Adaptive Noise Cancellation

VLSI PROJECT REPORT

Electronics and Telecommunication Engineering

Submitted by:

Jyotirmaya Mohanta-B215020
Suraj Tripathy-B215059
DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION
IIIT Bhubaneswar.

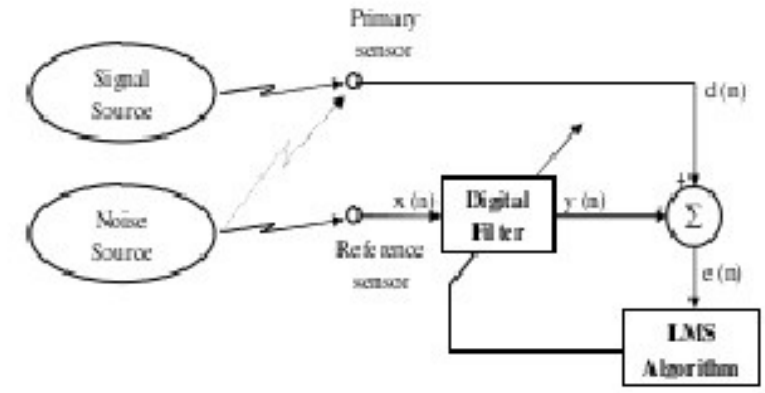
TABLE OF CONTENTS

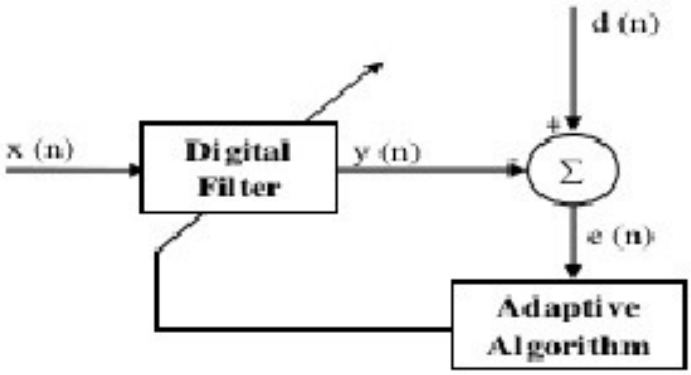
CHAPTER 1
INTRODUCTION

Adaptive filters, as part of digital signal systems, have been widely used in communication industry, as well as in applications such as adaptive noise cancellation, adaptive beam forming, and channel equalization. However, its implementation takes a great deal and becomes a very important field in digital system design. An adaptive filter is usually implemented in DSP processors because of their capability of performing fast floating-point arithmetic .But when FPGA (Field Programmable Logic Array) grows in area and provides a lot of facilities to the designers, it becomes an important competitor in the signal processing market. In addition, FPGA is a form of programmable logic, which offers flexibility for repetitive reconfiguration. Since FPGA consists of slices organized as array of rows and columns, a great deal of parallelism can be explored. Although it is not efficient to use floating-point arithmetic in FPGA due to its need fora large area, it is sufficient to use fixed-point arithmetic for the adaptive filter to work well. In general FIR structure has been used more successfully than IIR structure in adaptive filters. The output FIR filters is the convolution of its input with its coefficients which have constant values. However, when the adaptive FIR filter was made this required appropriate algorithm to update the filter coefficients. The algorithm used to update the filter coefficient is the Least Mean Square (LMS) algorithm which is known for its simplification, low computational complexity, and better performance in different running environments. When compared to other algorithms used for implementing adaptive filters the LMS algorithm is seen to perform very well in terms of the number of iterations required for convergence. Recursive Least Squares algorithm, for example, is faster in convergence than the

LMS but is then very complex to implement, hence detaining system performance in terms of speed and FPGA area used. One of the adaptive filter applications is the adaptive noise canceller. Figure 2 describes its structure where the desired response is composed of a signal plus noise, which is uncorrelated with the signal. The filter input is a sequence of noise which is correlated with the noise in the desired signal. By using the LMS algorithm inside the adaptive filter, the error term e (n) produced by this system is then the original signal

Noise Cancellation scheme





CHAPTER 2
ADAPTIVE FILTER ALGORITM

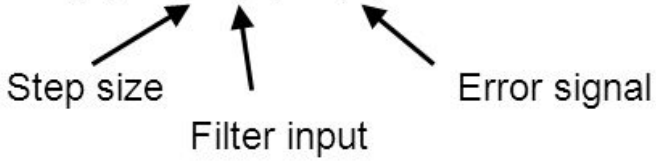
The equalization algorithm is very essential to the performance of equalizer. With the development of equalization technology, there are many algorithms we can use. Of all kinds of adaptive algorithms, Least Mean Square (LMS) algorithm is known for its simplification, low computational complexity, and better performance in different running environments. LMS algorithm may be described using the following equations:

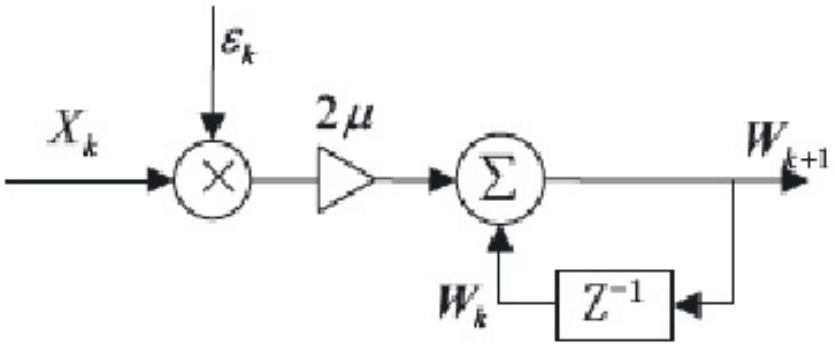
$$y(n)_{new} = y(n)_{old} + w_{new} * x(n)$$
$$w(n)_{new} = w(n)_{old} + \mu * x(n) * e(n)$$
$$e(n) = d(n) - y(n)$$

Where μ means the step size.

The update equation for the LMS algorithm is

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \mu \mathbf{u}(n) e^*(n)$$

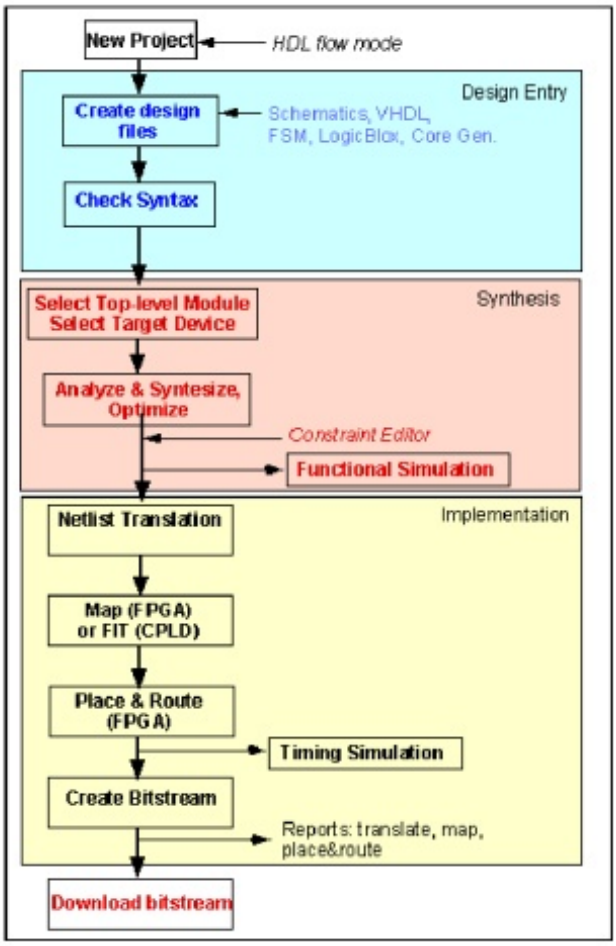




CHAPTER 3
SOFTWARE PLATFORM

General description on VHDL

Hardware description languages (HDLs) are used to describe hardware for the purpose of simulation, modelling, design, testing and documentation. These languages are used to represent the functional and wiring details of digital systems in a compact form. It must uniquely and unambiguously define the hardware at various levels of abstraction. The IEEE standard VHDL is such a language. VHDL stands for VHSICHDL where VHSIC stands for Very High Speed Integrated Circuit. It was develop din 1981 by the US Department of Defence as a language to describe the structure and function of hardware. IEEE standardized it in 1987. It is widely used as a standard tool for the design, manufacturing and documentation of digital circuits of various levels of complexity. The only other major HDL which is prevalent is the IEEE standard Verilog HDL.VHDL is a general-purpose hardware description language which can be used to describe and simulate the operation of a wide range of digital systems starting with those containing only a few gates up to systems formed by the interconnection of many complex integrated circuits. VHDL can describe a system at the behavioural, data flow and structural levels. At the behavioural level, the digital system is described by the functions that it performs. It provides no details as



to how the design is implemented. Complex hardware units are first described at the behavioural level to simulate and test the design ideas. At the data flow level, the system is described in terms of the flow of control and the movement of data. This involves the architecture of busses and control hardware. The structural description is the lowest and most detailed level of description and is the simplest to synthesize into hardware. It includes a list of concurrently active components and their interconnections .A VHDL description has two main parts. They are the entity part and the architecture part. The entity describes the system as a single component as seen by external devices. This part provides information about the system's interface while revealing nothing about its internal structure and behaviour. The architecture part of a VHDL description is used to specify the behaviour and the internal structure of the system.

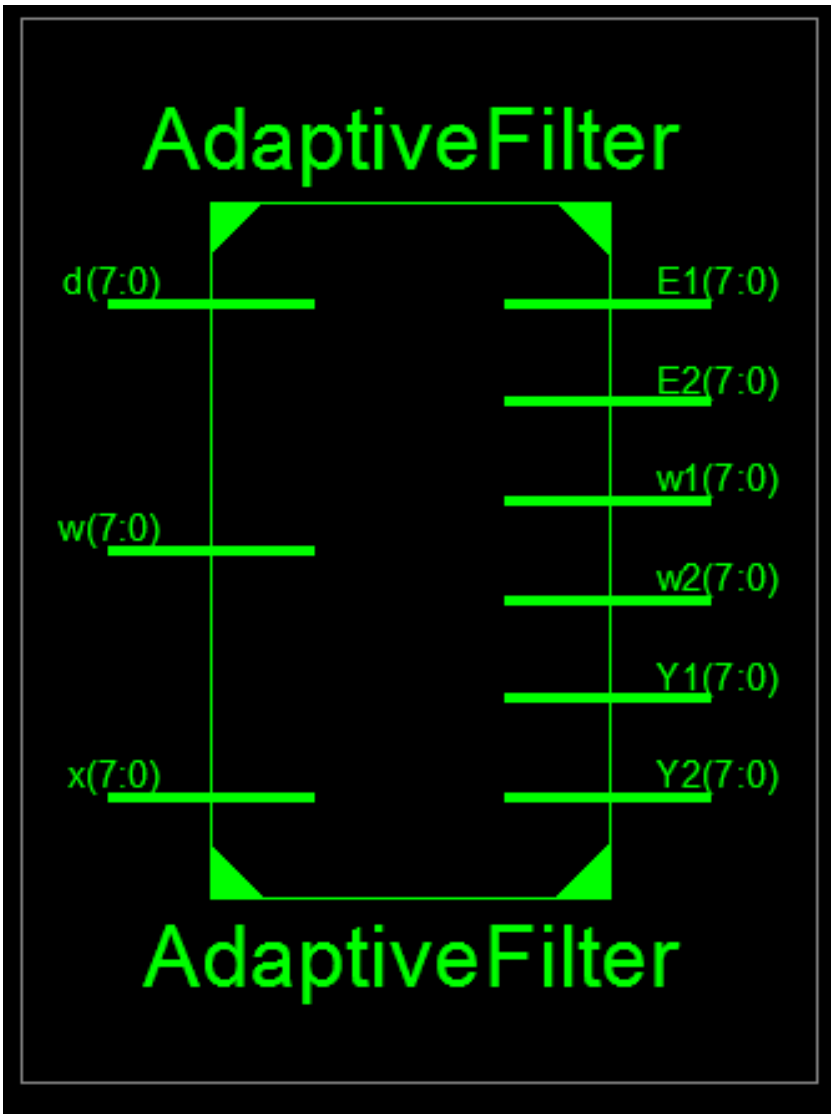
Steps in VHDL Design Process

VHDL is conducive to top-down design. So the whole system is describe that the block diagram level and each component in the diagram is described in further detail. The system can be simulated at each level of design to observe the output.

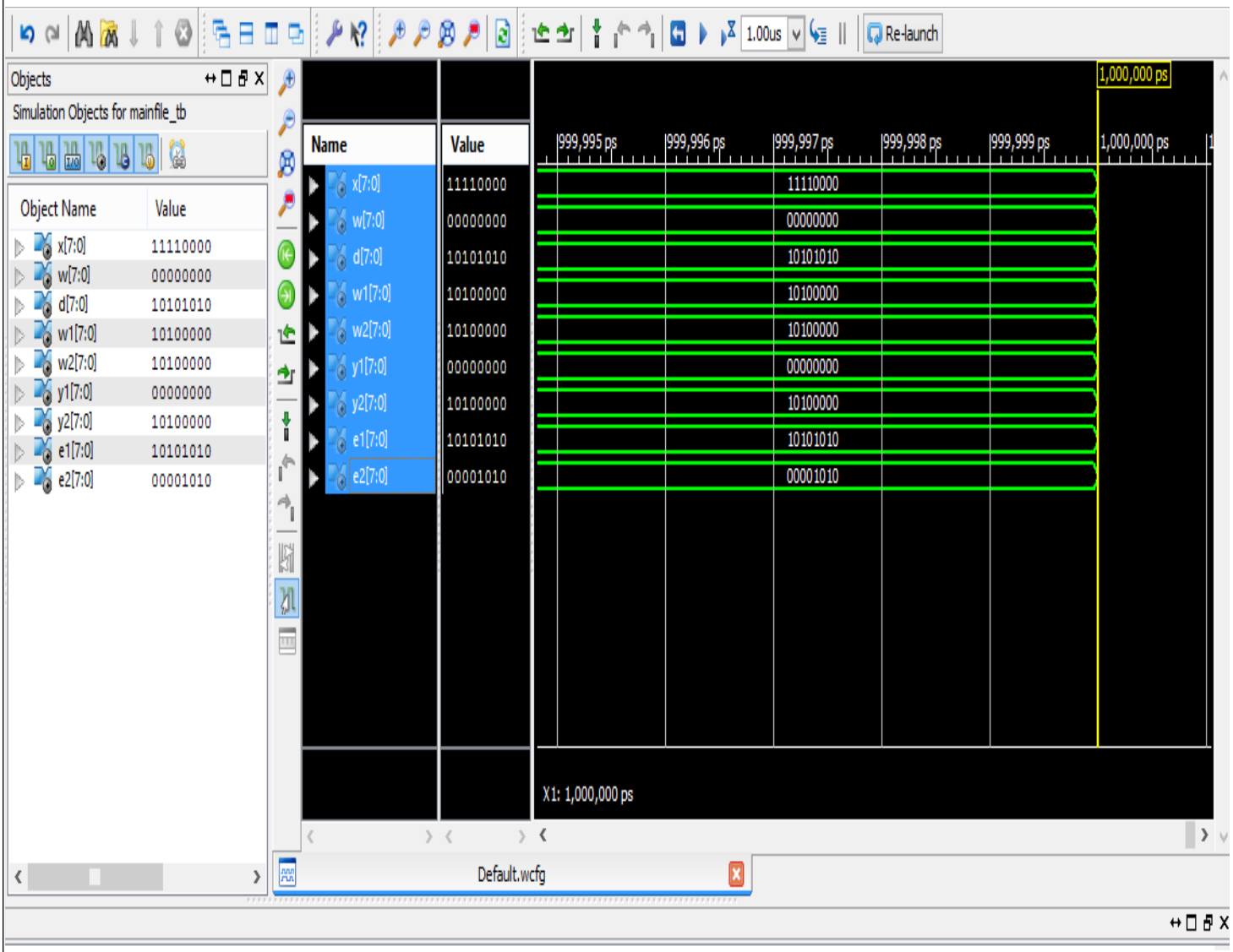
CHAPTER 4
SIMULATION AND RESULTS

An adaptive filter module is designed using VHDL. The inputs to this module are

RTL Schematic:



Simulation output:



VHDL CODE:

MainFile.vhd (Adaptive Filter)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity AdaptiveFilter is
        Port ( x: in STD_LOGIC_VECTOR(7 downto 0);
               w: in STD_LOGIC_VECTOR(7 downto 0);
                d: in STD_LOGIC_VECTOR(7 downto 0);
                w1: inout STD_LOGIC_VECTOR(7 downto 0);
               w2: inout STD_LOGIC_VECTOR(7 downto 0);
                Y1: inout STD_LOGIC_VECTOR(7 downto 0);
               Y2: inout STD_LOGIC_VECTOR(7 downto 0);
               E1: inout STD_LOGIC_VECTOR(7 downto 0);
                E2: inout STD_LOGIC_VECTOR(7 downto 0));
end AdaptiveFilter;

architecture Structural of AdaptiveFilter is

signal Eu1: STD_LOGIC_VECTOR(7 downto 0);
signal Eux1: STD_LOGIC_VECTOR(7 downto 0);
signal Eu2: STD_LOGIC_VECTOR(7 downto 0);
signal Eux2: STD_LOGIC_VECTOR(7 downto 0);

component fadder
port ( a: in STD_LOGIC_VECTOR(7 downto 0);
      b: in STD_LOGIC_VECTOR(7 downto 0);
      sum:out STD_LOGIC_VECTOR(7 downto 0));
end component;
component fmultiplier
```

```
port ( a1: in STD_LOGIC_VECTOR(7 downto 0);
       b1: in STD_LOGIC_VECTOR(7 downto 0);
       mul: out std_logic_vector(7 downto 0) );
 end component;
 begin
mul0: fmultiplier port map(x,w,Y1);
add0: fadder port map(Y1,d,E1);
mul1: fmultiplier port map(E1,"11111111",Eu1);
mul2: fmultiplier port map(Eu1,x,Eux1);
add1: fadder port map(w,Eux1,w1);

mul3: fmultiplier port map(w1,x,Y2);
add2: fadder port map(Y2,d,E2);
mul4: fmultiplier port map(E2,"11111111",Eu2);
mul5: fmultiplier port map(Eu2,x,Eux2);
add3: fadder port map(w1,Eux2,w2);


end Structural;
```

fadder.vhd:

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

entity fadder is
port ( a: in STD_LOGIC_VECTOR(7 downto 0);
       b: in STD_LOGIC_VECTOR(7 downto 0);
       sum:out STD_LOGIC_VECTOR(7 downto 0) );
end fadder;

architecture Behavioral of fadder is


begin
 sum(0)<= a(0) xor b(0);
 sum(1)<= a(1) xor b(1);
 sum(2)<= a(2) xor b(2);
 sum(3)<= a(3) xor b(3);
 sum(4)<= a(4) xor b(4);
 sum(5)<= a(5) xor b(5);
 sum(6)<= a(6) xor b(6);
 sum(7)<= a(7) xor b(7);
```

```
end Behavioral;
```

fmultiplier.vhd:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity fmultiplier is
PORT ( a1: in STD_LOGIC_VECTOR(7 downto 0);
                    b1: in STD_LOGIC_VECTOR(7 downto 0);
                    mul:out STD_LOGIC_VECTOR(7 downto 0) );
end fmultiplier;

architecture Behavioral of fmultiplier is

begin
mul(0)<= a1(0) and b1(0);
mul(1)<= a1(1) and b1(1);
mul(2)<= a1(2) and b1(2);
mul(3)<= a1(3) and b1(3);
mul(4)<= a1(4) and b1(4);
mul(5)<= a1(5) and b1(5);
mul(6)<= a1(6) and b1(6);
mul(7)<= a1(7) and b1(7);

end Behavioral;
```

MainFile_tb.vhd (Test bench):

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY MainFile_tb IS
END MainFile_tb;

ARCHITECTURE behavior OF MainFile_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT AdaptiveFilter
    PORT(
         x : IN  std_logic_vector(7 downto 0);
         w : IN  std_logic_vector(7 downto 0);
         d : IN  std_logic_vector(7 downto 0);
         w1 : INOUT  std_logic_vector(7 downto 0);
         w2 : INOUT  std_logic_vector(7 downto 0);
         Y1 : INOUT  std_logic_vector(7 downto 0);
         Y2 : INOUT  std_logic_vector(7 downto 0);
         E1 : INOUT  std_logic_vector(7 downto 0);
         E2 : INOUT  std_logic_vector(7 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal x : std_logic_vector(7 downto 0) := (others => '0');
   signal w : std_logic_vector(7 downto 0) := (others => '0');
   signal d : std_logic_vector(7 downto 0) := (others => '0');

            --BiDirs
   signal w1 : std_logic_vector(7 downto 0);
   signal w2 : std_logic_vector(7 downto 0);
   signal Y1 : std_logic_vector(7 downto 0);
   signal Y2 : std_logic_vector(7 downto 0);
   signal E1 : std_logic_vector(7 downto 0);
   signal E2 : std_logic_vector(7 downto 0);
   -- No clocks detected in port list. Replace <clock> below with
   -- appropriate port name

--   constant <clock>_period : time := 10 ns;

BEGIN

            -- Instantiate the Unit Under Test (UUT)
```

```
  uut: AdaptiveFilter PORT MAP (
        x => x,
        w => w,
        d => d,
        w1 => w1,
        w2 => w2,
        Y1 => Y1,
        Y2 => Y2,
        E1 => E1,
        E2 => E2
      );

  -- Clock process definitions
--  <clock>_process :process
  --begin
                    --<clock> <= '0';
                    --wait for <clock>_period/2;
                    --<clock> <= '1';
                    --wait for <clock>_period/2;
  --end process;


  -- Stimulus process
  stim_proc: process
  begin
    -- hold reset state for 100 ns.
    wait for 100 ns;

   -- wait for <clock>_period*10;

    -- insert stimulus here
                    w<="00000000";
    x<="11110000";
                    d<="10101010";
    wait;
  end process;

END;
```

CONCLUSION:

Optimal weights are acquired by the continuous updation of the initial weight selection, over many iterations, which in turn reduces the error in the input signal significantly.
Thus, the noise is reduced in the given input signal.