Agnostic UVM-XX
Testbench Generation

Jacob Andersen, SyoSil ApS, Copenhagen, Denmark
Stephan Gerth, Fraunhofer IIS, Dresden, Germany
Filippo Dughetti, SyoSil ApS, Copenhagen, Denmark
Introduction

• Code generation and **model driven software development** present for long time in ASIC verification

• **Simplify** usage of UVM by **abstraction** of the testbench

• Generate SystemVerilog and SystemC testbenches

• Reusable UVM Verification Components (UVCs) and testbenches supporting hardware in the loop (HiL)
Eclipse Modeling Framework

• UVM-XX generator implemented in the open source Eclipse modeling Framework (EMF)

• EMF benefits:
  – Preservable user regions
  – Easiness of updating the abstract specification
  – Multiple input formats
  – Multiple output formats via simple templates
  – Eclipse context aware editing and syntax highlighting
EMF Flow

Model Description
XSD  eBNF  UML  ECore

generate

EMF Model

Templates

EMF APPLICATION

input

output

© Accellera Systems Initiative
UVM Testbench Abstraction

- Pragmatism when defining the abstraction
- Layered abstraction:
  - **LUVC**: implementation of UVCs
  - **LB2B**: implementation of back2back testbench
  - **LTB**: implementation of a RTL testbench
- Common **Domain Specific Language (DSL)** for all three layers and their relations
  - compact, easy to read format
  - Verification Environment format (VE)
UVM Testbench Abstraction

- EMF supports XTEXT for defining an eBNF-like specification of the VE DSL
- Single DSL used for all three layers: a separation on syntax level is needed
- Two types: VerificationComponent (LUVC) or EnvironmentComponent (LB2B, LTB).

```
Component:
  VerificationComponent | EnvironmentComponent
;
VerificationComponent:
  'type' = "vc"
  'name' = name = STRING
...
EnvironmentComponent:
  (imports+=Import)*
  'type' = "env"
  'name' = name = STRING
  'mode' = mode = STRING
...```

© Accellera Systems Initiative
UVM Testbench Abstraction

• Finding the correct level of abstraction versus expressiveness for specifying a UVC is the key

```verilog
VerificationComponent:
  'type' = "vc"
  'name' = name = STRING
  ( constantBlocks += ConstantBlock
  | typeBlocks += TypeBlock
  | agentBlocks += AgentBlock
  | driverBlocks += DriverBlock
  | interfaceBlocks += InterfaceBlock
  | checkBlocks += CheckBlock
  | transactionBlocks += TransactionBlock
  | configurationBlocks += ConfigurationBlock)*
```

• Protocol specific code and types handling
UVM Testbench Abstraction

• Type handling can get difficult when targeting SystemVerilog AND SystemC

• Three ways to support types
  – Raising the abstraction level of types
  – Full SV to SC type translator which would parse the existing SV string and translate that into the appropriate SC type
  – Simpler version of (2) with fixed mapping of standard types

• Addition of an optional `sc_type` string
UVM Testbench Abstraction

- EnvironmentComponent defines information needed for generation of LB2B and LTB layers

```plaintext
EnvironmentComponent:
  (imports+=Import)*
  'type' '=' ''env''
  'name' '=' name = STRING
  'mode' '=' mode = STRING
  ( verifcompBlocks += VerifcompBlock
    | scoreboardBlocks += ScoreboardBlock
    | envconfigBlocks += EnvconfigBlock
    | sequenceBlocks += SequenceBlock
    | testBlocks += TestBlock)*

; Import:
  'import' importURI = STRING
```

© Accellera Systems Initiative 9
Generating UVM

- UVM code generation is straight forward due to the EMF framework
Wishbone Example - UVC

- Wishbone UVC
- VE file with type field set to “vc” has been defined

```plaintext
//-------------------------------
// Definition for the WB VC
// Lead UVMGen example.
//-------------------------------
type="vc"
name="wb"
```
Wishbone Example - UVC

- VE file continued with remaining blocks as defined by the DSL

```helloworld
// Wishbone protocol interface definition
agent { numInst = 1; protocol = wishbone; } 
driver wishbone {
  variant = "master" comment="Master side of protocol";
  variant = "slave" comment="Slave side of protocol"; }
interface {
  parameter {
    name="WB_DATA_BYTE_NUM" type="int" value="pk_wb::WB_DATA_BYTE_NUM"
    comment="WB data number of bytes";
    name="WB_DATA_WIDTH" type="int" value="pk_wb::WB_DATA_WIDTH"
    comment="WB data width";
    ...
  }
  clock { name="clk" type="logic" comment="Main clock"; }
  reset { name="rst" type="logic" comment="Main reset"; }
  signal {
    name="dat_o" type="logic[WB_DATA_WIDTH-1:0]" driver="master"
    resetval="x" comment="WISHBONE data with same direction as adr";
    name="adr" type="logic[WB_ADDR_WIDTH-1:0]" driver="master"
    resetval="x" comment="WISHBONE address";
    name="sel" type="logic[WB_DATA_BYTE_NUM-1:0]" driver="master"
    resetval="x" comment="WISHBONE byte select"; ...
  }
}
```
Wishbone Example - UVC

- UVMGen generates the UVM classes for the UVC
Wishbone Example – B2B TB

• Similar approach for generating a back to back testbench

```verilog
// Definition for the WB back2back verification setup.
// Lead UVMGen example.

import "wishbone.ve"
type="env"
name="wb_b2b"
mode="back2back"
```
Wishbone Example – B2B TB

- VE file continued with remaining blocks as defined by the DSL

```c
// List of verification components to use
// Type must be known VC
verifcomp {
    name="wb_master"  type="wb"  active="yes"  driver="master";
    name="wb_slave"   type="wb"  active="yes"  driver="slave";
    name="wb_monitor" type="wb"  active="no" ; }

// List of configurations to be used by tests
envconfig {
    name="basicconfig"  base="";       // Declares cl_config_basicconfig,
    extends cl_wb_b2b_config
    name="userconfig"   base="basicconfig"; }// Declares cl_config_userconfig,
    extends cl_config_basic
...```
Wishbone Example – B2B TB

• UVMGen generates the UVM classes for the B2B TB

```vhdl
class cl_wb_b2b_env extends uvm_env;
---
cl_wb_b2b_env::init();
---
cl_wb_b2b_env::build_phase(uvm_phase phase);
---
void build_phase(uvm::uvm_PHASE phase);
---
cl_wb_b2b_env::exec_phase();
---
void exec_phase();
---
cl_wb_b2b_env::final_phase();
---
void final_phase();
---
endclass
---
class cl_wb_b2b_env: public uvm::uvm_env {
---
cl_wb_b2b_env::init();
---
cl_wb_b2b_env::build_phase(uvm_phase phase);
---
void build_phase(uvm::uvm_PHASE phase);
---
cl_wb_b2b_env::exec_phase();
---
void exec_phase();
---
cl_wb_b2b_env::final_phase();
---
void final_phase();
---
endclass
---
```

Demo

```
-rw-r--r-- 1 filippo syosilg  8372 oct 11 12:23 cl_wb_wishbone_monitor.svh
-rw-r--r-- 1 filippo syosilg  7939 oct 11 12:23 if_wb.sv
-rw-r--r-- 1 filippo syosilg  5795 oct 11 12:23 if_wb_svachecker.sv
-rw-r--r-- 1 filippo syosilg  2785 oct 11 12:23 pk_wb.sv
-rw-r--r-- 1 filippo syosilg  3308 oct 11 12:23 wb_common.svh
-rw-r--r-- 1 filippo syosilg  1305 oct 11 12:23 wb_defines.svh
-rw-r--r-- 1 filippo syosilg  1810 oct 11 12:23 wb_vc.mk
```

```
[filippo@ssas01 DVCon_2016]$ ll output/src/wb_vc/sc
total 96
-rw-r--r-- 1 filippo syosilg  5830 oct 11 12:23 cl_wb_config.cpp
-rw-r--r-- 1 filippo syosilg  4599 oct 11 12:23 cl_wb_env.cpp
-rw-r--r-- 1 filippo syosilg 11102 oct 11 12:23 cl_wb_seq_item.cpp
-rw-r--r-- 1 filippo syosilg 10140 oct 11 12:23 cl_wb_seq_lib.cpp
-rw-r--r-- 1 filippo syosilg  3886 oct 11 12:23 cl_wb_sequencer.cpp
-rw-r--r-- 1 filippo syosilg  5802 oct 11 12:23 cl_wb_wishbone_agent.cpp
-rw-r--r-- 1 filippo syosilg  9462 oct 11 12:23 cl_wb_wishbone_driver.cpp
-rw-r--r-- 1 filippo syosilg  6565 oct 11 12:23 cl_wb_wishbone_monitor.cpp
-rw-r--r-- 1 filippo syosilg  4354 oct 11 12:23 if_wb.cpp
-rw-r--r-- 1 filippo syosilg  2790 oct 11 12:23 pk_wb.cpp
-rw-r--r-- 1 filippo syosilg  3445 oct 11 12:23 wb_common.cpp
-rw-r--r-- 1 filippo syosilg  1299 oct 11 12:23 wb_defines.cpp
-rw-r--r-- 1 filippo syosilg  1372 oct 11 12:23 wb_vc_GCC.mk
```

© Accellera Systems Initiative
Future Extensions

• Extend this to handle LTB as well
• Abstraction of functional coverage
• Extension of UVMGen limitless as template mechanism provides everything necessary
  – Example: testbench documentation could be generated from the same EMF model including a figure showing the testbench architecture, since the EMF model contains this information
Questions