TÉCNICO LISBOA

VIVADO TUTORIAL 101: CADEADO DIGITAL

VERSÃO 1.0

- SISTEMAS DIGITAIS -

MEEC

Este tutorial inclui notas adicionais na margem esquerda da página (do tipo **G(X.X)**). Estas notas referram-se ao(s) passo(s) **X.X** do **Guia de Utilização do Vivado Design Suite WebPack** (disponível na página da cadeira), que deve ser consultado para facilitar o desenvolvimento deste trabalho e esclarecer algumas duvidas.

Aleksandar Ilic - Pedro Tomás - Nuno Roma

Pretende-se com este tutorial que os alunos se familiarizem com a utilização de ferramentas de simulação e prototipagem de circuitos digitais. Para o efeito, será realizada uma apresentação tutorial da ferramenta *Vivado WebPack* da *Xilinx* (versão 2016.4), que será utilizada para descrever (implementar) e simular um circuito combinatório (usando a linguagem VHDL).

A preparação prévia deste tutorial está dependente da instalação (bem-sucedida) da ferramenta Vivado WebPack (versão 2016.4) da Xilinx. A instalação deste software pode ser realizada com a ajuda do Guia de Instalação do Vivado Design Suite WebPack, disponível na página da cadeira. Durante a realização deste trabalho é imprescindível consultar o **Guia de Utilização do Vivado Design Suite WebPack**, disponível na página da cadeira. Para além disso, poderá ser útil consultar os slides das aulas teóricas, nomeadamente da aula "*Linguagens de Descrição e Simulação de Circuitos Digitais*".

1. CIRCUITO COMBINATÓRIO: CADEADO DIGITAL

Considere que se pretende desenvolver um cadeado digital, o qual abre se for pressionado o botão de abertura e o código inserido estiver correto. Para simplificação do trabalho, assume-se que o cadeado abre com o código 0111_2, i.e., 7_{10}. O resultado da operação deverá ser indicado por duas saídas, **OK** e **Err**, as quais estarão ativas (valor lógico 1) quando o botão de abertura for pressionado (sinal **xopen**) e o código inserido (sinal **code**) estiver certo/errado.

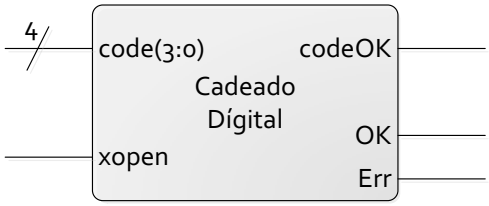

Figura 1. Esquema do cadeado digital a projetar (`entity`).

Para além dos dois sinais anteriores, deverá ser projetado outro sinal, **codeOK**, que deverá servir para teste do circuito projetado. Assim, o sinal **codeOK** deverá estar ativo sempre que o código estiver correto (independentemente do valor do sinal open).

2. IMPLEMENTAÇÃO (DESIGN SOURCES)

G(1.1 até 1.4) 2.1. Inicie o Vivado. Crie de um projeto novo com o nome ***`tutorial_cadeado`***, especifique o dispositivo e faça a configuração do Vivado.

G(2.1b, 2.2) 2.2. Crie um ficheiro novo de descrição do circuito (*Design Source*), indicando o *cadeado_digital* como o nome do ficheiro. Na janela *"Define Module"*, deixe o *cadeado_digital* como o nome do componente ("*Entity Name*") e *Behavioral* como o nome da arquitetura ("*Architecture Name*"). Na parte do "*I/O Port Definitions*" é preciso preencher conforme o esquema do componente apresentado na Figura 1. Após o preenchimento, a janela "*Define Model*" deverá ter um aspeto semelhante ao representado na Figura 2. (Pressione *OK*.)

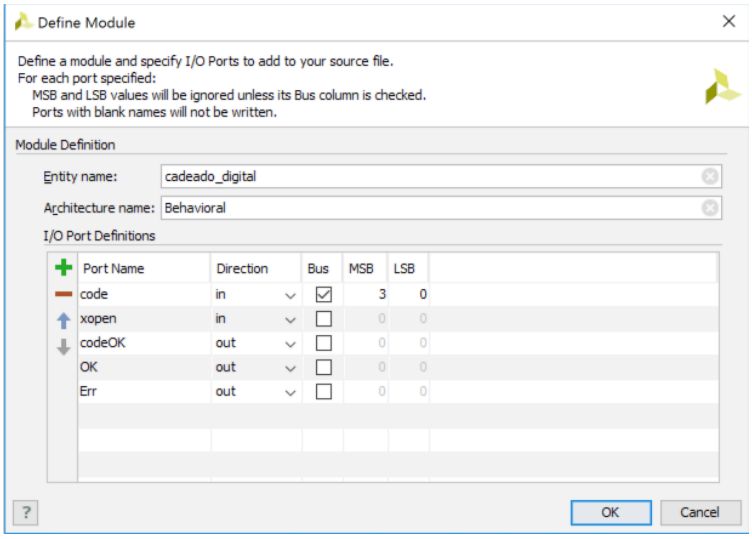

Figura 2. Esquema do cadeado digital a projectar.

G(2.4, 2.5)

2.3. Defina o `cadeado_digital` como o módulo de topo. Abra a respetiva descrição em VHDL e inspecione o conteúdo.

2.4. É possível observar que a parte ***`entity`*** do componente está completamente especificada (i.e., a descrição do componente em VHDL), conforme ilustrado na Figura 3. Compare a ***`entity`*** do componente com o esquema do circuito ilustrado na Figura 1.

```
34 entity cadeado_digital is
35     Port ( code : in STD_LOGIC_VECTOR (3 downto 0);
36            xopen : in STD_LOGIC;
37            codeOK : out STD_LOGIC;
38            OK : out STD_LOGIC;
39            Err : out STD_LOGIC);
40 end cadeado_digital;
41
42 architecture Behavioral of cadeado_digital is
43
44 begin
45
46 end Behavioral;
```

Figura 3. Descrição do cadeado digital (`entity`).

A parte ***`architecture`*** da descrição VHDL encontra-se ainda por preencher. Por conseguinte, será necessário preencher esta parte do ficheiro de acordo com o circuito desejado, representado na Figura 4.

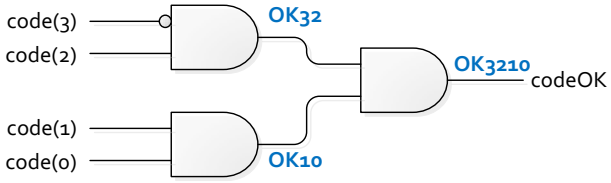

Figura 4. Implementação do cadeado digital (`architecture`).

Em primeiro lugar, é necessário declarar todos os sinais internos do componente (ou seja, fios intermédios), nomeadamente, *OK32*, *OK10* e *OK3210*, tal como mostrado na Figura 4.

Para o efeito, é necessário declarar estes sinais entre as linhas "`architecture Behavioral of cadeado_digital is`" e "`begin`", conforme ilustrado na Figura 5.

```
42 architecture Behavioral of cadeado_digital is
43  -- declaração dos sinais (fios) internos ao componente
44 signal OK32, OK10, OK3210 : std_logic;
45
46 begin
47
48 end Behavioral;
```

Figura 5. Declaração dos sinais internos (`architecture`).

A descrição da implementação deste componente `cadeado_digital` deverá ser feita entre as linhas "`begin`" e "`end Behavioral;`" (no bloco `architecture`). De acordo com a Figura 4, é necessário inserir o seguinte troço do código em VHDL:

```
42 architecture Behavioral of cadeado_digital is
43  -- declaração dos sinais (fios) internos ao componente
44 signal OK32, OK10, OK3210 : std_logic;
45 begin
46  -- Cálculo do resultado
47 OK32   <= (not code(3)) and code(2);
48 OK10   <= code(1) and code(0);
49 OK3210 <= OK32 and OK10;
50  -- Atribuição do valor de saída
51 codeOK <= OK3210;
52 end Behavioral;
```

Figura 6. Implementação do cadeado digital (`architecture`).

Guarde o ficheiro. Após a gravação (e durante a edição do ficheiro), a ferramenta Vivado deteta automaticamente os erros de síntaxe. Verifique se não são mostradas mensagens de erro no tab *Messages* no painel P4 (mensagens do tipo *info* e *status* podem também aparecer e podem, em geral, ser ignoradas). Corrija todos os erros.

G(4.1 até 4.4)

2.5. Para fazer a **verificação completa**, realize a **síntese do circuito**. Após a síntese, observe o tab *Messages* no painel P4, que terá um aspeto semelhante ao representado na Figura 7.

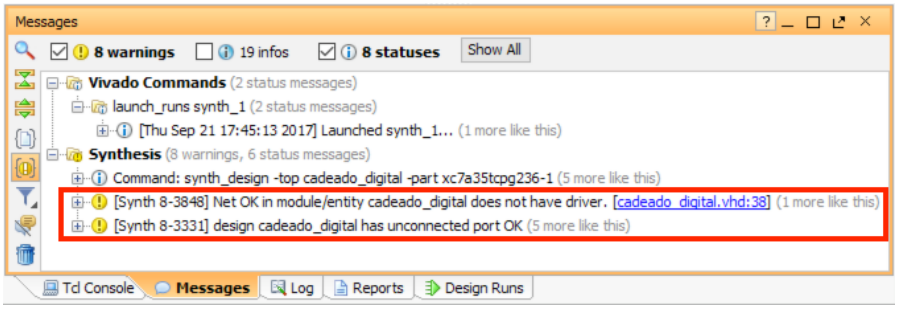

Figura 7. Implementação do cadeado digital (`architecture`).

Neste caso, a ferramenta mostra 8 *warnings*. Os *warnings* destacados na Figura 7 sugerem a existência de alguns *ports* definidos na `entity` do componente (nesse caso, o port *OK*) cujo comportamento não é implementado na descrição que foi preenchida no bloco `architecture`. Ou seja, estes *warnings* sugerem que a implementação do circuito ainda não está completa.

2.6. Para completar a implementação do `cadeado_digital` é ainda necessário descrever o comportamento das saídas *OK* e *Err*, de acordo com a Figura 8.

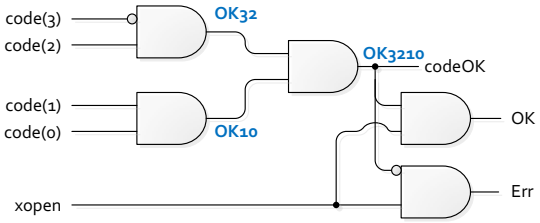

Figura 8. Implementação completa do cadeado digital (`architecture`).

Para isso, é ainda necessário acrescentar duas linhas de código no bloco ***`architecture`***, antes da linha "***`end`*** *`Behavioral;`*", tal como mostrado na Figura 9.

```
50  -- Atribuicao do valor de saida
51  codeOK <= OK3210;
52  OK <= OK3210 and xopen;
53  Err <= (not OK3210) and xopen;
54  end Behavioral;
```

Figura 9. Atribuição do valor de saída (`architecture`).

G(4.1) 2.7. Faça a síntese do circuito. Deverão desaparecer todos os *warnings* previamente observados, sinalizando que a implementação está completa.

3. SIMULAÇÃO (SIMULATION SOURCES)

Para validar o funcionamento do circuito implementado, é necessário verificar o valor das suas saídas para todas as combinações das suas entradas. Assim, o objetivo desta parte do tutorial é criar um *TestBench* para testar o funcionamento do `cadeado_digital` (implementado na Secção 2), instanciar uma unidade de teste, e projetar o gerador de sinais capaz de gerar todas as combinações de entradas da tabela de verdade (ver a Tabela 1) de forma a ser possível verificar o valor das saídas.

Tabela 1. Tabela de verdade do cadeado digital

ENTRADAS		SAÍDAS ESPERADAS		
code	xopen	codeOK	OK	Err
0000	0	0	0	0
0001	0	0	0	0
...	...	...	...	...
0110	0	0	0	0
0111	0	1	0	0
1000	0	0	0	0
...	...	...	...	...
0110	1	0	0	1
0111	1	1	1	0
1000	1	0	0	1
...	...	...	...	...

G(3.1b, 3.2) 3.1. Crie um ficheiro novo para efetuar a simulação do circuito (*Simulation Source*), indicando o seguinte nome para o ficheiro: *cadeado_digital_testbench*. Na janela *"Define Module"*, deixe o nome *cadeado_digital_testbench* como o nome do componente ("*Entity Name*") e *Behavioral* como o nome da arquitetura ("*Architecture Name*"). A parte do "*I/O Port Definitions*" não é necessário preencher. (Pressione *OK* e na janela seguinte *Yes*.)

G(2.4, 3.3)

3.2. Defina o `cadeado_digital_testbench` como o módulo de topo. Abra a descrição em VHDL e inspecione o conteúdo.
É possível observar que a parte ***entity*** está completamente vazia, conforme ilustrado na Figura 10. Neste caso, o *TestBench* não tem qualquer entrada ou saída, uma vez que contemplará apenas os sinais internos para testar o funcionamento do `cadeado_digital`. Na parte ***architecture*** é necessário declarar e instanciar o componente (`cadeado_digital`) e todos os sinais internos para o *TestBench*.

```
34 -- Definicao do nome da entidade, sem qualquer entrada ou saida
35 entity cadeado_digital_testbench is
36 --  Port ( );
37 end cadeado_digital_testbench;
38
39 architecture Behavioral of cadeado_digital_testbench is
40
41 begin
42
43
44 end Behavioral;
```

Figura 10. Definição do TestBench (`entity`).

A declaração do componente é sempre feita entre as linhas "***architecture*** *Behavioral* ***of*** *cadeado_digital_testbench* ***is***" e "***begin***" e representa uma cópia quase perfeita da descrição da entidade do componente que se pretende simular (neste caso, `cadeado_digital`). As únicas diferenças residem na primeira e última linhas, conforme se pode observar na Figura 11.

```
39 architecture Behavioral of cadeado_digital_testbench is
40 -- Declaracao do componente cadeado_digital original
41 component cadeado_digital
42     Port ( code : in STD_LOGIC_VECTOR (3 downto 0);
43            xopen : in STD_LOGIC;
44            codeOK : out STD_LOGIC;
45            OK : out STD_LOGIC;
46            Err : out STD_LOGIC);
47 end component;
48
49 begin
50
51 end Behavioral;
```

Figura 11. Declaração do componente para testar.

Depois da declaração do componente (e ainda antes do "***begin***"), é necessário declarar todos os sinais (internos) necessários para o *TestBench*. Os sinais do *TestBench* podem ser declarados usando o mesmo nome dos sinais no componente (não é obrigatório), ou seja, como uma cópia das declarações de todas as entradas e saídas do circuito original, tal como ilustrado na Figura 12 (no caso do `cadeado_digital`).

```
39 architecture Behavioral of cadeado_digital_testbench is
40 -- Declaracao do componente cadeado_digital original
41 component cadeado_digital
42     Port ( code : in STD_LOGIC_VECTOR (3 downto 0);
43            xopen : in STD_LOGIC;
44            codeOK : out STD_LOGIC;
45            OK : out STD_LOGIC;
46            Err : out STD_LOGIC);
47 end component;
48
49 -- Declaracao dos sinais para o testbench
50 signal code : STD_LOGIC_VECTOR (3 downto 0);
51 signal xopen, codeOK, OK, Err : STD_LOGIC;
52
53 begin
54
55 end Behavioral;
```

Figura 12. Declaração dos sinais para TestBench.

A implementação do *TestBench* é feita entre as linhas "`begin`" e "`end Behavioral;`" (no bloco `architecture`). Em primeiro lugar é necessário incluir uma instância da unidade sob teste, fazendo o *port map* do componente previamente declarado e ligando os sinais do *TestBench* nas entradas e saídas do componente. A Figura 13 mostra a declaração da unidade de teste (`Utest`) no caso do `cadeado_digital`.

```
53  begin
54  -- Declaracao da unidade de teste ... O nome dos sinais
55  -- no circuito e neste caso (nao obrigatorio) o mesmo
56  -- que o nome dos sinais no componente
57  Utest: cadeado_digital Port map (
58             code => code, xopen => xopen,
59             codeOK => codeOK, OK => OK, Err => Err);
60
61  end Behavioral;
```

Figura 13. Declaração da unidade de teste.

Depois, é preciso descrever os geradores dos sinais (*processes*). Neste caso, optou-se por separar a tabela de verdade (ver Tabela 1) em duas partes, criando assim dois *processes* (um para cada entrada), nomeadamente, para os sinais `code` e `xopen`. Os *processes* apresentados na Figura 14 utilizam macros[1], i.e., um contador para o *process* do sinal `code` e um inversor para o *process* do sinal `xopen`. De notar que estes *processes* executam-se em simultâneo, ou seja, eles mudam os seus sinais ao mesmo tempo (independente um de outro).

```
53  begin
54  Utest: cadeado_digital Port map (
55             code => code, xopen => xopen,
56             codeOK => codeOK, OK => OK, Err => Err);
57  -- Descricao do gerador de sinais
58  -- Geracao do sinal CODE
59  process
60  begin
61      code <= code + 1;
62      wait for 10 ns;
63  end process;
64  -- Geracao do sinal XOPEN
65  process
66  begin
67      xopen <= not xopen;
68      wait for 16*10 ns;
69  end process;
70
71  end Behavioral;
```

Figura 14. Declaração dos geradores dos sinais (processes).

Guarde o ficheiro. Após a gravação e durante a edição do ficheiro, a ferramenta *Vivado* deteta automaticamente os erros de sintaxe. Verifique se é mostrada alguma mensagem de erro no tab *Messages* no painel P4 (as mensagens de *info* e *status* podem aparecer e podem, em geral, ser ignoradas). Corrija todos os erros.

G(3.4, 4.3)

3.3. Inicie a simulação do modulo `cadeado_digital` usando o *TestBench* previamente criado (`cadeado_digital_testbench`). Após o início da simulação, será aberta uma janela

[1] Existem varias formas de descrever o gerador do sinal e algumas foram apresentadas na aula teórica 12. É recomendável consultar os slides desta aula teórica e ainda implementar o mesmo *TestBench* usando as diferentes formas.

"*Critical Messages*" onde a ferramenta reporta um erro, semelhante ao ilustrado na Figura 14. (Pressione *OK*.)

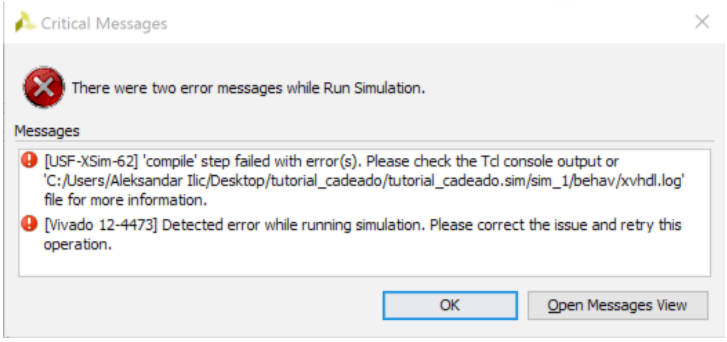

Figura 15. Declaração dos geradores dos sinais (processes).

Conforme se pode observar, nem todos os erros são apresentados quando se grava o ficheiro. Neste caso, a existência dos erros é apenas verificada no tab "*Messages*" no painel P4. Para obter mais informações sobre a causa do erro, observe o conteúdo apresentado no tab "*Tcl Console*" no painel P4 (ver Figura 16) e procure a ocorrência do texto tipo:

ERROR: *<informação sobre o erro>*[C:/.../**nome_do_ficheiro**.vhd:**LINHA**].

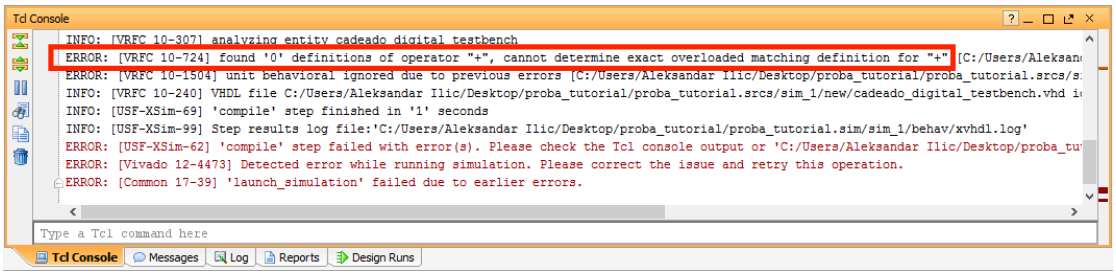

Figura 16. Simulação do circuito: Erros na descrição VHDL

Em particular, o erro apresentado na Figura 16 surge devido à falta da declaração da biblioteca *std_logic_unsigned* com predefinições requeridas para efectuar as operações aritméticas no *TestBench* (e.g., `code <= code + 1;`). Para corrigir este erro é preciso acrescentar:

```
use IEEE.STD_LOGIC_UNSIGNED.all;
```

na parte da declaração de bibliotecas (no ficheiro `cadeado_digital_testbench`), i.e., por baixo das linhas:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

G(3.4) 3.4. Após corrigir o erro, volte a iniciar a simulação. Neste caso, a ferramenta executa a simulação, mas apresenta valores <u>não definidos</u> para todas as entradas e saídas, conforme ilustrado na Figura 17.

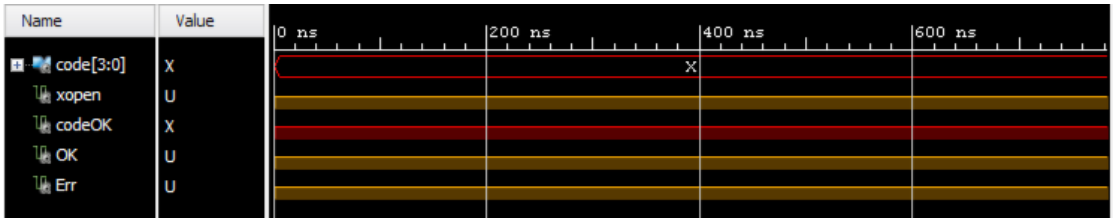

Figura 17. Simulação do circuito: Erros na descrição VHDL

Esta situação ocorre habitualmente quando os valores de algumas entradas não são totalmente especificados. Neste caso, os *processes* que geram os sinais `code` e `xopen` fazem as operações de soma e negação (respetivamente), mas o valor inicial destes sinais nunca foi definido. Por este motivo, as entradas do circuito apresentam um valor não definido, que será propagado para todas as saídas que delas dependem. Para corrigir esse erro, é necessário editar o *TestBench* e atribuir o valor inicial nos sinais `code` e `xopen` na parte onde estes sinais são declarados, ou seja, entre as linhas "***architecture*** *Behavioral* ***of*** *cadeado_digital_testbench* ***is***" e "***begin***", conforme ilustrado na Figura 18.

```
41    architecture Behavioral of cadeado_digital_testbench is
42    -- Declaracao do componente cadeado_digital original
43    component cadeado_digital
44        Port ( code : in STD_LOGIC_VECTOR (3 downto 0);
45               xopen : in STD_LOGIC;
46               codeOK : out STD_LOGIC;
47               OK : out STD_LOGIC;
48               Err : out STD_LOGIC);
49    end component;
50
51    -- Declaracao dos sinais para o testbench
52    signal code : STD_LOGIC_VECTOR (3 downto 0) := "0000";
53    signal xopen : STD_LOGIC := '0';
54    signal codeOK, OK, Err : STD_LOGIC;
55
56    begin
```

Figura 18. Atribuição do valor inicial dos sinais.

G(3.4 até 3.8) 3.5. Após corrigir este erro, volte a iniciar a simulação, considerando um tempo de simulação de 320ns e contemplando os seguintes sinais internos da `UTest`: *OK32*, *OK10* e *OK3210*. Compare os resultados da simulação com os valores apresentados na tabela de verdade (ver Tabela 1).
A simulação final deve ter um aspeto semelhante ao represando na Figura 19.

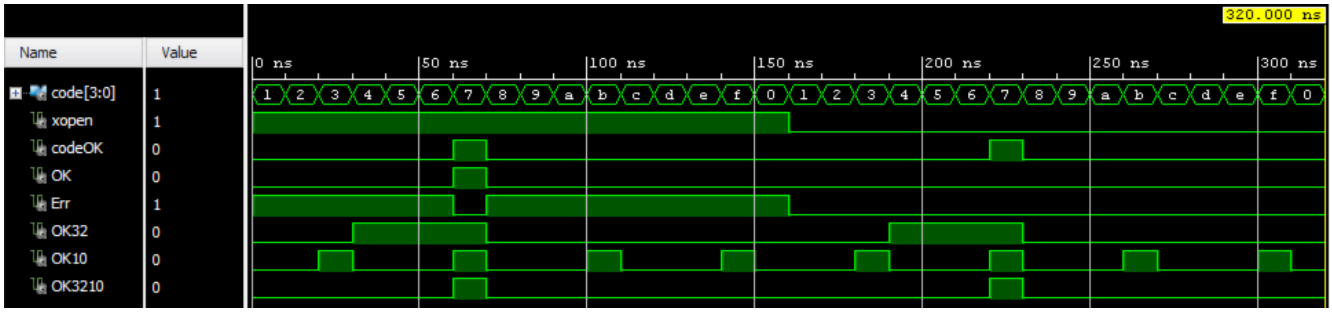

Figura 19. Simulação final do circuito `cadeado_digital`.