Case Studies in SystemC

UVM for SystemC Users

John Stickley, Mentor Graphics
Gordon Allan, Mentor Graphics
UVM for SystemC users

- The case for hybrid testbenches
- Standards for hybrid testbenches
- What is UVM-Connect?
- UVM-Connect primer
- Reuse of legacy IP models
- Interchangeable testbench heads
- Dual use UVM drivers
- Hybrid testbench examples
- Summary
The case for hybrid testbenches

- Mounting use of TLM 2 as the common interconnect for multi-language HVL testbench environments:
  - TLM-2 standard is already “baked in” to IEEE 1666 SystemC and SV-UVM standards
  - Open-source public donations for “cross language connectivity fabric” are TLM-2 based and are already out there:
    - UVM-Connect - from Mentor Graphics
    - UVM-ML - from Accellera MLWG

- It makes sense to combine the strengths of 2 verification methodologies (SV-UVM, SystemC) into hybrid testbenches
  - Power of SystemVerilog UVM:
    - Constrained random traffic/sequence generation
    - Coverage, scoreboarding
  - Power of SystemC:
    - Good Linux host system interfacing capability which it gets for free simply by being written in C++
      - Direct access to host resources disks, networks, device drivers, X-windows displays, etc.
    - Stimulus via real host machine system interfaces and virtual platforms
The case for hybrid testbenches

- Types of hybrid testbenches:
  - Reuse of legacy IP models
    - Often RTL verification IP models come with C/C++ based APIs based on SystemVerilog P1800 DPI standard
    - Desirable to integrate this IP into SV-UVM testbench environments
  - Interchangeable testbench heads
    - A single source verification IP model can get reuse from multiple testbench modeling methodologies
    - Providing a TLM compliant socket API allows “interchangeable testbench heads”
  - Dual use UVM drivers
    - UVM drivers that provide interfaces to RTL models can be equipped with a 2\textsuperscript{nd} TLM port (in addition to traditional sequencer port) to provide an extra inbound traffic channel from peer TLM compliant models – possibly cross language ones
  - Virtual platform (VP) hybrid testbenches
    - Starting to see a number of QEMU derivatives out there which pair fast host based virtual platforms with RTL designs under test (DUTs)
Standards for hybrid testbenches: Achieving Interop with Standard Interfaces

- To interoperate two components must agree on
  - information to exchange (i.e., the data type)
  - means of exchanging that information (i.e., the interface)

- To be reusable, easy to use, components must be
  - Independent of their context, not expose implementation

Analogy: Media Server and TV
- They don’t know about each other; independent design
- They agree on common data (video) & interface (HDMI)
- *Both can be connected to many other devices*
What is UVM-Connect?

Enabling Technology

- Enables TLM communication between SV+SC
  - Using native TLM calls
  - Implemented over SystemVerilog P1800 DPI standard
- Leverages UVM TLM 2.0 implementation
  - Also supports TLM 1.0 and Analysis ports
- Provided as separate SV and SC packages
- Cross language binding “rendezvous” via a uvmc::connect() method
  - Uses a string to identify corresponding sockets
What is UVM-Connect?

Trans-language TLM connections

- Connect SysC & SV-UVM models using standard TLM1, TLM2 interface ports
  - TLM GP handled automatically
- Access and control UVM from SystemC via *command* API
  - Messaging, configuration, and factory methods supported
  - Synchronize SystemC to UVM phases
- Standards based, available today
  - Vendor-independent
  - Fully open-sourced, *Apache licensed* package just like the UVM 1.1b base package is

Can obtain from Mentor’s *Verification Academy* here:

http://verificationacademy.com/verification-methodology/uvm-connect
Add a scoreboard, add a connection.

```uvmc
#include "uvmc.h"
#include "consumer.h"

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

```sv
import uvm_pkg::*;
import uvmc_pkg::*;
`include "producer.sv"
`include "scoreboard.sv"

module sv_main;
  producer prod = new("prod");
  scoreboard sb = new("sb");
  initial begin
    prod.ap.connect(sb.expect_in);
    uvmc_tlm #()::
      connect(prod.out,"foo");
    uvmc_tlm1 #(uvm_tlm_gp)::
      connect(sb.actual_in,"bar");
    run_test();
  end
endmodule
```

© 2014 Mentor Graphics Corp.
UVM-Connect primer

TLM Connection – UVM-Aware SC \(\rightarrow\) SV

```c
#include <systemc.h>
using namespace sc_core;
#include "producer.h"
#include "uvmc.h"
using namespace uvmc;

struct prod_alt : public producer {
    prod_alt(sc_module_name nm) :
        producer(nm) {
        SC_THREAD(objector);
    }
    SC_HAS_PROCESS(prod_uvm)
    void objector() {
        uvmc_raise_objection("run");
        wait(done);
        uvmc_drop_objection("run");
    }
};

int sc_main(int argc, char* argv[]) {
    prod_alt prod("producer");
    uvmc_connect(prod.in, "42");
    sc_start(-1);
    return 0;
}
```

---

**raise objection, wait for “done”, drop objection**

**extend base producer**

**background thread**

---

```c
import uvm_pkg::*;
import uvmc_pkg::*;
`include "consumer.sv"

module sv_main;
    consumer cons = new("cons");
    initial begin
        uvmc_tlm #()::connect(cons.in, "42");
        uvmc_init();
        run_test();
        end
endmodule
```

---

**SV side must initialize UVMC command API**

---

March 3, 2014 9
Reuse of legacy IP models

SV-UVM Test

DPI based API

UVM-Connect’ions

APB Master TlmDriver

APB Slave DpiDriver

APB Monitor TlmDriver

TestbenchEnv

Scoreboard

RandWriteReadSeq

actualQ

compare

expectQ

apbUve Agent

apbUvc Agent

apbCover

Monitor

Driver

SystemC Testbench

APB Bus

Top

APB Master Xactor

APB Slave Xactor

APB Monitor Xactor

DPI = TLM-2.0 initiator -> target socket

= TLM analysis port -> subscribers

= Legacy DPI driver

March 3, 2014

SV-UVM Testbench

APB Bus

Top

APB Master Xactor

APB Slave Xactor

APB Monitor Xactor

DPI

DPI

DPI

DPI

= Legacy DPI driver

March 3, 2014

© 2014 Mentor Graphics Corp.
Reuse of legacy IP models

TLM-2 initiator <-> target sockets

---

```
// UVM-Connect'ed SV-UVM uvm_driver "initiator"
class ApbBusMasterDriver
    extends uvm_driver
    `uvm_component_utils(ApbBusMasterDriver)
protected
    string peerId; // UVM-Connect ID
    uvm_tlm_b_initiator_socket #(uvm_tlm_generic_payload)
        initiatorSocket;

    function new( string name, uvm_component parent );
    function void connect_phase( uvm_phase phase );
    task run_phase( uvm_phase phase );
endclass
```

```
// UVM-Connect'ed SystemC "target"
class Testbench : public sc_module {
    ApbMasterTlmDriver *apbMaster;
    Testbench( sc_module_name name );
};

class ApbMasterTlmDriver : public sc_module,
    public virtual tlm::tlm_fw_transport_if<>
{
    tlm::tlm_target_socket<32> socket;
    ApbMasterTlmDriver( sc_module_name name, const char *transactorPath );
    void b_transport( tlm::tlm_generic_payload &trans, sc_time &delay );
}
```
Reuse of legacy IP models

TLM-2 initiator <-> target sockets

SV-UVM Testbench

TestbenchEnv

ApbUve Agent

Monitor

Driver

APB Master TlmDriver

SystemC Testbench

APB Master DpiDriver

// UVM-Connect'ed SV-UVM uvm_driver "initiator"
class ApbBusMasterDriver
extends uvm_driver #(uvm_tlm_generic_payload); {
`uvm_component_utils(ApbBusMasterdriver)
protected string peerId; // UVM-Connect ID
uvm_tlm_b_initiator_socket#(uvm_tlm_generic_payload
initiatorSocket;

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

function void connect_phase( uvm_phase phase );
super.connect_phase( phase );
// Retrieve HDL path from UVM config DB
assert( get_config_string( "peerId", peerId ) )

uvmc_tlm #(uvm_tlm_generic_payload,
uvm_tlm_phase_e );
::connect( initiatorSocket, peerId );
endfunction

task run_phase( uvm_phase phase );

uvm_tlm_generic_payload request;
forever begin
  seq_item_port.get( request );
  initiatorSocket.b_transport( request, delay );
  seq_item_port.put( request
endfunction

endclass // }

// UVM-Connect'ed SystemC "target"
class Testbench : public sc_module {
  ApbMasterTlmDriver *apbMaster;
  Testbench( sc_module_name name ) : sc_module(name)
  apbMaster = new ApbMasterTlmDriver(
    "apbMaster", "Top.apbMasterTransactor" );
  uvmc_connect( apbMaster->socket, "master" );
}

class ApbMasterTlmDriver : public sc_module,
public virtual tlm_fw_transport_if<> {
  tlm_target_socket<32> socket;

  ApbMasterTlmDriver( sc_module_name name, const char *transactorPath )
  : sc_module( name ), socket( "socket"), ...
  dTransactorPath( transactorPath )
  { socket( *this ); ... }

  void b_transport(
    tlm_generic_payload &trans, sc_time &delay){
    // Extract fields of TLM GP and convert to DPI calls
  }

March 3, 2014
Reuse of legacy IP models

TLM-2 analysis broadcasters -> subscribers

// UVM-Connect’ed SV-UVM "subscriber"

class ApbBusMonitor
  extends uvm_subscriber
    #(uvm_tlm_generic_payload); // {
  `uvm_component_utils(ApbBusMonitor)
protected string peerId; // UVM-Connect ID
uvm_analysis_port #( uvm_tlm_generic_payload ) analysisPort;

function new(string name, uvm_component parent);
function void connect_phase( uvm_phase phase );
function void write( uvm_tlm_generic_payload t );
endclass // }

// UVM-Connect’ed SystemC "broadcaster"

class Testbench : public sc_module {
  ApbMonitorTlmDriver *apbMonitor;
  Testbench( sc_module_name name );
}

class ApbMonitorTlmDriver : public sc_module,
  public tlm::tlm_analysis_port<tlm::tlm_generic_payload> {
  tlm::tlm_generic_payload dMonitorRecordTrans;
  ApbMonitorTlmDriver( 
    sc_module_name name, const char *transactorPath );
  void write( const svBitVecVal *monitorRecord );
}

// Import "DPI-C" function
extern "C" void ApbMonitorWrite( 
  const svBitVecVal *monitorRecord );

ApbMonitorTlmDriver *me = (ApbMonitorTlmDriver *)svGetUserData( svGetScope(),
  (void *)&ApbMonitorWrite );
me->write( monitorRecord );

APB Monitor TlmDriver

TLM-2 analysis port (broadcaster)

SystemC Testbench

SV-UVM Test

RandWriteReadSeq

Scoreboard

TestbenchEnv

APB Monitor TlmDriver

APB Monitor Xactor

DPI

= Legacy DPI driver

= TLM-2.0 initiator -> target socket

= TLM analysis port -> subscribers

March 3, 2014

© 2014 Mentor Graphics Corp.
Reuse of legacy IP models
TLM-2 analysis broadcasters -> subscribers

SV-UVM Testbench

// UVM-Connect'ed SV-UVM "subscriber"
class ApbBusMonitor
extends uvm_subscriber
  #(uvm_tlm_generic_payload); // {
  `uvm_component_utils(ApbBusMonitor)
protected string peerId; // UVM-Connect ID
uvm_analysis_port #( uvm_tlm_generic_payload
analysisPort;
function new(string name, uvm_component parent);
  super.new(name, parent);
analysisPort = new( "analysisPort", this );
endfunction

function void connect_phase( uvm_phase phase );
  super.connect_phase( phase );
  // Retrieve peer ID from UVM config DB. If cannot
  // be found, assume this TB is not interested in
  // hooking up the monitor. Else, UVM-Connect it.
  // Note reference to 'analysis_export' data member
  // of base class uvm_subscriber.
  if( get_config_string( "peerId", peerId ) )
    uvmc_tlm1 #(uvm_tlm_generic_payload)
      ::connect( analysis_export, peerId );
endfunction

function void write( uvm_tlm_generic_payload t );
  analysisPort.write( t );
endfunction
endclass // }

SystemC Testbench

// UVM-Connect'ed SystemC "broadcaster"
class Testbench : public sc_module {
  ApbMonitorTlmDriver *apbMonitor;
  Testbench( sc_module_name name ) : sc_module(name)
  apbMonitor = new ApbMonitorTlmDriver( 
    "apbMonitor", "Top.apbMonitor" );
  uvmc_connect( *apbMonitor, "monitor" );
};

class ApbMonitorTlmDriver :
  public sc_module,
  public tlm::tlm_analysis_port<tlm::tlm_generic_payload
{ tlm::tlm_generic_payload dMonitorRecordTrans;
  ApbMonitorTlmDriver( 
    sc_module_name name, const char *transactorPath )
    : sc_module( name ),
      tlm::tlm_analysis_port<
        tlm::tlm_generic_payload>(name),
      dTransactorPath( transactorPath ) { }
  void write( const svBitVecVal *monitorRecord ){
    dMonitorRecordTrans.set_data_ptr( 
      const_cast<unsigned char *>(
        reinterpret_cast< const unsigned char *>(
          monitorRecord ) ) );
    tlm::tlm_analysis_port<
      tlm::tlm_generic_payload>::write(
        dMonitorRecordTrans );
  }
};
Interchangeable testbench heads

UART transactor example

UartTargetTransactor has 4 interchangeable initiator clients:
- SV-UVM TLM 2 test sequence client
- SV-UVM TLM 2 file i/o client
- SystemC TLM-2.0 xterm client
- SystemC TLM-2.0 file i/o client

= TLM-2.0 initiator -> target socket

= UVM-Connect “hidden” TLM conduit infrastructure

March 3, 2014
© 2014 Mentor Graphics Corp.
This configuration demonstrates a dual use SV-UVM driver equipped with both a sequencer port and a TLM-2 port.

In this case the TLM-2 port in both the SV-UVM driver and the slave memory (back-door port) are unused.
This configuration demonstrates a dual use SV-UVM driver equipped with both a sequencer port and a TLM-2 port.

In this case the sequencer port is unused.
Hybrid testbenches: Virtual platform example

Guest OS (Android/Linux)

Virtual Machine (FastModel FVP Simulator)

HVL Testbench (SystemC and/or SystemVerilog)

TestbenchEnv

AxiTalker
clock & Reset

AXI Agent

AXI Master Driver+Xactor

AXI Slave Driver+Xactor

AXI "Agent"

AXI Bridge Tlm2Rtl

SystemC

TBX Co-model Channel (DPI)

Veloce

soc_hdl_top

soc_dut_wrapper

AXI_IF

clocks, reset

Clock & Reset Generation

Graphics Subsystem

Interrupt Monitor

Co-Model Host

ARM FastModel Virtual Platform

RTSM CortexA8 Versatile EB System

Coverage Harness

SV-UVM

AxiCover

AxiTalker

Monitor

Driver

AXI "Agent"

AXI Master Driver+Xactor

AXI Slave Driver+Xactor

Clock & Reset Generation

Graphics Subsystem

Interrupt Monitor

Co-Model Host

ARM FastModel Virtual Platform

RTSM CortexA8 Versatile EB System

Coverage Harness

SV-UVM

AxiCover

AxiTalker

Monitor

Driver
Hybrid testbenches: Ethernet packet router example

This configuration uses SystemC DPI based legacy models, ...

Each “dotted line” TLM crossing is a UVM-Connect’ion!

... a SystemC golden reference model, ...

... an SV constraint solver, ...

... and an SV coverage analyzer.

SyscTestbench (SystemC)

Testbench (SystemVerilog)

Stimulus Generator (SV Constraint Solver)

Generic Router

Scoreboard

Coverage Analyzer

Router Configurator

Pbus Proxy

Pbus Xactor

DUT Router

MiiMaster Proxy

MiiMaster Xactor

MiiSlave Xactor

MiiSlave Proxy

inPort

outPort

inPort

inPort

inPort

inPort

inPort

inPort

procIf

expected

actual

reference

stimulus

actualTap

... an SV scoreboard, ...

sys (HDL)
Summary

- There is a good case to made for hybrid SV-UVM SystemC testbenches:
  - Supporting legacy IP models
  - Interchangeable testbenches can be coupled with reusable verification IP models
  - UVM drivers can be designed for "dual use" to accommodate alternative TLM channels for input
  - Hybrid testbenches allow taking combining of strengths of SystemC …
    - Virtual platforms, "real system" interfaces
    … with those of SV
    - Constrained random sequences, coverage, scoreboarding

- The TLM 1 and 2 standards are well supported by both SV-UVM and SystemC methodologies
  - There are now open-source "cross-language" TLM connectivity packages readily available - and being considered for standardization
Thank you