Relatório: Atividade 3

Filipe Augusto Parreira Almeida

RA: 2320622

17 de abril de 2024

Conteúdo

1 Resumo

No presente relatório, é abordado os conceitos teóricos de lógica reconfigurável e sua aplicação prática em um projeto utilizando a linguagem de descrição de hardware VHDL. O foco é implementar um contador em um FPGA com a capacidade de exibir valores de 0 a 99 em dois displays de 7 segmentos, controlados por oito switches — quatro para cada display — para ajustar os valores apresentados em cada display. Este projeto explora a ideia de código concorrente, característica fundamental da programação em VHDL, que busca permitir a execução eficiente e simultânea de múltiplas operações.

2 Introdução

Um código *VHDL* pode ser concorrente ou sequencial, tais implementações de lógica vai depender exclusivamente da demanda do projeto. Para circuitos combinacionais utiliza-se o **código concorrente**, onde temos que a **saída** do sistema **depende exclusivamente da entrada atual**, não possui realimentação; já os códigos do tipo **sequencial** temos que ele **serve tanto para circuitos combinacionais quanto para circuitos sequenciais**, onde temos realimentação, ou seja, a saída atual depende de valores anteriores. Códigos sequenciais fogem do escopo deste relatório, portanto é abordado somente o código concorrente.

2.1 Código Concorrente

Código concorrente podem ser estruturados com as instruções ***WHEN***, ***SELECT*** e ***GENERATE***, juntamente com os operadores padrão da linguagem de descrição de hardware **VHDL**. Para o escopo do presente relatório é abordado somente as instruções ***WHEN*** e ***SELECT***.

2.1.1 WHEN

Dentre as instruções de um código concorrente, ela é a **mais simples**. Ela funciona como uma estrutura de atribuição condicional, onde um **valor alvo** recebe determinado valor dependendo de uma certa **condição**. Abaixo encontra-se sua sintaxe, e também um exemplo de sua utilização:

```
assignment_expression WHEN conditions ELSE
    assignment_value WHEN conditions ELSE
    ...;
```

Código 1: Sintaxe WHEN

```
outp <= "0000" WHEN reset='0' OR inp1="00" ELSE
        "1111" WHEN inpt2='0' AND inp3='1' ELSE
        "ZZZZ";
```

Código 2: Exemplo WHEN

2.1.2 SELECT

A instrução ***SELECT***, funciona como uma estrutura de controle de fluxo, comparando com a linguagem de programação C, ela se assemelha a estrutura do comando **switch**, onde é dada uma expressão (identificador) e testa-se todos os casos para executar um comando. No caso do *VHDL*, temos que a instrução ***SELECT*** atribui determinado valor a um valor alvo de acordo com o valor contido no identificador. Para isso testa-se todos os casos, portanto é uma boa prática utilizar o comando ***OTHERS***, para generalizar. Segue a sintaxe da instrução e um exemplo para melhor entendimento.

```
WITH identifier SELECT
    assignment_expression WHEN values,
        assignment_values WHEN values,
        ...;
```

Código 3: Sintaxe SELECT

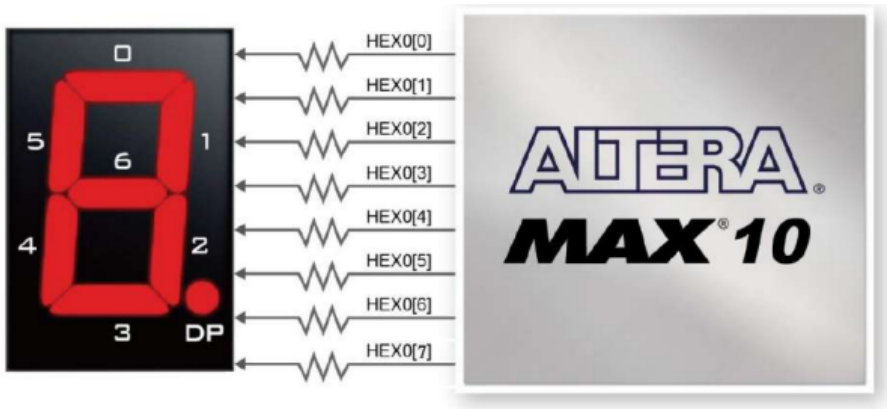
```
WITH sel SELECT
    output <= inp1 WHEN 0,
              inp2 WHEN 1,
              inp3 WHEN OTHERS;
```

Código 4: Exemplo SELECT

2.2 Display de 7 segmentos

Para o presente relatório é utilizado dois dos **seis *displays* de 7 segmentos** da placa de desenvolvimento ***DE10-Lite***. Cada um deles possui, como o próprio nome indica, sete segmentos, onde podemos manipular quais serão ligados ou não. Esta manipulação é feita aplicando nível lógico **baixo para ligar**, e **alto para desligar**. Cada segmento é indexado de **0 até 6**, e também possui a indicação de ponto flutuante (ou decimal point, DP), como é indicado na Figura 1. A Figura 2 e a Figura 3 mostram a lista de pinos para cada segmento.

Figura 1: Diagrama de Blocos - Display 7 Segmentos



Fonte: (TERASIC TECHNOLOGIES, 2020)

Figura 2: Tabela de Pinos Display 7 Segmentos - Primeira Parte

Signal Name	FPGA Pin No.	Description	I/O Standard
HEX00	PIN_C14	Seven Segment Digit 0[0]	3.3-V LVTTL
HEX01	PIN_E15	Seven Segment Digit 0[1]	3.3-V LVTTL
HEX02	PIN_C15	Seven Segment Digit 0[2]	3.3-V LVTTL
HEX03	PIN_C16	Seven Segment Digit 0[3]	3.3-V LVTTL
HEX04	PIN_E16	Seven Segment Digit 0[4]	3.3-V LVTTL
HEX05	PIN_D17	Seven Segment Digit 0[5]	3.3-V LVTTL
HEX06	PIN_C17	Seven Segment Digit 0[6]	3.3-V LVTTL
HEX07	PIN_D15	Seven Segment Digit 0[7], DP	3.3-V LVTTL
HEX10	PIN_C18	Seven Segment Digit 1[0]	3.3-V LVTTL
HEX11	PIN_D18	Seven Segment Digit 1[1]	3.3-V LVTTL
HEX12	PIN_E18	Seven Segment Digit 1[2]	3.3-V LVTTL
HEX13	PIN_B16	Seven Segment Digit 1[3]	3.3-V LVTTL

Fonte: (TERASIC TECHNOLOGIES, 2020)

Figura 3: Tabela de Pinos Display 7 Segmentos - Segunda Parte

HEX14	PIN_A17	Seven Segment Digit 1[4]	3.3-V LVTTL
HEX15	PIN_A18	Seven Segment Digit 1[5]	3.3-V LVTTL
HEX16	PIN_B17	Seven Segment Digit 1[6]	3.3-V LVTTL
HEX17	PIN_A16	Seven Segment Digit 1[7] , DP	3.3-V LVTTL
HEX20	PIN_B20	Seven Segment Digit 2[0]	3.3-V LVTTL
HEX21	PIN_A20	Seven Segment Digit 2[1]	3.3-V LVTTL
HEX22	PIN_B19	Seven Segment Digit 2[2]	3.3-V LVTTL
HEX23	PIN_A21	Seven Segment Digit 2[3]	3.3-V LVTTL
HEX24	PIN_B21	Seven Segment Digit 2[4]	3.3-V LVTTL
HEX25	PIN_C22	Seven Segment Digit 2[5]	3.3-V LVTTL
HEX26	PIN_B22	Seven Segment Digit 2[6]	3.3-V LVTTL
HEX27	PIN_A19	Seven Segment Digit 2[7] , DP	3.3-V LVTTL
HEX30	PIN_F21	Seven Segment Digit 3[0]	3.3-V LVTTL
HEX31	PIN_E22	Seven Segment Digit 3[1]	3.3-V LVTTL
HEX32	PIN_E21	Seven Segment Digit 3[2]	3.3-V LVTTL
HEX33	PIN_C19	Seven Segment Digit 3[3]	3.3-V LVTTL
HEX34	PIN_C20	Seven Segment Digit 3[4]	3.3-V LVTTL
HEX35	PIN_D19	Seven Segment Digit 3[5]	3.3-V LVTTL
HEX36	PIN_E17	Seven Segment Digit 3[6]	3.3-V LVTTL
HEX37	PIN_D22	Seven Segment Digit 3[7] , DP	3.3-V LVTTL
HEX40	PIN_F18	Seven Segment Digit 4[0]	3.3-V LVTTL
HEX41	PIN_E20	Seven Segment Digit 4[1]	3.3-V LVTTL
HEX42	PIN_E19	Seven Segment Digit 4[2]	3.3-V LVTTL
HEX43	PIN_J18	Seven Segment Digit 4[3]	3.3-V LVTTL
HEX44	PIN_H19	Seven Segment Digit 4[4]	3.3-V LVTTL
HEX45	PIN_F19	Seven Segment Digit 4[5]	3.3-V LVTTL
HEX46	PIN_F20	Seven Segment Digit 4[6]	3.3-V LVTTL
HEX47	PIN_F17	Seven Segment Digit 4[7] , DP	3.3-V LVTTL
HEX50	PIN_J20	Seven Segment Digit 5[0]	3.3-V LVTTL
HEX51	PIN_K20	Seven Segment Digit 5[1]	3.3-V LVTTL
HEX52	PIN_L18	Seven Segment Digit 5[2]	3.3-V LVTTL
HEX53	PIN_N18	Seven Segment Digit 5[3]	3.3-V LVTTL
HEX54	PIN_M20	Seven Segment Digit 5[4]	3.3-V LVTTL
HEX55	PIN_N19	Seven Segment Digit 5[5]	3.3-V LVTTL
HEX56	PIN_N20	Seven Segment Digit 5[6]	3.3-V LVTTL
HEX57	PIN_L19	Seven Segment Digit 5[7] , DP	3.3-V LVTTL

Fonte: (TERASIC TECHNOLOGIES, 2020)

3 Implementação

Para a implementação em si, foi utilizada a linguagem de descrição de hardware *VHDL*, juntamente com os conceitos de **programação concorrente**. Mais precisamente, por escolha, foi utilizado a instrução ***SELECT***, para que fosse realizada a atribuição de valores, que resultaram na representação dos algarismos nos *displays*.

3.1 Declaração de Bibliotecas/Pacotes

A declaração de bibliotecas/pacotes até então está sendo a mesma para todas as atividades práticas, que é a biblioteca ***IEEE*** e o pacote ***STD_LOGIC_1164***, permitindo o uso do tipo ***STD_LOGIC_VECTOR***. Segue o trecho da declaração de bibliotecas/pacotes:

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
```

Código 5: Declaração de Bibliotes/Pacotes

3.2 Entidade

Definida como ***atividade_3*** temos que na entidade é definido as duas entradas e as duas saídas, que representam respectivamente os conjuntos de *switches* e os dois *displays* de 7 segmentos. A entradas e saídas estão sendo declaradas como o tipo ***STD_LOGIC_VECTOR***, que nada mais é do que um **vetor de bits** (para a presente situação); para as entradas temos que cada uma é um vetor de tamanho **4**, ou seja, cada bit deste vetor representa o estado de um *switch*; e temos que cada uma das saídas é um vetor de tamanho **7**, onde cada bit do vetor é o estado de um segmento do **display**. Segue o trecho de código referente a declaração da entidade:

```
1 ENTITY atividade_3 IS PORT (
2   ssd_in_1, ssd_in_2: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
3   ssd_out_1, ssd_out_2: OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
4 );
5 END atividade_3;
```

Código 6: Declaração da Entidade

3.3 Arquitetura

A arquitetura, definida como *main*, contém somente o código concorrente que descreve o funcionamento desejado. A lógica foi feita utilizando a instrução concorrente ***SELECT***, onde, dada a situação, analisa o valor de cada entrada e dependendo deste valor atribui um outro valor para uma saída. Mais precisamente, caso se queira exibir o valor 5 em um dos *displays*, é identificado o valor 5 em binário de acordo com o estado de cada um dos *switches*, no caso, o valor 5 é descrito por **"0101"**, então é atribuído o valor **"0010010" ao vetor de saída**, este valor representa o formato do algarismo 5 no *display* de 7 segmentos. Como temos **4 bits**, é possível representar valores até o 16, porém, para o escopo deste relatório, aborda-se somente **um *display* para cada conjunto de *switches***, portanto, para **valores maiores que 9 é exibido o caractere 'E'**, indicando erro, é possível visualizar esta implementação com a condição ***OTHERS*** da instrução ***SELECT***. Segue o trecho de código referente à arquitetura:

```
ARCHITECTURE main OF atividade_3 IS

  BEGIN

    WITH ssd_in_1 SELECT
      ssd_out_1 <= "1000000" WHEN "0000",
              "1111001" WHEN "0001",
              "0100100" WHEN "0010",
              "0110000" WHEN "0011",
              "0011001" WHEN "0100",
              "0010010" WHEN "0101",
              "0000010" WHEN "0110",
              "1111000" WHEN "0111",
              "0000000" WHEN "1000",
              "0010000" WHEN "1001",
              "0000110" WHEN others;
      WITH ssd_in_2 SELECT
      ssd_out_2 <= "1000000" WHEN "0000",
              "1111001" WHEN "0001",
              "0100100" WHEN "0010",
              "0110000" WHEN "0011",
              "0011001" WHEN "0100",
              "0010010" WHEN "0101",
              "0000010" WHEN "0110",
              "1111000" WHEN "0111",
              "0000000" WHEN "1000",
              "0010000" WHEN "1001",
              "0000110" WHEN others;
END ARCHITECTURE;
```

Código 7: Declaração da Arquitetura

3.4 Algoritmo Completo e Diagrama RTL

Abaixo encontra-se então todo o código VHDL utilizado para implementação:

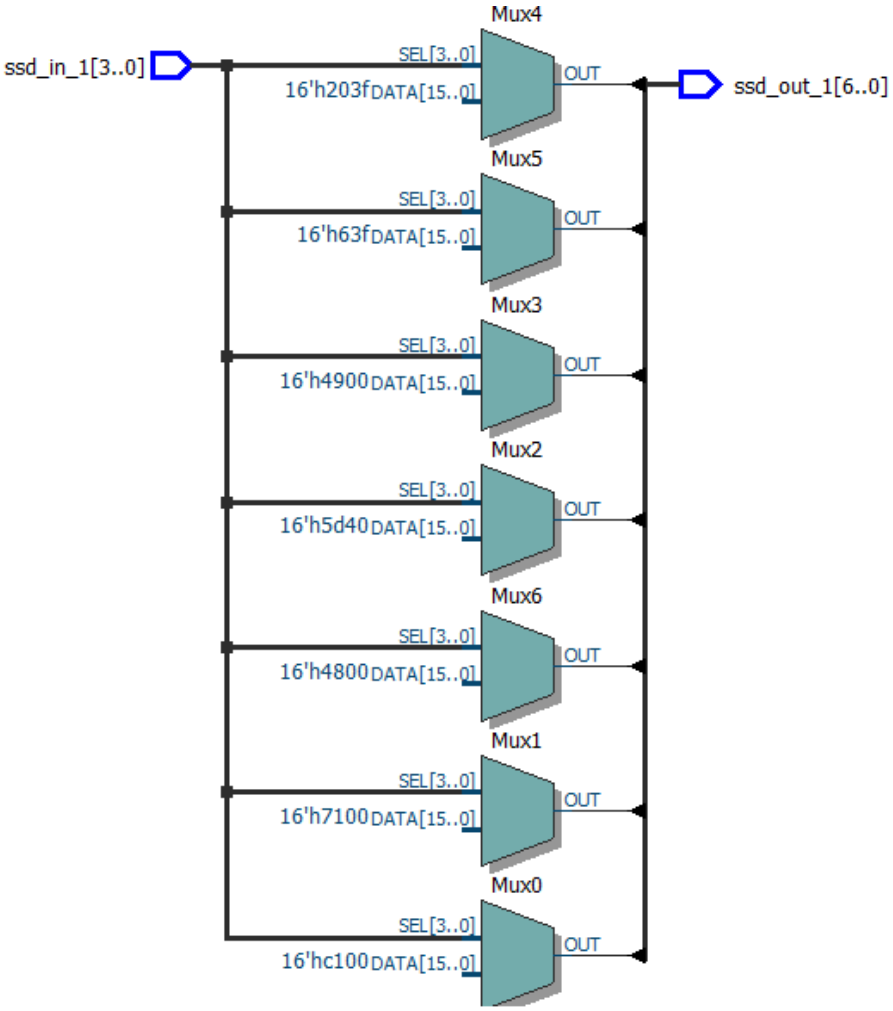
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY atividade_3 IS PORT (
  ssd_in_1, ssd_in_2: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
  ssd_out_1, ssd_out_2: OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
);
END atividade_3;

ARCHITECTURE main OF atividade_3 IS

  BEGIN

    WITH ssd_in_1 SELECT
      ssd_out_1 <= "1000000" WHEN "0000",
              "1111001" WHEN "0001",
```

```
17                "0100100" WHEN "0010",
18                "0110000" WHEN "0011",
19                "0011001" WHEN "0100",
20                "0010010" WHEN "0101",
21                "0000010" WHEN "0110",
22                "1111000" WHEN "0111",
23                "0000000" WHEN "1000",
24                "0010000" WHEN "1001",
25                "0000110" WHEN others;
26       WITH ssd_in_2 SELECT
27       ssd_out_2 <= "1000000" WHEN "0000",
28                "1111001" WHEN "0001",
29                "0100100" WHEN "0010",
30                "0110000" WHEN "0011",
31                "0011001" WHEN "0100",
32                "0010010" WHEN "0101",
33                "0000010" WHEN "0110",
34                "1111000" WHEN "0111",
35                "0000000" WHEN "1000",
36                "0010000" WHEN "1001",
37                "0000110" WHEN others;
38 END ARCHITECTURE;
```

Código 8: Algoritmo Completo

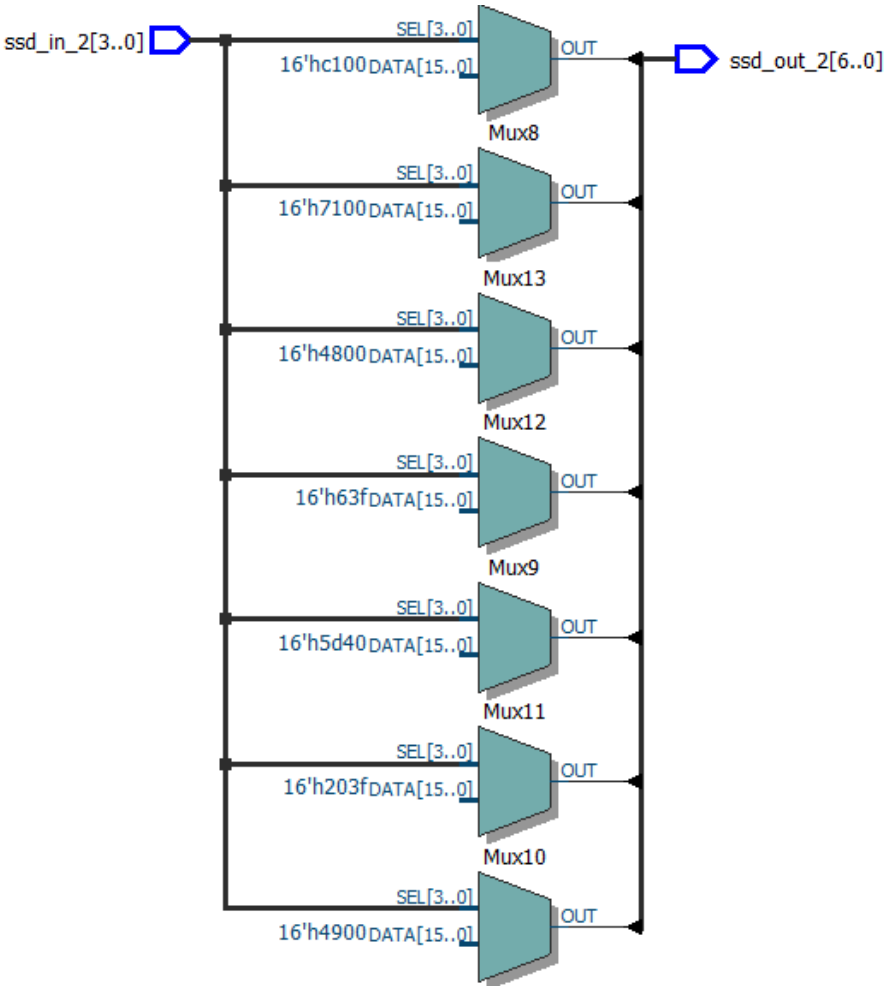
A partir do código *VHDL* acima foi gerado o diagrama de circuitos que o representa:

Figura 4: Entrada e Saída 1



Fonte: Autoria Própria

Figura 5: Entrada e Saída 2

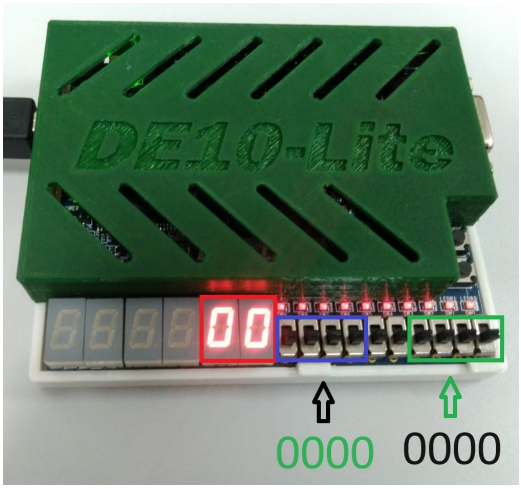


Fonte: Autoria Própria

4 Conclusão

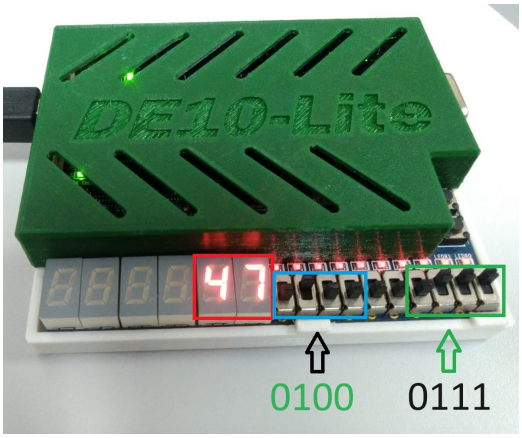
Já realizada a implementação do algoritmo na placa de desenvolvimento, segue as fotos da placa de desenvolvimento, onde é exibido no *display* de 7 segmentos diferentes valores de entrada, vindas dos *switches*.

Figura 6: Entrada "0000"e "0000"



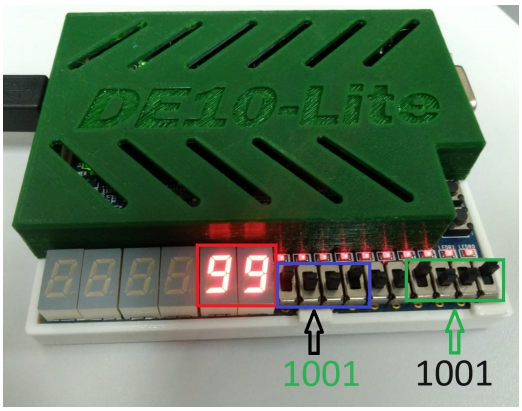
Fonte: Autoria Própria

Figura 7: Entrada "0100"e "0111"



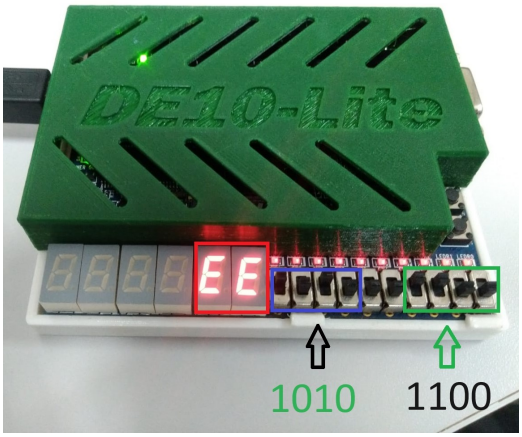
Fonte: Autoria Própria

Figura 8: Entrada "1001"e "1001"



Fonte: Autoria Própria

Figura 9: Entrada "1010"e "1100"



Fonte: Autoria Própria

A partir da análise das imagens, conclui-se que obteve-se **êxito** na implementação da lógica na placa de desenvolvimento para o que foi pedido. Onde é possível notar que para as entradas, foram geradas as saídas esperadas.

Referências

TERASIC TECHNOLOGIES. **Manual da Placa de Desenvolvimento DE10-Lite**. [S.l.], 2020. Disponível em: https://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=China&No=1021&FID=a13a2782811152b477e60203d34b1baa.