Universidade Federal do Rio Grande do Norte
Centro de Tecnologia - CT
Curso de Ciências e Tecnologia

Filtro FIR

ELE2715 - Laboratório 12

Isaac de Lyra Junior

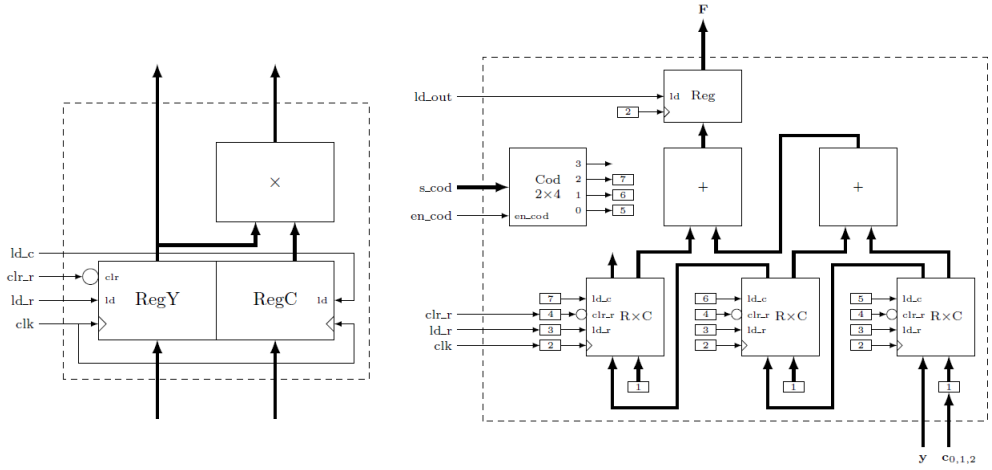
Natal
19 de abril de 2021

Sumário

1 Desenvolvimento

A problemática do laboratório consiste na implementação de um circuito lógico para um filtro FIR com 3 taps. O circuito possui um vetor de entrada (y) de 4 bits com os valores a serem filtrados, uma outra entrada ($c_{0,1,2}$) de 4 bits para definir os valores das constantes c_0, c_1, c_2 do filtro, uma entrada clr_r=0 para limpar todos os registradores RegY, uma entrada ld_r=1 para permitir atualizar o valor no registrador RegY, uma entrada en_cod=1 para permitir atualizar o valor em um dos registradores RegC, uma entrada s_cod de 2 bits para definir qual dos registradores RegC será atualizado, uma entrada ld_out=1 para permitir atualizar o registrador de saída e, por fim, uma saída (**F**) de 10 bits para apresentar o valor filtrado. A simulação deve iniciar com a definição dos valores das constantes $c_{0,1,2}$ para em seguida serem introduzidos os valores da entrada y. Na Figura 1 é possível ver o diagrama de blocos do circuito operacional do filtro FIR com 3 taps.

Figura 1: Circuito a ser projetado



Fonte: Dados da atividade (2021).

Para a resolução do problema foi utilizado o método mais comum para projetos de processadores, denominado de projeto em nível de transferência entre registradores (Register Transfer Level), que consiste na especificação dos registradores de um circuito, descrição das possíveis transferências e operações a serem realizadas com os dados de entrada, de saída e dos registradores e definição do controle que especifica quando transferir e operar com os dados (VAHID, 2009).

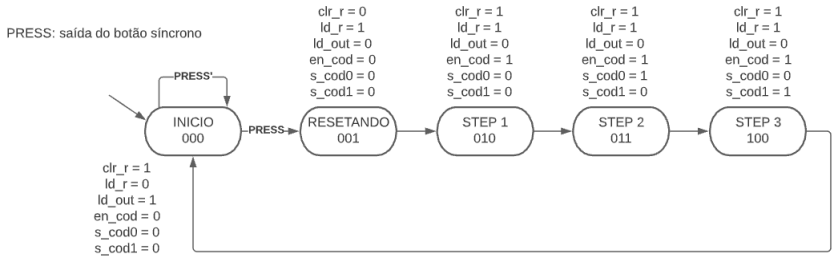
Na prática, o Projeto RTL é realizando utilizando uma gama variedade de métodos, mas é bastante útil a definição de um método geral. Tal método, segundo Vahid (2009), consiste em uma sequência de passos, onde o primeiro passo é capturar o comportamento desejado, e os demais fazem a conversão deste comportamento em um circuito.

1.1 Máquina de estados finitos do bloco de controle

No projeto RTL, é comum que, após a captura do comportamento desejado de uma máquina, seja pensado o bloco operacional, porém, como este bloco e seus componentes, bem como as suas conexões com o bloco de controle já foi dado na atividade, foi possível adiantar para a implementação de máquina de estados finitos do bloco de controle.

O modelo de máquina desta solução foi do tipo Moore, assim, para a implementação da máquina de estados finitos do bloco de controle foi pensado primeiramente no diagrama de estados de uma máquina de estados de baixo nível, onde é definido as saídas do bloco de controle para cada estado de nossa máquina, o resultado é visto na Figura 2.

Figura 2: Máquina de estados de baixo nível



Fonte: Elaborado pelo autor (2021).

Seguindo adiante, o próximo passo foi converter o diagrama visto na Figura 2 em uma tabela de transição de estados, isso é importante para que possamos definir a lógica combinacional do controlador para que ele possa gerar as saídas com relação as entradas e também ao estado atual. A tabela de transição de estados resultante é vista na Figura 3.

Figura 3: Tabela de transição de estados

ESTADOS	ENTRADAS				ESTADOS	SAÍDAS								
	Q2	Q1	Q0	PRESS		D2	D1	D0	clr_r	ld_r	ld_out	en_cod	s_cod0	s_cod1
INICIO	0	0	0	0	INICIO	0	0	0	1	0	1	0	0	0
	0	0	0	1	RESETANDO	0	0	1	1	0	1	0	0	0
RESETANDO	0	0	1	X	STEP 1	0	1	0	0	1	0	0	0	0
STEP 1	0	1	0	X	STEP 2	0	1	1	1	1	0	1	0	0
STEP 2	0	1	1	X	STEP 3	1	0	0	1	1	0	1	1	0
STEP 3	1	0	0	X	INICIO	0	0	0	1	1	0	1	0	1

Fonte: Elaborado pelo autor (2021).

Com a tabela de transição de estados foi possível definir todas as saídas de nosso bloco de controle, utilizando o conceito de mapas de Karnaugh para encontrar a lógica combinacional de cada saída do controlador. As equações booleanas encontradas podem ser vistas na Figura 4.

Figura 4: Lógica combinacional das saídas do bloco de controle

SAÍDAS	LÓGICA COMBINACIONAL
D2	Q2' Q1 Q0
D1	Q2' Q1' Q0 + Q2' Q1 Q0'
D0	Q2' Q0' PRESS + Q2' Q1 Q0'
clr_r	Q1' Q0' + Q2' Q1
ld_r	Q2' Q0 + Q2' Q1 + Q2 Q1' Q0'
ld_out	Q2' Q1' Q0'
en_cod	Q2' Q1 + Q2 Q1' Q0'
s_cod0	Q2' Q1 Q0
s_cod1	Q2 Q1' Q0'

Fonte: Elaborado pelo autor (2021).

Com a lógica combinacional do bloco de controle definida, foi possível realizar a implementação que será explicada na seção de Resultados. É importante citar que, por estamos lidando com circuitos sequenciais e clocks variados, a entrada que indica que meu filtro deve processar uma saída (B=1), não pode ser um *pushbutton*, pois geraria uma série de erros caso o *clock* fosse alto demais, isso por que o usuário deveria apertar durante um pulso de *clock* apenas. Para solucionar isto, a entrada citada foi definida na máquina implementada como sendo uma entrada para um botão sincronizado, este botão vai ser responsável por pegar apenas um pulso de *clock*, e mesmo que o usuário permaneça com o botão pressionado, não irá interferir na realização correta do processo de nossa máquina.

2 Resultados

Toda a implementação do filtro FIR foi realizada utilizando os seguintes softwares: Quartus Prime Lite Edition em sua versão 20.1.1 para debugar o código, Visual Studio Code em sua versão 1.55.2 para escrita do código e, por fim, ModelSim em sua versão 2020.1 para simulação do código. O código completo pode ser visto no Anexo A.

2.1 Bloco de controle

O primeiro passo foi implementar a entidade responsável pela lógica combinacional do bloco de controle definida na seção anterior, o resultado desta implementação pode ser visto na Figura 5.

Figura 5: Lógica combinacional do bloco de controle implementada

```
ENTITY LOG_COMB IS
PORT(Q2, Q1, Q0, PRESS: IN BIT;
    D2, D1, D0, clr_r, ld_r, ld_out, en_cod, s_cod0, s_cod1: OUT BIT);
END LOG_COMB;

ARCHITECTURE CKT OF LOG_COMB IS

BEGIN

D2 <= NOT Q2 AND Q1 AND Q0;
D1 <= (NOT Q2 AND NOT Q1 AND Q0) OR ( NOT Q2 AND Q1 AND NOT Q0);
D0 <= (NOT Q2 AND NOT Q0 AND PRESS) OR  (NOT Q2 AND Q1 AND NOT Q0);
clr_r <= (NOT Q1 AND NOT Q0) OR  (NOT Q2 AND Q1);
ld_r <= (NOT Q2 AND Q0) OR  (NOT Q2 AND Q1) OR  (Q2 AND NOT Q1 AND NOT Q0);
ld_out <= NOT Q2 AND NOT Q1 AND NOT Q0;
en_cod <= (NOT Q2 AND Q1) OR  (Q2 AND NOT Q1 AND NOT Q0);
s_cod0 <= NOT Q2 AND Q1 AND Q0;
s_cod1 <= Q2 AND NOT Q1 AND NOT Q0;

END CKT;
```

Fonte: Elaborado pelo autor (2021).

Em seguida, a entidade da lógica combinacional implementada juntamente com a entidade responsável pelo botão sincronizado foram chamadas como componente da entidade denominada *BLOCK_CON*, tal entidade representa o nosso bloco de controle completo. Na Figura 6 é possível ver a declaração da entidade do bloco de controle e suas portas, bem como os componentes que foram chamados na arquitetura da entidade.

Figura 6: Definição da entidade BLOCO_CON e suas componentes

```
ENTITY BLOCK_CON IS
PORT(B, CLK: IN BIT;
    clr_r, ld_r, ld_out, en_cod, s_cod0, s_cod1: OUT BIT);
END BLOCK_CON;

ARCHITECTURE CKT OF BLOCK_CON IS

COMPONENT ffd IS
    port ( clk ,D ,P , C : IN BIT ;
    q : OUT BIT );
END COMPONENT;

COMPONENT LOG_COMB IS
PORT(Q2, Q1, Q0, PRESS: IN BIT;
    D2, D1, D0, clr_r, ld_r, ld_out, en_cod, s_cod0, s_cod1: OUT BIT);
END COMPONENT;

COMPONENT BS IS
PORT(CLK, B: IN BIT;
    PRESS: OUT BIT);
END COMPONENT;
```

Fonte: Elaborado pelo autor (2021).

Com tudo definido, foi possível definir o circuito da entidade, como mostra a Figura 7.

Figura 7: Circuito da entidade BLOCO_CON

```
SIGNAL Q2, Q1, Q0, PRESS, D2, D1, D0: BIT;

BEGIN

DEF_PRESS: BS PORT MAP(CLK, B, PRESS);

LOGIC: LOG_COMB PORT MAP (Q2, Q1, Q0, PRESS, D2, D1, D0, clr_r, ld_r, ld_out, en_cod, s_cod0, s_cod1);

FFD1: ffd PORT MAP(CLK, D2, '1', '1', Q2);
FFD2: ffd PORT MAP(CLK, D1, '1', '1', Q1);
FFD3: ffd PORT MAP(CLK, D0, '1', '1', Q0);

END CKT;
```

Fonte: Elaborado pelo autor (2021).

2.2 Bloco Operacional

O bloco operacional foi implementado seguindo fielmente o projeto dado na atividade, onde é utilizado três blocos RxC, um codificador 2x4 com entrada *en_cod=1* para ser ativado, dois somadores sendo um de 8 bits e outro de 9 bits e um registrador de 10 bits.

2.2.1 Codificador 2x4

O codificador 2x4 será o responsável por selecionar em qual dos blocos RxC será carregado as contantes $c_{0,1,2}$, para isso, este bloco possui uma entrada en_cod, responsável por habilitar a seleção, como também uma entrada de 2 bits denominada de s_cod, responsável por selecionar qual saída deve possuir nível lógico alto quando en_cod=1. A declaração da entidade do codificador 2x4, bem como seu circuito podem ser vistos na Figura 8.

Figura 8: Entidade COD24 e seu circuito

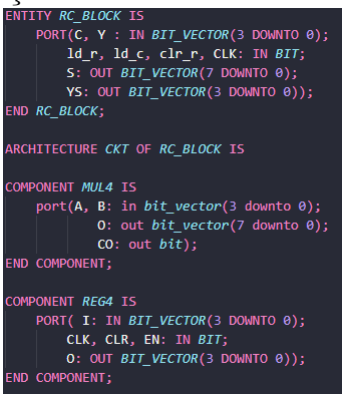
```
ENTITY COD24 IS
    PORT(SC0, SC1, EN: IN BIT;
         S0,S1,S2,S3: OUT BIT);
END COD24;

ARCHITECTURE CKT OF COD24 IS

BEGIN

S0 <= NOT SC0 AND NOT SC1 AND EN;
S1 <= SC0 AND NOT SC1 AND EN;
S2 <= NOT SC0 AND SC1 AND EN;
S3 <= SC0 AND SC1 AND EN;

END CKT;
```

Fonte: Elaborado pelo autor (2021).

2.2.2 Bloco R x C

O elemento principal do bloco operacional é o bloco RxC, ele é responsável por fazer a multiplicação do sinal y com as constantes $c_{0,1,2}$ e também fornecer aos demais blocos o sinal y deslocado. Para implementação deste bloco foi utilizado dois registradores e um multiplicador, sendo todos de 4 bits, como componentes. A definição da entidade do bloco

Figura 9: Definição da entidade RC e suas componentes



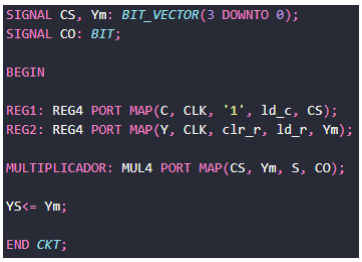
```
ENTITY RC_BLOCK IS
    PORT(C, Y : IN BIT_VECTOR(3 DOWNTO 0);
         ld_r, ld_c, clr_r, CLK: IN BIT;
         S: OUT BIT_VECTOR(7 DOWNTO 0);
         YS: OUT BIT_VECTOR(3 DOWNTO 0));
END RC_BLOCK;

ARCHITECTURE CKT OF RC_BLOCK IS

COMPONENT MUL4 IS
    port(A, B: in bit_vector(3 downto 0);
             O: out bit_vector(7 downto 0);
             CO: out bit);
END COMPONENT;

COMPONENT REG4 IS
    PORT( I: IN BIT_VECTOR(3 DOWNTO 0);
          CLK, CLR, EN: IN BIT;
          O: OUT BIT_VECTOR(3 DOWNTO 0));
END COMPONENT;
```

Fonte: Elaborado pelo autor (2021).

RxC, bem como os componentes chamados na arquitetura da entidade pode ser vista na Figura 9.

Com todos os componentes definidos, foi possível descrever o circuito da entidade do bloco RxC, como mostra a Figura 10.

Figura 10: Circuito da entidade RC



```
SIGNAL CS, Ym: BIT_VECTOR(3 DOWNTO 0);
SIGNAL CO: BIT;

BEGIN

REG1: REG4 PORT MAP(C, CLK, '1', ld_c, CS);
REG2: REG4 PORT MAP(Y, CLK, clr_r, ld_r, Ym);

MULTIPLICADOR: MUL4 PORT MAP(CS, Ym, S, CO);

YS<= Ym;

END CKT;
```

Fonte: Elaborado pelo autor (2021).

2.2.3 Entidade principal do bloco operacional

A entidade principal do bloco operacional foi denominada de *BLOCO_OP*, ela recebe os dados das contantes $c_{0,1,2}$ de 4 bits e também o valor y de 4 bits, além disso, possui as entradas de clock, clr_r, ld_r, ld_out, en_cod, s_cod0 e s_cod1 sendo a maioria delas vindas do bloco de controle. Esta entidade é responsável por gerar o dado de saída **F** do sinal filtrado no filtro FIR. A declaração da entidade e suas componentes podem ser vistos na Figura 11.

Com todos os componentes definidos, foi possível descrever o circuito da entidade do bloco operacional, como é mostrado na Figura 12.

2.3 Filtro FIR

A entidade do filtro FIR foi denominada de FIR, tal entidade recebe os dados do valor y e das contantes $c_{0,1,2}$, ambos de 4 bits, além disso, também recebe a entrada responsável por dar início ao processo, denominada de B e a entrada de clock. Esta entidade só fornece a saída **F** do nosso sinal filtrado. Para esta entidade ser implementada bastou chamar as

Figura 11: Definição da entidade BLOCO_OP e suas componentes

```
ENTITY BLOCK_OP IS
    PORT( C, Y : IN BIT_VECTOR (3 DOWNTO 0);
        clr_r, ld_r, ld_out, en_cod, s_cod0, s_cod1, CLK: IN BIT;
        F: OUT BIT_VECTOR(9 DOWNTO 0));
END BLOCK_OP;

ARCHITECTURE CKT OF BLOCK_OP IS

COMPONENT COD24 IS
    PORT(SC0, SC1, EN: IN BIT;
        S0,S1,S2,S3: OUT BIT);
END COMPONENT;

COMPONENT RC_BLOCK IS
    PORT(C, Y : IN BIT_VECTOR(3 DOWNTO 0);
        ld_r, ld_c, clr_r, CLK: IN BIT;
        S: OUT BIT_VECTOR(7 DOWNTO 0);
        YS: OUT BIT_VECTOR(3 DOWNTO 0));
END COMPONENT;

COMPONENT REG10 IS
    PORT( I: IN BIT_VECTOR(9 DOWNTO 0);
        CLK, CLR, EN: IN BIT;
        O: OUT BIT_VECTOR(9 DOWNTO 0));
END COMPONENT;

COMPONENT ADD9 is
    port(A, B: in bit_vector(8 downto 0);
            O: out bit_vector(8 downto 0);
            CO: OUT BIT );
end COMPONENT;

COMPONENT ADD8 is
    port(A, B: in bit_vector(7 downto 0);
            O: out bit_vector(7 downto 0);
            CO: OUT BIT );
end COMPONENT;
```

Fonte: Elaborado pelo autor (2021).

Figura 12: Circuito da entidade BLOCO_OP

```
SIGNAL  ld_c0, ld_c1, ld_c2, ld_c3, CO1,CO2: BIT;
SIGNAL Y1, Y2, Y3: BIT_VECTOR(3 DOWNTO 0);
SIGNAL YC1, YC2, YC3, AUXSUM1: BIT_VECTOR(7 DOWNTO 0);
SIGNAL AUX, SUM1, AUXSUM2: BIT_VECTOR(8 DOWNTO 0);
SIGNAL SUM2: BIT_VECTOR(9 DOWNTO 0);

BEGIN

C_DEF: COD24 PORT MAP(s_cod0, s_cod1 , en_cod, ld_c0, ld_c1, ld_c2, ld_c3);


RC1: RC_BLOCK PORT MAP(C, Y, ld_r, ld_c0, clr_r, CLK, YC1, Y1);
RC2: RC_BLOCK PORT MAP(C, Y1, ld_r, ld_c1, clr_r, CLK, YC2, Y2);
RC3: RC_BLOCK PORT MAP(C, Y2, ld_r, ld_c2, clr_r, CLK, YC3, Y3);

AUX(8) <= '0';
AUX(7 DOWNTO 0) <= YC3;

SOMA1: ADD8 PORT MAP(YC1, YC2, AUXSUM1, CO1);
SOMA2: ADD9 PORT MAP(SUM1, AUX, AUXSUM2, CO2);

SUM1(8)<= CO1;
SUM1(7 DOWNTO 0) <= AUXSUM1;

SUM2(9)<= CO2;
SUM2(8 DOWNTO 0) <= AUXSUM2 ;

REGISTRA: REG10 PORT MAP(SUM2, CLK, '1', ld_out, F);

END CKT;
```

Fonte: Elaborado pelo autor (2021).

entidades do bloco de controle e bloco operacional como componentes e realizar os *port maps* adequados em seu circuito como mostra a Figura 13.

Figura 13: Entidade FIR

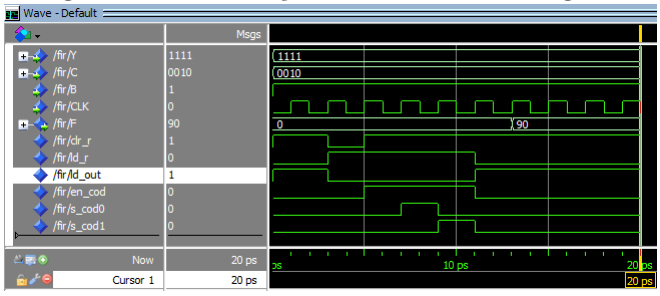
```
ENTITY FIR IS
    PORT(Y, C : IN BIT_VECTOR(3 DOWNTO 0);
         B, CLK: IN BIT;
         F: OUT BIT_VECTOR(9 DOWNTO 0));
END FIR;

ARCHITECTURE CKT OF FIR IS

COMPONENT BLOCK_CON IS
PORT(B, CLK: IN BIT;
    clr_r, ld_r, ld_out, en_cod, s_cod0, s_cod1: OUT BIT);
END COMPONENT;

COMPONENT BLOCK_OP IS
    PORT( C, Y : IN BIT_VECTOR (3 DOWNTO 0);
         clr_r, ld_r, ld_out, en_cod, s_cod0, s_cod1, CLK: IN BIT;
         F: OUT BIT_VECTOR(9 DOWNTO 0));
END COMPONENT;

SIGNAL clr_r, ld_r, ld_out, en_cod, s_cod0, s_cod1: BIT;

BEGIN

BLOCOCONTROLE: BLOCK_CON PORT MAP(B, CLK, clr_r, ld_r, ld_out, en_cod, s_cod0, s_cod1);
BLOCOOPERACIONAL: BLOCK_OP PORT MAP(C, Y, clr_r, ld_r, ld_out, en_cod, s_cod0, s_cod1, CLK, F);

END CKT;
```

Fonte: Elaborado pelo autor (2021).

2.4 Simulação

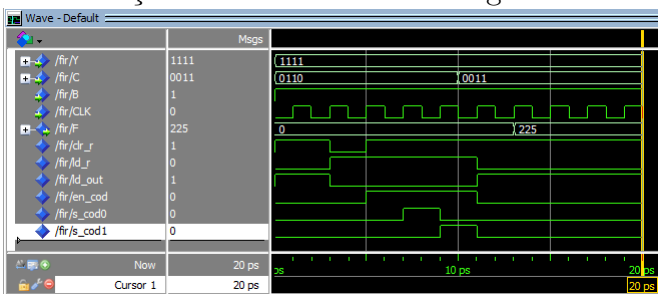
Para simular o código implementado foi utilizado o software ModelSim. Foi realizada 3 simulações, a primeira foi utilizando o valor das contantes $c_{0,1,2}$ iguais, outra com dois valores iguais e um distinto e, por fim, com os valores completamente distintos. O resultado das simulações podem ser vistos nas Figuras 14, 15 e 16.

Figura 14: Simulação com constantes iguais



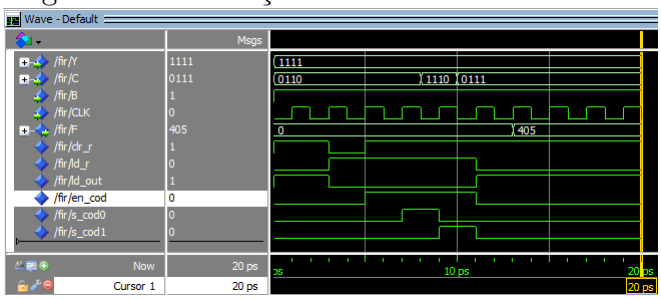
Fonte: Elaborado pelo autor (2021).

Figura 15: Simulação com duas constantes iguais e outra distinta



Fonte: Elaborado pelo autor (2021).

Figura 16: Simulação com constantes distintas



Fonte: Elaborado pelo autor (2021).

3 Conclusão

O presente relatório teve por finalidade apresentar a solução para o problema de projetar um filtro FIR com 3 taps. O filtro é capaz de receber um valor de sinal y e o valor das constantes $c_{0,1,2}$, ambos de 4 bits e fornecer em sua saída um dado **F** de 10 bits, correspondente ao sinal filtrado. Para solução deste problema foi utilizado o método de projeto Register Transfer Level(RTL), onde foi definido que a máquina responsável pelo filtro FIR de 3 taps possuiria 5 estados, sendo 3 deles responsáveis pelos taps e os outros dois de standby e reset da filtro, sendo definido as saídas em cada um dos estados. Por fim, foi possível implementar a lógica combinacional das saídas destes estados em um circuito e também simular por completo utilizando o software ModelSim.

Referências

VAHID, F. *Sistemas Digitais: Projeto, Otimização e HDLs*. [S.l.]: Bookman Editora, 2009.

A Código VHDL completo

```
--==============--
-- MEIO SOMADOR --
--==============--
entity HALF_ADD is
    port(A, B: in bit;
         S, CO: out bit);
end HALF_ADD;

architecture CKT of HALF_ADD is

begin

    S <= A xor B;
    CO <= A and B;

end CKT;


--==================--
-- SOMADOR COMPLETO --
--==================--
entity COMP_ADD is
    port(A, B, CI: in bit;
         S, CO: out bit);
end COMP_ADD;

architecture CKT of COMP_ADD is

begin

    S <= A xor B xor CI;
    CO <= (B and CI) or (A and CI) or (A and B);

end CKT;


--==================--
-- SOMADOR 8 BITS --
--==================--
entity ADD8 is
    port(A, B: in bit_vector(7 downto 0);
         O: out bit_vector(7 downto 0);
         CO: OUT BIT);
end ADD8;

architecture CKT of ADD8 is
```

```
component HALF_ADD is
    port(A, B: in bit;
         S, CO: out bit);
end component;

component COMP_ADD
    port(A, B, CI: in bit;
             S, CO: out bit);
end component;

signal VAI_UM : bit_vector(6 downto 0);

begin

    S0: HALF_ADD port map(A(0), B(0), O(0), VAI_UM(0));
    S1: COMP_ADD port map(A(1), B(1), VAI_UM(0), O(1), VAI_UM(1));
    S2: COMP_ADD port map(A(2), B(2), VAI_UM(1), O(2), VAI_UM(2));
    S3: COMP_ADD port map(A(3), B(3), VAI_UM(2), O(3), VAI_UM(3));
    S4: COMP_ADD port map(A(4), B(4), VAI_UM(3), O(4), VAI_UM(4));
    S5: COMP_ADD port map(A(5), B(5), VAI_UM(4), O(5), VAI_UM(5));
    S6: COMP_ADD port map(A(6), B(6), VAI_UM(5), O(6), VAI_UM(6));
    S7: COMP_ADD port map(A(7), B(7), VAI_UM(6), O(7), CO);

end CKT;


--==================--
-- SOMADOR 9 BITS --
--==================--
entity ADD9 is
    port(A, B: in bit_vector(8 downto 0);
             O: out bit_vector(8 downto 0);
             CO: OUT BIT );
end ADD9;

architecture CKT of ADD9 is

component HALF_ADD is
    port(A, B: in bit;
         S, CO: out bit);
end component;

component COMP_ADD
    port(A, B, CI: in bit;
             S, CO: out bit);
end component;
```

```
signal VAI_UM: bit_vector(7 downto 0);

begin

    S0: HALF_ADD port map(A(0), B(0), O(0), VAI_UM(0));
    S1: COMP_ADD port map(A(1), B(1), VAI_UM(0), O(1), VAI_UM(1));
    S2: COMP_ADD port map(A(2), B(2), VAI_UM(1), O(2), VAI_UM(2));
    S3: COMP_ADD port map(A(3), B(3), VAI_UM(2), O(3), VAI_UM(3));
    S4: COMP_ADD port map(A(4), B(4), VAI_UM(3), O(4), VAI_UM(4));
    S5: COMP_ADD port map(A(5), B(5), VAI_UM(4), O(5), VAI_UM(5));
    S6: COMP_ADD port map(A(6), B(6), VAI_UM(5), O(6), VAI_UM(6));
        S7: COMP_ADD port map(A(7), B(7), VAI_UM(6), O(7), VAI_UM(7));
    S8: COMP_ADD port map(A(8), B(8), VAI_UM(7), O(8), CO);

end CKT;

-- ========================
-- MULTIPLICADOR 4 BITS --
-- ========================
entity MUL4 is
    port(A, B: in bit_vector(3 downto 0);
            O: out bit_vector(7 downto 0);
            CO: out bit);
end MUL4;

architecture hardware of MUL4 is

component ADD8
    port(A, B: in bit_vector(7 downto 0);
            O: out bit_vector(7 downto 0);
            CO: out bit);
end component;

signal PP1, PP2, PP3, PP4, S0, S1: bit_vector(7 downto 0);
signal VAI_UM: bit_vector(1 downto 0);

begin
    PP1(0) <= B(0) and A(0);
    PP1(1) <= B(0) and A(1);
    PP1(2) <= B(0) and A(2);
    PP1(3) <= B(0) and A(3);
    PP1(4) <= '0';
    PP1(5) <= '0';
    PP1(6) <= '0';
    PP1(7) <= '0';

    PP2(0) <= '0';
    PP2(1) <= B(1) and A(0);
```

```
    PP2(2) <= B(1) and A(1);
    PP2(3) <= B(1) and A(2);
    PP2(4) <= B(1) and A(3);
    PP2(5) <= '0';
    PP2(6) <= '0';
    PP2(7) <= '0';

    PP3(0) <= '0';
    PP3(1) <= '0';
    PP3(2) <= B(2) and A(0);
    PP3(3) <= B(2) and A(1);
    PP3(4) <= B(2) and A(2);
    PP3(5) <= B(2) and A(3);
    PP3(6) <= '0';
    PP3(7) <= '0';

    PP4(0) <= '0';
    PP4(1) <= '0';
    PP4(2) <= '0';
    PP4(3) <= B(3) and A(0);
    PP4(4) <= B(3) and A(1);
    PP4(5) <= B(3) and A(2);
    PP4(6) <= B(3) and A(3);
    PP4(7) <= '0';

    SOMA0: ADD8 port map(PP1, PP2, S0, VAI_UM(0));
    SOMA1: ADD8 port map(S0, PP3, S1, VAI_UM(1));
    SOMA2: ADD8 port map(S1, PP4, O, CO);

end hardware;

--============
-- FLIPFLOP D --
--============
ENTITY ffd IS
        port ( clk ,D ,P , C : IN BIT ;
                q : OUT BIT );
END ffd ;

ARCHITECTURE ckt OF ffd IS
        SIGNAL qS : BIT;
BEGIN
    PROCESS ( clk ,P ,C )
            BEGIN
            IF P = '0' THEN qS <= '1';
            ELSIF C = '0' THEN qS <= '0';
            ELSIF clk = '1' AND clk ' EVENT THEN
            qS <= D ;
```

```
            END IF;
    END PROCESS ;
q <= qS ;
END ckt ;


--=====================--
-- MULTIPLEXADOR 2x1 --
--=====================--
entity MUX21 is
    port(A, B, S: in bit;
       O: out bit);
end MUX21;

architecture CKT of MUX21 is

begin

    O <= (B and S) or (A and (not S));

end CKT;


--=======================--
--  REGISTRADOR 4 BITS --
--=======================--
ENTITY REG4 IS
    PORT( I: IN BIT_VECTOR(3 DOWNTO 0);
          CLK, CLR, EN: IN BIT;
          O: OUT BIT_VECTOR(3 DOWNTO 0));
END REG4;

ARCHITECTURE CKT OF REG4 IS

COMPONENT ffd IS
    port ( clk ,D ,P , C : IN BIT ;
    q : OUT BIT );
END COMPONENT;

COMPONENT MUX21 is
    port(A, B, S: in bit;
       O: out bit);
 end COMPONENT;

SIGNAL CLEAR:BIT;
SIGNAL Q,D: BIT_VECTOR(3 DOWNTO 0);

BEGIN
```

```
    CLEAR <= CLR;

    MUX1:   MUX21 PORT MAP(Q(0), I(0), EN, D(0));
    MUX2:   MUX21 PORT MAP(Q(1), I(1), EN, D(1));
    MUX3:   MUX21 PORT MAP(Q(2), I(2), EN, D(2));
    MUX4:   MUX21 PORT MAP(Q(3), I(3), EN, D(3));


    FFD1: ffd PORT MAP (CLK, D(0), '1', CLEAR, Q(0));
    FFD2: ffd PORT MAP (CLK, D(1), '1', CLEAR, Q(1));
    FFD3: ffd PORT MAP (CLK, D(2), '1', CLEAR, Q(2));
    FFD4: ffd PORT MAP (CLK, D(3), '1', CLEAR, Q(3));

    O<= Q;

END CKT;


--=======================--
-- REGISTRADOR 10 BITS --
--=======================--
ENTITY REG10 IS
    PORT( I: IN BIT_VECTOR(9 DOWNTO 0);
          CLK, CLR, EN: IN BIT;
          O: OUT BIT_VECTOR(9 DOWNTO 0));
END REG10;

ARCHITECTURE CKT OF REG10 IS

COMPONENT ffd IS
    port ( clk ,D ,P , C : IN BIT ;
    q : OUT BIT );
END COMPONENT;

COMPONENT MUX21 is
    port(A, B, S: in bit;
       O: out bit);
 end COMPONENT;

SIGNAL CLEAR:BIT;
SIGNAL Q,D: BIT_VECTOR(9 DOWNTO 0);

BEGIN

CLEAR <= CLR;

MUX1:   MUX21 PORT MAP(Q(0), I(0), EN, D(0));
```

```
MUX2:  MUX21 PORT MAP(Q(1), I(1), EN, D(1));
MUX3:  MUX21 PORT MAP(Q(2), I(2), EN, D(2));
MUX4:  MUX21 PORT MAP(Q(3), I(3), EN, D(3));
MUX5:  MUX21 PORT MAP(Q(4), I(4), EN, D(4));
MUX6:  MUX21 PORT MAP(Q(5), I(5), EN, D(5));
MUX7:  MUX21 PORT MAP(Q(6), I(6), EN, D(6));
MUX8:  MUX21 PORT MAP(Q(7), I(7), EN, D(7));
MUX9:  MUX21 PORT MAP(Q(8), I(8), EN, D(8));
MUX10: MUX21 PORT MAP(Q(9), I(9), EN, D(9));

FFD1: ffd PORT MAP (CLK, D(0), '1', CLEAR, Q(0));
FFD2: ffd PORT MAP (CLK, D(1), '1', CLEAR, Q(1));
FFD3: ffd PORT MAP (CLK, D(2), '1', CLEAR, Q(2));
FFD4: ffd PORT MAP (CLK, D(3), '1', CLEAR, Q(3));
FFD5: ffd PORT MAP (CLK, D(4), '1', CLEAR, Q(4));
FFD6: ffd PORT MAP (CLK, D(5), '1', CLEAR, Q(5));
FFD7: ffd PORT MAP (CLK, D(6), '1', CLEAR, Q(6));
FFD8: ffd PORT MAP (CLK, D(7), '1', CLEAR, Q(7));
FFD9: ffd PORT MAP (CLK, D(8), '1', CLEAR, Q(8));
FFD10: ffd PORT MAP (CLK, D(9), '1', CLEAR, Q(9));

O<= Q;

END CKT;


--============--
-- BLOCO RxC --
--============--
ENTITY RC_BLOCK IS
    PORT(C, Y : IN BIT_VECTOR(3 DOWNTO 0);
         ld_r, ld_c, clr_r, CLK: IN BIT;
         S: OUT BIT_VECTOR(7 DOWNTO 0);
         YS: OUT BIT_VECTOR(3 DOWNTO 0));
END RC_BLOCK;

ARCHITECTURE CKT OF RC_BLOCK IS

COMPONENT MUL4 IS
    port(A, B: in bit_vector(3 downto 0);
         O: out bit_vector(7 downto 0);
         CO: out bit);
END COMPONENT;

COMPONENT REG4 IS
    PORT( I: IN BIT_VECTOR(3 DOWNTO 0);
         CLK, CLR, EN: IN BIT;
         O: OUT BIT_VECTOR(3 DOWNTO 0));
```

```
END COMPONENT;

SIGNAL CS, Ym: BIT_VECTOR(3 DOWNTO 0);
SIGNAL CO: BIT;

BEGIN

REG1: REG4 PORT MAP(C, CLK, '1', ld_c, CS);
REG2: REG4 PORT MAP(Y, CLK, clr_r, ld_r, Ym);

MULTIPLICADOR: MUL4 PORT MAP(CS, Ym, S, CO);

YS<= Ym;

END CKT;


--==================--
-- CODIFICADOR 2X4 --
--==================--
ENTITY COD24 IS
    PORT(SC0, SC1, EN: IN BIT;
         S0,S1,S2,S3: OUT BIT);
END COD24;

ARCHITECTURE CKT OF COD24 IS

BEGIN

S0 <= NOT SC0 AND NOT SC1 AND EN;
S1 <= SC0 AND NOT SC1 AND EN;
S2 <= NOT SC0 AND SC1 AND EN;
S3 <= SC0 AND SC1 AND EN;

END CKT;


--====================--
-- BLOCO OPERACIONAL --
--====================--
ENTITY BLOCK_OP IS
    PORT( C, Y : IN BIT_VECTOR (3 DOWNTO 0);
          clr_r, ld_r, ld_out, en_cod, s_cod0, s_cod1, CLK: IN BIT;
          F: OUT BIT_VECTOR(9 DOWNTO 0));
END BLOCK_OP;

ARCHITECTURE CKT OF BLOCK_OP IS
```

```
COMPONENT COD24 IS
    PORT(SC0, SC1, EN: IN BIT;
         S0,S1,S2,S3: OUT BIT);
END COMPONENT;

COMPONENT RC_BLOCK IS
    PORT(C, Y : IN BIT_VECTOR(3 DOWNTO 0);
         ld_r, ld_c, clr_r, CLK: IN BIT;
         S: OUT BIT_VECTOR(7 DOWNTO 0);
         YS: OUT BIT_VECTOR(3 DOWNTO 0));
END COMPONENT;

COMPONENT REG10 IS
    PORT( I: IN BIT_VECTOR(9 DOWNTO 0);
         CLK, CLR, EN: IN BIT;
         O: OUT BIT_VECTOR(9 DOWNTO 0));
END COMPONENT;

COMPONENT ADD9 is
    port(A, B: in bit_vector(8 downto 0);
         O: out bit_vector(8 downto 0);
         CO: OUT BIT );
end COMPONENT;

COMPONENT ADD8 is
    port(A, B: in bit_vector(7 downto 0);
         O: out bit_vector(7 downto 0);
         CO: OUT BIT );
end COMPONENT;


SIGNAL ld_c0, ld_c1, ld_c2, ld_c3, CO1,CO2: BIT;
SIGNAL Y1, Y2, Y3: BIT_VECTOR(3 DOWNTO 0);
SIGNAL YC1, YC2, YC3, AUXSUM1: BIT_VECTOR(7 DOWNTO 0);
SIGNAL AUX, SUM1, AUXSUM2: BIT_VECTOR(8 DOWNTO 0);
SIGNAL SUM2: BIT_VECTOR(9 DOWNTO 0);

BEGIN

C_DEF: COD24 PORT MAP(s_cod0, s_cod1 , en_cod, ld_c0, ld_c1, ld_c2, ld_c3


RC1: RC_BLOCK PORT MAP(C, Y, ld_r, ld_c0, clr_r, CLK, YC1, Y1);
RC2: RC_BLOCK PORT MAP(C, Y1, ld_r, ld_c1, clr_r, CLK, YC2, Y2);
RC3: RC_BLOCK PORT MAP(C, Y2, ld_r, ld_c2, clr_r, CLK, YC3, Y3);

AUX(8) <= '0';
AUX(7 DOWNTO 0) <= YC3;
```

```
SOMA1: ADD8 PORT MAP(YC1, YC2, AUXSUM1, CO1);
SOMA2: ADD9 PORT MAP(SUM1, AUX, AUXSUM2, CO2);

SUM1(8)<= CO1;
SUM1(7 DOWNTO 0) <= AUXSUM1;

SUM2(9)<= CO2;
SUM2(8 DOWNTO 0) <= AUXSUM2 ;

REGISTRA: REG10 PORT MAP(SUM2, CLK, '1', ld_out, F);

END CKT;


--==============================--
-- LÓGICA BOTÃO SINCRONIZADO --
--==============================--
ENTITY LOG_BS IS
    PORT(Q1, Q0, B: IN BIT;
         D1, D0, PRESS: OUT BIT);
END LOG_BS;

ARCHITECTURE CKT OF LOG_BS IS

BEGIN

D1 <= (NOT Q1 AND Q0) OR (Q1 AND NOT Q0 AND B);
D0 <= NOT Q1 AND NOT Q0 AND B;
PRESS <= NOT Q1 AND Q0;

END CKT;


--====================--
-- BOTÃO SINCRONIZADO --
--====================--
ENTITY BS IS
PORT(CLK, B: IN BIT;
     PRESS: OUT BIT);
END BS;

ARCHITECTURE CKT OF BS IS

COMPONENT ffd IS
     port ( clk ,D ,P , C : IN BIT ;
     q : OUT BIT );
END COMPONENT;
```

```
COMPONENT LOG_BS IS
    PORT(Q1, Q0, B: IN BIT;
         D1, D0, PRESS: OUT BIT);
END COMPONENT;

SIGNAL Q1,Q0,D1,D0:BIT;

BEGIN

LOGIC: LOG_BS PORT MAP (Q1, Q0, B, D1, D0, PRESS);
FFD1: ffd PORT MAP(CLK, D1, '1', '1', Q1);
FFD2: ffd PORT MAP(CLK, D0, '1', '1', Q0);

END CKT;
```

```
--=============================================-
-- L G I C A  COMBINACIONAL DO BLOCO DE CONTROLE --
--=============================================-
ENTITY LOG_COMB IS
PORT(Q2, Q1, Q0, PRESS: IN BIT;
     D2, D1, D0, clr_r, ld_r, ld_out, en_cod, s_cod0, s_cod1: OUT BIT);
END LOG_COMB;

ARCHITECTURE CKT OF LOG_COMB IS

BEGIN

D2 <= NOT Q2 AND Q1 AND Q0;
D1 <= (NOT Q2 AND NOT Q1 AND Q0) OR ( NOT Q2 AND Q1 AND NOT Q0);
D0 <= (NOT Q2 AND NOT Q0 AND PRESS) OR (NOT Q2 AND Q1 AND NOT Q0);
clr_r <= (NOT Q1 AND NOT Q0) OR (NOT Q2 AND Q1);
ld_r <= (NOT Q2 AND Q0) OR (NOT Q2 AND Q1) OR (Q2 AND NOT Q1 AND NOT Q0
ld_out <= NOT Q2 AND NOT Q1 AND NOT Q0;
en_cod <= (NOT Q2 AND Q1) OR (Q2 AND NOT Q1 AND NOT Q0);
s_cod0 <= NOT Q2 AND Q1 AND Q0;
s_cod1 <= Q2 AND NOT Q1 AND NOT Q0;

END CKT;
```

```
--=====================-
-- BLOCO DE CONTROLE --
--=====================-
ENTITY BLOCK_CON IS
PORT(B, CLK: IN BIT;
     clr_r, ld_r, ld_out, en_cod, s_cod0, s_cod1: OUT BIT);
```

```
END BLOCK_CON;

ARCHITECTURE CKT OF BLOCK_CON IS

COMPONENT ffd IS
     port ( clk ,D ,P , C : IN BIT ;
     q : OUT BIT );
END COMPONENT;

COMPONENT LOG_COMB IS
PORT(Q2, Q1, Q0, PRESS: IN BIT;
     D2, D1, D0, clr_r , ld_r , ld_out , en_cod , s_cod0 , s_cod1: OUT BIT);
END COMPONENT;

COMPONENT BS IS
PORT(CLK, B: IN BIT;
     PRESS: OUT BIT);
END COMPONENT;

SIGNAL Q2, Q1, Q0, PRESS, D2, D1, D0: BIT;

BEGIN

DEF_PRESS: BS PORT MAP(CLK, B, PRESS);

LOGIC: LOG_COMB PORT MAP (Q2, Q1, Q0, PRESS, D2, D1, D0, clr_r , ld_r , ld_

FFD1: ffd PORT MAP(CLK, D2, '1', '1', Q2);
FFD2: ffd PORT MAP(CLK, D1, '1', '1', Q1);
FFD3: ffd PORT MAP(CLK, D0, '1', '1', Q0);

END CKT;


--============--
-- FILTRO FIR --
--============--
ENTITY FIR IS
        PORT(Y, C : IN BIT_VECTOR(3 DOWNTO 0);
                B, CLK: IN BIT;
                F: OUT BIT_VECTOR(9 DOWNTO 0));
END FIR;

ARCHITECTURE CKT OF FIR IS

COMPONENT BLOCK_CON IS
PORT(B, CLK: IN BIT;
     clr_r , ld_r , ld_out , en_cod , s_cod0 , s_cod1: OUT BIT);
```

```
END COMPONENT;

COMPONENT BLOCK_OP IS
    PORT( C, Y : IN BIT_VECTOR (3 DOWNTO 0);
          clr_r, ld_r, ld_out, en_cod, s_cod0, s_cod1, CLK: IN BIT;
          F: OUT BIT_VECTOR(9 DOWNTO 0));
END COMPONENT;

SIGNAL clr_r, ld_r, ld_out, en_cod, s_cod0, s_cod1: BIT;

BEGIN

BLOCOCONTROLE: BLOCK_CON PORT MAP(B, CLK, clr_r, ld_r, ld_out, en_cod, s_
BLOCOOPERACIONAL: BLOCK_OP PORT MAP(C, Y, clr_r, ld_r, ld_out, en_cod, s_

END CKT;
```