PCS3225 - Sistemas Digitais II
Trabalho 01 - Memórias em VHDL

Sergio Roberto de Mello Canovas

05/08/2019

O objetivo deste trabalho é exercitar a descrição de memórias em VHDL, que servirão como memórias para o projeto do processador.

Uma RAM e uma ROM.

Introdução

Em VHDL, pode-se criar um tipo para armazenamento de dados correspondente a um vetor cujo tipo do elemento modela cada palavra a ser armazenada. Veja o exemplo mem_tipo:

```
type mem_tipo is array(0 to 255) of bit_vector(7 downto 0);
```

O tipo declarado como mem_tipo corresponde a um vetor de 256 posições indexadas de 0 a 255, e o tipo de cada elemento é um vetor de bits bit_vector(7 **downto** 0), estabelecendo que cada palavra tem tamanho de 8 bits. Com relação a uma memória implementada com base nesse tipo, dizemos que ela tem profundidade de 256 palavras e largura de 8 bits. Uma instância deste tipo pode então ser declarada como um **signal**:

Em inglês, *depth*=256 e *width*=8.

```
signal mem: mem_tipo;
```

Atividades

T1A1 (5 pontos) Implemente um componente em VHDL correspondente a uma memória RAM com escrita síncrona que respeite a seguinte entidade:

Trabalho 1, Atividade 1

```
entity ram is
  generic (
    addressSize : natural := 64;
    wordSize    : natural := 32
  );
  port (
    ck, wr : in  bit;
    addr   : in  bit_vector(addressSize-1 downto 0);
    data_i : in  bit_vector(wordSize-1 downto 0);
    data_o : out bit_vector(wordSize-1 downto 0)
  );
end ram;
```

A escrita síncrona significa que o dado colocado em data_i deve ser escrito na memória na ocorrência de uma borda de subida do *clock* quando o sinal de escrita estiver ativo. Considere que wr é ativo alto. A leitura deve ocorrer de forma assíncrona, isto é, sem depender de uma borda de subida do *clock*. Basta alterar o valor da entrada addr e o sinal data_o será atualizado com o conteúdo armazenado na posição addr. Observe que o número de bits do barramento de endereço e o tamanho da palavra devem ser implementados por

`clock`=ck
escrita=wr

meio de *generics*, ou seja, sua memória poderá ser instanciada em um projeto com qualquer tamanho de barramento de endereço (não necessariamente 64 bits) e qualquer tamanho de palavra de dados (não necessariamente 32 bits). A lista completa dos sinais é a seguinte:

- ck: *Clock*;
- wr: Sinal de escrita. Quando estiver em ALTO e ocorrer uma borda de subida em ck, o conteúdo de data_i deve ser escrito na posição da memória determinada por addr;
- addr: Endereço;
- data_i: Conteúdo de entrada para escrever na memória com o uso do sinal wr;
- data_o: Conteúdo lido da memória. Deve corresponder sempre ao valor que está na palavra indexada por addr.

T1A2 (5 pontos) Implemente um componente em VHDL correspondente a uma memória ROM que respeite a seguinte entidade: Trabalho 1, Atividade 2

```
entity rom is
  generic (
    addressSize : natural := 64;
    wordSize    : natural := 32;
    mifFileName : string  := "rom.dat"
  );
  port (
    addr : in  bit_vector(addressSize-1 downto 0);
    data : out bit_vector(wordSize-1 downto 0)
  );
end rom;
```

Vimos em aula que existem tipos de ROM que, apesar do nome, também permitem a escrita de dados, embora seja uma operação mais complicada e menos frequente que a leitura. Porém, esta é uma ROM convencional que não permite escrita, e por isso ela só possui uma entrada e uma saída:

- addr: Endereço;
- data: Conteúdo armazenado correspondente ao endereço addr.

O conteúdo desta memória ROM deve ser carregado na inicialização a partir de um arquivo DAT. Isso pode ser feito em VHDL por meio de uma função de inicialização de memória. Pesquise como fazer isso e monte exemplos de arquivos DAT para que sua implementação funcione. Uma referência pode ser encontrada em `http://myfpgablog.blogspot.com/2011/12/memory-initialization-methods.html`. Veja a seção **VHDL with external data files**. O exemplo fornecido é capaz de ler arquivos DAT que, em essência, são arquivos-texto comuns em que cada linha contém o conteúdo de uma palavra na ordem dos endereços, em binário. Por exemplo, o conteúdo mostrado a seguir é o conteúdo de um arquivo DAT usado na inicialização de uma memória de profundidade 4 com largura de 8 bits:

```
00000000
00001111
11110000
11111111
```

Este arquivo indica o seguinte preenchimento na inicialização:

Endereço	Conteúdo (hex)
0	`00`
1	`0F`
2	`F0`
3	`FF`

Além do número de bits do barramento de endereço e do tamanho da palavra, o nome do arquivo DAT a ser carregado também deve ser um **generic**. Ou seja, quem elaborar um projeto que instancie sua memória pode determinar o nome do arquivo de inicialização que desejar. Nos seus testes, observe que o conteúdo do arquivo DAT que você criar deve ser condizente com a profundidade e largura da ROM que você instanciou.

Instruções para Entrega

Você deve acessar `https://pelicano.pcs.usp.br`, logar com o email cadastrado no Júpiter e enviar os seus dois arquivos, um de cada vez. Durante o envio do arquivo, será solicitada a *tag* do problema a ser resolvido, que está abaixo:

Note que os *tags* são diferentes.

Tarefa	Problema	*tag*
T1A1	RAM em VHDL	RAM19
T1A2	ROM em VHDL com carga	ROM19

O juiz corrigirá imediatamente sua submissão e retornará com a nota. Caso não esteja satisfeito, você pode enviar novamente e somente a última nota para aquele problema será válida. Neste trabalho, os problemas valem no máximo 10 pontos no juiz, porém a nota deste trabalho é composta pela média de ambas as notas do juiz (e.g. se o juiz te avaliar com 10 na submissão para a RAM e 5 na submissão para a ROM, ficará com 7,5 neste trabalho). Como sugestão, faça seu *testbench* e utilize um simulador de VHDL para validar sua solução antes de postá-la para o juiz.

A quantidade de submissões para estes problemas foi limitada a 5 por problema.