

---

Pontifícia Universidade Católica de Minas Gerais  
Instituto de Ciências Exatas e Informática  
Departamento de Engenharia de Computação

# Relatório: Trabalho Prático 2

## Registradores em VHDL

**Professores:** Antônio Hamilton Magalhães

Bruno Luiz Dias Alves de Castro  
Rafael Ramos de Andrade

Belo Horizonte  
Campus Coração Eucarístico

9 de novembro de 2024

## Conteúdo

<b>1</b>	<b>Introdução</b>	<b>3</b>
1.1	Objetivos . . . . .	3
1.2	Simulação via Quartus II . . . . .	3
<b>2</b>	<b>w_reg</b>	<b>3</b>
2.1	Implementação . . . . .	3
2.2	Simulação . . . . .	4
<b>3</b>	<b>fsr_reg</b>	<b>5</b>
3.1	Implementação . . . . .	5
3.2	Simulação . . . . .	6
<b>4</b>	<b>status_reg</b>	<b>7</b>
4.1	Implementação . . . . .	7
4.2	Simulação . . . . .	8
<b>5</b>	<b>stack</b>	<b>10</b>
5.1	Implementação . . . . .	10
5.2	Simulação . . . . .	11
<b>6</b>	<b>Conclusão</b>	<b>13</b>

# 1 Introdução

Durante as aulas da disciplina de Sistemas Reconfiguráveis, fomos introduzidos à linguagem VHDL. VHDL (**V**HSIC **H**ardware **D**escription **L**anguage) é uma linguagem de descrição de hardware. Com ela, podemos montar circuitos lógicos de maneira totalmente textual, o que garante à linguagem uma grande vantagem ante às soluções visuais.

## 1.1 Objetivos

## 1.2 Simulação via Quartus II

Nessa etapa realizamos testes no software Quartus II da altera.

# 2 w\_reg

O registrador w\_reg é um registrador simples. Possui um barramento de dados para escrita, habilitada por um sinal wr\_in.

As entradas e saídas do circuito são descritas na tabela a baixo:

Nome	Tamanho	Tipo	Descrição
nrst	1 bit	<i>Input</i>	Entrada de <i>reset</i> assíncrono.
clk_in	1 bit	<i>Input</i>	Entrada de <i>clock</i> .
d_in	8 bits	<i>Input</i>	Entrada de dados para escrita.
wr_en	1 bit	<i>Input</i>	Entrada de habilitação de escrita.
w_out	8 bits	<i>Output</i>	Saída de dados.

Tabela 1: Entradas e Saídas de fsr\_reg

## 2.1 Implementação

O registrador w\_reg foi implementado utilizando a linguagem VHDL.

O código na íntegra está abaixo:

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.std_logic_unsigned.all;
4 USE ieee.numeric_std.all;
5
6 ENTITY w_reg IS
7     PORT (
8         -- Inputs
9         nrst : IN STD_LOGIC;           -- Reset
10        clk_in: IN STD_LOGIC;           -- Clock
11        d_in:  IN STD_LOGIC_VECTOR(7 DOWNTO 0); -- Dados
12        wr_en : IN STD_LOGIC;           -- Enable
13
14        -- Outputs
15        w_out : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) -- Dados
16    );
17 END ENTITY;
18
19 ARCHITECTURE w_reg OF w_reg IS
20     SIGNAL mem_reg: STD_LOGIC_VECTOR(7 DOWNTO 0);
21 BEGIN
22
23     PROCESS (nrst, clk_in)
24     BEGIN
```

```

25     IF nrst = '0' THEN
26         mem_reg <= "00000000";
27     ELSIF RISING_EDGE(clk_in) then
28         IF wr_en = '1' THEN
29             mem_reg <= d_in;
30         END IF;
31     END IF;
32 END PROCESS;
33
34 w_out <= mem_reg;
35
36 END w_reg;

```

Listing 1: Código VHDL w\_reg

## 2.2 Simulação

Nesta imagem é realizado 3 testes para verificar a funcionalidade do registrador, nos primeiros 60ns é alterado os bits da entrada de dados (d\_in) para nível lógico alto, o bit de reset (nrst) que é ativo em baixa, é desativado, ou seja, nível lógico alto e o bit de ativação (wr\_en) é colocado em nível lógico alto após 10ns. Assim é possível verificar a mudança na saída (w\_out) com um tempo de delay de 6ns. No segundo teste a partir de 60ns até 140ns é resetado os bits da memória do registrador colocando reset em nível lógico zero, o resultado é propagada para a saída após o tempo de delay de aproximadamente 6ns. No terceiro teste foi verificados se o bit de ativação de escrita está funcionando corretamente, portanto com o bit 6 da saída em nível lógico alto esse valor será escrito apenas no tempo 160ns quando é colocado a porta de ativação do registrador em nível lógico alto e o registrador é escrito.

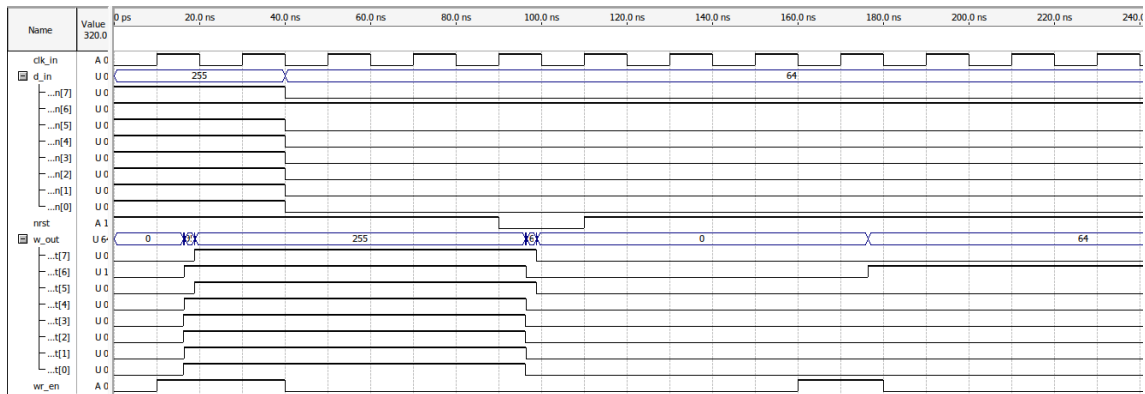


Figura 1: Simulação bloco w\_reg

### 3 fsr\_reg

O registrador FSR é um registrador semelhante ao implementado anteriormente. A principal diferença entre os dois está na presença de um sistema de endereçamento, e de duas entradas binárias independentes para habilitação da escrita e da leitura. Os requisitos são descritos na tabela abaixo.

Nome	Tamanho	Tipo	Descrição
nrst	1 bit	<i>Input</i>	Entrada de <i>reset</i> assíncrono.
clk_in	1 bit	<i>Input</i>	Entrada de <i>clock</i> .
abus_in	9 bit	<i>Input</i>	Entrada de endereçamento.
dbus_in	8 bits	<i>Input</i>	Entrada de dados para escrita.
wr_en	1 bit	<i>Input</i>	Entrada de habilitação de escrita.
rd_en	1 bit	<i>Input</i>	Entrada de habilitação de leitura.
dbus_out	8 bits	<i>Output</i>	Saída de dados habilitada por rd_en.

Tabela 2: Entradas e Saídas de fsr\_reg

#### 3.1 Implementação

O registrador fsr\_reg foi implementado utilizando a linguagem VHDL.

O código na íntegra está abaixo:

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.std_logic_unsigned.all;
4 USE ieee.numeric_std.all;
5
6 ENTITY fsr_reg IS
7     PORT (
8         -- Inputs
9         nrst : IN STD_LOGIC;           -- Reset
10        clk_in: IN STD_LOGIC;          -- Clock
11        abus_in: IN STD_LOGIC_VECTOR(8 DOWNTO 0); -- Endereçamento
12        dbus_in: IN STD_LOGIC_VECTOR(7 DOWNTO 0); -- Dados
13        wr_en : IN STD_LOGIC;          -- Enable escrita
14        rd_en : IN STD_LOGIC;          -- Enable leitura
15
16        -- Outputs
17        dbus_out : OUT STD_LOGIC_VECTOR(7 DOWNTO 0); -- Dados
18        fsr_out : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) -- Registrador
19    );
20 END ENTITY;
21
22 ARCHITECTURE fsr_reg OF fsr_reg IS
23     SIGNAL mem_reg: STD_LOGIC_VECTOR(7 DOWNTO 0);
24 BEGIN
25     PROCESS (nrst, clk_in, mem_reg, abus_in, dbus_in)
26     BEGIN
27         IF nrst = '0' THEN
28             mem_reg <= "00000000";
29         ELSIF abus_in(6 DOWNTO 0) = "0000100" THEN
30             IF RISING_EDGE(clk_in) THEN
31                 IF wr_en = '1' THEN
32                     mem_reg <= dbus_in;
33                 END IF;
34             END IF;
35         END IF;
36     END PROCESS;
37
38     dbus_out <= mem_reg WHEN rd_en = '1' ELSE "ZZZZZZZZ";
39     fsr_out <= mem_reg;
40 END fsr_reg;
```

Listing 2: Código VHDL fsr\_reg

### 3.2 Simulação

Para testar nosso código VHDL e certificar-nos de que nosso circuito funciona de maneira esperada, simulamos alguns casos de testes utilizando o software Quatus II.

Os testes realizados foram os seguintes:

1. Escrita com endereçamento incorreto (diferente de XX0000100).

- **Comportamento esperado:**

- dbus\_out em alta impedância;
- fsr\_out sem alteração;

2. Leitura habilitada e escrita desabilitada.

- **Comportamento esperado:**

- dbus\_out = frs\_out = último valor escrito;

3. Leitura desabilitada e escrita habilitada.

- **Comportamento esperado:**

- dbus\_out em alta impedância;
- frs\_out = dbus\_in;

4. *Reset* com leitura habilitada.

- **Comportamento esperado:**

- dbus\_out = frs\_out = “0b00000000”;

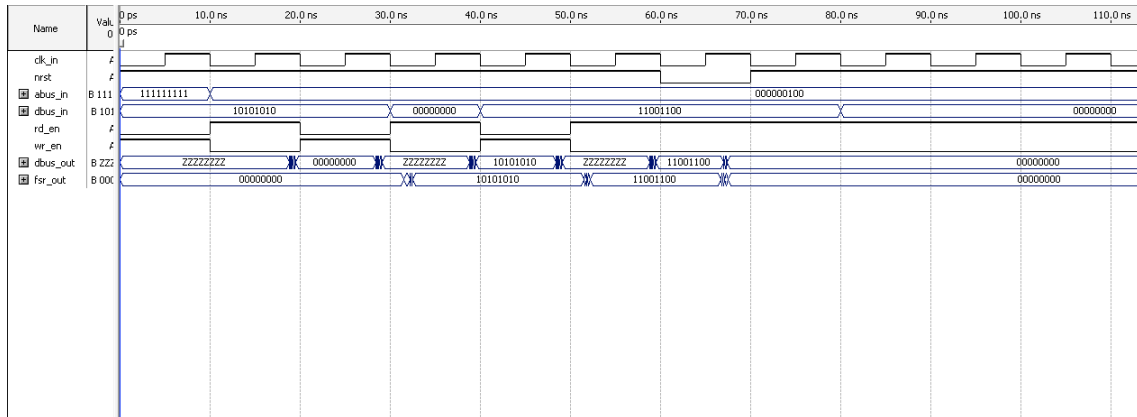


Figura 2: Simulação fsr\_reg

## 4 status\_reg

O status é um registrador semelhante ao implementado anteriormente. A diferença esta na presença de sinais de entrada e saída para controlar bits específicos. Assim como o anterior, existe um sistema de endereçamento que deve ser conferido para alterar o registrador.

As entradas e saídas do circuito são descritas na tabela a baixo:

Nome	Tamanho	Tipo	Descrição
nrst	1 bit	<i>Input</i>	Entrada de <i>reset</i> assíncrono.
clk_in	1 bit	<i>Input</i>	Entrada de <i>clock</i> .
abus_in	9 bit	<i>Input</i>	Entrada de endereçamento.
dbus_in	8 bits	<i>Input</i>	Entrada de dados para escrita.
wr_en	1 bit	<i>Input</i>	Entrada de habilitação de escrita.
rd_en	1 bit	<i>Input</i>	Entrada de habilitação de leitura.
z_in	1 bit	<i>Input</i>	Entrada de dado para escrita no bit 2 do registrador.
dc_in	1 bit	<i>Input</i>	Entrada de dado para escrita no bit 1 do registrador.
c_in	1 bit	<i>Input</i>	Entrada de dado para escrita no bit 0 do registrador.
z_wr_en	1 bit	<i>Input</i>	Entrada para habilitação da escrita no bit 2 do registrador.
dc_wr_en	1 bit	<i>Input</i>	Entrada para habilitação da escrita no bit 1 do registrador.
c_wr_en	1 bit	<i>Input</i>	Entrada para habilitação da escrita no bit 0 do registrador.
dbus_out	8 bits	<i>Output</i>	Saída de dados hailitada por rd_en.
irp_out	1 bit	<i>Output</i>	Saída correspondente ao bit 7 do registrador.
rp_out	2 bits	<i>Output</i>	Saída correspondente aos bits 6 e 5 do registrador.
z_out	1 bit	<i>Output</i>	Saída correspondente ao bit 2 do registrador.
dc_out	1 bit	<i>Output</i>	Saída correspondente ao bit 1 do registrador.
c_out	1 bit	<i>Output</i>	Saída correspondente ao bit 0 do registrador.

Tabela 3: Entradas e Saídas de status\_reg

### 4.1 Implementação

O status\_reg foi implementado utilizando a linguagem VHDL.

O código na íntegra está abaixo:

```
1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.std_logic_unsigned.all;
4 USE ieee.numeric_std.all;
5
6 ENTITY status_reg IS
7     PORT (
8         -- Inputs
9         nrst: IN STD_LOGIC;                -- Reset
10        clk_in: IN STD_LOGIC;               -- Clock
11        abus_in: IN STD_LOGIC_VECTOR(8 DOWNTO 0); -- Endereçamento
12        dbus_in: IN STD_LOGIC_VECTOR(7 DOWNTO 0); -- Dados
13        wr_en: IN STD_LOGIC;               -- Enable escrita
14        rd_en: IN STD_LOGIC;               -- Enable leitura
15        z_in: IN STD_LOGIC;                -- Dados bit 2
16        dc_in: IN STD_LOGIC;               -- Dados bit 1
17        c_in: IN STD_LOGIC;                -- Dados bit 0
18        z_wr_en: IN STD_LOGIC;             -- Enable escrita bit 2
19        dc_wr_en: IN STD_LOGIC;            -- Enable escrita bit 1
20        c_wr_en: IN STD_LOGIC;             -- Enable escrita bit 0
21
22        -- Outputs
23        dbus_out: OUT STD_LOGIC_VECTOR(7 DOWNTO 0); -- Dados
24        irp_out: OUT STD_LOGIC;             -- Dados bit 7
25        rp_out: OUT STD_LOGIC_VECTOR(1 DOWNTO 0); -- Dados bit 6 e 5
```

```

26     z_out: OUT STD_LOGIC;           -- Dados bit 2
27     dc_out: OUT STD_LOGIC;         -- Dados bit 1
28     c_out: OUT STD_LOGIC           -- Dados bit 0
29 );
30 END ENTITY;
31
32 ARCHITECTURE status_reg OF status_reg IS
33     SIGNAL mem_reg: STD_LOGIC_VECTOR(7 downto 0);
34 BEGIN
35     PROCESS(nrst, clk_in, mem_reg, wr_en, z_in, dc_in, c_in)
36     BEGIN
37         IF nrst = '0' THEN
38             mem_reg <= "00000000";
39         ELSIF RISING_EDGE(clk_in) THEN
40             IF wr_en = '1' AND abus_in(6 DOWNT0 0) = "0000011" THEN
41                 mem_reg <= dbus_in;
42             END IF;
43             IF z_wr_en = '1' THEN
44                 mem_reg(2) <= z_in;
45             END IF;
46             IF dc_wr_en = '1' THEN
47                 mem_reg(1) <= dc_in;
48             END IF;
49             IF c_wr_en = '1' THEN
50                 mem_reg(0) <= c_in;
51             END IF;
52         END IF;
53     END PROCESS;
54
55     dbus_out <= mem_reg WHEN rd_en = '1' AND abus_in(6 DOWNT0 0) = "0000011" ELSE "ZZZZZZZZ";
56     irp_out <= mem_reg(7);
57     rp_out <= mem_reg(6 DOWNT0 5);
58     z_out <= mem_reg(2);
59     dc_out <= mem_reg(1);
60     c_out <= mem_reg(0);
61 END status_reg;

```

Listing 3: Código VHDL status\_reg

## 4.2 Simulação

Para testar nosso código VHDL e certificar-nos de que nosso circuito funciona de maneira esperada, simulamos alguns casos de testes utilizando o software Quatus II.

Os testes realizados foram os seguintes:

1. Escrita com endereçamento incorreto (diferente de “0bXX0000011”).

- **Comportamento esperado:**

- dbus\_out em alta impedância;
- rp\_out = “0b00”
- irp\_out = z\_out = dc\_out = c\_out = “0b0”

2. Leitura desabilitada e escrita habilitada;

- dbus\_in = “0b01011000”.
- z\_in = dc\_in = c\_in = 1;
- z\_wr\_en = dc\_wr\_en = c\_wr\_en = 0;

- **Comportamento esperado:**

- dbus\_out em alta impedância;
- rp\_out = “10b”
- irp\_out = z\_out = dc\_out = c\_out = “0b0”

3. Leitura habilitada e escrita desabilitada.



- Valor salvo = “0b01011000”.
- **Comportamento esperado:**
  - dbus\_out = “0b01011000”;
  - rp\_out = “10b”
  - irp\_out = z\_out = dc\_out = c\_out = “0b0”

4. Leitura desabilitada e escrita habilitada;

- dbus\_in = “0b10100000”.
- z\_in = dc\_in = c\_in = 1;
- z\_wr\_en = dc\_wr\_en = c\_wr\_en = 1;
- **Comportamento esperado:**
  - dbus\_out em alta impedância;
  - rp\_out = “01b”
  - irp\_out = z\_out = dc\_out = c\_out = “0b1”

5. Leitura habilitada e escrita desabilitada.

- Valor salvo = “0b10100111”.
- **Comportamento esperado:**
  - dbus\_out = “0b10100111”;
  - rp\_out = “10b”
  - irp\_out = z\_out = dc\_out = c\_out = “0b1”

6. *Reset* com leitura habilitada.

- **Comportamento esperado:**
  - dbus\_out = “0b00000000”;
  - rp\_out = “00b”
  - irp\_out = z\_out = dc\_out = c\_out = “0b0”

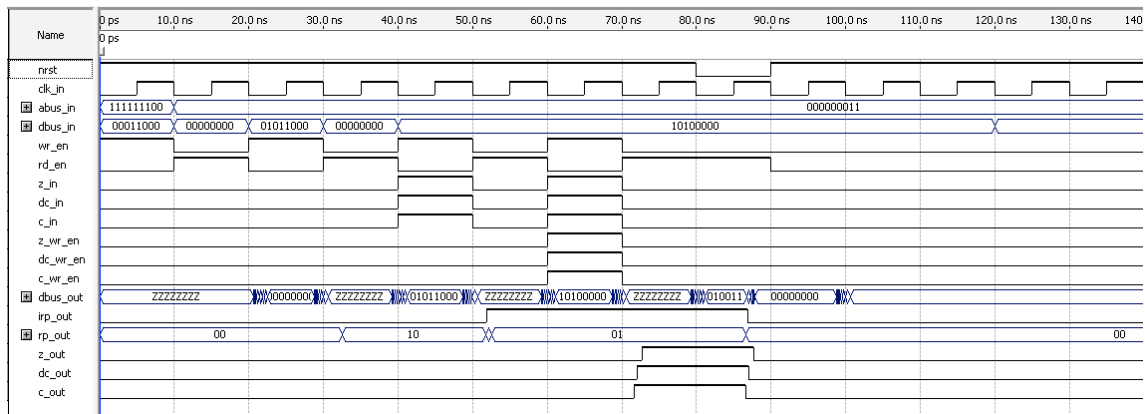


Figura 3: Simulação status\_reg

## 5 stack

O bloco **stack** é um conjunto de 8 registradores de 13 bits. As operações de *push* e *pop* adicionam e removem dados da pilha, respectivamente.

As entradas e saídas deste bloco estão descritas na tabela abaixo:

Nome	Tamanho	Tipo	Descrição
nrst	1 bit	<i>Input</i>	Entrada de <i>reset</i> assíncrono.
clk_in	1 bit	<i>Input</i>	Entrada de <i>clock</i> .
stack_in	13 bit	<i>Input</i>	Entrada de dados para a pilha.
stack_push	1 bit	<i>Input</i>	Entrada de habilitação para colocar valores na pilha.
stack_pop	1 bit	<i>Input</i>	Entrada de habilitação para retirar valores da pilha.
stack_out	13 bits	<i>Output</i>	Saída correspondente à primeira posição da pilha.

Tabela 4: Entradas e Saídas do bloco stack

### 5.1 Implementação

O bloco stack foi implementado utilizando a linguagem VHDL.

O código na íntegra está abaixo:

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.std_logic_unsigned.all;
4  USE ieee.numeric_std.all;
5
6  ENTITY stack IS
7      PORT (
8          -- Inputs
9          nrst: IN STD_LOGIC;                -- Reset
10         clk_in: IN STD_LOGIC;               -- Clock
11         stack_in: IN STD_LOGIC_VECTOR(12 DOWNTO 0); -- Dados
12         stack_push: IN STD_LOGIC;           -- Enable push op
13         stack_pop: IN STD_LOGIC;            -- Enable pop op
14
15         -- Outputs
16         stack_out: OUT STD_LOGIC_VECTOR(12 DOWNTO 0) -- Stack output
17     );
18 END ENTITY;
19
20 ARCHITECTURE stack OF stack IS
21     SIGNAL mem_reg1, mem_reg2, mem_reg3, mem_reg4, mem_reg5, mem_reg6, mem_reg7, mem_reg8 :
22     STD_LOGIC_VECTOR(12 DOWNTO 0);
23 BEGIN
24     PROCESS(nrst, clk_in, stack_push, stack_pop)
25     BEGIN
26         IF nrst = '0' THEN
27             mem_reg1 <= "0000000000000";
28             mem_reg2 <= "0000000000000";
29             mem_reg3 <= "0000000000000";
30             mem_reg4 <= "0000000000000";
31             mem_reg5 <= "0000000000000";
32             mem_reg6 <= "0000000000000";
33             mem_reg7 <= "0000000000000";
34             mem_reg8 <= "0000000000000";
35         ELSIF RISING_EDGE(clk_in) THEN
36             stack_out <= "0000000000000";
37             IF stack_pop = '1' THEN
38                 stack_out <= mem_reg1;
39                 mem_reg1 <= mem_reg2;
40                 mem_reg2 <= mem_reg3;
41                 mem_reg3 <= mem_reg4;
42                 mem_reg4 <= mem_reg5;
```

```

42         mem_reg5 <= mem_reg6;
43         mem_reg6 <= mem_reg7;
44         mem_reg7 <= mem_reg8;
45         mem_reg8 <= "000000000000";
46     ELSIF stack_push = '1' THEN
47         mem_reg8 <= mem_reg7;
48         mem_reg7 <= mem_reg6;
49         mem_reg6 <= mem_reg5;
50         mem_reg5 <= mem_reg4;
51         mem_reg4 <= mem_reg3;
52         mem_reg3 <= mem_reg2;
53         mem_reg2 <= mem_reg1;
54         mem_reg1 <= stack_in;
55     END IF;
56 END IF;
57 END PROCESS;
58 END stack;

```

Listing 4: Código VHDL status\_reg

## 5.2 Simulação

Para testar nosso código VHDL e certificar-nos de que nosso circuito funciona de maneira esperada, simulamos alguns casos de testes utilizando o software Quatus II.

Os testes realizados foram os seguintes:

1. *Push* até pilha cheia.

- stack\_in: Sequência de “0” à “7”;
- stack\_push = “1”;
- stack\_pop = “0”;
- **Comportamento esperado:**
  - stack\_out = “0”;

2. *Pop* até pilha vazia.

- stack\_push = “0”;
- stack\_pop = “1”;
- **Comportamento esperado:**
  - stack\_out = Sequência de “7” à “0”;

3. *Push* até *Stack Overflow*.

- stack\_in: Sequência de “0” à “9”;
- stack\_push = “1”;
- stack\_pop = “0”;
- **Comportamento esperado:**
  - stack\_out = “0”;

4. *Pop* até *Stack Underflow*.

- stack: Sequência de “9” à “2”;
- stack\_push = “0”;
- stack\_pop = “1”;
- **Comportamento esperado:**
  - stack\_out = Sequência de “9” à “2”, depois, “0”;

## 5. *Push* e *Pop* simultâneo.

- stack: “1”;
- stack\_push = “1”;
- stack\_pop = “1”;
- **Comportamento esperado:**
  - Preferência do *pop*.
  - stack\_out = “1”, depois, “0”;

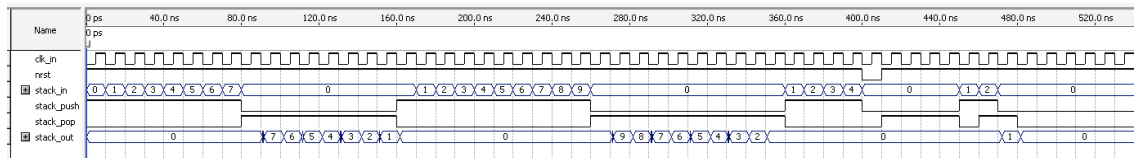


Figura 4: Simulação bloco stack

## 6 Conclusão

Com estes dois projetos simples, tivemos um excelente primeiro contado com a linguagem VHDL, bem como à programação concorrente e desenvolvimento de circuitos FPGA. Os dois circuitos implementados (Multiplexador de Endereçamento e Unidade Lógica Aritimética) são blocos de construção chave para a maior parte dos circuitos complexos, e serão de suma importância não só para os demais trabalhos práticos que realizaremos ao longo do semestre, mas para nosso desenvolvimento acadêmico e profissional.