Relatório do 13 Laborátorio

Discentes:
Erika Costa Alves

Circuitos Digitais, 2019.2

Docente:
Samaherni Morais Dias

Departamento de Engenharia da Computação e Automação
Universidade Federal do Rio Grande do Norte
Natal-RN, Novembro de 2019

1 Introdução

Este relatorio visa em explicar como foi implementado e como foi as conclusões finais referentes á atividade treze de laborátorio. Para está atividade, foi-se implementado um **filtro FIR** - *Finite Impulse Response*-. O filtro tem como entrada um vetor de 4 Bits chamado **Y**, e ao longo de três pulsos de clock essa entrada Y vai passado por três registradores diferentes, nos quais cada registrador é multiplicado por uma constante **C** - *0, 1, 2..* -, e no fim cada valor multiplicado será somado para formar a saída final, chamada de **F** que possui 10 Bits. é possivel vizualisar melhor o Bloco Operacional na **Figura 1**.

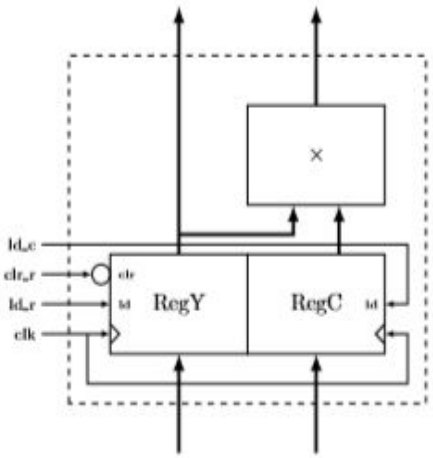

Figura 1: Bloco R × C

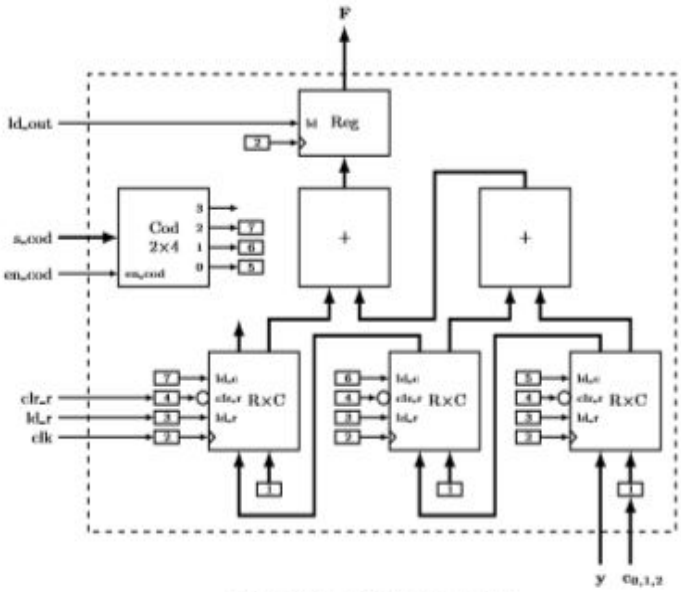

Figura 2: Bloco FIR

Figure 1: Circuito á ser Projetado.

Para uma melhor noção, é possivel descrever o processo á cima com uma equação matematica a um filtro FIR de 3 pulsos de clock. Veja a equação a baixo.

$$F = Y(k)C0 + Y(k-1)C1 + Y(k-2)C2 \quad (1)$$

2 Desenvolvimento

Para implementar o filtro FIR, foi-se dividido o problema em duas partes, a primeira parte consiste no *Bloco Operacional*, que é exatamente a parte que está descrita na Figura 1. E a segunda parte é o *Bloco Combinacional*, ou seja, uma maquina de estados criada para fazer o funcionamnto do filtro FIR.

2.1 Bloco Operacional

Ao implementar o bloco operacional foi necessario diversos blocos basicos, como por exemplo um registrador com carga paralela, um somador, um decodificador, e por fim um multiplicador.

- **Registrador de Carga Paralela** : para este problema foi utilizado um registrador de carga paralela, pois, toda vez que *ld_tot, ld_r, ld_c* estivessem em alto, o valor seria carregado via preset.
- **Multiplicador** : aqui o multiplicador foi utilizado para multiplicar os valores de Y com C.
- **Somadores** : somar os resultados da multiplicação de Y com C, e em seguida passar para o registrador total.
- **Decodificador** : serve para decodificar o valor de saida o decodificador. No caso desse projeto foi implementado um decodificador 2x4, ou seja, dependendo do bit de entrada haverá uma saida respectiva. Por exemplo, caso o vetor de entrada seja 00 a saída será 0001, caso a entrada seja 01 a saída será 0010.

2.2 Bloco Combinacional

Neste projeto não haveria necessidade de se fazer uma maquina de estados, ele poderia funcionar perfeitamente apenas com o bloco operacional. Contudo com fins de curiosidade foi-se implementado uma maquina de estados para tal. Na **Figura 2** é possivel ver como foi projetado.

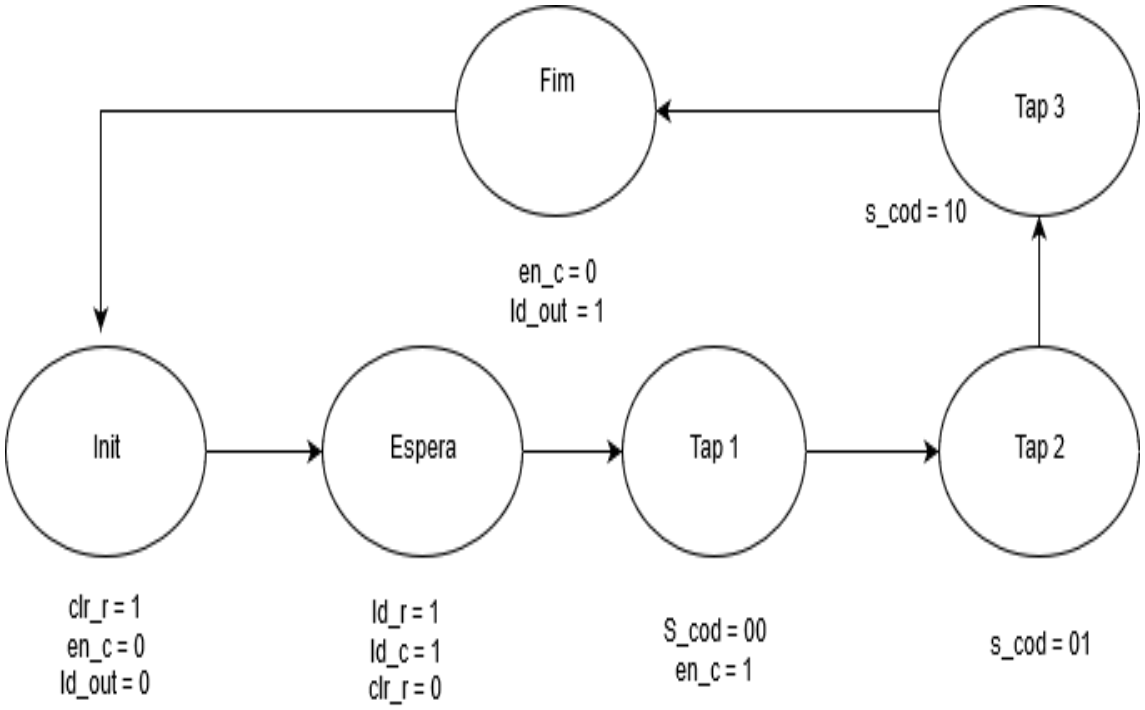

Figure 2: Maquina de Estados.

3 Conclusão

A simulação final do projeto ocorreu quase como previsto, **Figura 3**. Entretanto estamos trabalhando com um filtro FIR, logo se era necessario que o valro final se mantesse. Contudo, no projeto implementado não foi isso que foi feito. Na maquina de estados criada, depois que chega-se ao fim a maquina retorna ao seu estado inicial, no qual o valor dos registradores são resetados.

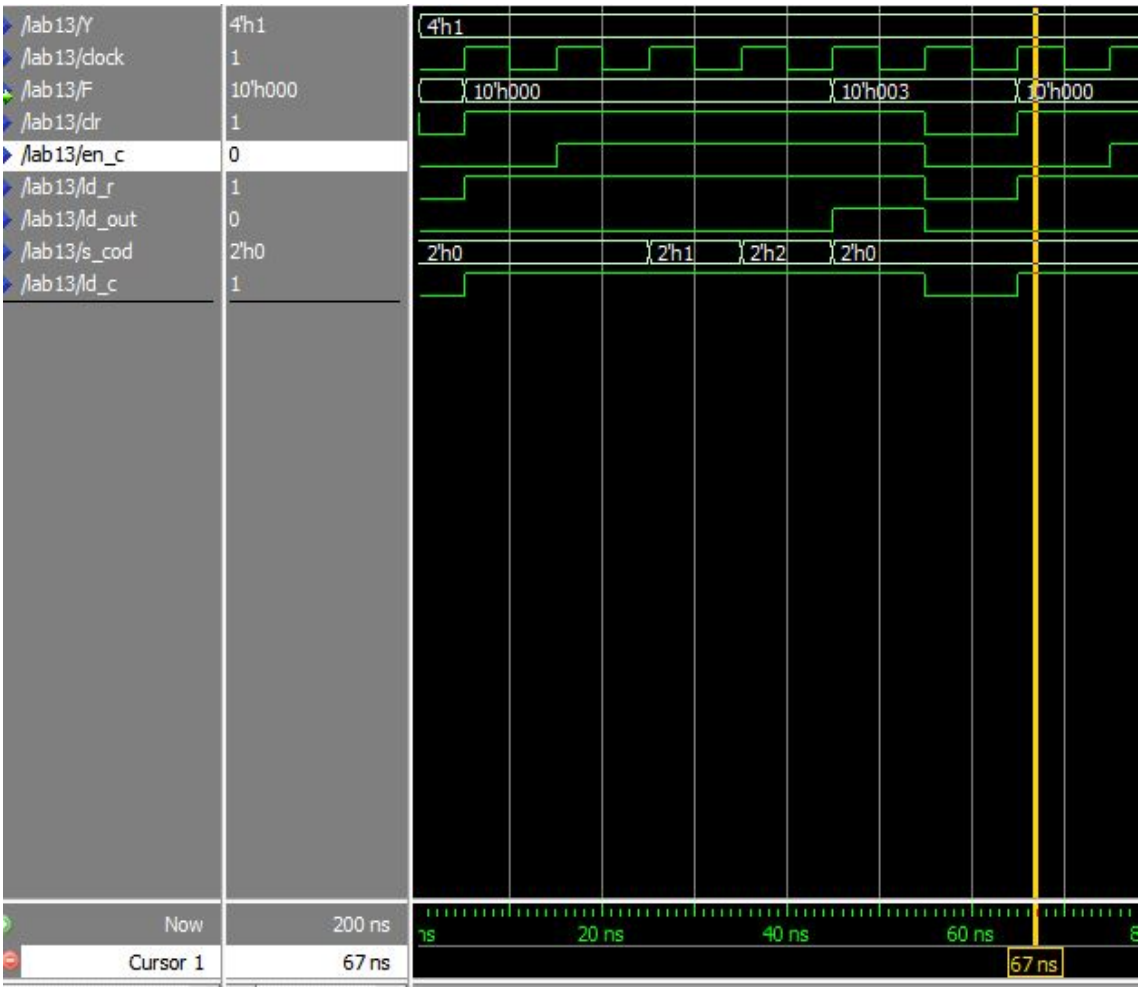

Figure 3: Simulação.

4 Apendice A : VHDL

Neste apendice possui as estruturas gerais implementadas em VHDl. Tais estruturas são o decodificador, bloco opercional e bloco combinacional.

```
entity encoder is
port(   enable, A, B    : in bit;
        Y3,Y2,Y1,Y0     : out bit
);
end encoder;

architecture ckt of encoder is

begin

        Y3 <= enable and A and B;
        Y2 <= enable and A and (not B);
        Y1 <= enable and (not A) and B;
        Y0 <= enable and (not A) and (not B);

end ckt;
```

Figure 4: Decodificador.

```
entity BlocoOperacional is
port(   Y       : in bit_vector(3 downto 0);
        clock   : in bit;
        clr_r   : in bit;
        en_c    : in bit;
        ld_r    : in bit;
        ld_c    : in bit;
        s_cod   : in bit_vector(1 downto 0);
        ld_out  : in bit;
        F       : out bit_vector(9 downto 0)
);
end BlocoOperacional;

architecture ckt of BlocoOperacional is
        -- Componenetes --
component blocoRxC is
port(   Y       : in bit_vector(3 downto 0);
        C       : in bit_vector(3 downto 0);
        clk,clr : in bit;
        ldc,ldr : in bit;
        Ys      : out bit_vector(3 downto 0);
        YxC     : out bit_vector(9 downto 0)
);
end component;

component Adder10Bits is
port(   A       : in bit_vector(9 downto 0);
        B       : in bit_vector(9 downto 0);
        Ci      : in bit;
        Co      : out bit;
        S       : out bit_vector(9 downto 0)
);
end component;
```

Figure 5: Bloco Operacional.

```
component Register10Bits is
port(   A       : in bit_vector(9 downto 0);
        I       : in bit_vector(9 downto 0);
        enable  : in bit;
        clock   : in bit;
        clr     : in bit;
        B       : out bit_vector(9 downto 0)
);
end component;

        -- Sinais --

        signal C0, C1, C2       : bit_vector(3 downto 0);
        signal Y0, Y1, Y2       : bit_vector(3 downto 0);
        signal Yxc0, Yxc1, Yxc2 : bit_vector(9 downto 0);
        signal co0, co1         : bit;
        signal Sum0, sum1       : bit_vector(9 downto 0);
        signal Af               : bit_vector(9 downto 0);

begin

        -- Nossas variaveis C0,1,2 --

        C0(3 downto 0) <= "0001";
        C1(3 downto 0) <= "0001";
        C2(3 downto 0) <= "0001";
```

Figure 6: Bloco Operacional.

```
entity BlocoCombinacional is
port(   clock   : in bit;
        clr_r   : out bit;
        en_c    : out bit;
        ld_r    : out bit;
        ld_c    : out bit;
        s_cod   : out bit_vector(1 downto 0);
        ld_out  : out bit
);
end BlocoCombinacional;

architecture ckt of BlocoCombinacional is
        -- Componentes --
component FFD is
port( clk, D, P, C: in bit;
        q: out bit
);
end component;

        -- Sinais --

        signal nexts    : bit_vector(2 downto 0);
        signal s        : bit_vector(2 downto 0);
        signal nclr_r   : bit;

begin
```

Figure 7: Bloco Combinacional.

```
        -- Sinais --

        signal nexts    : bit_vector(2 downto 0);
        signal s        : bit_vector(2 downto 0);
        signal nclr_r   : bit;

begin

        nexts(0) <= ((not s(0)) and (not s(2))) or (s(2) and
        nexts(1) <= ((not s(2)) and (not s(1)) and s(0)) or (
        nexts(2) <= ((not s(2)) and s(1) and s(0)) or (s(2) a

        nclr_r  <= (not s(2)) and (not s(1)) and (not s(0));
        en_c    <= s(2) or s(1);
        ld_r    <= s(2) or s(1) or s(0);
        ld_c    <= s(2) or s(1) or s(0);
        s_cod(0)<= ((not s(2)) and s(1) and s(0));
        s_cod(1)<= (s(2) and (not s(1)) and (not s(0)));
        ld_out  <= s(2) and (not s(1)) and s(0);

        clr_r <= not nclr_r;

        chinela0 : FFD
        port map(clock, nexts(0), '1', '1', s(0));

        chinela1 : FFD
        port map(clock, nexts(1), '1', '1', s(1));

        chinela2 : FFD
        port map(clock, nexts(2), '1', '1', s(2));

end ckt;
```

Figure 8: Bloco Combinacional.