Plano de Verificação PBL - Processador Single Cycle

Marcio Antônio Matias Costa

24 de Fevereiro de 2026

Sumário

1 Introdução

Este documento descreve o Plano de Verificação baseado em UVM (Universal Verification Methodology) para o processador RISC-V RV32I desenvolvido em SystemVerilog. A verificação será realizada utilizando o ambiente UVM disponível no Vivado Simulator.

O plano contempla:

- Verificação modular (unitária)
- Verificação de integração
- Verificação do processador completo
- Estratégia de cobertura funcional
- Estratégia de checagem automática (scoreboard)

2 Objetivos da Verificação

- Garantir conformidade com a ISA RV32I.
- Verificar correção funcional de cada módulo isoladamente.
- Validar integração entre datapath e control unit.
- Garantir funcionamento correto de branches, jumps e memória.
- Obter 100% de cobertura funcional dos tipos de instruções.

3 Estratégia Geral de Verificação

A verificação será dividida em duas fases principais:

3.1 Fase 1 – Verificação Modular

Cada módulo será verificado isoladamente com um ambiente UVM dedicado.

3.2 Fase 2 – Verificação do Processador Completo

Será utilizado um ambiente UVM full-chip contendo:

- Driver de instruções
- Monitor de memória
- Reference Model comportamental
- Scoreboard

4 Arquitetura do Ambiente UVM

4.1 Componentes UVM

O ambiente será composto por:

- **uvm_test**
- **uvm_env**
- **uvm_agent**
- **uvm_driver**
- **uvm_monitor**
- **uvm_sequencer**
- **uvm_scoreboard**
- **reference model**
- **coverage collector**

5 Diagrama de Blocos da Arquitetura de Verificação UVM

Esta seção apresenta a arquitetura do ambiente de verificação baseado na metodologia UVM utilizada para validar o processador RISC-V RV32I. O diagrama de blocos ilustra a organização hierárquica dos componentes de verificação e o fluxo de estímulo, observação e checagem funcional.

5.1 Descrição Geral da Arquitetura

O ambiente de verificação está estruturado de forma hierárquica e modular, seguindo as boas práticas da metodologia UVM. A arquitetura é composta pelos seguintes níveis principais:

- **UVM Test** — responsável pela configuração global do ambiente.
- **uvm_env** — encapsula todos os componentes de verificação.
- **Agents** — responsáveis pela interface com o DUT.
- **Scoreboard** — realiza a checagem funcional.
- **Reference Model** — modelo comportamental da ISA.
- **Coverage Collector** — coleta métricas de cobertura funcional.
- **DUT** — processador RISC-V sob verificação.

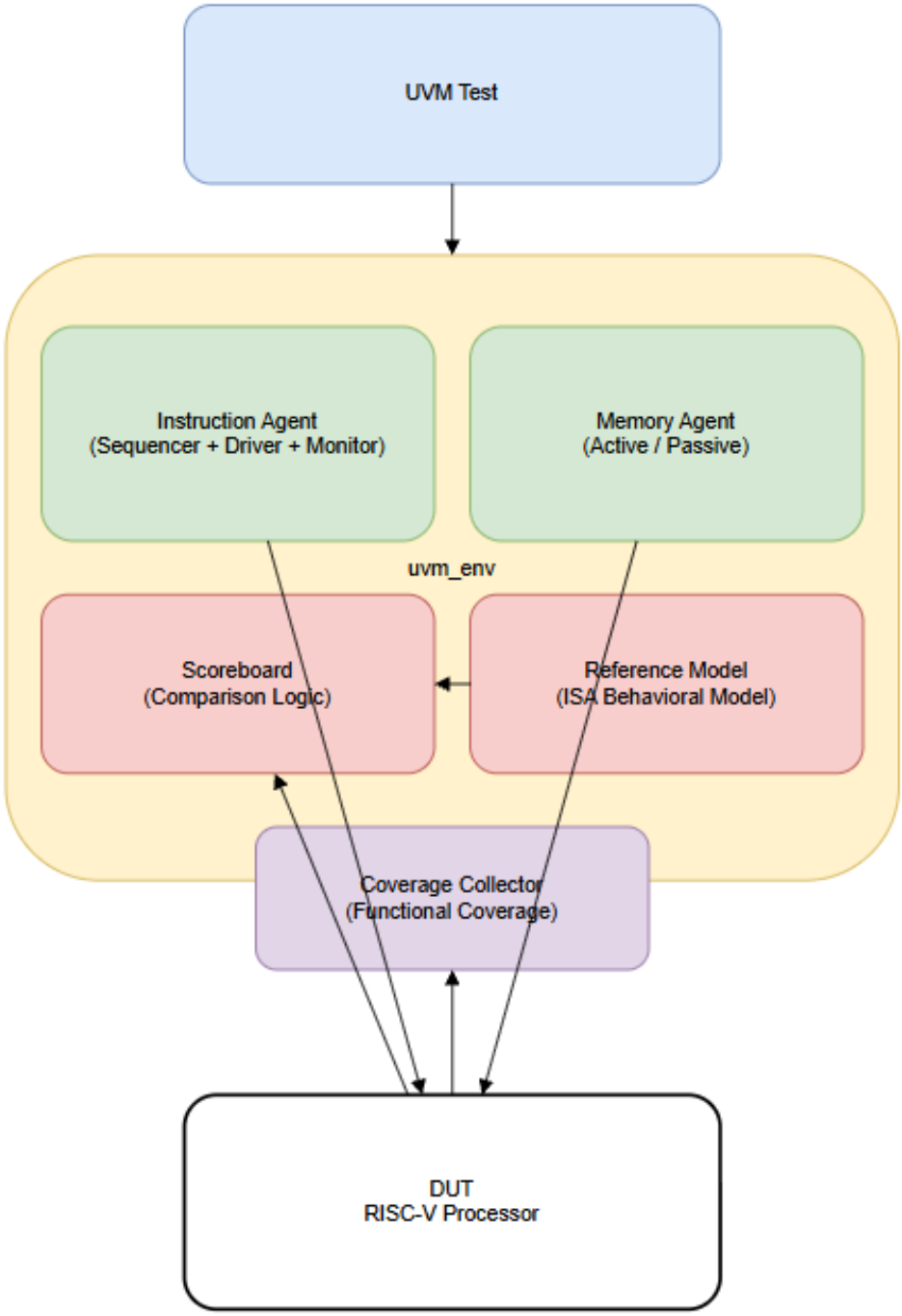

Figura 1: Arquitetura do Ambiente de Verificação UVM para o Processador RISC-V

5.2 UVM Test

O bloco **UVM Test** é o ponto mais alto da hierarquia de verificação. Ele é responsável por:

- Configurar parâmetros via `uvm_config_db`;
- Aplicar overrides via factory;
- Selecionar e iniciar sequências;

- Definir o modo de operação (modular ou full-chip).

5.3 UVM Environment (uvm_env)

O `uvm_env` encapsula todos os componentes do ambiente de verificação, garantindo organização estrutural e reuso. Ele contém:

- Instruction Agent;
- Memory Agent;
- Scoreboard;
- Reference Model;
- Coverage Collector.

5.4 Instruction Agent

O Instruction Agent é responsável por fornecer estímulos ao processador. Ele é composto por:

- Sequencer — gera transações de instruções;
- Driver — converte transações em sinais RTL;
- Monitor — observa instruções executadas.

Este agente opera em modo ativo durante a verificação full-chip.

5.5 Memory Agent

O Memory Agent interage com a interface de memória de dados do processador. Ele pode operar em dois modos:

- **Ativo** — gera respostas de leitura e controla latências;
- **Passivo** — apenas monitora transações.

Essa flexibilidade permite reutilização futura em integração de SoC.

5.6 Reference Model

O Reference Model implementa um modelo comportamental da ISA RV32I. Ele executa as mesmas instruções aplicadas ao DUT e gera o estado esperado do sistema:

- Atualização do PC;
- Atualização do banco de registradores;
- Operações de memória;
- Decisão de branch.

5.7 Scoreboard

O Scoreboard recebe:

- Transações observadas pelos monitores;
- Resultados esperados do Reference Model.

Ele realiza a comparação entre o comportamento do DUT e o modelo de referência, reportando divergências por meio de mensagens de erro UVM.

5.8 Coverage Collector

O Coverage Collector coleta métricas de cobertura funcional, incluindo:

- Cobertura de opcode;
- Cobertura de funct3/funct7;
- Branch taken vs not taken;
- Acessos à memória;
- Cruzamentos relevantes.

Essa abordagem permite aplicar a metodologia Coverage-Driven Verification (CDV).

5.9 Fluxo de Operação

O fluxo de dados no ambiente ocorre da seguinte forma:

1. Sequências geram instruções;
2. O driver aplica estímulos ao DUT;
3. Monitores capturam respostas;
4. O Reference Model calcula o comportamento esperado;
5. O Scoreboard compara os resultados;
6. O Coverage Collector registra métricas.

5.10 Conclusão

A arquitetura apresentada garante:

- Separação clara entre estímulo e checagem;
- Reuso modular dos componentes;
- Escalabilidade para integração futura em SoC;
- Compatibilidade com metodologias industriais de verificação.

6 Verificação Modular

6.1 1. MUX2

Objetivo: Validar seleção correta entre entradas A e B.

Testes:

- s = 0 → saída = B
- s = 1 → saída = A
- Valores extremos (0x00000000 e 0xFFFFFFFF)

Cobertura:

- Toggle de s
- Cobertura cruzada (s x valores)

6.2 2. Register File

Objetivos:

- Verificar leitura dupla simultânea.
- Verificar escrita síncrona.
- Garantir que x0 permaneça sempre zero.

Testes:

- Escrita e leitura no mesmo ciclo.
- Escrita em registradores aleatórios.
- Tentativa de escrita em x0.

Cobertura:

- Endereços 0–31
- Operações de escrita
- Escrita em x0

6.3 3. ALU

Operações verificadas:

- ADD, SUB
- AND, OR, XOR
- SLT
- Shift lógico e aritmético

Estratégia:

- Sequências randomizadas
- Comparação com modelo comportamental SystemVerilog

6.4 4. Control Unit

Objetivo: Verificar geração correta de sinais de controle para todos os opcodes RV32I suportados.

Cobertura:

- Todos os opcodes
- Todos os funct3
- Todos os funct7 relevantes

7 Verificação do Processador Completo

7.1 Arquitetura do Testbench

O ambiente full-chip conterá:

- Instruction Agent
- Memory Agent
- Scoreboard
- Reference Model em SystemVerilog

7.2 Reference Model

Será implementado um modelo comportamental simples da ISA RV32I que:

- Decodifica instruções
- Atualiza registradores
- Atualiza PC
- Simula memória

O Scoreboard comparará:

- PC esperado vs PC real
- Banco de registradores esperado vs DUT
- Escritas de memória

8 Estratégia de Testes

8.1 Testes Direcionados

- ADD básico
- Branch tomado e não tomado
- JAL e JALR
- Load e Store

8.2 Testes Randomizados

- Sequência randômica de instruções válidas
- Restrições para evitar instruções ilegais

8.3 Testes de Stress

- 10.000+ instruções
- Branches consecutivos
- Escritas contínuas em registradores

9 Cobertura Funcional

9.1 Cobertura de Instruções

- Tipo R
- Tipo I
- Tipo S
- Tipo B
- Tipo J

9.2 Cobertura Cruzada

- opcode x branch tomado
- opcode x escrita em registrador
- opcode x acesso à memória

Meta: 100% de cobertura funcional.

10 Critérios de Aceitação

O projeto será considerado verificado quando:

- 100% de cobertura funcional atingida
- 0 erros no scoreboard
- 0 mismatches entre DUT e reference model
- Todos os testes direcionados aprovados

11 Ferramentas Utilizadas

- Vivado Simulator
- SystemVerilog
- UVM 1.2

12 Configuração e Reuso

Esta seção descreve a estratégia de configuração do ambiente UVM e as diretrizes de reuso visando escalabilidade futura para integração em nível de SoC.

12.1 Estratégia de Configuração via uvm_config_db

A configuração do ambiente será realizada utilizando o mecanismo `uvm_config_db`, permitindo desacoplamento entre componentes e maior flexibilidade estrutural.

Objetivos

- Evitar dependência hierárquica rígida.
- Permitir parametrização dinâmica de agentes.
- Facilitar reuso do ambiente em diferentes contextos.

Itens configuráveis

Os seguintes parâmetros serão propagados via `uvm_config_db`:

- Virtual interfaces (instruction_if, data_if)
- Ativação ou desativação de agentes
- Habilitação de cobertura funcional
- Modo de operação (unitário ou full-chip)
- Número de instruções da sequência

Estratégia de Implementação

- A configuração será realizada no `build_phase` do `uvm_test`.
- Componentes internos recuperarão parâmetros utilizando `uvm_config_db::get()`.
- Valores padrão serão definidos para permitir execução independente.

Essa abordagem garante flexibilidade e independência hierárquica.

12.2 Controle de Agentes Ativos e Passivos

Para permitir reutilização em diferentes níveis de abstração, os agentes serão configuráveis como:

- **Ativos** — incluem driver + sequencer + monitor.
- **Passivos** — incluem apenas monitor.

Aplicação no Projeto

- Na verificação modular: agentes ativos.
- Na verificação full-chip:
  - Instruction Agent → ativo.
  - Memory Agent → passivo ou ativo dependendo do cenário.

  O modo será definido via parâmetro configurável no `uvm_config_db`.

Benefícios

- Permite integração futura em SoC maior.
- Evita duplicação de ambientes.
- Suporta cenários de co-simulação.

12.3 Estratégia de Reuso para Integração em SoC

O ambiente será projetado para permitir integração futura do processador como IP em um SoC.

Diretrizes de Reuso

- Interfaces padronizadas e desacopladas.
- Ausência de referências hierárquicas diretas ao DUT.
- Uso exclusivo de virtual interfaces.
- Separação clara entre estímulo e checagem.

13 Conclusão

Este plano de verificação estabelece uma metodologia estruturada e escalável baseada em UVM para validar completamente o processador RISC-V RV32I. A abordagem modular seguida de integração total garante robustez, reuso e alta cobertura funcional.