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

Relatório: Trabalho Prático 1

Multiplexador de Endereçamento e ULA

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

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 à soluções visuais.

1.1 Objetivos

1.2 Simulação via Quartus II

Nessa etapa realizamos testes no software Quartus II da altera.

1.2.1 Bloco w_reg

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 nivel 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 colocoado 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.

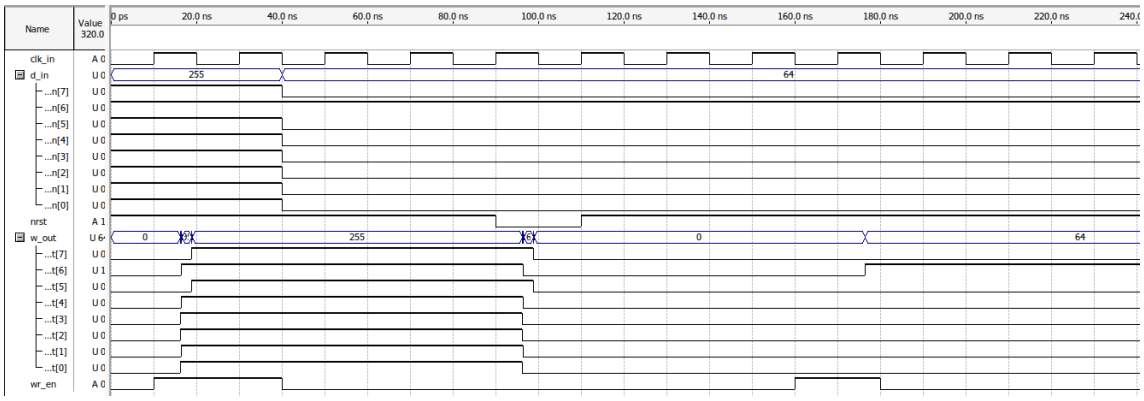

Figura 1: Simulação bloco w_reg

2 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.

2.1 Implementação

O registrador fsr_reg foi implementado utilizando a linguagem VHDL.

O código na íntegra está abaixo:

Nome	Tamanho	Tipo	Descrição
nrst	1 bit	*Input*	Entrada de *reset* assíncrono.
clk_in	1 bit	*Input*	Entrada de *clock*.
abus_in	9 bit	*Input*	Entrada de enderençamento.
dbus_in	8 bits	*Input*	Entrada de dados para escrita.
wr_en	1 bit	*Input*	Entrada de habilitação de escrita.
rd_en	1 bit	*Input*	Entrada de habilitação de leitura.
dbus_out	8 bits	*Output*	Saída de dados hailitada por rd_en.

Tabela 1: Entradas e Saídas de fsr_reg

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;

ENTITY fsr_reg IS
    PORT (
        -- Inputs
        nrst : IN STD_LOGIC;                              -- Reset
        clk_in: IN STD_LOGIC;                             -- Clock
        abus_in: IN STD_LOGIC_VECTOR(8 DOWNTO 0);         -- Enderecamento
        dbus_in: IN STD_LOGIC_VECTOR(7 DOWNTO 0);         -- Dados
        wr_en : IN STD_LOGIC;                             -- Enable escrita
        rd_en : IN STD_LOGIC;                             -- Enable leitura

        -- Outputs
        dbus_out : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);      -- Dados
        fsr_out : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)        -- Registrador
    );
END ENTITY;

ARCHITECTURE fsr_reg OF fsr_reg IS
    SIGNAL mem_reg: STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
    PROCESS (nrst, clk_in, mem_reg, abus_in, dbus_in)
    BEGIN
        IF nrst = '0' THEN
            mem_reg <= "00000000";
        ELSIF abus_in(6 DOWNTO 0) = "0000100" THEN
            IF RISING_EDGE(clk_in) THEN
                IF wr_en = '1' THEN
                    mem_reg <= dbus_in;
                END IF;
            END IF;
        END IF;
    END PROCESS;

    dbus_out <= mem_reg WHEN rd_en = '1' ELSE "ZZZZZZZZ";
    fsr_out <= mem_reg;
END fsr_reg;
```

Listing 1: Código VHDL fsr_reg

2.2 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 seguites:

1. Escrita com enderaçamento 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 impendância;
     - frs_out = dbus_in;

4. *Reset* com leitura habilitada.

   - **Comportamento esperado:**
     - dbus_out = frs_out = “0b00000000”;

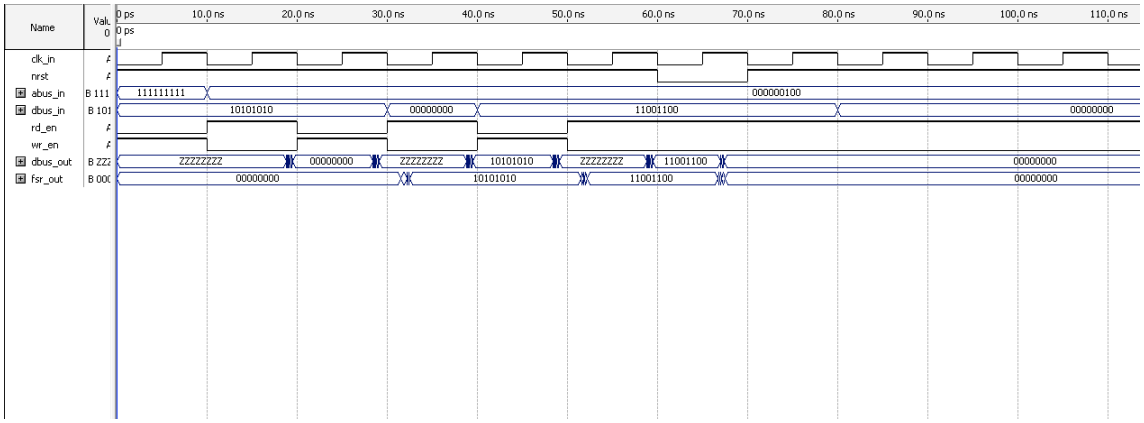

Figura 2: Simulação fsr_reg

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

Tabela 2: Entradas e Saídas de status_reg

3.1 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);       -- Enderecamento
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
```

```
        z_out: OUT STD_LOGIC;                           -- Dados bit 2
        dc_out: OUT STD_LOGIC;                          -- Dados bit 1
        c_out: OUT STD_LOGIC                            -- Dados bit 0
    );
END ENTITY;

ARCHITECTURE status_reg OF status_reg IS
    SIGNAL mem_reg: STD_LOGIC_VECTOR(7 downto 0);
BEGIN
    PROCESS(nrst, clk_in, mem_reg, wr_en, z_in, dc_in, c_in)
    BEGIN
        IF nrst = '0' THEN
            mem_reg <= "00000000";
        ELSIF RISING_EDGE(clk_in) THEN
            IF wr_en = '1' AND abus_in(6 DOWNTO 0) = "0000011" THEN
                mem_reg <= dbus_in;
            END IF;
            IF z_wr_en = '1' THEN
                mem_reg(2) <= z_in;
            END IF;
            IF dc_wr_en = '1' THEN
                mem_reg(1) <= dc_in;
            END IF;
            IF c_wr_en = '1' THEN
                mem_reg(0) <= c_in;
            END IF;
        END IF;
    END PROCESS;

    dbus_out <= mem_reg WHEN rd_en = '1' AND abus_in(6 DOWNTO 0) = "0000011" ELSE "ZZZZZZZZ";
    irp_out <= mem_reg(7);
    rp_out <= mem_reg(6 DOWNTO 5);
    z_out <= mem_reg(2);
    dc_out <= mem_reg(1);
    c_out <= mem_reg(0);
END status_reg;
```

Listing 2: Código VHDL status_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 seguites:

1. Escrita com enderaçamento 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 impendância;
     - 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 impendância;
     - 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”

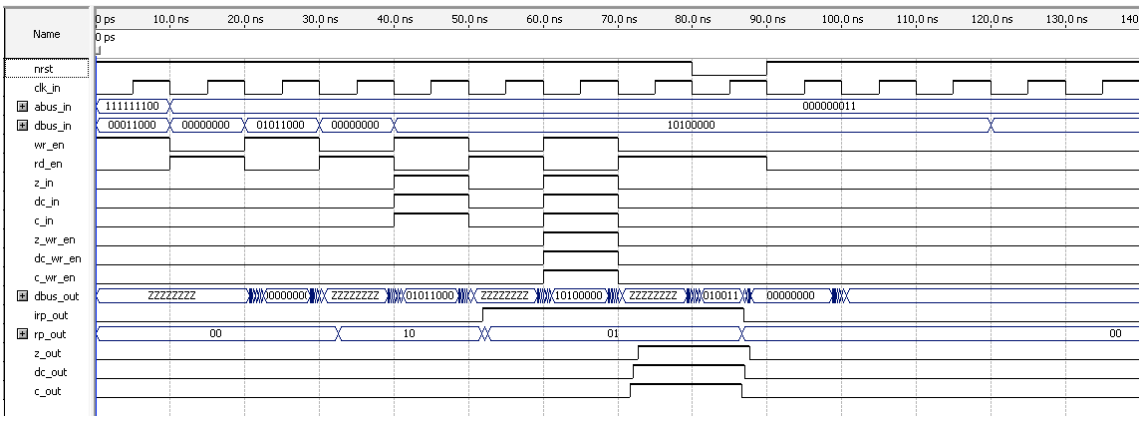

Figura 3: Simulação status_reg

4 status_reg

4.1 Implementação

4.2 Simulação

4.3

5 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.