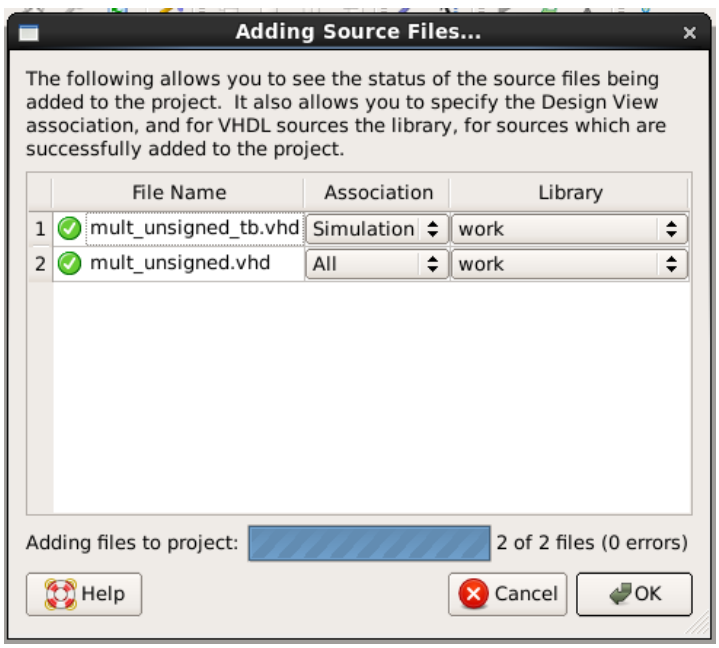
Computer Logic – Practical 3

Objective:

To investigate unsigned and signed multipliers.

Tasks:

1) A new project called *"Project3"* was created and the provided files *"mult_unsigned.vhd"* and *"mult_unsigned_tb.vhd"* were added to it from *Project: Add Source*. The association was set to *All* for the *mult_unsigned.vhd* file and to *Simulation* for the *mult_unsigned_tb.vhd* file.



2) The file *mult_unsigned.vhd* was then opened. This included a *shift_mux* process which contained for loops and also if statements. However, this process was not complete and so was modified as follows:

a) The contents of row corresponding to Algorithm 1, Line 3 were generalized to be true for all iterations by altering the code as shown.

```
-- Task 2a: Algorithm 1, Line 3
row := (others => '0');          -- set row to 0
row(n-1+i downto i) := a;        -- set n bits to a
```

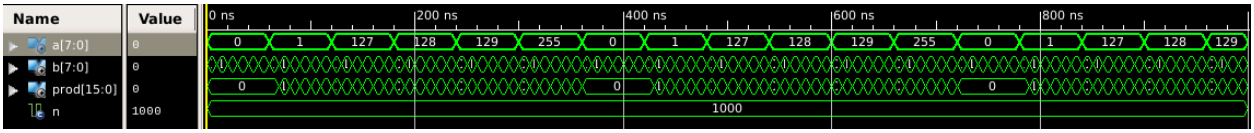
b) Algorithm 1, Line 4 was modified to add the value of the row to the total in the if statement as can be seen.

```
-- Task 2b; Algorithm 1, Line 4
if b(i) = '1' then
```

3) The test bench *mult_unsigned_tb.vhd* was then opened and modified with an additional 4 hexadecimal values so that there are 6 in all.

```
-- stimulus process
stim_proc : process
    type arr is array(0 to 5) of std_logic_vector(7 downto 0);
    constant values : arr := (
        x"00",                      -- hex 00
        x"01",                      -- hex 01
        x"7F",
        x"80",
        x"81",
        x"FF"
        );
```

After this was done, the test bench was simulated using *Behavioral Simulation* to test the multiplier as can be seen in the next snippet. The Radix was changed to *Unsigned Decimal* since the multiplication was unsigned and all 36 possible products were verified to give the expected output for correctness sake.



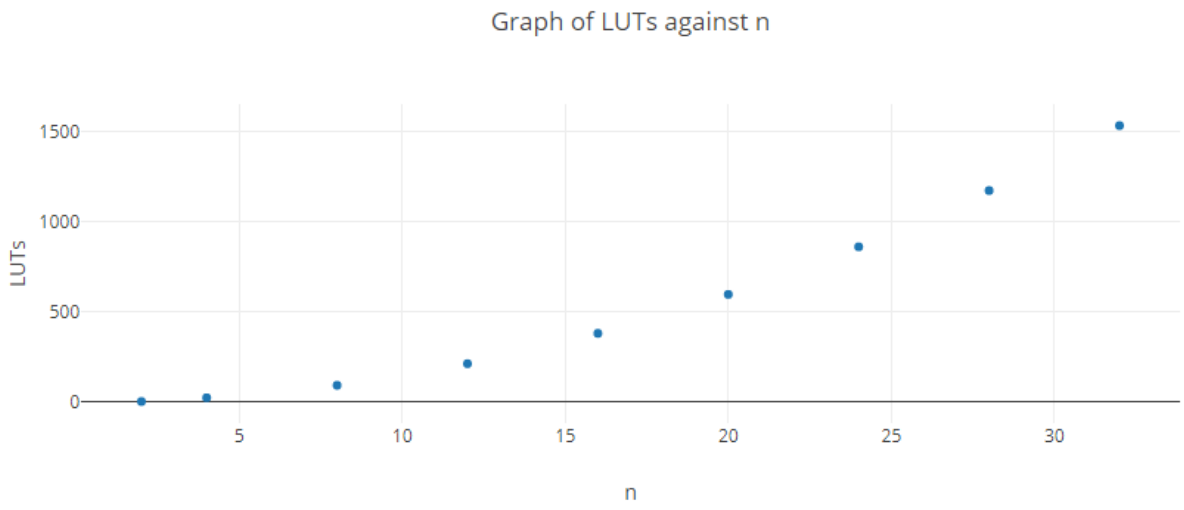
4) The hardware area of the multiplier was measured next. This was done automatically by the program when double clicking the *Synthesize* option found in the *Implementation view*, on the *Processes* pane when the multiplier is selected. The hardware footprint of the design was then found from the *Design Summary*, in the *Device Utilization Summary* when reading the *Number of Slice LUTs*. For this particular multiplier, the hardware area was found to be 94.

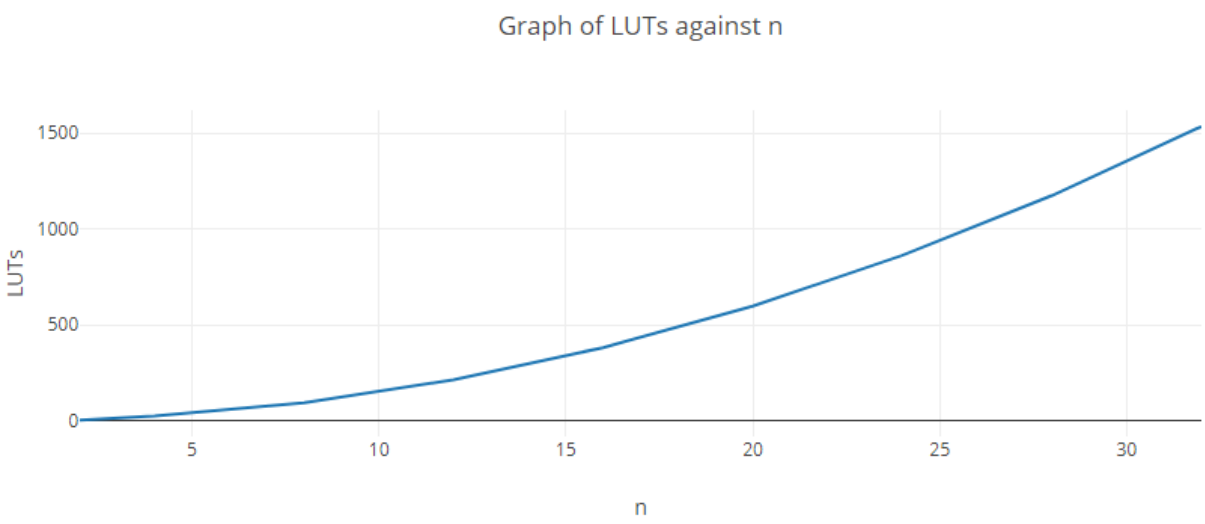
Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	94	27288	0%
Number of fully used LUT-FF pairs	0	94	0%
Number of bonded IOBs	32	296	10%

5) In this next step, *n* was changed in the range of $2 \leq n \leq 32$ inside the generic part of the entity and step 4 was repeated each time to obtain a measure of the hardware area for these different values. The results are listed on the table below

n	LUTs
2	4
4	25
8	94
12	214
16	382
20	598
24	862
28	1174
32	1534

6) A graph of LUTs (y-axis) against n (x-axis) was plotted and this graph seemed to relate mostly to a quadratic curve. The reason why is simple. The larger the value of *n*, the bigger the hardware area required so it would not make sense for such a graph to represent a square curve or a cubic one implying that for certain small values of *n*, the hardware area required is larger than for other values of *n* in which *n* is larger than before. The 2 snippets below represent this graph distribution, one with the plotted points and the other with the points connected.





7) The whole *mult_unsigned.vhd* file was copied to a new project file entitled *mult_signed.vhd*. In this file any appearances of *mult_unsigned* were changed to *mult_signed*. The *shift_mux* process was also altered to perform signed multiplication instead of unsigned as explained in the following 3 steps:
   a) All appearances of *mult_unsigned* were changed to *mult_signed*.
   b) *a* was sign-extended when setting the row variable this time. This was done by adding an extra line as shown together with the 2 previously written lines.

```
row := (others => '0');          -- set row to 0
row(n-1+i downto i) := a;        -- set n bits to a
row(2*n-1 downto n+i) := (others => a(n-1));
```

This line copies the MSB of *a* and adds it to the sign extension bits

c) The line *total := total + unsigned(row);* was replaced with the following conditional statement since in signed multiplication, both addition and subtraction are performed when calculating the total.

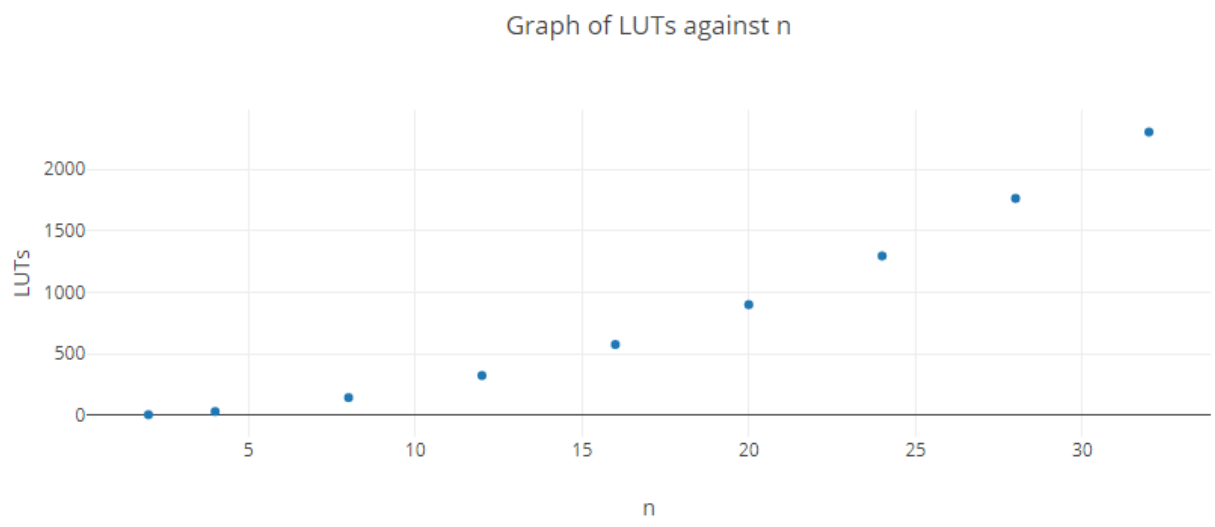
```
if i = n-1 then-- Algorithm 2, Line 5
   total := total - signed(row);-- Algorithm 2, Line 6
else -- Algorithm 2, Line 7
   total := total + signed(row);-- Algorithm 2, Line 8
end if;-- Algorithm 2, Line 9
```

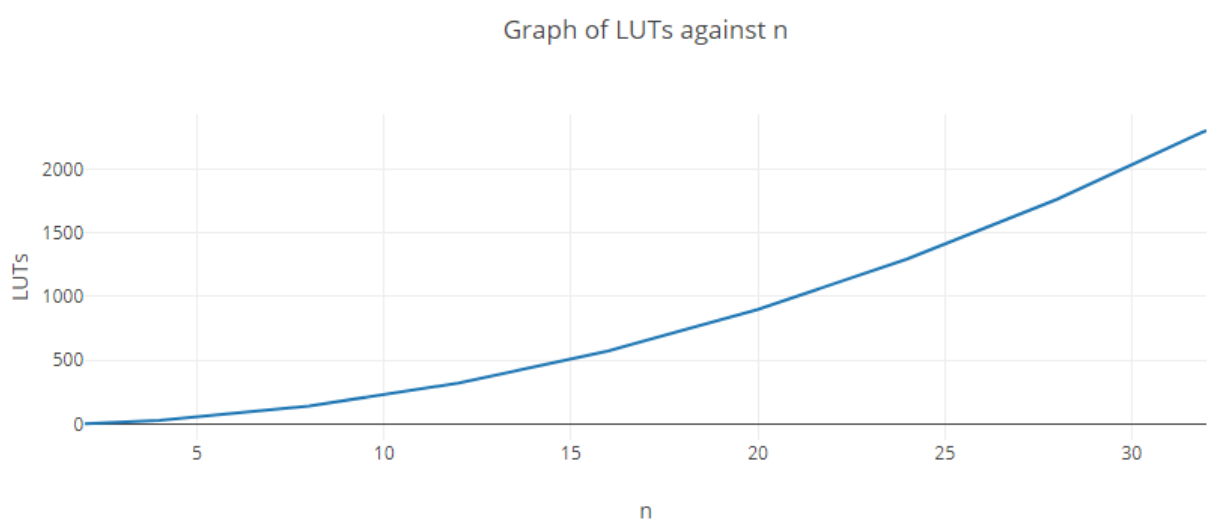
8) The file *mult_unsigned_tb.vhd* was copied to a newly created file *mult_signed_tb.vhd* which was linked to this project. All appearances of *mult_unsigned* were once again changed to *mult_signed* and the test bench was simulated and checked to be correct. The radix of the simulation was changed to *Signed Decimal*.

9) The file *mult_signed.vhd* was set as the **Top Module** and a measure of the hardware area in the range $2 \leq n \leq 32$ was obtained. A table with the possible values was written down. From this table, 2 versions of a graph of LUTs (y-axis) against n (x-axis) were drawn up as shown. The first version shows the plotted points and the second shows the points connected. This graph, just like the one shown in step 6 is a quadratic curve for the same reason explained already.

n	**LUTs**
2	4
4	30
8	143
12	323
16	575

20	899
24	1295
28	1763
32	2303





Appendix (Code):

mult_usnigned.vhd:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mult_unsigned is

    -- Generic constants can be used to set the sizes of ports.
    generic (
        n : integer := 8);

    port (
        a    : in  std_logic_vector(n - 1 downto 0);
        b    : in  std_logic_vector(n - 1 downto 0);
        prod : out std_logic_vector(2 * n - 1 downto 0));

end entity mult_unsigned;

architecture behavioral of mult_unsigned is

begin  -- architecture behavioral

    -- Whenever a signal in the process sensitivity list (a, b) changes,
    -- the outputs are changed according to the process contents.
    shift_mux : process (a, b) is

        -- Variables go here. Unlike signals, variables are sequential,
        -- that is, they can be changed multiple times in sequence.
        variable row   : std_logic_vector(2 * n - 1 downto 0);
        variable total : unsigned(2 * n - 1 downto 0);
    begin  -- process shift_mux

        total := (others => '0');       -- Algorithm 1, Line 1

        -- iterate for each row
        for i in 0 to n - 1 loop        -- Algorithm 1, Line 2

            -- Task 2a: Algorithm 1, Line 3
            row := (others => '0');         -- set row to 0
            row(n-1+i downto i) := a;       -- set n bits to a


            -- Task 2b; Algorithm 1, Line 4
            if b(i) = '1' then
                -- Since total is of type unsigned, we can add using '+',
                -- but row needs to be cast to unsigned as well.
                total := total + unsigned(row);  -- Algorithm 1, Line 5

            end if;  -- Algorithm 1, Line 6

        end loop;  -- Algorithm 1, Line 7

        -- Variables are local to the process, so the result
        -- should be copied to the output signal prod.
        prod <= std_logic_vector(total);
    end process shift_mux;

end behavioral;
```

mult_unsigned_tb.vhd:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mult_unsigned_tb is

end entity mult_unsigned_tb;

architecture behavioral of mult_unsigned_tb is

    component mult_unsigned is
        generic (
            n : integer);
        port (
            a    : in  std_logic_vector(n - 1 downto 0);
            b    : in  std_logic_vector(n - 1 downto 0);
            prod : out std_logic_vector(2 * n - 1 downto 0));
    end component mult_unsigned;

    -- generics
    constant n : integer := 8;

    -- inputs
    signal a : std_logic_vector(n - 1 downto 0) := (others => '0');
    signal b : std_logic_vector(n - 1 downto 0) := (others => '0');

    -- outputs
    signal prod : std_logic_vector(2 * n - 1 downto 0);

begin  -- architecture behavioral

    uut : mult_unsigned
        generic map (
            n => n)
        port map (
            a    => a,
            b    => b,
            prod => prod);

    -- stimulus process
    stim_proc : process
        type arr is array(0 to 5) of std_logic_vector(7 downto 0);

        constant values : arr := (
            x"00",                      -- hex 00
            x"01",                      -- hex 01
            x"7F",
            x"80",
            x"81",
            x"FF"
            );
    begin
        for i in 0 to 5 loop
            a <= values(i);
            for j in 0 to 5 loop
                b <= values(j);
                wait for 10 ns;
            end loop;  -- j
        end loop;  -- i
    end process;

end architecture behavioral;
```

mult_signed.vhd:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mult_signed is

    -- Generic constants can be used to set the sizes of ports.
    generic (
        n : integer := 8);

    port (
        a    : in  std_logic_vector(n - 1 downto 0);
        b    : in  std_logic_vector(n - 1 downto 0);
        prod : out std_logic_vector(2 * n - 1 downto 0));

end entity mult_signed;

architecture behavioral of mult_signed is

begin  -- architecture behavioral

    -- Whenever a signal in the process sensitivity list (a, b) changes,
    -- the outputs are changed according to the process contents.
    shift_mux : process (a, b) is

        -- Variables go here. Unlike signals, variables are sequential,
        -- that is, they can be changed multiple times in sequence.
        variable row   : std_logic_vector(2 * n - 1 downto 0);
        variable total : signed(2 * n - 1 downto 0);
    begin  -- process shift_mux

        total := (others => '0');       -- Algorithm 2, Line 1

        -- iterate for each row
        for i in 0 to n - 1 loop        -- Algorithm 2, Line 2

            -- Algorithm 2, Line 3
            row := (others => '0');         -- set row to 0
            row(n-1+i downto i) := a;       -- set n bits to a
            row(2*n-1 downto n+i) := (others => a(n-1));

            -- Algorithm 2, Line 4

            if b(i) = '1' then

               if i = n-1 then-- Algorithm 2, Line 5
                  total := total - signed(row);-- Algorithm 2, Line 6
               else -- Algorithm 2, Line 7
                  total := total + signed(row);-- Algorithm 2, Line 8
               end if;-- Algorithm 2, Line 9

            end if;  -- Algorithm 2, Line 10

        end loop;  -- Algorithm 2, Line 11

        -- Variables are local to the process, so the result
        -- should be copied to the output signal prod.
        prod <= std_logic_vector(total);
    end process shift_mux;

end behavioral;
```

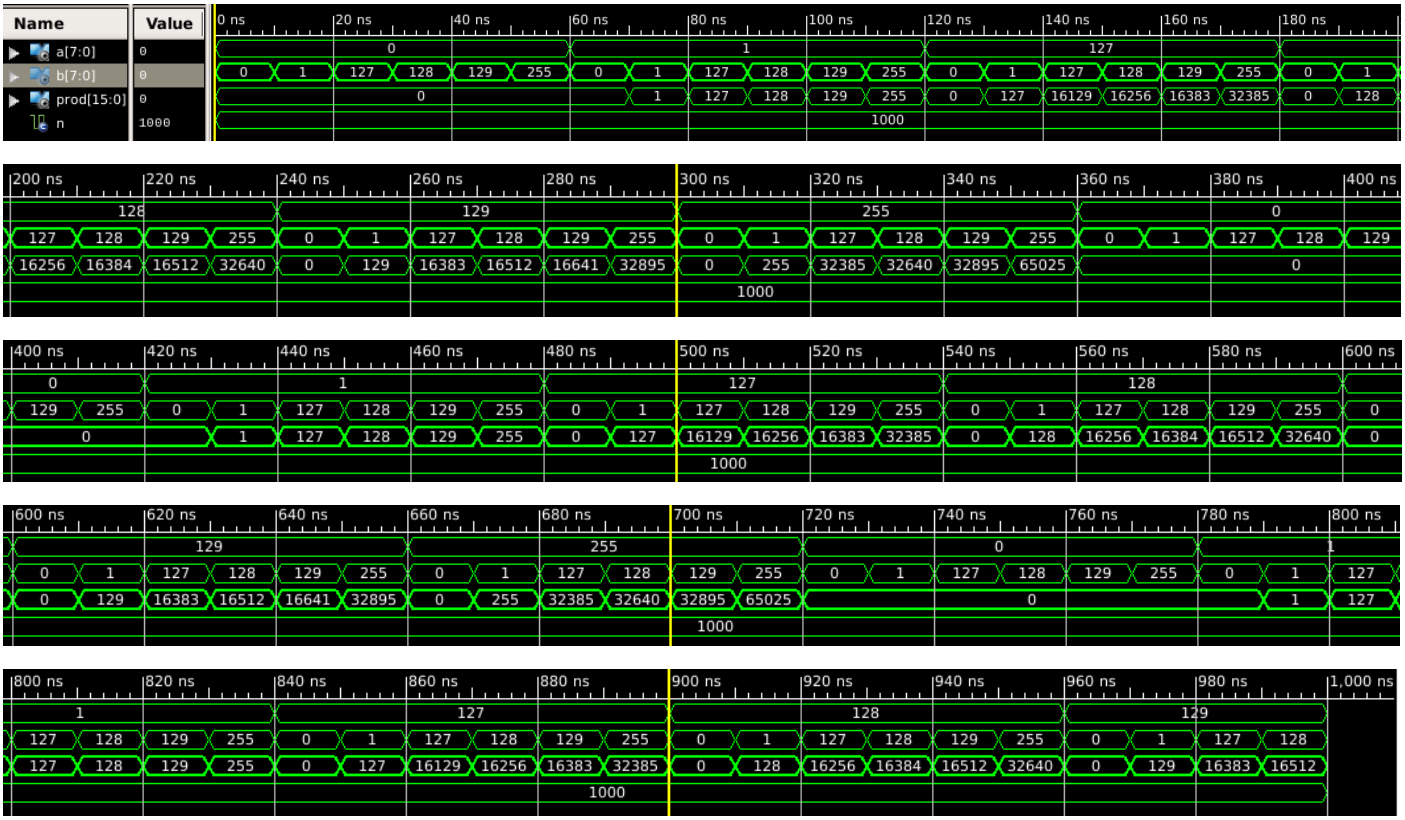
mult_signed_tb.vhd:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mult_signed_tb is

end entity mult_signed_tb;

architecture behavioral of mult_signed_tb is

    component mult_signed is
        generic (
            n : integer);
        port (
            a    : in  std_logic_vector(n - 1 downto 0);
            b    : in  std_logic_vector(n - 1 downto 0);
            prod : out std_logic_vector(2 * n - 1 downto 0));
    end component mult_signed;

    -- generics
    constant n : integer := 8;

    -- inputs
    signal a : std_logic_vector(n - 1 downto 0) := (others => '0');
    signal b : std_logic_vector(n - 1 downto 0) := (others => '0');

    -- outputs
    signal prod : std_logic_vector(2 * n - 1 downto 0);

begin  -- architecture behavioral

    uut : mult_signed
        generic map (
            n => n)
        port map (
            a    => a,
            b    => b,
            prod => prod);

    -- stimulus process
    stim_proc : process
        type arr is array(0 to 5) of std_logic_vector(7 downto 0);

        constant values : arr := (
            x"00",                      -- hex 00
            x"01",                      -- hex 01
            x"7F",
            x"80",
            x"81",
            x"FF"
            );
    begin
        for i in 0 to 5 loop
            a <= values(i);
            for j in 0 to 5 loop
                b <= values(j);
                wait for 10 ns;
            end loop;  -- j
        end loop;  -- i
    end process;

end architecture behavioral;
```

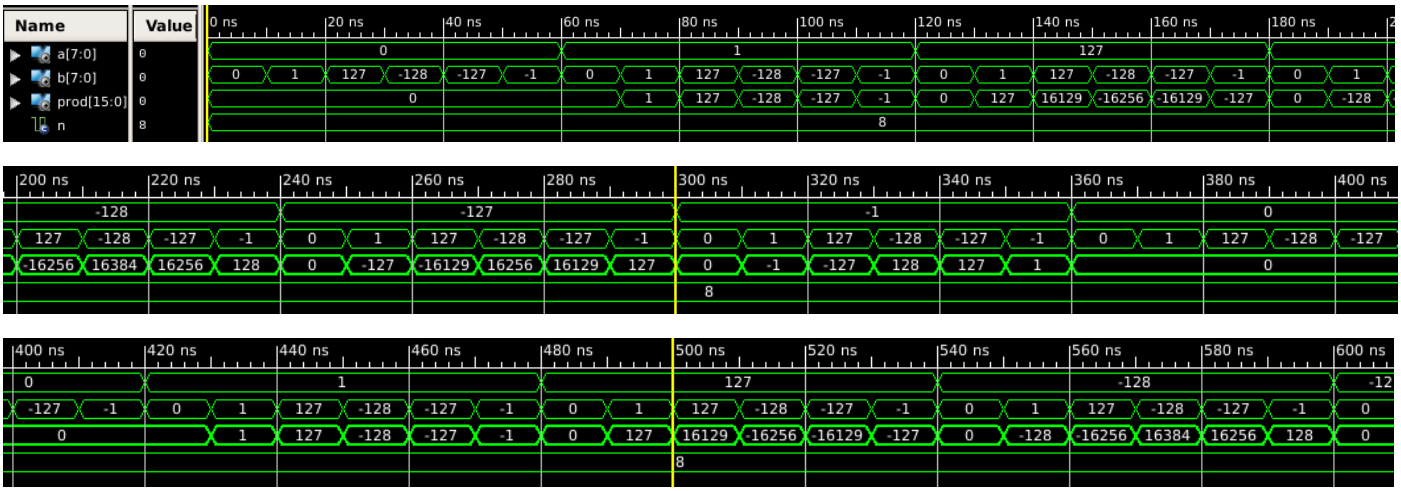
Simulated Behavioral Models:

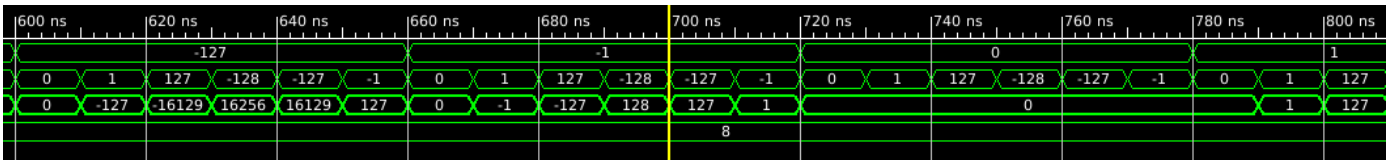
Each snippet below shows only 200ns of the Simulated Model. One snippet leads onto the next 200ns.

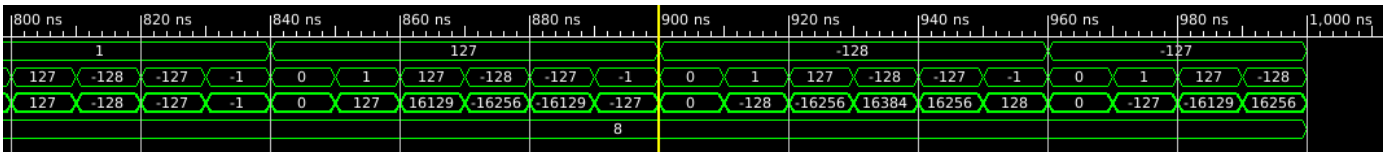
mult_unsigned_tb.vhd:



mult_signed_tb.vhd:







Conclusion:

Signed and unsigned multipliers were successfully implemented in this practical.