20 de Junho de 2018

ISL – PRÁTICA 04: MAIOR DIVISOR COMUM EM VERILOG

NOMES:

Isabela Marina Ferreira Pires
Isadora Alves de Salles
Joao Marcos Machado Couto
Thiago Martin Poppe

OBJETIVO:

Implementar, utilizando um ambiente de desenvolvimento da linguagem Verilog, um algoritmo que, dados dois números inteiros de 16 bits, retorne o maior divisor comum entre eles.

COMPILAÇÃO/EXECUÇÃO

O código foi escrito em uma máquina Linux, distribuição Ubuntu. Foi criado um arquivo mdc.v (contendo o código em Verilog para o problema), um arquivo mdc.vvp e um arquivo mdc.vcd que foi usado para realizar a plotagem das "formas de onda" usando o software GTKWave Analyzer v3.3.66. Para compilar, usamos o comando "iverilog mdc.v -o mdc.vvp -Wall" (como solicitado nos slides para gerar o arquivo .vvp), "vvp mdc.vvp mdc.vcd" (como solicitado nos slides para gerar o arquivo .vcd) e finalmente "gtkwave mdc.vcd" (como solicitado nos slides para executarmos o .vcd no GTKWave Analyzer).

METODOLOGIA/LÓGICA:

A fim de possibilitar o calculo do mdc, resolvemos implementar o algoritmo euclidiano, no qual, dado dois números a e b, com a = bq+r| 0<=r <b, o mdc(a,b) = mdc(b,r). Isto é, o mdc de dois números é igual ao mdc entre o maior deles e o resto da divisão do maior pelo menor.

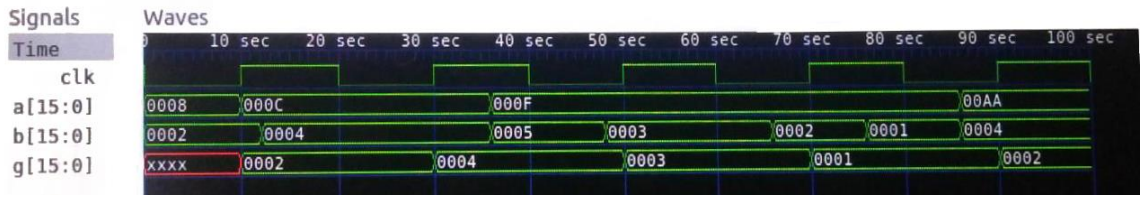
No código incluso no mesmo zip desta documentação, os valores de 'a' e 'b' serão nossos inputs (juntamente com um sinal de clock) que serão do tipo wire (fios), e o valor 'g' será o nosso output (também sendo do tipo wire), onde g = mdc(a, b).

Criamos dois módulos principais, um para o mdc propriamente dito e outro para os testes dele no gerador de ondas acoplado, GTKWave. Foi utilizado um clock para que o nosso bloco de instruções para o cálculo do mdc seja funcional apenas durante a borda de subida do mesmo, o que garante assim, maior controle sobre o mesmo (evitando possíveis hazards, etc).

Para a computação de tal valor, usamos 3 variáveis do tipo reg (registradores na linguagem Verilog) de 16 bits. Uma variável do tipo reg para armazenar o resto da divisão de 'a' por 'b', outro para armazenar o valor de 'a' ao longo dos cálculos e outro para armazenar o valor de 'b' ao longo dos cálculos. Foi utilizado um bloco always @(posedge clk), onde clk seria o nosso clock. Dentro desse bloco, usamos um while que ficará rodando até que o valor do resto seja igual a 0 (condição de parada do algoritmo euclidiano). Após o término do while, teremos então o valor do mdc(a, b) que será o nosso output através do fio 'g'.

O bloco always @(posedge clk) será executado sempre que o nosso clock estiver em uma borda de subida, ou seja, apenas calculamos o mdc(a, b) quando o clock estiver em uma borda de subida.

Para maior entendido durante a leitura do código, adicionamos uma serie de comentários contendo uma versão breve das informações contidas aqui. Temos aqui as formas de ondas geradas durante os testes.



DIFICULDADES:

Uma dificuldade está relacionada à própria linguagem e suas variáveis. Um exemplo disso, está na variável do tipo reg e do tipo wire. Certas operações podem ser feitas sobre elas sobre certas circunstâncias apenas... Por exemplo, o tipo wire deve ter o bloco assign antes de sofrer alguma atribuição. Coisas desse tipo, dificultaram um pouco a lógica ao longo do desenvolvimento do código em Verilog, mas no final, com muitas pesquisas e dúvidas esclarecidas, foi possível entregar o que foi solicitado pela prática.

Porem a maior dificuldade foi a de trabalhar com a comunicação entre dois módulos citados acima, visto que, em uma linguagem de descrição de hardware, aparentemente muitas coisas acontecem, em algum sentido, ao mesmo tempo. Assim nos tivemos que investir uma quantidade significante de tempo na adaptação a essa lógica na qual as instruções não são tão sequenciais como estamos acostumados.Ademais, não conseguimos fazer com que o WtkWave impressimise em sua interface de visualização os valores em binário ou decimal, mantendo então a configuração default de hexadecimal