Aluna: Jaqueline Ferreira de Brito
Data: 02 de Março de 2025

RELATÓRIO VERIFICAÇÃO FORMAL E FUNCIONAL
ANALYSIS FIFO, COVERAGE E AGENT

Este relatório apresenta a continuação de uma série de atividades abrangendo a evolução da primeira simulação UVM explorada em aula. A simulação inicial era composta apenas pelos componentes fundamentais: fonte (source), transação (transaction) e sorvedouro (sink). Nesta versão, esses componentes foram reorganizados e integrados à classe test, tornando o ambiente de teste mais modular e flexível. Além disso, foram implementadas variações para incorporar conceitos como ambientes de teste (env), pacotes (package), filas FIFO (fifo), modelos de referência (reference model), e mais recentemente, a evolução para analysis source, cobertura e agentes, aprimorando a estrutura e a eficiência da simulação.

1. INTRODUÇÃO

Este documento analisa uma evolução significativa na arquitetura de um testbench UVM: a transição de uma implementação tradicional baseada em source com uvm_blocking_put_port para uma abordagem mais flexível usando analysis_source com uvm_analysis_port. Esta mudança não apenas aprimora a capacidade do testbench para suportar múltiplas conexões simultâneas, mas também estabelece a base para funcionalidades avançadas como cobertura funcional e implementação de agentes UVM completos.

2. DESENVOLVIMENTO

Diferenças entre Source e Analysis Source na Metodologia UVM:

a. Limitações da Source Tradicional

A implementação tradicional usando uvm_blocking_put_port apresenta duas limitações críticas:

1. **Conexão única**: Permite apenas uma conexão downstream, restringindo o envio de transações a um único componente consumidor
2. **Comunicação bloqueante**: O método put(tr) força a fonte a esperar o processamento da transação pelo consumidor.

```
// Implementação tradicional com limitações
uvm_blocking_put_port #(j_jb) out;  // Permite apenas uma conexão
// A transação é enviada de forma bloqueante com:
out.put(tr);  // Método bloqueante
```

Esta abordagem cria gargalos no fluxo de verificação, reduzindo a eficiência do testbench.

b. A Solução com Analysis Port

A implementação de analysis_source com uvm_analysis_port supera estas limitações

```
// Nova implementação com maior flexibilidade
uvm_analysis_port #(j_jb) out; // Permite múltiplas conexões
// A transação é distribuída de forma não bloqueante com:
out.write(tr);// Método não bloqueante
```

Vantagens Principais:

A uvm_analysis_port oferece três benefícios fundamentais:

1. **Múltiplas conexões**: Uma única transação pode ser distribuída para diversos componentes simultaneamente
2. **Operação não-bloqueante**: O método write() não espera pelo processamento do receptor
3. **Desacoplamento**: Reduz dependências entre componentes, facilitando a implementação de monitores de cobertura e outros observadores

Esta evolução na arquitetura do testbench aumenta significativamente sua flexibilidade, eficiência e capacidade de expansão para verificações mais complexas.

c. Cobertura

Com a análise port, agora podemos facilmente adicionar cobertura ao testbench:A classe coverage_in herda de bvm_cover, que é uma extensão de uvm_subscriber. Esta hierarquia permite que o componente:

1. Receba transações através de uma porta de análise
2. Monitore a cobertura automaticamente
3. Sinalize quando 100% da cobertura for atingida, permitindo terminar a simulação

d. Agente UVM

O agente é um componente fundamental na metodologia UVM, que encapsula:Vantagens desta implementação:

1. Encapsulamento da fonte dentro do agente
2. Exposição de uma única porta de análise para o ambiente
3. Melhor organização hierárquica (seguindo o padrão UVM)

e. Conexões no Ambiente

A nova implementação permite conexões mais flexíveis e funcionais no ambiente:

```
function void connect_phase(uvm_phase phase);
  agent_Jaqueline.out.connect(coverage_in_Jaqueline.analysis_export);
  agent_Jaqueline.out.connect(agent_refmod.analysis_export);
  refmod_h.in.connect(agent_refmod.get_export);
  refmod_h.out.connect(sink_h.in);
endfunction
```

A mesma porta agent_Jaqueline.out pode ser conectada a dois destinos diferentes:

1. Ao componente de cobertura coverage_in_Jaqueline
2. Ao FIFO agent_refmod que alimenta o modelo de referência

f. Vantagens Finais da Migração

1. Melhor escalabilidade: Múltiplos componentes podem observar as mesmas transações
2. Controle de simulação baseado em cobertura: A simulação termina quando atinge 100% de cobertura
3. Maior modularidade: Componentes podem ser adicionados sem afetar o fluxo existente
4. Conformidade com UVM: Segue o padrão de design recomendado pela metodologia UVM

Esta migração representa um passo importante na maturidade do testbench, preparando-o para verificações mais complexas e abrangentes.

As vantagens críticas desta mudança são:

1. Broadcast de Transações: Uma única transação pode ser distribuída para múltiplos componentes
2. Comunicação Não-Bloqueante: A fonte não precisa esperar que os consumidores processem a transação
3. Desacoplamento de Componentes: Reduz dependências estreitas entre geradores e consumidores

g. Componentes Implementados

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

   uvm_analysis_port #(j_jb) out;

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

   task run_phase (uvm_phase phase);
      j_jb  tr;

      forever begin
      #60; // Espera algum tempo antes da transação começar
      tr = j_jb::type_id::create("tr", this);
      assert(tr.randomize ());
      `bvm_begin_tr(tr)
      `uvm_info ("SOURCE", "Sending transaction by Aluna: Brito, Jaqueline",
UVM_LOW)
      out.write (tr);  // Envia transação para porta de análise
      end
   endtask
endclass
```

```
class coverage_in extends bvm_cover #(j_jb);
   `uvm_component_utils(coverage_in)

   covergroup transaction_covergroup;
      option.per_instance = 1;
      coverpoint coverage_transaction.j {
      bins d[4] = {[0:40]}; // 4 bins no intervalo 0-40
      option.at_least = 4;  // Pelo menos 4 ocorrências em cada bin
      }
   endgroup
   `bvm_cover_utils(j_jb)
endclass
```

```
class agent extends uvm_agent;
  `uvm_component_utils(agent)

   uvm_analysis_port #(j_jb) out;
   analysis_source source_Jaqueline;

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

   function void build_phase(uvm_phase phase);
```

```
        super.build_phase(phase);
        out = new("Saída", this);
        source_Jaqueline = analysis_source::type_id::create("source_Jaqueline",
this);
    endfunction

    function void connect_phase(uvm_phase phase);
        source_Jaqueline.out.connect(out);
    endfunction
endclass
```

```
class env extends uvm_env;
    `uvm_component_utils(env)

    agent agent_Jaqueline;
    coverage_in coverage_in_Jaqueline;
    uvm_tlm_analysis_fifo #(j_jb) agent_refmod;
    refmod refmod_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);
        agent_Jaqueline = agent::type_id::create("agent_Jaqueline", this);
        coverage_in_Jaqueline =
coverage_in::type_id::create("coverage_in_Jaqueline", this);
        agent_refmod = new("agent_refmod",this);
        refmod_h = refmod::type_id::create("refmod_h", this);
        sink_h = sink::type_id::create("sink_h", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        agent_Jaqueline.out.connect (coverage_in_Jaqueline.analysis_export);
        agent_Jaqueline.out.connect (agent_refmod.analysis_export);
        refmod_h.in.connect (agent_refmod.get_export );
        refmod_h.out.connect( sink_h.in );
    endfunction

endclass
```

h. Explicação do Fluxo de Transações

1. Geração de Transações: O analysis_source dentro do agente gera transações aleatórias a cada 60 unidades de tempo.
2. Broadcast de Transações: Cada transação é enviada simultaneamente para:
   O componente de cobertura (coverage_in_Jaqueline)
   O FIFO de análise (agent_refmod) que alimenta o modelo de referência

3. Processamento de Cobertura: O coverage_in_Jaqueline coleta estatísticas sobre as transações, monitorando se todos os bins estão sendo adequadamente cobertos.
4. Processamento pelo Modelo de Referência: O refmod_h recebe a transação do FIFO e gera a saída esperada.
5. Comparação (implícita): O sink_h receberia transações do DUT (não mostrado explicitamente) e as compararia com as do modelo de referência.
6. Término da Simulação: Quando todos os bins de cobertura atingem o mínimo de 4 hits cada (100% de cobertura), a simulação termina automaticamente.

Benefícios da Nova Arquitetura

1. Controle Baseado em Cobertura: A simulação não termina após um número arbitrário de transações, mas quando todos os cenários de interesse foram adequadamente testados.
2. Observabilidade Aprimorada: Múltiplos componentes podem monitorar o mesmo fluxo de transações sem interferência mútua.
3. Estrutura Hierárquica Adequada: Seguindo o padrão UVM, com o agente encapsulando a fonte e expondo uma interface unificada.
4. Flexibilidade para Extensão: Novos componentes de análise podem ser adicionados sem modificar os existentes.

i. Saída Esperada da Implementação

```
UVM_INFO @ 0: reporter [RNTST] Running test test_basic...
UVM_INFO @ 0: reporter [UVMTOP] UVM testbench hierarchy configuration:
...
UVM_INFO @ 60: uvm_test_top.env.agent_Jaqueline.source_Jaqueline [SOURCE] Sending
transaction by Aluna: Brito, Jaqueline
UVM_INFO @ 60: uvm_test_top.env.refmod_h [REFMOD] Received transaction:
               j=15  jb=xxx
UVM_INFO @ 60: uvm_test_top.env.refmod_h [REFMOD] Generating output:
               jb=15
UVM_INFO @ 120: uvm_test_top.env.agent_Jaqueline.source_Jaqueline [SOURCE] Sending
transaction by Aluna: Brito, Jaqueline
UVM_INFO @ 120: uvm_test_top.env.refmod_h [REFMOD] Received transaction:
               j=32  jb=xxx
UVM_INFO @ 120: uvm_test_top.env.refmod_h [REFMOD] Generating output:
               jb=32
...
[Continua até atingir a cobertura de 100%]
...
UVM_INFO @ 720: reporter [COVERG] Coverage: bins d[0] (0 to 10): 4/4 hits - 100%
UVM_INFO @ 720: reporter [COVERG] Coverage: bins d[1] (11 to 20): 5/4 hits - 125%
UVM_INFO @ 720: reporter [COVERG] Coverage: bins d[2] (21 to 30): 4/4 hits - 100%
UVM_INFO @ 720: reporter [COVERG] Coverage: bins d[3] (31 to 40): 4/4 hits - 100%
UVM_INFO @ 720: reporter [COVERG] Overall coverage: 100%
UVM_INFO @ 720: reporter [OBJTN] All coverage goals achieved, ending simulation
UVM_INFO @ 720: reporter [TEST_DONE] Test completed successfully
```

CONCLUSÃO

A implementação de cobertura funcional através da analysis_port permite que a verificação seja orientada por objetivos de cobertura claros, garantindo que todos os cenários necessários sejam exercitados antes do término da simulação. Esta abordagem representa um caso exemplar de aplicação dos princípios da metodologia UVM para criar um ambiente de verificação robusto, flexível e eficiente. À medida que a complexidade dos designs de hardware continua aumentando, arquiteturas de testbench como estas se tornam essenciais para assegurar a qualidade e a confiabilidade dos sistemas desenvolvidos.