Verificação Funcional de Modelos Transacionais de Processadores

Trabalho de Conclusão de Curso 2

Jair Dias de Oliveira Junior

Universidade de Brasília

24 de junho de 2014

Proposta e Objetivos

Proposta

Desenvolver um ambiente de verificação funcional para um modelo de processador descrito em SystemC TLM-2.0.

Proposta e Objetivos

Proposta

Desenvolver um ambiente de verificação funcional para um modelo de processador descrito em SystemC TLM-2.0.

Objetivos

- Buscar ferramentas e metodologias praticadas no mercado

Proposta e Objetivos

Proposta

Desenvolver um ambiente de verificação funcional para um modelo de processador descrito em SystemC TLM-2.0.

Objetivos

- Buscar ferramentas e metodologias praticadas no mercado
- Utilizar como estudo de caso a ISA do MIPS32

Proposta e Objetivos

Proposta

Desenvolver um ambiente de verificação funcional para um modelo de processador descrito em SystemC TLM-2.0.

Objetivos

- Buscar ferramentas e metodologias praticadas no mercado
- Utilizar como estudo de caso a ISA do MIPS32
- Desenvolver o plano de verificação

Proposta e Objetivos

Proposta

Desenvolver um ambiente de verificação funcional para um modelo de processador descrito em SystemC TLM-2.0.

Objetivos

- Buscar ferramentas e metodologias praticadas no mercado
- Utilizar como estudo de caso a ISA do MIPS32
- Desenvolver o plano de verificação
- Construir o *testbench* utilizando a UVM + UVMC

Proposta e Objetivos

Proposta

Desenvolver um ambiente de verificação funcional para um modelo de processador descrito em SystemC TLM-2.0.

Objetivos

- Buscar ferramentas e metodologias praticadas no mercado
- Utilizar como estudo de caso a ISA do MIPS32
- Desenvolver o plano de verificação
- Construir o *testbench* utilizando a UVM + UVMC
- Executar simulações e interpretar resultados

A Evolução das Metodologias de Verificação

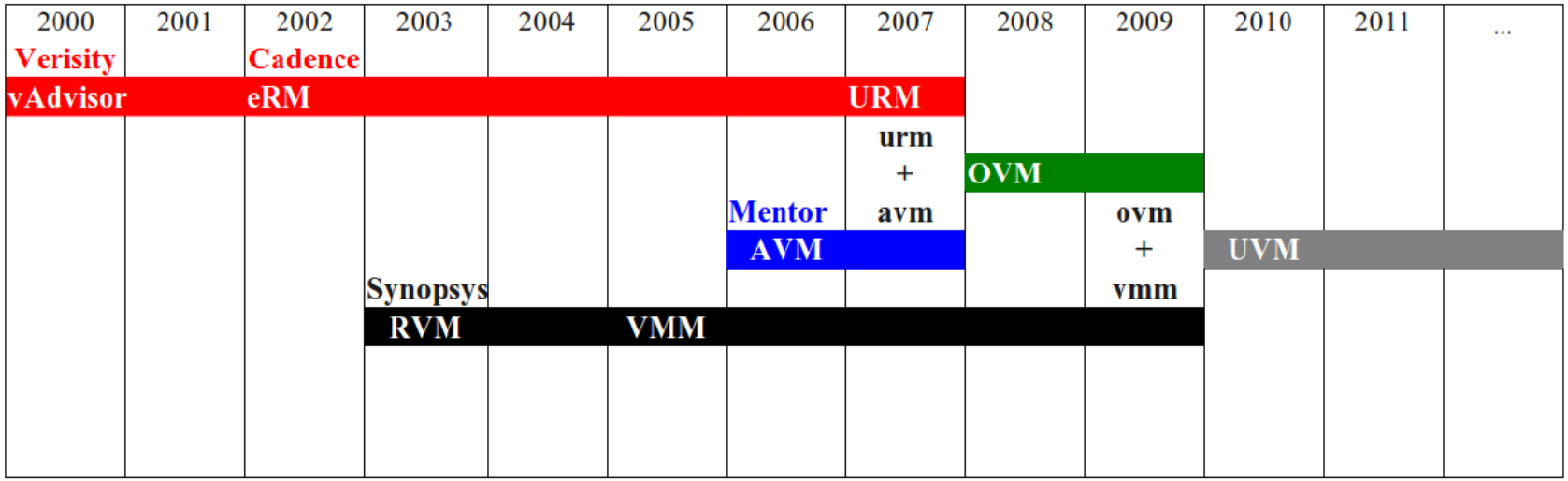


Figura: Linha do tempo das metodologias [Pagliarini 2011]

Grupo ASI (*Accellera Systems Initiative*)

Membros Corporativos

Membros Associados

Figura: Principais membros do grupo ASI

Características da UVM

Principais Características:

- Suportada pelos líderes de mercado EDA

Características da UVM

Principais Características:

- Suportada pelos líderes de mercado EDA
- Abordagem uniformizada

Características da UVM

Principais Características:

- Suportada pelos líderes de mercado EDA
- Abordagem uniformizada
- Biblioteca de classes SystemVerilog *open source*

Características da UVM

Principais Características:

- Suportada pelos líderes de mercado EDA
- Abordagem uniformizada
- Biblioteca de classes SystemVerilog *open source*
- *Testbenches* configuráveis e flexíveis

Características da UVM

Principais Características:

- Suportada pelos líderes de mercado EDA
- Abordagem uniformizada
- Biblioteca de classes SystemVerilog *open source*
- *Testbenches* configuráveis e flexíveis
- Possibilita a criação de componentes reutilizáveis

Características da UVM

Principais Características:

- Suportada pelos líderes de mercado EDA
- Abordagem uniformizada
- Biblioteca de classes SystemVerilog *open source*
- *Testbenches* configuráveis e flexíveis
- Possibilita a criação de componentes reutilizáveis
- Geração de estímulos aleatórios e ajustados

Características da UVM

Principais Características:

- Suportada pelos líderes de mercado EDA
- Abordagem uniformizada
- Biblioteca de classes SystemVerilog *open source*
- *Testbenches* configuráveis e flexíveis
- Possibilita a criação de componentes reutilizáveis
- Geração de estímulos aleatórios e ajustados
- Checagem automática e cobertura funcional

Características da UVM

Principais Características:

- Suportada pelos líderes de mercado EDA
- Abordagem uniformizada
- Biblioteca de classes SystemVerilog *open source*
- *Testbenches* configuráveis e flexíveis
- Possibilita a criação de componentes reutilizáveis
- Geração de estímulos aleatórios e ajustados
- Checagem automática e cobertura funcional
- Suporte para modelos em várias HDLs

Características da UVM

Principais Características:

- Suportada pelos líderes de mercado EDA
- Abordagem uniformizada
- Biblioteca de classes SystemVerilog *open source*
- *Testbenches* configuráveis e flexíveis
- Possibilita a criação de componentes reutilizáveis
- Geração de estímulos aleatórios e ajustados
- Checagem automática e cobertura funcional
- Suporte para modelos em várias HDLs
- Suporte completo para TLM-2.0

Hierarquia de Classes

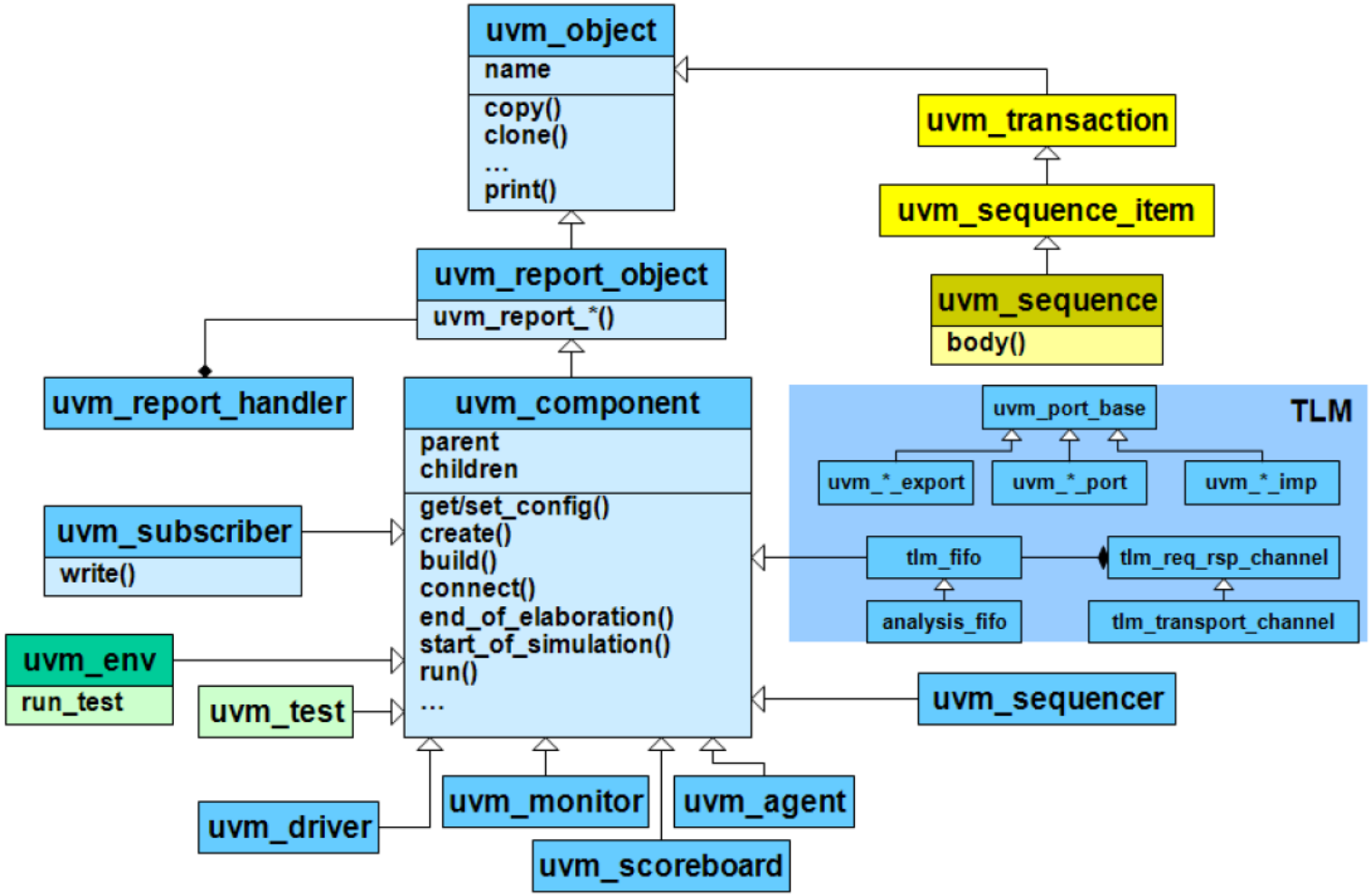
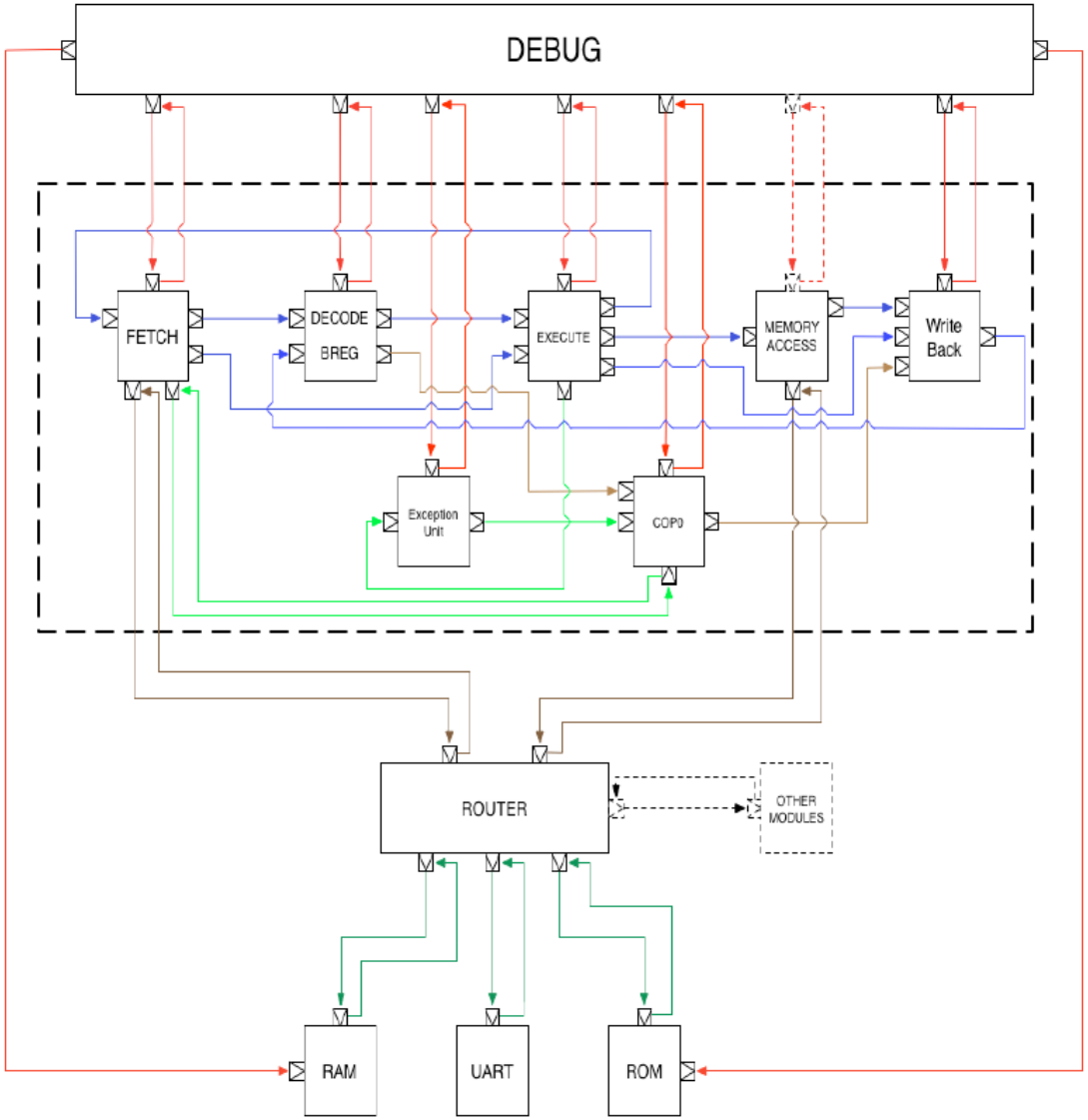


Figura: Hierarquia de classes UVM (parcial)

MIPS32 em SystemC TLM-2.0



Instruções da ISA MIPS32

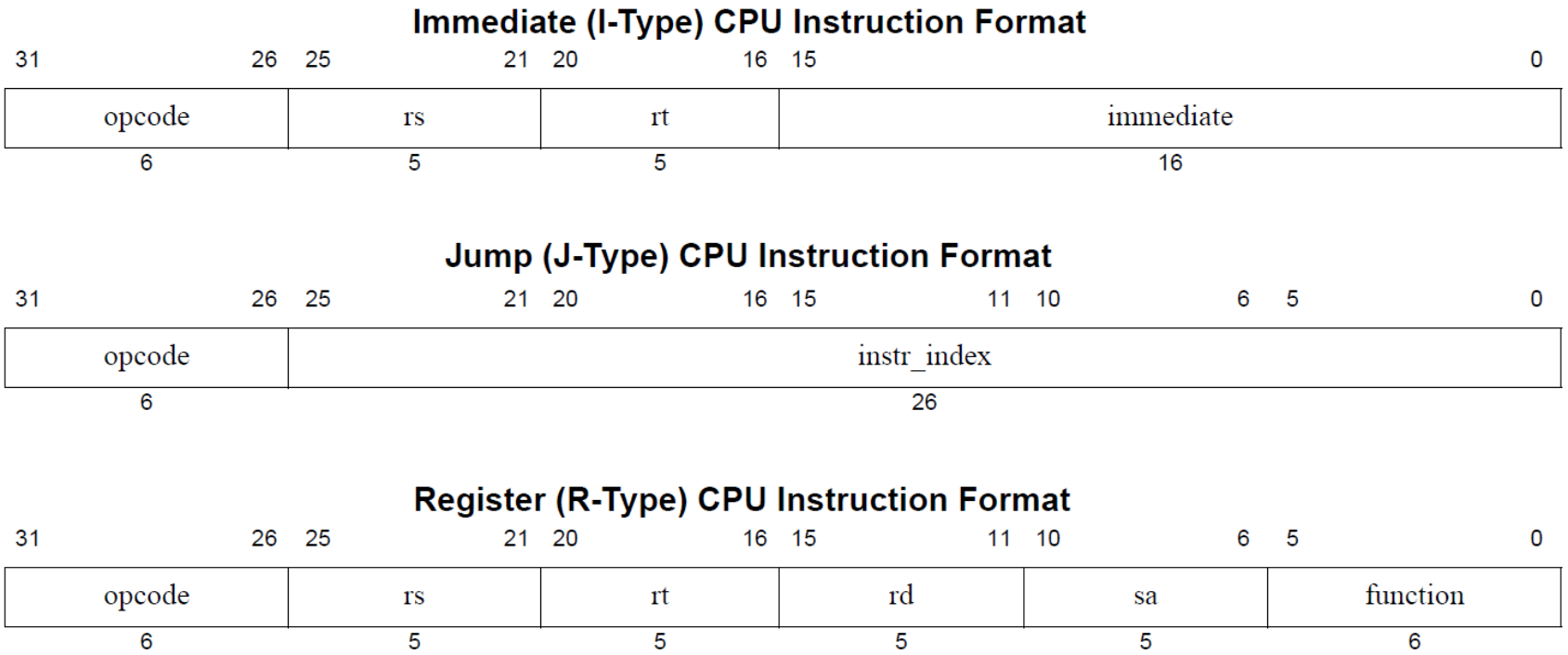


Figura: Tipos de instruções ISA MIPS32

O Plano de Verificação

Documento que define o escopo do problema, *o que* verificar e *como* verificar.

O Plano de Verificação

Documento que define o escopo do problema, *o que* verificar e *como* verificar.

Informações Prévias:

- Especificações arquiteturais

O Plano de Verificação

Documento que define o escopo do problema, *o que* verificar e *como* verificar.

Informações Prévias:

- Especificações arquiteturais
- Modos de operação do dispositivo

O Plano de Verificação

Documento que define o escopo do problema, *o que* verificar e *como* verificar.

Informações Prévias:

- Especificações arquiteturais
- Modos de operação do dispositivo
- Comportamentos esperados

O Plano de Verificação

Documento que define o escopo do problema, *o que* verificar e *como* verificar.

Informações Prévias:

- Especificações arquiteturais
- Modos de operação do dispositivo
- Comportamentos esperados
- Padrões que devem ser atendidos

O Plano de Verificação

Documento que define o escopo do problema, *o que* verificar e *como* verificar.

Informações Prévias:

- Especificações arquiteturais
- Modos de operação do dispositivo
- Comportamentos esperados
- Padrões que devem ser atendidos
- Lista de interfaces de entrada e saída

O Plano de Verificação

Documento que define o escopo do problema, *o que* verificar e *como* verificar.

Informações Prévias:

- Especificações arquiteturais
- Modos de operação do dispositivo
- Comportamentos esperados
- Padrões que devem ser atendidos
- Lista de interfaces de entrada e saída
- Cenários críticos

O Plano de Verificação

Documento que define o escopo do problema, *o que* verificar e *como* verificar.

Informações Prévias:

- Especificações arquiteturais
- Modos de operação do dispositivo
- Comportamentos esperados
- Padrões que devem ser atendidos
- Lista de interfaces de entrada e saída
- Cenários críticos
- Aplicações onde o dispositivo será usado

Nível de Verificação

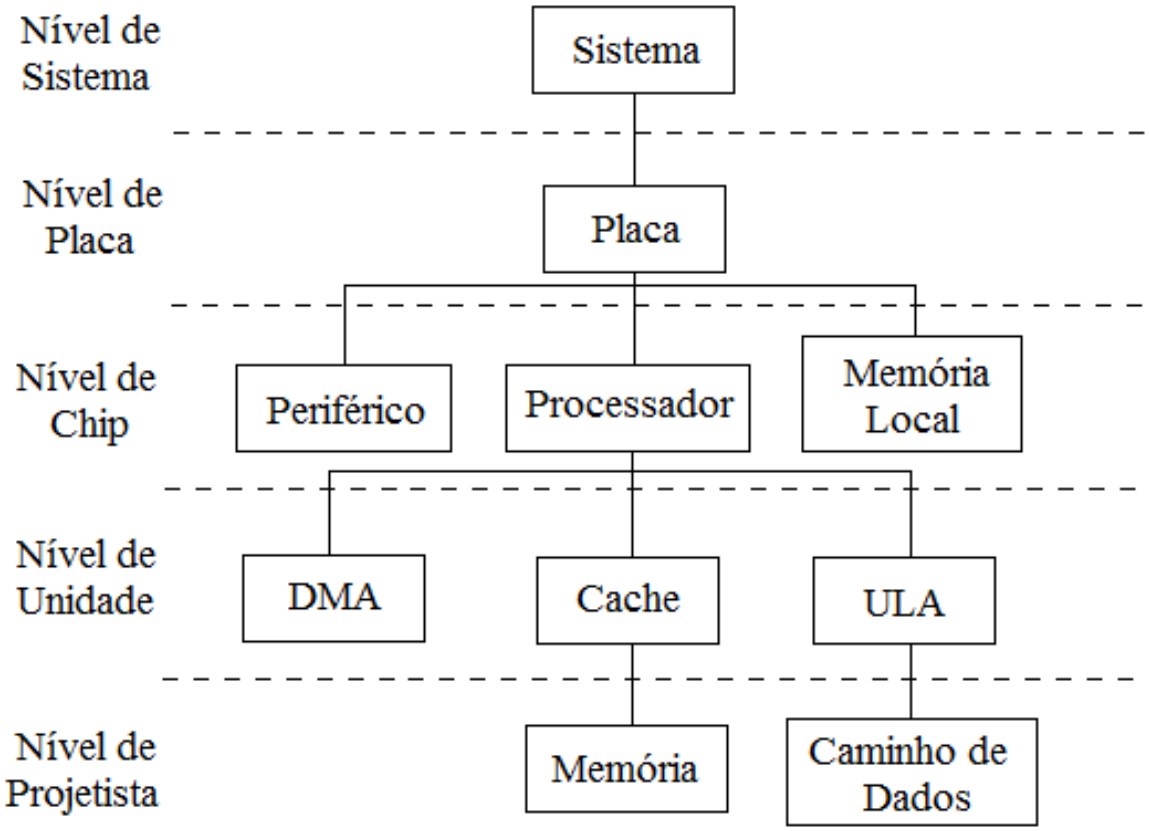


Figura: Níveis de Verificação

Métricas de Cobertura

Definem quais pontos deseja-se verificar em um modelo e sua medida indica o progresso da verificação.

Métricas de Cobertura

Definem quais pontos deseja-se verificar em um modelo e sua medida indica o progresso da verificação.

Componentes de Verificação Envolvidos:

- Agente
  - Monitor

Métricas de Cobertura

Definem quais pontos deseja-se verificar em um modelo e sua medida indica o progresso da verificação.

Componentes de Verificação Envolvidos:

- Agente
  - Monitor
- *Subscriber*

Métricas de Cobertura

Definem quais pontos deseja-se verificar em um modelo e sua medida indica o progresso da verificação.

Componentes de Verificação Envolvidos:

- Agente
  - Monitor
- *Subscriber*

Metas de Cobertura:

- Escrita em cada registrador: 10000 vezes

Métricas de Cobertura

Definem quais pontos deseja-se verificar em um modelo e sua medida indica o progresso da verificação.

Componentes de Verificação Envolvidos:

- Agente
  - Monitor
- *Subscriber*

Metas de Cobertura:

- Escrita em cada registrador: 10000 vezes
- Leitura em cada registrador: 10000 vezes

Métricas de Cobertura

Definem quais pontos deseja-se verificar em um modelo e sua medida indica o progresso da verificação.

Componentes de Verificação Envolvidos:

- Agente
  - Monitor
- *Subscriber*

Metas de Cobertura:

- Escrita em cada registrador: 10000 vezes
- Leitura em cada registrador: 10000 vezes
- Valor do campo de *opcodes*: 1000 vezes cada valor possível

Geração de Estímulos

Os estímulos gerados são responsáveis por exercitar o DUT de maneira que suas funcionalidades sejam completamente verificadas.

Geração de Estímulos

Os estímulos gerados são responsáveis por exercitar o DUT de maneira que suas funcionalidades sejam completamente verificadas.

Componentes de Verificação Envolvidos:

- Agente
  - Sequenciador
  - *Driver*

Geração de Estímulos

Os estímulos gerados são responsáveis por exercitar o DUT de maneira que suas funcionalidades sejam completamente verificadas.

Componentes de Verificação Envolvidos:

- Agente
  - Sequenciador
  - *Driver*

Estímulos Gerados:

- Escrita:
  - Registrador: direta e sequencial
  - Dado: aleatório

Geração de Estímulos

Os estímulos gerados são responsáveis por exercitar o DUT de maneira que suas funcionalidades sejam completamente verificadas.

Componentes de Verificação Envolvidos:

- Agente
  - Sequenciador
  - *Driver*

Estímulos Gerados:

- Escrita:
  - Registrador: direta e sequencial
  - Dado: aleatório
- Leitura: aleatória

Geração de Estímulos

Os estímulos gerados são responsáveis por exercitar o DUT de maneira que suas funcionalidades sejam completamente verificadas.

Componentes de Verificação Envolvidos:

- Agente
  - Sequenciador
  - *Driver*

Estímulos Gerados:

- Escrita:
  - Registrador: direta e sequencial
  - Dado: aleatório
- Leitura: aleatória
- *Opcode*: aleatório

Checagem de Respostas

A constatação do correto funcionamento do DUT é feita comparando-se os dados recebidos como resposta com um modelo comportamental esperado.

Checagem de Respostas

A constatação do correto funcionamento do DUT é feita comparando-se os dados recebidos como resposta com um modelo comportamental esperado.

Componentes de Verificação Envolvidos:

- Agente
  - Monitor

Checagem de Respostas

A constatação do correto funcionamento do DUT é feita comparando-se os dados recebidos como resposta com um modelo comportamental esperado.

Componentes de Verificação Envolvidos:

- Agente
  - Monitor
- *Scoreboard*

Checagem de Respostas

A constatação do correto funcionamento do DUT é feita comparando-se os dados recebidos como resposta com um modelo comportamental esperado.

Componentes de Verificação Envolvidos:

- Agente
  - Monitor
- *Scoreboard*
- FIFOs de Análise

Checagem de Respostas

A constatação do correto funcionamento do DUT é feita comparando-se os dados recebidos como resposta com um modelo comportamental esperado.

Componentes de Verificação Envolvidos:

- Agente
  - Monitor
- *Scoreboard*
- FIFOs de Análise

Modo de comparação:

- Captura de dados na saída do módulo

Checagem de Respostas

A constatação do correto funcionamento do DUT é feita comparando-se os dados recebidos como resposta com um modelo comportamental esperado.

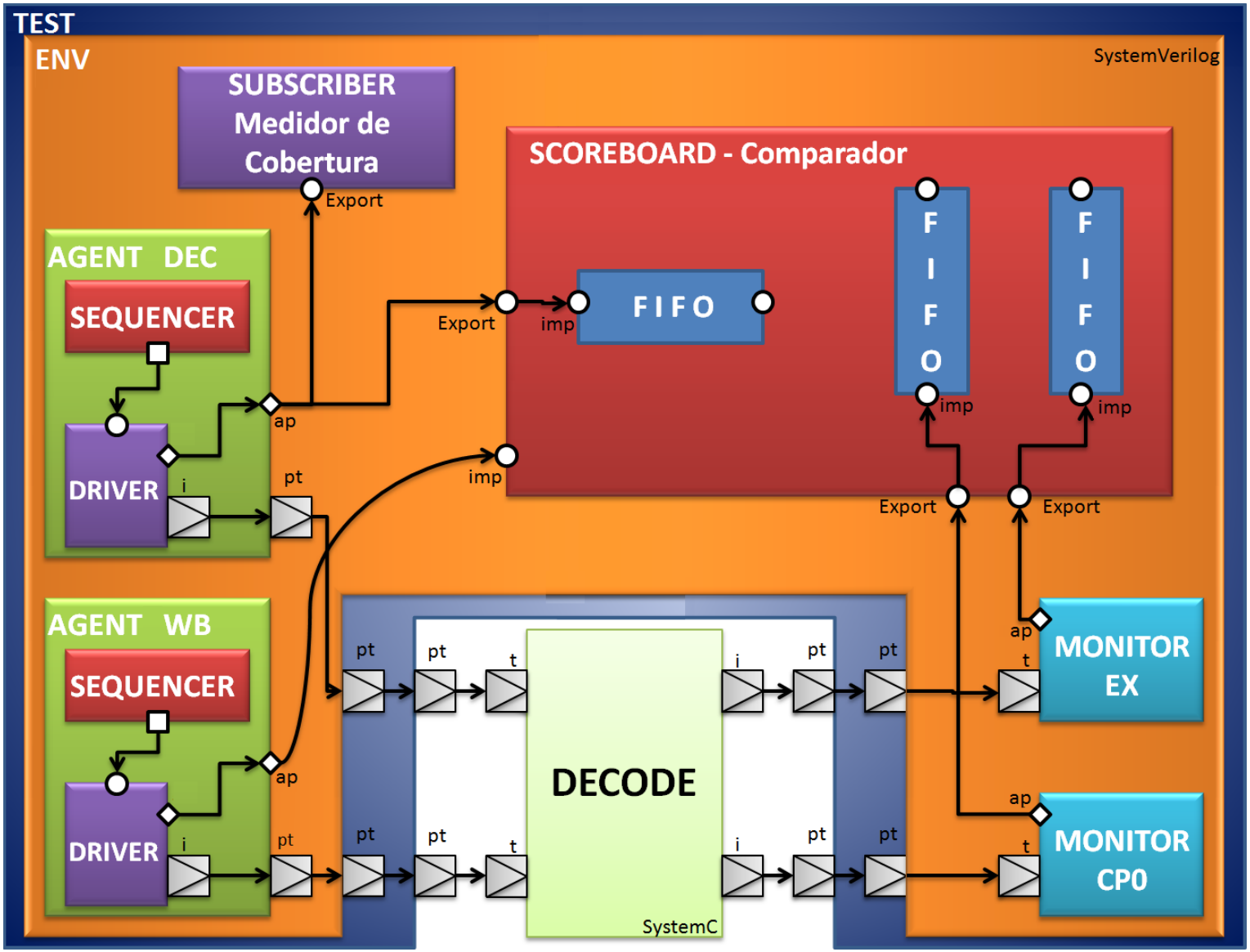
Componentes de Verificação Envolvidos:

- Agente
  - Monitor
- *Scoreboard*
- FIFOs de Análise

Modo de comparação:

- Captura de dados na saída do módulo
- Comparação com um modelo criado no *testbench*

Estrutura do Testbench



Código - Env

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

    uvm_tlm_b_passthrough_initiator_socket #() env_dec_pt_socket;
    uvm_tlm_b_passthrough_initiator_socket #() env_wb_pt_socket;
    uvm_tlm_b_passthrough_target_socket #() env_ex_pt_socket;
    uvm_tlm_b_passthrough_target_socket #() env_cp0_pt_socket;

    my_agent        agent_wb;
    my_agent        agent_dec;
    my_monitor      monitor_ex;
    my_monitor      monitor_cp0;
    my_subscriber   subscriber_h;
    my_scoreboard   scoreboard_h;

    function new(string name, uvm_component parent=null);
        super.new(name,parent);
        env_wb_pt_socket = new("env_wb_pt_socket", this);
        env_dec_pt_socket = new("env_dec_pt_socket", this);
        env_ex_pt_socket = new("env_ex_pt_socket", this);
        env_cp0_pt_socket = new("env_cp0_pt_socket", this);
    endfunction
```

Código - Env

```
    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        agent_dec = my_agent::type_id::create("agent_dec", this);
        agent_wb    = my_agent::type_id::create("agent_wb", this);
        monitor_ex  = my_monitor::type_id::create("monitor_ex", this);
        monitor_cp0 = my_monitor::type_id::create("monitor_cp0", this);
        subscriber_h = my_subscriber::type_id::create("subscriber_h", this);
        scoreboard_h = my_scoreboard::type_id::create("scoreboard_h", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        agent_dec.agent_pt_socket.connect   ( env_dec_pt_socket );
        agent_wb.agent_pt_socket.connect    ( env_wb_pt_socket );
        env_ex_pt_socket.connect    ( monitor_ex.tsocket );
        env_cp0_pt_socket.connect   ( monitor_cp0.tsocket );
        agent_dec.aport.connect( subscriber_h.analysis_export );
        agent_wb.aport.connect   ( scoreboard_h.imp_data );
        agent_dec.aport.connect  ( scoreboard_h.xport_in );
        monitor_ex.aport.connect ( scoreboard_h.xport_ex );
        monitor_cp0.aport.connect( scoreboard_h.xport_cp0 );
    endfunction
endclass
```

Resultados

```
UVM_INFO ./common/testbench.sv(693) @ 0: teste [TEST] Starting...
UVM_INFO ./common/testbench.sv(202) @ 0: teste.my_env_h.monitor_ex [MONITOR] Starting...
(...)
UVM_INFO ./common/testbench.sv(702) @ 0: teste [TEST] Data Sequence finished his job
UVM_INFO ./common/testbench.sv(703) @ 0: teste [TEST] All register were succesfull writen
UVM_INFO ./common/testbench.sv(268) @ 0: teste.my_env_h.subscriber_h [SUBSCRIBER] Coverage: 0%
(...)
UVM_INFO ./common/testbench.sv(268) @ 0: teste.my_env_h.subscriber_h [SUBSCRIBER] Coverage: 50%
(...)
UVM_INFO ./common/testbench.sv(268) @ 0: teste.my_env_h.subscriber_h [SUBSCRIBER] Coverage: 99%
UVM_INFO ./common/testbench.sv(268) @ 0: teste.my_env_h.subscriber_h [SUBSCRIBER] Coverage: 100%
UVM_INFO ./common/testbench.sv(716) @ 0: teste [TEST] Coverage Goals have been achieved!
UVM_INFO ./common/testbench.sv(717) @ 0: teste [TEST] Ending test and drop objection...
UVM_INFO /home/alunos/junior_tcc2/uvm-1.1d/src/base/uvm_objection.svh(1268) @ 0: reporter [TEST_DONE] 'run' phase is ready to proce
UVM_INFO ./common/testbench.sv(571) @ 0: teste.my_env_h.scoreboard_h [SCOREBOARD] Matches:       328238
UVM_INFO ./common/testbench.sv(572) @ 0: teste.my_env_h.scoreboard_h [SCOREBOARD] Mismatches: 0
UVM_INFO ./common/testbench.sv(281) @ 0: teste.my_env_h.subscriber_h [COVERAGE COLLECTOR] Number of Transactions: 328238
```

Trabalhos Futuros

- Melhorar a geração de estímulos

Trabalhos Futuros

- Melhorar a geração de estímulos
- Melhorar a medida de cobertura

Trabalhos Futuros

- Melhorar a geração de estímulos
- Melhorar a medida de cobertura
- Gerar mais testes com cenários diferentes

Trabalhos Futuros

- Melhorar a geração de estímulos
- Melhorar a medida de cobertura
- Gerar mais testes com cenários diferentes
- Tornar o *testbench* mais configurável

Trabalhos Futuros

- Melhorar a geração de estímulos
- Melhorar a medida de cobertura
- Gerar mais testes com cenários diferentes
- Tornar o *testbench* mais configurável
- Efetuar a verificação de outros módulos do processador

Trabalhos Futuros

- Melhorar a geração de estímulos
- Melhorar a medida de cobertura
- Gerar mais testes com cenários diferentes
- Tornar o *testbench* mais configurável
- Efetuar a verificação de outros módulos do processador
- Verificar todo o processador a nível de *chip*

Trabalhos Futuros

- Melhorar a geração de estímulos
- Melhorar a medida de cobertura
- Gerar mais testes com cenários diferentes
- Tornar o *testbench* mais configurável
- Efetuar a verificação de outros módulos do processador
- Verificar todo o processador a nível de *chip*
- Publicar artigos

Obrigado!