**Description:**

The circuit that was designed was a 16 bit calculator, which had two 16 bit inputs for the two numbers, and a 5 bit operation code parameter. The output was 16 bit as well. This was done using an algorithmic method which was implemented sequentially instead of concurrently. An std\_logic\_vector was used which allowed both logic and arithmetic functions to be performed. This method made sense for this project because it allowed if statements to be written based on the Opcode.

**VHDL Code:**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity CALC is

port(X,Y: in std\_logic\_vector (15 downto 0);

OPC: in std\_logic\_vector (4 downto 0);

OUTPUT: out std\_logic\_vector (15 downto 0));

end CALC;

architecture BEHAV\_CALC of CALC is

begin

CALC\_PROCESS: process (OPC,X,Y)

begin

if (OPC="10000")then

OUTPUT <= Y;

elsif (OPC="10001") then

OUTPUT <= Y+'1';

elsif (OPC="10011") then

OUTPUT <= X+Y;

elsif (OPC="10100") then

OUTPUT <= NOT Y;

elsif (OPC="10101") then

OUTPUT <= 0-Y;

elsif (OPC="10111") then

OUTPUT <= Y-X;

elsif (OPC="11000") then

OUTPUT <= Y-'1';

elsif (OPC="11001") then

OUTPUT <= X-Y;

elsif (OPC="11011") then

OUTPUT <= NOT X;

elsif (OPC="11100") then

OUTPUT <= X AND Y;

elsif (OPC="11101") then

OUTPUT <= X OR Y;

elsif (OPC="11110") then

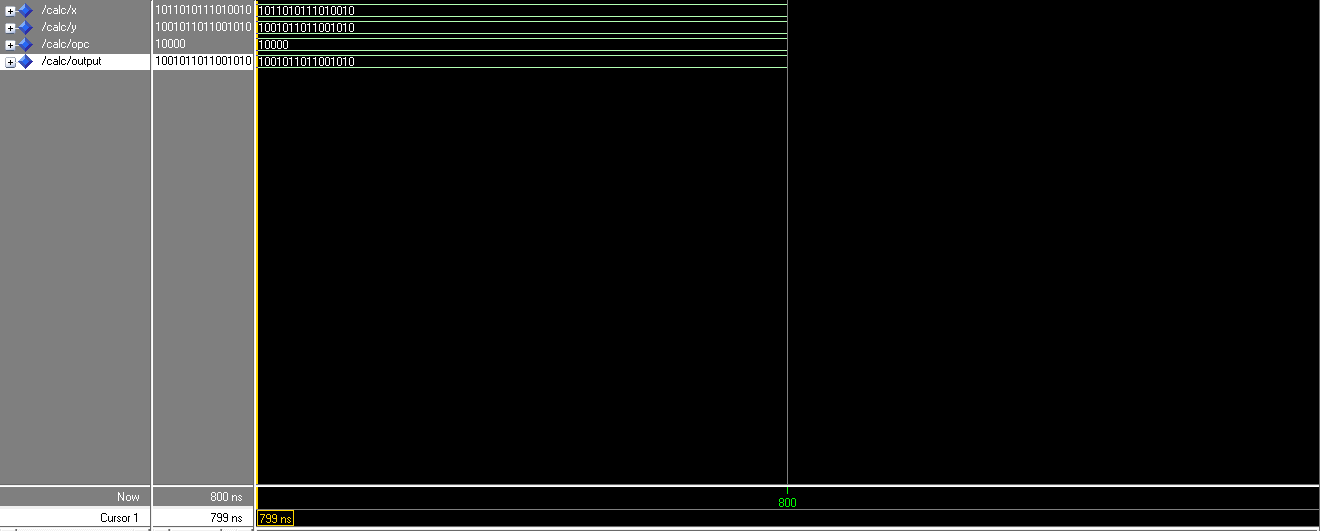
OUTPUT <= X XOR Y;

end if;

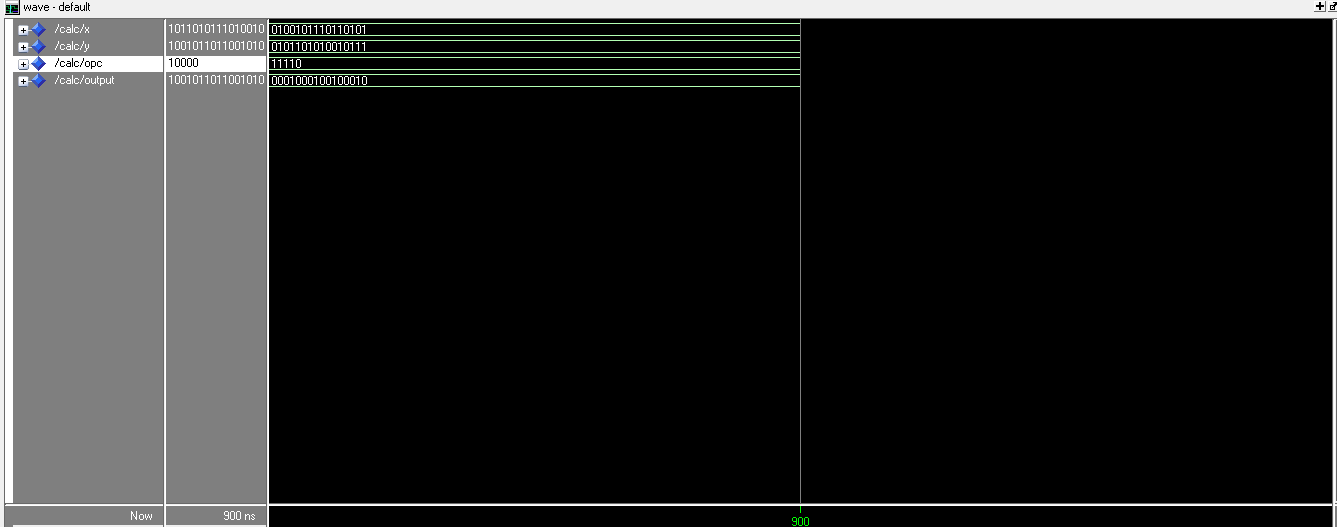
end process;

end BEHAV\_CALC;

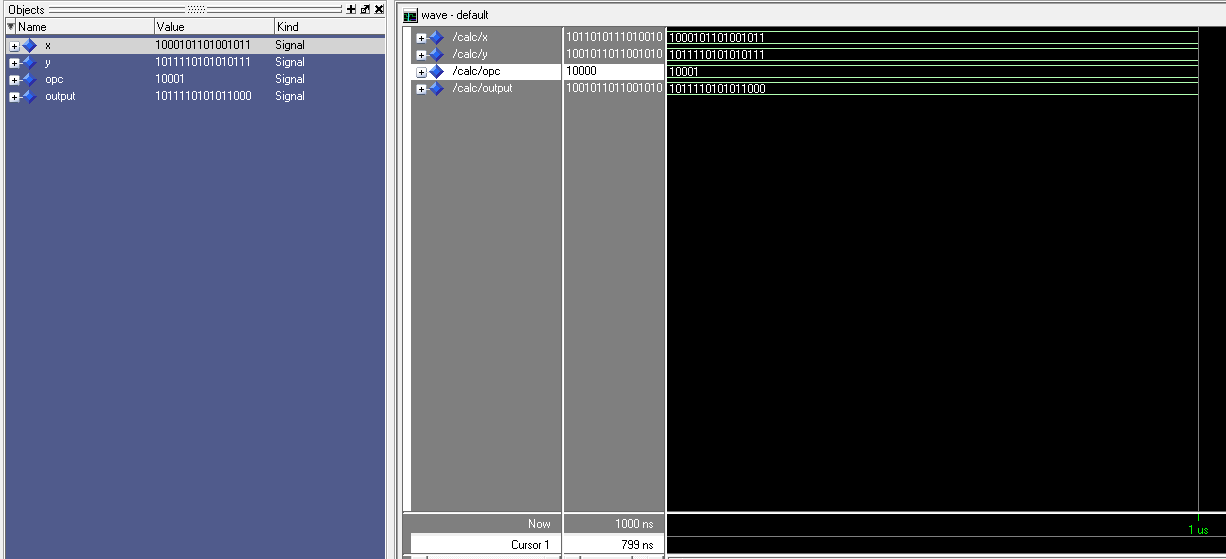
**Simulation:**



The first simulation uses the opcode 10000 which corresponds to outputting the input Y. It can be seen that “ouput” and “y” are the same.

**

The second simulation uses the opcode 11110 which performs X XOR Y. x = 18868 y = 23191 in decimal “18868 XOR 23191” was input in wolfram alpha and returned 4899, which corresponds to the output in binary.



The last simulation uses the opcode 10001 which is Y+1. It can be seen between Y and output that only the 4 least significant bits are changed from the input to output. The change is from “0111” to “1000” which Is the same as adding 1.

**Do-script:**

force -freeze sim:/calc/x 1011010111010010

force -freeze sim:/calc/y 1001011011001010

force -freeze sim:/calc/opc 10000

run

force -freeze sim:/calc/x 0100101110110101

force -freeze sim:/calc/y 0101101010010111

force -freeze sim:/calc/opc 11110

run

force -freeze sim:/calc/x 1000101101001011

force -freeze sim:/calc/y 1011110101010111

force -freeze sim:/calc/opc 10001

run

force -freeze sim:/calc/x 1000101110101001

force -freeze sim:/calc/y 0010110101101011

force -freeze sim:/calc/opc 10101

run