[**SIEMENS**](https://verificationacademy.com/)西门子

[AHomeC](https://verificationacademy.com/)[Verification Methodology CookbooksC](https://verificationacademy.com/cookbook)[UVM](https://verificationacademy.com/cookbook/uvm-universal-verification-methodology)

UVM性能指南

尽管UVM提高了验证效率，但仍存在某些问题。

当涉及到性能和可伸缩性考虑时，应该谨慎使用方法的某些方面，或者

[6](https://verificationacademy.com/topic/11e76ebf-1af0-3396-9193-00c43f615912/toggle-marked?page_path=cookbook/uvm-universal-verification-methodology/uvm-performance-guidelines)

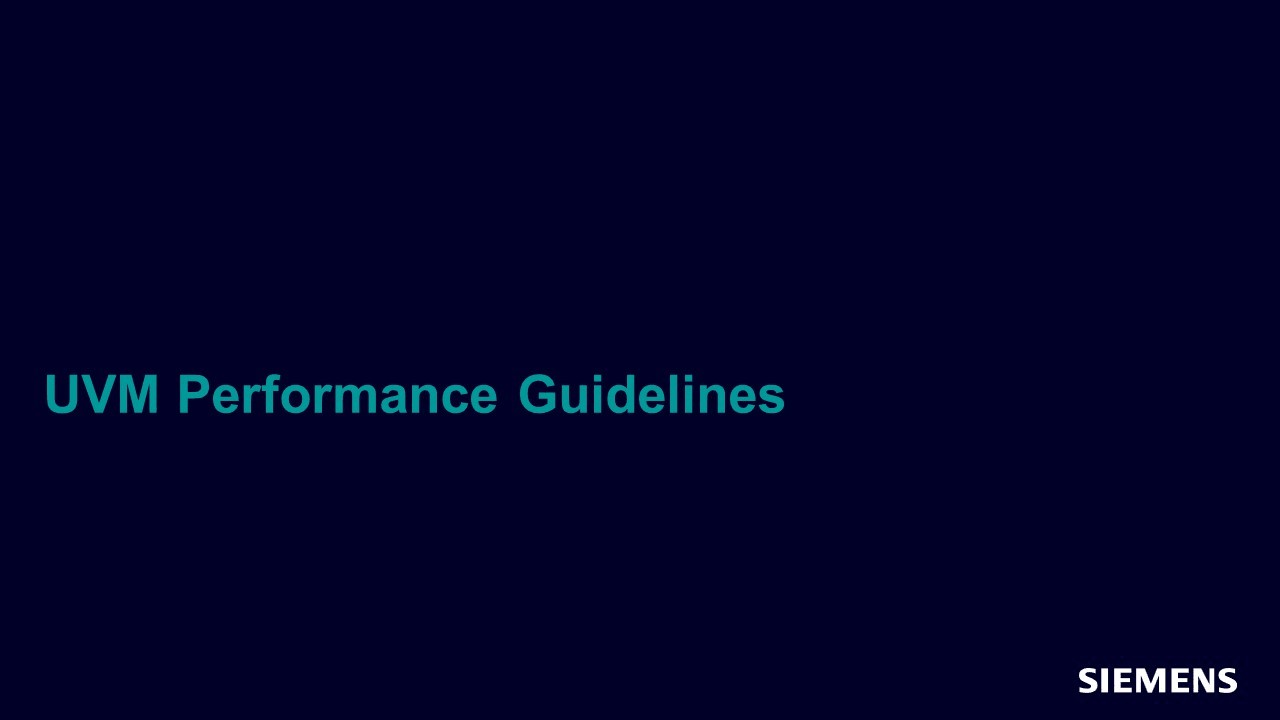
[UVM- 通用验证方法](https://verificationacademy.com/topics/uvm-universal-verification-methodology/)

核查**方法小组**

最**后更新于2014年3月**

O[**UVM**](https://verificationacademy.com/all-content?tag=UVM)**，**[**SystemVerilog**](https://verificationacademy.com/all-content?tag=SystemVerilog)**，**[**附录**](https://verificationacademy.com/all-content?tag=Appendix)**，**[**指南**](https://verificationacademy.com/all-content?tag=Guidelines)**，**[**性能**](https://verificationacademy.com/all-content?tag=Performance)**，**[**初学者**](https://verificationacademy.com/all-content?audience=Beginner)

[Q**标记查看**](https://verificationacademy.com/3accb135-e461-3e39-a31b-0a5fcf56278e/toggle-marked?page_path=cookbook/uvm-universal-verification-methodology/uvm-performance-guidelines)





在UVM测试台的模拟运行期间，有两个不同的活动周期。

第一个是UVM阶段的集合，它与配置、构建和连接测试平台组件层次结构有关，第二个是运行时活动，其中所有的激励都是

进行分析活动。两个活动期间的性能考虑是分开的。

这些业绩准则应与其他方法一并阅读

cookbook准则，在有些情况下，需要判断权衡性能、重用和可伸缩性问题。

UVM测试台配置和构建性能指南

UVM测试台配置和构建过程的一般要求是，

应该很快，以便运行时阶段可以开始。对于仅包含少数组件的小型测试台-例如当测试台的组件超过10哥组价，构建过程应该很短，

对于具有100个或可能1000个组件的较大测试台，构建阶段将明显较慢。本节中的指南适用于UVM测试台尺寸的全部范围

避免自动注册

自动配置是一种继承自OVM的方法，

配置变量自动设置为使用set\_config\_int()、set\_config\_string()、uvm\_config\_db #(..)：：set()等在组件层次结构的更高级别。

为了使用自动配置，在组件中使用字段宏，并且需要在build\_phase()期间调用super.build

然后，配置进程尝试通过uvm\_component中名为apply\_config\_settings()的方法将组件中的字段与配置数据库从性能的角度来看，这是非常昂贵的，而且不能扩展。

### 低性能版

class my\_env extends uvm\_component;

bit has\_axi\_agent;

bit has\_ahb\_agent;

string system\_name;

axi\_agent m\_axi\_agent;

ahb\_agent m\_ahb\_agent;

// Required for auto-configuration

`uvm\_component\_utils\_begin(my\_env)

`uvm\_field\_int(has\_axi\_agent, UVM\_DEFAULT)

`uvm\_field\_int(has\_ahb\_agent, UVM\_DEFAULT)

`uvm\_field\_string(system\_name, UVM\_DEFAULT)

`uvm\_component\_utils\_end

function new(string name = "my\_env", uvm\_component parent = null);

super.new(name, parent);

endfunction

function void build\_phase(uvm\_phase phase);

super.build\_phase(phase); // Auto-configuration called here

if(has\_axi\_agent == 1) begin

m\_axi\_agent = axi\_agent::type\_id::create("m\_axi\_agent", this);

end

if(has\_ahb\_agent == 1) begin

m\_ahb\_agent = ahb\_agent::type\_id::create("m\_ahb\_agent", this);

end

`uvm\_info("build\_phase", $sformatf("%s built", system\_name))

endfunction: build\_phase

endclass: my\_env

更高性能版本

class my\_env extends uvm\_component;

my\_env\_config cfg;

axi\_agent m\_axi\_agent;

ahb\_agent m\_ahb\_agent;

`uvm\_component\_utils(my\_env)

function new(string name = "my\_env", uvm\_component parent = null);

super.new(name, parent);

endfunction

function void build\_phase(uvm\_phase phase);

// Get the configuration, note class variables not required

if(!uvm\_config\_db #(my\_env\_config)::get(this, "", "my\_env\_config", cfg)) begin

`uvm\_error("build\_phase", "Unable to find my\_env\_config in uvm\_config\_db")

end

if(cfg.has\_axi\_agent == 1) begin

m\_axi\_agent = axi\_agent::type\_id::create("m\_axi\_agent", this);

end

if(cfg.has\_ahb\_agent == 1) begin

m\_ahb\_agent = ahb\_agent::type\_id::create("m\_ahb\_agent", this);

end

`uvm\_info("build\_phase", $sformatf("%s built", cfg.system\_name))

endfunction: build\_phase

endclass: my\_env

建议的做法是不要在组件中使用字段宏，也不要调用super.build\_phase()如果您要扩展的类来自UVM组件基类(例如uvm\_component)。即使这样，当组件没有build\_phase()方法时，

实现时，将调用uvm\_component基类中的默认build\_phase()

在UVM 1.1b中，添加了一个修复程序，如果组件中没有字段宏，则停止apply\_config\_settings()方法继续

尽量减少使用uvm\_config\_db

uvm\_config\_db是一个数据库，与任何数据库一样，随着它的大小增加，搜索时间会更长uvm\_config\_db基于uvm\_resource和uvm\_resource\_db类。uvm\_resource\_db使用正则表达式和组件层次结构字符串进行匹配，它尝试检查每个可能的匹配，然后返回最接近的匹配搜索是昂贵的，并且搜索时间随着数据库的增长而呈指数增长。

因此，如果要使用uvm\_config\_db的话，应该少用这也适用于set/get\_config\_xxx()方法，因为它们反过来又基于uvm\_config\_db。

使用配置对象将配置数据传递给组件

最小化uvm\_config\_db条目数量的一种方法是对组件进行分组配置变量到配置对象中。这样只需要在uvm\_config\_db中设置一个对象这具有重用的好处，并且是配置可重复使用的验证组件，如代理。

较低性能版本

class static\_test extends uvm\_test;

// Test that builds an env containing an AXI agent

virtual axi\_if AXI; // Used by the AXI agent

// Only consider the build method:

function void build\_phase(uvm\_phase phase);

// Configuration code for the AXI agent

if(!uvm\_config\_db #(virtual axi\_if)::get(this, "", "AXI", AXI)) begin

`uvm\_error("build\_phase", "AXI vif not found in uvm\_config\_db")

end

uvm\_config\_db #(virtual axi\_if)::set(this, "env.axi\_agent\*", "v\_if", AXI);

uvm\_config\_db #(uvm\_active\_passive\_enum)::set(

this, "env.axi\_agent\*", "is\_active", UVM\_ACTIVE);

uvm\_config\_db #(int)::set(this, "env.axi\_agent\*", "max\_burst\_size", 16);

// Other code

endfunction: build\_phase

endclass: static\_test

// The AXI agent:

class axi\_agent extends uvm\_component;

// Configuration parameters:

virtual axi\_if AXI;

uvm\_active\_passive\_enum is\_active;

int max\_burst\_size;

axi\_driver driver;

axi\_sequencer sequencer;

axi\_monitor monitor;

function void build\_phase(uvm\_phase phase);

if(!uvm\_config\_db #(virtual axi\_if)::get(this, "", "AXI", AXI)) begin

`uvm\_error("build\_phase", "AXI vif not found in uvm\_config\_db")

end

if(!uvm\_config\_db #(uvm\_active\_passive\_enum)::get(this,"", "is\_active", is\_active)) begin

`uvm\_error("build\_phase", "is\_active not found in uvm\_config\_db")

end

if(!uvm\_config\_db #(int)::get(this, "", "max\_burst\_size", max\_burst\_size)) begin

`uvm\_error("build\_phase", "max\_burst\_size not found in uvm\_config\_db")

end

monitor = axi\_monitor::type\_id::create("monitor", this);

if(is\_active == UVM\_ACTIVE) begin

driver = axi\_driver::type\_id::create("driver", this);

sequencer = axi\_sequencer::type\_id::create("sequencer", this);

end

endfunction: build\_phase

function void connect\_phase(uvm\_phase phase);

monitor.AXI = AXI;

if(is\_active == UVM\_ACTIVE) begin

driver.AXI = AXI;

driver.max\_burst\_size = max\_burst\_size;

end

endfunction: connect\_phase

endclass: axi\_agent

更高性能版本

// Additional agent configuration class:

class axi\_agent\_config extends uvm\_object;

`uvm\_object\_utils(axi\_agent\_config)

virtual axi\_if AXI;

uvm\_active\_passive\_enum is\_active = UVM\_ACTIVE;

int max\_burst\_size = 64;

function new(string name = "axi\_agent\_config");

super.new(name);

endfunction

endclass: axi\_agent\_config

class static\_test extends uvm\_test;

// Test that builds an env containing an AXI agent

axi\_agent\_config axi\_cfg; // Used by the AXI agent

// Only consider the build method:

function void build\_phase(uvm\_phase phase);

// Configuration code for the AXI agent

axi\_cfg = axi\_agent\_config::type\_id::create("axi\_cfg");

if(!uvm\_config\_db #(virtual axi\_if)::get(this, "", "AXI", axi\_cfg.AXI)) begin

`uvm\_error("build\_phase", "AXI vif not found in uvm\_config\_db")

end

axi\_cfg.is\_active = UVM\_ACTIVE;

axi\_cfg.max\_burst\_size = 16;

uvm\_config\_db #(axi\_agent\_config)::set(this,"env.axi\_agent\*", "axi\_agent\_config", axi\_cfg);

// Other code

endfunction: build\_phase

endclass: static\_test

// The AXI agent:

class axi\_agent extends uvm\_component;

// Configuration object:

axi\_agent\_config cfg;

axi\_driver driver;

axi\_sequencer sequencer;

axi\_monitor monitor;

function void build\_phase(uvm\_phase phase);

if(!uvm\_config\_db #(axi\_agent\_config)::get(this,"", "axi\_agent\_config", cfg)) begin

`uvm\_error("build\_phase", "AXI agent config object not found in uvm\_config\_db")

end

monitor = axi\_monitor::type\_id::create("monitor", this);

if(cfg.is\_active == UVM\_ACTIVE) begin

driver = axi\_driver::type\_id::create("driver", this);

sequencer = axi\_sequencer::type\_id::create("sequencer", this);

end

endfunction: build\_phase

function void connect\_phase(uvm\_phase phase);

monitor.AXI = cfg.AXI;

if(cfg.is\_active == UVM\_ACTIVE) begin

driver.AXI = cfg.AXI;

driver.max\_burst\_size = cfg.max\_burst\_size;

end

endfunction: connect\_phase

endclass: axi\_agent

该示例的更高性能版本使用一个uvm\_config\_db #(...)：：set()调用，两次get()调用，而较低性能版本中有三次set()和四次get()调用。

uvm\_config\_db条目也只有两个，而不是四个。对于大量的组件，这种形式的优化可以带来相当大的性能提升。

最小化uvm\_config\_db #(...)：：get()调用

从uvm\_config\_db执行get()的过程开销很大，只有在真正需要的时候才应该例如，在一个代理中，只需要获取()配置对象，并将句柄分配给那里在驱动程序和监视器组件内部进行单独的get()调用是不必要的开销。

较低性能版本

// Agent configuration class - configured and set() by the test class

class axi\_agent\_config extends uvm\_object;

`uvm\_object\_utils(axi\_agent\_config)

virtual axi\_if AXI;

uvm\_active\_passive\_enum is\_active = UVM\_ACTIVE;

int max\_burst\_size = 64;

function new(string name = "axi\_agent\_config");

super.new(name);

endfunction

endclass: axi\_agent\_config

// The AXI agent:

class axi\_agent extends uvm\_component;

// Configuration object:

axi\_agent\_config cfg;

axi\_driver driver;

axi\_sequencer sequencer;

axi\_monitor monitor;

function void build\_phase(uvm\_phase phase);

if(!uvm\_config\_db #(axi\_agent\_config)::get(this, "", "axi\_agent\_config", cfg)) begin

`uvm\_error("build\_phase", "AXI agent config object not found in uvm\_config\_db") end

monitor = axi\_monitor::type\_id::create("monitor", this);

if(cfg.is\_active == UVM\_ACTIVE) begin

driver = axi\_driver::type\_id::create("driver", this);

sequencer = axi\_sequencer::type\_id::create("sequencer", this);

end

endfunction: build\_phase

endclass: axi\_agent

// The axi monitor:

class axi\_monitor extends uvm\_component;

axi\_if AXI;

axi\_agent\_config cfg;

function void build\_phase(uvm\_phase phase);

if(!uvm\_config\_db #(axi\_agent\_config)::get(this, "", "axi\_agent\_config", cfg)) begin

`uvm\_error("build\_phase", "AXI agent config object not found in uvm\_config\_db")

end

AXI = cfg.AXI;

endfunction: build\_phase

endclass: axi\_monitor

// The axi driver:

class axi\_monitor extends uvm\_component;

axi\_if AXI;

int max\_burst\_size;

axi\_agent\_config cfg;

function void build\_phase(uvm\_phase phase);

if(!uvm\_config\_db #(axi\_agent\_config)::get(this, "", "axi\_agent\_config", cfg)) begin

`uvm\_error("build\_phase", "AXI agent config object not found in uvm\_config\_db")

end

AXI = cfg.AXI;

max\_burst\_size = cfg.max\_burst\_size;

endfunction: build\_phase

endclass: axi\_driver

更高性能版本

// The AXI agent:

class axi\_agent extends uvm\_component;

// Configuration object:

axi\_agent\_config cfg;

axi\_driver driver;

axi\_sequencer sequencer;

axi\_monitor monitor;

function void build\_phase(uvm\_phase phase);

if(!uvm\_config\_db #(axi\_agent\_config)::get(this,"", "axi\_agent\_config", cfg)) begin

`uvm\_error("build\_phase", "AXI agent config object not found in uvm\_config\_db")

end

monitor = axi\_monitor::type\_id::create("monitor", this);

if(cfg.is\_active == UVM\_ACTIVE) begin

driver = axi\_driver::type\_id::create("driver", this);

sequencer = axi\_sequencer::type\_id::create("sequencer", this);

end

endfunction: build\_phase

// Direct assignment to the monitor and driver variables

// from the configuration object variables.

function void connect\_phase(uvm\_phase phase);

monitor.AXI = cfg.AXI;

if(cfg.is\_active == UVM\_ACTIVE) begin

driver.AXI = cfg.AXI;

driver.max\_burst\_size = cfg.max\_burst\_size;

end

endfunction: connect\_phase

endclass: axi\_agent

// The axi\_monitor and axi\_driver are implemented without

// a uvm\_config\_db #()::get()

性能更高的版本对uvm\_config\_db #()：：get()方法的调用减少了两次

改进.

在uvm\_config\_db set()和get()调用中使用特定的字符串

搜索中使用的正则表达式算法试图根据UVM组件在测试平台层次结构中的位置和键字符串的值来获得最接近的匹配如果

在set()或get()过程中使用通配符，这会增加搜索的模糊性

例如，将上下文字符串设置为“\*”意味着在返回结果之前，将搜索整个组件层次结构

性能版

// In one component, setting config for env.sb

sb\_cfg = sb\_config::type\_id::create("sb\_cfg");

// Configure content of sb\_cfg ...

umv\_config\_db #(sb\_config)::set(this, "\*", "\*\_config", sb\_cfg);

// In the env.sb component:

sb\_config cfg;

if(!uvm\_config\_db #(sb\_config)::get(this, "", "\*\_config", cfg)) begin

`uvm\_error(...)

end

更高性能版本

// In one component, setting config for env.sb

sb\_cfg = sb\_config::type\_id::create("sb\_cfg");

// Configure content of sb\_cfg ...

umv\_config\_db #(sb\_config)::set(this, "env.sb", "sb\_config", sb\_cfg);

// In the env.sb component:

sb\_config cfg;

if(!uvm\_config\_db #(sb\_config)::get(this, "", "sb\_config", cfg)) begin

`uvm\_error(...)

end

在这段代码的高性能版本中，作用域非常具体，并且只在单个组件上匹配单个键，这减少了在uvm\_config\_db中的搜索时间

最大限度地减少通过uvm\_config\_db从TB模块传递到UVM环境的虚拟接口句柄的数量

将虚拟接口句柄合并到单个配置对象中

使用包将虚拟接口句柄从testbench顶级模块传递到UVM测试的另一种虚拟接口句柄，并在uvm\_config\_db中设置配置对象之前，在顶级模块中进行虚拟接口分配这将用于传递虚拟接口句柄的uvm\_config\_db条目的数量减少到一个。

// Virtual interface configuration object:

class vif\_handles extends uvm\_object;

`uvm\_object\_utils(vif\_handles)

virtual axi\_if AXI;

virtual ddr2\_if DDR2;

endclass: vif\_handles

// In the top level testbench module:

module top\_tb;

import uvm\_pkg::\*;

import test\_pkg::\*;

// Instantiate the static interfaces:

axi\_if AXI();

ddr2\_if DDR2();

// Virtual interface handle container object:

vif\_handles v\_h;

// Hook up to DUT ....

// UVM initial block:

initial begin

// Create virtual interface handle container:

v\_h = vif\_handles::type\_id::create("v\_h");

// Assign handles

v\_h.AXI = AXI;

v\_h.DDR2 = DDR2;

// Set in uvm\_config\_db:

uvm\_config\_db #(vif\_handles)::set("uvm\_test\_top", "", "V\_H", vh);

run\_test();

end

endmodule: top\_tb

通过类层次结构引用传递配置信息

最大限度地减少uvm\_config\_db的使用的最终方法是根本不使用它在构建时通过类层次结构将句柄传递给配置对象是使用句柄赋值是最有效的方法。

较低性能版本

// In the test, with the env configuration object containing

// nested configuration objects for its agents:

function void build\_phase(uvm\_phase phase);

env\_cfg = env\_config\_object::type\_id::create("env\_cfg");

// Populate the env\_cfg object with axi\_cfg, ddr2\_cfg etc

env = test\_env::type\_id::create("env");

uvm\_config\_db #(env\_config\_object)::set(this, "env", "env\_cfg", env\_cfg);

// ...

endfunction: build\_phase

// In the env, building the axi and ddr2 agents:

env\_config\_object cfg;

function void build\_phase(uvm\_phase phase);

if(!uvm\_config\_db #(env\_config\_object)::get(this, "", "env\_cfg", cfg)) begin

`uvm\_error(...)

end

// Create AXI agent and set configuration for it:

axi = axi\_agent::type\_id::create("axi", this);

uvm\_config\_db #(axi\_agent\_config)::set(this, "axi", "axi\_agent\_config", cfg.axi\_cfg);

// Also for the DDR2 agent:

ddr2 = ddr2\_agent::type\_id::create("ddr2", this);

uvm\_config\_db #(ddr2\_agent\_config)::set(this,"ddr2", "ddr2\_agent\_config", cfg.ddr2\_cfg);

// etc

endfunction: build\_phase

更高性能版本

// In the test, with the env configuration object containing

// nested configuration objects for its agents:

function void build\_phase(uvm\_phase phase);

env\_cfg = env\_config\_object::type\_id::create("env\_cfg");

// Populate the env\_cfg object with axi\_cfg, ddr2\_cfg etc

env = test\_env::type\_id::create("env");

// Assign the env configuration object handle directly:

env.cfg = env\_cfg;

// ...

endfunction: build\_phase

// In the env, building the axi and ddr2 agents:

env\_config\_object cfg;

function void build\_phase(uvm\_phase phase);

// Create AXI agent and set configuration for it:

axi = axi\_agent::type\_id::create("axi", this);

// Assign axi agents configuration handle directly:

axi.cfg = cfg.axi\_cfg;

// Also for the DDR2 agent:

ddr2 = ddr2\_agent::type\_id::create("ddr2", this);

ddr2.cfg = cfg.ddr2\_cfg;

// etc

endfunction: build\_phase

更高性能的示例完全避免了使用uvm\_config\_db，提供了最终的配置和构建性能增强。使用这个的影响一种方法是，它要求将赋值链接在一起;它要求代理代码在尝试获取配置对象句柄之前测试空配置对象句柄;并且任何激励层次结构都需要负责将句柄获取到testbench

资源，如注册模型。

这种直接分配配置对象句柄的方法的另一个主要考虑因素是，如果VIP被重用，那么它很可能在实现时期望其配置对象将在uvm\_config\_db中设置这意味着可能必须使用uvm\_config\_db来支持现有VIP的重用

尽量减少使用UVM工厂

UVM工厂允许UVM组件或对象被派生对象覆盖这是一项功能强大的技术，但无论何时构建组件，

以确定要构造的对象类型如果存在覆盖，则此查找将变得更加复杂，并且存在性能损失。尝试管理用于减少查找开销的工厂重写。

UVM测试台运行时性能指南

这里介绍的指南代表了UVM的领域，这些领域被认为会导致

在测试平台的运行时间阶段的性能问题，他们主要关注刺激的产生。还有其他SystemVerilog编码实践，

来增强运行时性能，[SystemVerilog性能指南](https://verificationacademy.com/cookbook/uvm-universal-verification-methodology/uvm-performance-guidelines/cookbook/uvm-universal-verification-methodology/systemverilog-performance-guidelines)一文中对此进行了描述

避免轮询uvm\_config\_db以获取更改

不要使用uvm\_config\_db在testbench的不同部分之间进行通信，例如，在一个组件中设置一个新的变量值，并将其放入另一个组件的轮询循环这两个组件拥有一个公共对象的句柄并引用对象中变量的值要有效得多

较低性能版本

// In a producer component setting the value inside a loop:

int current\_id = 0;

forever begin

// Lots of work making a transfer occur

// Communicate the current id:

uvm\_config\_db #(int)::set(null, "\*", "current\_id", current\_id);

current\_id++;

end

// In a consumer component looking out for the current\_id value

int current\_id;

forever begin

uvm\_config\_db #(int)::wait\_modified(this, "\*", "current\_id");

if(!uvm\_config\_db #(int)::get(this, "", "current\_id", current\_id)) begin

`uvm\_error( ....)

end

// Lots of work to track down a transaction with the current\_id

End

更高性能版本

// Config object containing current\_id field:

packet\_info\_cfg pkt\_info =

packet\_info\_cfg::type\_id::create("pkt\_info");

// This created in the producer component and the consumer component

// has a handle to the object:

// In the producer component:

forever begin

// The work resulting in a current\_id update

pkt\_info.current\_id = current\_id;

current\_id++;

end

// In the consumer component:

forever begin

@(pkt\_info.current\_id);

// Start working with the new id

End

更高性能版本的工作原理是current\_id信息位于对象内部。使用者组件和生产者组件共享相同的对象，因此当生产者对象对current\_id字段进行更改时，它通过句柄对消费者组件可见这避免了在uvm\_config\_db中重复使用set()和get()调用，也避免了使用昂贵的wait\_modified()方法。

不要在事务中使用UVM宏

UVM字段宏似乎是确保实现各种do\_copy()、do\_compare()方法的一种方便方法

性能 如果您的测试平台开始大量使用sequence\_items，这一点就变得非常明显。

较低性能版本

// APB Bus sequence\_item

class apb\_seq\_item extends uvm\_sequence\_item;

bit[31:0] addr;

bit[31:0] data;

apb\_opcode\_e we;

// Field macros:

`uvm\_object\_utils\_begin(apb\_seq\_item)

`uvm\_field\_int(addr, UVM\_DEFAULT)

`uvm\_field\_int(data, UVM\_DEFAULT)

`uvm\_field\_enum(we, apb\_opcode\_e, UVM\_DEFAULT)

`uvm\_object\_utils\_end

function new(string name = "apb\_seq\_item");

super.new(name);

endfunction

endclass: apb\_seq\_item

更高性能版本

// APB Bus sequence\_item

class apb\_seq\_item extends uvm\_sequence\_item;

bit[31:0] addr;

bit[31:0] data;

apb\_opcode\_e we;

`uvm\_object\_utils(apb\_seq\_item)

function new(string name = "apb\_seq\_item");

super.new(name);

endfunction

// Sequence Item convenience method prototypes:

extern function void do\_copy(uvm\_object rhs);

extern function bit do\_compare(uvm\_object rhs, uvm\_comparer comparer);

extern function string convert2string();

extern function void do\_print(uvm\_printer printer);

extern function void do\_record(uvm\_recorder recorder);

extern function void do\_pack();

extern function void do\_unpack();

endclass: apb\_seq\_item

尽管性能较低的代码示例看起来更紧凑，但使用-eprety标志进行编译更高性能的示例显示了各种uvm\_object便利方法的模板，手动实现，这将始终提高性能，并在需要时增强调试

关于使用或不使用这些和其他各种UVM宏的权衡的权威指南

最大限度地减少刺激对象的工厂覆盖

UVM工厂可用于覆盖或更改在调用对象句柄的：：type\_id：：create()方法时创建的对象类型在刺激产生期间，这可能是应用于改变序列或sequence\_item的行为，而不重写测试台代码。然而，这种覆盖能力是以扩展的

在每次创建对象时在工厂中查找若要减少创建在工厂中覆盖的对象的影响，请创建一次对象，然后在每次使用时克隆它，以避免使用工厂。

### 低性能版

低

// apb\_seq\_item has been factory overridden with apb\_seq\_error\_item

class reg\_bash\_seq extends uvm\_sequence #(apb\_seq\_item);

task body;

apb\_seq\_item item;

repeat(200) begin

item = apb\_seq\_item::type\_id::create("item");

start\_item(item);

assert(item.randomize() with {addr inside {[`reg\_low:`reg\_high]};});

finish\_item(item);

end

endtask:body

endclass: reg\_bash\_seq

更高性能版本

// apb\_seq\_item has been factory overridden with apb\_seq\_error\_item

class reg\_bash\_seq extends uvm\_sequence #(apb\_seq\_item);

task body;

apb\_seq\_item original\_item = apb\_seq\_item::type\_id::create("item");

apb\_seq\_item item;

repeat(200) begin

$cast(item, original\_item.clone());

start\_item(item);

assert(item.randomize() with {addr inside {[`reg\_low:`reg\_high]};});

finish\_item(item);

end

end

endtask:body

endclass: reg\_bash\_seq

更高性能的示例只进行一次工厂创建调用，并使用clone()创建它的更多副本，因此每次都节省了扩展的工厂查找在较低性能的示例中，生成循环的时间循环。

避免在事务中嵌入covergroup

在事务中嵌入covergroup会增加其内存占用，但由于事务是一次性的，因此也没有意义收集保险的正确位置是成分可以通过基于事务内容对组件中的覆盖组进行采样来收集事务覆盖

### 较低性能版本

// APB Sequence item

class apb\_seq\_item extends uvm\_sequence\_item;

bit[31:0] addr;

bit[31:0] data;

apb\_opcode\_e we

covergroup register\_space\_access\_cg;

ADDR\_RANGE: coverpoint addr[7:0];

OPCODE: coverpoint we {

bins rd = {APB\_READ};

bins\_wr = {APB\_WRITE};

}

ACCESS: cross ADDR\_RANGE, OPCODE;

endgroup: register\_space\_access\_cg;

function void sample();

register\_space\_access\_cg.sample();

endfunction: sample

// Rest of the sequence item ...

endclass: apb\_seq\_item

// Sequence producing the sequence item:

class bus\_access\_seq extends uvm\_sequence #(apb\_seq\_item);

task body;

apb\_seq\_item apb\_item = apb\_seq\_item::type\_id::create("apb\_item");

repeat(200) begin

start\_item(apb\_item);

assert(apb\_item.randomize());

apb\_item.sample();

finish\_item(apb\_item);

end

endtask: body

更高性能版本

// apb\_seq\_item implemented without a covergroup

// Therefore bus\_access\_seq does not sample covergroup

// Sampling coverage in the driver:

class apb\_coverage\_driver extends apb\_driver;

covergroup register\_space\_access\_cg() with function sample(bit[7:0] addr, apb\_opcode\_e we);

ADDR\_RANGE: coverpoint addr;

OPCODE: coverpoint we {

bins rd = {APB\_READ};

bins\_wr = {APB\_WRITE};

}

ACCESS: cross ADDR\_RANGE, OPCODE;

endgroup: register\_space\_access\_cg;

task run\_phase(uvm\_phase phase);

apb\_seq\_item apb\_item;

forever begin

seq\_item\_port.get(apb\_item);

register\_space\_access\_cg.sample(apb\_item.addr[7:0], apb\_item.we);

// Do the signal level APB cycle

seq\_item\_port.item\_done();

end

endtask: run\_phase

// ....

endclass: apb\_coverage\_driver

较低性能的示例显示了在事务中使用covergroup来收集输入刺激功能覆盖信息。这会给

事务，这是由更高性能的示例所避免的，该示例基于事务的内容在静态组件中收集覆盖率

使用UVM报告宏

原始UVM报告方法在消息组装中的所有昂贵的字符串格式化操作完成之前不会检查消息的详细程度的`uvm\_info()、`uvm\_warning()、`uvm\_error()和`uvm\_fatal()宏首先检查消息的详细程度，然后只在要打印消息时进行字符串格式化。

较低性能版本

function void report\_phase(uvm\_phase phase);

if(errors != 0) begin

uvm\_report\_error("report\_phase", $sformatf(%0d errors found in %0d transfers", errors, n\_tfrs));

end

else if(warnings != 0) begin

uvm\_report\_warning("report\_phase", $sformatf("%0d warnings issued for %0d transfers", warnings, n\_tfrs));

end

else begin

uvm\_report\_info("report\_phase", $sformatf("%0d transfers with no errors", n\_tfrs));

end

endfunction: report\_phase

更高性能版本

function void report\_phase(uvm\_phase phase);

if(errors != 0) begin

`uvm\_error("report\_phase", $sformatf("%0d errors found in %0d transfers", errors, n\_tfrs))

end

else if(warnings != 0) begin

`uvm\_warning("report\_phase", $sformatf("%0d warnings issued for %0d transfers", warnings, n\_tfrs))

end

else begin

`uvm\_info("report\_phase", $sformatf("%0d transfers with no errors", n\_tfrs), UVM\_MEDIUM)

end

endfunction: report\_phase

在所示的示例中，在每种情况下都将生成相同的报告，但是如果详细程度设置被设置为禁止消息，则更高性能的版本将在生成字符串之前检查详细程度在一个测试台上，消息，并且报告详细程度已设置为低，这可能会对性能产生很大影响

不要使用uvm\_printer类

uvm\_printer是一个便利类，最初设计用于使用字段宏，以便以多种格式之一打印出组件层次结构或事务内容。的类会带来性能开销，使用convert2string()方法。convert2string()方法返回一个可以使用UVM消息传递宏显示或打印的字符串

较低性能版本

apb\_seq\_item bus\_req = abp\_seq\_item::type\_id::create("bus\_req");

repeat(20) begin

start\_item(bus\_req);

assert(bus\_req.randomize());

finish\_item(bus\_req);

bus\_req.print();

end

end

更高性能版本

apb\_seq\_item bus\_req = abp\_seq\_item::type\_id::create("bus\_req");

repeat(20) begin

start\_item(bus\_req);

assert(bus\_req.randomize());

finish\_item(bus\_req);

`uvm\_info("BUS\_SEQ", bus\_req.convert2string(), UVM\_HIGH)

end

end

print()方法调用$display()而不检查详细设置。

## 避免在UVM寄存器代码中使用get\_xxx\_by\_name()

使用get\_field\_by\_name()或get\_register\_by\_name()函数涉及到一个常规的表达式搜索寄存器模型中的所有寄存器字段名称或寄存器名称字符串，以返回字段或寄存器的句柄随着寄存器模型的增长，这种搜索将变得越来越昂贵。

使用寄存器模型中的分层路径来访问寄存器内容，这要高效得多

较低性能版本

task set\_txen\_field(bit[1:0] value);

uvm\_reg\_field txen;

txen = rm.control.get\_field\_by\_name("TXEN");

txen.set(value);

rm.control.update();

endtask: set\_txen\_field

更高性能版本

task set\_txen\_field(bit[1:0] value);

rm.control.txen.set(value);

rm.control.update();

endtask: set\_txen\_field

set\_txen\_field的更高性能版本避免了对字段名称字符串进行昂贵的正则

在UVM寄存器代码中尽量减少使用get\_registers()或get\_fields()

这些调用和其他类似的调用返回对象句柄的返回队列，这是为了方便起见，因为队列是一个未调整大小的数组。调用这些方法需要填充队列，如果寄存器模型的大小合理，则这可能是一项开销反复呼唤这些方法是没有意义的，它们应该只需要在一个范围内被调用一次或两次。

### 较低性能版本

int [$];int nums;

uvm\_reg regs[$];

randc int idx;

int no\_regs;

repeat(200) begin

regs = rm.encoder.get\_registers();

no\_regs = regs.size();

repeat(no\_regs) begin

tassert(this.randomize() with {idx =< no\_regs;});

assert(regs[idx].randomize());

regs[idx].update();

end

end

更高性能版本

uvm\_reg regs[$];

randc int idx;

int no\_regs;

regs = rm.encoder.get\_registers();

repeat(200) begin

regs.shuffle();

foreach(regs[i]) begin

assert(regs[i].randomize());

regs[i].update();

end

end

代码的高性能版本只执行一次get\_registers()调用，避免了与低性能版本中的重复调用相关的开销。

使用UVM异议，但要明智

提出UVM反对的目的是阻止一个阶段完成，直到一个线程准备好让它完成。提出和删除反对会导致组件层次结构被遍历，反对在所有组件中被提出或删除，一直到层次结构的顶部。因此，提出和降低一个反对是昂贵的，随着测试平台层次结构的深度增加，成本会变得对象应该只由控制线程使用，并且放置对象的适当位置是在顶级测试类的运行时方法中，或者在虚拟类的主体方法中。

顺序在任何其他地方使用它们可能是不必要的，也会导致性能下降。

较低性能版本

class adpcm\_seq extends uvm\_sequence #(adpcm\_seq\_item);

//...

task body;

uvm\_objection objection = new("objection");

adpcm\_seq\_item item = adpcm\_seq\_item::type\_id::create("item");

repeat(10) begin

start\_item(item);

assert(item.randomize());

objection.raise\_objection(this);

finish\_item(item);

objection.drop\_objection(this);

end

// Inside the virtual sequence

adpcm\_sequencer ADPCM;

task body;

adpcm\_seq do\_adpcm = adpcm\_seq::type\_id::create("do\_adpcm");

do\_adpcm.start(ADPCM);

endtask

更高性能版本

// Sequence to be called:

class adpcm\_seq extends uvm\_sequence #(adpcm\_seq\_item);

//...

task body;

adpcm\_seq\_item item = adpcm\_seq\_item::type\_id::create("item");

repeat(10) begin

start\_item(item);

assert(item.randomize());

finish\_item(item);

end

// Inside the virtual sequence

adpcm\_sequencer ADPCM;

task body;

uvm\_objection objection = new("objection");

adpcm\_seq do\_adpcm = adpcm\_seq::type\_id::create("do\_adpcm");

objection.raise\_objection(ADPCM);

do\_adpcm.start(ADPCM);

objection.drop\_objection(ADPCM);

endtask

在代码的更高性能版本中，在sequence并在最后丢弃，在时间上包含发送给驱动程序的所有sequence\_items

尽量减少使用UVM回调

在UVM中执行回调在与注册和执行它们相关联的存储器使用代码方面都是昂贵的并发症主要是这是由于保留了注册回调的顺序为为了提高性能，通过使用替代方法来实现相同的功能，避免使用UVM回调

例如，通过扩展uvm\_reg类或使用回调类，可以使用事务查看来记录和查看寄存器访问

从uvm\_reg扩展的类重载pre\_read()和pre\_write()方法，以便在调用寄存器read()或write()方法时开始事务，并重载post\_read()方法以及post\_write()方法，用于在寄存器传输完成时结束事务这将导致为每个寄存器访问记录一个事务，只要扩展类用作寄存器模型的基类

另一种方法是使用一个uvm\_reg\_cbs类，它包含uvm\_reg的回调pre\_read()、pre\_write()、post\_read()和post\_write()方法。与扩展类一样，pre\_xxx()方法开始记录事务，post\_xxx()方法结束记录交易然后使用包函数enable\_reg\_recording()为每个寄存器注册回调类对象

使用回调的低性能版本

//

// Call-Back class for recording register transactions:

//

class record\_reg\_cb extends uvm\_reg\_cbs;

virtual task pre\_write(uvm\_reg\_item rw);

endtask

virtual task post\_write(uvm\_reg\_item rw);

endtask

virtual task pre\_read(uvm\_reg\_item rw);

endtask

function void do\_record(uvm\_recorder recorder);

endfunction

endclass : record\_reg\_cb

//

// Package function for enabling recording:

//

function void enable\_reg\_recording(uvm\_reg\_block reg\_model,

reg\_recording\_mode\_t record\_mode = BY\_FIELD);

uvm\_reg regs[$];

record\_reg\_cb reg\_cb;

//Set the recording mode

uvm\_config\_db #(reg\_recording\_mode\_t)::set(null,"\*","reg\_recording\_mode", record\_mode);

//Get the queue of registers

reg\_model.get\_registers(regs);

//Assign a callback object to each one

foreach (regs[ii]) begin

reg\_cb = new({regs[ii].get\_name(), "\_cb"});

uvm\_reg\_cb::add(regs[ii], reg\_cb);

end

reg\_cb = null;

uvm\_reg\_cb::display();

endfunction : enable\_reg\_recording

使用类扩展的更高性能版本

//

// Extension of uvm\_reg enables transaction recording

//

class record\_reg extends uvm\_reg;

virtual task pre\_write(uvm\_reg\_item rw);

endtask

virtual task post\_write(uvm\_reg\_item rw);

endtask

virtual task pre\_read(uvm\_reg\_item rw);

endtask

function void do\_record(uvm\_recorder recorder);

endfunction

endclass : record\_reg

在这种情况下使用回调的主要论点是，它不需要将寄存器模型与扩展类一起使用，这意味着它可以“翻新”到使用标准UVM uvm\_reg类的寄存器模型然而，这是以巨大的开销为代价的--在寄存器模型中，每个寄存器都有一个额外的回调对象结构来调用回调对象中的方法。

假设可能会生成寄存器模型，并且在较大的设计中可能会有数千

P

以最小的不便给用户的性能。



[先前](https://verificationacademy.com/cookbook/uvm-universal-verification-methodology/uvm-guidelines)

[下](https://verificationacademy.com/cookbook/uvm-universal-verification-methodology/migrating-to-uvm-1-2-release)

**UVM性能指南** \*

关于UVM性能指南的最新论坛讨论

[**从测试的UVM报告摘要中屏蔽“Report counts by id”**](https://verificationacademy.com/forums/t/mask-report-counts-by-id-from-tests-uvm-report-summary/38076)

UVM

[**配置**](https://verificationacademy.com/forums/t/configuration/35631)

UVM

[**如果工具是基于事件的，如何使我们的uvm环境兼容.**](https://verificationacademy.com/forums/t/if-tool-is-event-based-the-how-to-make-our-uvm-environment-compatble-to-tool/33657)

U...

[**Pre\_randomize在SystemVerilog性能指南中的用法**](https://verificationacademy.com/forums/t/pre-randomize-usage-in-systemverilog-performance-guidelines/33413)

UVM

[**UVM加速TB(联合仿真)**](https://verificationacademy.com/forums/t/uvm-accelerated-tb-co-emulation/29928)

UVM

[问一个问题[](https://verificationacademy.com/forums/new-topic?tags=uvm-performance-guidelines)](https://verificationacademy.com/forums/new-topic?tags=uvm-performance-guidelines)

西门子

西门子数字工业软件

[4月](https://facebook.com/Siemens)[20](https://twitter.com/Siemens)[日](https://youtube.com/Siemens)

奥特尔

[云](https://www.sw.siemens.com/en-US/digital-transformation/cloud/)

[设计、制造和PLM软件](https://plm.sw.siemens.com/en-US/)[电子设计自动化](https://eda.sw.siemens.com/en-US/)

[InsightsHub](https://plm.sw.siemens.com/en-US/insights-hub/)[菜单](https://www.mendix.com/)

**如何购买**[购买西门子](https://www.sw.siemens.com/en-US/buy/)[在线购买](https://www.dex.siemens.com/)

[合作伙伴](https://www.sw.siemens.com/en-US/partners/find-a-partner/)[学术](https://www.sw.siemens.com/en-US/academic/)[更新](https://www.sw.siemens.com/en-US/support-services/renewal/)

**西门子**[关于我们](https://www.sw.siemens.com/en-US/)

[社区](https://community.sw.siemens.com/s/)[活动](https://events.sw.siemens.com/en-US/)

[领导](https://www.sw.siemens.com/en-US/leadership/)[新闻和新闻](https://newsroom.sw.siemens.com/)[信托中心](https://www.sw.siemens.com/en-US/trust-center/)

接触

[联系我们](https://verificationacademy.com/contact)

[联系我们](https://hls.academy/contact)

[- 联系我们](https://plm.sw.siemens.com/en-US/contact-plm/)[EDA -联系我们](https://resources.sw.siemens.com/en-US/contact-eda)[全球办事处](https://www.sw.siemens.com/en-US/office-locations/)[支持中心](https://support.sw.siemens.com/en-US?ref=footer)

[PLM](https://plm.sw.siemens.com/en-US/contact-plm/)

[提供反馈](https://webtac.industrysoftware.automation.siemens.com/feedback/)[报告盗版](https://www.sw.siemens.com/en-US/piracy-prevention/)

西门子2024

[使用条款](https://www.siemens.com/global/en/general/terms-of-use.html)[隐私政策](https://www.sw.siemens.com/en-US/privacy-policy/)[Cookie声明](https://www.siemens.com/global/en/general/cookie-notice.html)[DMCA](https://sw.siemens.com/en-US/dmca)[举报](https://www.siemens.com/global/en/company/about/compliance/reporting-channels.html)