;
    begin
        process(clk)
        begin
            if rising_edge(clk) then -- After reset all bits for counter and macsum should set to 0
                if rst = '1' then -- asyn reset should not be shown in sensitivity list
                    counter <= (others => '0'); -- all bits set to 0
                    macsum <= (others => '0'); -- all bits set to 0
                    ready_temp  <= '0'; -- set ready to 0 again
                else
                    macsum <= std_logic_vector(signed(macsum) + signed(x)*signed(y));
                    counter <= counter + 1; -- after macsum operation, temporary counter add by 1
                    if (counter = N) then -- whenever temporary counter reaches N (input from testbench) should write to txt file
                        ready_temp <= '1';
                    end if;
                end if;
            end if;
        end process;
        mac <= macsum; -- output the mac
        ready <= ready_temp; -- output ready (ready_out in testbench)

    end MAC;
```

Figure 4.7 VHDL code for batch 2

```
15  ----------------------------------------------------------------------------
16
17  component g02_MAC is
18    port (
19          x     : in std_logic_vector(7 downto 0);
20        y     : in std_logic_vector(7 downto 0);
21        N     : in std_logic_vector(7 downto 0);
22        clk   : in std_logic;
23        rst   : in std_logic;
24        mac   : out std_logic_vector (16 downto 0);
25        ready : out std_logic);
26  end component g02_MAC;
27
28  ----------------------------------------------------------------------------
29  -- Testbench Internal Signals
30  ----------------------------------------------------------------------------
31
32  file file_VECTORS_X : text;
33  file file_VECTORS_Y : text;
34  file file_RESULTS   : text;
35
36  constant clk_PERIOD : time := 100 ns;
37
38  signal x_in       :  std_logic_vector(7 downto 0);
39  signal y_in       :  std_logic_vector(7 downto 0);
40  signal N_in       :  std_logic_vector(7 downto 0);
41  signal clk_in     :  std_logic;
42  signal rst_in     :  std_logic;
43  signal mac_out    :  std_logic_vector (16 downto 0);
44  signal ready_out  :  std_logic;
45
46  begin -- Instantiate MAC
47   g02_MAC_INST : g02_MAC
48    port map (
49      x => x_in,
50      y => y_in,
51      N => N_in,
52      clk => clk_in,
53      rst => rst_in,
54      mac => mac_out,
55      ready => ready_out
56    );
57
58  ----------------------------------------------------------------------------
59  -- Clock Generation
60  ----------------------------------------------------------------------------
61
62  clk_generation : process
63  begin
64      clk_in <= '1';
65      wait for clk_PERIOD / 2;
66      clk_in <= '0';
67      wait for clk_PERIOD / 2;
68  end process clk_generation;
69
70  ----------------------------------------------------------------------------
71  -- Providing Inputs
```

Figure 4.8 Change for testbench batch 2

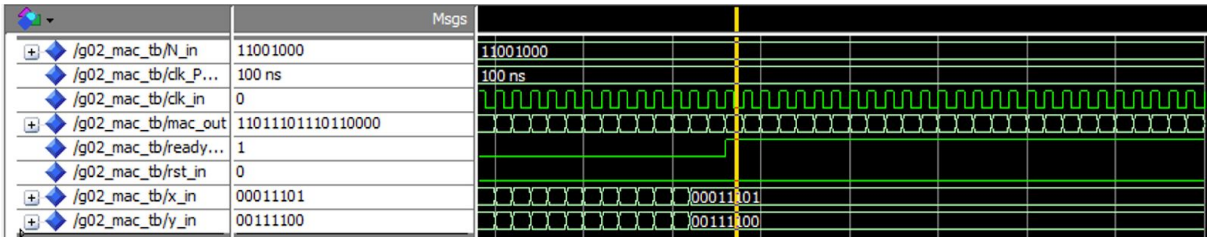

Figure 4.9 Result for batch 2

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_unsigned.ALL;

entity g02_MAC is
port(
        x   : in std_logic_vector(5 downto 0); --input x
        y   : in std_logic_vector(5 downto 0); --input y
        N   : in std_logic_vector(7 downto 0); --total number of inputs
        clk : in std_logic; --clock
        rst : in std_logic; --asynchronous reset
        mac : out std_logic_vector(12 downto 0); --output of MAC unit
        ready   : out std_logic); --denotes the validity of mac signal
end g02_MAC;

architecture MAC of g02_MAC is
    signal  macsum      : std_logic_vector(12 downto 0);
    signal  counter     : std_logic_vector(12 downto 0);
    signal ready_temp   : std_logic;
    begin
        process(clk)
        begin
            if rising_edge(clk) then -- After reset all bits for counter and macsum should set to 0
                if rst = '1' then -- asyn reset should not be shown in sensitivity list
                    counter <= (others => '0'); -- all bits set to 0
                    macsum <= (others => '0'); -- all bits set to 0
                    ready_temp  <= '0'; -- set ready to 0 again
                else
                    macsum <= std_logic_vector(signed(macsum) + signed(x)*signed(y));
                    counter <= counter + 1; -- after macsum operation, temporary counter add by 1
                    if (counter = N) then -- whenever temporary counter reaches N (input from testbench) should write to txt file
                        ready_temp <= '1';
                    end if;
                end if;
            end if;
        end process;
        mac <= macsum; -- output the mac
        ready <= ready_temp; -- output ready (ready_out in testbench)

    end MAC;
```

Figure 4.10 VHDL code for batch 3

```
15  ----------------------------------------------------------------------------
16
17  component g02_MAC is
18    port (
19          x     : in std_logic_vector(5 downto 0);
20        y     : in std_logic_vector(5 downto 0);
21        N     : in std_logic_vector(7 downto 0);
22        clk   : in std_logic;
23        rst   : in std_logic;
24        mac   : out std_logic_vector (12 downto 0);
25        ready : out std_logic);
26  end component g02_MAC;
27
28  ----------------------------------------------------------------------------
29  -- Testbench Internal Signals
30  ----------------------------------------------------------------------------
31
32  file file_VECTORS_X : text;
33  file file_VECTORS_Y : text;
34  file file_RESULTS   : text;
35
36  constant clk_PERIOD : time := 100 ns;
37
38  signal x_in       :  std_logic_vector(5 downto 0);
39  signal y_in       :  std_logic_vector(5 downto 0);
40  signal N_in       :  std_logic_vector(7 downto 0);
41  signal clk_in     :  std_logic;
42  signal rst_in     :  std_logic;
43  signal mac_out    :  std_logic_vector (12 downto 0);
44  signal ready_out  :  std_logic;
45
46  begin -- Instantiate MAC
47   g02_MAC_INST : g02_MAC
48    port map (
49      x => x_in,
50      y => y_in,
51      N => N_in,
52      clk => clk_in,
53      rst => rst_in,
54      mac => mac_out,
55      ready => ready_out
56    );
57
58  ----------------------------------------------------------------------------
59  -- Clock Generation
60  ----------------------------------------------------------------------------
61
62  clk_generation : process
63  begin
64      clk_in <= '1';
65      wait for clk_PERIOD / 2;
66      clk_in <= '0';
67      wait for clk_PERIOD / 2;
68  end process clk_generation;
69
70  ----------------------------------------------------------------------------
71  -- Providing Inputs
```

Figure 4.11 Testbench for batch 3

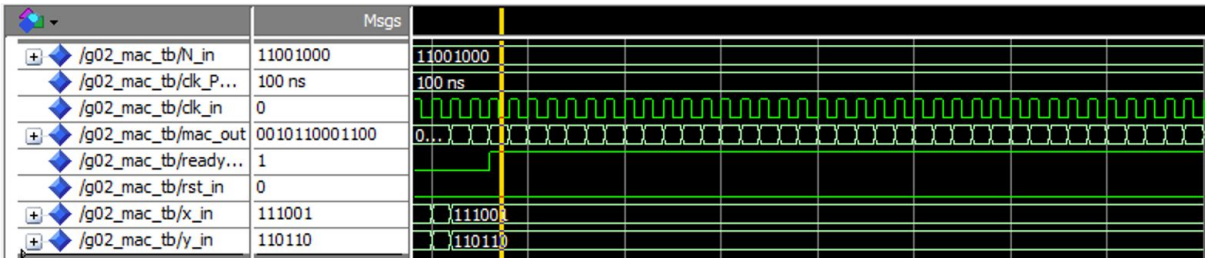

Figure 4.12 Result for batch 3

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_unsigned.ALL;

entity g02_MAC is
port(
        x   : in std_logic_vector(8 downto 0); --input x
        y   : in std_logic_vector(8 downto 0); --input y
        N   : in std_logic_vector(7 downto 0); --total number of inputs
        clk : in std_logic; --clock
        rst : in std_logic; --asynchronous reset
        mac : out std_logic_vector(18 downto 0); --output of MAC unit
        ready   : out std_logic); --denotes the validity of mac signal
end g02_MAC;

architecture MAC of g02_MAC is
    signal  macsum      : std_logic_vector(18 downto 0);
    signal  counter     : std_logic_vector(18 downto 0);
    signal ready_temp   : std_logic;
    begin
        process(clk)
        begin
            if rising_edge(clk) then -- After reset all bits for counter and macsum should set to 0
                if rst = '1' then -- asyn reset should not be shown in sensitivity list
                    counter <= (others => '0'); -- all bits set to 0
                    macsum <= (others => '0'); -- all bits set to 0
                    ready_temp  <= '0'; -- set ready to 0 again
                else
                    macsum <= std_logic_vector(signed(macsum) + signed(x)*signed(y));
                    counter <= counter + 1; -- after macsum operation, temporary counter add by 1
                    if (counter = N) then -- whenever temporary counter reaches N (input from testbench) should write to txt file
                        ready_temp <= '1';
                    end if;
                end if;
            end if;
        end process;
        mac <= macsum; -- output the mac
        ready <= ready_temp; -- output ready (ready_out in testbench)

    end MAC;
```

Figure 4.13 VHDL code for batch 4

```
15  --------------------------------------------------------------------------------
16
17  component g02_MAC is
18    port (
19          x     : in std_logic_vector(8 downto 0);
20        y     : in std_logic_vector(8 downto 0);
21        N     : in std_logic_vector(7 downto 0);
22        clk   : in std_logic;
23        rst   : in std_logic;
24        mac   : out std_logic_vector (18 downto 0);
25        ready : out std_logic);
26  end component g02_MAC;
27
28  --------------------------------------------------------------------------------
29  -- Testbench Internal Signals
30  --------------------------------------------------------------------------------
31
32  file file_VECTORS_X : text;
33  file file_VECTORS_Y : text;
34  file file_RESULTS   : text;
35
36  constant clk_PERIOD : time := 100 ns;
37
38  signal x_in       :  std_logic_vector(8 downto 0);
39  signal y_in       :  std_logic_vector(8 downto 0);
40  signal N_in       :  std_logic_vector(7 downto 0);
41  signal clk_in     :  std_logic;
42  signal rst_in     :  std_logic;
43  signal mac_out    :  std_logic_vector (18 downto 0);
44  signal ready_out  :  std_logic;
45
46  begin -- Instantiate MAC
47   g02_MAC_INST : g02_MAC
48    port map (
49      x => x_in,
50      y => y_in,
51      N => N_in,
52      clk => clk_in,
53      rst => rst_in,
54      mac => mac_out,
55      ready => ready_out
56    );
57
58  --------------------------------------------------------------------------------
59  -- Clock Generation
60  --------------------------------------------------------------------------------
61
62  clk_generation : process
63  begin
64      clk_in <= '1';
65      wait for clk_PERIOD / 2;
66      clk_in <= '0';
67      wait for clk_PERIOD / 2;
68  end process clk_generation;
69
70  --------------------------------------------------------------------------------
71  -- Providing Inputs
```

Figure 4.14 Testbench for Batch 4

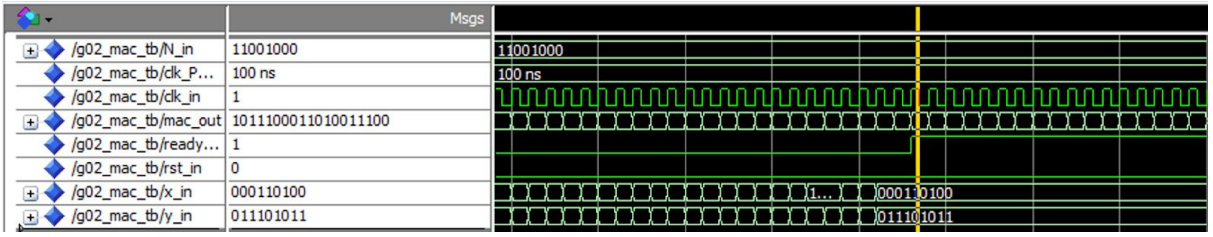

Figure 4.15 Result for batch 4

Batch number	*W precision (include sign bit)*	*F precision*
0	*2*	*3*
1	*3*	*5*
2	*2*	*6*
3	*2*	*4*
4	*2*	*7*

5. Conclusion

In this laboratory, we learned basic knowledge of fixed-point representations, VHDL testbench creation and functional verification by using the ModelSim. The simulation output result complies with our estimation and satisfies the design requirements. Thus, we can conclude that our design of the multiply-accumulation unit is successful. This lab is also a useful practice for VHDL design validation and simulation.