PCS3225 - Sistemas Digitais II
Trabalho 2 - Registradores em VHDL

Bruno de Carvalho Albertini

27/08/2019

O objetivo deste trabalho é exercitar a descrição de registradores em VHDL, que serão usados como componentes internos do PoliLEGv8.

O PC e o banco de registradores.

Introdução

O registrador é um dos componentes básicos do processador. Ele é o elemento de memória mais próximo das unidades funcionais e também o mais rápido, pois está diretamente ligado a elas através do fluxo de dados interno. Todas as operações importantes que acontecem no processador tem como origem ou destino um registrador, normalmente no **banco de registradores**.

No LEGv8 são 32 de 64 bits

Em sistemas digitais, um registrador nada mais é que um conjunto de *flip-flops*. Em VHDL, um *flip-flop* é descrito usando-se uma atribuição incompleta dentro de um bloco sequencial. Veja o exemplo abaixo de um bloco sequencial:

```
ffdr: process(clock, reset)
begin
    if reset='1' then
        q <= '0';
        q_n <= '1';
    elsif clock='1' and clock'event then
        if en = '1' then
            q <= d;
            q_n <= not d;
        end if;
    end if;
end process;
```

Neste bloco, descrevemos um *flip-flop* tipo D com *reset* assíncrono. Note que o processo tem em sua lista de sensibilidade o `clock` e o `reset`, mas o `if` divide as ações em assíncronas (se houver *reset*) ou síncronas (sensível à borda de subida do `clock`). Porém, se não houver um *reset* e também não houver uma borda de subida do *clock*, o comportamento não é especificado. Chamamos isso de **atribuição incompleta**, e a ação padrão do sintetizador será de manter os sinais inalterados.

O *reset* é prioritário pois é avaliado antes.

Para manter os sinais como estão, deve existir um elemento de memória. Nesse caso, o elemento de memória que forçamos o sintetizador a usar para manter o valor é exatamente um *flip-flop*. Caso desejássemos, poderíamos colocar um `else` contendo uma atribuição para o mesmo sinal, mas isto é desnecessário. Este tipo de atribuição incompleta também funciona para outros sinais. O sintetizador se encarregará de colocar o número de *flip-flops* adequado para manter o valor de um vetor ou sinal inalterado.

`q<=q;`

Atividades

T2A1 (5 pontos) Implemente um componente em VHDL correspondente a um registrador parametrizável, respeitando a seguinte entidade:

Trabalho 2, Atividade 1

```
entity reg is
generic(wordSize: natural :=4);
port(
  clock: in  bit; --! entrada de clock
  reset: in  bit; --! clear assincrono
  load:  in  bit; --! write enable (carga paralela)
  d:     in  bit_vector(wordSize-1 downto 0); --! entrada
  q:     out bit_vector(wordSize-1 downto 0) --! saida
);
end reg;
```

O registrador é parametrizável através do parâmetro `wordSize`, que determina o seu tamanho em bits. Recebe um sinal de *clock* periódico e é sensível a borda de subida deste sinal. O `reset` é assíncrono e força todos os bits para zero. Quando o sinal `load` á alto, a escrita é considerada habilitada e o valor na entrada `d` é gravado nos *flip-flops* quando ocorrer uma borda de subida do *clock*, refletindo então na saída `q`. Contrariamente, quando o sinal `load` é baixo, nada acontece e a saída permanece inalterada. A saída é assíncrona e mostra o conteúdo atual do registrador.

Tanto a entrada `d` quando a saída `q` são paralelas e contém `wordSize` bits.

T2A2 (5 pontos) Implemente um componente em VHDL correspondente a um bloco de registradores que respeite a seguinte entidade:

Trabalho 2, Atividade 2

```
entity regfile is
        generic(
                regn: natural := 32;
                wordSize: natural := 64
        );
        port(
                clock:    in  bit;
                reset:    in    bit;
                regWrite: in    bit;
                rr1, rr2, wr: in bit_vector(natural(ceil(log2(real(regn))))-1 downto 0);
                d:        in    bit_vector(wordSize-1 downto 0);
                q1, q2:   out   bit_vector(wordSize-1 downto 0)
        );
end regfile;
```

Os parâmetros determinam o número de registradores (`regn`) e o tamanho da palavra de cada registrador (`wordSize`). Por padrão, os registradores são numerados de `0` até `regn-1`. A linha de declaração das entradas `rr1,rr2,wr` faz exatamente isso, declarando estes sinais em função de `regn`, fazendo $\lceil log_2 regn \rceil - 1$. No caso de um banco com 32 registradores, estas entradas serão de 5 bits (`bit_vector(4 downto 0)`). Para fazer este cálculo dependente dos parâmetros, usamos a biblioteca matemática, então você obrigatoriamente deverá incluir as cláusulas de uso `use ieee.math_real.ceil;`

Se não usou a biblioteca `ieee`, terá que declará-la também.

e `use ieee.math_real.log2;` na sua descrição.

Assim como no caso dos registradores do PoliLEGv8, o último registrador não deve aceitar escritas (não há efeito algum em escrever nele) e sempre retorna zero quando lido.

No caso de 32 registradores, o de número 31 é o último.

É opcional, mas fortemente aconselhável, usar o registrador da atividade anterior como componente. O restante das caraterísticas do registrador permanecem as mesmas: todos os registradores do banco são sensíveis à borda de subida do *clock* e o *reset* é assíncrono.

O *reset* vale para todos os registradores do banco.

A entrada de dados `d` possui largura de `wordSize`, mas na borda de subida do *clock* somente o registrador apontado por `wr` será escrito, se o `regWrite` for alto. Exemplo: para 32 registradores, se `wr=01010` o registrador 10 (ou o décimo primeiro) amostrará a entrada `d` para seus *flip-flops*. Caso o `regWrite` seja baixo na borda de subida do *clock*, a escrita não acontece.

A leitura é assíncrona e o banco possui duas saídas de leitura. O registrador apontado pela entrada `rr1` coloca o seu valor na saída `q1` e o apontado pela entrada `rr1` coloca na saída `q2`. Exemplo: para 32 registradores, se `rr1=00011` registrador 3 (ou o quarto) coloca o seu valor armazenado na saída `q1`.

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*
T2A1	Registrador	REG19
T2A2	Banco	BREGS

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 5 pontos no juiz e a nota para este trabalho é composta pela soma das notas dadas pelo juiz. 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 10 por problema.