**EXPERIMENT -8**

**AIM:** WRITE VHDL CODE FOR ALU AND SHIFT REGISTER AND SIMULATE IT USING

MODELSIM.

**THEORY**: An **arithmetic logic unit (ALU)** is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer. Modern CPUs contain very powerful and complex ALUs. In addition to ALUs, modern CPUs contain a control unit (CU).

Most of the operations of a CPU are performed by one or more ALUs, which load data from input registers.

In digital circuits, a **shift register** is a cascade of flip flops, sharing the same clock, in which the output of each flip-flop is connected to the "data" input of the next flip-flop in the chain, resulting in a circuit that shifts by one position the "bit array" stored in it, "shifting in" the data present at its input and 'shifting out' the last bit in the array, at each transition of the clock input.

More generally, a shift register may be multidimensional, such that its "data in" and stage outputs are themselves bit arrays; this is implemented simply by running several shift registers of the same bit-length in parallel.

**SIPO SHIFT REGISTER –** For Serial in – parallel out shift registers, all data bits appear on the parallel outputs following the data bits enters sequentially through each flipflop. The following circuit is a four-bit Serial in – parallel out shift register constructed by D flip-flops.

**PIPO SHIFT REGISTER –** For parallel in – parallel out shift registers, all data bits appear on the parallel outputs immediately following the simultaneous entry of the data bits. The following circuit is a four-bit parallel in – parallel out shift register constructed by D flip-flops.

**VHDL CODE:**

**ALU CODE**

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

USE ieee.std\_logic\_arith.all;

USE ieee.std\_logic\_unsigned.all;

USE ieee.numeric\_std.all;

ENTITY ALU IS

Port (

A, B : in STD\_LOGIC\_VECTOR(7 downto 0); -- 2 inputs 8-bit

ALU\_Sel : in STD\_LOGIC\_VECTOR(3 downto 0); -- 1 input 4-bit for selecting function

ALU\_Out : out STD\_LOGIC\_VECTOR(7 downto 0); -- 1 output 8-bit

Carryout : out std\_logic -- Carryout flag

);

END ENTITY ALU;

--

ARCHITECTURE ALU\_BEHAV OF ALU IS

signal ALU\_Result : std\_logic\_vector (7 downto 0);

signal tmp: std\_logic\_vector (8 downto 0);

BEGIN

process(A,B,ALU\_Sel)

begin

case(ALU\_Sel) is

when "0000" => -- Addition

ALU\_Result <= A + B ;

when "0001" => -- Subtraction

ALU\_Result <= A - B ;

when "0010" => -- Multiplication

ALU\_Result <= std\_logic\_vector(to\_unsigned ( (to\_integer(unsigned(A) ) \* ( to\_integer(unsigned(B))),8))) ;

when "0011" => -- Division

ALU\_Result <= std\_logic\_vector(to\_unsigned(to\_integer(unsigned(A)) / to\_integer(unsigned(B)),8)) ;

when "0100" => -- Logical shift left

ALU\_Result <= std\_logic\_vector(A sll 1);

when "0101" => -- Logical shift right

ALU\_Result <= std\_logic\_vector(A srl 1);

when "0110" => -- Rotate left

ALU\_Result <= std\_logic\_vector(A rol 1);

when "0111" => -- Rotate right

ALU\_Result <= std\_logic\_vector(A ror 1);

when "1000" => -- Logical and

ALU\_Result <= A and B;

when "1001" => -- Logical or

ALU\_Result <= A or B;

when "1010" => -- Logical xor

ALU\_Result <= A xor B;

when "1011" => -- Logical nor

ALU\_Result <= A nor B;

when "1100" => -- Logical nand

ALU\_Result <= A nand B;

when "1101" => -- Logical xnor

ALU\_Result <= A xnor B;

when "1110" => -- Greater comparison

if(A>B) then

ALU\_Result <= x"01" ;

else

ALU\_Result <= x"00" ;

end if;

when "1111" => -- Equal comparison

if(A=B) then

ALU\_Result <= x"01" ;

else

ALU\_Result <= x"00" ;

end if;

when others => ALU\_Result <= A + B ;

end case;

end process;

ALU\_Out <= ALU\_Result; -- ALU out

tmp <= ('0' & A) + ('0' & B);

Carryout <= tmp(8); -- Carryout flag

END ARCHITECTURE ALU\_BEHAV;

**SHIFT REGISTER SIPO**

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

USE ieee.std\_logic\_arith.all;

ENTITY SHIFTREG\_SIPO IS

port(CLOCK,Input\_Data: in std\_logic; Q: out std\_logic\_vector(3 downto 0) );

END ENTITY SHIFTREG\_SIPO;

--

ARCHITECTURE SHIFTREG\_SIPO\_BEHAV OF SHIFTREG\_SIPO IS

signal qout : std\_logic\_vector ( 3 downto 0);

BEGIN

process (CLOCK)

begin

if (rising\_edge(CLOCK)) then

qout (3 downto 1) <= qout(2 downto 0);

qout(0) <= Input\_Data;

end if;

end process;

Q<= qout;

END ARCHITECTURE SHIFTREG\_SIPO\_BEHAV;

**SHIFT REGISTER PIPO**

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

USE ieee.std\_logic\_arith.all;

ENTITY SHIFTREG\_PIPO IS

port(CLOCK : in std\_logic;D: in std\_logic\_vector(3 downto 0);

Q: out std\_logic\_vector(3 downto 0));

END ENTITY SHIFTREG\_PIPO;

--

ARCHITECTURE SHIFTREG\_PIPO\_BEHAV OF SHIFTREG\_PIPO IS

BEGIN

process (CLOCK)

begin

if (rising\_edge(CLOCK)) then

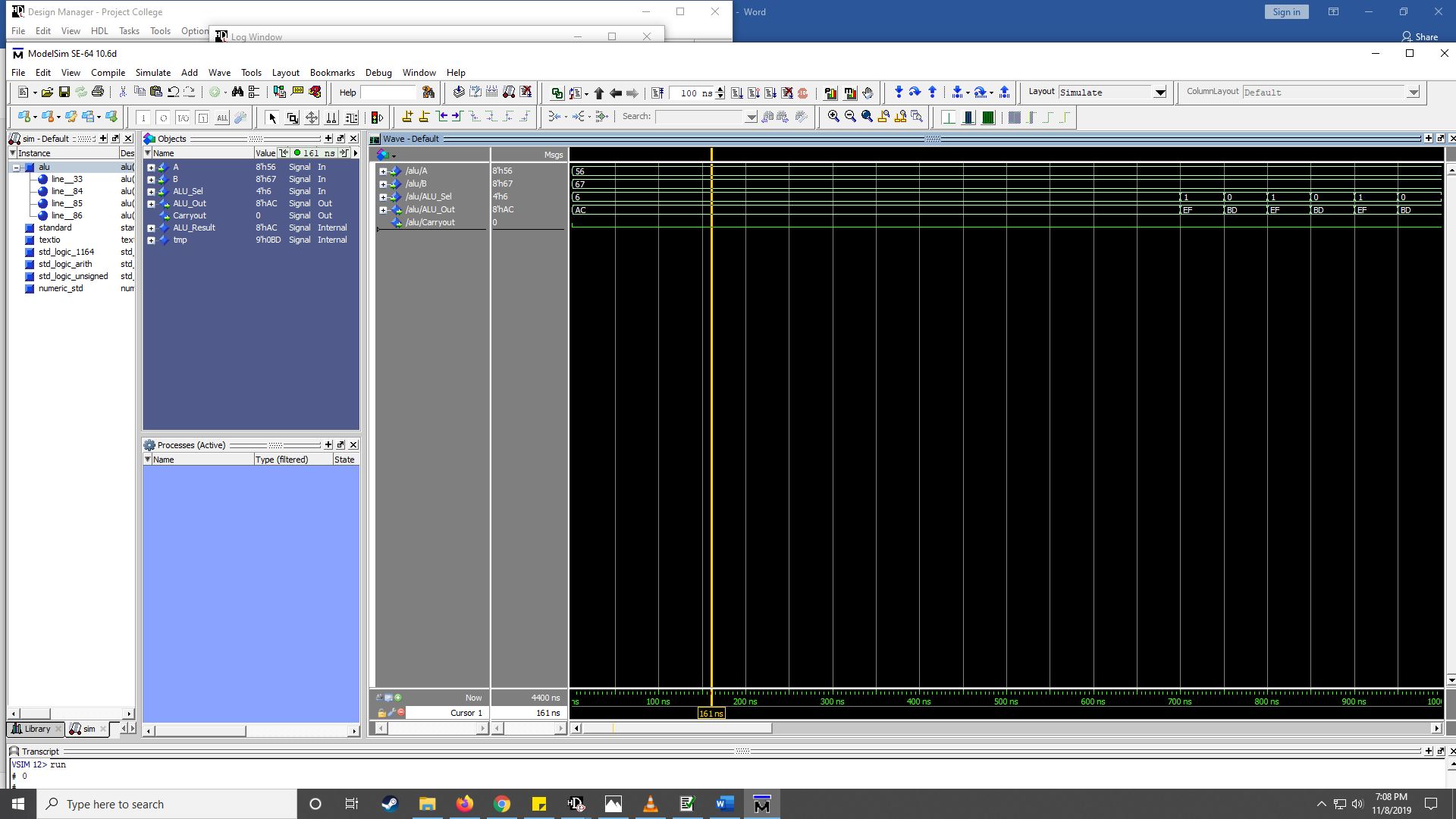
Q <= D;

end if;

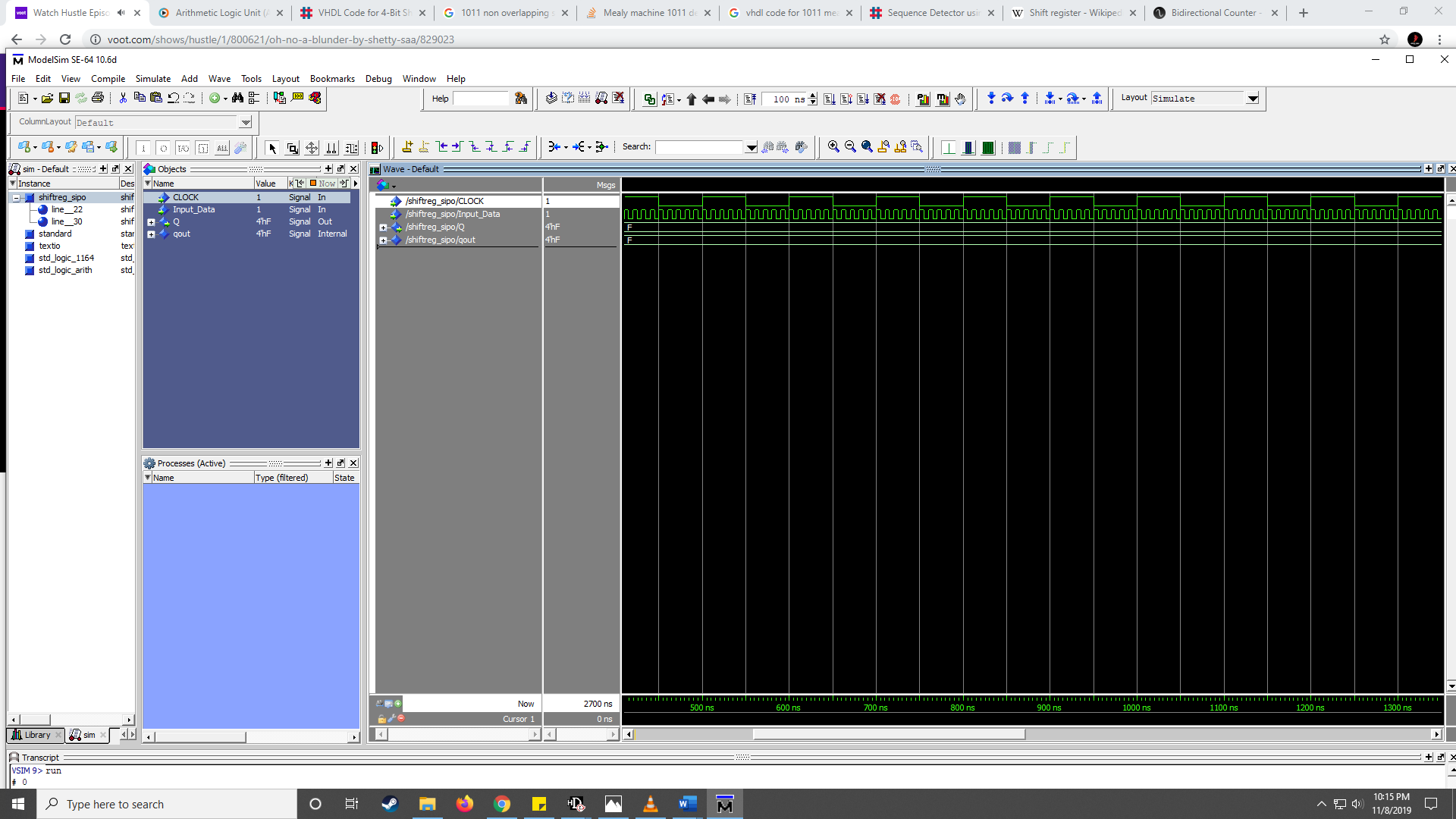
end process;

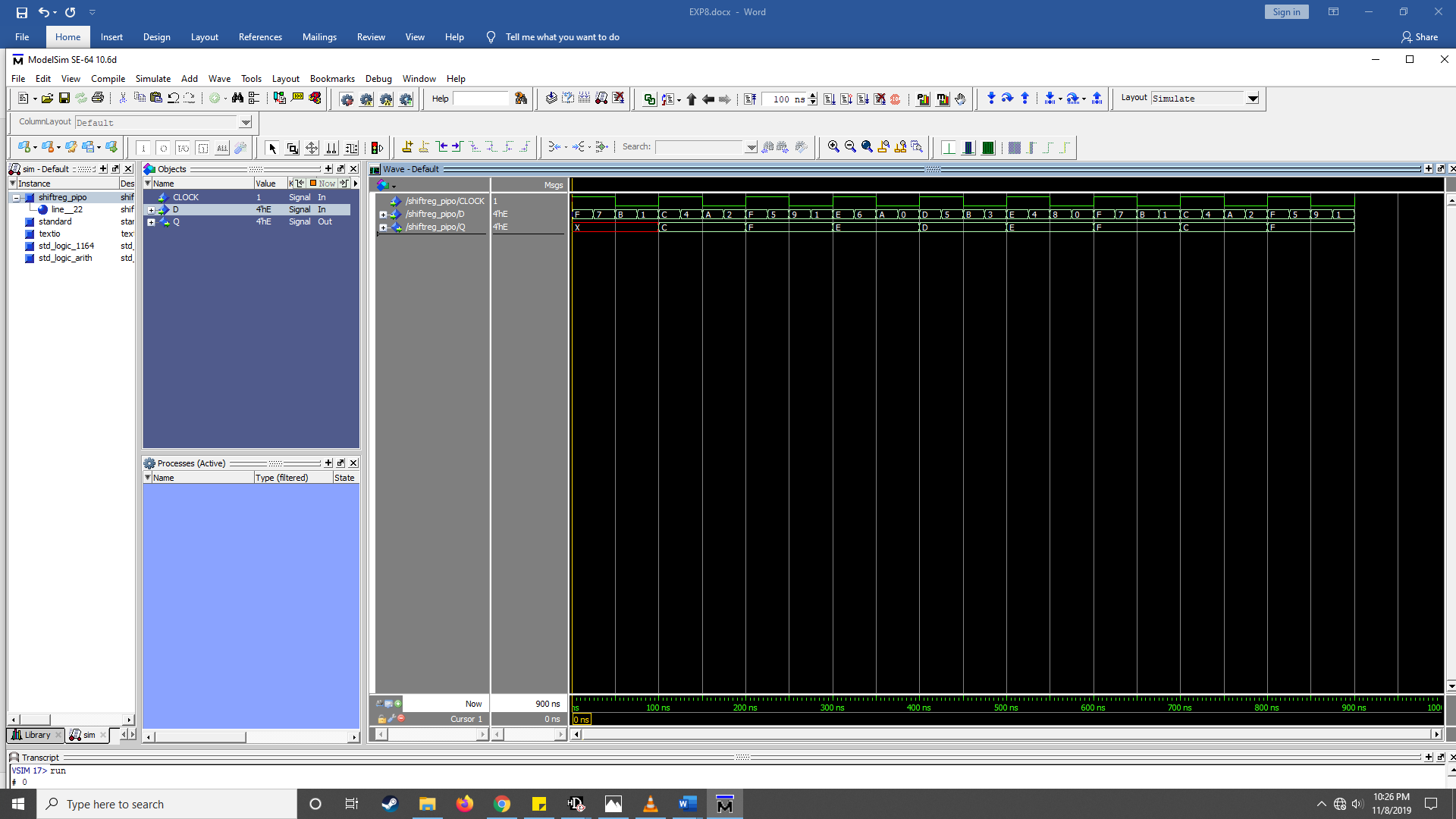
END ARCHITECTURE SHIFTREG\_PIPO\_BEHAV;

**RESULT:**



**ARITHMETIC LOGIC UNIT (ALU)**



**SHIFT REGISTER SIPO**

**SHIFT REGISTER PIPO**