LABORATÓRIO DE ARQUITETURA DE COMPUTADORES

Experimento 1

Instruction Fetch

(Memória de Instrução e Contador de Programa)

SUMÁRIO

1 Introdução teórica 1
1.1 Componentes 1
1.2 ALTSYNCRAM 1
2 Procedimento Preparatório 3
3 Experimento 5
4 Resultados 5
5 Bibliografia 6

1 Introdução teórica

Projetos em hierarquia permitem o desenvolvimento modular (unidades de projeto) e re-uso de partes de projeto. Um componente é uma entidade de projeto empregada na arquitetura de outra entidade (D'Amore, 2005). A utilização desses elementos permite a interligação de múltiplas entidades de projeto, de modo a formar uma entidade mais complexa em um projeto hierárquico. **A interligação dos componentes é similar à definição de uma rede de ligações em um diagrama esquemático, onde todos os pontos interligados recebem uma identificação.**

Uma unidade de memória pode ser inserida como um componente do sistema. Como exemplo, utilizaremos o módulo de memória ALTSYNCRAM para formar o circuito do experimento de hoje. As sub-seções a seguir descrevem os conceitos envolvidos.

1.1 Componentes

Para a utilização de um componente em uma arquitetura, é necessário que se faça uma declaração do componente. A declaração contém a lista de portas e pode conter uma lista de genéricos, semelhante à declaração de uma entidade. **A declaração de um componente é equivalente à declaração da entidade que descreve o componente.**

```
COMPONENT nome_componente_x
        GENERIC (n : tipo_n := valor);
        PORT (   sinal_a : modo_a   tipo_sinal_a;
                 sinal_b : modo_b   tipo_sinal_b;
                 sinal_c : modo_c   tipo_sinal_c);
END COMPONENT;
```

A declaração de componentes pode ser inserida no corpo da arquitetura ou em pacotes. Na arquitetura os sinais dos componentes são mapeados para sinais da unidade integradora (ver exemplo em 1.2).

1.2 ALTSYNCRAM

Um módulo de memória pode ser representado como na Figura 1. O sinal ***Address*** indica a posição de memória a ser lida ou escrita. A entrada ***Write Data*** deve conter um dado quando o sinal ***MemWrite*** estiver **ativo alto**, habilitando para a escrita do dado na posição de memória endereçada. Quando ***MemWrite*** é **ativo baixo**, o dado da posição de memória endereçada é apresentado na saída ***Read Data***.

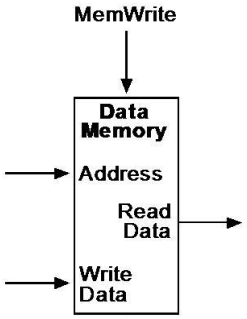

Figura 1 – Unidade de Memória

Uma unidade de memória ALTSYNCRAM pode ser inserida incluindo a biblioteca correspondente e realizando o mapeamento das portas e constantes genéricas, como mostra o exemplo da Figura 7. Este exemplo mostra uma unidade de memória ROM (somente leitura),

utilizando palavras de 32 bits (Word), e 256 posições (8 bits de endereço) de memória. A memória deve ser sensível à borda de subida do clock.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;  -- Tipo de sinal STD_LOGIC e STD_LOGIC_VECTOR
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.ALL; -- Componente de memoria

ENTITY Mem IS
    PORT(<descreva aqui as portas in e out de acesso a sua unidade Mem>);
END Mem;
ARCHITECTURE behavior OF Mem IS
<descreva aqui os demais sinais internos>
BEGIN
    -- Descrição da Memória
    data_memory: altsyncram -- Declaracao do compomente de memoria
    GENERIC MAP(
            operation_mode => "ROM",
            width_a        => 32, -- tamanho da palavra (Word)
            widthad_a      => 8,  -- tamanho do barramento de endereco
            lpm_type       => "altsyncram",
            outdata_reg_a  => "UNREGISTERED",
            init_file      => "program.mif",  -- arquivo com estado inicial
            intended_device_family => "Cyclone")
    PORT MAP(
            address_a      => <insira o sinal com o endereço>,
            q_a            => <insira o sinal de dado de saída>,
            clock0         => <insira o sinal de clock>);
END behavior;
```

Figura 2 – EXEMPLO de Estrutura da unidade de Memória ROM

Um arquivo contendo o estado inicial da memória pode ser definido. No caso do exemplo da Figura 2, foi definido um arquivo de nome ***program.mif***. A Figura 3 mostra um exemplo de arquivo para definição do estado inicial de memória.

```
--  Configuracao da memoria
DEPTH = 256;          % valor decimal com numero de enderecos %
WIDTH = 32;           % valor decimal com numero de bits da palavra %
-- Configuracao da apresentacao
ADDRESS_RADIX = HEX;      % Opcional: endereco em formato hexadecimal %
DATA_RADIX = HEX;         % Opcional: dado em formato hexadecimal %
-- Define o conteudo inicial da memoria
Content
Begin
-- atribui valores a enderecos especificos
 00 : 8C020000;      % atribui dado hexa 8C020000 para endereco 00 %
 01 : 8C030001;      % atribui dado hexa 8C030001 para endereco 01 %
 02 : 00430820;      % atribui dado hexa 00430820 para endereco 02 %
 03: AC010003;
 04: 1022FFFF;
 05: 1021FFFA;
-- valores defaut, preenche memoria com zeros
 [06..FF] : 00000000;
End;
```

Figura 3 – Estado Inicial de Memória

2 Procedimento Preparatório

Este experimento deve implementar o circuito da Figura 4, composto por 3 unidades: a unidade Top Level (Exp01) e os componentes Ifetch e LCD_Display. A unidade Ifetch (Figura 5) por sua vez é composta por 3 partes: o componente de memória, o somador (ADD) e o registrador (PC).

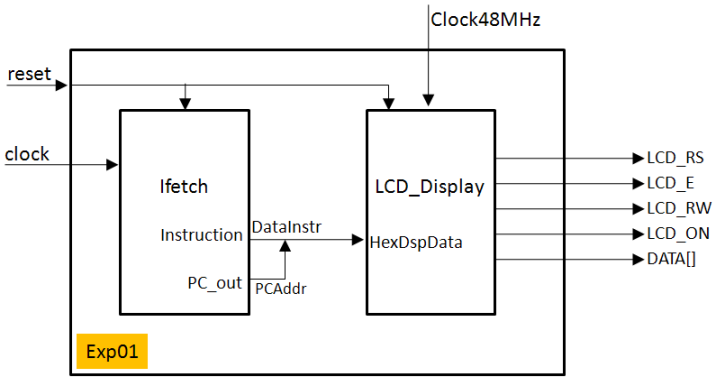

Figura 4 – Arquitetura do Experimento com 2 componentes: Ifetch e LCD_Display

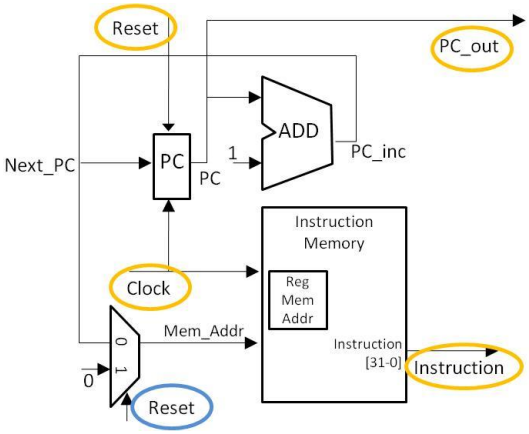

Figura 5 – Unidade Instruction Fetch

A estrutura básica para a descrição das 3 unidades de projeto (arquivos VHDL) é fornecida. A unidade LCD_Display não deve ser editada, esta estrutura fornece um *driver* para apresentar o endereço de memória sendo lido e seu conteúdo. As unidades Exp01 e Ifetch estão incompletas e devem ser finalizadas. A descrição fornecida para a unidade Ifetch é apresentada na Figura 7. Utilize obrigatoriamente os nomes de sinais indicados nas figuras.

Obs: o MIPS endereça o contador de programa (PC) byte a byte e endereça a memória (Mem_Addr) palavra a palavra (Word de 4 bytes), mas vamos endereçar ambos por palavra.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY Exp01 IS
    -- Mapear os sinais de entrada e saida para os pinos físicos da placa
    PORT(   reset                   : IN STD_LOGIC; -- SW0 de reset
            Clock48MHz              : IN STD_LOGIC; -- Clock da plada para o LCD
            -- Mapeamento do LCD
            LCD_RS, LCD_E           : OUT    STD_LOGIC;
            LCD_RW, LCD_ON          : OUT STD_LOGIC;
            DATA                    : INOUT  STD_LOGIC_VECTOR(7 DOWNTO 0);
            clock                   : IN STD_LOGIC); -- Pushbutton Key0 para clock
END Exp01;
```

```
ARCHITECTURE exec OF Exp01 IS
-- Declaração do componente LCD_Display
COMPONENT LCD_Display
    GENERIC(NumHexDig: Integer:= 11);
    PORT(   reset, clk_48Mhz : IN STD_LOGIC;
            HexDisplayData   : IN STD_LOGIC_VECTOR((NumHexDig*4)-1 DOWNTO 0);
            LCD_RS, LCD_E    : OUT STD_LOGIC;
            LCD_RW           : OUT STD_LOGIC;
            DATA_BUS         : IN OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;
-- Declaração do component Ifetch
COMPONENT Ifetch
    PORT(< Descreva aqui a declaração do componente Ifetch>);
END COMPONENT;
-- Lista de sinais internos
SIGNAL DataInstr     : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL PCAddr        : STD_LOGIC_VECTOR(9 DOWNTO 0);
BEGIN
    LCD_ON <= '1';
    lcd: LCD_Display
    PORT MAP(       reset           => reset,
                    clk_48Mhz       => clock48MHz,
                    HexDisplayData  => <insira aqui o que visualizar>,
                    LCD_RS          => LCD_RS,
                    LCD_E           => LCD_E,
                    LCD_RW          => LCD_RW,
                    DATA_BUS        => DATA);

    IFT: Ifetch
    PORT MAP(<Insira aqui o mapeamento para a unidade Ifetch>);
END exec;
```

Figura 6 – Estrutura da unidade de Memória e indicações para a fase de Instruction Fetch

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;  -- Tipo de sinal STD_LOGIC e STD_LOGIC_VECTOR
USE IEEE.STD_LOGIC_ARITH.ALL;  -- Operacoes aritmeticas
USE IEEE.STD_LOGIC_UNSIGNED.ALL; -- Operacoes aritmeticas sobre binarios
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.ALL; -- Componente de memoria
ENTITY Ifetch IS
    PORT(<descreva aqui as portas in e out da sua unidade Instruction Fetch>);
END Ifetch;
ARCHITECTURE behavior OF Ifetch IS
<descreva aqui os demais sinais internos>
BEGIN
    -- Descrição da Memória
    data_memory: altsyncram -- Declaracao do compomente de memoria
    GENERIC MAP(
            operation_mode => "ROM",
            width_a        => 32, -- tamanho da palavra (Word)
            widthad_a      => 8,  -- tamanho do barramento de endereco
            lpm_type       => "altsyncram",
            outdata_reg_a  => "UNREGISTERED",
            init_file      => "program.mif",  -- arquivo com estado inicial
            intended_device_family => "Cyclone")
```

```
PORT MAP(
        address_a       => <insira o sinal com o endereço (parte do PC)>,
        q_a             => <insira o sinal de dado de saída (Instruction)>,
        clock0          => <insira o sinal de clock>);
    -- Descricao do somador
    <insira aqui o somador>
    -- Descricao do registrador
    < insira aqui o registrador>

    <insira qualquer codigo adicional para interligar as partes: registrador, somador e memoria atraves dos sinais internos>
END behavior;
```

Figura 7 – Estrutura da unidade de Memória e indicações para a fase de Instruction Fetch

O circuito deve funcionar da seguinte forma:

- O sinal de reset deve zerar o PC.
- A cada pulso do clock (SW0) do sistema o PC deve ser incrementado de forma a ler a próxima instrução e apresentar no display de LCD o endereço da memória sendo lida e seu conteúdo.
- O display de LCD deve receber os sinais a serem apresentados no seguinte formato: bits complementares iguais a zero (4 bits, "0000") + endereço da memória (08 bits) + instrução (32 bits). Os bits devem ser concatenados nesta ordem.

3 Experimento

Implemente a descrição no simulador Quartus II e compile. Simule e teste o modelo descrito para verificar se o comportamento é como o esperado.

- Crie um sinal de *clock* com período de 100pS e *duty cycle* de 50%, início ativo alto (1).
- Crie um sinal de reset que inicia ativo alto (1) e permanece até 30pS, indo para zero até o final da simulação.
- Adicione os sinais de saída que deseja monitorar.

Somente se a simulação for bem sucedida, faça *upload* para a placa após o *check list*. Para teste na placa faça:

- Ative o reset, após desative o reset.
- O conteúdo da primeira posição de memória deve aparecer no display de LCD.
- Pulse o clock para ler as demais posições.

4 Resultados

Faça um relatório e entregue até data definida, através do moodle em arquivo pdf e impressa na aula seguinte à data de entrega.

- Capa com nome dos membros do grupo.
- Resumo do experimento (1/2 a 1 página)
- Código VHDL da entidade Ifetch
- Imagem do simulador com as ondas de simulação e teste.

- Explicação pontual **RELACIONANDO** o código VHDL com as mudanças nos sinais de saída para a imagem de simulação apresentada. Esta explicação deve deixar claro que a simulação representa o funcionamento correto do código VHDL descrito.

5 Bibliografia

D'Amore, R. VHDL: Descrição e Síntese de Circuitos Digitais. LTC. 2005.