Relatório: Atividade 4

Filipe Augusto Parreira Almeida

RA: 2320622

2 de maio de 2024

Conteúdo

1 Resumo **2**

2 Introdução **3**
2.1 Multiplexadores 3
2.2 Comando ***GENERATE*** 4

3 Implementação **6**
3.1 Entidade 6
3.2 Arquitetura 7

4 Conclusão **12**
4.1 4 entradas de 2 bits cada (N = 2 e M = 2) 12
4.2 16 entradas de 8 bits cada (N = 4 e M = 8) 14

Referências **15**

1 Resumo

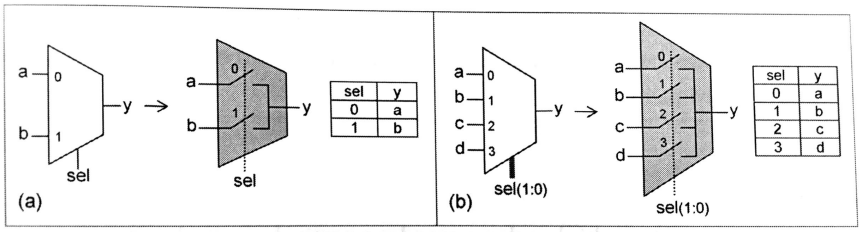
O presente relatório busca descrever o processo para a implementação de um multiplexador genérico, ou seja, fica a critério do usuário definir os parâmetros para este multiplexador. Em todo o processo foi utilizado a linguagem de descrição de hardware, **VHDL**, utilizando como base teórica os fundamentos de códigos concorrentes em **VHDL**, mais precisamente o comando ***GENERATE***. O código escrito foi então implementado na placa de desenvolvimento DE10-LITE, logo, para testes em hardware foi utilizado todos os *switches* da placa de desenvolvimento, onde foram oito para representar a entrada, e por escolha, foram utilizados cinco dos seis *displays* de sete segmentos da placa, quatro para a as entradas, um para cada entrada, e um para saída, todas em decimal.

2 Introdução

2.1 Multiplexadores

Para melhor entendimento da implementação realizada em **VHDL** é necessário que melhor se entenda os conceitos que envolvem os **multiplexadores**. Um multiplexador é um dos circuitos mais populares para manipulação de dados, conceitualmente ele é composto por **N** entradas, uma porta de seleção e uma saída. A porta de seleção define qual será a entrada a ser escolhida para "passar" para a saída. Abaixo é possível visualizar exemplos de multiplexadores de 2 e 4 entradas, com seus circuitos conceituais construídos com chaves, e também suas tabelas-verdade.

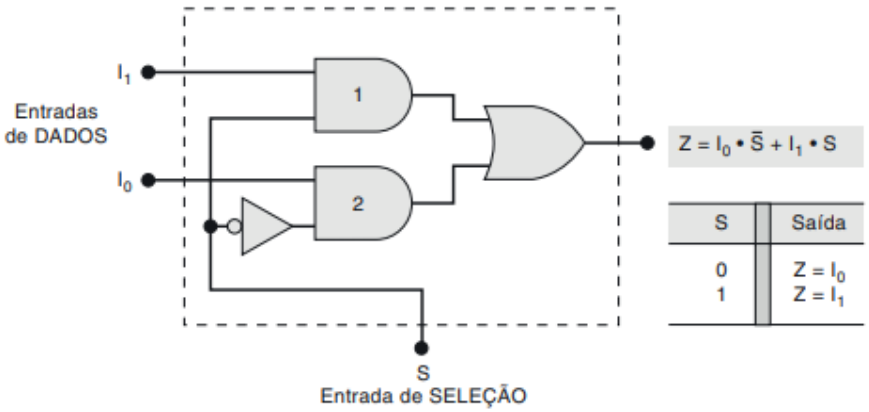
Figura 1: Exemplo de Multiplexador de 2 e 4 entradas



Fonte: (PEDRONI, 2010)

Um meio mais detalhado de representar um multiplexador é descrever seu funcionamento através de **portas lógicas**, onde para um exemplo mais simples, temos um multiplexador de **duas entradas**, onde a porta de seleção é composta por um único bit, logo, dependendo do nível lógico da porta de seleção será determinada qual porta **AND** será habilitada, permitindo então que o dado passe pela porta **OR** e então para a saída. Segue a esquemática deste circuito, juntamente com sua tabela verdade, e a representação algébrica da saída.

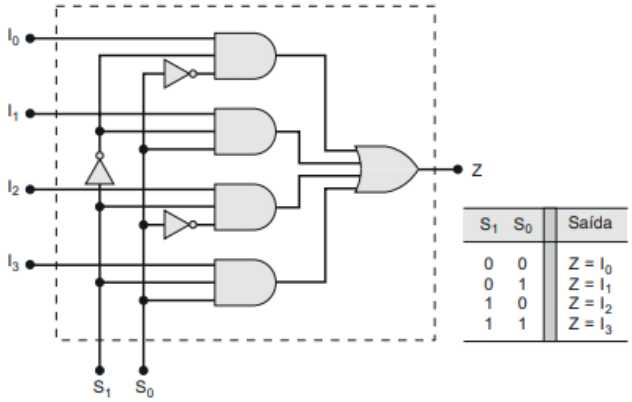
Figura 2: Exemplo de Multiplexador de 2 entradas com tabela-verdade



Fonte: (TOCCI; WIDMER; MOSS, 2011)

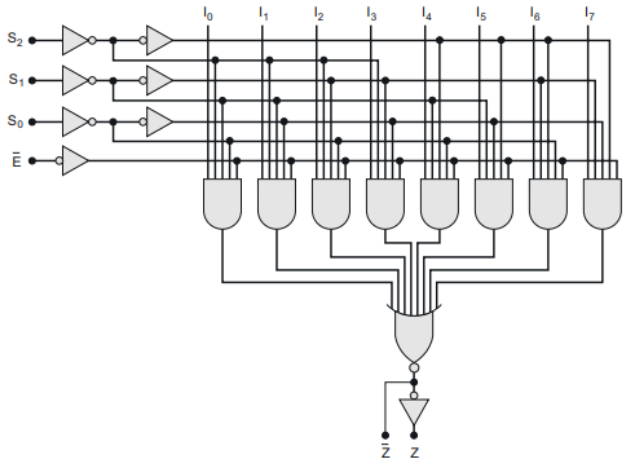
A título de comparação segue a esquemática de um multiplexador de 4 e oito entradas formados por portas lógicas:

Figura 3: Exemplo de Multiplexador de 4 entradas com tabela-verdade



Fonte: (TOCCI; WIDMER; MOSS, 2011)

Figura 4: Exemplo de Multiplexador de 8 entradas



Fonte: (TOCCI; WIDMER; MOSS, 2011)

2.2 Comando *GENERATE*

O comando ***GENERATE*** faz parte do conjunto de comandos que envolvem os conceitos de **código concorrente**, que é utilizado para a construção de circuitos combinacionais, junto do ***GENERATE*** é englobado neste conjunto os comandos ***WHEN*** e ***SELECT***, que foram abordados na atividade anterior. A instrução ***GENERATE*** é equivalente a instrução ***LOOP*** de um código sequencial (assunto que será abordado em atividades futuras) sua sintaxe é definida da seguinte forma:

```
label: FOR identifier IN range GENERATE
    [declarative_part
BEGIN]
    (statements_part)
END GENERATE [label];
```

Código 1: Sintaxe GENERATE

Ele pode ser implementado não somente com o comando *"for"*, mas também com os comandos *"if"* e *"case"*, porém sua forma mais comum é utilizando o ***for***. Fazendo uma com-

paração com a linguagem de programação em **C**, ele funciona de maneira semelhante com o ***"for"***, onde ele atua como um ***loop***, que percorre um vetor,ou então que pode gerar circuitos lógicos, como o exemplo abaixo, que gera circuitos lógicos **AND** e os armazena em um vetor, logo, sua saída é um vetor de 7 elementos **AND**.

```
1 SIGNAL a, b, x: STD_ULOGIC_VECTOR(7 DOWNTO 0);
2 ...
3 gen: FOR i IN 0 TO 7 GENERATE
4     x(i) <= a(i) AND b(7-1);
5 END GENERATE;
```

Código 2: Exemplo GENERATE

3 Implementação

Com os conceitos introdutórios tanto de multiplexadores quanto do que é o comando ***GENERATE*** em **VHDL** já consolidados, é possível então que se combine esses dois conceitos para "construir" um **multiplexador genérico em VHDL**, que é o objetivo deste relatório. Em todos os casos, a declaração de bibliotecas e pacotes permanece a mesma, se comparado aos outros relatórios, portanto esta parte será omitida, sendo apresentada somente no algoritmo completo.

3.1 Entidade

Como requisitos para a implementação deste multiplexador, é necessário que se defina somente **dois parâmetros**, a quantidade de **bits de seleção** e também o **tamanho de cada entrada**. Esses parâmetros são declarados através de constantes genéricas declaradas na entidade, onde **N** representa a quantidade de bits para a porta de seleção e **M** representa o tamanho de cada entrada. Ainda na entidade temos a declaração das **portas**, que é onde parte da implementação lógica do multiplexador genérico é feito. A porta de seleção, declarada como porta de entrada do tipo inteiro, foi definida como sendo um número inteiro de 0 até 2^{N-1}, esse range advém do valor máximo que pode ser representado por **N** bits da porta de seleção; e foi definido como inteiro para uma melhor implementação lógica na parte de arquitetura. Como exemplo, temos que caso seja definido que a porta de entrada tenha **4 bits**, teremos:

$$N = 4$$

$$sel = [0 : (2^4) - 1] = [0 : 15]$$

Temos então que, neste exemplo, ***sel*** assume-se como um número inteiro que vai de 0 até 15.

Além da porta de seleção é definido também a porta de saída e entrada, das duas, **a saída é a mais simples**, pois ela só precisa ter o tamanho **igual** ao tamanho de cada entrada, logo, se cada entrada é composta por **3 bits**, a saída obrigatoriamente também precisa ser composta por **3 bits**, de forma genérica ela é um vetor de bits de tamanho **M**, onde **M é o número de bits de cada entrada**.

Para as entradas, a lógica de certa forma é bem simples; visando uma melhor manipulação futura, a melhor opção foi pensar na entrada como sendo um vetor de bits que tem seu tamanho total como sendo a **soma do tamanho de todas as entradas**, ou, analisando de outra forma, **a multiplicação da quantidade de entradas com a quantidade de bits de cada uma**. De forma generalizada, tem-se que o tamanho do vetor de entrada é denotado pela seguinte equação:

$$2^N * M$$

Onde **N é o número de bits da porta de seleção**, e **M é o número de bits de cada entrada**. Por exemplo, se queremos que um multiplexador tenha **8 entradas**, com **4 bits cada uma**, teremos:

$$M = 4$$

$$N = \log_2 8 = 3$$

Então: len(entrada) $= 2^3 * 4 = 32$ Logo, o vetor de bits correspondente às entradas, é um vetor que vai de 31 até 0.

Considerando uma melhor visualização para as entradas e saídas de teste, foram também declaradas na entidade os vetores referentes aos *displays* de 7 segmentos. Segue o trecho de código correspondente à entidade:

```
-- Entidade
ENTITY atividade_4 IS
  -- Constantes
  GENERIC (N: INTEGER := 4; --Quantidade de Bits de Selecao
  M: INTEGER := 8); --Tamanho de cada entrada

  -- Portas Entradas, Saida e Selecao
  PORT (sel: in INTEGER range 0 to 2**N-1;
      saida: out STD_LOGIC_VECTOR(M-1 DOWNTO 0);
      --saida: buffer STD_LOGIC_VECTOR(M-1 DOWNTO 0); -- Mudada para
   ↪ teste em placa
      entradas: in STD_LOGIC_VECTOR(((2**N) * M) - 1 DOWNTO 0));
      -- ssd_out_0, ssd_out_1, ssd_out_2,ssd_out_3, ssd_out_4,
   ↪ ssd_out_5: out STD_LOGIC_VECTOR(6 DOWNTO 0));
END ENTITY;
```

Código 3: Declaração da Entidade

3.2 Arquitetura

Definida como *main*, a arquitetura emprega o papel de implementar de fato a seleção do dado de entrada que irá para a saída; por meio do comando ***GENERATE***, definido como *gen*, temos uma estrutura for em seu interno, onde o código dentro da estrutura de repetição é executada **M** vezes, onde **M** é o tamanho do vetor de saída, logo, **i** assume valores de 0 até $M-1$. Seu código interno é composto por somente um comando, que é um comando de atribuição onde se percorre todas as posições do vetor de saída, atribuindo em cada posição o valor correspondente ao dado selecionado. Ou seja, em seu laço de repetição, temos o índice **i** que assumirá valores de **0 até o tamanho da saída**, sendo assim percorrerá cada posição de saída; com relação ao vetor de entradas, temos que a operação matemática em seu interno serve como ponteiro, que aponta o início da entrada selecionada. Para um melhor compreensão segue o processo lógico de *debug* quando temos **4** entradas com **2** bits cada:

```
entradas: [ 0 1 1 1 0 0 1 0 ]
sel = 2
sa da esperada: 0 0
M = 2

Primeira itera  o (i=0):
y(0) = entradas((2*2) + 0); - - entradas(4) = 0

Segunda itera  o (i=1):
y(1) = entradas((2*2) + 1); - - entradas(5) = 0

y =  0  0  ;
```

Vale ressaltar que um vetor tem seu início em 0. Analisando o processo de *debug* acima, conclui-se que a saída é a esperada, e que podemos generalizar esse algoritmo para 2^N entradas de **M** bits cada. Gerando assim um **multiplexador genérico**. Abaixo é apresentado o código

referente à arquitetura, nele está acrescentado a parte "prática" de teste, onde foi implementada utilizando os *displays* de sete segmentos.

```
-- Arquitetura
ARCHITECTURE main OF atividade_4 IS
BEGIN
  gen: for i in saida'range GENERATE
    saida(i) <= entradas((sel*M) + i);
  END GENERATE;

-- Implementacao na placa
/* Para manter o comentario de bloco, e necessario VHDL 2008
  --Quando a quantidade de entradas for 4 (N = 2 e M = 2)
  --Exibir as sa das nos SSDs
  --Entrada 1
  WITH entradas(1 DOWNTO 0) SELECT
    ssd_out_0 <=   "1000000" WHEN "00",
                   "1111001" WHEN "01",
                   "0100100" WHEN "10",
                   "0110000" WHEN "11",
                   "0000110" WHEN others;
   --Entrada 2
  WITH entradas(3 DOWNTO 2) SELECT
    ssd_out_1 <=   "1000000" WHEN "00",
                   "1111001" WHEN "01",
                   "0100100" WHEN "10",
                   "0110000" WHEN "11",
                   "0000110" WHEN others;
   --Entrada 3
  WITH entradas(5 DOWNTO 4) SELECT
    ssd_out_2 <=   "1000000" WHEN "00",
                   "1111001" WHEN "01",
                   "0100100" WHEN "10",
                   "0110000" WHEN "11",
                   "0000110" WHEN others;

  --Entrada 4
  WITH entradas(7 DOWNTO 6) SELECT
    ssd_out_3 <=   "1000000" WHEN "00",
                   "1111001" WHEN "01",
                   "0100100" WHEN "10",
                   "0110000" WHEN "11",
                   "0000110" WHEN others;

  ssd_out_4 <= "1111110";

  --WITH saida SELECT
    --ssd_out_5 <= "1000000" WHEN "00",
    --             "1111001" WHEN "01",
    --             "0100100" WHEN "10",
    --             "0110000" WHEN "11",
    --             "0000110" WHEN others;
```

```
50
51 */
52 END ARCHITECTURE;
```

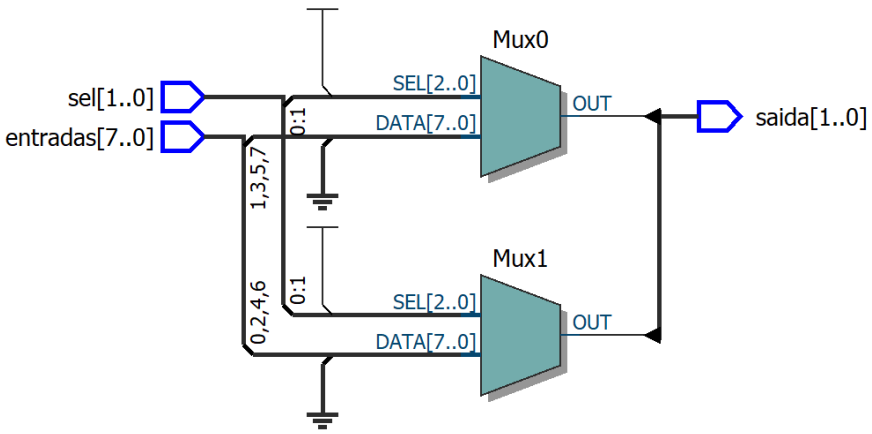
Abaixo, segue o algoritmo completo em **VHDL**:

```
1  -- Declara  o de Biblioteca/Pacote
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4
5  -- Entidade
6  ENTITY atividade_4 IS
7    -- Constantes
8    GENERIC (N: INTEGER := 4; --Quantidade de Bits de Selecao
9    M: INTEGER := 8); --Tamanho de cada entrada
10
11   -- Portas Entradas, Saida e Selecao
12   PORT (sel: in INTEGER range 0 to 2**N-1;
13       saida: out STD_LOGIC_VECTOR(M-1 DOWNTO 0);
14       --saida: buffer STD_LOGIC_VECTOR(M-1 DOWNTO 0); -- Mudada para
     ↪ teste em placa
15       entradas: in STD_LOGIC_VECTOR(((2**N) * M) - 1 DOWNTO 0));
16       -- ssd_out_0, ssd_out_1, ssd_out_2,ssd_out_3, ssd_out_4,
     ↪ ssd_out_5: out STD_LOGIC_VECTOR(6 DOWNTO 0));
17 END ENTITY;
18
19 -- Arquitetura
20 ARCHITECTURE main OF atividade_4 IS
21 BEGIN
22   gen: for i in saida'range GENERATE
23     saida(i) <= entradas((sel*M) + i);
24   END GENERATE;
25
26 -- Implementacao na placa
27 /* Para manter o comentario de bloco, e necessario VHDL 2008
28   --Quando a quantidade de entradas for 4 (N = 2 e M = 2)
29   --Exibir as sa das nos SSDs
30   --Entrada 1
31   WITH entradas(1 DOWNTO 0) SELECT
32     ssd_out_0 <=   "1000000" WHEN "00",
33                "1111001" WHEN "01",
34                "0100100" WHEN "10",
35                "0110000" WHEN "11",
36                "0000110" WHEN others;
37    --Entrada 2
38   WITH entradas(3 DOWNTO 2) SELECT
39     ssd_out_1 <=   "1000000" WHEN "00",
40                "1111001" WHEN "01",
41                "0100100" WHEN "10",
42                "0110000" WHEN "11",
43                "0000110" WHEN others;
44    --Entrada 3
45   WITH entradas(5 DOWNTO 4) SELECT
```

```
46      ssd_out_2 <=   "1000000" WHEN "00",
47                     "1111001" WHEN "01",
48                     "0100100" WHEN "10",
49                     "0110000" WHEN "11",
50                     "0000110" WHEN others;
51
52    --Entrada 4
53    WITH entradas(7 DOWNTO 6) SELECT
54      ssd_out_3 <=   "1000000" WHEN "00",
55                     "1111001" WHEN "01",
56                     "0100100" WHEN "10",
57                     "0110000" WHEN "11",
58                     "0000110" WHEN others;
59
60    ssd_out_4 <= "1111110";
61
62    --WITH saida SELECT
63      --ssd_out_5 <= "1000000" WHEN "00",
64      --             "1111001" WHEN "01",
65      --             "0100100" WHEN "10",
66      --             "0110000" WHEN "11",
67      --             "0000110" WHEN others;
68
69  */
70  END ARCHITECTURE;
```

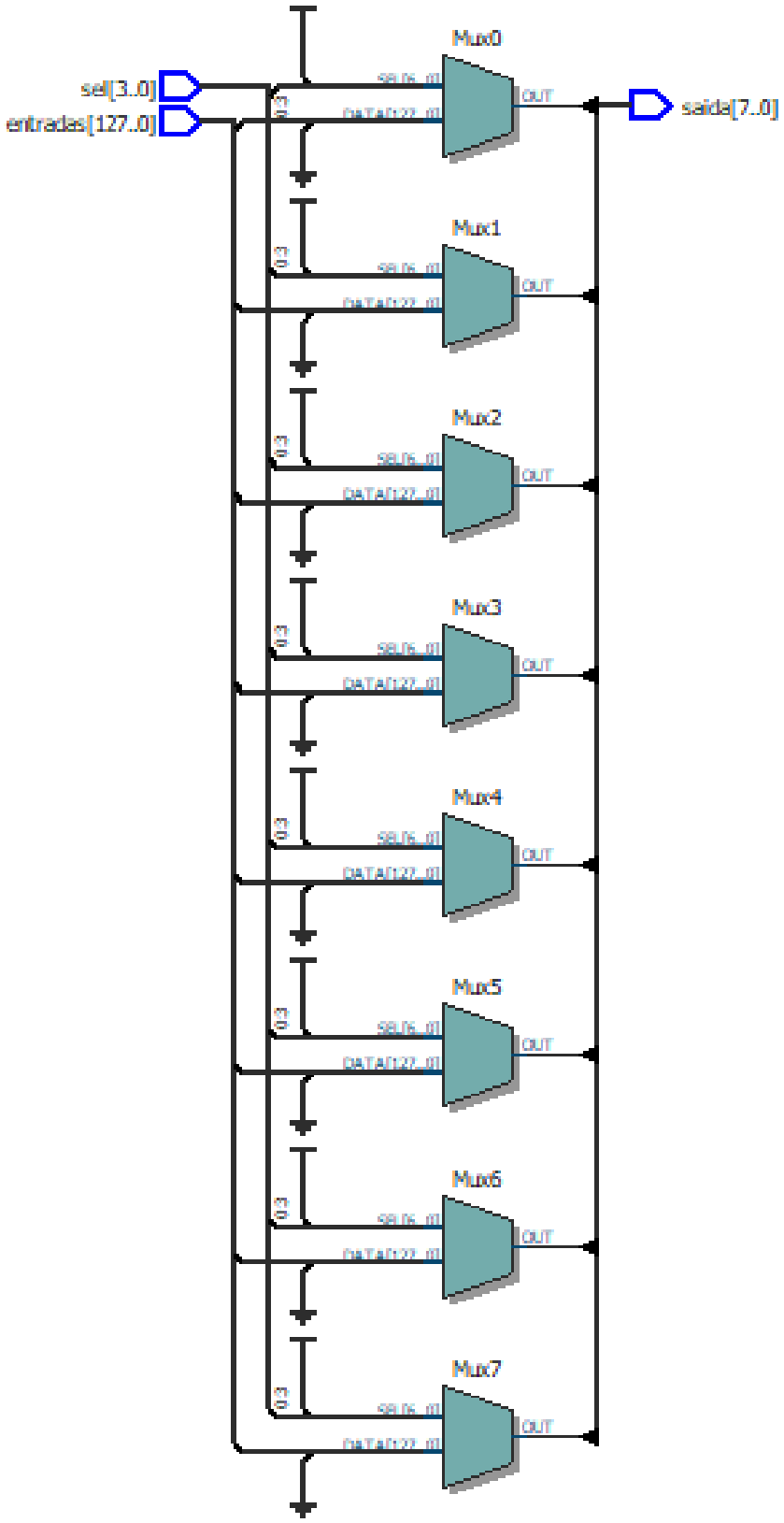
Com a implementação do algoritmo em si, foi gerado o diagrama **RTL** que o representa, abaixo é apresentado o diagrama para **4 entradas de 2 bits cada**, e para **16 entradas de 8 bits cada**:

Figura 5: Diagrama RTL - 4 entradas de 2 bits cada



Fonte: Autoria Própria

Figura 6: Diagrama RTL - 16 entradas de 8 bits cada



Fonte: Autoria Própria

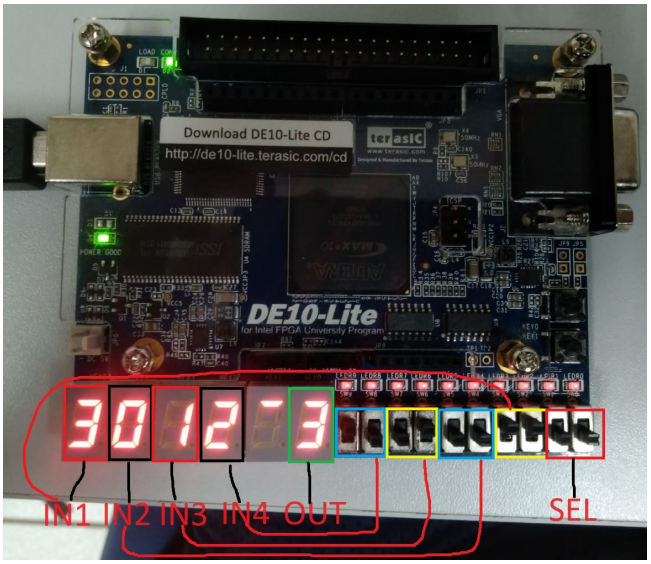
4 Conclusão

Há dois resultados que vão ser levados em consideração, o referente ao funcionamento do multiplexador para **4 entradas de 2 bits** cada, implementado na placa de desenvolvimento utilizando os *displays* de sete segmentos para visualização de seu funcionamento; e o referente ao funcionamento do multiplexador para **16 entradas de 8 bits** cada, no qual seu funcionamento será demonstrado via simulação de forma de onda.

4.1 4 entradas de 2 bits cada (N = 2 e M = 2)

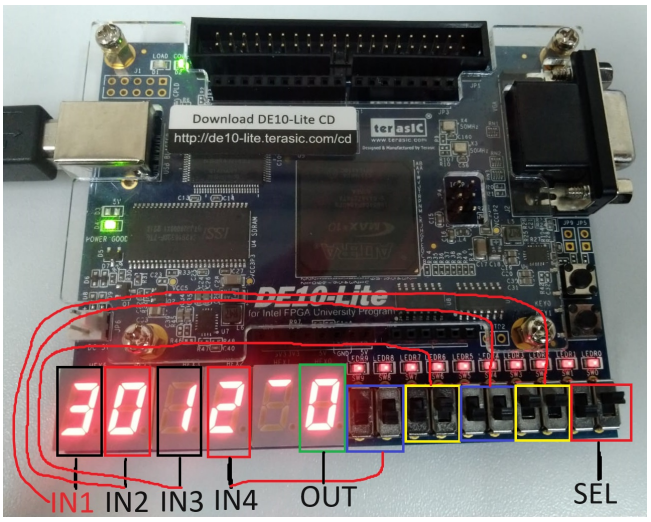
Referente a este resultado temos, que as entradas foram representadas em sua forma decimal, logo, o vetor de bits, entradas, presente no código em **VHDL** é composto por ***"11000110"***, separando cada entrada, e convertendo a entrada em decimal, temos: **3**, como primeira entrada; **0**, como segunda; **1**, como terceira; e **2**, como quarta entrada. Todas as quatro entradas são representadas nos *displays* de 7 segmentos, mais precisamente os 4 primeiros *displays*, da esquerda para a direita. Essas entradas foram definidas utilizando os *switches* presentes na placa, foi utilizado oito *switches* para definição das entradas, onde a cada par de *switches* temos uma entrada. Considerando os *switches* referentes às entradas, restaram apenas dois, que são os que representam os dois bits de seleção, nos testes demonstrados, a única manipulação que houve foi neles, que realizava a alternação entre as entradas que iriam para a saída. Abaixo segue as fotografias de cada um dos testes, vale ressaltar que o último *display* de sete segmentos é o referente a saída:

Figura 7: Seleção - 00



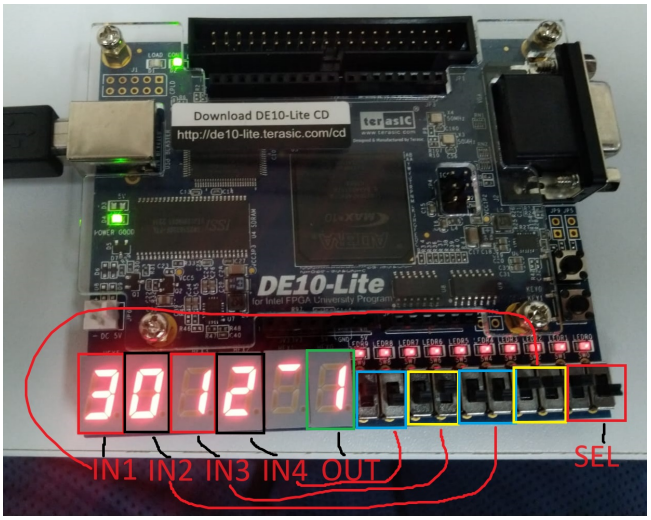
Fonte: Autoria Própria

Figura 8: Seleção - 01



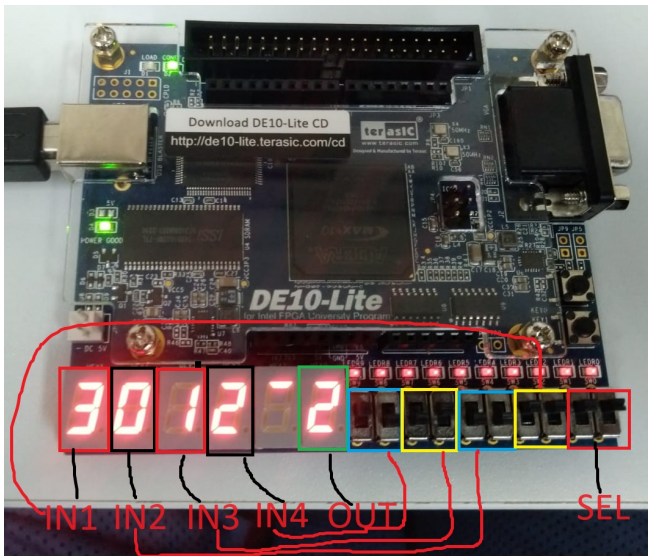
Fonte: Autoria Própria

Figura 9: Seleção - 10



Fonte: Autoria Própria

Figura 10: Seleção - 11

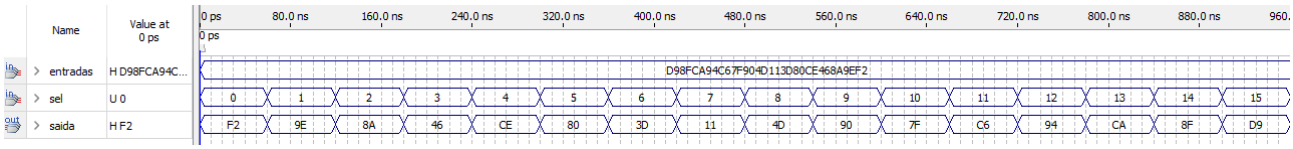


Fonte: Autoria Própria

4.2 16 entradas de 8 bits cada (N = 4 e M = 8)

Para a visualização do funcionamento do multiplexador para **16 entradas de 8 bits cada**, foi utilizado o simulador em forma de onda, para "setar" o multiplexador para essa configuração de 16 entradas de 8 bits, modificou-se os valores de **N e M**, ou seja, **N** recebe $\log_2 16$, e **M** recebe o valor **8**. Lembrando que **N** é a quantidade de bits para a porta de seleção, e **M** é justamente a quantidade de bits de cada entrada. Segue o trecho da entidade, local onde se alterou os valores de **N e M**. Pelo fato de não utilizar um teste via placa de desenvolvimento para esta quantidade de entradas, temos que os trechos de código referentes às saídas para os *displays*, que também envolvem os comandos ***SELECT***, estão sem uso, podendo então serem definidos como comentários. Visando uma melhor visualização do funcionamento do multiplexador na simulação em forma de onda, as entradas e saídas, estão definidas como base hexadecimal, esta definição não irá alterar os resultados. Segue o teste, onde se percorre todas as entradas, considerando uma entrada aleatória fixa.

Figura 11: Simulação Waveform



Fonte: Autoria Própria

Conclui-se então, levando em consideração ambos os testes, que o funcionamento do multiplexador genérico, descrito em **VHDL**, foi o esperado. Provando também a ampla utilidade do comando ***GENERATE***, para códigos concorrentes, utilizado para gerar o multiplexador.

Referências

PEDRONI, Volnei A. **Eletrônica digital moderna e VHDL**. Rio de Janeiro, RJ: Elsevier, 2010. P. 619. ISBN 9788535234657.

TOCCI, Ronald J.; WIDMER, Neal S.; MOSS, Gregory L. **Sistemas digitais: princípios e aplicações**. 11. ed. São Paulo, SP: Pearson Prentice Hall, 2011. P. xxii, 817. ISBN 9788576059226.