Departamento de Engenharia Eletrônica - Universidade Federal de Minas Gerais

Trabalho Final : Laboratório de Sistemas Digitais

Sistema de controle para elevador

André Prado Procópio
Augusto Guerra de Lima
Lucas Ribeiro da Silva

Belo Horizonte, Minas Gerais, Brasil; Inverno de 2024

Trabalho Final : Laboratório de Sistemas Digitais

Departamento de Engenharia Eletrônica - Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais, Brasil; Inverno de 2024

André Prado Procópio
approcopio@ufmg.br
Augusto Guerra de Lima
guerraaugusto@ufmg.br
Lucas Ribeiro da Silva
lucasrsilvak@ufmg.br

Sumário

1 Introdução

Este documento tem como objetivo expor a implementação de um projeto digital em *register-transfer-level* (RTL) para uma controladora e caminho de dados destinados a sistemas de elevadores.

O sistema, cuja descrição será detalhada a seguir, será capaz de registrar chamadas de andares e realizar a navegação entre eles. A implementação do projeto também aborda aspectos de segurança, como a carga do elevador, e inclui um estado de emergência.

Nas seções subsequentes, serão apresentados os diagramas projetados, bem como a descrição dos componentes do circuito na linguagem de descrição de *hardware VHDL*, acompanhados de suas respectivas visualizações e simulações.

1.1 Descrição do problema e da solução implementada

Conforme mencionado anteriormente, o problema consiste na implementação de um sistema RTL, seguindo uma metodologia de dois níveis para o desenvolvimento de um sistema digital de elevador. Foram implementados diversos componentes, majoritariamente no paradigma comportamental.

1.2 Disposição física

Em um nível de abstração mais alto, a disposição física das entradas e saídas do projeto pode ser compreendida conforme o diagrama abaixo.

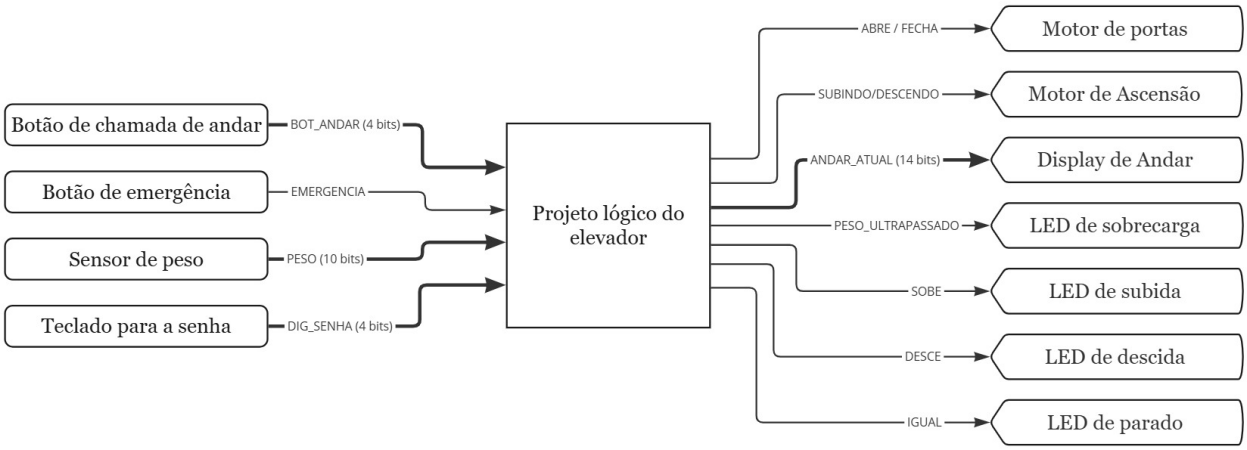

Figura 1: Diagrama de blocos da disposição física dos componentes do projeto.

Botão de Chamada

- **Descrição:** O botão de acionamento, tanto interno quanto externo, é utilizado pelo usuário para solicitar o deslocamento do elevador para o andar desejado.
- **Entradas:**
  - 4 *bits bits* `ANDAR`: contém o valor numérico do andar desejado para deslocamento.
  - 1 *bit* `ESCOLHER`: indica que um andar foi solicitado/escolhido.
- **Saída:** -

Botão de Emergência

- **Descrição:** O botão de emergência é utilizado pelo usuário para alertar o painel de controle sobre uma situação de emergência.

- **Entradas:** Bit `EMERGENCIA`: contém a informação se o Botão de Emergência foi pressionado.
- **Saída:** -

Teclado Numérico para Senha

- **Descrição:** O teclado numérico é utilizado pelo usuário para digitar a senha necessária para as funções do elevador.
- **Entradas:** 4 *bits* `SENHA`: contém um dos 4 dígitos da senha.
- **Saída:** -

Sensor de Peso

- **Descrição:** O sensor de peso retorna o peso da carga dentro do elevador.
- **Entradas:** 10 *bits* `PESO`: contém o valor numérico da carga no elevador.
- **Saída:** -

Motor de Ascensão

- **Descrição:** O motor de ascensão é responsável pela movimentação vertical do elevador.
- **Entradas:** -
- **Saídas:**
  - Bit `SUBINDO`: contém a informação para iniciar o processo de subida.
  - Bit `DESCENDO`: contém a informação para iniciar o processo de descida.

Motor de Portas

- **Descrição:** O motor de portas é responsável pela abertura e fechamento das portas.
- **Entradas:** -
- **Saídas:**
  - Bit `ABRIR`: contém a informação para iniciar o processo de abertura das portas.
  - Bit `FECHAR`: contém a informação para iniciar o processo de fechamento das portas.

Display de Andar

- **Descrição:** O display de andar indica o andar atual do elevador.
- **Entradas:** -
- **Saídas:**
  - 7 *bits* `ANDAR_UNIDADE`: contém a unidade do andar atual.
  - 7 *bits* `ANDAR_DEZENA`: contém a dezena do andar atual.

Led de Subida

- **Descrição:** O led de subida indica se o elevador está subindo.
- **Entradas:** -
- **Saída:** Bit `SOBE`: contém a informação se o elevador está subindo.

Led de Descida

- **Descrição:** O led de descida indica se o elevador está descendo.
- **Entradas:** -
- **Saída:** Bit `DESCE`: contém a informação se o elevador está descendo.

Led de Paragem

- **Descrição:** O led de paragem indica se o elevador está parado.
- **Entradas:** -
- **Saída:** Bit `IGUAL`: contém a informação se o elevador está parado.

Alerta de Sobrecarga

- **Descrição:** O alerta de sobrecarga indica se a carga no elevador excede o limite permitido.
- **Entradas:** -
- **Saída:** Bit `PESO_ULTRAPASSADO`: contém a informação se o elevador está com peso além do permitido.

1.3 Máquina de estados finitos alto nível

A imagem a seguir representa a máquina de estados finitos alto nível que descreve o comportamento do projeto, os estados juntamente com as componentes do caminho de dados serão descritos a seguir.

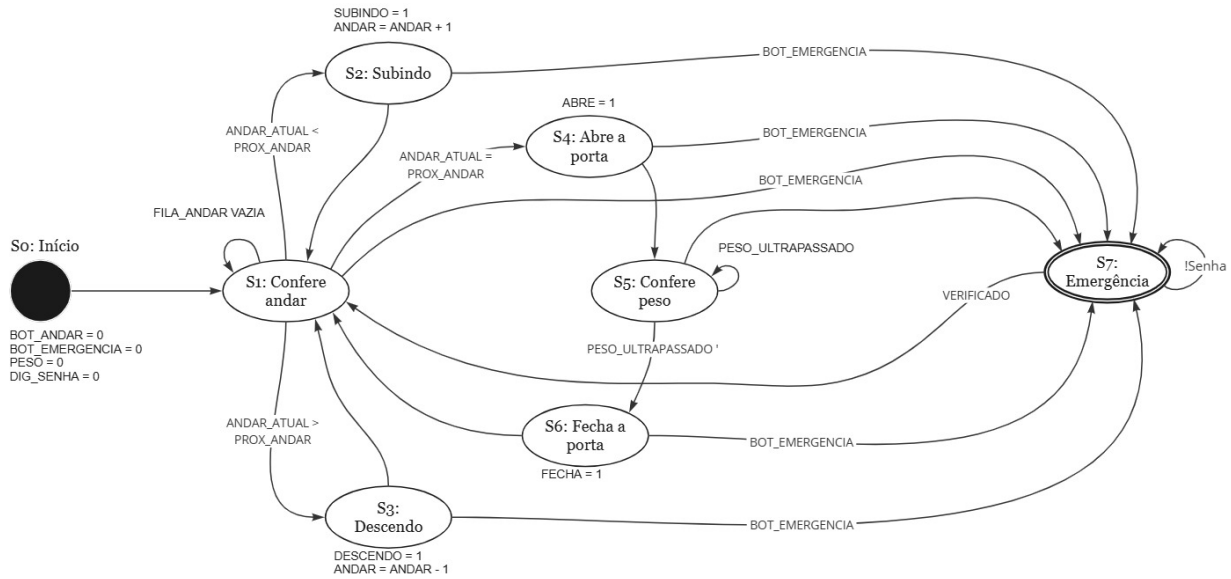

Figura 2: Diagrama de estados da máquina alto nível (humano) do projeto implementado.

A máquina de estados de alto nível inicia com todas as suas condições em zero, permanecendo nesse estado até que um andar seja solicitado no registrador da fila de andares (`Reg_FilaAndar`). Esse andar pode ser solicitado através dos botões de cada andar ou do painel interno do elevador.

Quando um andar é solicitado, ele será registrado no (`Reg_FilaAndar`). No (`Reg_FilaAndar`) ele será comparado com o andar atual do elevador pelo `Comparador`. Três estados podem ser ativados: (1) Sobe, se o andar atual for menor que o solicitado, (2) Desce, se for maior, e (3) Abre Porta, se o andar solicitado

for o mesmo que o atual. A componente `ContadorAndar` possui um registrador interno para o *andar atual*, enquanto a componente `ContadorProximo` itera pelo registrador da fila de andares (`Reg_FilaAndar`) até que um andar passado pelo multiplexador (`Mux`) seja verdadeiro.

Ao abrir as portas, a máquina passa para o próximo estado, que verifica a carga. Se o peso ultrapassar o limite permitido, o elevador permanecerá parado até que o peso seja reduzido suficientemente. Uma vez atingida essa condição, a máquina passa para o estado de fechamento de portas e volta a verificar se há um novo andar solicitado (`Reg_Peso` e `Comparador_Peso`).

É importante destacar que o estado de emergência pode ser ativado devido a possíveis falhas do elevador. Para sair do estado de emergência, é necessário que uma senha seja inserida (`Reg_Senha` e `Comparador_Senha`).

1.4 Caminho de dados

A Figura 2 demonstra em um diagrama os componentes implementados do caminho de dados. Na sessão 2 deste texto serão apresentadas as implementações com descrições, visualizações e as simulações de cada componente.

A tabela descreve as entradas e saídas do caminho de dados bem como os registradores utilizados internamente. Essas entradas e saídas serão melhores exploradas ma sessão da máquina de estados alto nível.

Entrada	Saídas	Registradores
Bot_Andar, Bot_Emergencia, Escolher, Digito, Peso	Sobe, Desce, Igual, BCD_Andar, Emergencia, Abre, Fecha	AndarAtual, FilaAndar, PesoMaximo, SenhaSistema

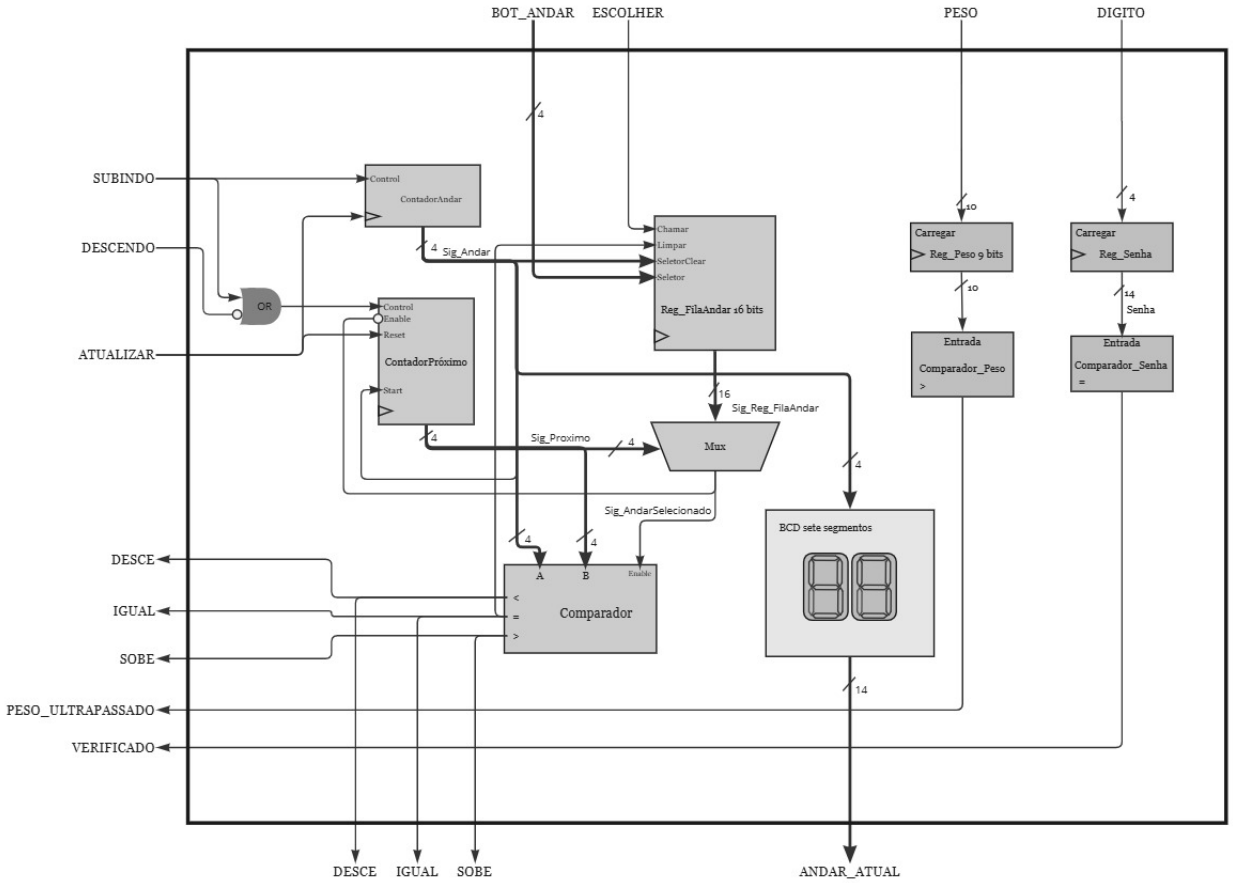

Figura 3: Desenho conceitual de projeto do caminho de dados e a comunicação entre os componentes implementados.

1.5 Diagrama de duas caixas

O projeto como um todo é constituído por uma unidade controladora, um caminho de dados, e entradas dispostas fisicamente conforme apresentado na subseção 1.2.. O diagrama a seguir ilustra a interconexão desses principais componentes do projeto, onde a unidade controladora é representada por uma máquina de estados de baixo nível.

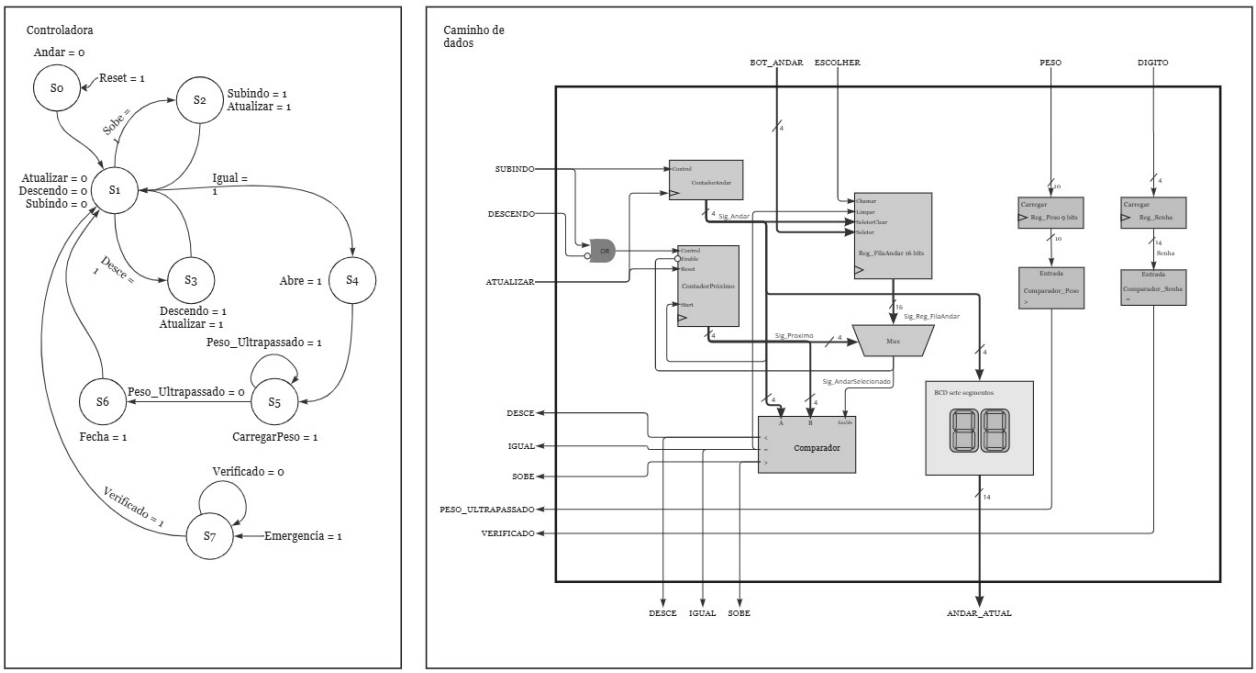

Figura 4: Diagrama da controladora e do caminho de dados e suas conexões do projeto.

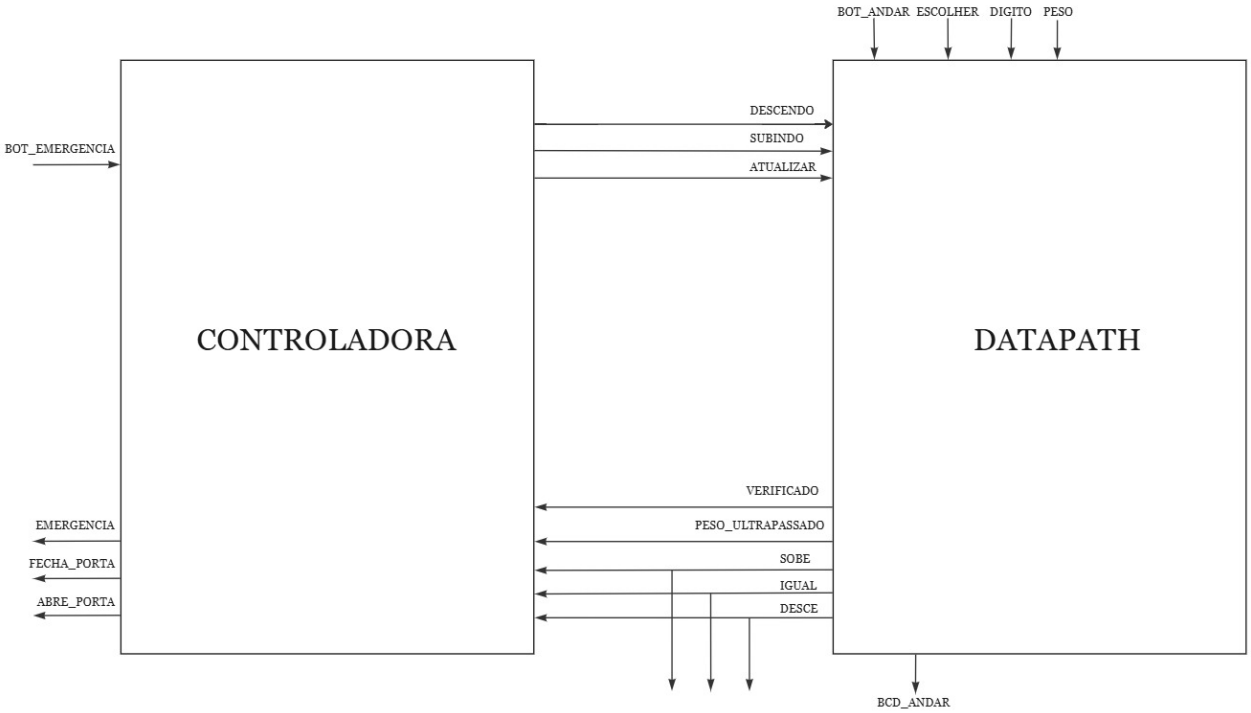

Figura 5: Diagrama de conexão entre a controladora e o caminho de dados.

2 Implementação em linguagem de descrição de hardware

Esta seção apresenta a implementação dos componentes do projeto em VHDL, acompanhada das visualizações e simulações geradas.

2.1 Comparador

Descrição e implementação

Esta implementação **comportamental** do comparador é fundamental no caminho de dados, pois permite determinar se o andar de destino corresponde ao andar atual do elevador. Com base nessa comparação, o sistema pode decidir se deve acionar a abertura das portas e seguir para os próximos estados do projeto. O comparador possuí 16 *bits* e compara dois sinais de entrada A e B, acompanhado de um *enable* que controla seu funcionamento.

Este comparador também é instanciado para comparar a senha (14 *bits*) e o peso (10 *bits*).

```vhdl
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Comparador is
    generic (
        W : natural := 16
    );
    port (
        A       : in  std_logic_vector(W-1 downto 0);
        B       : in  std_logic_vector(W-1 downto 0);
        Enable  : in  std_logic;
        Maior   : out std_logic;
        Menor   : out std_logic;
        Igual   : out std_logic
    );
end Comparador;

architecture Behavioral of Comparador is
begin
    process (A, B, Enable)
    begin
        if Enable = '1' then
            if signed(A) > signed(B) then
                Maior <= '1';
            else
                Maior <= '0';
            end if;

            if signed(A) < signed(B) then
                Menor <= '1';
            else
                Menor <= '0';
            end if;

            if signed(A) = signed(B) then
                Igual <= '1';
            else
                Igual <= '0';
```

```
40              end if;
41          else
42              Maior <= '0';
43              Menor <= '0';
44              Igual <= '0';
45          end if;
46      end process;
47  end Behavioral;
```

Código 1: Implementação do comparador de 16 *bits* em *VHDL*.

Visualização

A visualização a seguir é gerada na sessão *RTL Viewer* do *Quartus*, nela podemos acompanhar os sinais de entrada e saída do comparador implementado.

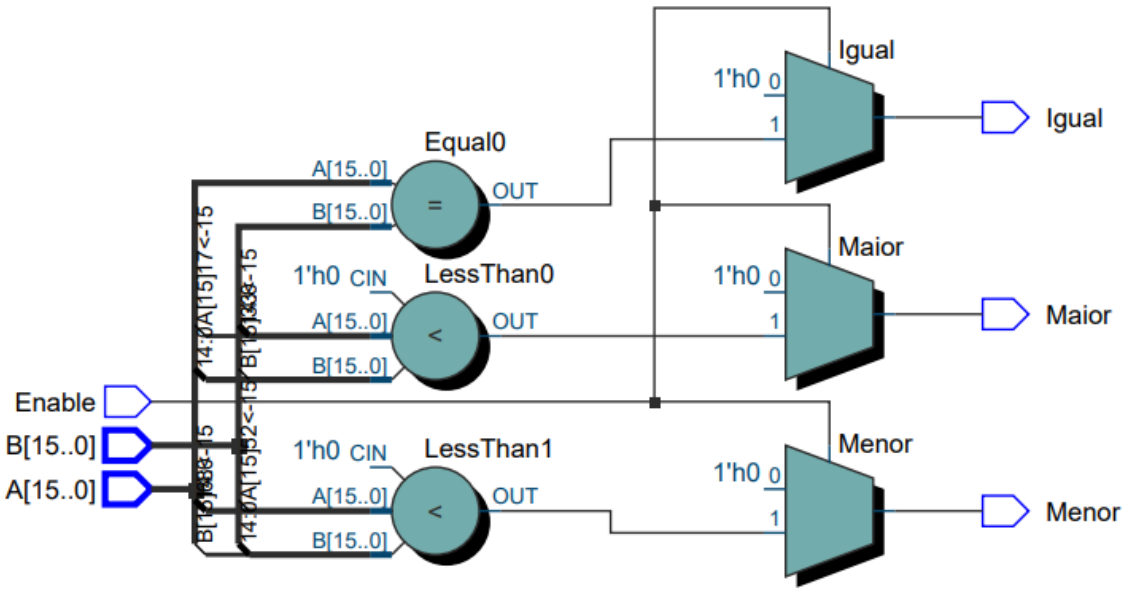

Figura 6: Visualização do RTL *Viewer* da implementação do comparador.

Simulação

O teste de funcionamento realizado no comparador é mostrado abaixo. Na etapa (1) o sinal de *enable* é ativado e o componente começa a operar normalmente, como A e B são iguais o sinal de igualdade está em 1 enquanto os demais sinais de saída estão em alto nível, em (2) A é maior que B tornando o sinal de saída `Maior` igual a 1. Na etapa (3) a desigualdade se inverte e em (4) quando o *enable* é colocado em 0 o componente deixa de operar como um comparador de sinais.

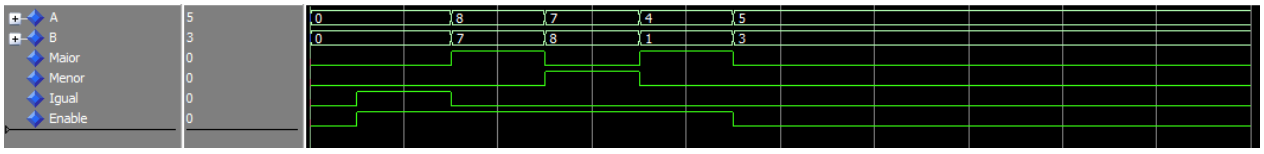

Figura 7: Simulação do comparador implementado na ferramenta *ModelSim*.

2.2 Multiplexador

Descrição e implementação

O multiplexador é responsável por converter o sinal de entrada proveniente de um registrador que armazena os andares solicitados. A seleção do andar é realizada com base em um sinal que representa o número do andar em formato binário. O multiplexador, portanto, traduz o sinal de um andar solicitado em um valor binário: 1 (indicando que o andar foi chamado) ou 0 (indicando que o andar não foi chamado).

O paradigma seguido também foi o paradigma **comportamental**. O `Mux` tem seu código mostrado abaixo. A entrada `Input_vector` advinda de um registrador de 16 *bits* e a seleção ocorre a partir de uma entrada de 4 *bits*.

```
1  library IEEE;
2  use IEEE.std_logic_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity Mux is
6    generic (
7      W : integer := 16;
8      Log2W : integer := 4
9    );
10   port (
11     Input_vector : in  std_logic_vector(W-1 downto 0);
12     Selector     : in  std_logic_vector(log2W-1 downto 0);
13     Output_value : out std_logic
14   );
15 end Mux;
16
17 architecture Behavioral of Mux is
18   begin
19     process(Input_vector, Selector)
20     begin
21       if to_integer(unsigned(Selector)) < W then
22         Output_value <= Input_vector(to_integer(unsigned(Selector)));
23       else
24         Output_value <= 'Z';
25       end if;
26   end process;
27 end Behavioral;
```

Código 2: Implementação do multiplexador em *VHDL*.

Visualização

A visualização gerada do componente é bastante simples, refletindo diretamente suas entradas e sua saída.

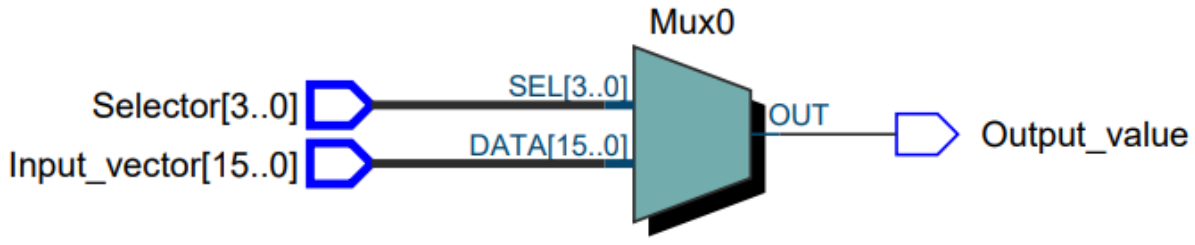

Figura 8: Visulização do RTL *Viewer* do multiplexador implementado.

Simulação

O teste realizado no componente consiste em aplicar uma *string* binária de entrada e conferir se o índice da *string* está em sinal baixo ou não, que é equivalente a verificar se um determinado andar foi ou não solicitado no registrador da fila de andares.

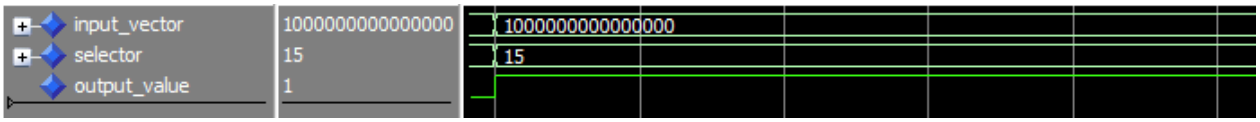

Figura 9: Sim, o décimo quinto andar foi solicitado, portanto quando o seletor está em 15 a saída deve ser 1.

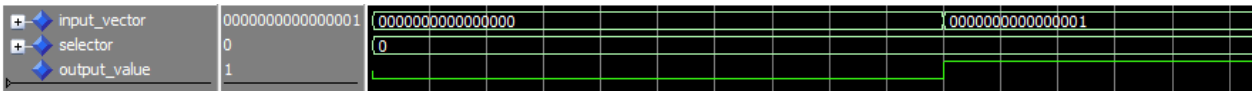

Figura 10: Inicialmente o andar térreo não havia sido chamado, posteriormente ele foi chamado e o *bit* menos significativo se torna 1 e então a saída do multiplexador é 1 já que ele está verificando tal posição no registrador da fila de andares.

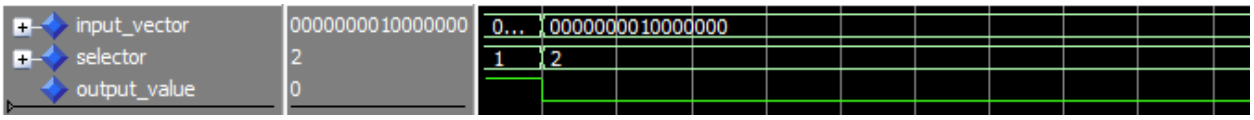

Figura 11: Não, o segundo andar não foi solicitado, apenas o sétimo andar foi solicitado na *string* 000000010000000.

2.3 Incrementador e decrementador 0 a 15

Descrição e implementação

Este componente é capaz de realizar contagens crescentes ou decrescentes no intervalo de inteiros não negativos $(0, 1, 2, ..., 15)$, o que requer o uso de 4 *bits*. Ele é sensível às bordas de subida dos sinais de *clock*. O contador é essencial para a contagem de andares e para a varredura no registrador de andares. A entrada `Control` gerencia a operação de incremento e decremento: quando seu valor é 1, o contador opera em modo crescente, enquanto quando é 0, opera em modo decrescente. Ele também inverte a ordem sempre que chega nos limites extremos de contagem.

```
1  library IEEE;
2  use IEEE.std_logic_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity Contador is
6      port (
7          Clock   : in  std_logic;
8          Reset   : in  std_logic;
9          Control : in  std_logic;
10         Start   : in  std_logic_vector(3 downto 0);
11         Enable  : in  std_logic;
12         Count   : out std_logic_vector(3 downto 0)
13     );
14 end Contador;
15
```

```vhdl
architecture Behavioral of Contador is
    signal cnt : std_logic_vector(3 downto 0) := (others => '0');
    signal W_int : integer := 15;
    signal counting_up : boolean := true;
    process (Clock, Reset)
    begin
        if Reset = '1' then
            cnt <= Start;
            counting_up <= Control = '1';
        elsif rising_edge(Clock) then
            if Enable = '1' then
                if Control = '1' then
                    if counting_up then
                        if to_integer(unsigned(cnt)) < W_int then
                            cnt <= std_logic_vector(unsigned(cnt) + 1);
                        else
                            counting_up <= false;
                            cnt <= std_logic_vector(unsigned(cnt) - 1);
                        end if;
                    else
                        if to_integer(unsigned(cnt)) > 0 then
                            cnt <= std_logic_vector(unsigned(cnt) - 1);
                        else
                            counting_up <= true;
                            cnt <= std_logic_vector(unsigned(cnt) + 1);
                        end if;
                    end if;
                elsif Control = '0' then
                    if counting_up then
                        if to_integer(unsigned(cnt)) > 0 then
                            cnt <= std_logic_vector(unsigned(cnt) - 1);
                        else
                            counting_up <= false;
                            cnt <= std_logic_vector(unsigned(cnt) + 1);
                        end if;
                    else
                        if to_integer(unsigned(cnt)) < W_int then
                            cnt <= std_logic_vector(unsigned(cnt) + 1);
                        else
                            counting_up <= true;
                            cnt <= std_logic_vector(unsigned(cnt) - 1);
                        end if;
                    end if;
                end if;
            end if;
        end if;
    end process;

    Count <= cnt;
end Behavioral;
```

Código 3: Implementação do incrementador e decrementador em *VHDL*.

Visualização

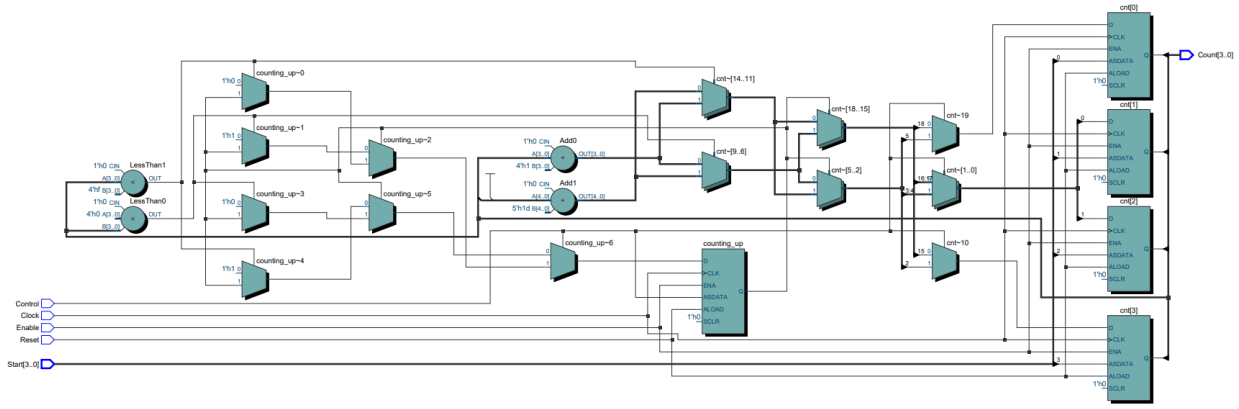

Figura 12: Visualização RTL do decrementador e incrementador implementado.

Simulação

O teste realizado no Contador consiste em testar as lógicas de direções implementadas, verificando os diferentes métodos de funcionamento, com o controle de incrementação/decrementação pelo Control e pelos limites extremos.

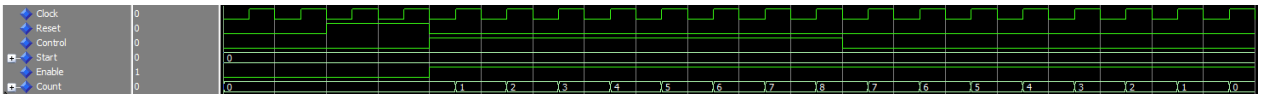

Figura 13: O Contador sobe de 0 até 8 com Control = 1, e descresce de 8 até 0 quando o Control inverte.

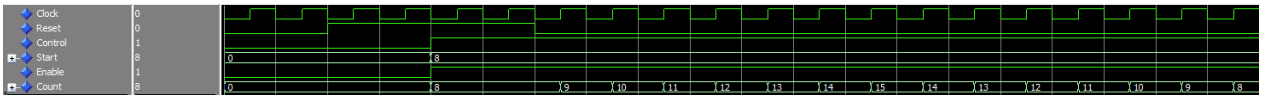

Figura 14: O Contador começa de 8, com Start = 8 quando Reset = 1 e começa a subir até o limite = 15 quando volta a descrescer.

2.4 Registrador da fila de andares de 16 *bits*

Descrição e implementação

O registrador da fila de andares, denominado `Reg_FilaAndar`, armazena em seus índices um sinal *booleano*. Cada *flip-flop* do registrador corresponde a um sinal, estabelecendo uma bijeção na qual o índice 0 representa o térreo e o índice 16 representa o décimo quinto andar. Este componente é responsável por armazenar os andares solicitados em uma espécie de fila. Ele está conectado diretamente ao multiplexador, onde a seleção de um determinado índice ocorre por meio da representação binária do número correspondente ao andar.

```
1  library IEEE;
2  use IEEE.std_logic_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity Reg_FilaAndar is
6      generic (
7          W     : natural := 16;
```

```
        log2W : natural := 4
    );
    port (
        Chamar    : in  std_logic;
        Limpar    : in  std_logic;
        Carregar  : in  std_logic;
        Clock     : in  std_logic;
        Seletor   : in  std_logic_vector(log2W - 1 downto 0);
        SeletorClear : in  std_logic_vector(log2W - 1 downto 0);
        Saida     : out std_logic_vector(W-1 downto 0)
  );
end entity Reg_FilaAndar;

architecture Behavioral of Reg_FilaAndar is
  signal reg : std_logic_vector(W-1 downto 0) := (others => '0');
begin
  process (Clock)
  begin
    if rising_edge(Clock) then
      if Carregar = '1' then
        if Chamar = '1' then
          reg(to_integer(unsigned(Seletor))) <= '1';
        elsif Limpar = '1' then
          reg(to_integer(unsigned(SeletorClear))) <= '0';
        end if;
      end if;
    end if;
  end process;

  Saida <= reg;

end architecture Behavioral;
```

Código 4: Implementação do registrador da fila de andares em *VHDL*.

Visualização

A visualização do projeto revela a lógica combinacional incorporada, que opera antes do sinal ser efetivamente armazenado no registrador de 16 índices. Essa lógica adicional é essencial para que o registrador possa armazenar as chamadas nos índices desejados corretamente.

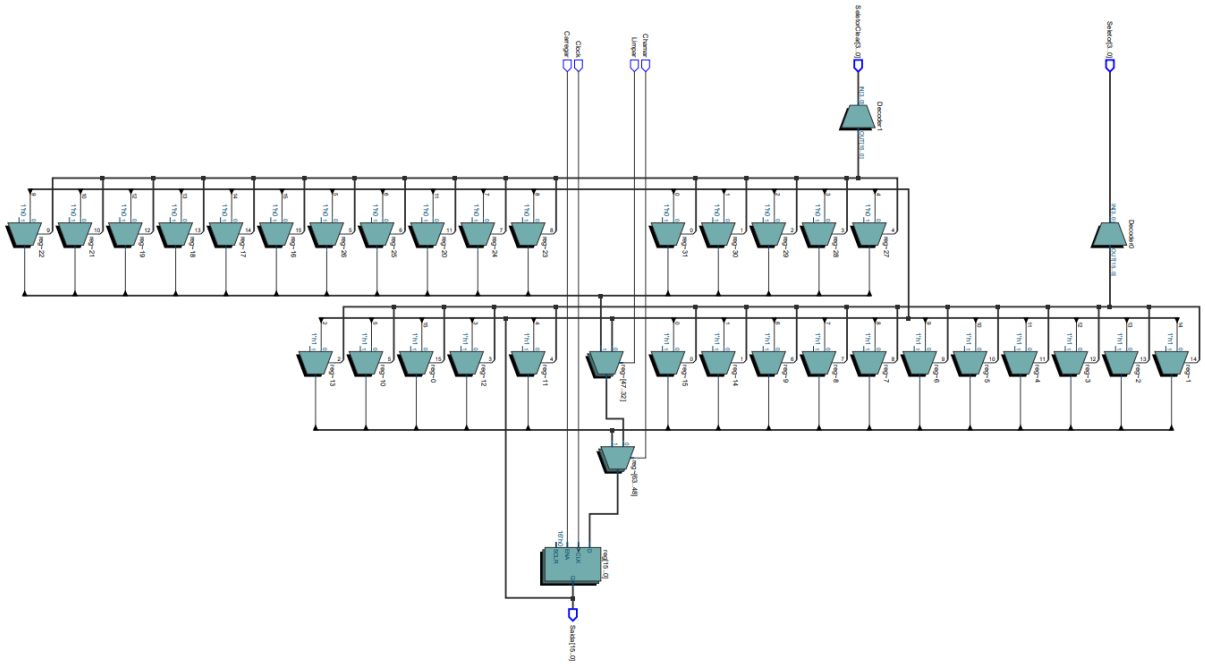

Figura 15: Visualização RTL do registrador da fila de andares implementado.

Simulação

Na simulação do teste de funcionamento do registrador da fila de andares, foram avaliadas a chamada (escrita) e o acesso (sobrescrita) aos andares nos 16 índices.

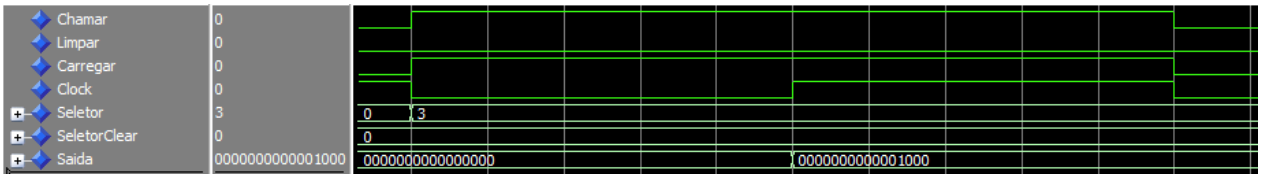

Figura 16: Simulação, o terceiro andar é solicitado e é então salvo no registrador da fila de andares.

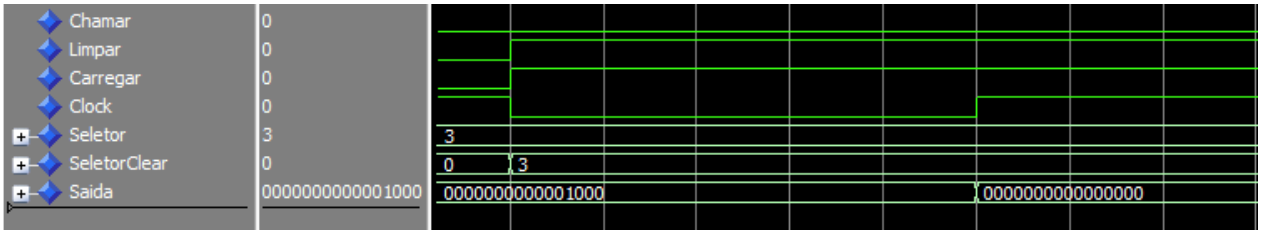

Figura 17: Posteriormente o terceiro andar é acessado e então é retirado da fila de andares.

2.5 Registrador de peso

Descrição e implementação

O Registrador de Peso é o componente responsável por implementar um Registrador de 10 bits reponsável por armazenar e repassar o valor numérico do peso detectado no elevador.

```vhdl
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Reg_Peso is
    generic (
        W : natural := 10
    );
    port (
        Carregar : in std_logic;
        Clock    : in std_logic;
        Dado     : in std_logic_vector(W-1 downto 0);
        Saida    : out std_logic_vector(W-1 downto 0)
    );
end entity Reg_Peso;

architecture Behavioral of Reg_Peso is
    signal reg : std_logic_vector(W-1 downto 0) := (others => '0');
begin
    process (Clock)
    begin
        if rising_edge(Clock) then
            if Carregar = '1' then
                reg <= Dado;
            end if;
        end if;
    end process;

    Saida <= reg;
end architecture Behavioral;
```

Código 5: Implementação do registrador de peso em *VHDL*.

Visualização

A visualização do projeto revela a lógica combinacional incorporada, que inclui um registrador de 10 bits que simplesmente armazena a entrada recebida quando for permitido

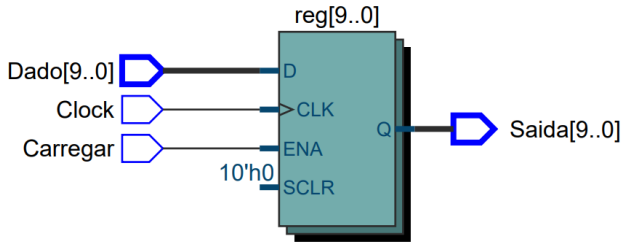

Figura 18: Visualização RTL do registrador da fila de andares implementado.

Simulação

O teste de funcionamento realizado no Reg_Peso é mostrado abaixo. Na etapa (1) a entrada de valor 10 *enable* é carregada na memória e o componente começa a retornar o valor contido na memória a partir do próximo Clock, em (2) *A* o valor de 149 é carregado na memória, sobreescrevendo o anterior. Na etapa (3) o valor de 767 é carregado na memória.

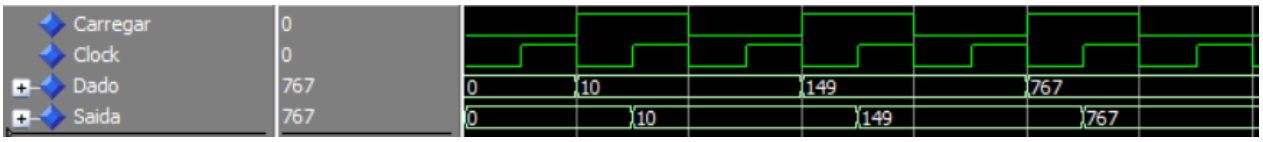

Figura 19: Simulação do Reg_Peso implementado na ferramenta *ModelSim*.

2.6 Registrador de senha

Descrição e implementação

O Reg_Senha é o componente responsável por implementar um Registrador Shifter, com 16 bits, sendo estes particionados em 4 grupos de 4 bits e executando uma função Shift para a direita conforme a senha é recebida pelo Registrador.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity Reg_Senha is
7      Port (
8          clk      : in  std_logic;
9          reset    : in  std_logic;
10         carregar: in  std_logic;
11         digito   : in  std_logic_vector (3 downto 0);
12         senha    : out std_logic_vector (15 downto 0)
13     );
14 end Reg_Senha;
15
16 architecture Behavioral of Reg_Senha is
17     signal sig_senha : std_logic_vector (15 downto 0) := (others => '0');
18     signal last_carregar : std_logic := '0';
19     signal count : integer := 0;
20 begin
21
22 process(clk, reset)
23 begin
24     if reset = '1' then
25         sig_senha <= (others => '0');
26         count <= 0;
27     elsif rising_edge(clk) then
28         if carregar = '1' and last_carregar = '0' then
29             if count < 4 then
30                 sig_senha(15 downto 12) <= sig_senha(11 downto 8);
31                 sig_senha(11 downto 8) <= sig_senha(7 downto 4);
32                 sig_senha(7 downto 4) <= sig_senha(3 downto 0);
33                 sig_senha(3 downto 0) <= digito;
34                 count <= count + 1;
35             end if;
36         end if;
37         last_carregar <= carregar;
38     end if;
39 end process;
40
```

```
41 senha <= sig_senha;
42
43 end Behavioral;
```

Código 6: Implementação do registrador de senha *VHDL*.

Visualização

A visualização do projeto revela a lógica combinacional incorporada, que opera deslocando a senha em 4 bits até essa ser completamente preenchida. Essa lógica é importante para implementar um teclado onde 1 dígito é digitado por vez.

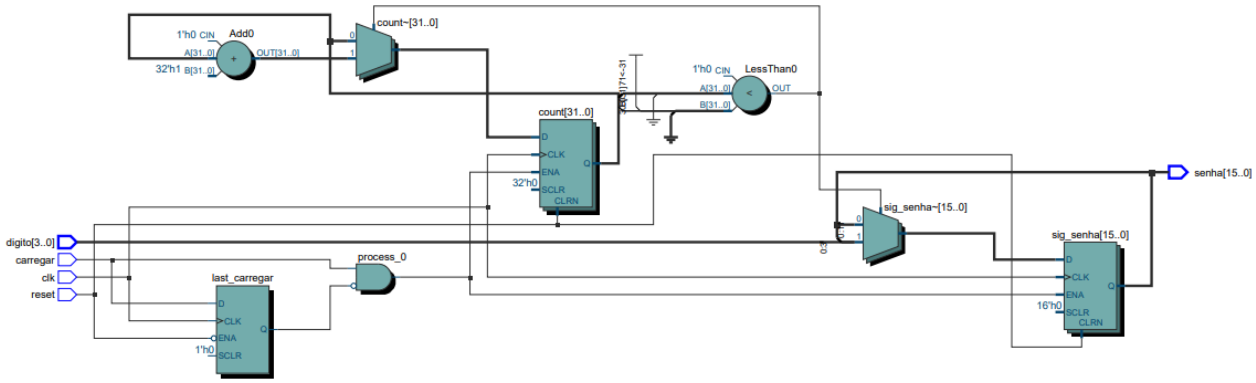

Figura 20: Visualização do registrador de senha.

Simulação

Na simulação do teste de funcionamento do registrador de senha, foram inseridos os dígitos um a um, e a saída do componente é a concatenação dos dígitos inseridos.

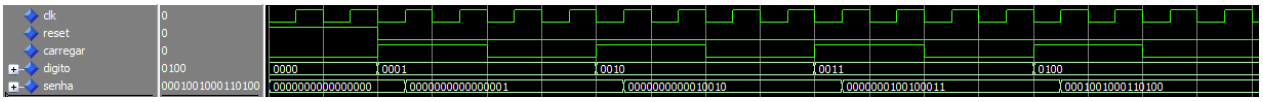

Figura 21: Simulação, os dígitos são enviados um a um para o registrador.

---

3 Bibliografia

Referências

VAHID, F. *"Digital Design: A Systems Approach"*. 2. ed. Hoboken: Wiley, 2010.

MEALY, B and TAPPERO, F. *"Free Range VHDL book"*. 1. ed.
Disponível em: https://github.com/fabriziotappero/Free-Range-VHDL-book?tab=readme-ov-file;
Acesso em: 25 de julho de 2024.