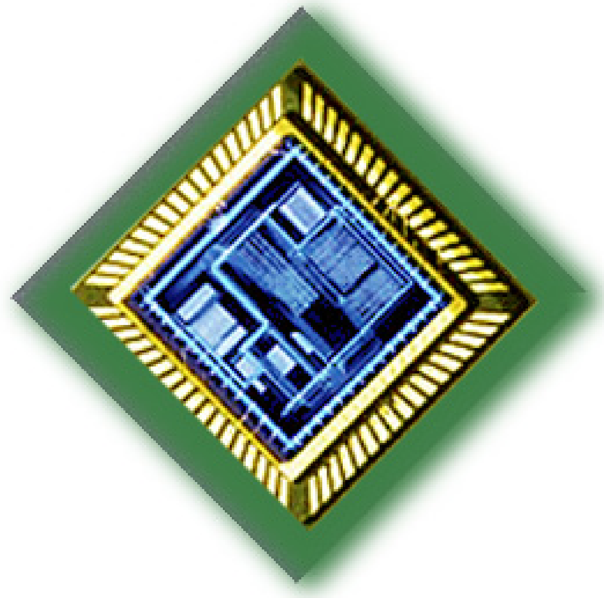


Rumo a UVM

Curso do Brazil-IP

Felipe Gonçalves Assis
UFCG
fgassis@lad.dsc.ufcg.edu.br

Roteiro

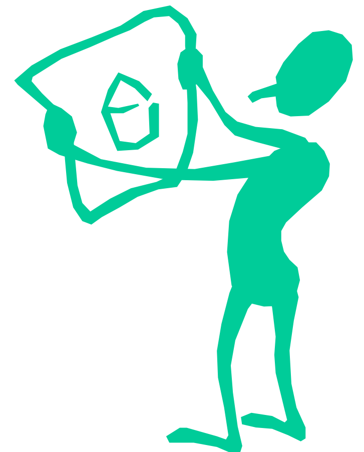
1)Introdução

2)Mudanças básicas

3)Tests e Environments

4)Factory

5)Seqüências



Introdução

- Mais que portar nosso código para uma nova biblioteca.
- Usar novos recursos possibilitados pela biblioteca.
    - Vários já existiam em OVM.
- Aproximar-se da *metodologia* UVM.

Mudanças básicas

- O vira U.
- tlm_*fifo vira uvm_tlm_*fifo.
- ovm_transaction vira uvm_sequence_item.
- task run() vira

  task run_phase(uvm_phase phase).

Tests e Environment

- Environment instancia todos os blocos do testbench e os conecta.
- Test instancia o environment e o configura.
- TB apenas instancia DUV e interfaces e chama run_test().
- Linha de comando seleciona o test a ser executado com a opção +UVM_TESTNAME=*test_name*.

Entrando em fase

- Em vez de instanciar e conectar tudo no construtor, recomenda-se usar as funções build_phase e run_phase.
- Útil para seguir padrões.
- Essencial para configurabilidade.

O environment

```
class my_env extends uvm_env;
   `uvm_components_utils(my_env)
   ...
   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction

   function void connect_phase(uvm_phase phase);
      ...

   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      ...
endclass
```

Test simples

```
class basic_test extends uvm_test;
   `uvm_components_utils(basic_test)

   my_env env;

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

   function void build_phase(uvm_phase phase);
      ...

   task run_phase(uvm_phase phase);
      ...
endclass
```

Test simples

```
class basic_test extends uvm_test;
   `uvm_components_utils(basic_test)

   my_env env;

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

   ...
endclass
```

Test simples

```
class basic_test extends uvm_test;
   `uvm_components_utils(basic_test)

   my_env env;

   ...

   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      env = new(“env”, this);
   endfunction

   ...
endclass
```

Test simples

```
class basic_test extends uvm_test;
   `uvm_components_utils(basic_test)

   my_env env;

   ...

   task run_phase(uvm_phase phase);
      phase.raise_objection(this);
      #1ms phase.drop_objection(this);
   endtask
endclass
```

TB sem DUV

```
module tb();
   initial
      run_test();
endmodule
```

Linha de comando:

```
$ irun -uvm tb.sv +UVM_TESTNAME=basic_test
```

Ligando o testbench ao DUV

```
module tb();
   ...
   my_if my_rif(.*);
   duv m_duv(my_rif);
   initial begin
      uvm_resource_db#(virtual my_if)::set(
         “ifs”, “duv_if”, my_rif);
      run_test();
endmodule
```

Ligando o testbench ao DUV

```
class my_env extends uvm_env;
   ...
   virtual my_if my_vif;
   function void build_phase(uvm_phase phase);
      ...
      assert(
         uvm_resource_db#(virtual my_if)
   ::read_by_name(“ifs”, “duv_if”, my_vif) );

      uvm_config_db#(virtual my_if)
  ::set(this, “*”, “my_vif”, my_vif);
      ...
```

Ligando o testbench ao DUV

```
class my_if_driver extends uvm_component;
   ...
   virtual my_if my_vif;
   function void build_phase(uvm_phase phase);
      ...
      assert(
         uvm_config_db#(virtual my_if)
  ::get(this, “”, “my_vif”, my_vif) );
      ...
```

Factory

- Velho padrão de programação.
- Um dos meios de reconfiguração do testbench.

Registro na factory

```
class driver extends uvm_component;
   `uvm_component_utils(driver)
   ...
endclass


class packet extends uvm_sequence_item;
   `uvm_object_utils(packet)
   ...
endclass
```

Registro na factory

```
class driver #(type T = int) extends
  uvm_component;
   `uvm_component_param_utils(driver#(T))
   ...
endclass

class packet #(WIDTH = 4) extends
  uvm_sequence_item;
   `uvm_object_param_utils(packet#(WIDTH))
   ...
endclass
```

Uso da factory

```
driver_h = new("driver_h", this);
```

vira

```
driver_h =
  driver::type_id::create("driver_h",
  this);
```

```
(Vale para componentes e transações.
Não vale para covergroups e FIFOs.)
```

Mudando de classe

- Só pode descer na hierarquia.
- Pode trocar a classe de objetos específicos ou de todos os objetos de uma dada classe

```
driver::type_id::set_inst_override(
   driverxx::get_type(), “path”, this);

driver::type_id::set_type_override(
   driverxx::get_type());
```

Aplicação: Inserindo falhas no RefMod

```
class refmod extends uvm_component;
   `uvm_component_utils(refmod)
   ...
   virtual function int compute();
      ...
endclass
```

Aplicação: Inserindo falhas no RefMod

```
class tweaked_refmod extends refmod;
   `uvm_component_utils(tweaked_refmod)
   ...
   virtual function int compute();
      // something very bad.
      ...
endclass
```

Aplicação: Inserindo falhas no RefMod

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

   my_env env;

   ...
   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      refmod::type_id::set_inst_override(
         tweaked_refmod::get_type(), "env.refmod2",
   this);
      ...
endclass
```

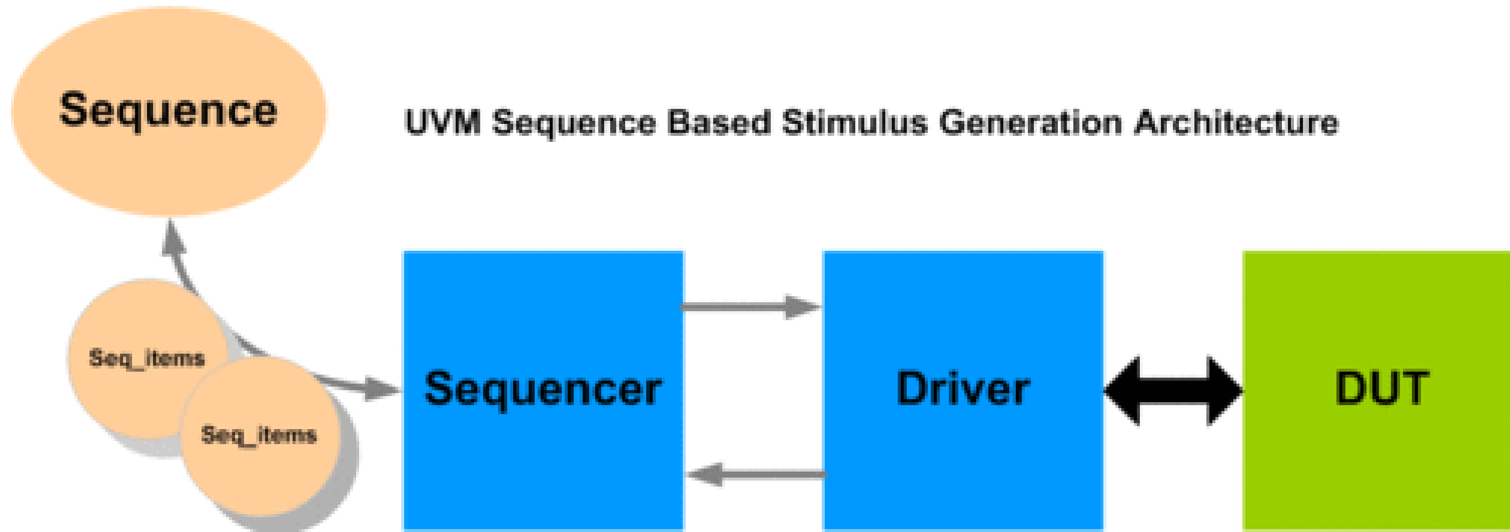
```
$ irun -uvm tb.sv
  +UVM_TESTNAME=bad_refmod_test
```

Seqüências

- Estímulo é uma ou mais seqüências de transações, tipicamente aleatórias.
- Pode ser importante que as transações não sejam i.i.d, ou mesmo que haja dependência entre seqüências em paralelo.
  - Exemplo: *Read after Write*.
- Solução:
  - uvm_sequence;
  - uvm_sequencer;
  - uvm_virtual_sequence.

Seqüenciadores

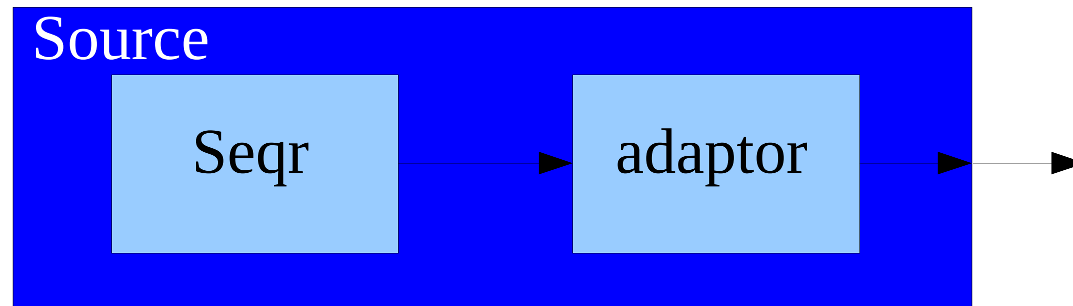
- Seqüenciador roda seqüência.
- Comunicação bidirecional com driver por interface específica.



Fonte: http://verificationacademy.com (checar copie-direito)

Uma gambiarra

- Para deixar o *Sequencer* da UVM parecido com o nosso velho Source, colocamo-lhe um adaptador na frente.
  - Arquivo sequencer_source_adaptor.svh.
  - Perda da bidirecionalidade.
- Embrulhamos tudo.



Escrevendo seqüências

```
class my_seq extends uvm_sequence#(packet);
   `uvm_object_utils(my_seq)

   function new (string name = ””);
      super.new(name);
   endfunction

   task body;
      ...
   endtask
endclass
```

Escrevendo seqüências

```
task body;
    packet tx;
    forever begin
        tx =
packet::type_id::create("tx");
        start_item(tx);
        assert( tx.randomize() );
        finish_item(tx);
    end
endtask
```

Seqüencias em seqüências

```
task body;
   my_seq seq;
   forever begin
      seq =
         my_seq::type_id::create("seq");
      seq.start(m_sequencer, this);
      ...
   end
endtask
```

Rodando uma seqüência

```
class basic_test extends uvm_test;

   ...

   task run_phase(uvm_phase phase);
      my_seq seq;
      ...
      seq = my_seq::type_id::create("seq");
      seq.start(env.source_h.sequencer_h);
   endtask
endclass
```

Mais sobre seqüências

- Coordenando seqüências em paralelo com seqüências virtuais.
- Usando respostas do driver para gerar seqüência.
- Múltiplas seqüências num mesmo seqüenciador.
- E muito mais!

- Conferir referências :)

Referências

http://verificationacademy.com/course-modules/uvm-ovm-verification/basic-uvm-universal-verification-methodology.