Universidade Federal de Santa Catarina
Centro Tecnológico - CTC
Departamento de Engenharia Elétrica - EEL

Eduardo Kohler (16100722)
Guilherme Antônio F. Silva (16100732)

1208A

<eduardo.kohler98@gmail.com >
<antonioguil@gmail.com >

Relatório de Projeto Final EEL5105
2016.1

Florianópolis, 9 de Julho de 2015.

Conteúdo

1. Introdução

O objetivo deste trabalho foi implementar um código em VHDL, que simulasse uma máquina interativa com funcionamento similar a um telefone celular, utilizando a placa DE1-SoC para isto.

Para este projeto, utilizamos: quatro registradores, dois comparadores, dois contadores, uma máquina de estados, um decoder, dois multiplexadores, um conversor de clock de 50 MHz para 1 Hz e 4 Hz, além de cinco decodificadores de 7 segmentos. Ainda utilizamos uma memória ROM e um ButtonSync fornecidos pelo professor.

Nos registradores, a função é de armazenar em cada um dos quatro, 5 bits oriundos de um vetor de entrada de 20 bits, que pode ser a concatenação de um dígito da senha, um nome da agenda, a contagem ascendente, ou um vetor de zeros.

Um dos comparadores confere se a senha inserida pelo usuário é a correta, gerando diferentes sinais dependendo dos casos. O comparador seguinte garante que só possam ser feitas ligações caso ainda exista saldo disponível.

Nos contadores, há a contagem do tempo de ligação, no formato MM:SS, de forma crescente. Há também uma contagem decrescente, a qual representa o número de créditos que o usuário tem.

A máquina de estados gera os sinais de carga para os registradores, os sinais de seleção dos multiplexadores, o número de tentativas do usuário ao inserir senha, o sinal que possibilita a checagem da senha e o estado no qual o circuito se encontra.

O decoder transforma os 10 bits referentes aos digitos da senha em valores de 5 bits. Os multiplexadores são responsáveis por fornecer na saída apenas o valor correspondente ao sinal de seleção vindo da máquina de estados.

O conversor de clock utiliza o clock de 50 MHz da placa para gerar outros dois clocks de 1 e 4 Hz, para fins de contagem. A memória ROM é utilizada na forma de agenda, contendo os contatos que o usuário pode selecionar.

Os decodificadores de sete segmentos transformam os vetores de 5 bits em vetores de 7 bits, para ativar os segmentos apropriados e mostrar os caracteres no display corretamente.

1.1 Registradores

No bloco de registradores, a entrada é um vetor de 20 bits chamado de REG. Ele pode ser uma concatenação de um dígito da senha, um nome da agenda, a contagem ascendente, ou um vetor de zeros. Este vetor é dividido entre quatro registradores, cada um deles armazenando 5 bits deste vetor. Cada registrador tem um sinal de carga (C3, C2, C1, C0) e um reset assíncrono. A saída de cada registrador são vetores 5 bits (SEQ_3, SEQ_2, SEQ_1, SEQ_0), e o valor destes depende dos sinais de carga de cada um.

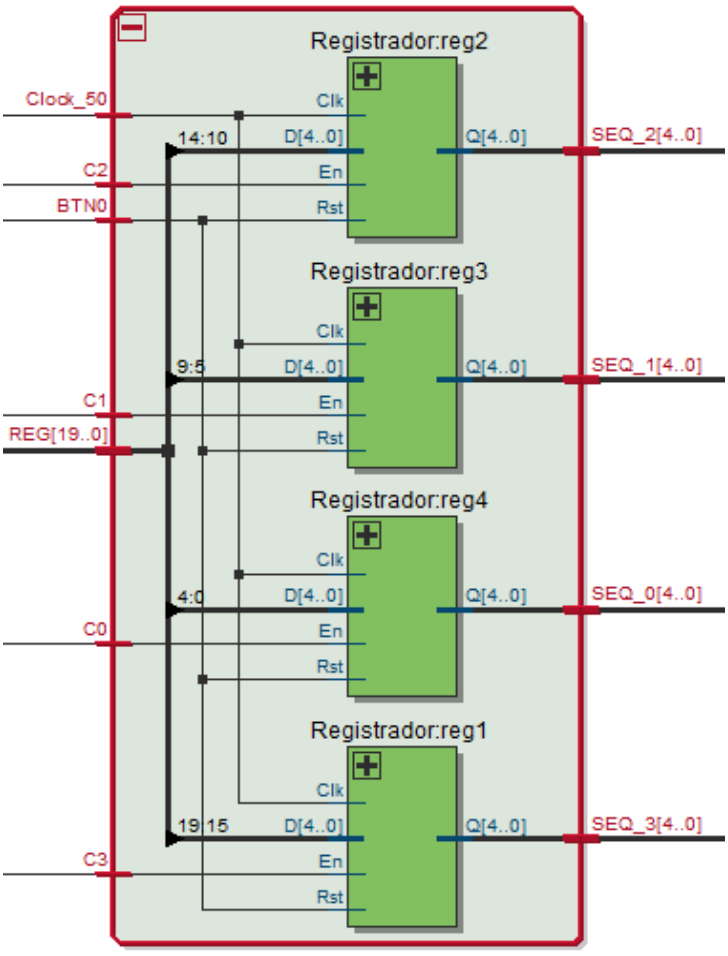

Figura 1 – Diagrama de blocos do bloco registrador

Na simulação, é possível ver o funcionamento destes. Enquanto o reset está em 0 (variável BTN0), todos os registradores estão zerados. Ao mudar o valor do reset, cada registrador recebe a respectiva parte de 5 bits que foi colocada no REG, já que os sinais de carga estão ativos. Ao colocarmos o sinal de carga de um deles (variável C3) pra zero e mudarmos o valor de REG, o registrador que pega este pedaço não muda de valor, pois seu sinal de carga está desativado. Após colocarmos o sinal como ativo outra vez, o registrador carrega o novo valor.

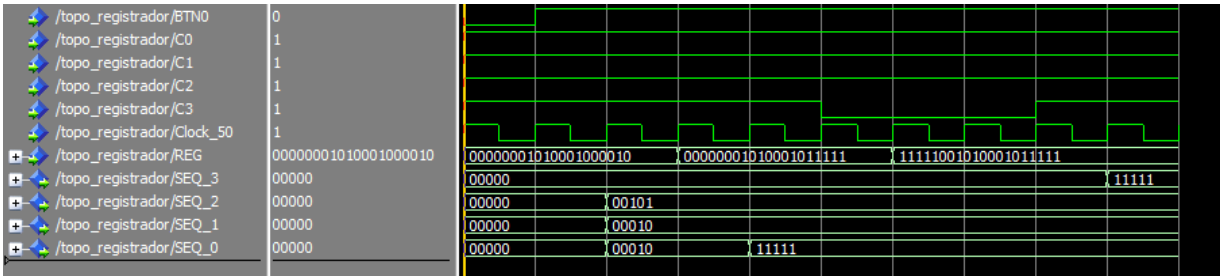

Figura 2 - Simulação do bloco registrador

1.2 Agenda

Nesse projeto, utilizamos uma memória ROM (Read-only memory) fornecida como material de apoio. Foram feitas, porém, algumas modificações. A memória implementada tem capacidade de armazenar 16 palavras, as quais representam os nomes dos contatos do telefone. Cada nome possui um tamanho de 20 bits. Para acessar a palavra desejada, deve-se colocar o endereço correspondente ao nome desejado, e a memória fornecerá esse dado na sua saída.

```
architecture behavioral of ROM1 is
  type mem is array ( 0 to 2**4 - 1) of std_logic_vector(19 downto 0);
  constant my_Rom : mem := (
    0  => "11110010101010111111", -- PAI
    1  => "10011110011001101010", -- LULA
    2  => "10000110011010111111", -- gUI
    3  => "01110011011100111111", -- edU
    4  => "10100010101010001010", -- nAnA
    5  => "10001110011000010110", -- HUGO
    6  => "01101101101011111000", -- dORy
    7  => "01101110010110111001", -- dUdU
    8  => "11011010100111011111", -- MAe
    9  => "11100101010110010110", -- ZICO
    10 => "11101110011110101010", -- XUXA
    11 => "10010110010110001010", -- JUCA
    12 => "01010011011011111000", -- Adry
    13 => "01011101110101010100", -- bRAn
    14 => "01110101111010101100", -- eRic
    15 => "01010101111100001010"); -- ARyA
```

Figura 3 – Estrutura ROM

```
begin
   process (address)
   begin
     case address is
       when "0000" => data <= my_rom(0);
       when "0001" => data <= my_rom(1);
       when "0010" => data <= my_rom(2);
       when "0011" => data <= my_rom(3);
       when "0100" => data <= my_rom(4);
       when "0101" => data <= my_rom(5);
       when "0110" => data <= my_rom(6);
       when "0111" => data <= my_rom(7);
       when "1000" => data <= my_rom(8);
       when "1001" => data <= my_rom(9);
       when "1010" => data <= my_rom(10);
       when "1011" => data <= my_rom(11);
       when "1100" => data <= my_rom(12);
       when "1101" => data <= my_rom(13);
       when "1110" => data <= my_rom(14);
       when "1111" => data <= my_rom(15);
       when others => data <= "00000000000000000000";
     end case;
  end process;
end architecture behavioral;
```

Figura 4 – Estrutura ROM

Através da imagens percebe-se como ocorreu o preenchimento da memória, assim como o endereço de cada nome.

1.3 Comparador

No bloco de comparadores, optamos por utilizar o método comportamental. O comparador de saldo tem como entrada apenas a variável Conta_Des, um vetor de 10 bits oriundo do bloco contador. A saída dessa variável, chamada saldo, depende do valor de entrada. Caso Conta_Des esteja com o valor zero, o saldo será 0. Qualquer outro valor que a variável Conta_Des tenha, fará com que a saída saldo tenho seu valor em 1.

O comparador de senha recebe como entrada um vetor de 20 bits chamado SEQ, resultado da concatenação dos valores de SEQ_3, SEQ_2, SEQ_1 e SEQ_0 dos registradores. Recebe também um sinal vindo da máquina de estados, chamado de Teste_Pass. A saída deste comparador depende das suas duas entradas: caso o sinal Teste_Pass esteja em 1 e a concatenação das sequências seja igual à senha correta (0722), então está saída será 1. Se alguma das condições não for verdadeira, a saída será sempre 0.

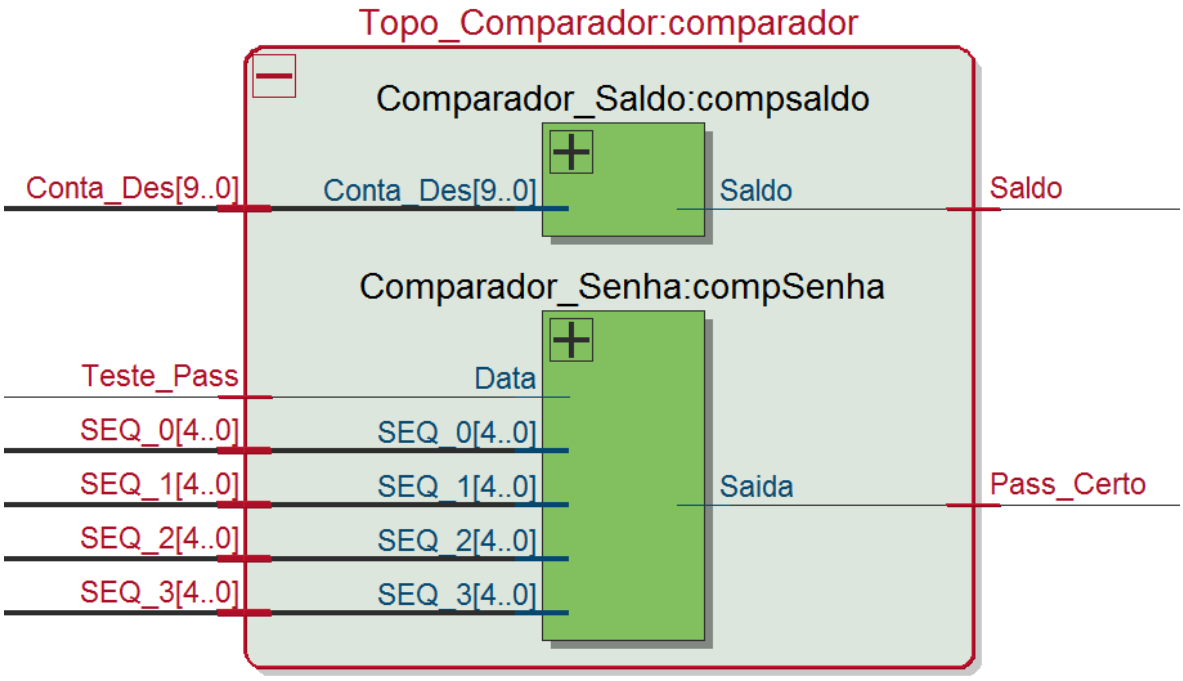

Figura 5 – Diagrama de blocos do bloco comparador

Na simulação, é possível ver que caso os valores contidos nas variáveis SEQ_3, SEQ_2, SEQ_1 e SEQ_0 estejam diferentes da senha, a saída, chama de Pass_Certo, será zero. Caso este valor esteja igual à senha e o sinal Teste_Pass da máquina de estados for 1, então a saída Pass_Certo receberá o valor 1.

Já no caso da saída chamada de saldo, seu valor depende do valor de Conta_Des que está vindo dos contadores. Quando esse chegar em zero, o saldo terá o valor zero. Em qualquer outro valor, a saída saldo deverá ser um.

Figura 6 – Simulação do bloco comparador

1.4 Contadores

Para gerar os clocks 1 e 2 necessários para as contagens, utilizou-se o clock de 50 MHz da placa e lógicas parecidas com as feitas no laboratório.

Como as oscilações do clock da placa acontecem a cada 20 nanossegundos, o componente responsável por gerar os clocks 1 e 2 faz contagens até 49.999.999 e 12.499.999, e quando esses valores são atingidos, incrementa-se o valor das saídas CLK 1 e 2 para 1. Dessa forma, gera-se os clocks 1 e 2 com as frequências de 1 e 4 Hz, respectivamente.

```
process (CLOCK_50, contador, contador2)
    begin
        if CLOCK_50'event and CLOCK_50 = '1' then
            contador <= contador + 1;
            contador2 <= contador2 + 1;
        end if;

        if contador = x"2FAF07F" then
            contador <= x"0000000";
            CLK1 <= '1';
        else
            CLK1 <= '0';
        end if;

        if contador2 = x"BEBC1F" then
            contador2 <= x"0000000";
            CLK2 <= '1';
        else
            CLK2 <= '0';
        end if;
    end process;
```

Figura 7 – Lógica implementada para controle de clock

Implementação usada para se obter o comportamento esperado dos clocks. Fica clara a interação entre o clock da placa e os sinais gerados pelo componente. A partir desses, então, foi possível criar os contadores ascendente e descendente.

Para o contador ascendente, o qual representa o tempo durante uma ligação, foi exigido o padrão de contagem no formato MM:SS (minutos e segundos). Dessa maneira, tornou-se necessária a implementação de um código capaz de contar através do clock de 1 Hz e manipular valores, a fim de mostrar valores coerentes nos displays.

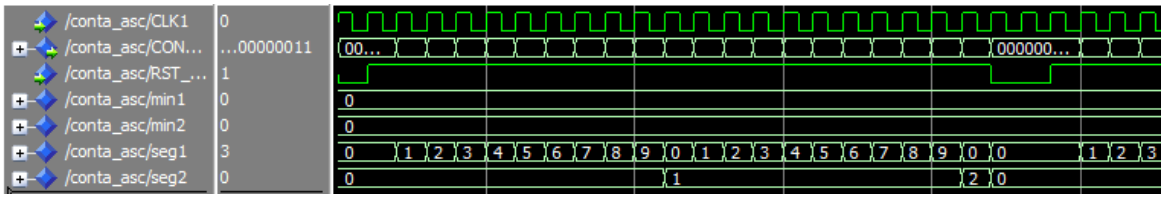

Figura 8 – Simulação da contagem ascendente

A simulação acima possibilita a visualização do comportamento dos sinais e o funcionamento do RST_CONT. Esse, quando recebe valor 0, é responsável por zerar a contagem, para que em cada nova ligação o processo inicie em 00:00.

No contador descendente, utilizamos um método semelhante. A contagem desse componente, no entanto, começa em 1010 e acontece em uma frequência de 4 Hz, ou seja, 4 decrementações por segundo. Ela acontece ao mesmo tempo que o procedimento ascendente, porém é mostrado nos leds. A decrementação representa o saldo do usuário do telefone, de maneira que, quando o valor apresentado atinge 0, o telefone é desligado, voltando ao estado incial. Além disso, se uma ligação é finalizada antes do saldo acabar, a próxima contagem é retomada de onde a última parou.

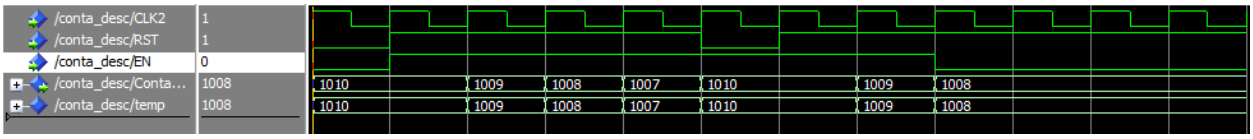

Figura 9 – Simulação da contagem descendente

A imagem mostra que, quando RST e o EN recebem 1, a contagem ocorre normalmente. Quando EN = 0, o último valor é mantido, e quando RST = 0, o valor inicial recebe 1010 novamente.

1.5 Selectores

No bloco de seletores, a entrada do multiplexador de 2 entradas e 1 saída são os vetores Tentativas e Conta_Des. A saída deste multiplexador dependerá do sinal de seleção que é gerado pela máquina de estados. Esta saída está ligada diretamente com os LEDS da placa.

Já no decoder, a entrada é apenas o vetor de 10 bits com o valor selecionado nos switches. A função dele é transformar este vetor de 10 bits em outro de 5 bits. Para isso, dependendo da posição do switch que o usuário levantar, o decoder gerará o valor correspondente ao switch utilizando apenas 5 bits. Desta forma, caso o usuário levante o último swtich (SW(9)) e deixe os outros abaixados, o decoder irá gerar o número 9 em 5 bits (01001).

A saída do decoder tem o nome de In_Pass. Essa saída é concatenada num vetor de 20 bits, que é uma das entradas do multiplexador de 4 entradas e uma saída. As outras entradas deste multiplexador são um vetor de 20 bits referente a Agenda e um outro vetor de 20 bits referente ao contador ascendente. A saída deste mux será o valor de REG e dependerá do sinal de seleção da máquina de estados.

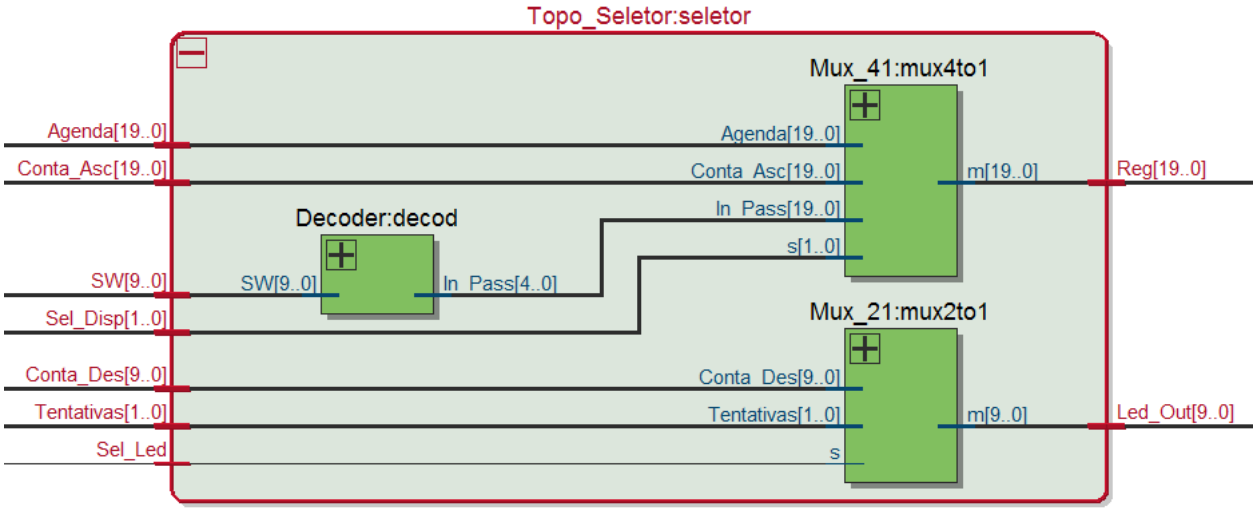

Figura 10 – Digrama de blocos do bloco seletor

Na simulação do mux 2:1, podemos ver que caso o sinal da máquina de estados (Sel_Led) esteja em 0, a saída do mux assume o valor contido em Tentativas concatenado a “00000000”. Caso o sinal de seleção esteja em 1, o valor da saída é o contido em Conta_Des.

Quanto ao mux 4:1, caso o valor de seleção seja “00”, sua saída assume um vetor de 20 bits só de 0's. Caso esteja em “01”, recebe o valor de In_Pass já concatenado. Com o valor de seleção em “10”, o valor recebido é o contido na entrada Agenda. Por fim, caso o sinal de seleção esteja em “11”, o valor fornecido na saída do mux é o da variável Conta_Asc.

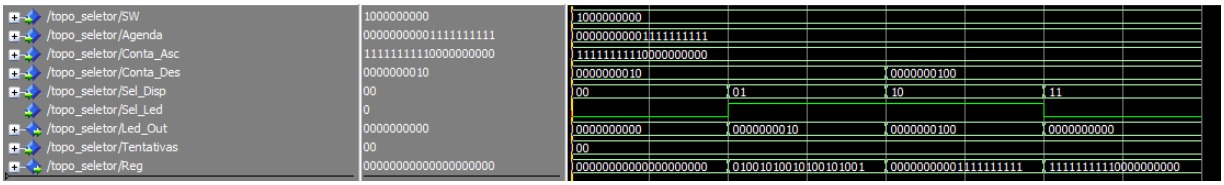

Figura 11 – Simulação do bloco seletor

2 Controlador

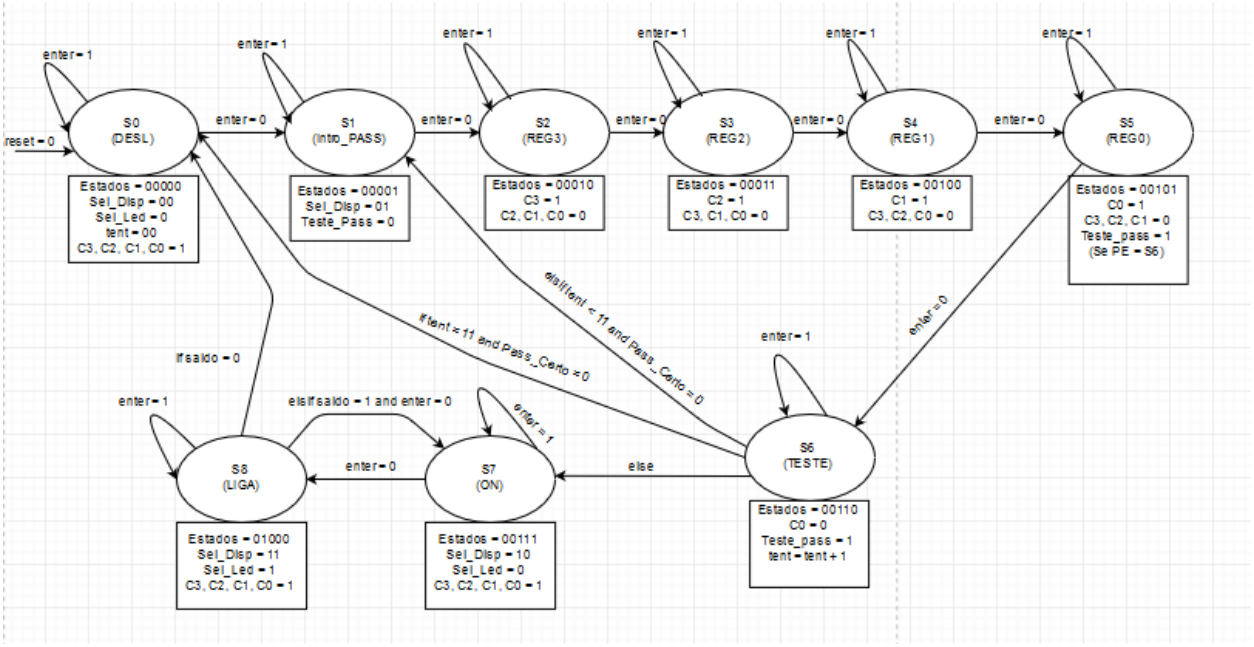

Figura 12 – FSM do projeto

S0: Estado inicial da FSM. Sel_Disp e tent recebem 00 a fim de zerar os displays e o número de tentativas. Sel_Led recebe 0 para mostrar o número de tentativas feitas.

S1: Segundo estado. Teste_Pass recebe 0 e Sel_Disp recebe 01, para que apareçam nos displays os caracteres digitados.

S2: Terceiro estado. O primeiro registrador recebe carga 1, e os demais carga 0, armazenando o valor do primeiro caractere digitado.

S3: Quarto estado. O segundo registrador recebe carga 1, e os demais carga 0, armazenando o valor do segundo caractere digitado.

S4: Quinto estado. O terceiro registrador recebe carga 1, e os demais carga 0, armazenando o valor do terceiro caractere digitado.

S5: Sexto estado. O quarto registrador recebe carga 1, e os demais carga 0, armazenando o valor do quarto caractere digitado. Além disso, Teste_Pass recebe 1 no caso do próximo estado for o S6.

S6: Sétimo estado. Nesse estado o teste é realizado sobre a senha, e dependendo dos resultados, o proximo estado pode ser o 0, o 1, ou o 7. A carga do quarto registrador recebe 0. Teste_Pass, que já está com o valor atribuído devido ao último estado permite a checagem da senha. Há também a soma de 1 no total de tentativas.

S7: Nono estado. Estado correspondente à lista de contatos. Com o objetivo de mostrar os nomes da agenda, o Sel_Disp recebe 10 e todos os registradores recebem 1 de carga.

S8: Último estado. Estado no qual a ligação ocorre. Sel_Disp recebe 11 para permitir a visualização da contagem ascendente nos displays, e Sel_Led recebe 1 a fim de mostrar a contagem descendente nos leds. Dependendo do valor do saldo e do enter, os próximos estados (que não o mesmo) podem ser 0 ou o 7.

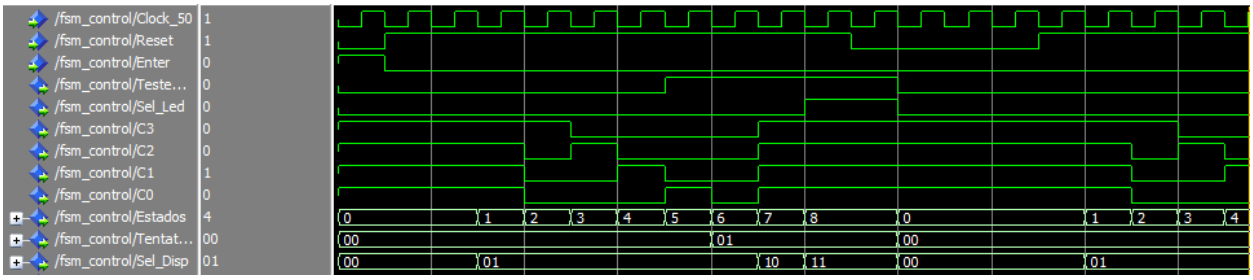

Figura 13 – Simulação da FSM

A figura mostra o funcionamento dos botões reset e enter e as variações de estado, assim como os valores assumidos por cada sinal (carga de registradores, seleção de mux, nº de tentativas, nº do estado) a partir do estado atual. Também é possível ver que há uma dependência pela borda de descida do clock. Esse foi o método pelo qual optamos para conseguir registrar valores importantes para as operações do sistema.

3. Resultados e conclusões

De maneira geral, o processo de síntese do projeto ocorreu da maneira planejada, pois cada bloco foi feito analisando quais seriam as funções desse dentro do trabalho. Em relação à conexão dos blocos, poucas foram as dúvidas, uma vez que a estrutura geral do projeto estava bem clara. Além disso, as simulações forneceram os valores esperados e, pelo fato de ajudarem também na interpretação de cada componente, ao fazer testes na placa, os problemas que surgiram puderam ser resolvidos com calma.

Anexo A – Observações

Durante o projeto, encontramos dificuldades em alguns pontos, como por exemplo, o melhor modo de fazer a contagem decrescente e como trabalhar com dois processos. Felizmente, todos os problemas foram resolvidos e o projeto foi concluído com sucesso.

O projeto abordou todos os pontos vistos em aula, possibilitando que compreendêssemos o VHDL a fim de desenvolver métodos eficientes para resolver problemas e projetar circuitos.