

Declarações paralelas, concorrentes e sequenciais

Disciplina:
Laboratório de Sistemas Digitais

Professor:
Jhonattan Córdoba Ramírez

Alunos:
João Pedro Copelli - 2021014414
João Victor Gomes - 2020072690
Marcelle Christine Aquino Silva - 2021014546

7 de outubro de 2022

Sumário

1	Introdução	3
2	Parte Teórica	3
2.1	Qual a diferença entre concorrência e paralelismo?	3
2.2	Por que as construções concorrentes em VHDL não são chamadas de paralelas?	3
2.3	Quando as construções concorrentes em VHDL podem apresentar comportamento paralelo?	4
2.4	Qual a diferença entre se atribuir o valor a m sinal dentro de um Process ou fora dele?	4
2.5	Qual a função da lista de sensibilidades de um processo?	4
2.6	Quais as regras para se definir o que deve ser incluído na lista de sensibilidade de um Processo?	5
3	Parte Prática	5
4	Conclusão	9

1 Introdução

Neste relatório serão discutidos conceitos de Declarações concorrentes,Declaração Process em VHDL,Lista de sensibilidade, Declarações sequenciais em VHDL.

2 Parte Teórica

2.1 Qual a diferença entre concorrência e paralelismo?

Concorrência é sobre a execução sequencial e disputada de um conjunto de tarefas independentes. Sob o ponto de vista de um sistema operacional, o responsável por esse gerenciamento é o escalonador de processos. Já sob o ponto de vista de concorrência em uma linguagem de programação, o responsável é o scheduler interno da linguagem. Escalonadores preemptivos (como é o caso dos sistemas operacionais modernos) favorecem a concorrência pausando e resumindo tarefas (no caso de sistemas operacionais estamos falando de processos e threads no que chamamos de trocas de contexto) para que todas tenham a oportunidade de serem executadas.

Paralelismo é sobre a execução paralela de tarefas, ou seja, mais de uma por vez (de forma simultânea), a depender da quantidade de núcleos (cores) do processador. Quanto mais núcleos, mais tarefas paralelas podem ser executadas. É uma forma de distribuir processamento em mais de um núcleo.

Também podemos dizer paralelismo é uma forma de atingir concorrência e que cada linha de execução paralela também é concorrente, pois os núcleos estarão sendo disputados por várias outras linhas de execução e quem gerencia o que o núcleo vai executar em dado momento do tempo é o escalonador de processos. Paralelismo implica concorrência, mas o contrário não é verdadeiro, pois é possível ter concorrência sem paralelismo, é só pensar no caso de uso de uma única thread gerenciando milhares de tarefas, pausando e resumindo-as, esse é um modelo de concorrência sem paralelismo.

2.2 Por que as construções concorrentes em VHDL não são chamadas de paralelas?

São conceitos diferentes: construções paralelas são utilizadas para serem executadas de formas simultâneas, enquanto construções concorrente é sobre uma execução sequencial, é possível ter uma construção concorrente que não seja paralela.

2.3 Quando as construções concorrentes em VHDL podem apresentar comportamento paralelo?

Dentro de um process a avaliação dos comandos é sequencial mas fora dele tudo é avaliado em paralelo, a simulação deve refletir o comportamento paralelo do hardware onde vários processos trabalham simultaneamente.

2.4 Qual a diferença entre se atribuir o valor a m sinal dentro de um Process ou fora dele?

Uma atribuição de sinal pode aparecer dentro de um PROCESS ou fora dele. Se a atribuição estiver fora do PROCESS, ela é considerada uma sentença de atribuição “concorrente” (a menos do tempo de propagação da (cadeia de eventos) com as demais sentenças do projeto. Quando uma sentença de atribuição aparece dentro de um PROCESS, a atribuição é considerada sequencial e é executada em sequência com as demais atribuições presentes no mesmo bloco PROCESS. Neste caso, quando a sentença de atribuição é executada, seu valor é calculado naquele instante, mas o resultado não é “carregado” imediatamente no sinal. Você pode imaginar que ele é “agendado” para ser atribuído ao sinal após o atraso de tempo ΔT . Portanto, normalmente não podemos contar com este resultado na linha de programação seguinte. Se for necessário usar o “conteúdo” atribuído nas linhas seguintes dentro do PROCESS, o mais adequado é usar uma VARIABLE, que é de caráter temporário e é atualizada instantaneamente.

2.5 Qual a função da lista de sensibilidades de um processo?

uma lista de sensibilidade indica quais são as variáveis e sinais cuja alteração deve levar à reavaliação da saída. No simulador funcional, quando uma variável da lista é modificada, o processo é simulado novamente. Um process é ativado sempre que um dos sinais na sua lista de sensibilidade é atualizado e Quando isto ocorre o circuito executa o algoritmo descrito e volta a ficar inativo. Caso nenhum destes varie, os sinais dentro do processo mantém-se inalterados. Ou seja, o valor anterior é memorizado.

2.6 Quais as regras para se definir o que deve ser incluído na lista de sensibilidade de um Processo?

A lista de sensibilidade de um process sequencial deve possuir somente o clock em caso de circuitos com reset síncrono ou o clock e o reset em caso de circuitos com reset assíncrono. No primeiro caso o reset só é amostrado na borda do clock logo podemos omitir ele da lista de sensibilidade. No caso de circuitos combinacionais, é de extrema importância adicionar todas as entradas do process na lista de sensibilidade. Uma lista de sensibilidade incompleta pode gerar uma inconsistência entre a simulação e a síntese já que a síntese observa o comportamento do circuito e gera o hardware a partir disso, muitas vezes ignorando a lista de sensibilidade.

3 Parte Prática

Criamos um novo projeto no Quartus II e adicionamos o arquivo *mean_4_clocks.vhd* ao projeto e realizamos a compilação.

Durante a análise das atribuições das variáveis, percebemos que a ordem estava invertida, e que, levando em consideração o funcionamento de variáveis em VHDL, todas iriam receber o mesmo valor lido, sem manter o histórico das últimas medições. Outro erro encontrado foi na precisão das medidas e do cálculo da média móvel. Para salvarmos um dado, era feita a leitura da entrada e dividia-se esse valor por 4. Ao final do processo, soma-se as quatro medidas. Porém, ao fazer desta maneira, perdímos precisão. Uma maneira melhor seria fazer a leitura e dividir o valor somente por 2 e, ao final do processo, somar as quatro medidas e novamente dividir por 2.

Com tudo verificado e o código alterado conforme a descrição acima, fizemos a configuração do *testbench* e executamos a simulação pelo ModelSim. Com a simulação também funcional, podemos fazer o planejamento de quais pinos serão utilizados na placa real. Definimos que os pinos utilizados serão:

- Entradas INPUT:
 - Chave SW0 - PIN_C10
 - Chave SW1 - PIN_C11
 - Chave SW2 - PIN_D12
 - Chave SW3 - PIN_C12
- Entradas RESET:
 - Chave SW8 - PIN_B14

- Entradas CLOCK:
 - Chave SW9 - PIN_F15
- Saída OUTPUT:
 - LED0 - PIN_A8
 - LED1 - PIN_A9
 - LED2 - PIN_A10
 - LED3 - PIN_B10

Com isso, realizamos a gravação da placa para os testes.

Com a placa gravada, realizamos o seguinte teste: Mantivemos a entrada INPUT como "0100" e acionamos o CLOCK 4 vezes. Durante o acionamento do CLOCK, observamos o comportamento do OUTPUT. A cada pulso no CLOCK, o OUTPUT deveria ser incrementado em 1. Ao final do processo, acionamos o RESET para recomeçarmos a contagem.

Os resultados obtidos foram esses:



Figura 1: INPUT igual a '0000', sinal de CLOCK igual a '0', OUTPUT igual a '000'



Figura 2: INPUT igual a '0100', sinal de CLOCK igual a '1', OUTPUT igual a '001'



Figura 3: INPUT igual a '0100', sinal de CLOCK igual a '1', OUTPUT igual a '010'



Figura 4: INPUT igual a '0100', sinal de CLOCK igual a '1', OUTPUT igual a '011'



Figura 5: INPUT igual a '0100', sinal de CLOCK igual a '1', OUTPUT igual a '100'



Figura 6: INPUT igual a '0100', sinal de CLOCK igual a '1', RESET igual a '1', OUTPUT igual a '000'

4 Conclusão

Como podemos observar pelas figuras acimas e corrigindo os erros encontrados, os resultados obtidos foram iguais aos resultados esperados.