Relatorio 08

Erika Costa Alves

25 de setembro de 2019

0.1 Introdução

Nesse relatorio será mostrado como foi projetado o cicuito da **Figura 1**. O Circuito abaixo tem como objetivo de projetar um contador, que possui duas entradas, **A** e **B**, ambas de 4 bits. O contador vai começar do menor valor de entrada e vai contar até o maior valor de entrada. Além disso haverá duas entradas chamadas **Enable**, que vai ativar o circuito, e **Clear**, para dar um reset no circuito e deixar tudo em baixo.
O design do circuito será feito utilizando do software chamado **ModelSim**, e implementado com a linguagem VHDL.

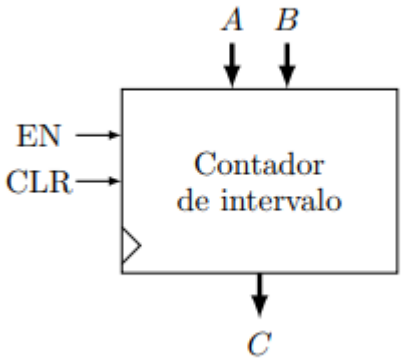

Figura 1: Circuito á ser Projetado.

0.2 Desenvolvimento

O Projeto foi desenvolvido em três partes diferentes, mas todas igualmente importantes. Sendo a primeira parte o **Contador Up/Down** com **carga paralela**, a segunda parte é o circuito lógico que vai limitar a contagem entre o maior valor e o menor valor, e por fim a terceira parte que vai ser o circuito lógico que vai comparar para saber qual é o maior ou menor valor de entrada.

0.2.1 Contador Up/Down com carga paralela

Para fazer o design de um contador Up/Down, ou seja, crescente e decrescente, foi feito um projeto de maquina de estado para um contador de 3 bits. Mas por que 3 bits? é muito complicado trabalhar com tabala verdade de mais de 4 bits. Contudo isso não é um problema, por que durante o processo de implementação foi possivel ver um padrão de combinação logica no contador. Por fim, foi-se feito uma tabela verdade utilizando uma tabela de **estado presente e estado futuro do Flip Flop JK**.

Q	Qn	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Tabela 1: Flip Flop Next State

Após fazer o design do contador em si, para fazer a carga paralela basta utilizar uma **nand** na entrada do **preset** do flip flop JK. No nand, os valores que serão colocados são exatametne um ativador, que chamaremos de **enable2**, e a carga de entrada A ou B.
Por fim, no contador implementado falta colocar o **Enable** que fazer com que toda a logica do circuito funcione. Para isso, observemos que no circuito há varios **and**, e cada and terá mais uma entrada que será exatamente o Enable.
Logo abaixo, na figura 2, vemos o resultado final do contador Up/Down.

M	QA	QB	QC	QAn	QBn	Qcn
0	0	0	0	0	0	1
0	0	0	1	0	1	0
0	0	1	0	0	1	1
0	0	1	1	1	0	0
0	1	0	0	1	0	1
0	1	0	1	1	1	0
0	1	1	0	1	1	1
0	1	1	1	0	0	0
1	0	0	0	0	0	1
1	0	0	1	0	1	0
1	0	1	0	0	1	1
1	0	1	1	1	0	0
1	1	0	0	1	0	1
1	1	0	1	1	1	0
1	1	1	0	1	1	1
1	1	1	1	0	0	0

Tabela 2: Tabela verdade para o Estado Presente e Futuro

0.2.2 Limitador de contagem

A segunda parte do design do projeto foi feita utilizando dois comparadores e uma flip flop JK. Os dois comparadores vão comparar com a saida do contador com A ou B. Caso A seja o menor número, e B seja o maior número, o contador irá contar de A até B.
Entretanto, como será a logica para que o bit do Up/Down seja alto ou baixo? Para isso foi utilizado um flip flop, no qual o clock está aterrado e as entradas J e K não importam. As saidas dos contadores vão ficar no set e preset do flip flop.

0.2.3 Menor ou Maior valor

Por fim, para saber qual é o maior ou menor valor foi utilizado um um comparador de magnetude, e depois, para fazer a seleção das entradas foi utlizado um **Mux 2x1**.

0.3 Implementação em VHDL

```
entity FFJK is
port(   clk, j, k, p, c : in bit;
        q               : out bit
);
end FFJK;

architecture ckt of FFJK 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
                        if j = '1' and k = '1' then qs <= not qs;
                        elsif j = '1' and k = '0' then qs <= '1';
                        elsif j = '0' and k = '1' then qs <= '0';
                        end if;
                end if;
        end process;

        q <= qs;

end ckt;
```

Figura 2: Flip Flop JK

```
---- Comaprador de 4 bits ----
--- Z é A igual a B   ----
--- X é A maior que B ----
--- Y é A menor que B ----


entity Comparator4Bits is
port(   A, B    : in bit_vector(3 downto 0);
        Z       : out bit;
        X       : out bit;
        Y       : out bit
);
end Comparator4Bits;

architecture ckt of Comparator4Bits is

        signal sig1     : bit_vector(7 downto 0);

begin
        sig1(0) <= not(A(0) xor B(0));
        sig1(1) <= not(A(1) xor B(1));
        sig1(2) <= not(A(2) xor B(2));
        sig1(3) <= not(A(3) xor B(3));

        Z <= sig1(0) and sig1(1) and sig1(2) and sig1(3);
        X <= (A(3) and (not B(3))) or (sig1(3) and A(2) and
        Y <= (B(3) and (not A(3))) or (sig1(3) and B(2) and

end ckt;
```

Figura 3: Comparador.

1

[1]No VHDL do Contador Up/Down há um pequeno erro na imagem. Na linha 43 deveria ser sig1(0) não sig1(3).

```
entity Mux2x1 is
port(   I0, I1  : in bit;
        s       : in bit;
        enable  : in bit;
        Y       : out bit
);
end Mux2x1;

architecture ckt of Mux2x1 is

        signal sig1, sig2       : bit;

begin

        sig1 <= (not s) and I0;
        sig2 <= s and I1;

        Y <= enable and(sig1 or sig2);

end ckt;
```

Figura 4: Multiplexador

```
--- Contador U/D com entrada paralela ---
entity CounterUD is
port(   UD      : in bit;
        enable1 : in bit; -- funcionar o circuito
        enable2 : in bit; -- preset
        clock   : in bit;
        clear   : in bit;
        Entrada : in bit_vector(3 downto 0);
        Saida   : out bit_vector(3 downto 0)
);
end CounterUD;

architecture ckt of CounterUD is
                --- Flip Flop JK ---
component FFJK is
port(   clk, j, k, p, c : in bit;
        q               : out bit
);
end component;

                --- Signals ---

        signal sig1     : bit_vector(3 downto 0); -- Preset
        signal sig2     : bit_vector(3 downto 0); -- logica do jk
        signal sig3     : bit_vector(3 downto 0); -- saidas dos ff jk
        signal aux1     : bit_vector(2 downto 0);
        signal aux2     : bit_vector(2 downto 0);

begin

        --- Entrada no preset ---

        sig1(0) <= enable2 and Entrada(0);
        sig1(1) <= enable2 and Entrada(1);
        sig1(2) <= enable2 and Entrada(2);
        sig1(3) <= enable2 and Entrada(3);
```

Figura 5: Contador parte 1.

0.4 Conclusão e Execução

No final, temos os resultados finais dos codigos. é possivel observar que o design está funcionamente normalmente como previsto. É interessante ver como a ideia do registrador é tão simples e bastante util. Algo tão simples como pode ter tanta utilizades.

```
--- Logica do Contador ---

sig2(0) <= '1';

aux1(0) <= (not UD) and sig3(0) and enable1;
aux2(0) <= UD and (not sig3(3)) and enable1;
sig2(1) <= UD xor sig3(0);

aux1(1) <= aux1(0) and sig3(1) and enable1;
aux2(1) <= aux2(0) and (not sig3(1)) and enable1;
sig2(2) <= aux1(1) or aux2(1);

aux1(2) <= aux1(1) and sig3(2) and enable1;
aux2(2) <= aux2(1) and (not sig3(2)) and enable1;
sig2(3) <= aux1(2) or aux2(2);

--- Port map ---

bit0 : FFJK
port map(clock, sig2(0), sig2(0), sig1(0), clear, sig3(0));

bit1 : FFJK
port map(clock, sig2(1), sig2(1), sig1(0), clear, sig3(1));

bit2 : FFJK
port map(clock, sig2(2), sig2(2), sig1(0), clear, sig3(2));

bit3 : FFJK
port map(clock, sig2(3), sig2(3), sig1(0), clear, sig3(3));
```

Figura 6: Contador parte 2.

```
entity lab08 is
port(   A, B    : in bit_vector(3 downto 0);
        en      : in bit;
        enp     : in bit;
        clk     : in bit;
        clr     : in bit;
        C       : out bit_vector(3 downto 0)
);
end lab08;

architecture  ckt of lab08 is
        -- Contador U/D --
component CounterUD is
port(   UD      : in bit;
        enable1 : in bit;
        enable2 : in bit;
        clock   : in bit;
        clear   : in bit;
        Entrada : in bit_vector(3 downto 0);
        Saida   : out bit_vector(3 downto 0)
);
end component;
        --  Comparador  --
component Comparator4Bits is
port(   A, B    : in bit_vector(3 downto 0);
        Z       : out bit;
        X       : out bit;
        Y       : out bit
);
end component;

        -- Flip Flop JK --
component FFJK is
port(   clk, j, k, p, c : in bit;
        q               : out bit
);
end component;
```

Figura 7: VHDL final 1.

Usando o tipo de arquivo de simulação o ’.do’ é possivel simular o VHDL final projetado.

```
        --      MUX 2x1         --
component Mux2x1 is
port(   I0, I1  : in bit;
        s       : in bit;
        enable  : in bit;
        Y       : out bit
);
end component;

        --      Signals         --

        signal FFToCounter      : bit;
        signal Z1,X1,Y1         : bit;
        signal Z2,X2,Y2         : bit;
        signal Z3,X3,Y3         : bit;
        signal aux1, aux2       : bit;
        signal sig0, sig1       : bit_vector(3 downto 0);
        signal aux3             : bit_vector(3 downto 0);

begin

        C(3 downto 0) <= aux3(3 downto 0);

        contador : CounterUD -- contador
        port map(FFToCounter, en, enp, clk, clr, sig0(3 downto 0), aux3(3 downto 0));

        comparador1 : Comparator4Bits -- Condicao para crescer
        port map(sig0(3 downto 0), aux3(3 downto 0), Z1, X1, Y1);

        comparator2 : Comparator4Bits -- condicao para decrescer
        port map(sig1(3 downto 0), aux3(3 downto 0), Z2, X2, Y2);

        aux1 <= not Z1;
        aux2 <= not Z2;

        chinelinha : FFJK
        port map('0', '0', '0', aux2, aux1, FFToCounter);
```

Figura 8: VHDL final 2.

```
        -- Saber ser A é maior ou menor que B --

        comparator3 : Comparator4Bits
        port map(A(3 downto 0), B(3 downto 0), Z3, X3, Y3);

        -- Para A

        mux0    : mux2x1
        port map(A(0), B(0), X3, '1', sig0(0));

        mux1    : mux2x1
        port map(A(1), B(1), X3, '1', sig0(1));

        mux2    : mux2x1
        port map(A(2), B(2), X3, '1', sig0(2));

        mux3    : mux2x1
        port map(A(3), B(3), X3, '1', sig0(3));

        -- Para B

        mux00   : mux2x1
        port map(B(0), A(0), X3, '1', sig1(0));

        mux10   : mux2x1
        port map(B(1), A(1), X3, '1', sig1(1));

        mux20   : mux2x1
        port map(B(2), A(2), X3, '1', sig1(2));

        mux30   : mux2x1
        port map(B(3), A(3), X3, '1', sig1(3));

end ckt;
```

Figura 9: VHDL final 3.

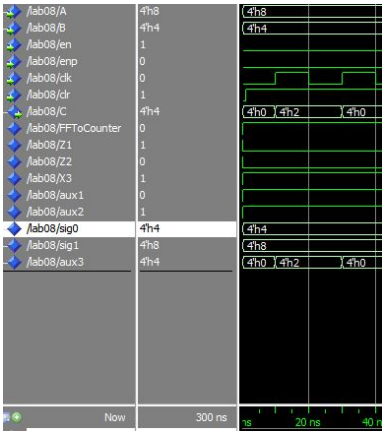

Figura 10: Simulação parte 1.

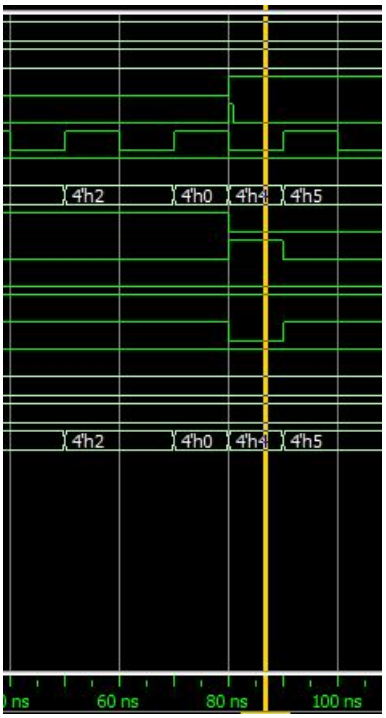

Figura 11: Simulação parte 2.

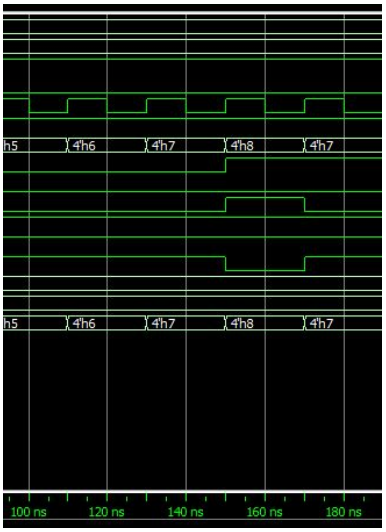

Figura 12: Simulação parte 3.

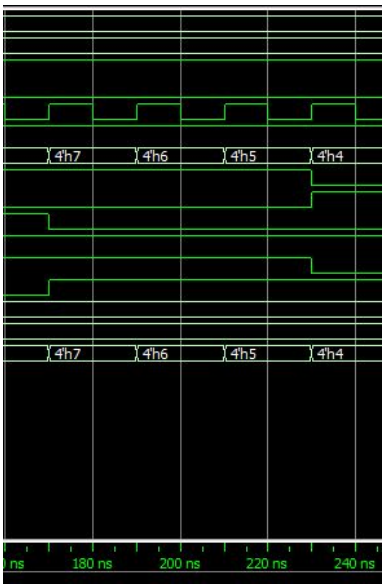

Figura 13: simulação parte 4.