Universidade de Brasília
INSTITUTO DE CIÊNCIAS EXATAS – IE
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO – CIC

Disciplina: CIC 116394 – Organização e Arquitetura de Computadores – Turma C
Semestre: 1/2017
Professor: Marcelo Grandi Mandelli

TRABALHO 3 – ULA E BANCO DE REGISTRADORES

OBJETIVOS

- Compreender o funcionamento de uma Unidade Lógica e Aritmética (ULA) e de um banco de registradores;
- Compreender o processo de desenvolvimento e simulação de projetos de hardware utilizando a linguagem VHDL;

ESPECIFICAÇÃO DO TRABALHO

Este trabalho consiste no desenvolvimento e simulação de uma Unidade Lógica e Aritmética (ULA) e de um banco de registradores utilizando a linguagem VHDL. Para isso, as ferramentas Quartus II e Modelsim-Altera serão utilizadas.

Unidade Lógica e Aritmética (ULA)

A Unidade Lógica e Aritmética (ULA) a ser implementada deverá ter as seguintes características:

- O tamanho das palavras de dados da ULA (operandos e resultado) será definido por uma constante DATA_WIDTH, definida na entidade da ULA utilizando a primitiva generic.
- Por padrão, a constante DATA_WIDTH terá tamanho 32, indicando que a ULA fará operações de dados de 32 bits.
- A ULA terá duas entradas de dados **input1** e **input2**, as quais terão tamanho de palavra DATA_WIDTH.
- As entradas **input1** e **input2** serão os operandos da operação a ser realizada pela ULA.
- A operação a ser realizada pela ULA será definida pela entrada **operation** de tamanho 4 bits.
- A ULA terá uma saída **output**, a qual terá tamanho de palavra DATA_WIDTH.
- A saída **output** da ULA apresentará o resultado da operação definida em **operation** entre os operandos definidos por **input1** e **input2.**
- A ULA terá uma saída **zero** (de tamanho um bit), a qual detectará um valor zero na saída **output.** Se a saída **output** for igual a 0 a saída **zero** será 1, caso contrário será 0.

- A ULA terá uma saída **negative** (de tamanho um bit), a qual detectará um valor negativo na saída **output.** Se a saída **output** apresentar um valor negativo a saída **negative** será 1, caso contrário será 0.
- A ULA terá uma saída **carry** (de tamanho um bit), a qual terá o valor do último carry nas operações de soma da ULA.
- A ULA terá uma saída **overflow** (de tamanho um bit), a qual detectará overflow nas operações de soma e subtração da ULA. A saída **overflow** será 1 quando uma operação de soma ou subtração gerar resultado que ultrapasse o limite de representação do tamanho da palavra DATA_WIDTH, caso contrário será 0.
- **As operações serão realizadas somente em complemento de 2.**
- As operações que serão realizadas pela ULA são apresentadas na tabela a seguir, considerando que (A é **input1**, B é **input2** e Z é **output**):

Operação	Significado	**operation**
and A, B	Z recebe a operação lógica A and B, bit a bit	0000
or A, B	Z recebe a operação lógica A or B, bit a bit	0001
add A, B	Z recebe a soma das entradas A, B	0010
sub A, B	Z recebe A - B	0011
slt A, B	Z = 1 se A < B	0100
nor A, B	Z recebe a operação lógica A nor B, bit a bit	0101
xor A, B	Z recebe a operação lógica A xor B, bit a bit	0110
sll A, B	Z recebe A deslocado em B bits à esquerda	0111
srl A, B	Z recebe A deslocado em B bits à direita sem ext. de sinal	1000
sra A, B	Z recebe A deslocado em B bits à direita com ext. de sinal	1001

Os packages e entidade da ULA em VHDL será :

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

entity alu is
	generic (DATA_WIDTH : natural := 32);
	port (
		input1, input2		: in std_logic_vector(DATA_WIDTH -1 downto 0);
		operation		: in std_logic_vector(3 downto 0);
		output			: out std_logic_vector(DATA_WIDTH -1 downto 0);
		zero, negative		: out std_logic;
		carry, overflow		: out std_logic
	);
end entity alu;
```

Simulação e Verificação: simular o funcionamento da ULA de forma a verificar o funcionamento de cada uma das suas operações. Verificar igualmente a geração das saídas ***zero, negative, carry*** *e* ***overflow***. Utilizar o ModelSim Altera para a simulação, desenvolvendo um *testbench* para acionamento das entradas. O testbench deve incluir a execução de ao menos um teste para cada operação da ULA. As operações aritméticas devem ser testadas para resultado zero, negativo, positivo e *overflow*.

Banco de Registradores

O banco de registradores a ser implementado deverá ter as seguintes características:

- O tamanho das palavras do banco de registradores será definido por uma constante DATA_WIDTH.
- O tamanho das palavras de endereçamento dos registradores será definido por uma constante ADDRESS_WIDTH.
- As constantes DATA_WIDTH e ADDRESS_WIDTH serão definidas na entidade do banco de registradores utilizando a primitiva generic.
- Por padrão, a constante DATA_WIDTH terá tamanho 32.
- O número de registradores será definido por $2^{\text{ADDRESS\_WIDTH}}$, onde ADDRESS_WIDTH.
- Por padrão, o banco de registradores será constituído por 32 registradores. Dessa forma, ADDRESS_WIDTH tem por padrão o valor 5, pois $2^5 = 32$.
- Cada um dos registradores terá um endereço, correspondendo a um índice de 0 até $2^{\text{ADDRESS\_WIDTH}}$.
- O registrador 0 (índice zero) é constante, não pode ser alterado. Qualquer leitura deste registrador retorna o valor zero e escritas não afetam o seu valor.
- O banco do registradores terá como entradas:
    - **clk** : entrada de tamanho 1 bit que receberá o sinal de relógio (clock).
    - **wren** : entrada de tamanho 1 bit responsável pela habilitação de escrita no registrador definido pelo endereço **wadd**. Dessa forma, quando **wren** estiver ativado (for igual a 1), o registrador endereçado por **wadd** é escrito com o valor presente no barramento **wdata** na transição de subida do relógio.
    - **wadd** : entrada de tamanho ADDRESS_WIDTH bits que define o endereço do registrador a ser escrito com o valor presente no barramento **wdata** caso **wren** esteja ativado na transição de subida do relógio.
    - **wdata** : entrada de tamanho DATA_WIDTH bits que contém o valor a ser escrito no registrador endereçado por **wadd** caso **wren** esteja ativado na transição de subida do relógio.
    - **radd1** : entrada de tamanho ADDRESS_WIDTH bits que define o endereço de um dos dois registradores a serem lidos. Na transição de subida de relógio, o conteúdo do registrador endereçado por **radd1** estará disponível na saída **rdata1**.
    - **radd2** : entrada de tamanho ADDRESS_WIDTH bits que define o endereço de um dos dois registradores a serem lidos. Na transição de subida de relógio, o conteúdo do registrador endereçado por **radd2** estará disponível na saída **rdata2**.
- O banco de registradores terá como saídas:
    - **rdata1** : saída de tamanho DATA_WIDTH bits que contém o valor lido do registrador endereçado por **radd1** na transição de subida do relógio.
    - **rdata2** : saída de tamanho DATA_WIDTH bits que contém o valor lido do registrador endereçado por **radd2** na transição de subida do relógio.

Os packages e a entidade do banco de registradores em VHDL será:

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

entity reg_bank is
	generic
	(
		DATA_WIDTH : natural := 32;
		ADDRESS_WIDTH : natural := 5
	);
	port (
		clk, wren		: in std_logic;
		radd1, radd2, wadd	: in std_logic_vector(ADDRESS_WIDTH-1 downto 0);
		wdata			: in std_logic_vector(DATA_WIDTH -1 downto 0);
		rdata1, rdata2		: out std_logic_vector(DATA_WIDTH -1 downto 0))
	);
end entity reg_bank;
```

Simulação e Verificação: simular o funcionamento do banco de registradores de forma a verificar a escrita e leitura de todos os registradores. Utilizar o ModelSim Altera para a simulação, desenvolvendo um *testbench* para acionamento das entradas.

O testbench deve incluir:

- escrever uma sequência de valores 1, 2, 3, 4 … 31 nos registradores do banco e, após, realizar a leitura de todos os registradores, em todas as combinações possíveis: leitura no registrador 1 (endereçado por radd1), leitura no registrador 2 (endereçado por radd2) e leitura simultânea em ambos (radd1 e radd2 apresentam o mesmo registrador)
- escrever no registrador zero e verificar se ele não é alterado
- escrita e leitura no mesmo ciclo, do mesmo registrador. O que acontece?

GRUPOS

O trabalho deverá ser realizado em grupos de 2 ou 3 alunos.

ENTREGA

Deverá ser entregue:

- **código VHDL comentado** para ULA e banco de registradores
- **testbench comentado** para ULA e banco de registradores
- **relatório mostrando os resultados da simulação (imagens)** tanto da ula como do banco de registradores. Nesse relatório você colocará as imagens das formas de onda obtidas em simulação, descrevendo os testes que estão sendo realizados em cada imagem.

Entregar um arquivo compactado em formato .zip no moodle (aprender.unb.br) até às 23h55 do dia 04/06/2017

O nome do arquivo deve conter as matrículas dos integrantes do grupo:
matriculaaluno1_matriculaaluno2_matriculaaluno3.zip

Será descontado 2 pontos por dia de atraso na entrega do trabalho.

CRITÉRIOS DE AVALIAÇÃO

Unidade Lógica e Aritmética – 40% da nota

I - Teste das operações da ULA – 20% da nota (0,2 pontos para cada operação que funciona corretamente)

II - Teste das saídas zero, negative, carry e overflow – 8% da nota (0,2 pontos para cada saída correta)

III – Testbench – 12% da nota (será verificado se o aluno testou todas as operações da ULA / saídas zero, negative, carry e overflow)

Banco de Registradores – 40% da nota

I – Teste de escrita/leitura de todos os registradores exceto registrador 0 – 20% da nota

II - Teste de escrita/leitura no registrador 0 – 5% da nota

III - Testbench: cobertura de testes - 15% da nota
- III.I - Teste de todos os registradores – 7% da nota
- III.II - Gerador de clock – 4% da nota
- III.III - Teste de escrita e leitura no mesmo ciclo – 4% da nota

Comentários nos códigos e relatório de simulação – 20% da nota

Esta especificação pode ser atualizada para se efetuar correções de texto ou alterações para se deixar mais claro o que está sendo pedido.

Caso a especificação sofrer uma atualização, os alunos serão informados via moodle (aprender.unb.br).

Última atualização: 18/05/2017 às 13:40

Dicas para implementação da ULA:

- Algumas operações necessitam de palavras com ou sem sinal. Para isso converta de std_logic_vector para **signed** ou **unsigned.** Exemplo: signed(input1)
- Algumas operações (ex.: +, -) não funcionarão com std_logic_vector. Para isso, será necessário converter para **signed** ou **unsigned**, realizar a operação, e depois reconverter pra std_logic_vector. Exemplo:

```
output <= std_logic_vector(signed(input1) <operação> signed(input2));
```

- Para as operações sll, srl e sra utilize as funções shift_left ou shift_right. **Será necessário ter cuidado com conversões de tipo!**