Mentor Graphics®

UVM Connect

Part 2 – Connections

Adam Erickson
Verification Technologist

academy@mentor.com
www.verificationacademy.com

VERIFICATION ACADEMY

UVM Connect Presentation Series

- **Part 1 – UVMC Introduction**
  - Learn what UVMC is and why you need it
  - Review the principles behind the TLM1 and TLM2 standards
  - Review basic port/export/interface connections in both SC and SV
- **Part 2 – UVMC Connections**
  - Learn how to establish connections between TLM-based components in SC and SV
- **Part 3 – UVMC Converters**
  - Learn how to write the converters that are needed to transfer transaction data across the language boundary
- **Part 4 – UVMC Command API**
  - Learn how to access and control key aspects of UVM simulation from SystemC

UVM Connections – Agenda

- **Review TLM components**
  - Show implementations of simple SC & SV producers and consumers
- **Review Native TLM connections**
  - Review how to make *native* component connections in SC and SV
- **Introduce the UVMC connect functions**
  - For registering TLM ports, exports, and imps for cross-language communication
- **Demonstrate how to use UVMC connect functions**
  - Point-to-point connections
  - Hierarchical connections (e.g. SC wraps SV)

UVM Connections – Agenda

- **Review TLM components**
  - Show implementations of simple SC & SV producers and consumers
- **Review Native TLM connections**
  - Review how to make *native* component connections in SC and SV
- **Introduce the UVMC connect functions**
  - For registering TLM ports, exports, and imps for cross-language communication
- **Demonstrate how to use UVMC connect functions**
  - Point-to-point connections
  - Hierarchical connections (e.g. SC wraps SV)

TLM Component Review – SV Producer

```
class producer extends uvm_component;

   uvm_tlm_b_initiator_socket #() out;

   `uvm_component_utils(producer)
   function new(string name, uvm_component parent=null);
      super.new(name,parent);
      out = new("out", this);
   endfunction

   task run_phase (uvm_phase phase);
      uvm_tlm_gp gp = uvm_tlm_gp::type_id::create("gp",this);
      uvm_tlm_time delay = new("del",1e-12);
      int num_trans = 2;
      phase.raise_objection(this);
      uvm_config_db #(uvm_bitstream_t)::get(this,"","num_trans",num_trans);
      for (int i = 0; i < num_trans; i++) begin
         delay.set_abstime(10,1e-9);
         assert(gp.randomize() with { constraints } );
         uvm_info("PRODUCER/PKT/SEND",{"\n",gp.sprint()},UVM_MEDIUM)
         out.b_transport(gp,delay);
      end
      `uvm_info("PRODUCER/END_TEST","Finished sending",UVM_LOW)
      phase.drop_objection(this);
   endtask
endclass
```

declare socket

allocate socket

REUSE this trans over and over!

don't let phase end!

config # of trans

randomize trans

send trans to (unknown) target

now let phase end

TLM Component Review – SC Producer

```
struct producer : public sc_module {
  simple_initiator_socket<producer> out; // tlm_gp
  int num_trans;
  sc_event done;

  producer(sc_module_name nm) : out("out"), num_trans(2) {
    SC_THREAD(run);
  }
  SC_HAS_PROCESS(producer);

  void run() {
    tlm_generic_payload gp;
    char unsigned data[8];
    gp.set_data_ptr(data);
    sc_time delay;
    while (num_trans--) {
      // randomize trans
      // (see example source, in kit)
      out->b_transport(gp,delay);
    }
    cout << sc_time_stamp() << "Ending test" << endl;
    done.notify();
  }
};
```

simple socket

declare ‘run’ thread

REUSE this trans over and over!

data outside class!

generate loop

send trans to (unknown) target

Wake up any waiters

TLM Component Review – SV Consumer

```
class consumer extends uvm_component;

   uvm_tlm_b_target_socket #(consumer) in;
   uvm_analysis_port #(uvm_tlm_generic_payload) ap;

   `uvm_component_utils(consumer)

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

   // task called via 'in' socket
   virtual task b_transport (uvm_tlm_gp t, uvm_tlm_time delay);
     `uvm_info("CONSUMER/PKT/RECV",{"\n",t.sprint()},UVM_MEDIUM)
     ...execute transaction...
     #(delay.get_realtime(1ns,1e-9));
     delay.reset();
     ap.write(t);
   endtask

endclass
```

blocking-only target socket

analysis port

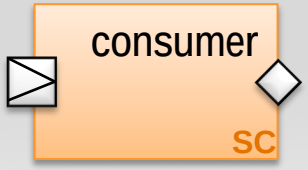
allocate ports

this task will be called by (unknown) initiators(s) via our “in” socket

send out analysis port to (unknown) subscribers

TLM Component Review – SC Consumer

```
class consumer : public sc_module
{
  public:
  simple_target_socket<consumer> in; // defaults to tlm_gp
  sc_port<tlm_analysis_if<tlm_generic_payload>,
          0,SC_ZERO_OR_MORE_BOUND> ap;

  consumer(sc_module_name nm) : in("in"), ap("ap") {
    in.register_b_transport( this, &consumer::b_transport );
  }

  virtual void b_transport(tlm_generic_payload &gp, sc_time &t) {
    ...execute transaction...
    wait(t);
    t = SC_ZERO_TIME;
    if (ap.size())
      ap->write(gp);
  }
};
```



simple socket

analysis port

allocate ports

register b_transport implementation

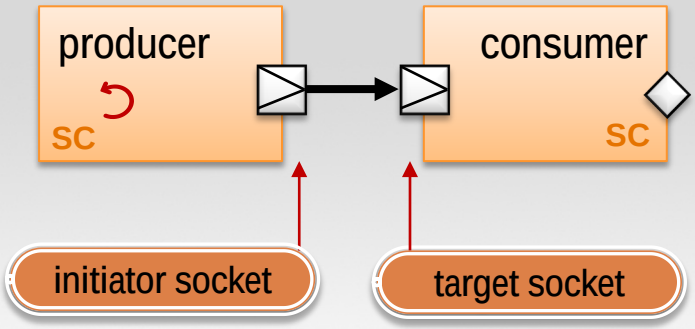
this function will be called by (unknown) initiator(s) via our “in” socket

send out analysis port to (unknown) subscribers

Agenda

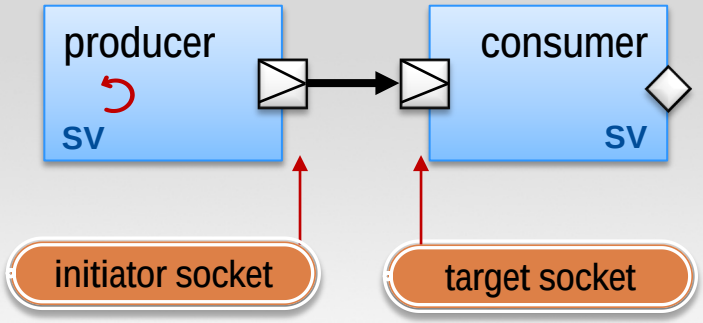
- Review TLM components
  - Show implementations of simple SC & SV producers and consumers
- **Review Native TLM connections**
  - Review how to make *native* component connections in SC and SV
- Introduce the UVMC connect functions
  - For registering TLM ports, exports, and imps for cross-language communication
- Demonstrate how to use UVMC connect functions
  - Point-to-point connections
  - Hierarchical connections (e.g. SC wraps SV)

TLM Connection – Native SC




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

int sc_main(int argc, char* argv[])
{
  producer prod("producer");
  consumer cons("consumer");
  prod.out.bind(cons.in);
  sc_start(-1);
  return 0;
}
```

TLM Connection – Native SV

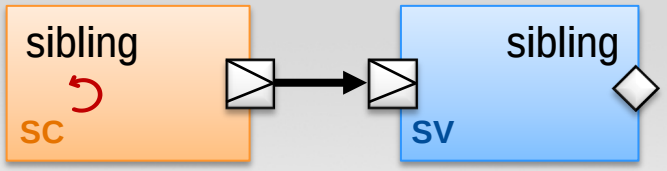



```
import uvm_pkg::*;
`include "producer.sv"
`include "consumer.sv"

module sv_main;
  producer prod = new("prod");
  consumer cons = new("cons");

  initial begin
    prod.out.connect(cons.in);
    run_test();
  end
endmodule
```

Agenda

- **Review TLM components**
  - Show implementations of simple SC & SV producers and consumers
- **Review Native TLM connections**
  - Review how to make *native* component connections in SC and SV
- **Introduce the UVMC connect functions**
  - For registering TLM ports, exports, and imps for cross-language communication
- **Demonstrate how to use UVMC connect functions**
  - Point-to-point connections
  - Hierarchical connections (e.g. SC wraps SV)

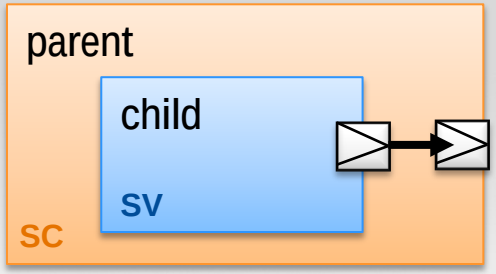
Register a port, export, or imp/interface for *point-to-point* cross-language communication



- **SV**
  - uvmc_tlm #(**T**)::connect (**port**, *lookup*); TLM2
  - uvmc_tlm1 #(**T**)::connect (**port**, *lookup*); TLM1 unidirectional
  - uvmc_tlm1 #(**REQ,** *RSP*)::connect (**port**, *lookup*); TLM1 bidirectional
- **SC**
  - uvmc_connect (**port**, *lookup*); TLM1 or TLM2

T **REQ, RSP**	The transaction type(s) being used by the port. SV only. For SV TLM1 bidirectional, if RSP not specified, default is REQ.
port	The port, export, or interface/imp instance to register for cross-language connection. Required.
lookup	An additional lookup string to associate with the port. During elaboration, UVMC will try to match the port's hierarchical name and this lookup string with other registered ports. Optional.

Register a port, export, or imp/interface for *hierarchical* cross-language communication



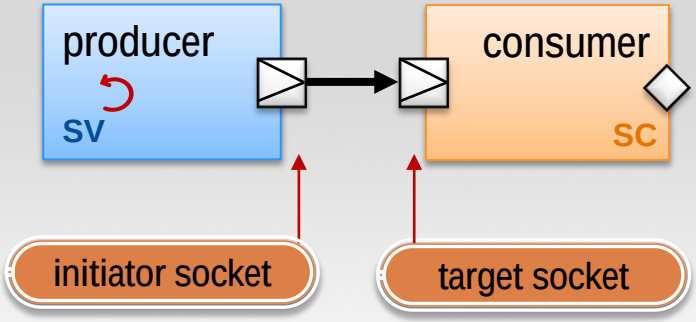
- **SV**
  - uvmc_tlm #(**T**)::connect_hier (**port**, *lookup*); TLM2
  - uvmc_tlm1 #(**T**)::connect_hier (**port**, *lookup*); TLM1 unidirectional
  - uvmc_tlm1 #(**REQ,** *RSP*)::connect_hier (**port**, *lookup*); TLM1 bidirectional
- **SC**
  - uvmc_connect_hier (**port**, *lookup*); TLM1 or TLM2

T **REQ, RSP**	The transaction types being used by the port. Required only for SV-side. For SV TLM1 bidirectional, if RSP not specified, default is REQ.
port	The port, export, or interface/imp instance to register for cross-language connection. Required.
lookup	An additional lookup string to associate with the port. During elaboration, UVMC will try to match the port's hierarchical name and this lookup string with other registered ports. Optional.

Agenda

- **Review TLM components**
  - Show implementations of simple SC & SV producers and consumers
- **Review Native TLM connections**
  - Review how to make *native* component connections in SC and SV
- **Introduce the UVMC connect functions**
  - For registering TLM ports, exports, and imps for cross-language communication
- **Demonstrate how to use UVMC connect functions**
  - Point-to-point connections
  - Hierarchical connections (e.g. SC wraps SV)

TLM Connection – SV to SC



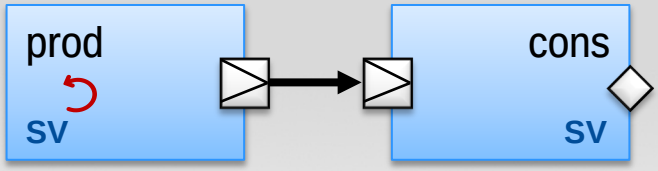
match by global lookup strings

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

module sv_main;
 producer prod = new("prod");
 initial begin
  uvmc_tlm #()::connect(prod.out,
                        "foo");
  run_test();
 end
endmodule
```

```
#include "uvmc.h"
using namespace uvmc;
#include "consumer.h"

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

TLM Connection – SV-SV using UVMC

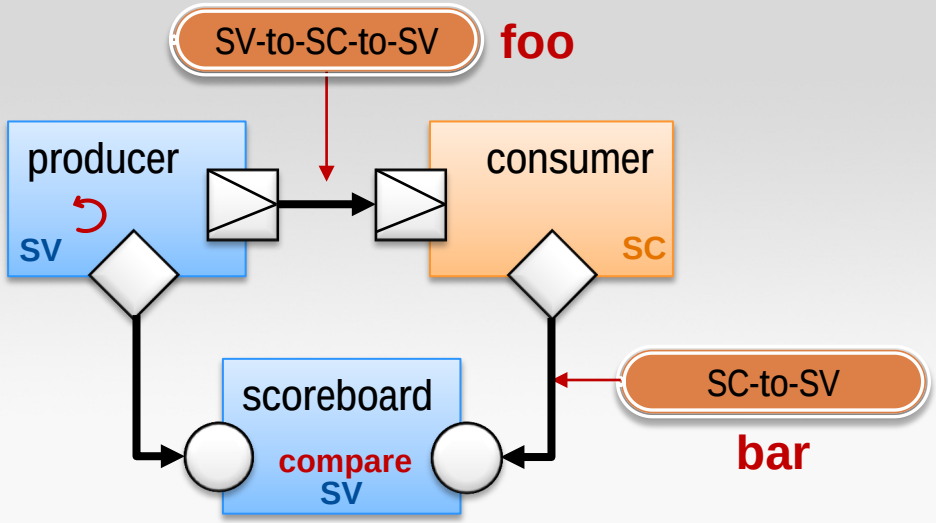
```
module sv_main;
  producer prod = new("prod");
  consumer cons = new("cons");

  initial begin

    uvmc_tlm #()::connect(prod.out, "sv2sv");
    uvmc_tlm #()::connect(cons.in,  "sv2sv");

    run_test();
  end
endmodule
```



match lookup strings

TLM Connection – SV using UVMC

Add a scoreboard, add a connection.

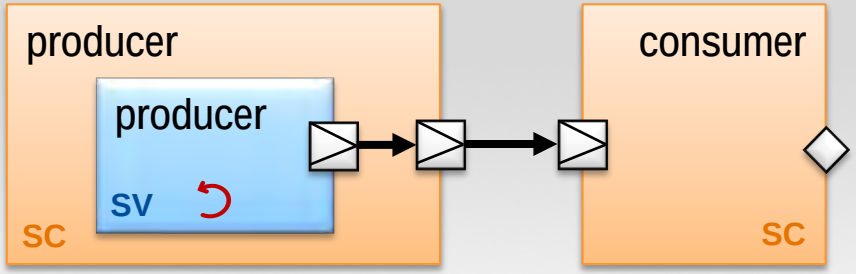



```
import uvm_pkg::*;
import uvmc_pkg::*;
`include "producer.sv"
`include "scoreboard.sv"

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
```

```
#include "uvmc.h"
using namespace uvmc;
#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;
}
```

Hierarchical Connection (SC wraps SV)

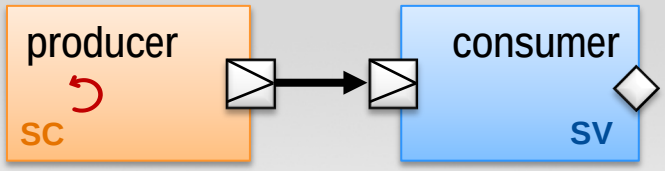
```
#include "systemc.h"
#include "tlm.h"
#include "simple_initiator_socket.h"

using namespace sc_core;
using namespace tlm;
using namespace tlm_utils;

#include "consumer.h"

class producer: public sc_module {
  public:
  tlm_initiator_socket<32> out;
  SC_CTOR(producer) : out("out") {
    uvmc_connect_hier(out,"sv_out");
  }
};

int sc_main(int argc, char* argv[])
{
  producer prod("producer");
  consumer cons("consumer");
  prod.out.bind(cons.in);
  sc_start();
  return 0;
};
```



Looks like pure SC testbench.
SC producer implemented as SV component.

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

module sv_main;
  producer prod = new("prod");
  initial begin
    uvmc_tlm #(uvm_tlm_generic_payload)::
       connect(prod.out,"sv_out");
    run_test();
  end

endmodule
```

TLM Connection – UVM-Aware SC → SV




```
#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;
}
```

extend base producer

background thread

raise objection, wait for “done”, drop objection

```
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

Summary

- **Reviewed TLM components**
  - Show implementations of simple SC & SV producers and consumers
- **Reviewed Native TLM connections**
  - Review how to make *native* component connections in SC and SV
- **Introduced the UVMC connect functions**
  - For registering TLM ports, exports, and imps for cross-language communication
- **Demonstrated how to use UVMC connect functions**
  - Point-to-point connections
  - Hierarchical connections (e.g. SC wraps SV)

UVM Connect Presentation Series

- **Part 1 – UVMC Introduction**
  - Learn what UVMC is and why you need it
  - Review the principles behind the TLM1 and TLM2 standards
  - Review basic port/export/interface connections in both SC and SV
- **Part 2 – UVMC Connections**
  - Learn how to establish connections between TLM-based components in SC and SV
- **Part 3 – UVMC Converters**
  - Learn how to write the converters that are needed to transfer transaction data across the language boundary
- **Part 4 – UVMC Command API**
  - Learn how to access and control key aspects of UVM simulation from SystemC

Mentor Graphics®

UVM Connect

Part 2 – Connections

Adam Erickson
Verification Technologist

academy@mentor.com
www.verificationacademy.com

VERIFICATION ACADEMY