Universidade de Brasília
Departamento da Engenharia Elétrica
Laboratório de Sistemas Digitais

Relatório 04

Projeto Modular

José Antônio Alcântara da Silva de Andrade Mat: 232013031

Professor:
Eduardo B. R. F. Paiva
Turma 08

Brasília, DF

10 de dezembro de 2024

1 Objetivos

1. Utilizar multiplexadores e decodificadores para implementar circuitos lógicos combinacionais.
2. Estudar técnicas de projeto modular em VHDL, desenvolvendo sistemas grandes a partir de circuitos menores interligados entre si.

2 Atividades

Essa sessão do laboratório tinha como objetivo a simulação de dois sistemas. Contudo, por requisitado, os códigos que implementarão a arquitetura do sistema não podem conter operações lógicas. Tal deverá ser realizado por meio de uso de outros circuitos e modulação.

2.1 Exercício 1

2.1.1 Modelagem do Problema

O primeiro sistema consiste de 3 bits de entrada (A, B e C) e 2 bits de saída (X e Y), seguindo a Equação Lógica 1.

$$\begin{aligned} X &= \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot \overline{C} + A \cdot B \\ Y &= \overline{A} \cdot \overline{B} + \overline{A} \cdot B \cdot \overline{C} + A \cdot B \cdot C \end{aligned} \tag{1}$$

Mais especificamente, é demandado a construção de um sistema tal que a tabela verdade siga a especificação da Tabela 1.

Tabela 1: Tabela verdade equivalente do exercício 1.

A	B	C	X	Y
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

Contudo, o exercício também estabelece algumas restrições:

1. Use apenas dois multiplexadores 4x1.
2. Use apenas uma porta NOT.
3. Não use operações lógicas dentro do circuito principal.

Devido a essas restrições, será necessário introduzir uma variável nas saídas. Ou seja, transformá-las em funções de uma certa variável. Pode-se fazer isso agrupando-se de duas em duas linhas da Tabela 1 e compará-las com os valores de C (o bit de menor significância). O resultado é a Tabela 2.

Com a Tabela 2, é possível usar um multiplexador 4x1 para implementar a saída X e outro para a saída Y. A porta lógica NOT será utilizada para inverter o sinal de C quando necessário.

Tabela 2: Introdução da variável C na Tabela 1.

A	B	$X(C)$	$Y(C)$
0	0	0	1
0	1	C	$\overline{C}$
1	0	$\overline{C}$	0
0	0	1	C

2.1.2 Módulos

Antes da implementação da Equação Lógica 1, tem-se de implementar os dois módulos requeridos: o multiplexador 4x1 e a porta lógica NOT.

Para o multiplexador, implementou-se o código da Listagem 1.

Listagem 1: Multiplexador 4x1.

```
library ieee;
use ieee.std_logic_1164.all;

entity MUX4X1 is
    port (
        S: in std_logic_vector(1 downto 0);
        D: in std_logic_vector(3 downto 0);
        Y: out std_logic
    );
end MUX4X1;

architecture MUX4X1_ARCH of MUX4X1 is
begin
    Y <= D(0) when S = "00" else
            D(1) when S = "01" else
            D(2) when S = "10" else
            D(3) when S = "11";
end MUX4X1_ARCH;
```

O código da Listagem 1 implementa os requisitos para a funcionalidade padrão de arquivos VHDL, e, então, usa de dois vetores para representar a entrada e um bit para a saída. Um vetor S (set) que indica qual bit do vetor D (data) será representado na saída Y.

Para a porta lógica NOT, usa-se de similar código, como visto na Listagem 2.

Listagem 2: Porta lógica NOT.

```
library ieee;
use ieee.std_logic_1164.all;

entity GATENOT is
    port (
        A: in std_logic;
        Y: out std_logic
    );
end GATENOT;

architecture GATENOT_ARCH of GATENOT is
begin
    Y <= not A;
end GATENOT_ARCH;
```

O código da Listagem 2 implementa a funcionalidade padrão de códigos VHDL, e procede para especificar um circuito que recebe uma entrada A e retorna seu inverso pela saída Y.

Ambos os circuitos são, então, salvos em arquivos separados (a Listagem 1 é salva como `mux4x1.vhd` e a Listagem 2 como `gateNOT.vhd`) e incluídos no projeto do ModelSim, a fim de usá-los para implementar a Equação Lógica 1.

2.1.3 Implementação

Finalmente, pode-se iniciar a implementação do sistema como requerido pelo exercício. Para tal, após definir-se as bibliotecas necessárias, obtém-se o código da Listagem 3.

Listagem 3: Requisitos para o sistema 1.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity SYSTEM1 is
5      port (
6          A: in std_logic;
7          B: in std_logic;
8          C: in std_logic;
9          X: out std_logic;
10         Y: out std_logic
11     );
12 end SYSTEM1;
13
14 architecture SYSTEM1_ARCH of SYSTEM1 is
15     -- ...
16 end SYSTEM1_ARCH;
```

Nas linhas 6, 7, 8, 9 e 10, define-se as entradas A, B e C e as saídas X e Y, respectivamente. Todas são bits simples de entrada e saída.

Logo em seguida, começa-se a definir a arquitetura do sistema. Primeiramente, deve-se declarar os componentes que serão utilizados: outros circuitos que serão importados para serem implementados no sistema 1. Adiciona-se, então, dentro do bloco da arquitetura `SYSTEM1_ARCH` o código da Listagem 4.

Listagem 4: Componentes necessários ao sistema 1.

```
1  component MUX4X1 is
2      port (
3          S: in std_logic_vector(1 downto 0);
4          D: in std_logic_vector(3 downto 0);
5          Y: out std_logic
6      );
7  end component;
8
9  component GATENOT is
10     port (
11         A: in std_logic;
12         Y: out std_logic
13     );
14 end component;
```

Ambos o circuito de multiplexador 4x1 (Listagem 1) e da porta lógica NOT (Listagem 2) são declarados como `component`, seguindo a mesma nomenclatura presente nos seus códigos originais. O multiplexador 4x1 é definido nas linhas 1–7 da Listagem 4 e a porta NOT nas linhas 9–14 da mesma.

Em seguida, ainda dentro do bloco de arquitetura, define-se na Listagem 5 os fios internos que seram utilizados para conexões entre componentes.

Listagem 5: Sinais internos do sistema 1.

```
1  signal setvect      : std_logic_vector(1 downto 0);
2  signal data1, data2 : std_logic_vector(3 downto 0);
3  signal notC         : std_logic;
```

Serão necessários três vetores e um bit para uso interno. O vetor de 2 bits `setvect` listado na linha 1 da Listagem 5 será usado como a entrada S do multiplexador 4x1. Já os vetores de 4 bits `data1` e `data2` listados na linha 2 serão utilizados como a entrada D do multiplexador 4x1 para simular as saídas X e Y, respectivamente. O sinal de bit simples `notC` listado na linha 3 representará o inverso da entrada C.

Por fim, na Listagem 6 (ainda dentro da arquitetura), inicia-se a lógica do sistema 1, correlacionando a Tabela 2 com os vetores necessários, e finalmente introduzindo-os aos multiplexadores para obter-se o resultado desejado.

Listagem 6: Operações do sistema 1.

```
1  begin
2      setvect(0) <= B;
3      setvect(1) <= A;
4
5      INT1 : GATENOT port map(C, notC);
6
7      data1(0) <= '0';
8      data1(1) <= C;
9      data1(2) <= notC;
10     data1(3) <= '1';
11
12     data2(0) <= '1';
13     data2(1) <= notC;
14     data2(2) <= '0';
15     data2(3) <= C;
16
17     INT2 : MUX4X1 port map(setvect, data2, Y);
18     INT3 : MUX4X1 port map(setvect, data1, X);
```

De início, adiciona-se os bits A e B às suas devidas posições no vetor que será utilizado como entrada S dos multiplexadores, nas linhas 2 e 3 da Listagem 6. Após definir o inverso de C na linha 5, usamos os vetores `data1` e `data2` para implementar X (nas linhas 7–10) e Y (nas linhas 12–15), respectivamente. Finalmente, nas linhas 17 e 18, chamamos o código do multiplexador para produzir as saídas desejadas.

Com isso, o sistema 1 está completamente implementado, como visto na Listagem 7.

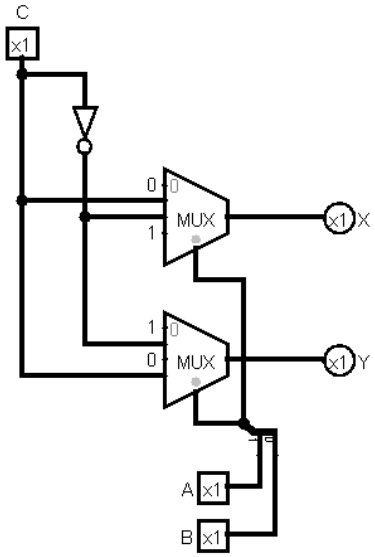
Listagem 7: Implementação completa do sistema 1.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity SYSTEM1 is
5      port (
6          A: in std_logic;
7          B: in std_logic;
8          C: in std_logic;
9          X: out std_logic;
10         Y: out std_logic
11     );
12 end SYSTEM1;
13
14 architecture SYSTEM1_ARCH of SYSTEM1 is
15 component MUX4X1 is
16     port (
17         S: in std_logic_vector(1 downto 0);
18         D: in std_logic_vector(3 downto 0);
19         Y: out std_logic
20     );
21 end component;
22
23 component GATENOT is
24     port (
25         A: in std_logic;
26         Y: out std_logic
27     );
28 end component;
29
30 signal setvect      : std_logic_vector(1 downto 0);
31 signal data1, data2 : std_logic_vector(3 downto 0);
32 signal notC         : std_logic;
33
34 begin
35     setvect(0) <= B;
36     setvect(1) <= A;
37
```

```
38      INT1 : GATENOT port map(C, notC);
39
40      data1(0) <= '0';
41      data1(1) <= C;
42      data1(2) <= notC;
43      data1(3) <= '1';
44
45      data2(0) <= '1';
46      data2(1) <= notC;
47      data2(2) <= '0';
48      data2(3) <= C;
49
50      INT2 : MUX4X1 port map(setvect, data2, Y);
51      INT3 : MUX4X1 port map(setvect, data1, X);
52  end SYSTEM1_ARCH;
```

Na Figura 1 temos uma visualização do circuito.

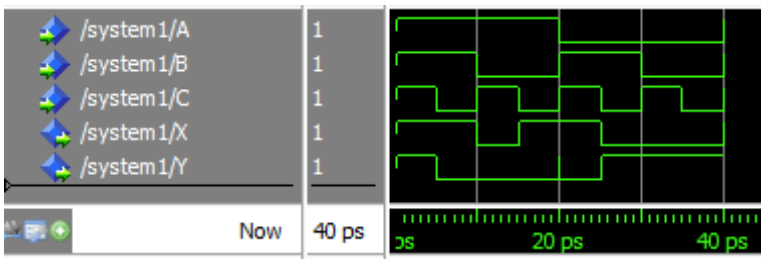
Figura 1: Visualização do Sistema 1.



2.1.4 Simulação

Como o sistema consiste de apenas 3 variáveis de bit simples de entrada, não há dificuldade na construção da simulação. Após propriamente designar os relógios, obtém-se o resultado da Figura 2.

Figura 2: Simulação sistema 1.



Vê-se, então, que a introdução de variável nos multiplexadores resultou na correta implementação da Equação Lógica 1 como requisitado pelo problema.

Uma observação a realçar é a presença de um 'pico' na saída Y na mudança de variáveis. Devido a condições da vida real, um sistema nunca altera-se instantaneamente, ou seja, pode-se ocorrer mudanças fantasmas na saída durante a transição de valores.

2.2 Exercício 2

2.2.1 Modelagem do Problema

Para o segundo exercício, é requisitado a construção de um sistema que atenda a Equação Lógica 2. Por questões de espaçamento e legibilidade, removeu-se os marcadores de operação AND, deixando-os implícitos.

$$Z = FG + ABCD\bar{E}\bar{F}G + \bar{A}\bar{B}\bar{C}\bar{D}\bar{E}\bar{F}G + A\bar{B}CEF\bar{G} + \\ + \bar{A}BCD\bar{E}F\bar{G} + ABCDE\bar{F}\bar{G} + A\bar{B}\bar{C}DE\bar{F}\bar{G} \quad (2)$$

Têm-se, então, a Tabela 3 como a tabela verdade da saída Z.

Tabela 3: Tabela verdade equivalente do exercício 2.

A	B	C	D	E	F	G	Z
x	x	x	x	x	1	1	1
1	1	1	1	1	0	0	1
1	1	1	1	0	0	1	1
1	0	1	x	1	1	0	1
1	0	0	1	1	0	0	1
0	1	1	1	0	1	0	1
0	0	0	0	0	0	1	1
Outros Valores							0

Onde "x" representa termos *don't care*.
Novamente, o exercício requisita o cumprimento de restrições:

1. Use apenas um decodificador 4x16.
2. Use apenas um multiplexador 8x1.
3. Apenas portas OR devem ser utilizadas, sem limites à sua quantidade.
4. Não use operações lógicas dentro do circuito principal.

Seguindo as restrições, como desejado, prova-se essencial a divisão do sistema em duas partes distintas. A primeira, consistindo dos bits de entrada A, B, C, D serão utilizados no decodificador para a seleção de mintermos desejados, enquanto os bits restantes, E, F, G serão usados no multiplexador para completar o sistema.

Pode-se usar da Tabela 3 para construir uma nova tabela que indica quais mintermos serão utilizados em quais entradas do decodificador. Obterá-se, então, a Tabela 4, que será posteriormente utilizada para a construção do código do sistema.

2.2.2 Módulos

Como anteriormente, será necessário a implementação dos circuitos para um decodificador 4x16 e um multiplexador 8x1, além da porta OR. Pode-se usar do código do multiplexador 4x1 (presente na Listagem 1) para a construção do novo circuito, basta apenas realizar uma adaptação em seu código para atender três variáveis, como visto na Listagem 8.

Tabela 4: Introdução de variável no multiplexador 8x1.

E	F	G	Entradas no mux
0	0	0	0
0	0	1	$ABCD + \bar{A}\bar{B}\bar{C}\bar{D}$
0	1	0	$\bar{A}BCD$
0	1	1	1
1	0	0	$ABCD + A\bar{B}\bar{C}D$
1	0	1	0
1	1	0	$A\bar{B}CD + A\bar{B}C\bar{D}$
1	1	1	1

Listagem 8: Multiplexador 8x1.

```
library ieee;
use ieee.std_logic_1164.all;

entity MUX8X1 is
    port (
        S: in std_logic_vector(2 downto 0);
        D: in std_logic_vector(7 downto 0);
        Y: out std_logic
    );
end MUX8X1;

architecture MUX8X1_ARCH of MUX8X1 is
begin
    Y <= D(0) when S = "000" else
         D(1) when S = "001" else
         D(2) when S = "010" else
         D(3) when S = "011" else
         D(4) when S = "100" else
         D(5) when S = "101" else
         D(6) when S = "110" else
         D(7) when S = "111";
end MUX8X1_ARCH;
```

Há poucas diferenças na implementação do 8x1 em comparação ao 4x1. Mais notavelmente, o vetor de entrada S agora possui três bits, e D possui 8. Contudo, a saída Y permanece um bit simples.

Similarmente, o código para a porta OR é construído de maneira similar ao código da porta NOT (presente na Listagem 2), implementado como segue na Listagem 9.

Listagem 9: Porta lógica OR.

```
library ieee;
use ieee.std_logic_1164.all;

entity GATEOR is
    port (
        A: in std_logic;
        B: in std_logic;
        Y: out std_logic
    );
end GATEOR;

architecture GATEOR_ARCH of GATEOR is
begin
    Y <= A or B;
end GATEOR_ARCH;
```

A porta OR recebe duas entradas simples A e B, com apenas uma saída simples Y. O resultado do circuito será 1 se qualquer uma das duas entradas forem 1.

Por fim, necessita-se implementar o decodificador 4x16. Para tal, usa-se o código presente na Listagem 10

Listagem 10: Decodificador 4x16.

```
library ieee;
use ieee.std_logic_1164.all;

entity DECOD4X16 is
    port (
        A: in std_logic_vector(3 downto 0);
        Y: out std_logic_vector(15 downto 0)
    );
end DECOD4X16;

architecture DECOD4X16_ARCH of DECOD4X16 is
begin
    with A select
    Y <= "0000000000000001" when "0000",
         "0000000000000010" when "0001",
         "0000000000000100" when "0010",
         "0000000000001000" when "0011",
         "0000000000010000" when "0100",
         "0000000000100000" when "0101",
         "0000000001000000" when "0110",
         "0000000010000000" when "0111",
         "0000000100000000" when "1000",
         "0000001000000000" when "1001",
         "0000010000000000" when "1010",
         "0000100000000000" when "1011",
         "0001000000000000" when "1100",
         "0010000000000000" when "1101",
         "0100000000000000" when "1110",
         "1000000000000000" when "1111",
         "0000000000000000" when others;
end DECOD4X16_ARCH;
```

O decodificador recebe apenas uma entrada, o vetor A de 4 bits que determina qual dos 16 bits disponíveis no vetor de saída Y deverá ser ligada. Por exemplo, ao enviar o vetor 0101 como entrada A, a saída Y_5 será acesa, enquanto as outras permanecem como 0.

Salva-se os arquivos no projeto, como `decoder4x16.vhd`, `mux8x1.vhd` e `gateOR.vhd`. Com isso, parte-se para a resolução do exercício, como requisitado.

2.2.3 Implementação

A implementação será dividida em três etapas. Primeiro, usa-se o decodificador nos bits de entrada A, B, C e D. Então, seleciona-se os mintermos relevantes e agrupa-os com portas OR. Finalmente, usando um multiplexador com os valores E, F e G como bits seletores, conectando as entradas data de acordo com a Tabela 4.

Contudo, antes da implementação das operações, deve-se estabelecer os requisitos funcionais do sistema. Se usará o código presente na Listagem **??**.

Listagem 11: Requisitos iniciais para o Sistema 2.

```
library ieee;
use ieee.std_logic_1164.all;

entity SYSTEM2 is
    port (
        A: in std_logic;
        B: in std_logic;
        C: in std_logic;
        D: in std_logic;
        E: in std_logic;
        F: in std_logic;
        G: in std_logic;
        Z: out std_logic
    );
end SYSTEM2;

architecture SYSTEM2_ARCH of SYSTEM2 is
```

```
18      -- ...
19  end SYSTEM2_ARCH;
```

Com a Listagem 11, as entradas e a saída estão declaradas. Segue-se para a montagem da arquitetura do sistema. Começa-se pela Listagem 12 (a qual será inserida na linha 18 da Listagem 11) ao introduzir os circuitos previamente implementados no sistema.

Listagem 12: Circuitos integrados no Sistema 2.

```
1   component MUX8X1 is
2       port (
3           S: in std_logic_vector(2 downto 0);
4           D: in std_logic_vector(7 downto 0);
5           Y: out std_logic
6       );
7   end component;
8
9   component GATEOR is
10      port (
11          A: in std_logic;
12          B: in std_logic;
13          Y: out std_logic
14      );
15  end component;
16
17  component DECOD4X16 is
18      port (
19          A: in std_logic_vector(3 downto 0);
20          Y: out std_logic_vector(15 downto 0)
21      );
22  end component;
```

Relembrando, na Listagem 12, as linhas 1–7 introduzem o circuito do multiplexador 8x1, as linhas 9–15 a porta OR e, por fim, as linhas 17–22 o decodificador 4x16.

Em seguida, são definidos os sinais que serão utilizados no Sistema 2, na Listagem 13. Precisa-se de vetores para as entradas e saídas do multiplexador e decodificador, além de fios para conectar as portas OR em suas devidas posições.

Listagem 13: Sinais internos do Sistema 2.

```
1   signal decvect   : std_logic_vector(3 downto 0);
2   signal setvect   : std_logic_vector(2 downto 0);
3   signal decresult : std_logic_vector(15 downto 0);
4   signal datain    : std_logic_vector(7 downto 0);
5
6   signal wire1, wire2, wire3 : std_logic;
```

O vetor `decvect` definido na linha 1 da Listagem 13 será utilizado como a entrada do decodificador, o vetor `setvect` da linha 2 como entrada set do multiplexador, o `decresult` da linha 3 como saída do decodificador e o vetor `datain` da linha 4 como entrada data do multiplexador.

Os fios `wire1`, `wire2` e `wire3` serão utilizados para implemetar as portas OR.

Listagem 14: Arquitetura funcional do Sistema 2.

```
1   begin
2       decvect(3) <= A;
3       decvect(2) <= B;
4       decvect(1) <= C;
5       decvect(0) <= D;
6
7       setvect(2) <= E;
8       setvect(1) <= F;
9       setvect(0) <= G;
10
11      INT1 : DECOD4X16 port map(decvect, decresult);
12
13      INT2 : GATEOR port map(decresult(0), decresult(15), wire1);
14      INT3 : GATEOR port map(decresult(15), decresult(9), wire2);
15      INT4 : GATEOR port map(decresult(10), decresult(11), wire3);
```

```
16
17      datain(0) <= '0';
18      datain(1) <= wire1;
19      datain(2) <= decresult(7);
20      datain(3) <= '1';
21      datain(4) <= wire2;
22      datain(5) <= '0';
23      datain(6) <= wire3;
24      datain(7) <= '1';
25
26      INT5 : MUX8X1 port map(setvect, datain, Z);
```

Por fim, realiza-se as operações como previamente descrito na Listagem 14.

Nas linhas 2–5, associa-se o valor de entrada A no valor de maior significância do vetor `decvect`, B no segundo, e assim por diante para os valores C e D. Realiza-se o mesmo, nas linhas 7–9, para o vetor `setvect` e as entradas E, F e G, respectivamente.

Na linha 11, declara-se o uso do circuito do decodificador, associando-se o vetor `decvect` como sua entrada, e `decresult` como sua saída.

Em seguida, realiza-se operações OR a fim de modelar os mintermos necessários para as conexões no multiplexador. Na linha 13 da Listagem 14, realiza-se a soma lógica do mintermo $\bar{A}\bar{B}\bar{C}\bar{D}$ (valor 0 do decodificador) com o mintermo $ABCD$ (valor 15) e envia-se o resultado para o fio `wire1`. Na linha 14, asoocia-se os mintermos $ABCD$ (15) e $\bar{A}BC\bar{D}$ (9) com um OR para o fio `wire2`. Por fim, associa-se $A\bar{B}CD$ (11) com $A\bar{B}C\bar{D}$ (10) no fio `wire3`.

Usando o vetor `datain` cria-se a entrada data que será utilizada no multiplexador, logo iniciando-se suas associações de acordo com a Tabela 4. Nas entradas 0 e 5 (linhas 17 e 22, respectivamente), insere-se uma constante baixa. Já nas 3 e 7 (linhas 20 e 24) insere-se uma constate alta. Para a entrada 1, na linha 18, usa-se o `wire1`, na entrada 4, na linha 21, o `wire2` e, na entrada 6, na linha 23, o `wire3`. Por último, diretamente conecta-se o mintermo $\bar{A}BCD$ do decodificador na entrada 2, na linha 19.

Para concluir o sistema 2, cria-se o multiplexador na linha 26: associa-se como entrada data o vetor `datain`, como entrada set o vetor `setvect` e como saída o bit Z. Assim, o sistema está completamente implementado, como visto na Listagem 15.

Listagem 15: Código final do Sistema 2.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity SYSTEM2 is
5      port (
6          A: in std_logic;
7          B: in std_logic;
8          C: in std_logic;
9          D: in std_logic;
10         E: in std_logic;
11         F: in std_logic;
12         G: in std_logic;
13         Z: out std_logic
14     );
15 end SYSTEM2;
16
17 architecture SYSTEM2_ARCH of SYSTEM2 is
18
19 component MUX8X1 is
20     port (
21         S: in std_logic_vector(2 downto 0);
22         D: in std_logic_vector(7 downto 0);
23         Y: out std_logic
24     );
25 end component;
26
27 component GATEOR is
28     port (
```

```
        A: in std_logic;
        B: in std_logic;
        Y: out std_logic
    );
end component;

component DECOD4X16 is
    port (
        A: in std_logic_vector(3 downto 0);
        Y: out std_logic_vector(15 downto 0)
    );
end component;

signal decvect   : std_logic_vector(3 downto 0);
signal setvect   : std_logic_vector(2 downto 0);
signal decresult : std_logic_vector(15 downto 0);
signal datain    : std_logic_vector(7 downto 0);

signal wire1, wire2, wire3 : std_logic;

begin
    decvect(3) <= A;
    decvect(2) <= B;
    decvect(1) <= C;
    decvect(0) <= D;

    setvect(2) <= E;
    setvect(1) <= F;
    setvect(0) <= G;

    INT1 : DECOD4X16 port map(decvect, decresult);

    INT2 : GATEOR port map(decresult(0), decresult(15), wire1);
    INT3 : GATEOR port map(decresult(15), decresult(9), wire2);
    INT4 : GATEOR port map(decresult(10), decresult(11), wire3);

    datain(0) <= '0';
    datain(1) <= wire1;
    datain(2) <= decresult(7);
    datain(3) <= '1';
    datain(4) <= wire2;
    datain(5) <= '0';
    datain(6) <= wire3;
    datain(7) <= '1';

    INT5 : MUX8X1 port map(setvect, datain, Z);

end SYSTEM2_ARCH;
```

Na Figura 3 temos uma visualização do circuito.

2.2.4 Simulação

Após a compilação dos códigos, realiza-se a simulação, colocando potências crescentes de 2 como valores para os períodos das entradas. O resultado é visualisado nas Figuras 4 e 5.

2.2.5 Compilação

Como segue na Figura 6, nenhum dos códigos obteve erros de compilação.

Figura 3: Visualização do Sistema 2.

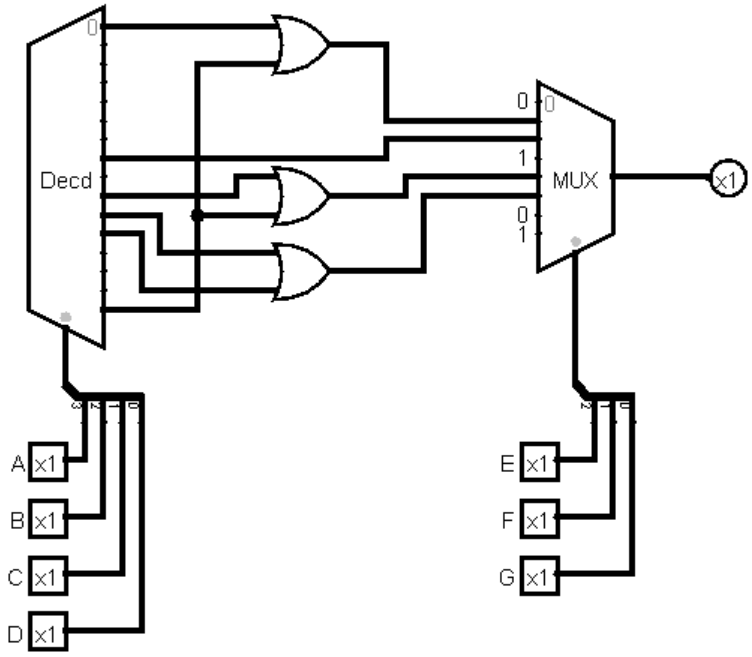


Figura 4: Simulação do sistema quando $A = 1$.

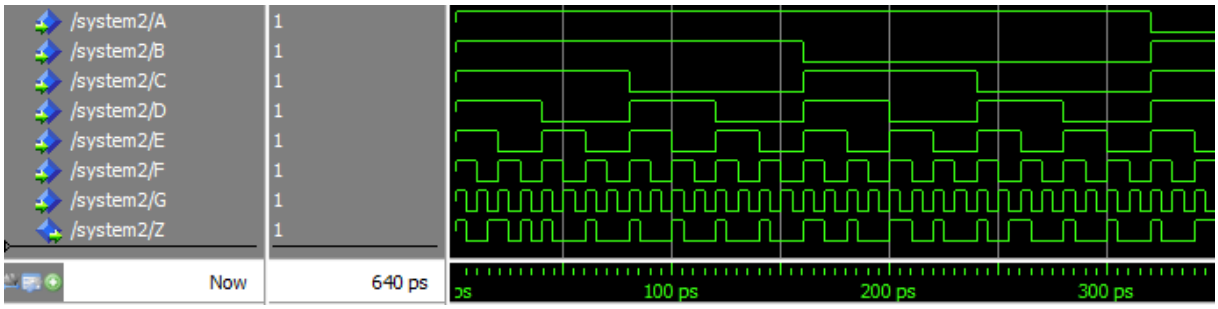


Figura 5: Simulação do sistema quando $A = 0$.

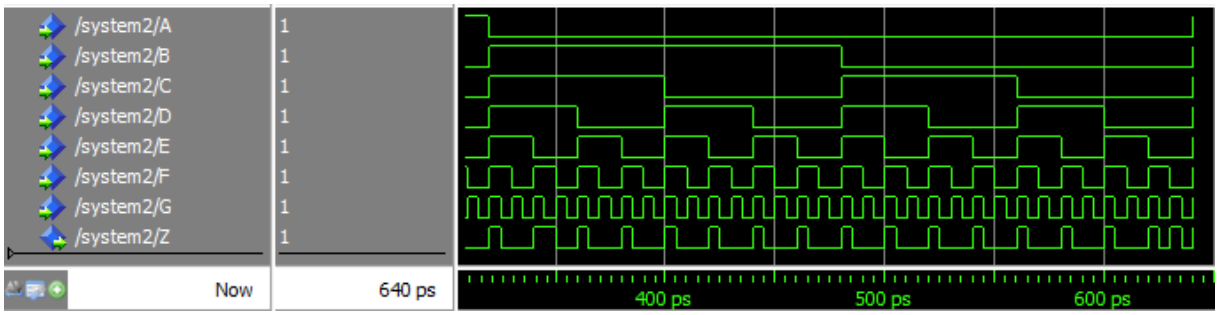


Figura 6: Compilação dos códigos.

```
VSIM 24> run
# Compile of decoder4x16.vhd was successful.
# Compile of e1.vhd was successful.
# Compile of e2.vhd was successful.
# Compile of gateNOT.vhd was successful.
# Compile of gateOR.vhd was successful.
# Compile of mux4x1.vhd was successful.
# Compile of mux8x1.vhd was successful.
# 7 compiles, 0 failed with no errors.
```