FILTRO DIGITAL FIR

Júlia Pessoa Souza

Teoria de Eletrônica Digital-Turma C
Faculdade Gama - Universidade de Brasília
15/0133294
Projeto Final

João Pedro Vergara Coneglian

Teoria de Eletrônica Digital-Turma C
Faculdade Gama - Universidade de Brasília
14/0145974
Laboratório 3 Parte I

RESUMO

O projeto consistiu em criar um filtro digital utilizando a linguagem VHDL. O filtro foi feito no programa VIVADO e posteriormente implementado na placa Basys3. Foi feito um filtro de 4 *taps*, com os coeficientes e as entradas sendo salvos na memória ROM. A implementação mostrou os resultados da somatória do filtro no display de 7 segmentos da placa; display que funcionava em uma frequência de 1Hz.

1. INTRODUÇÃO

Um filtro digital FIR (Finite Impulse Response, ou resposta ao impulso finita) atua sobre uma sequência de dados digitais provenientes de sinais de entrada como rádio, áudio e vídeo (sinais encontrados em vários dispositivos eletrônicos, como celulares, aparelhos de som, monitores cardíacos, etc.). Ele funciona removendo características particulares do sinal de entrada e produzindo um novo sinal de saída que não possui tais características.
Para se fazer um filtro, necessita-se de uma expressão matemática: uma somatória. Cada um dos seus termos é chamado de *tap*, que é um coeficiente multiplicado por uma entrada.

2. PROJETO

Para fazer o filtro digital, foi necessário fazer alguns códigos no programa VIVADO e juntá-los. O primeiro código foi o divisor de clock, que já foi utilizado em alguns experimentos, com a mudança de ser apenas 1Hz. Depois foi feito o código do filtro, utilizando somadores para somar os termos e multiplicadores para multiplicar os coeficientes pelas entradas, utilizamos como componente um flip-flop tipo D. No mesmo código definiu-se quais seriam os coeficientes na memória ROM. Foi feita uma máquina de estados para armazenar as entradas que seriam utilizadas na memória ROM e fazer elas passarem para o próximo estado automaticamente. Foram definidas 16 entradas, assim, são 16 saídas. As saídas devem ser mostradas no display de 7 segmentos na frequência mencionada, em números decimais. Por isso foi necessário fazer um conversor de binário para BCD, que transformou o número inteiro binário em sua centena, dezena e unidade, e outro conversor de BCD para 7 segmentos, que montou o display de acordo com o número que seria mostrado. Também foi feito um multiplexador para selecionar o anodo que seria utilizado para mostrar a centena, dezena e unidade. E, foi feito um contador para a variável seletora desse anodo, com outro divisor de clock para as dezenas, centenas e unidades poderem aparecer ao mesmo tempo para o observador, mudamos para 100Hz. .Enfim, foi feito o toplevel para juntar os códigos construidos.

2.1. Equações

O filtro pode ser descrito pela equação (1), na qual bk são os coeficientes, X são as entradas e Y é a saída. Cada termo corresponde a 1 tap. O filtro feito possui 4 taps, com os coeficientes e as entradas já definidos.

$$y[n] = b_0x[n] + b_1x[n-1] + \cdots + b_{N-1}x[n-(N-1)] + b_Nx[n-N] = \sum_{k=0}^{N} b_k \cdot x[n-k].$$

(1)

2.2. Figuras e Tabelas

COEFICIENTE	BINÁRIO	DECIMAL
H0	00000010	2
H1	00000001	1
H2	00000100	4
H3	00000011	3

Tabela 1: Coeficientes utilizados.

X binário	X decimal	Y decimal
0000	0	0
0001	1	10
0010	2	20
0011	3	30
0100	4	40
0101	5	50
0110	6	60
0111	7	70
1000	8	80
1001	9	90
1010	10	100
1011	11	110
1100	12	120
1101	13	130
1110	14	140
1111	15	150

Tabela 2: entradas e saídas

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

entity DFF is
  port(
     Q : out signed(11 downto 0);
     Clk :in std_logic;
     D :in  signed(11 downto 0)
  );
end DFF;

architecture Behavioral of DFF is

signal qt : signed(11 downto 0) := (others => '0');

begin

Q <= qt;

process(Clk)
begin
 if ( rising_edge(Clk) ) then
   qt <= D;
 end if;
end process;

end Behavioral;
```

Figura 1: Flip Flop tipo D

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
entity fir_4tap is
port(   Clk : in std_logic;
        X: in std_logic_vector(3 downto 0);
       Y :out std_logic_vector(11 downto 0)
        );
end fir_4tap;

architecture Behavioral of fir_4tap is
signal address1:STD_LOGIC_vector (3 downto 0);
signal Yout :signed(11 downto 0);
 type mem is array (15 downto 0) of signed(3 downto 0);
constant entradas : mem :=  (
 0  => "0000",
 1  => "0001",
 2  => "0010",
 3  => "0011",
 4  => "0100",
 5  => "0101",
 6  => "0110",
 7  => "0111",
 8  => "1000",
 9  => "1001",
 10 => "1010",
 11 => "1011",
 12 => "1100",
 13 => "1101",
 14 => "1110",
 15 => "1111");

type mem2 is array (3 downto 0) of signed(7 downto 0);
constant coeficientes : mem2 := (0 => "00000010",
1 => "00000001",
2 => "00000100",
3 => "00000011");

signal Xin: signed(3 downto 0);
component DFF is
   port(
      Q : out signed(11 downto 0);
      Clk :in std_logic;
      D :in  signed(11 downto 0)
   );
end component;
```

Figura 2: Memória ROM.

```
signal H0,H1,H2,H3 : signed(7 downto 0) := (others => '0');
signal MCM0,MCM1,MCM2,MCM3: signed(11 downto 0) := (others => '0');
signal Q1,Q2,Q3 : signed(11 downto 0) := (others => '0');
signal add_out1,add_out2,add_out3 : signed(11 downto 0) := (others => '0');

type tipo_estado is (E0, E1, E2, E3, E4, E5, E6, E7, E8, E9, E10, E11, E12, E13, E14, E15);
signal estado, prox_estado: tipo_estado;


begin
Xin <= signed(X);
--Coeficientes do filtro.
H0 <= coeficientes(0);
H1 <= coeficientes(1);
H2 <= coeficientes(2);
H3 <= coeficientes(3);


--Constantes de multiplicação.
MCM3 <= H3*Xin;
MCM2 <= H2*Xin;
MCM1 <= H1*Xin;
MCM0 <= H0*Xin;

--Somadores.
add_out1 <= Q1 + MCM2;
add_out2 <= Q2 + MCM1;
add_out3 <= Q3 + MCM0;

--flipflops.
dff1 : DFF port map(Q1,Clk,MCM3);
dff2 : DFF port map(Q2,Clk,add_out1);
dff3 : DFF port map(Q3,Clk,add_out2);

process(Clk)
begin
    if(rising_edge(Clk)) then
        Yout <= add_out3;
        Y<= std_logic_vector(Yout);
    end if;
end process;

mudaestado: process (clk)
begin
if clk'event and clk = '1' then
estado <= prox_estado;
end if;
end process;
```

Figura 3 : Segunda parte do filtro

```
--ROM
 process (address1)
 begin
   case address1 is
     when "0000" => Xin <= "0000";
     when "0001" => Xin <= entradas(1);
     when "0010" => Xin <= entradas(2);
     when "0011" => Xin <= entradas(3);
     when "0100" => Xin <= entradas(4);
     when "0101" => Xin <= entradas(5);
     when "0110" => Xin <= entradas(6);
     when "0111" => Xin <= entradas(7);
     when "1000" => Xin <= entradas(8);
     when "1001" => Xin <= entradas(9);
     when "1010" => Xin <= entradas(10);
     when "1011" => Xin <= entradas(11);
     when "1100" => Xin <= entradas(12);
     when "1101" => Xin <= entradas(13);
     when "1110" => Xin <= entradas(14);
     when "1111" => Xin <= entradas(15);
     when others => Xin <= "0000";
   end case;
end process;
--Maq de estados
transição: process (estado) begin
        case estado is
          when E0 => prox_estado <= E1;
          when E1 => prox_estado <= E2;
          when E2 => prox_estado <= E3;
          when E3 => prox_estado <= E4;
          when E4 => prox_estado <= E5;
          when E5 => prox_estado <= E6;
          when E6 => prox_estado <= E7;
          when E7 => prox_estado <= E8;
          when E8 => prox_estado <= E9;
          when E9 => prox_estado <= E10;
          when E10 => prox_estado <= E11;
          when E11 => prox_estado <= E12;
          when E12 => prox_estado <= E13;
          when E13 => prox_estado <= E14;
          when E14 => prox_estado <= E15;
          when E15 => prox_estado <= E0;
        end case;
end process;
```

Figura 4: Memória ROM das entradas.

```
saidas: process (estado) begin
case estado is
    when E0 => address1 <= "0000";
    when E1 => address1 <= "0001";
    when E2 => address1 <= "0010";
    when E3 => address1 <= "0011";
    when E4 => address1 <= "0100";
    when E5 => address1 <= "0101";
    when E6 => address1 <= "0110";
    when E7 => address1 <= "0111";
    when E8 => address1 <= "1000";
    when E9 => address1 <= "1001";
    when E10 => address1 <= "1010";
    when E11 => address1 <= "1011";
    when E12 => address1 <= "1100";
    when E13 => address1 <= "1101";
    when E14 => address1 <= "1110";
    when E15 => address1 <= "1111";
end case;
end process;

end Behavioral;
```

Figura 5: Final do filtro.

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

entity clock is
    Port ( clk : in STD_LOGIC;
           x : out STD_LOGIC);
end clock;

architecture Behavioral of clock is

signal clk_div: STD_LOGIC:='0';
signal count: INTEGER:=0;

begin

process (clk, clk_div) begin
if rising_edge(clk) then
    if count = 50000000 then -- (freq placa/ freq desejada)/2
    count <= 0;
    clk_div <= not clk_div;
    else
    count <= count + 1;
    end if;
end if;
x <= clk_div;
end process;
end Behavioral;
```

Figura 6: Clock dividido para 1Hz.

```
bcds_reg(15 downto 12) <= bcds(15 downto 12) + 3 when bcds(15 downto 12) > 4 else
                          bcds(15 downto 12);
bcds_reg(11 downto 8) <= bcds(11 downto 8) + 3 when bcds(11 downto 8) > 4 else
                         bcds(11 downto 8);
bcds_reg(7 downto 4) <= bcds(7 downto 4) + 3 when bcds(7 downto 4) > 4 else
                        bcds(7 downto 4);
bcds_reg(3 downto 0) <= bcds(3 downto 0) + 3 when bcds(3 downto 0) > 4 else
                        bcds(3 downto 0);

bcds_out_reg_next <= bcds when state = done else
                     bcds_out_reg;

bcd3 <= bcds_out_reg(15 downto 12);
bcd2 <= bcds_out_reg(11 downto 8);
bcd1 <= bcds_out_reg(7 downto 4);
bcd0 <= bcds_out_reg(3 downto 0);
```

Figura 7: Conversor para BCD.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux is
    Port ( an : out STD_LOGIC_vector(3 downto 0);
           sel: in STD_LOGIC_vector (1 downto 0);
           Cseg: in STD_LOGIC_vector (6 downto 0);
           Dseg: in STD_LOGIC_vector (6 downto 0);
           Useg: in STD_LOGIC_vector (6 downto 0);
           Seg: out STD_LOGIC_vector (6 downto 0));
end mux;

architecture Behavioral of mux is

begin
with sel select
an <= "1110" when "00",
      "1101" when "01",
      "1011" when "10",
      "1111" when others;

with sel select
Seg <= Cseg when "10",
       Dseg when "01",
       Useg when others;

end Behavioral;
```

Figura 8: Multiplexador.

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

entity contadorSEL is
    Port ( clock : in STD_LOGIC;
           Saida : out STD_LOGIC_vector (1 downto 0));
end contadorSEL;

architecture Behavioral of contadorSEL is
type tipo_estado is (E0, E1, E2);
signal estado, prox_estado: tipo_estado;
signal x: std_logic;

component clockANODO is
    Port ( clk : in STD_LOGIC;
           x : out STD_LOGIC);
end component;

begin
mudaestado: process (clock)
begin
if x'event and x = '1' then
estado <= prox_estado;
end if;
end process;

transição1: process (estado) begin
        case estado is
          when E0 => prox_estado <= E1;
          when E1 => prox_estado <= E2;
          when E2 => prox_estado <= E0;
        end case;
end process;


saidas: process (estado) begin
case estado is -- indica qual é cada estado
    when E0 => saida <= "00";
    when E1 => saida <= "01";
    when E2 => saida <= "10";
end case;
end process;
Clk: clockANODO port map(clk => clock, x => x);
end Behavioral;
```

Figura 9: Contador da Seletora.

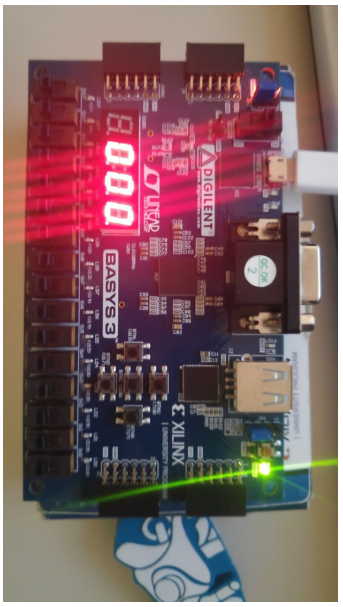

Figura 10: Placa Basys3 implementada.

3. RESULTADOS

Os resultados foram positivos. A Tabela 1 mostra os coeficientes do filtro, que já estão definidos para todas as somas feitas. A tabela 2 mostra as entradas utilizadas para cada somatória e suas respectivas saídas. A figura 1 mostra o flip flop tipo D que armazena um dado na memória, o qual posteriormente, vai ser usado para armazenar dados do filtro. As figuras de 2 a 5 mostram o código do filtro. Primeiramente as entradas e os coeficientes sendo gravados em memória ROM, e o componente do flip flop. Depois, são mostradas as multiplicações e as somas, as quais são guardadas no flip flop, e, o último resultado é mandado para a saída tipo signed, que depois é convertida em vetor. As Figuras 4 e 5 mostram as entradas sendo armazenadas em endereços da memória ROM, e esses endereços ficam mudando de estado para a saída poder ficar mostrando os resultados de cada entrada. A Figura 6 mostra a divisão de clock que foi utilizada para fazer 1Hz, mudando o número do contador também foi modificado para ter 100Hz para o anodo. Na Figura 7 mostra-se a parte mais importante do conversor de binário para BCD, que depois é passado para 7 segmentos como já foi feito em sala de aula. Neste conversor para BCD, bcd3 significa milha, bcd2 são as centenas, bcd1 são as dezenas e bcd0 são as unidades. A Figura 8 mostra o multiplexador. A Figura 9 mostra o contador feito para a variável seletora do anodo, na qual foi utilizado o clock de 100Hz. Na Figura 10 mostra-se a implementação da placa Basys3 e um dos resultados de y no display.

4. DISCUSSÃO E CONCLUSÕES

Foi observado na implementação que o display mostrava os resultados da saída Y que ia mudando para cada entrada com um tempo definido. O projeto obteve sucesso na implementação. Com as entradas e os coeficientes definidos, o anodo do milhar ficou sempre apagado porque a saída não chegava a 1000. Por isso, não foi necessário utilizá-lo. Foram omitidos os códigos do clock de 100Hz e do conversor para 7 segmentos, porque estes já foram utilizados em outros experimentos. Foi possível observar que os coeficientes e as entradas foram gravados na memória corretamente pois o programa não demonstrou nenhum erro ou aviso.

5. REFERÊNCIAS

[1] I. Idoeta e I. Valeije, "Elementos de Eletrônica Digital", Érica Ltda, São Paulo, 2008.

[2] R. Tocci, N.Widmer e G. Moss, "Sistemas Digitais: Princípios e aplicações" Pearson.

[3] G. Bezerra, "Apostila Prática de Eletrônica Digital I", 2017

[4] F. Vahid, "Sistemas Digitais: Projeto, otimização e HDLs", Artmed, Porto Alegre, 2008