Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Engenharia
Programa de Graduação em Engenharia da Computação

TRABALHO 1 – VALIDAÇÃO: SIMULAÇÃO FUNCIONAL

Gabriel Chieza Chiele e Maiki Buffet

Professora: Letícia Pöhls

Porto Alegre, 03 de outubro de 2017

Sumário

1. Introdução

Este trabalho tem como foco a aprendizagem do processo de projeto de hardware, utilizando as ferramentas *Cadence*, com a linguagem *VHDL*, nas etapas de validação e simulação funcional, tanto como a sua automatização com scripts *.do* e programação em *C*.

2. Validação

Etapa de compilação do *VHDL* e execução de *testbenches*.

Foi criado o script *compile_rtl.do* onde compilamos o *design*, simulamos e geramos arquivos como o *pif2wb.vcd*, que contém dados de operação dos sinais, além da geração de métricas de cobertura de código.

2.1. Code Coverage

As métricas abordadas neste trabalho – melhorar as métricas de *code coverage* – são:

- *Branch*: *branch coverage* analisa a execução de segmentos independentes de código, ou seja, pedaços de código que estão separados do fluxo de execução normal por uma instrução condicional (“if”);
- *Statement*: *statement coverage* analisa se todas as instruções presentes no código são executadas.

Para melhorar as métricas mencionadas, fora adicionado uma nova escrita de dados ao *testbench*:

```
-- ------------------------------------ Single Write at Address 0x80000000
-- -------------  Current Time:  1575ns
WAIT FOR 40 ns;
PIReqVALID <= '1';
PIReqCNTL <= "10000101";
PIReqDATA <= "00000000000000001111111111111111";
PIReqDataBE <= "0111";
-- ------------------------------------ End of Single Write
-- -------------  Current Time:  1595ns
WAIT FOR 20 ns;
PIReqVALID <= '0';
PIReqCNTL <= "11111111";
PIReqDataBE <= "0000";
```

Figura 1: trecho de código adicionado ao *testbench*.

Os bits 1 e 2 do dado *PIReqCNTL* são utilizados para realizar a atribuição do sinal *TOT_TRANSFER_I*. O *testbench* não realizava uma instrução onde estes bits continham a combinação “10”, apenas as outras combinações eram testadas.

```
case PIReqCNTL(2 downto 1) is
        when "00"   =>
            TOT_TRANSFER_I      <= 1;
        when "01"   =>
            TOT_TRANSFER_I      <= 3;
        when "10"   =>
            TOT_TRANSFER_I      <= 7;
        when "11"   =>
            TOT_TRANSFER_I      <= 15;
        when others =>
            TOT_TRANSFER_I      <= 0;
    end case;
```

Figura 2: *branch* e *statement* que foram adicionados ao *coverage*.

A adição desta escrita de dados ao *testbench* melhorou a métrica *statement* em 0.8% e a métrica *branch* em 1.8%.

```
================================================================================
=== File: pif2wb.vhd
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                          122        33        89      27.0
    Branches                        55         9        46      16.3
    FEC Condition Terms             21         0        21       0.0
    FEC Expression Terms             0         0         0     100.0
    Toggle Bins                     37         4        33      10.8
```

Figura 3: *coverage* antes das modificações.

```
================================================================================
=== File: pif2wb.vhd
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                          122        34        88      27.8
    Branches                        55        10        45      18.1
    FEC Condition Terms             21         0        21       0.0
    FEC Expression Terms             0         0         0     100.0
    Toggle Bins                     37         4        33      10.8
```

Figura 4: *coverage* após as modificações.

2.2. Parser

Criamos um *parser* de arquivos *.vcd* em *C++*. O *parser* deve nos informar os sinais que compõem cada módulo funcional, além do consumo de *Dynamic Power* do *design*. A saída deve conter de forma estruturada o nome do bloco funcional, seguido de seus sinais, além dos valores de *Dynamic Power* por bloco funcional e total, fora o sinal com maior atividade de *toggling*.

O cálculo do *Dynamic Power* utilizado para estudo, segundo a literatura, foi:

$$DynamicPower = \frac{Vdd^2 \times fClock \times Cl \times Esw}{2}$$

Sendo:

- Tensão de alimentação: Vdd = 1 V;
- Frequência do *clock*: fClock = 50 MHz;
- Carga de saída da capacitância: Cl = 1 mF;
- Fator da atividade de trocas dos sinais: Esw*.

* Utilizando-se o sinal com a maior troca de transições como fator 1 (100%), calculou-se o fator de cada sinal do circuito, e através do somatório desses fatores dos sinais – excluindo-se o sinal de fator 1 – dividiu-se o somatório pela quantidade total de sinais presentes no circuito.

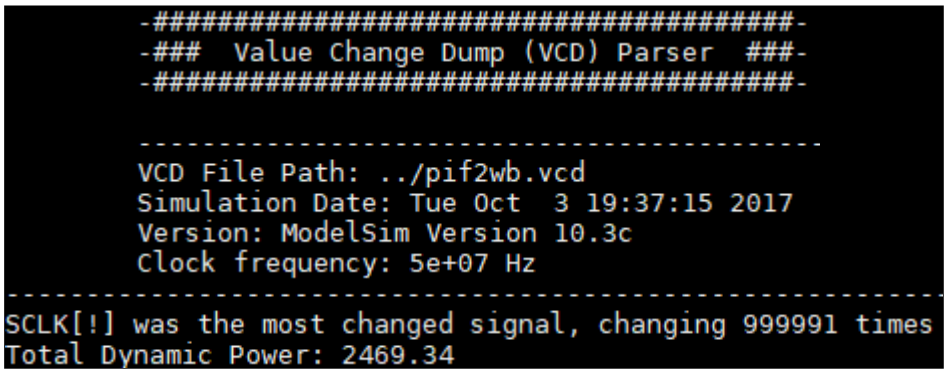

Figura 5: saída* do *parser* após análise do *pif2wb.vcd*.

* Na imagem não consta a saída dos sinais e a quantidade de troca de cada sinal.

3. Referências

http://vlsi.pro/code-coverage-fundamentals/

https://www.microsemi.com/document-portal/doc_view/131619-modelsim-user

http://www.siliconintelligence.com/people/binu/perception/node13.html