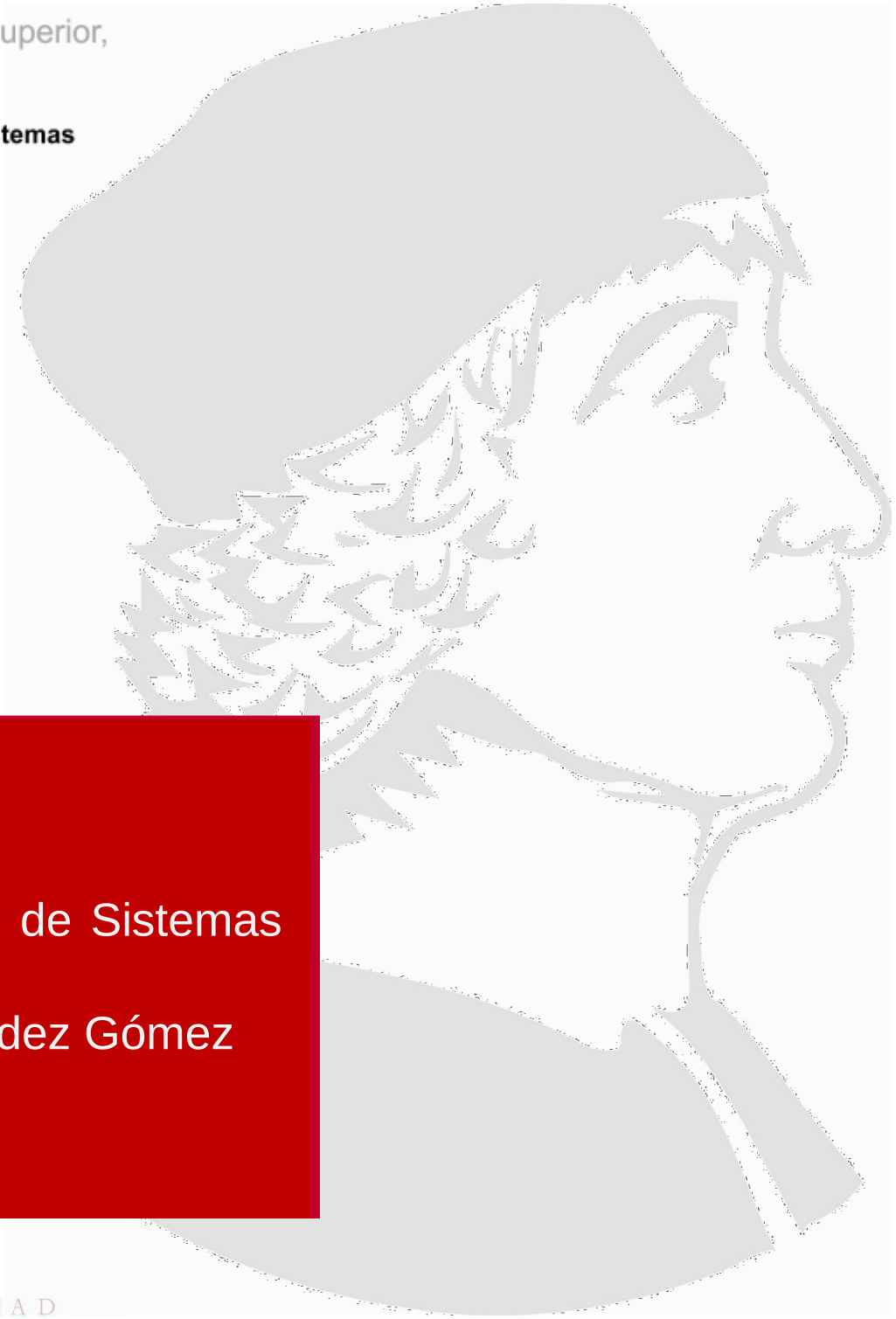
Escuela Politécnica Superior,
Grado en Informática

Diseño Automático de Sistemas
Prof: Luis Pérez Miguel



Práctica 5

Diseño Automático de Sistemas Fiables

José María Fernández Gómez
Álvaro Terrón
Gonzalo Vázquez

UNIVERSIDAD NEBRIJA

UNIVERSIDAD NEBRIJA

PRACTICA 5 FILTROS

INDICE

Qué es un filtro

Un filtro es cualquier componente que afecte a nuestra señal, normalmente los modelizamos tal que sea una función matemática que afecte a una señal de entrada para conseguir una función de salida. Pueden apliucarse para lgunas casuísticas como limpiar señales de entrada para conseguir una salida más limpia, dejar pasar ciertas bandas de señales como puede ser un filtro paso bajo o paso alto.

Métricas

1. Troughput

Cantidad de datos que procesa el sistema por unidad de entrada

2 . Latencia

Tiempo de respuesta en obtener una salida desde que se proporcionó una entrada

3. Area

Cantidad de operadores funcionales que el sistema emplea (sumadores, multiplicadores etc)

Filtros Digitales

Se pueden aplicar filtros digitales a señales tanto digitales como analógicas, ya que podemos transformar una señal analógica a una digital

Filtros Analógicos

Los filtros analógicos sólo pueden ser aplicados sobre señales analógicas

Filtros FIR

Finite impulse response, estos filtros, como su propio nombre indica, tienen un comienzo y un final, no tienen realimentación, por lo que la salida de este mismo no afecta al procesamiento de la siguiente señal de entrada

Filtros IR

Infinite impulse response, son filtros, que al contrario de los IR sí tienen realimentación, por lo que pueden entrar en un bucle infinito, tomando en cuenta ese valor de estado previo

DESCRIPCIÓN HW

¿Qué es un filtro?

Un filtro puede considerarse cualquier dispositivo, sistema o equipo que suministrándole unos datos de entrada, en este caso esos datos serían una señal, ya sea analógica o digital, realiza algún tipo de operativa sobre estas y devuelve una nueva señal habiéndole aplicado dicha operativa o función a la de entrada.

¿Qué es un pipeline?

Es una técnica que consiste en introducir registros entre medias del camino crítico para poder aumentar la frecuencia total del sistema, esto como consecuencia, también aumenta la latencia y ligeramente el área

Uso de Pipelines y Arrays de Vectores

1. Definición de Tipos de Señal para el Pipeline:

•

UNIVERSIDAD NEBRIJA

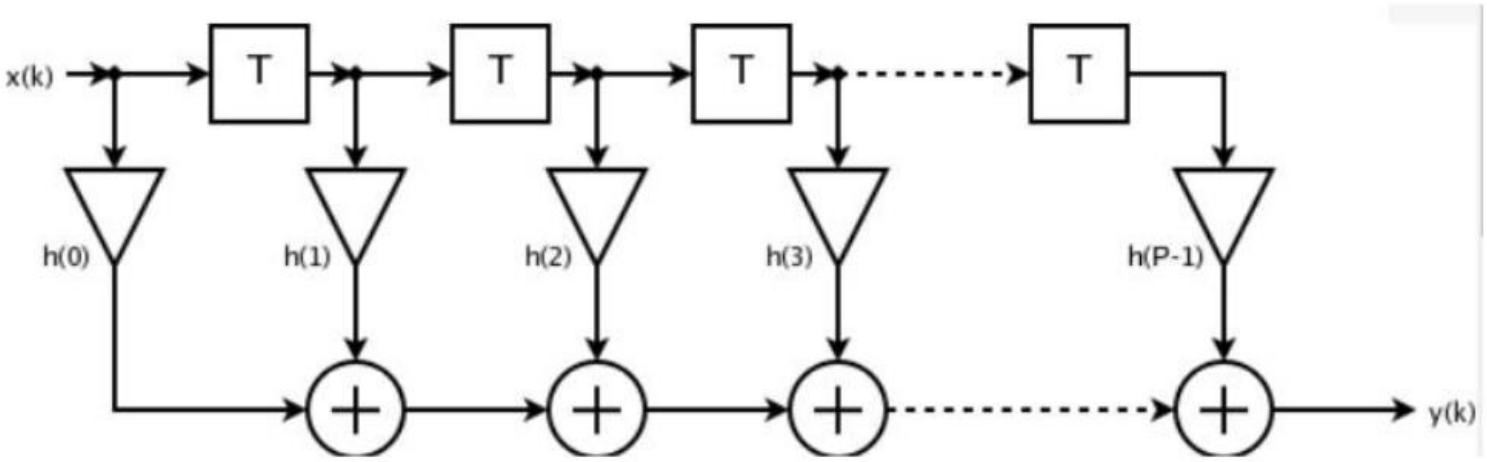
ARCHITECTURE

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

entity fir_filter is
    generic(
        g_width: integer := 8
    );
    port (
        clk         :in std_logic;
        rst         :in std_logic;
        -- Coeficientes
        beta1       :in std_logic_vector(g_width -1 downto 0);
        beta2       :in std_logic_vector(g_width -1 downto 0);
        beta3       :in std_logic_vector(g_width -1 downto 0);
        beta4       :in std_logic_vector(g_width -1 downto 0);
        -- Data input 8 bit
        i_data      :in std_logic_vector(g_width -1 downto 0);
        -- Filtered data
        o_data      :out std_logic_vector(7 downto 0)
    );

end fir_filter;
```

Aquí se muestra la arquitectura del filtro a implementar, siendo i_data la entrada de nuestra señal a filtrar y o_data la salida ya filtrada, las otras entradas, las betas, son los coeficientes del filtro, que son valores con los que el filtro realiza su operativa interna. Si como entrada a este filtro suministráramos una función impulso o delta de Dirac, veríamos de manera pura como esos coeficientes escalares afectarían a nuestra señal

UNIVERSIDAD NEBRIJA

DISEÑO FILTRO 4 TAPS



LECTURA DE PARÁMETROS DE ENTRADA

Se va a realizar una lectura sobre un archivo de texto plano que irá modificando los parámetros de entrada

```
  variable v_time : time;
  variable v_rst : integer;
  variable v_btn : integer;

  -- Variables para los valores de salida

  variable v_xptc_led : integer;

begin

file_open(v_estado_input, file_input, "../../../../input.txt", read_mode);
file_open(v_estado_input, file_output, "../../../../output.txt", write_mode);

-- Verificamos que se haya abierto
assert v_estado_input = open_ok
  report "Could not open file input.txt"
  severity failure;

assert v_estado_output = open_ok
  report "Could not open file output.txt"
  severity failure;
```

IMPLEMENTACION FILTRO

```
begin

    U_fir_test_data_generator : fir_test_data_generator
        port map (
            clk => clk,
            rst => rst,
            pattern_sel => pattern_sel,
            enable => enable,
            o_data => w_data_test
        );

    UUT : fir_filter
        port map (
            clk => clk,
            rst => rst,
            beta1 => beta1,
            beta2 => beta2,
            beta3 => beta3,
            beta4 => beta4,
            i_data => w_data_test,
            o_data => o_data
        );
```

```
component fir_filter
    port (
        clk     : in std_logic;
        rst     : in std_logic;
        beta1   : in std_logic_vector(7 downto 0);
        beta2   : in std_logic_vector(7 downto 0);
        beta3   : in std_logic_vector(7 downto 0);
        beta4   : in std_logic_vector(7 downto 0);
        i_data  : in std_logic_vector(7 downto 0);
        o_data  : out std_logic_vector(9 downto 0)
    );
end component;

file file_input : text;
constant freq : integer := 100_000;  -- KHZ
constant clk_period : time := (1 ms / freq);

signal clk : std_logic := '0';
signal rst : std_logic := '0';
signal pattern_sel : integer := 0;
signal enable : std_logic := '0';
signal w_data_test : std_logic_vector(7 downto 0) := (others => '0');
signal o_data : std_logic_vector(9 downto 0) := (others => '0');
```

Aquí, se instancia tanto el filtro como el generador de datos, una vez realizado este paso en el testbench, simplemente habría que realizar las conexiones pertinentes entre ambos dos, pero, como se pide en el enunciado, el selector de modo de actuación del filtro se ha realizado a través de un archivo de texto externo, por lo que el modo de actuación de este vendrá dada por la entrada que el usuario haya dictado a través de este archivo

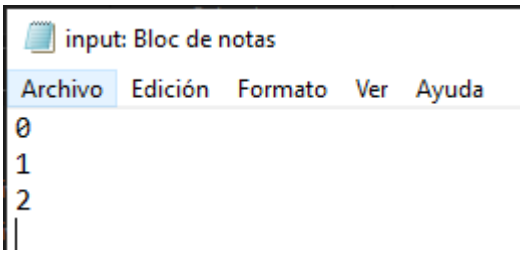
UNIVERSIDAD NEBRIJA

TESTBENCH

```
constant beta1 : std_logic_vector(7 downto 0) := std_logic_vector(to_signed(-10, 8));
constant beta2 : std_logic_vector(7 downto 0) := std_logic_vector(to_signed(110, 8));
constant beta3 : std_logic_vector(7 downto 0) := std_logic_vector(to_signed(127, 8));
constant beta4 : std_logic_vector(7 downto 0) := std_logic_vector(to_signed(-20, 8));

component fir_test_data_generator
    port (
        clk         : in  std_logic;
        rst         : in  std_logic;
        pattern_sel : in  integer;  -- 0 => delta; 1 => step; 2 => sine
        enable      : in  std_logic;
        o_data      : out std_logic_vector(7 downto 0)  -- to FIR
    );
end component;

component fir_filter
    port (
        clk     : in std_logic;
        rst     : in std_logic;
        beta1   : in std_logic_vector(7 downto 0);
        beta2   : in std_logic_vector(7 downto 0);
        beta3   : in std_logic_vector(7 downto 0);
        beta4   : in std_logic_vector(7 downto 0);
        i_data  : in std_logic_vector(7 downto 0);
        o_data  : out std_logic_vector(9 downto 0)
    );
end component;

file file_input : text;
constant freq : integer := 100_000;  -- KHZ
constant clk_period : time := (1 ms / freq);

signal clk : std_logic := '0';
signal rst : std_logic := '0';
signal pattern_sel : integer := 0;
```

```
begin

    U_fir_test_data_generator : fir_test_data_generator
        port map (
            clk => clk,
            rst => rst,
            pattern_sel => pattern_sel,
            enable => enable,
            o_data => w_data_test
        );

    UUT : fir_filter
        port map (
            clk => clk,
            rst => rst,
            beta1 => beta1,
            beta2 => beta2,
            beta3 => beta3,
            beta4 => beta4,
            i_data => w_data_test,
            o_data => o_data
        );

    clk_process : process
    begin
        while true loop
            clk <= '0';
            wait for clk_period / 2;
            clk <= '1';
            wait for clk_period / 2;
        end loop;
    end process clk_process;
```

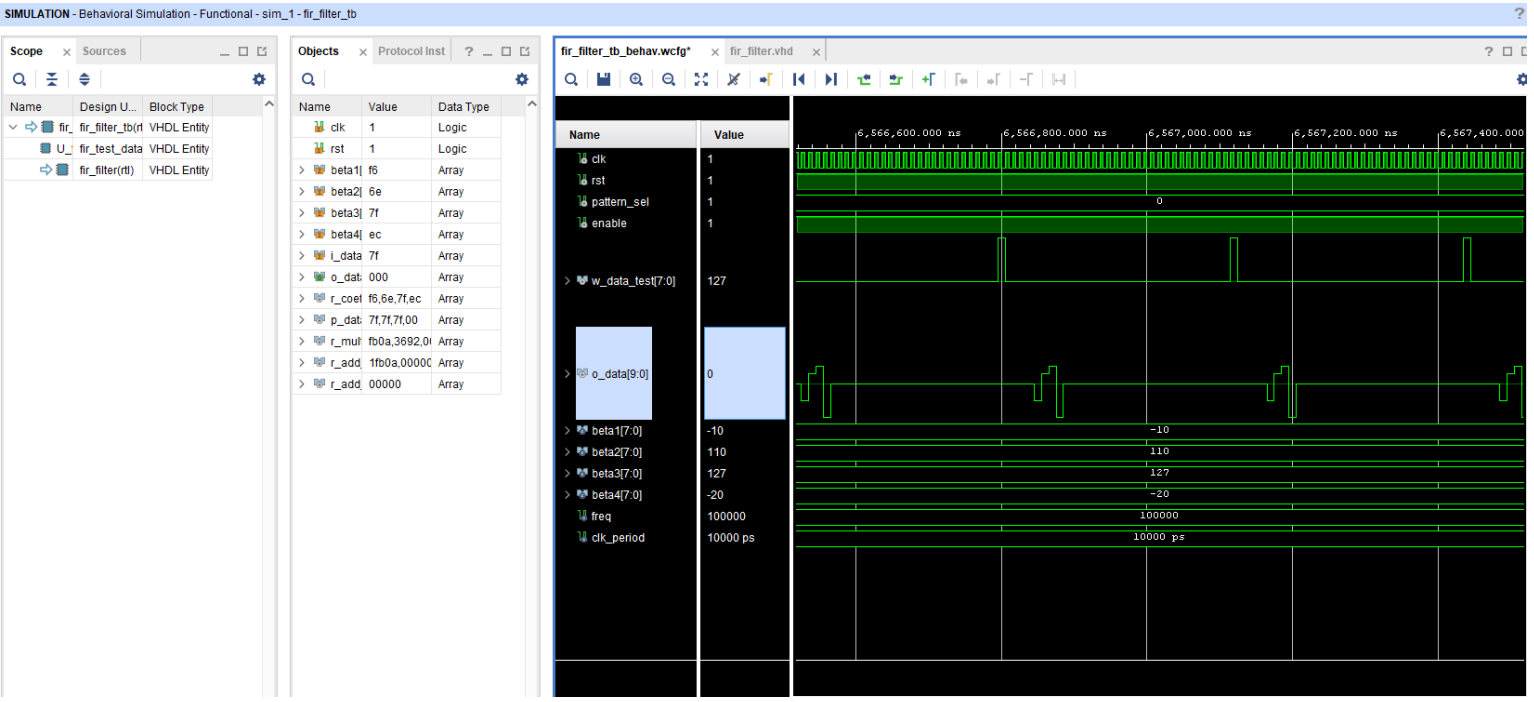
```
--Proceso de generacion de estimulos
--=============================================
process is

    variable v_input_line : line;
    variable v_pattern_sel : integer;
    variable v_file_status : file_open_status;

begin

   -- Abrir el archivo de entrada
    file_open(v_file_status, file_input, "../../../../input.txt", read_mode);

    -- Verificar si el archivo se abrió correctamente
    if v_file_status /= open_ok then
        report "Error"
            severity error;
        wait;
    end if;

    -- Leer el contenido del archivo línea por línea
    while not endfile(file_input) loop
        readline(file_input, v_input_line);
        read(v_input_line, v_pattern_sel);
        pattern_sel <= v_pattern_sel;
        enable <= '1';  -- Habilitar el generador de datos
        wait for 1 ms;  -- Esperar tiempo suficiente para observar la salida
        enable <= '0';  -- Deshabilitar el generador de datos
    end loop;

    -- Cerrar el archivo de entrada
    file_close(file_input);

    -- Secuencia de reset
    wait until clk'event and clk = '1';
    wait until clk'event and clk = '1';
```



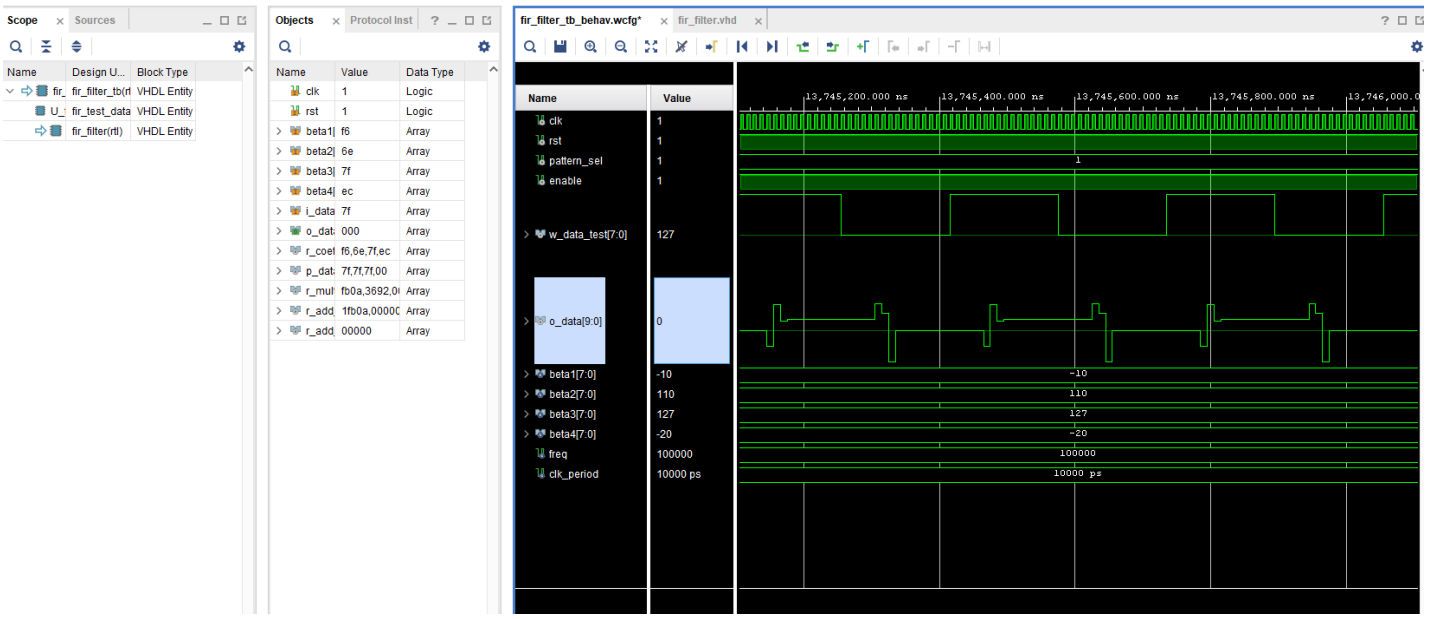
En el testbench, instanciamos nuestro filtro con el generador de señales, como cualquier otro componente en un testbench, tenemos que declarar su arquitectura interna a nivel de entradas y salidas, y, como ya hemos hecho en otras prácticas. Se han de declarar las variables de señal, ya que no podemos modificar directamente las entradas de un componente instanciado, pero estas señales sí podrán ser modificadas de manera externa, a través de un testbench o estimuladores físicos, como generadores de señal botones etc… Se ha optado por introducir los diferentes modos de actuación del filtro en un archivo txt, cada numero correspondiendo a un modo diferente siendo una función delta o impulso,1 siendo una función escalón, y 2 siendo una señal senoidal.

SIMULACION

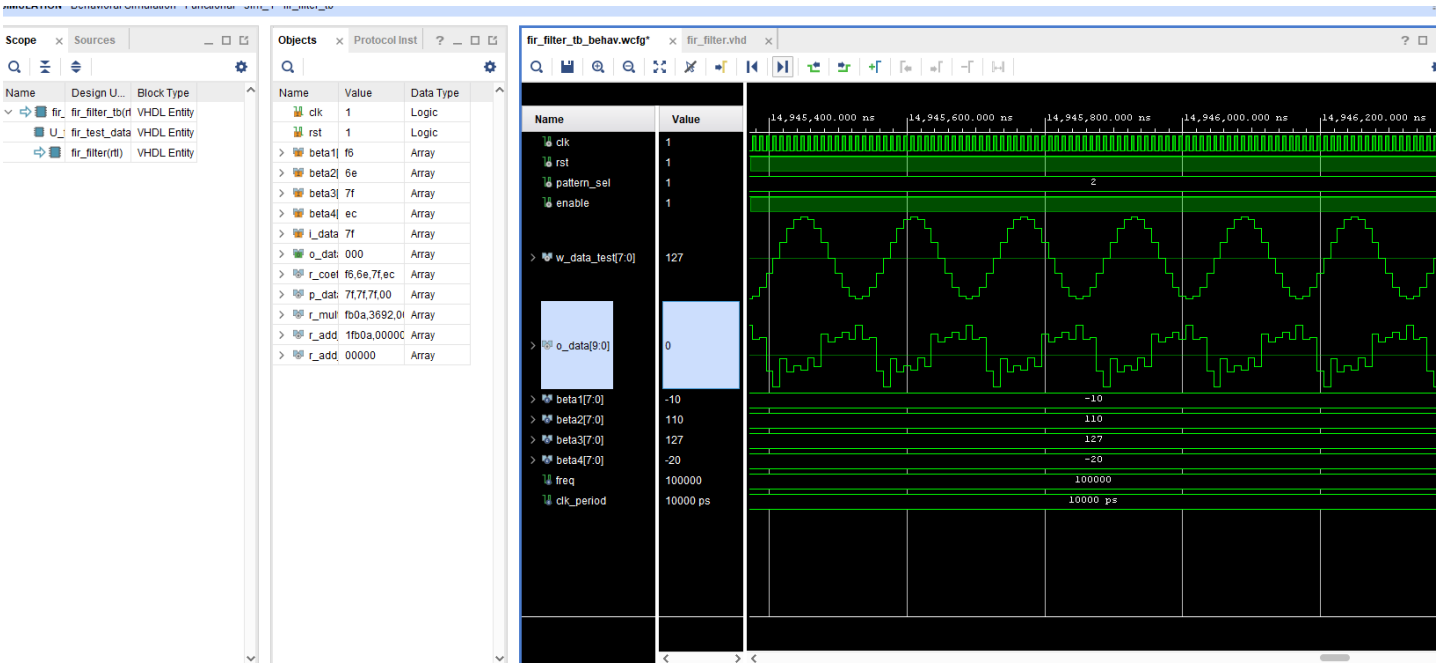
Se ha simulado el funcionamiento del filtro durante 1 segundo, para dar tiempo a los distintos modos para entrar en funcionamiento



En primero lugar, tenemos la respuesta del filtro al “impulso”, es cierto que la señal es algo mas “ancha” que un impulso unitario, pero se ve representado como un pulso único de longitud de onda muy estrecho, podemos observar como existe cierta latencia en el filtro, ya que la respuesta de este no es inmediata al impulso, pero estamos frente a una latencia de 8 pulsos de reloj, que teniendo en cuenta que nuestro reloj simulado es de 100 khz, esta latencia sería de 0.04 milisegundos



En este segundo caso, tenemos la respuesta al escalón, donde se le suministra un pulso cuadrado de varios ciclos de duración, en concreto 16 ciclos de reloj, también se observa esa respuesta a dicho escalón, y como el filtro deja pasar una banda distinta a cuando se le suministra un impulso.



Por último, tenemos la respuesta a una señal senoidal o sinusoide, aquí se puede apreciar mejor como el filtro deja pasar de manera activa ciertos valores de manera dinámica de dicha señal de entrada, en concreto nuestro filtro está dejando pasar las señales de frecuencia más baja en nuestra señal senoidal, como se puede observar en la imagen