2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

How Far Can You Take UVM Code Generation and Why Would You Want To?

John Aynsley, Doulos

DOULOS

Agenda

Goals for code generation

The Easier UVM Code Generator

Characteristics of UVM code

Template-driven code generation

How far can you take UVM code generation?

Why would you want to?

Conclusions based on experience

Characteristics

Goals for Code Generation (1)

The quality of the code

Better than I write

Goals for Code Generation (2)

The quality of the generator

Generates what I want

Iterative

Doulos – Easier UVM

Coding guidelines – "One way to do it"

Free and open

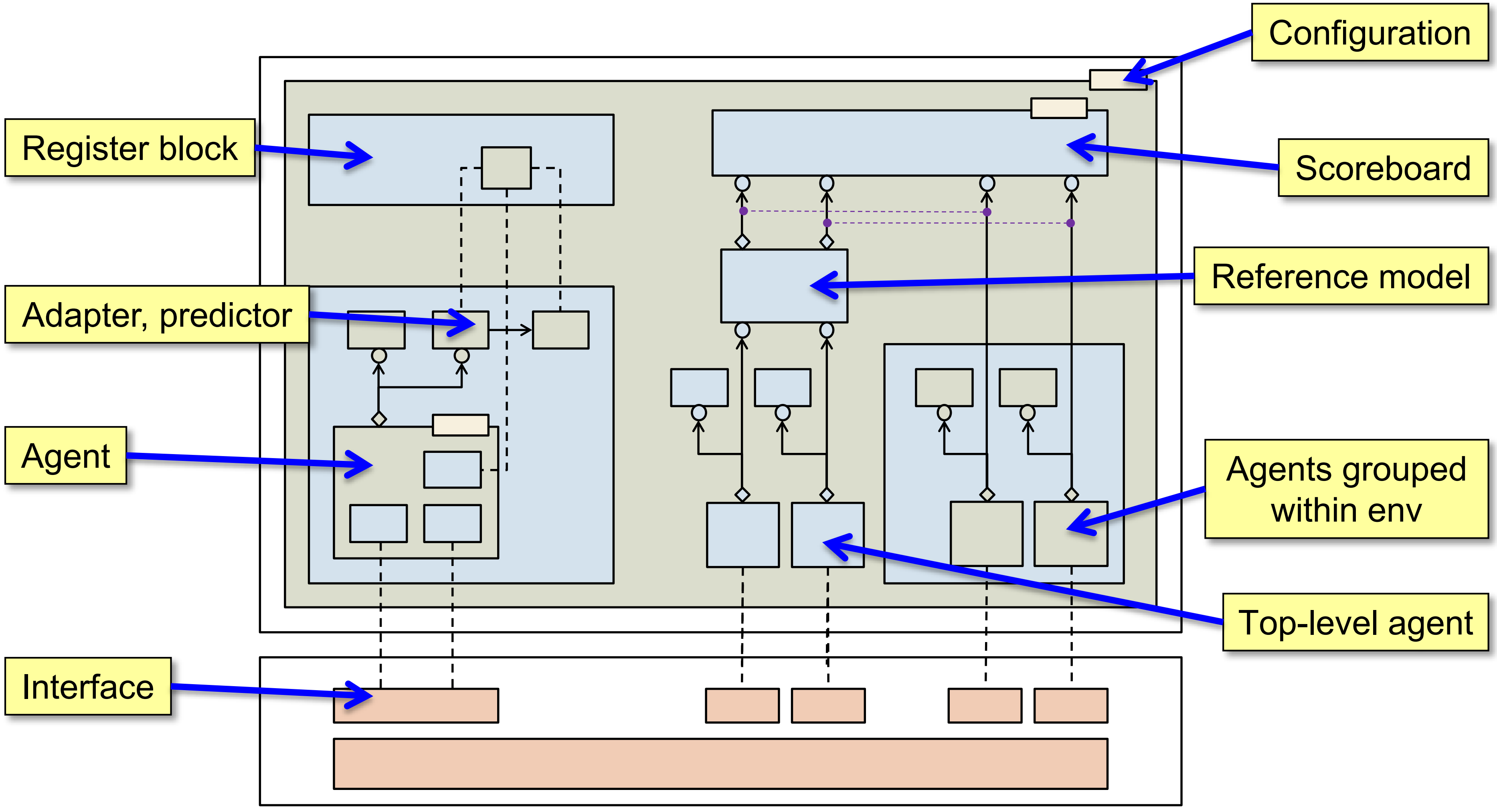
Automatic code generator

Apache 2.0 license

180 detailed guidelines with explanations and examples

Code generator complies with guidelines

Easier UVM Generator (1)



2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Easier UVM Generator (2)

Per-interface/agent:

uvm_sequence_item class
Configuration class
Sequencer class
Driver class
Monitor class
Agent class

Split transactors for emulation

Subscriber class

Default sequence class for agent

Default virtual sequence class for env

Register model adaptor
Agent package
SystemVerilog interface

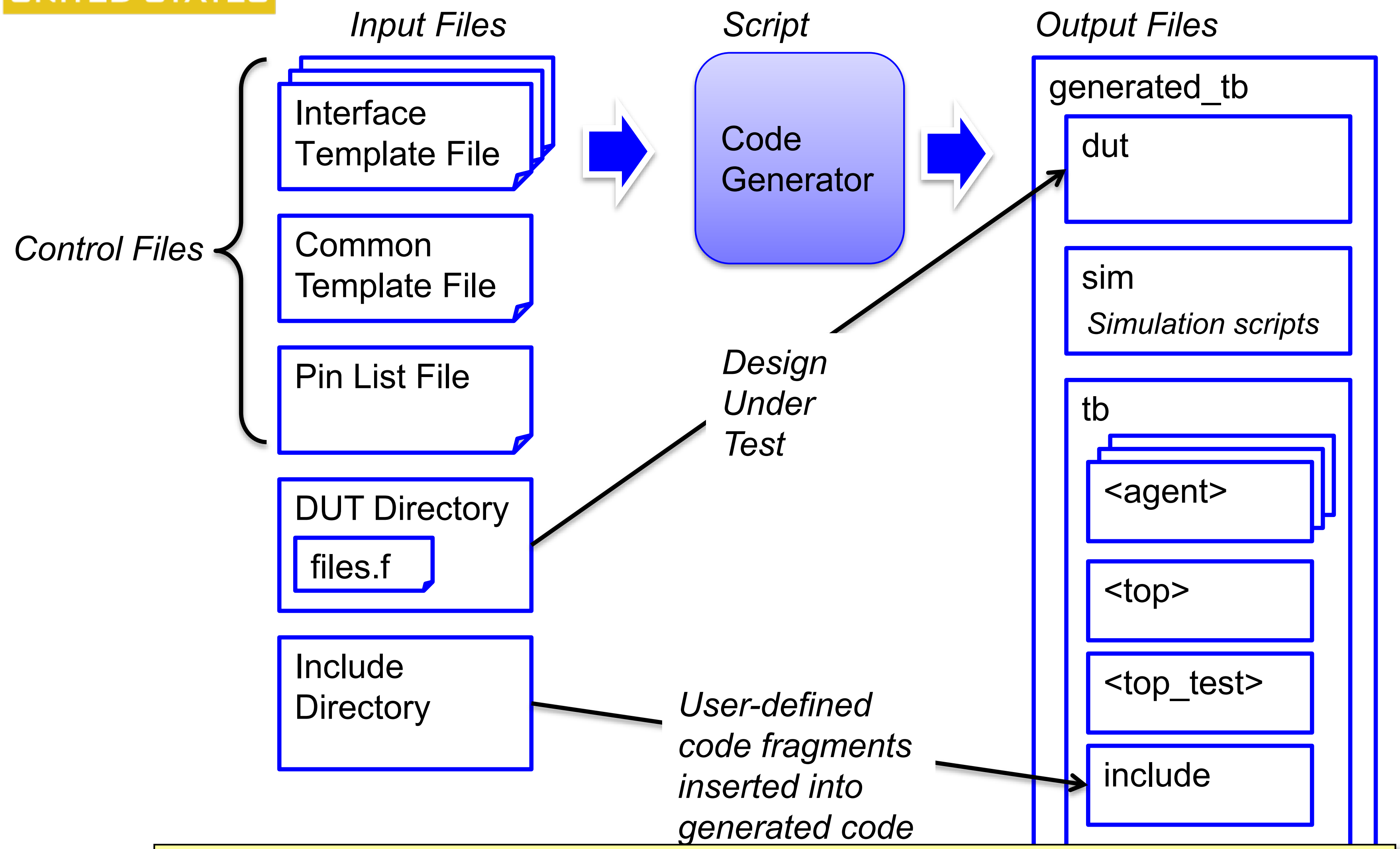
Top-level:

Top-level configuration object
Top-level env
Reference model

Instantiation of Syosil scoreboard

Instantiation of register block
Default top-level virtual sequence class
Top-level package
Test class
ackage
arness module to instantiate DUT
vel module

Code Generator Inputs & Outputs



Perl script, code ready-to-run, open source, well documented, regression tested

Ways to Use a Code Generator

Generate examples as a learning aid

An initial framework for production code

Continually regenerate code throughout the project

Agenda

Goals for code generation

The Easier UVM Code Generator

Characteristics of UVM code

Template-driven code generation

How far can you take UVM code generation?

Why would you want to?

Conclusions based on experience

Characteristics of UVM Code

```
`ifndef BUS_SEQ_ITEM_SV
`define BUS_SEQ_ITEM_SV

class bus_tx extends uvm_sequence_item;
  `uvm_object_utils(bus_tx)

  rand bit  cmd;
  rand byte addr;
  rand byte data;

  extern function new(string name = "");
  extern function void do_copy(uvm_object rhs);
  extern function bit  do_compare(uvm_object rhs, uvm_comparer comparer);
  extern function void do_print(uvm_printer printer);
  ...
```

Boilerplate with user-defined fields

User-defined

Boilerplate

Template-Driven Code Generation

Variable fields

Repeated line groups

Conditional line groups

... arbitrarily nested

Mark vertical alignment points

Mark insertion points

Mark groups of lines as suppressible

$\Big\}$ 95%

Ad hoc rules that require partially parsing user-defined code fragments

Variable fields assigned using arbitrary expressions

A Hypothetical Code Template

```
`ifndef ${agent_name}_ENV_SV
`define ${agent_name)_ENV_SV

--insert_inc_file $agent_env_inc_before_class{$agent_name};

class ${agent_name}_env extends uvm_env;

  `uvm_component_utils(${agent_name}_env)

  extern function new(string name, uvm_component parent);

--for ( $i = 0 ; $i < $number_of_instances{$agent_name}; $i++ ) {

--  $suffix = calc_suffix($i, $number_of_instances{$agent_name});

  ${agent_name}_config   m_${agent_name}${suffix}_config;
  ${agent_name}_agent    m_${agent_name}${suffix}_agent;
  ${agent_name}_coverage m_${agent_name}${suffix}_coverage;

--  if ( @env_list) {
  // Child environments
  ...
```

Variable fields

Location for include

Variable fields

Repetition

Ad hoc

Variable fields

Conditional

Agenda

Goals for code generation

The Easier UVM Code Generator

Characteristics of UVM code

Template-driven code generation

How far can you take UVM code generation?

Why would you want to?

Conclusions based on experience

How Far Can You Take UVM Code Generation? (1)

No limits! A few examples ...

Organising classes into packages using `include

Indentation and alignment – pretty printing

Which UVM features to use where

Where to set/get virtual interfaces

How Far Can You Take UVM Code Generation? (2)

Contributing to automation and productivity

Instantiating and connecting DUT and interfaces

Stitching together all the VIP at the top level

Generating methods for each transaction class

Generating simulation scripts

How Far Can You Take UVM Code Generation? (3)

Working example code

Default clock and reset generation

Default covergroups and sampling

Default end_of_elaboration to print diagnostics

Default sequences

Why Would You Want To? (1)

For learners

Reinforce training with complete, working examples

Why Would You Want To? (2)

For all users

Consistency

Productivity – automatic generation

Agenda

Goals for code generation

The Easier UVM Code Generator

Characteristics of UVM code

Template-driven code generation

How far can you take UVM code generation?

Why would you want to?

Conclusions based on experience

Issues - Regeneration

Must be able to regenerate the entire output

(lost or corrupted files, regression)

Do not modify the generated code until ...

... you are ready to burn your bridges

Issues - Flexibility

Predefined insertion points

Suppressing automatic generation

Extend classes and override

It still won't be enough!

Beautify the Code Generator!

Maintain a simple, consistent overall structure

Take care over the naming of internal variables

Keep the code templates close to plain text

Keep the meta-instructions regular and simple

Build a regression test suite

Summary

Consistency and automation are worth having!

Where Can I Get It?

http://www.doulos.com/easier

http://www.edaplayground.com

The simplest example

http://www.edaplayground.com/x/65x