Relatorio 11

Erika Costa Alves

21 de Outubro de 2019

1 Introdução

Neste relatorio será mostrado como foi resolvido o problema descrito no laboratorio 11, no qual será uma **Maquina de Vendas**. O circuito a ser projetado possui duas entradas principais, **A** de 8 bits, o valor da moeda que será coloca, e **S**, também de 8 bits, será o valor do produto que queremos da maquina de vendas. Há duas outras entradas secundarias, **c** para indicar quando foi colocado uma moeda, e **d** para indicar quando o produto foi liberado.

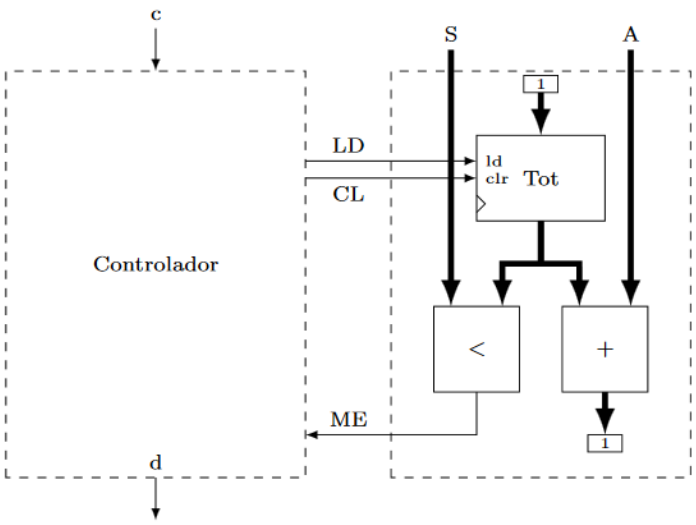

Figura 1: Circuito á ser Projetado.

2 Desenvolvimento

Nesse projeto, para ser resolvido foi divido em duas partes importantes, o **Bloco Operacional** e o **Bloco Combinacional**. Esses dois juntos formam o que conhecemos como **Projeto RTL**, ou seja, um maquina de estado de alto nivel. Logo a baixo, a **Figura 2** vemos uma maquina de estado de alto nivel para o projeto de Maquina de Vendas. E logo em seguida temos a versão reduzida, **Figura 3**, uma maquina de estado de baixo nivel.

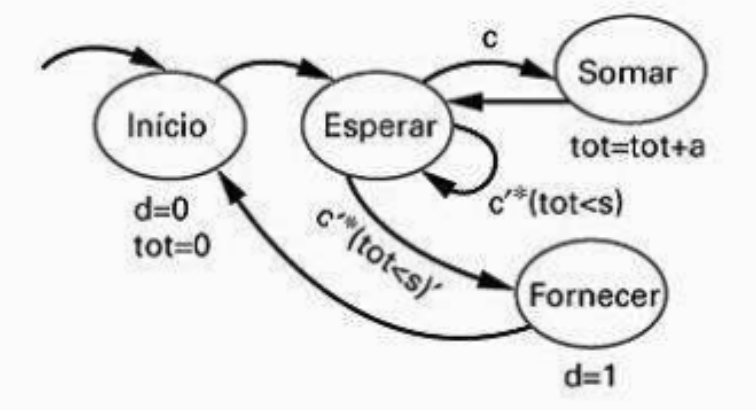

Figura 2: Maquina de Alto Nivel.

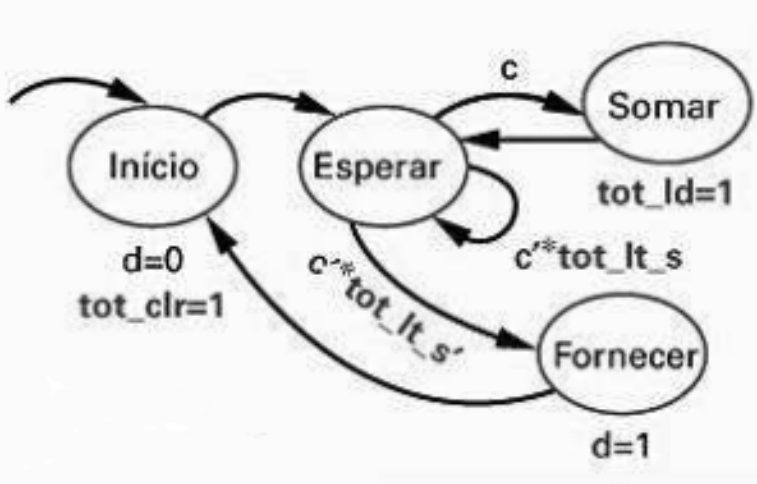

Figura 3: Maquina de Baixo Nivel.

2.1 Bloco Operacional

O bloco operacional será a parte que irá conter todos os componentes necessarios para o funcionamento do circuito, como por exemplo um multiplexador, um somador, um registrador, e por fim um comparador de magnetude. Note que o bloco operacional surgiu apartir da simplificação da maquina de de estado de alto nivel. Todas a condições que possuiam alguma operaçãos matematica foram retiradas e colcodas no lugar uma variavel.
Como foi mostrado na Maquina de Estado, toda vez que é inserido uma moeda, o valor da moeda é somado ao valores total, e para isso foi utilizado um somador de 8 Bits. O valor total é armazenado em um registrador de 8 bits, e no qual o valor da saída do comparador é comparado com o valor do produto. Caso o valor da saída será maior ou igual ao do produto, é liberado o produto.
é importante observar que, para fazer as ligações entre o registrador da soma total e o proprio somador é necessario um **multiplexador**, se não sempre irá ficar somando, e no caso só queremos que some se e somente se quando a variavel **tot ld** estiver em alto.

2.2 Bloco combinacional

O bloco combinacional é basicamente a logica para a mudança de estados, da maquina de estados, e os registradores que irão amazenar os bits de estado. Para montar a tabela verdade, como é possivel ver na **Tabela 1**, foi utilizados os bits de saida do bloco operacional. Por fim, as saídas do bloco combinacional serão tanto o **d** como os bits de sinal para zerar ou carregar o valor no registrador do valor total.

S1	S0	c	ME	N1	N0	LD	CL	d
0	0	0	0	0	1	0	1	0
0	0	0	1	0	1	0	1	0
0	0	1	0	0	1	0	1	0
0	0	1	1	0	1	0	1	0
0	1	0	0	1	1	0	0	0
0	1	0	1	0	1	0	0	0
0	1	1	0	1	0	0	0	0
0	1	1	1	1	0	0	0	0
1	0	0	0	0	1	1	0	0
1	0	0	1	0	1	1	0	0
1	0	1	0	0	1	1	0	0
1	0	1	1	0	1	1	0	0
1	1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	0	1

Tabela 1: Tabela Verdade para o Bloco Combinacional e a Maquina de Estados.

3 Codigo em VHDL

O comparador de 8 bits foi feito a partir de um comprador de 4 bits. Tal que primeiro comparamos os quatro bits mais significativos, caso os bits sejam iguais, iremos comparar os 4 bits menos significativos.

```
entity BlocoCombinacional is
port(   clk     : in bit;
        c       : in bit;
        lt_s    : in bit;
        d       : out bit;
        tot_ld  : out bit;
        tot_clr : out bit
);
end BlocoCombinacional;

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

        signal s        : bit_vector(1 downto 0);
        signal n        : bit_vector(1 downto 0);

begin

        n(0) <= (not s(0)) or ((not s(1)) and (not c));
        n(1) <= ((not s(1)) and s(0) and c) or ((not s(1)) and s(0) and (not lt_s));
        d <= s(1) and s(0);
        tot_ld <= s(1) and (not s(0));
        tot_clr <= (not s(0)) and (not s(1));


        chinela0 : FFD
        port map(clk, n(0),'1', '1', s(0));

        chinela1 : FFD
        port map(clk, n(1),'1', '1', s(1));

end ckt;
```

Figura 4: Bloco Combinacional.

4 Resultados e Execução

Uma coisa importante de se observar nos resultados é que a saída **d** só será alta quando pararmos de colocar moeda dentro da maquina, ou seja, quando **c** estiver em baixo. Observe que na simulação ocorre exatamente isso, o valor total é somando enquanto estivermos colocando moeda, e mesmo que o valor total, **Tot**, seja maior que o valor do produto **s**.

```
entity comparator8Bits is
port(   A       : in bit_vector(7 downto 0);
        B       : in bit_vector(7 downto 0);
        Z       : out bit;
        X       : out bit;
        Y       : out bit
);
end comparator8Bits;

architecture ckt of comparator8Bits is
component Comparator4Bits is
port(   A, B    : in bit_vector(3 downto 0);
        Z       : out bit;
        X       : out bit;
        Y       : out bit
);
end component;

        signal aux1     : bit_vector(2 downto 0);
        signal aux2     : bit_vector(2 downto 0);

begin

        Z <= aux1(0) and aux2(0);
        X <= aux1(1) or (aux1(0) and aux2(1));
        Y <= aux1(2) or (aux1(0) and aux2(2));


        comaprador0 : Comparator4Bits
        port map(A(7 downto 4), B(7 downto 4), aux1(0), aux1(1), aux1(2));

        comapradorl : Comparator4Bits
        port map(A(3 downto 0), B(3 downto 0), aux2(0), aux2(1), aux2(2));


end ckt;
```

Figura 5: Comparador de 8 bits.

```
entity lablll is
port(   A, S    : in bit_vector(7 downto 0);
        clk     : in bit;
        c       : in bit;
        d       : out bit
);
end lablll;

architecture ckt of lablll is
        -- Componentes --
component Register8Bits is
port(   A       : in bit_vector(7 downto 0);
        clock   : in bit;
        clr     : in bit;
        B       : out bit_vector(7 downto 0)
);
end component;
component Adder8Bits is
port(   A       : in bit_vector(7 downto 0);
        B       : in bit_vector(7 downto 0);
        Ci      : in bit;
        Co      : out bit;
        S       : out bit_vector(7 downto 0)
);
end component;
component Mux2x1 is
port(   I0, I1  : in bit;
        s       : in bit;
        enable  : in bit;
        Y       : out bit
);
end component;
```

Figura 6: Junção de Tudo pt1.

```
component comparator8Bits is
port(   A       : in bit_vector(7 downto 0);
        B       : in bit_vector(7 downto 0);
        Z       : out bit;
        X       : out bit;
        Y       : out bit
);
end component;
        -- Bloco combinacional --
component BlocoCombinacional is
port(   clk     : in bit;
        c       : in bit;
        lt_s    : in bit;
        d       : out bit;
        tot_ld  : out bit;
        tot_clr : out bit
);
end component;

        signal AdderToMux       : bit_vector(7 downto 0);
        Signal MuxToReg         : bit_vector(7 downto 0);
        Signal RegToComp        : bit_vector(7 downto 0);
        signal RegToMux         : bit_vector(7 downto 0);
        signal RegToAdder       : bit_vector(7 downto 0);
        signal Cout             : bit;
        signal LD, CL, ME       : bit;
        signal clnot,aux1,aux2  : bit;
        signal aux3             : bit_vector(7 downto 0);
```

Figura 7: Junção de tudo pt2.

```
begin

        blocoComb : BlocoCombinacional
        port map(clk, c, ME, d, LD, CL);

        clnot <= not CL;

        somador : Adder8Bits
        port map(A(7 downto 0), RegToAdder(7 downto 0), '0', Cout, AdderToMux(7 downto 0));

        aux3(7 downto 0) <= AdderToMux(7 downto 0);

        multiplexador0 : mux2x1
        port map(RegToMux(0), aux3(0), LD, '1', MuxToReg(0));

        multiplexador1 : mux2x1
        port map(RegToMux(1), aux3(1), LD, '1', MuxToReg(1));

        multiplexador2 : mux2x1
        port map(RegToMux(2), aux3(2), LD, '1', MuxToReg(2));

        multiplexador3 : mux2x1
        port map(RegToMux(3), aux3(3), LD, '1', MuxToReg(3));

        multiplexador4 : mux2x1
        port map(RegToMux(4), aux3(4), LD, '1', MuxToReg(4));

        multiplexador5 : mux2x1
        port map(RegToMux(5), aux3(5), LD, '1', MuxToReg(5));

        multiplexador6 : mux2x1
        port map(RegToMux(6), aux3(6), LD, '1', MuxToReg(6));

        multiplexador7 : mux2x1
        port map(RegToMux(7), aux3(7), LD, '1', MuxToReg(7));

        registrador : Register8Bits
        port map(MuxToReg(7 downto 0), clk, clnot, RegToComp(7 downto 0));
```

Figura 8: Junção de tudo pt3.

```
        registrador : Register8Bits
        port map(MuxToReg(7 downto 0), clk, clnot, RegToComp(7 downto 0));

        RegToAdder(7 downto 0) <= RegToComp(7 downto 0);
        RegToMux(7 downto 0) <= RegToComp(7 downto 0);

        comporador  : comparator8Bits
        port map(S(7 downto 0), RegToComp(7 downto 0), aux1, ME, aux2);


end ckt;
```

Figura 9: Junção de tudo pt4.

```
vsim lab111
add wave *

force A(0) 0 0
force A(1) 0 0
force A(2) 0 0
force A(3) 0 0
force A(4) 0 0
force A(5) 1 0
force A(6) 0 0
force A(7) 0 0

force S(0) 0 0
force S(1) 0 0
force S(2) 0 0
force S(3) 0 0
force S(4) 0 0
force S(5) 0 0
force S(6) 0 0
force S(7) 1 0

force c 0 0, 1 30 -repeat 90

force clk 0 0, 1 5 -repeat 10


run 200
```

Figura 10: Simulação.

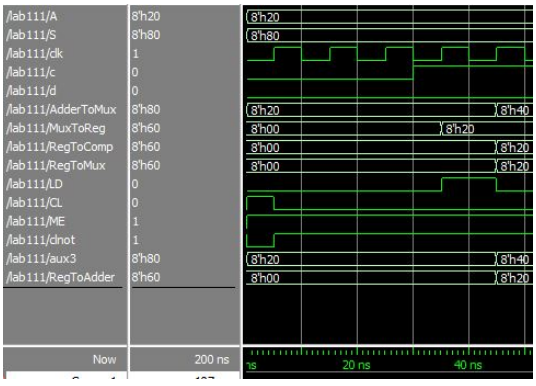

Figura 11: Resultado pt1.

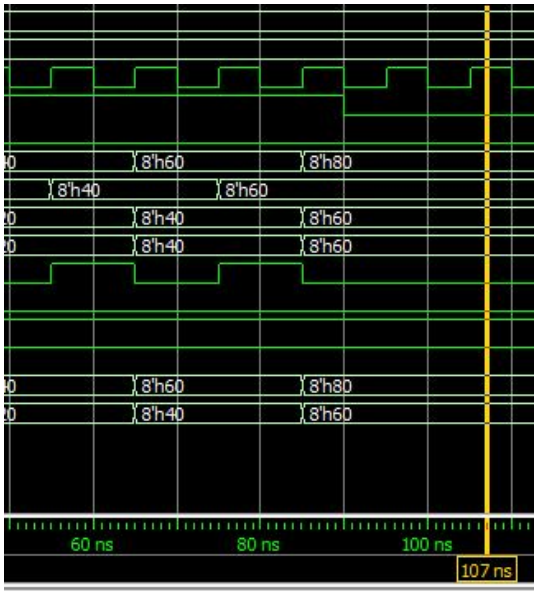

Figura 12: Resultado pt2.

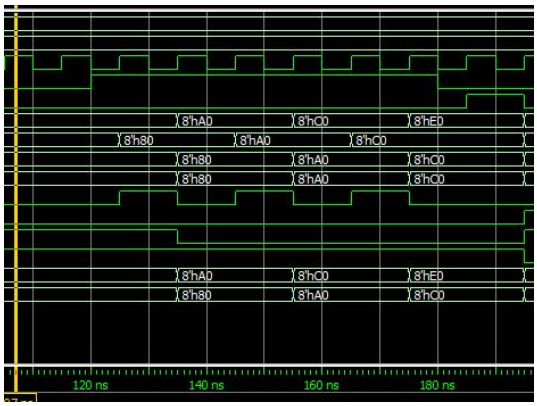

Figura 13: Resultado pt3.