Verificação Funcional de Modelos Transacionais de Processadores

Trabalho de Conclusão de Curso 1

Jair Dias de Oliveira Junior

Universidade de Brasília

6 de dezembro de 2013.

Sumário

1. Proposta e Objetivos
2. Motivação e Justificativa
3. O que é Verificação?
4. Métricas de Cobertura
5. Processo de Verificação
6. Modelagem de Sistemas
   - ESL - Electronic-System Level
   - TLM - Transaction Level Modeling
7. Metodologias de Verificação
   - Estrutura do Testbench
   - Metodologias Existentes
   - UVM - Universal Verification Methodology
8. Estudo de Caso

Proposta e Objetivos

Proposta

Desenvolver um ambiente de verificação para modelos de processadores descritos em SystemC TLM-2.0.

Proposta e Objetivos

Proposta

Desenvolver um ambiente de verificação para modelos de processadores descritos em SystemC TLM-2.0.

Objetivos

- Buscar ferramentas e metodologias praticadas no mercado

Proposta e Objetivos

Proposta

Desenvolver um ambiente de verificação para modelos de processadores descritos em SystemC TLM-2.0.

Objetivos

- Buscar ferramentas e metodologias praticadas no mercado
- Verificar a adequabilidade de tais ferramentas e metodologias

Proposta e Objetivos

Proposta

Desenvolver um ambiente de verificação para modelos de processadores descritos em SystemC TLM-2.0.

Objetivos

- Buscar ferramentas e metodologias praticadas no mercado
- Verificar a adequabilidade de tais ferramentas e metodologias
- Utilizar como caso de teste a ISA do MIPS32

Proposta e Objetivos

Proposta

Desenvolver um ambiente de verificação para modelos de processadores descritos em SystemC TLM-2.0.

Objetivos

- Buscar ferramentas e metodologias praticadas no mercado
- Verificar a adequabilidade de tais ferramentas e metodologias
- Utilizar como caso de teste a ISA do MIPS32
- Definir quais métodos serão aplicados para a verificação

Motivação e Justificativa

Desenvolver sistemas de alta complexidade se tornou um desafio devido à dificuldade em se manter a conformidade entre os requisitos iniciais e o sistema desenvolvido [Tobar 2010].

Motivação e Justificativa

Desenvolver sistemas de alta complexidade se tornou um desafio devido à dificuldade em se manter a conformidade entre os requisitos iniciais e o sistema desenvolvido [Tobar 2010].

Motivos para verificar:

- Aumento das falhas em silício

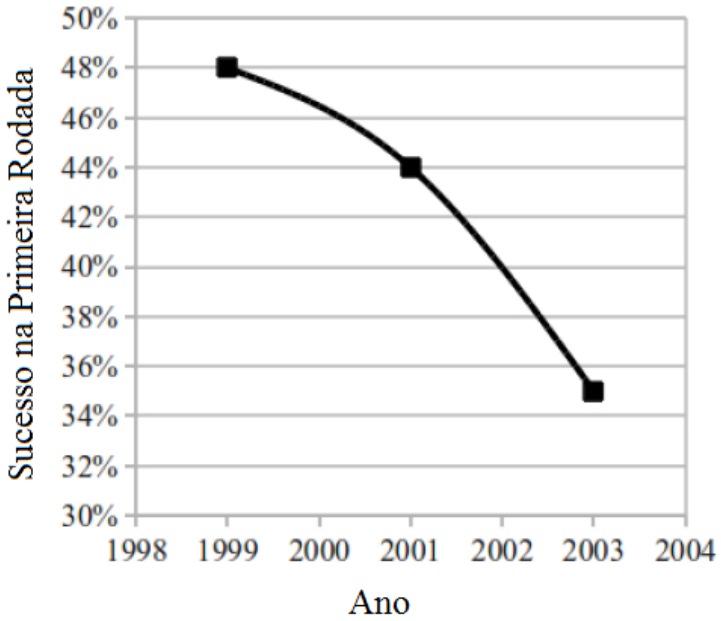


Figura: Pesquisa Collet International Research Inc., 2003

Motivação e Justificativa

Desenvolver sistemas de alta complexidade se tornou um desafio devido à dificuldade em se manter a conformidade entre os requisitos iniciais e o sistema desenvolvido [Tobar 2010].

Motivos para verificar:

- Aumento das falhas em silício
- Evitar gastos excedentes

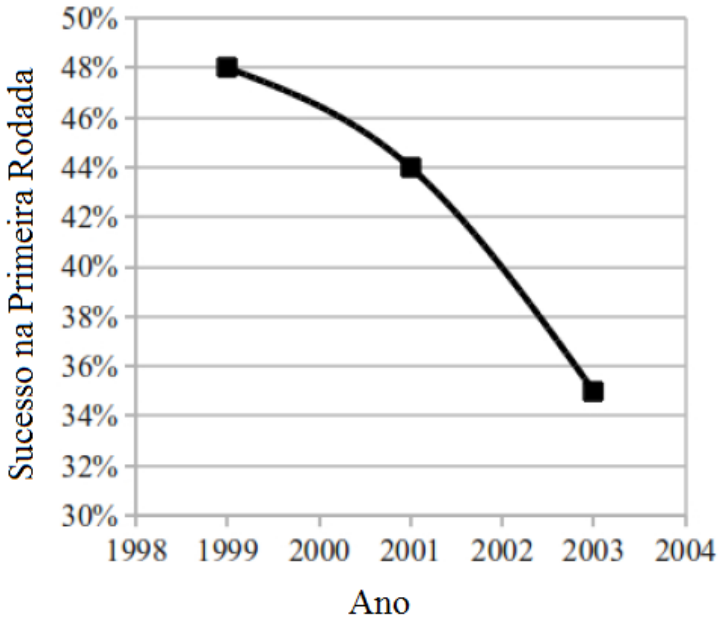


Figura: Pesquisa Collet International Research Inc., 2003

Motivação e Justificativa

Desenvolver sistemas de alta complexidade se tornou um desafio devido à dificuldade em se manter a conformidade entre os requisitos iniciais e o sistema desenvolvido [Tobar 2010].

Motivos para verificar:

- Aumento das falhas em silício
- Evitar gastos excedentes
- Utilização de IP

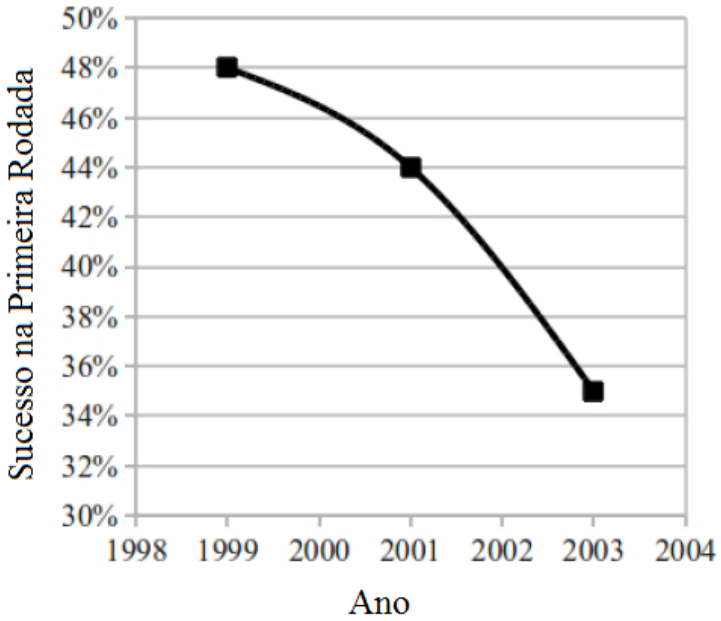


Figura: Pesquisa Collet International Research Inc., 2003

Motivação e Justificativa

Desenvolver sistemas de alta complexidade se tornou um desafio devido à dificuldade em se manter a conformidade entre os requisitos iniciais e o sistema desenvolvido [Tobar 2010].

Motivos para verificar:

- Aumento das falhas em silício
- Evitar gastos excedentes
- Utilização de IP
- Reputação

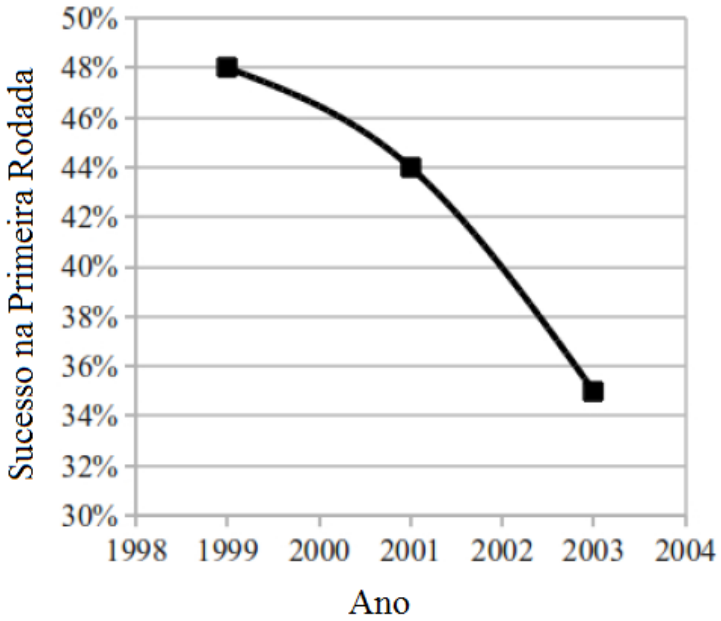


Figura: Pesquisa Collet International Research Inc., 2003

Motivação e Justificativa

Desenvolver sistemas de alta complexidade se tornou um desafio devido à dificuldade em se manter a conformidade entre os requisitos iniciais e o sistema desenvolvido [Tobar 2010].

Motivos para verificar:

- Aumento das falhas em silício
- Evitar gastos excedentes
- Utilização de IP
- Reputação
- Atender prazos de mercado

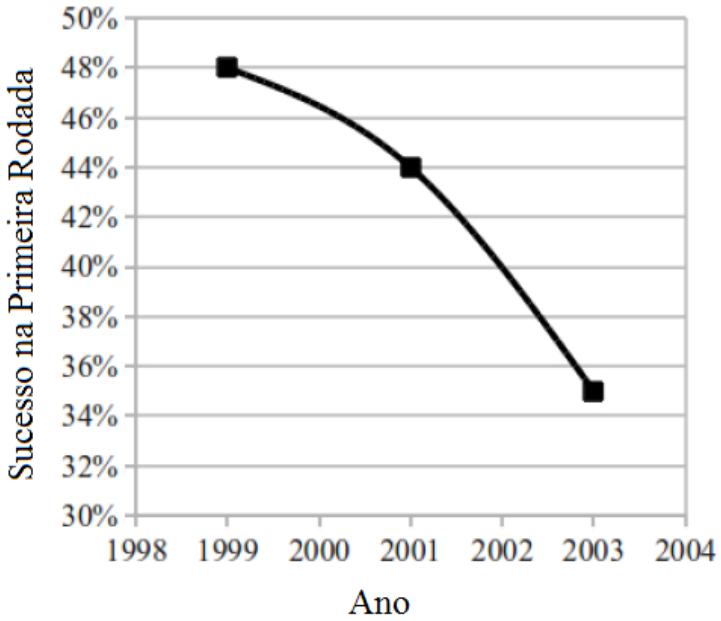


Figura: Pesquisa Collet International Research Inc., 2003

Motivação e Justificativa

Desenvolver sistemas de alta complexidade se tornou um desafio devido à dificuldade em se manter a conformidade entre os requisitos iniciais e o sistema desenvolvido [Tobar 2010].

Motivos para verificar:

- Aumento das falhas em silício
- Atender prazos de mercado
- Utilização de IP
- Reputação
- Evitar gastos excedentes
- *Gap* na verificação

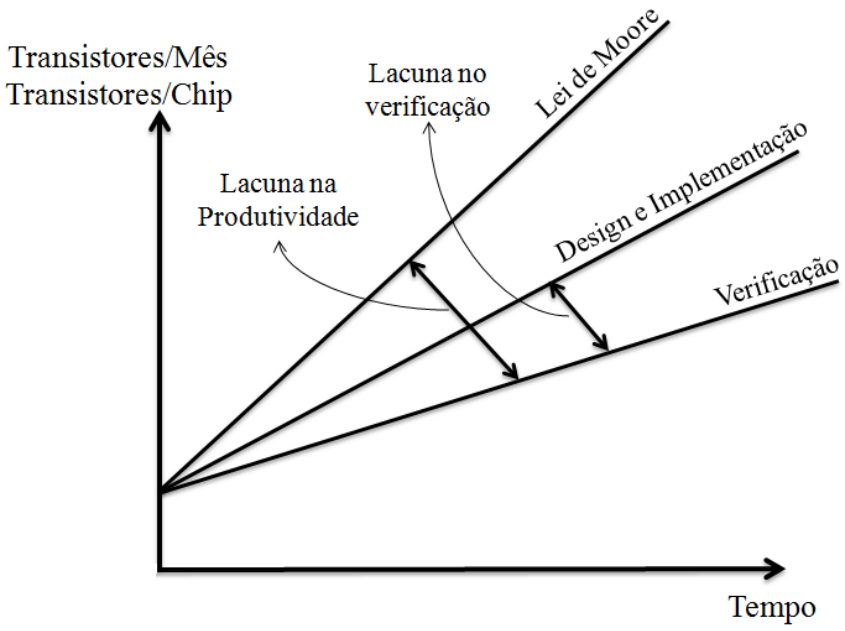


Figura: Lacuna de *design* e verificação

Motivação e Justificativa

Desenvolver sistemas de alta complexidade se tornou um desafio devido à dificuldade em se manter a conformidade entre os requisitos iniciais e o sistema desenvolvido [Tobar 2010].

Motivos para verificar:

- Aumento das falhas em silício
- Atender prazos de mercado
- Utilização de IP
- Reputação
- Evitar gastos excedentes
- *Gap* na verificação
- 70% das falhas são lógicas

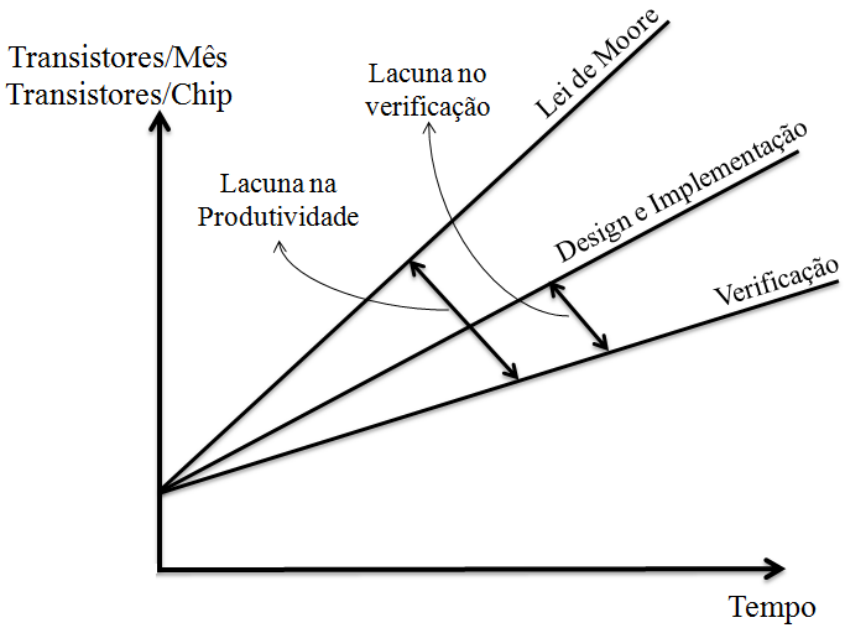


Figura: Lacuna de *design* e verificação

Pesquisas

Pesquisa Far West Research e Mentor Graphics, 2007.

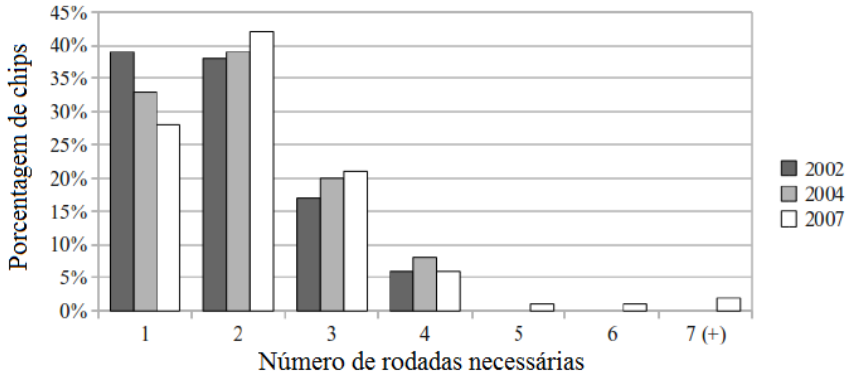


Figura: Aumento na quantidade de rodadas para obter-se sucesso

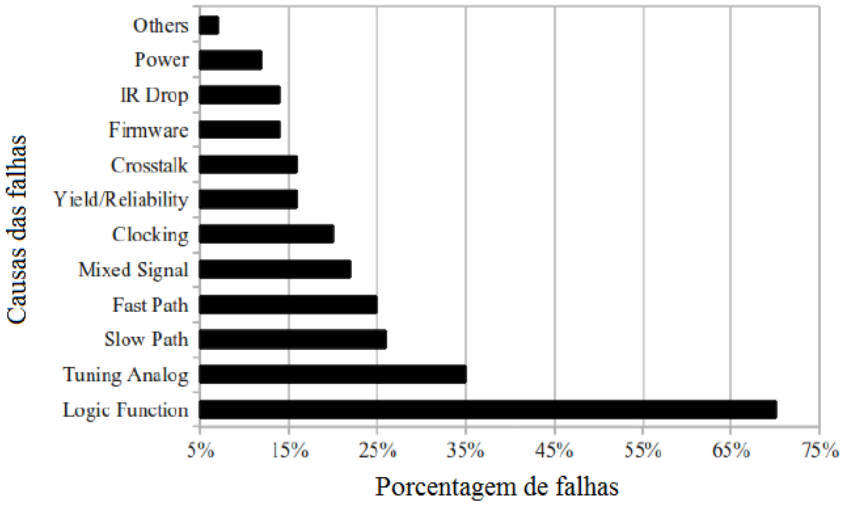


Figura: Causas que levam a falhas em silício e suas ocorrências

O que é Verificação

- Processo utilizado para demonstrar a corretude de um determinado projeto.

O que é Verificação

- Processo utilizado para demonstrar a corretude de um determinado projeto.
- Afirma se o que está sendo verificado implementa as funcionalidades desejadas

O que é Verificação

- Processo utilizado para demonstrar a corretude de um determinado projeto.
- Afirma se o que está sendo verificado implementa as funcionalidades desejadas

Como verificar?

Basicamente, através de códigos que:

- Estimulam as entradas DUT

O que é Verificação

- Processo utilizado para demonstrar a corretude de um determinado projeto.
- Afirma se o que está sendo verificado implementa as funcionalidades desejadas

Como verificar?

Basicamente, através de códigos que:

- Estimulam as entradas DUT
- Monitoram as saídas do DUT

O que é Verificação

- Processo utilizado para demonstrar a corretude de um determinado projeto.
- Afirma se o que está sendo verificado implementa as funcionalidades desejadas

Como verificar?

Basicamente, através de códigos que:

- Estimulam as entradas DUT
- Monitoram as saídas do DUT
- Validam os resultados obtidos

O que é Verificação

- Processo utilizado para demonstrar a corretude de um determinado projeto.
- Afirma se o que está sendo verificado implementa as funcionalidades desejadas

Como verificar?

Basicamente, através de códigos que:

- Estimulam as entradas DUT
- Monitoram as saídas do DUT
- Validam os resultados obtidos

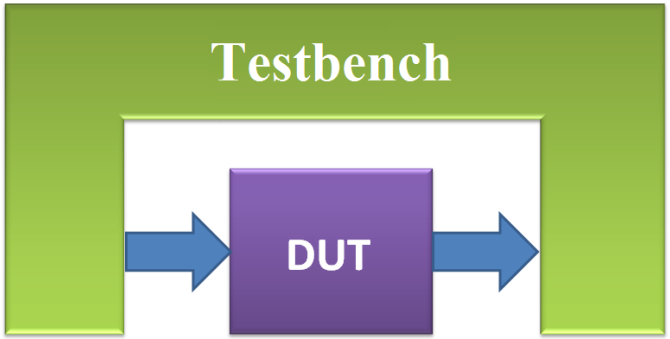


Figura: Representação de *testbench*

Este conjunto de códigos é denominado *Testbench*.

Métricas de Cobertura

São ferramentas importantes para avaliar e medir o progresso na etapa de verificação funcional.

Métricas de Cobertura

São ferramentas importantes para avaliar e medir o progresso na etapa de verificação funcional.

Tipos

- Cobertura de Código
  - Cobertura de Declarações, Ramos, Condições, Eventos, Mudanças.

Métricas de Cobertura

São ferramentas importantes para avaliar e medir o progresso na etapa de verificação funcional.

Tipos

- Cobertura de Código
  - Cobertura de Declarações, Ramos, Condições, Eventos, Mudanças.
- Cobertura de Asserções

Métricas de Cobertura

São ferramentas importantes para avaliar e medir o progresso na etapa de verificação funcional.

Tipos

- Cobertura de Código
  - Cobertura de Declarações, Ramos, Condições, Eventos, Mudanças.
- Cobertura de Asserções
- Cobertura Funcional

Métricas de Cobertura

São ferramentas importantes para avaliar e medir o progresso na etapa de verificação funcional.

Tipos

- Cobertura de Código
  - Cobertura de Declarações, Ramos, Condições, Eventos, Mudanças.
- Cobertura de Asserções
- Cobertura Funcional
  - Todos os cenários típicos foram cobertos?

Métricas de Cobertura

São ferramentas importantes para avaliar e medir o progresso na etapa de verificação funcional.

Tipos

- Cobertura de Código
  - Cobertura de Declarações, Ramos, Condições, Eventos, Mudanças.
- Cobertura de Asserções
- Cobertura Funcional
  - Todos os cenários típicos foram cobertos?
  - Foram testados cenários que geram erros?

Métricas de Cobertura

São ferramentas importantes para avaliar e medir o progresso na etapa de verificação funcional.

Tipos

- Cobertura de Código
  - Cobertura de Declarações, Ramos, Condições, Eventos, Mudanças.
- Cobertura de Asserções
- Cobertura Funcional
  - Todos os cenários típicos foram cobertos?
  - Foram testados cenários que geram erros?
  - Quais são os casos críticos do projeto?

Métricas de Cobertura

São ferramentas importantes para avaliar e medir o progresso na etapa de verificação funcional.

Tipos

- Cobertura de Código
  - Cobertura de Declarações, Ramos, Condições, Eventos, Mudanças.
- Cobertura de Asserções
- Cobertura Funcional
  - Todos os cenários típicos foram cobertos?
  - Foram testados cenários que geram erros?
  - Quais são os casos críticos do projeto?
  - Como considerar protocolos?

Processo de Verificação

Etapas do processo:

1. Plano de Verificação

Processo de Verificação

Etapas do processo:

1. Plano de Verificação
   - Aspectos de verificação

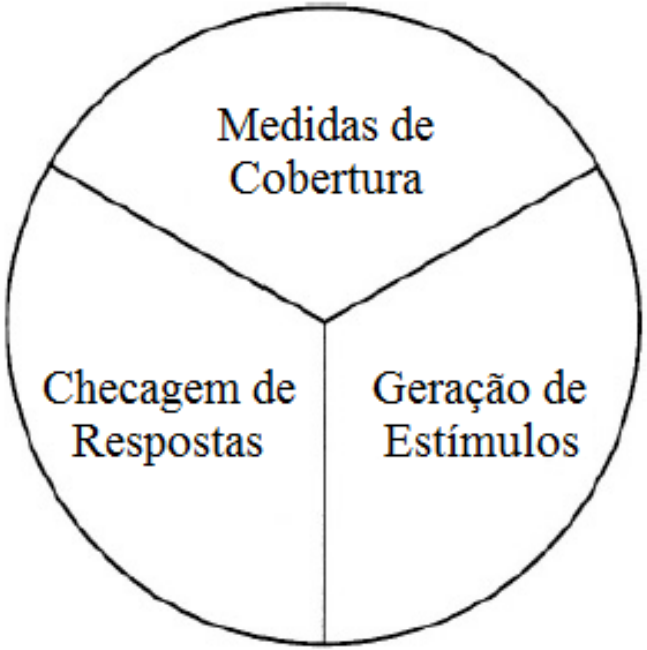


Figura: Aspectos da Verificação Funcional

Processo de Verificação

Etapas do processo:

1. Plano de Verificação
   - Aspectos de verificação
   - Requisitos de verificação

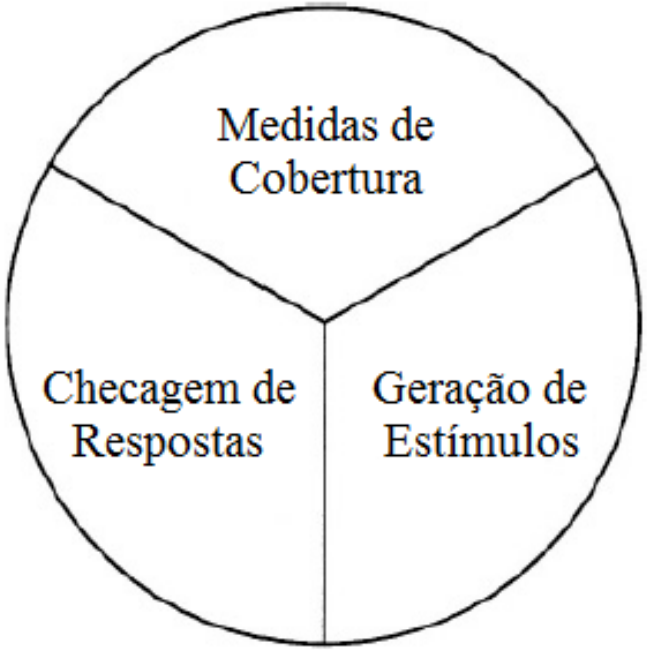


Figura: Aspectos da Verificação Funcional

Processo de Verificação

Etapas do processo:

1. Plano de Verificação
   - Aspectos de verificação
   - Requisitos de verificação
   - *O que* verificar e *como* verificar

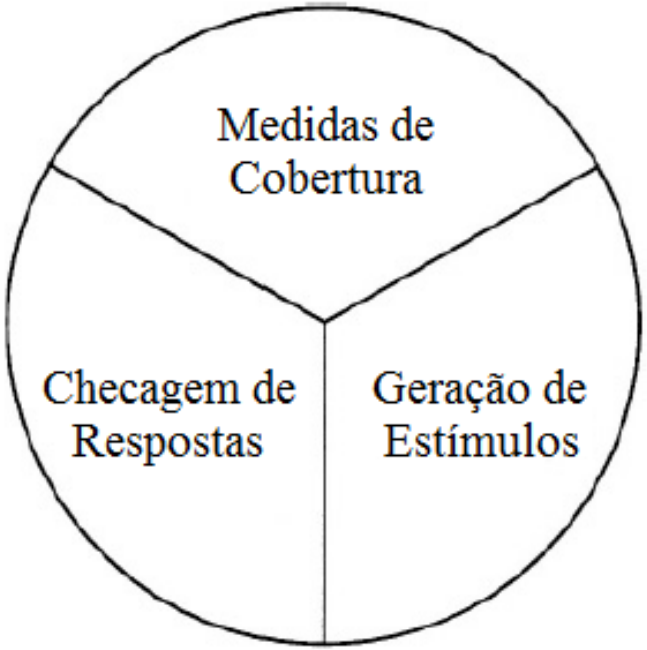


Figura: Aspectos da Verificação Funcional

Processo de Verificação

Etapas do processo:

1. Plano de Verificação
   - Aspectos de verificação
   - Requisitos de verificação
   - *O que* verificar e *como* verificar
2. Implementação do testbench

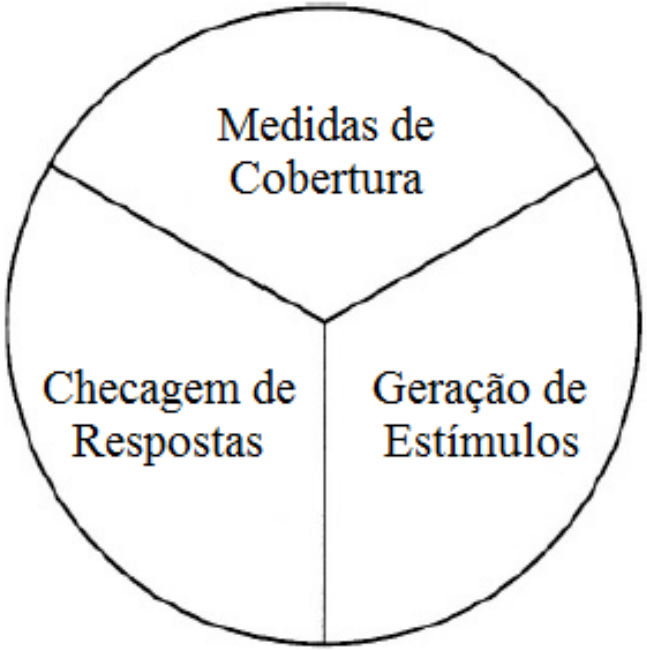


Figura: Aspectos da Verificação Funcional

Processo de Verificação

Etapas do processo:

1. Plano de Verificação
   - Aspectos de verificação
   - Requisitos de verificação
   - *O que* verificar e *como* verificar
2. Implementação do testbench
3. *Bring-Up*

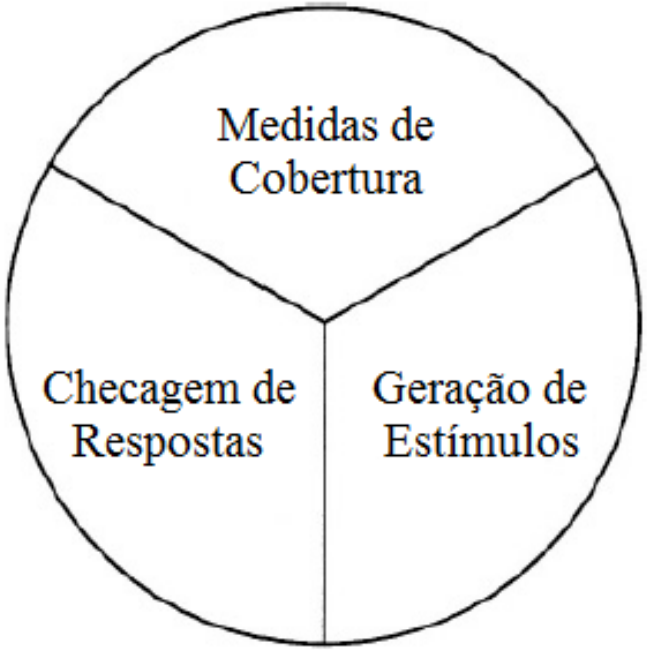


Figura: Aspectos da Verificação Funcional

Processo de Verificação

Etapas do processo:

1. Plano de Verificação
   - Aspectos de verificação
   - Requisitos de verificação
   - *O que* verificar e *como* verificar
2. Implementação do testbench
3. *Bring-Up*
4. Regressão

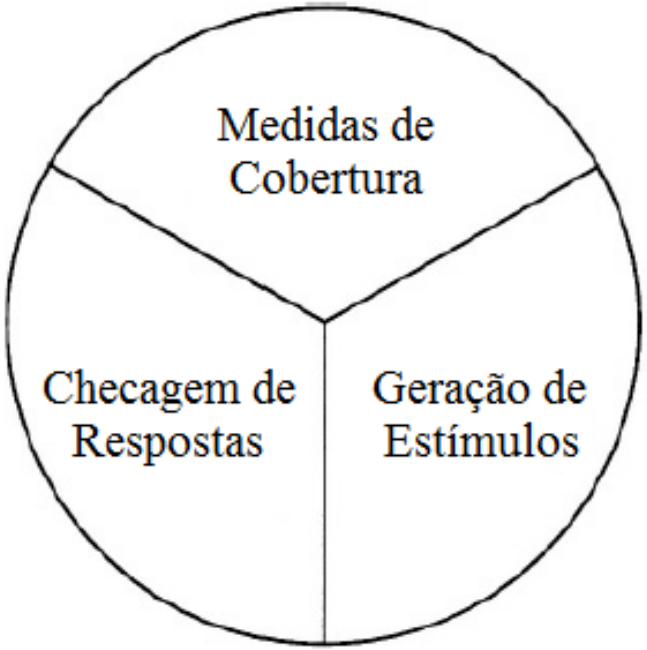


Figura: Aspectos da Verificação Funcional

Modelagem de Sistemas

ESL - Electronic-System Level

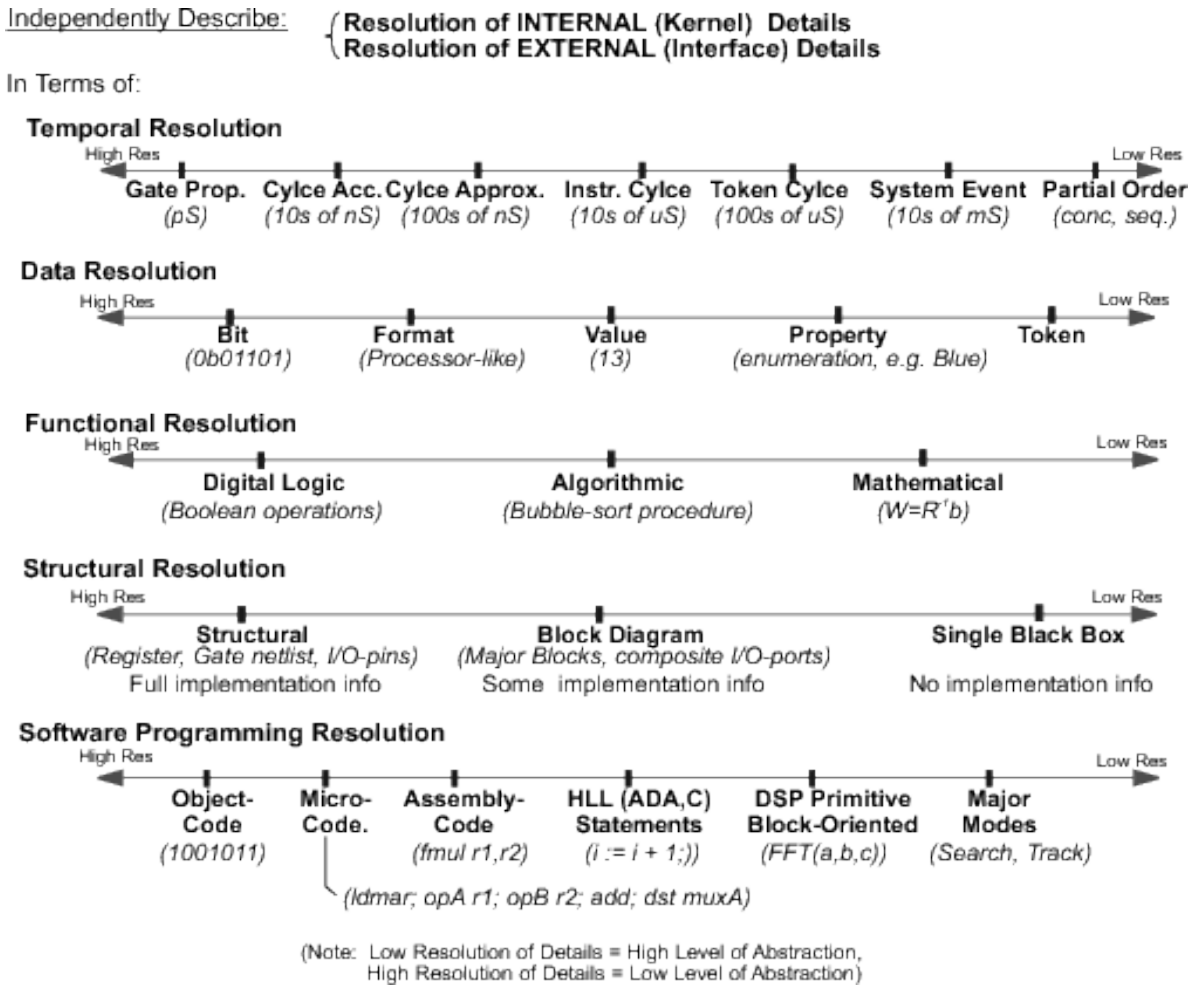


Figura: 5 eixos de abstração do ESL

Modelagem de Sistemas

TLM - Transaction Level Modeling

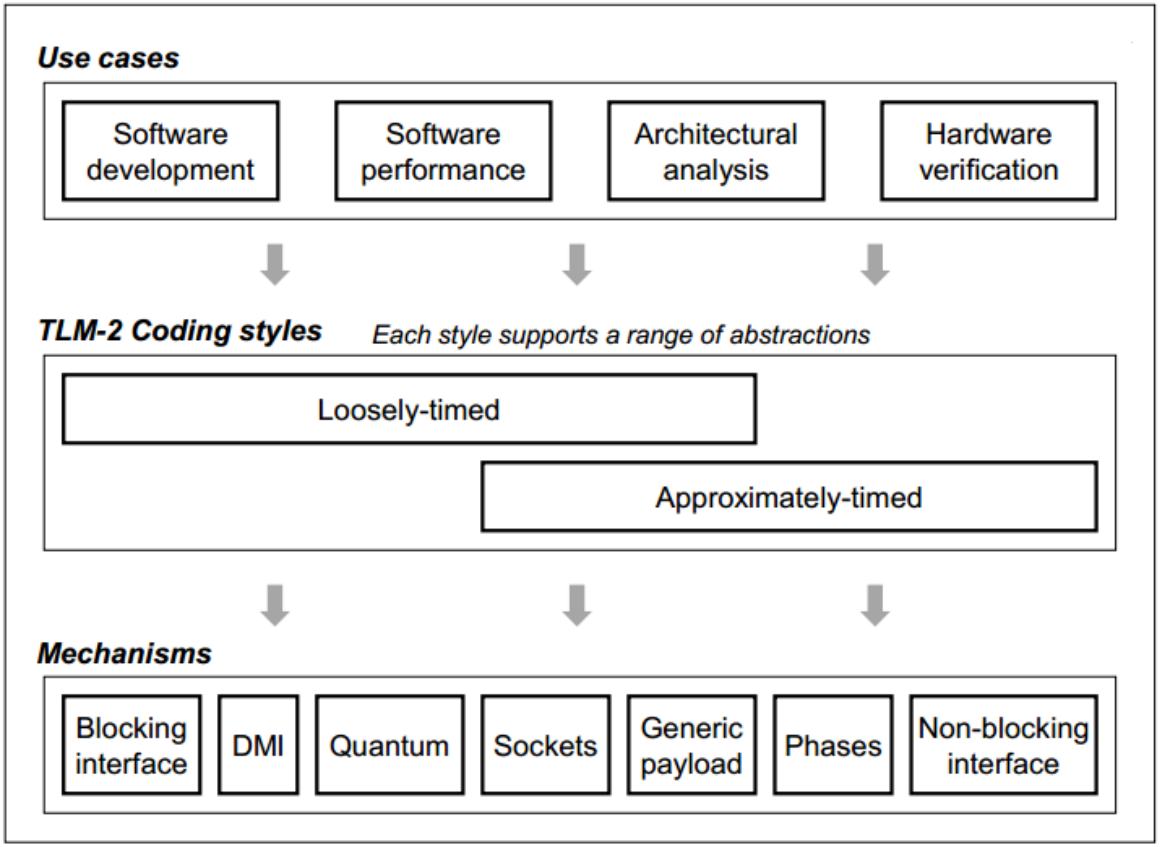


Figura: Aplicações e mecanismos segundo o estilo de código.

Metodologias de Verificação

Em pouco tempo, será alcançado um ponto onde será impossível verificar a exatidão de um sistema sem introduzir uma metodologia de verificação no processo de design [Sangiovanni-Vincentelli, McGeer e Saldanha 1996].

Metodologias de Verificação

Em pouco tempo, será alcançado um ponto onde será impossível verificar a exatidão de um sistema sem introduzir uma metodologia de verificação no processo de design [Sangiovanni-Vincentelli, McGeer e Saldanha 1996].

Componentes Gerais:

- *Sequencer*

Metodologias de Verificação

Em pouco tempo, será alcançado um ponto onde será impossível verificar a exatidão de um sistema sem introduzir uma metodologia de verificação no processo de design [Sangiovanni-Vincentelli, McGeer e Saldanha 1996].

Componentes Gerais:

- *Sequencer*
- *Driver*

Metodologias de Verificação

Em pouco tempo, será alcançado um ponto onde será impossível verificar a exatidão de um sistema sem introduzir uma metodologia de verificação no processo de design [Sangiovanni-Vincentelli, McGeer e Saldanha 1996].

Componentes Gerais:

- *Sequencer*
- *Driver*
- *Monitor*

Metodologias de Verificação

Em pouco tempo, será alcançado um ponto onde será impossível verificar a exatidão de um sistema sem introduzir uma metodologia de verificação no processo de design [Sangiovanni-Vincentelli, McGeer e Saldanha 1996].

Componentes Gerais:

- *Sequencer*
- *Driver*
- *Monitor*
- *Scoreboard*

Metodologias de Verificação

Em pouco tempo, será alcançado um ponto onde será impossível verificar a exatidão de um sistema sem introduzir uma metodologia de verificação no processo de design [Sangiovanni-Vincentelli, McGeer e Saldanha 1996].

Componentes Gerais:

- *Sequencer*
- *Driver*
- *Monitor*
- *Scoreboard*

Outros Componentes:

- *Agent*

Metodologias de Verificação

Em pouco tempo, será alcançado um ponto onde será impossível verificar a exatidão de um sistema sem introduzir uma metodologia de verificação no processo de design [Sangiovanni-Vincentelli, McGeer e Saldanha 1996].

Componentes Gerais:

- *Sequencer*
- *Driver*
- *Monitor*
- *Scoreboard*

Outros Componentes:

- *Agent*
- *Actor*

Metodologias Existentes

- CDV - *Coverage-Driven Verification*

Metodologias Existentes

- CDV - *Coverage-Driven Verification*
  - Plano de verificação muito bem elaborado
  - Geração automática de testes
  - Auto-checagem
  - Combina vários tipos de métricas

Metodologias Existentes

- CDV - *Coverage-Driven Verification*
  - Plano de verificação muito bem elaborado
  - Geração automática de testes
  - Auto-checagem
  - Combina vários tipos de métricas

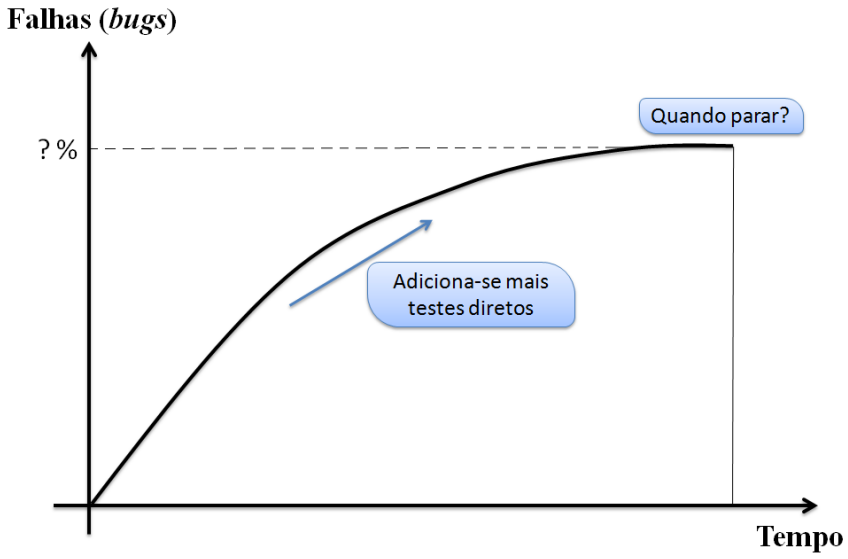


Figura: Simulação

Metodologias Existentes

- CDV - *Coverage-Driven Verification*
  - Plano de verificação muito bem elaborado
  - Geração automática de testes
  - Auto-checagem
  - Combina vários tipos de métricas

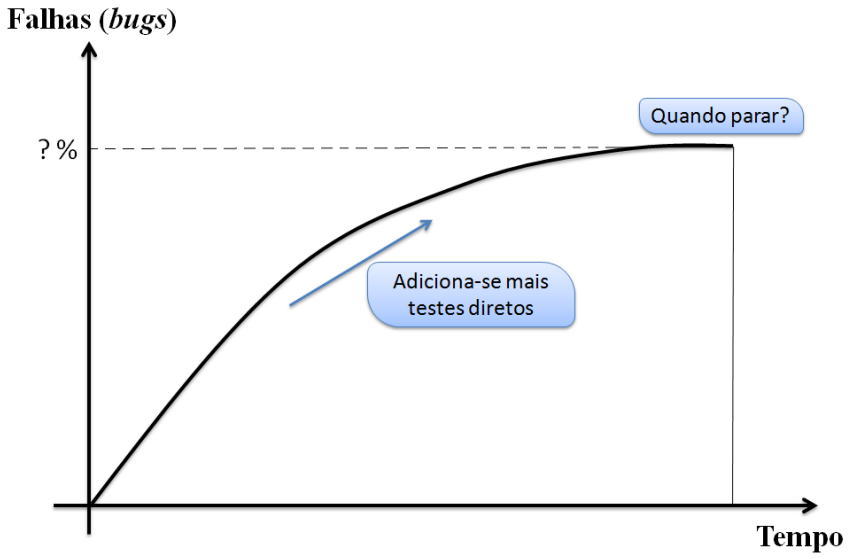


Figura: Simulação

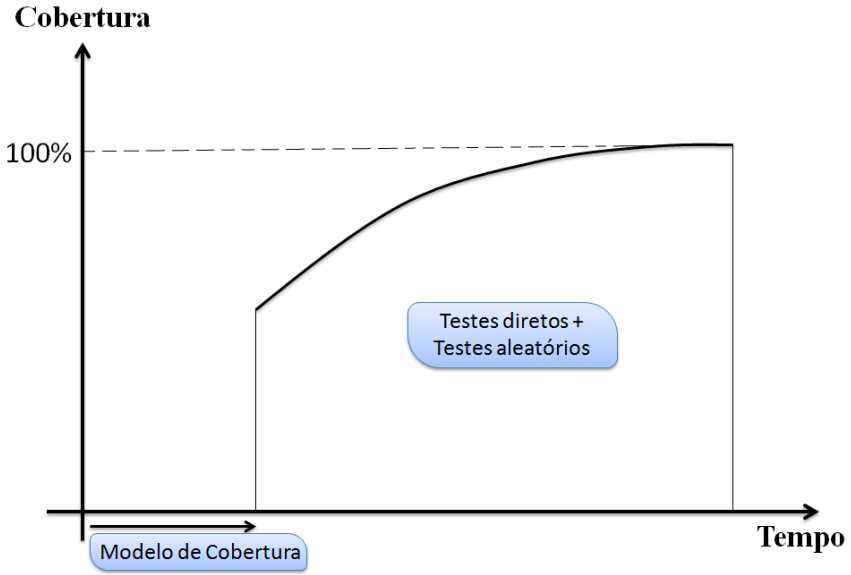


Figura: CDV

Metodologias Existentes

- VeriSC

Metodologias Existentes

- VeriSC
- BVM

Metodologias Existentes

- VeriSC
- BVM
- *e*RM

Metodologias Existentes

- VeriSC
- BVM
- *e*RM
- VMM

Metodologias Existentes

- VeriSC
- BVM
- *e*RM
- VMM
- OVM

Metodologias Existentes

- VeriSC
- BVM
- *e*RM
- VMM
- OVM
- UVM

Metodologias Existentes

- VeriSC
- BVM
- *e*RM
- VMM
- OVM
- UVM

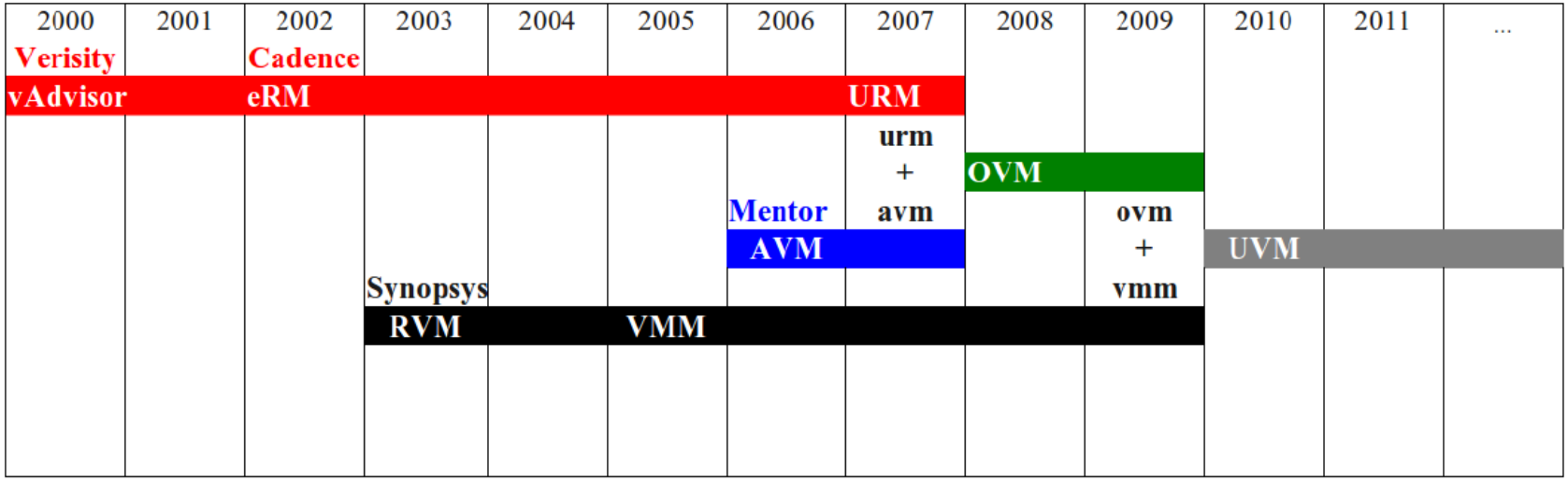


Figura: Metodologias ao longo do tempo

UVM - Universal Verification Methodology

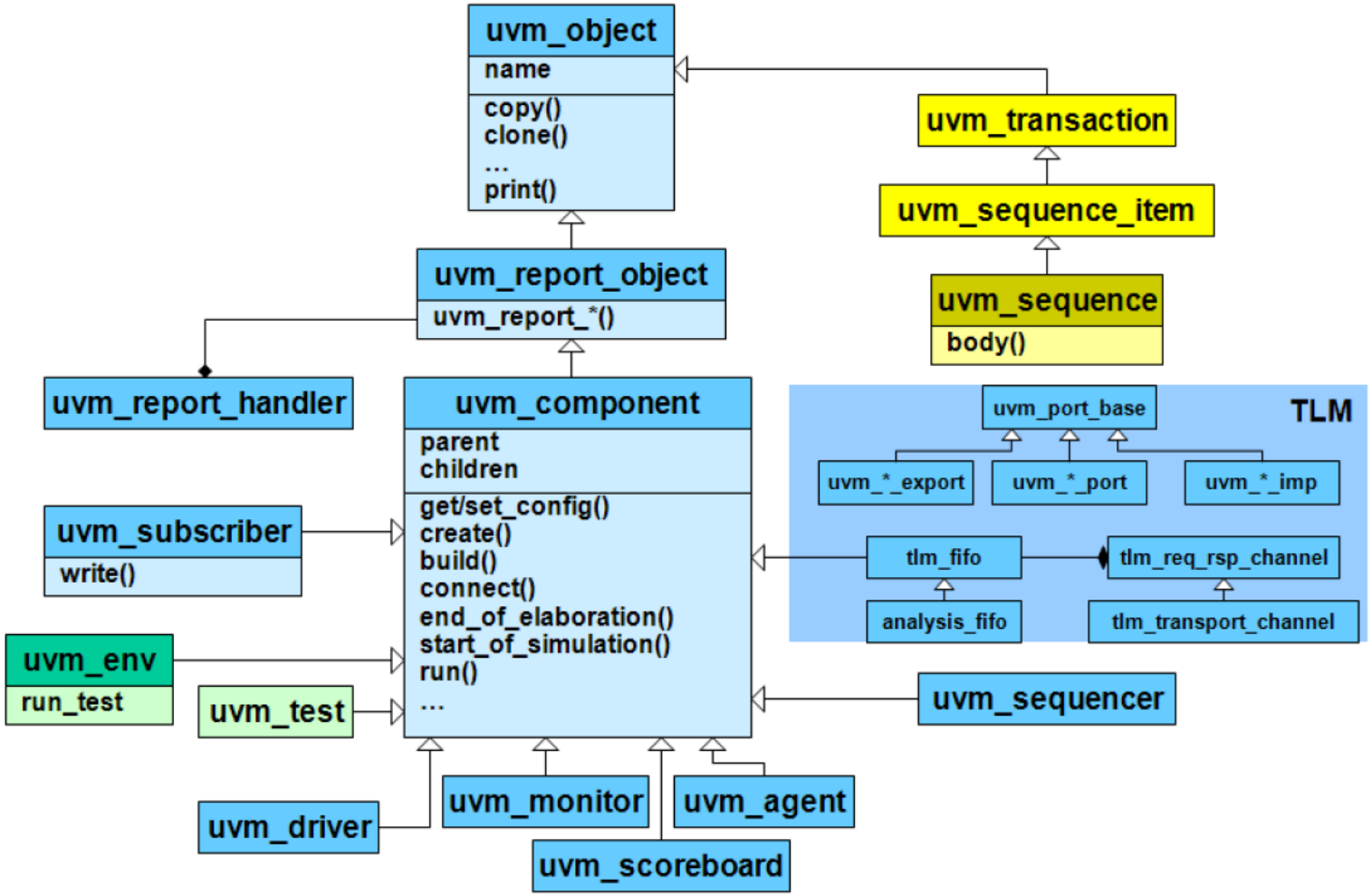


Figura: Hierarquia de classes UVM (parcial)

Exemplo

SystemVerilog - Main

```
import uvm_pkg::*;
import uvmc_pkg::*;

`include "producer.sv"

module sv_main;

  producer prod = new("prod");

  initial begin
    uvmc_tlm #()::connect(prod.out, "foo");
    run_test();
  end

endmodule
```

SystemC - Main

```
#include "uvmc.h"
using namespace uvmc;

#include "consumer.h"

int sc_main(int argc, char* argv[])
{
  consumer cons("cons");
  uvmc_connect(cons.in,"foo");
  sc_start();
  return 0;
}
```

Exemplo

SystemVerilog - Producer

```
import uvm_pkg::*;
`include "uvm_macros.svh"

class producer extends uvm_component;
    uvm_tlm_b_initiator_socket #() out;
    `uvm_component_utils(producer)

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

    task run_phase (uvm_phase phase);
        uvm_tlm_gp gp = new;
        uvm_tlm_time delay = new("del",1e-12);

        phase.raise_objection(this);
        ...
        out.b_transport(gp,delay);
        ...
        phase.drop_objection(this);
    endtask

endclass
```

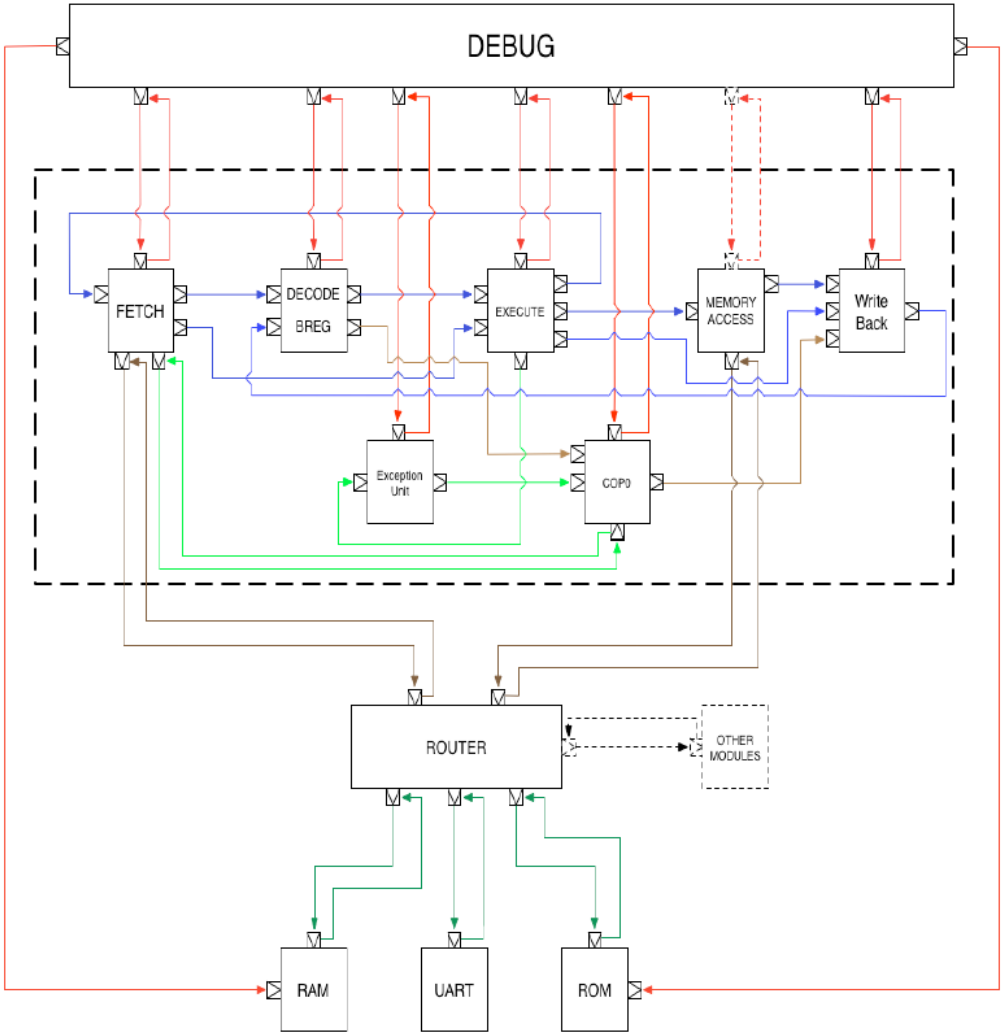
Exemplo

SystemC - Consumer

```
#include <systemc.h>
#include <tlm.h>
using namespace sc_core;
using namespace tlm;

#include "simple_target_socket.h"
using tlm_utils::simple_target_socket;

class consumer : public sc_module
{
  public:
  simple_target_socket<consumer> in;

  consumer(sc_module_name nm) : in("in")
  {
    in.register_b_transport(this, &consumer::b_transport);
  }

  virtual void b_transport(tlm_generic_payload &gp, sc_time &t) {
    char unsigned *data;
    int len;

    len = gp.get_data_length();
    data = gp.get_data_ptr();
    ...
    wait(t);
    t = SC_ZERO_TIME;
  }
};
```

MIPS32 em SystemC TLM-2.0

Diagrama de Blocos Implementados



Obrigado!