Aluna: Jaqueline Ferreira de Brito
Data: 08 de Fevereiro de 2025

RELATÓRIO VERIFICAÇÃO FORMAL E FUNCIONAL
Gravação de transações, Randomização e Usando uvm_test

Descrição: Este relatório apresenta uma variação da primeira simulação UVM explorada em aula. A simulação inicial continha apenas os componentes básicos: fonte (source), transação (transaction) e sorvedouro (sink). Nesta versão, esses componentes foram integrados à classe test, proporcionando maior modularidade e flexibilidade na execução dos testes.
Instruções:

- Implementar uma variação da primeira simulação apresentada em aula, incorporando a classe test.
- A variação pode ser baseada em um dos exemplos discutidos em aula ou ser uma abordagem nova, conforme a escolha do aluno.
- O relatório deve ser conciso (2 a 5 páginas) e entregue em formato PDF.

1. Introdução

Este relatório apresenta uma evolução da primeira simulação UVM discutida em aula. Inicialmente, a verificação envolvia apenas os componentes básicos: fonte (**source**), transação (**transaction**) e sorvedouro (**sink**). Nesta versão, esses elementos foram integrados à classe **test**, proporcionando maior modularidade e flexibilidade na execução dos testes.

2. Ambiente de Teste UVM

2.1 Gravação de Transações

A transação é definida pela classe Jaque_a_tr, que inclui uma variável randômica j sujeita a duas restrições: ser positiva e menor que 15. Para permitir o registro de transações, utiliza-se uvm_object_utils_begin, enquanto uvm_field_int(j, UVM_ALL_ON | UVM_DEC) garante a visualização dos valores em decimal.

```
class Jaque_a_tr extends uvm_sequence_item;

        rand int j;

        constraint a_positive { j > 0; }
        constraint a_small { j < 15; }

        `uvm_object_utils_begin(Jaque_a_tr) // Necessário para gravar a transação
        `uvm_field_int(j, UVM_ALL_ON | UVM_DEC)
        `uvm_object_utils_end

endclass
```

2.2 Adicionando Transações na fonte:

```
class source extends uvm_component;
   `uvm_component_utils (source)

   // creating a put_port which will accept a "Jaque_a_tr" type of data
   uvm_blocking_put_port #(Jaque_a_tr) out;

   function new (string name = "Jaqueline_Brito", uvm_component parent);
        super.new (name, parent);
        out = new ("Saída", this);
   endfunction

   task run_phase (uvm_phase phase);
        Jaque_a_tr  tr;

        phase.raise_objection(this); // avoid simulation to finish immediately

        // Trocando as 10 transações do professor por 6 transações que vão ser enviadas
via put_port
        repeat (6) begin
        #12;                        // Espera 12ns
        tr = new();                 // create transaction instance
        `bvm_begin_tr(tr)           // start transaction recording
```

```
        `uvm_info ("SOURCE", "Sending transaction by Jaqueline_Brito", UVM_LOW)
        out.put (tr);                // send transaction to put_port
        end
        #100;                        // allow for some downstream processing time
        phase.drop_objection(this);   // allow simulation to finish
    endtask
endclass
```

A classe source gera e envia transações do tipo Jaque_a_tr através de um put_port. Source herda de uvm_component e é registrada com a macro uvm_component_utils(source). A Criação do put_port: O uvm_blocking_put_port #(Jaque_a_tr) out; permite enviar transações do tipo Jaque_a_tr para outro componente. Construtor new: Inicializa o componente e cria o put_port com o nome "Saída". Execução da run_phase: Levanta objeção (phase.raise_objection(this)) para evitar que a simulação termine imediatamente. Gera e envia 6 transações, com intervalo de 12ns entre cada uma. Registra o início da transação com a macro bvm_begin_tr(tr). Envia a transação pelo put_port e imprime uma mensagem de depuração (uvm_info). Após um atraso de 100ns, a objeção é retirada (phase.drop_objection(this)), permitindo que a simulação finalize.

2.3 Finalizando a gravação de transações no sink:

A classe sink recebe as transações enviadas pelo source e finaliza o registro das mesmas, aguardando 6ns para simular o tempo de processamento:

```
class sink extends uvm_component;
        `uvm_component_utils(sink)

        uvm_blocking_put_imp #(Jaque_a_tr, sink) in;

        function new(string name = "sink", uvm_component parent);
        super.new(name, parent);
        in = new("Entrada", this);
        endfunction

        task put(Jaque_a_tr tr);
        // Aqui  nós recebemos as transações de source
        `uvm_info("SINK", "Receiving transaction by Jaqueline_Brito", UVM_LOW)
        #6; // Espera a transação por 6ns para ser completa
        `bvm_end_tr(tr)
        endtask
endclass
```

Sink herda de uvm_component e é registrada com a macro uvm_component_utils(sink). Criação da porta de entrada (put_imp): uvm_blocking_put_imp #(Jaque_a_tr, sink) in; permite que o componente receba transações do tipo Jaque_a_tr enviadas pelo source. Construtor new: Inicializa o componente e a porta de entrada in, nomeada como "Entrada". Método put recebe a transação do source e imprime uma mensagem de depuração (uvm_info). Aguarda 6ns para simular o tempo de processamento da transação. Registra o fim da transação com a macro bvm_end_tr(tr), garantindo que a gravação da transação seja completada.

2.4 A classe test contém todas as funcionalidades que estavam no módulo top na atividade anterior.

O código define a classe test, que representa um teste UVM responsável por conectar os componentes source e sink, permitindo a simulação do fluxo de transações:

```
class test extends uvm_test;
   `uvm_component_utils(test)

   source source_h;
   sink sink_h;

   function new(string name, uvm_component parent);
       super.new(name, parent);
   endfunction

   function void build_phase(uvm_phase phase);
       super.build_phase(phase);
       source_h = new("source_h", this);
       sink_h = new("sink_h", this);
   endfunction

   function void connect_phase(uvm_phase phase);
       source_h.out.connect (sink_h.in);
   endfunction

endclass
```

Test herda de uvm_test e é registrada com a macro uvm_component_utils(test). Declara duas instâncias: source_h → Gera e envia transações. Sink_h → Recebe e processa as transações. Construtor new: Inicializa a classe de teste, chamando o construtor da classe pai (super.new). Fase build_phase (Construção dos Componentes): Cria as instâncias source_h e sink_h, associando-as ao testbench. Fase connect_phase (Conexão dos Componentes): Conecta a porta de saída do source (source_h.out) à porta de entrada do sink (sink_h.in), garantindo que as transações fluam corretamente entre os dois componentes.

Fluxo do Teste:

1. O source_h gera e envia transações através do put_port.
2. O sink_h recebe essas transações pelo put_imp, processa e finaliza o registro.
3. A conexão entre os dois componentes é feita dentro da connect_phase.
4. A simulação pode ser iniciada com run_test("test") no ambiente UVM.

2.5 Definição do Módulo top

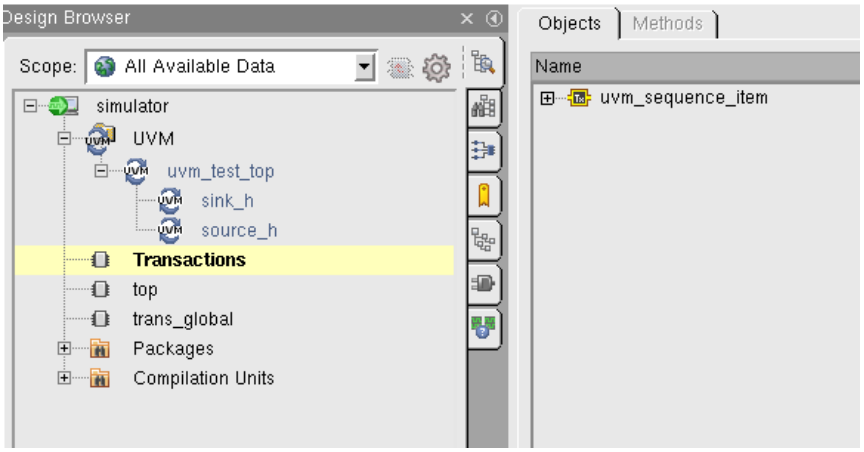
```
`include "uvm_macros.svh"
import uvm_pkg::*;
`include "bvm_macros.svh" //macros criado no IP do Brasil polo UFCG

`include "trans.svh"
`include "source.svh"
`include "sink.svh"
`include "test.svh"

module top;

        initial begin
        run_test("test");
        end
endmodule
```

Inclusão de Headers (.svh):

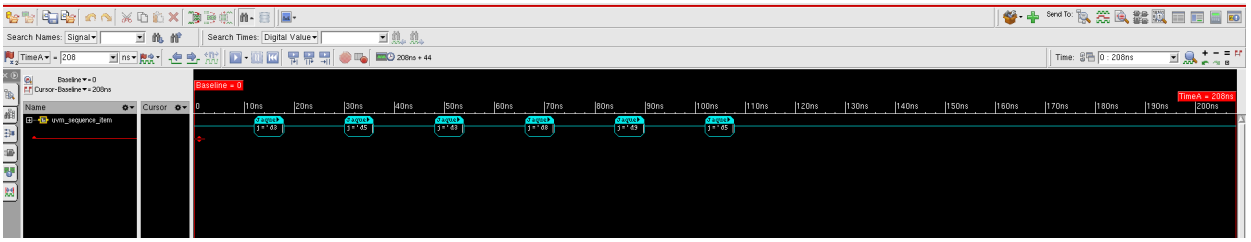
1. uvm_macros.svh → Contém macros UVM essenciais para verificação.
2. bvm_macros.svh → Macros específicas do IP do Brasil (UFCG), incluindo bvm_begin_tr e bvm_end_tr para gravação de transações.
3. trans.svh → Define a classe de transação Jaque_a_tr.
4. source.svh → Implementar o gerador de transações (source).
5. sink.svh → Implementa o receptor de transações (sink).
6. test.svh → Define o teste (test), que conecta source e sink.

3. Simulação

- run_test("test") → Inicia a simulação chamando a classe test.
- test.svh cria e conecta os componentes:
    - source → Gera e envia transações a cada 12ns.
    - sink → Recebe e processa cada transação em 6ns.
- As transações são registradas com as macros bvm_begin_tr(tr) e bvm_end_tr(tr).
- Após o processamento das transações, a simulação termina.



```
uvm: *E,UVMACC: UVM commands require read/write access for the verilog functions which implement the commands
uvm: *E,UVMACC: UVM commands require read/write access for the verilog functions which implement the commands
stop -create -name Randomize -randomize
Created stop Randomize

SDI/Verilog Transaction Recording Facility Version 24.09-s001
UVM_INFO source.svh(23) @ 12: uvm_test_top.source_h [SOURCE] Sending transaction by
Jaqueline_Brito

UVM_INFO sink.svh(13) @ 12: uvm_test_top.sink_h [SINK] Receiving transaction by Jaqueline_Brito
UVM_INFO source.svh(23) @ 30: uvm_test_top.source_h [SOURCE] Sending transaction by
Jaqueline_Brito

UVM_INFO sink.svh(13) @ 30: uvm_test_top.sink_h [SINK] Receiving transaction by Jaqueline_Brito
UVM_INFO source.svh(23) @ 48: uvm_test_top.source_h [SOURCE] Sending transaction by
Jaqueline_Brito

UVM_INFO sink.svh(13) @ 48: uvm_test_top.sink_h [SINK] Receiving transaction by Jaqueline_Brito
UVM_INFO source.svh(23) @ 66: uvm_test_top.source_h [SOURCE] Sending transaction by
Jaqueline_Brito

UVM_INFO sink.svh(13) @ 66: uvm_test_top.sink_h [SINK] Receiving transaction by Jaqueline_Brito
UVM_INFO source.svh(23) @ 84: uvm_test_top.source_h [SOURCE] Sending transaction by
Jaqueline_Brito

UVM_INFO sink.svh(13) @ 84: uvm_test_top.sink_h [SINK] Receiving transaction by Jaqueline_Brito
UVM_INFO source.svh(23) @ 102: uvm_test_top.source_h [SOURCE] Sending transaction by
Jaqueline_Brito

UVM_INFO sink.svh(13) @ 102: uvm_test_top.sink_h [SINK] Receiving transaction by Jaqueline_Brito
UVM_INFO
/usr/local/cds/XCELIUM2409/tools/methodology/UVM/CDNS-1.1d/sv/src/base/uvm_objection.svh(1268)
@ 208: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase


--- UVM Report catcher Summary ---


Number of demoted UVM_FATAL reports  :    0
Number of demoted UVM_ERROR reports  :    0
Number of demoted UVM_WARNING reports:    0
Number of caught UVM_FATAL reports   :    0
Number of caught UVM_ERROR reports   :    0
Number of caught UVM_WARNING reports :    0

--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :   15
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[RNTST]     1
[SINK]     6
[SOURCE]     6
[TEST_DONE]     1
[UVM_CMDLINE_PROC]     1
Simulation complete via $finish(1) at time 208 NS + 44
/usr/local/cds/XCELIUM2409/tools/methodology/UVM/CDNS-1.1d/sv/src/base/uvm_root.svh:457     $finish;
xcelium> run
```



4. Conclusão

A integração da classe test aumentou a modularidade e organização do ambiente UVM, garantindo melhor separação das responsabilidades dos componentes. O uso de macros UVM para registro e randomização melhorou a visualização das transações, proporcionando um fluxo mais estruturado e eficiente. Essa abordagem possibilita futuras expansões e otimizações na verificação funcional e formal.