UVM and SystemC Transactions – An Update

Dr David Long, Doulos
Agenda

• TLM in UVM and SystemC
• UVM Connect
• UVM-ML
• TLM1 Producer-Consumer
• TLM-2.0
• Future developments?
• Conclusions
Transaction-Level Communication

Simulate every event!

100-10,000 X faster simulation!
Transactions within UVM

- **Agent**
  - uvm_analysis_port
  - sequencer
    - uvm_seq_item_pull_export
  - driver
    - uvm_seq_item_pull_port

- **Subscriber**
  - uvm_analysis_port
  - UVM Analysis Port

- **Monitor**
  - uvm_analysis_port
  - UVM Analysis Port
UVM Execution Phases

A consistent temporal structure

- build
- connect
- end_of_elaboration
- start_of_simulation
- run
- extract/check/report
- final

pre_reset
reset
post_reset
pre_configure
configure
post_configure
pre_main
main
post_main
pre_shutdown
shutdown
post_shutdown
TLM1 Ports, Exports and Imps

```verilog
uvm_blocking_put_port #(tx_t) p1;

uvm_blocking_put_export #(tx_t) e1;

uvm_blocking_put_imp #(tx_t) e1;

Producer

Port

p1.put(tx);

Consumer

Export

Imp

task put(input my_tx tx);
... endtask
```
TLM-2.0 Sockets

- transactions use generic_payload
- nb_transport_bw only called when nb_transport_fw used
- SystemC sockets support both blocking and non-blocking
UVM Initiator and Target Sockets

• Blocking

• Non-blocking
Cross-Language Transactions

SystemC

- Initiator
  - Transaction object
  - do_pack

DPI Layer

- Ports associated using names (string)

SystemVerilog

- Target
  - Transaction object
  - do_unpack

Copy of original transaction

Transaction object
Agenda

- TLM in UVM and SystemC
- UVM Connect
- UVM-ML
- TLM1 Producer-Consumer
- TLM-2.0
- Future developments?
- Conclusions
UVM Connect

• Library of classes and functions that enable connection and synchronization of SystemVerilog and SystemC components via
  – TLM1 ports and exports
  – TLM-2.0 sockets using the generic payload
  – Analysis ports and exports

• Automatic and user-defined converters for transaction classes

• Based on standard SystemVerilog DPI-C so works with simulators from multiple vendors

• Developed by Mentor Graphics but released under Apache 2.0 License with all source code
TLM Connections with UVM Connect

```
UVM Target

function void connect_phase(uvm_phase phase);
    uvmc_tlm #(): connect(target1.in, "socket");
endfunction
```

```
SystemC

uvmc_connect(initiator.out, "socket");
```

SystemC

SystemVerilog UVM

![Diagram of SystemC Initiator connected to UVM Target](image-url)
Synchronization with UVM Connect

```cpp
uvmc_wait_for_phase("run", UVM_PHASE_STARTED);
uvmc_raise_objection("run");
...
sc_core::wait(run_done_ev);
uvmc_drop_objection("run");
```

```
initial
uvmc_init();
```
Agenda

• TLM in UVM and SystemC
• UVM Connect
• UVM-ML
• TLM1 Producer-Consumer
• TLM-2.0
• Future developments?
• Conclusions
UVM-ML

• Frameworks for SystemVerilog, SystemC and e verification environments connected via a multi-language backplane

• Provides SystemC versions of UVM base classes – SystemC components automatically synchronized to UVM phase scheduler

• Automatic and user-defined converters for transaction classes

• Based on standard SystemVerilog DPI-C so works with simulators from multiple vendors

• Developed by Cadence but released under Apache 2.0 License with all source code
TLM Connections with UVM-ML

SystemC Initiator

out

UVM Target

in

UVM-ML SC adapter

UVM-ML SV adapter

UVM-ML Backplane
UVM-ML Port/Export/Socket Register and Connect Functions

SystemC

```c
ml_tlm2_register_initiator(this, initiator.out, "socket", this->name());
```

SystemVerilog UVM

```c
uvm_ml::ml_tlm2 #()::register(target1.in);
```

SystemVerilog UVM

```c
bit res = uvm_ml::connect("sys.prod.socket", target1.in.get_full_name());
```
Agenda

- TLM in UVM and SystemC
- UVM Connect
- UVM-ML
- TLM1 Producer-Consumer
- TLM-2.0
- Future developments?
- Conclusions
TLM1 Example

SystemC

```
sctop
   sc_main
      put_port
         producer
```

SystemVerilog

```
topmodule
   svtop
      put_export
         consumer
```

3/2/2022
enum dir_t { READ, WRITE };

class trans_t {
public:
    int addr;
    std::vector<unsigned char> data;
    dir_t dir;

    trans_t() {
        data.reserve(8);  //allocate space for 8 chars
    }
    virtual void do_print(std::ostream& os) const;
};
UVM Connect SystemC Converter

• Template specialization of uvmc_converter class is required for user-defined transaction classes

```cpp
template <>
struct uvmc_converter<trans_t> {
    static void do_pack(const trans_t &t,
        uvmc::uvmc_packer &packer) {
        packer << t.addr << t.data << t.dir;
    }
    static void do_unpack(trans_t &t,
        uvmc::uvmc_packer &packer) {
        packer >> t.addr >> t.data >> t.dir;
    }
};
```
class producer : public sc_module {
public:
    sc_port<tlm::tlm_blocking_put_if<trans_t> > put_port;

void run() {
    trans_t * t;
    sc_core::wait(10, SC_NS);
    ...
    put_port->put(*t); ...
}

void run_proc() {
    uvmc_raise_objection("run");
    run();
    uvmc_drop_objection("run");
}
UVM Connect SystemVerilog
Transaction Class

class trans_t extends uvm_sequence_item;
  `uvm_object_utils(trans_t)
typedef enum {READ,WRITE} dir_t;
rand int addr;
rand byte data[];
rand dir_t dir;

function new(string name=""); ...
function void do_pack(uvm_packer packer); ...
function void do_unpack(uvm_packer packer); ...

dofunction function void do_pack(uvm_packer packer);
  `uvm_pack_int(addr)
  `uvm_pack_queue(data)
  `uvm_pack_enum(dir)
endfunction: do_pack
class consumer extends uvm_component;
    uvm_blocking_put_imp #(trans_t,consumer) put_export;
    `uvm_component_utils(consumer)

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

    task put (input trans_t p);
        `uvm_info(get_type_name(), ... ,UVM_LOW);
        #1ns;
    endtask

endclass
#include "uvm.h"
enum dir_t { READ, WRITE };

class trans_t : public uvm::uvm_object {
public:
    UVM_OBJECT_UTILS(trans_t)
    ...
    //attributes and constructor same as the UVMC version

    virtual void do_pack(uvm::uvm_packer &packer) const;
    virtual void do_unpack(uvm::uvm_packer &packer);
    virtual void do_print(std::ostream& os) const;
    virtual void do_copy(const uvm::uvm_object* rhs);
    virtual bool do_compare(const uvm::uvm_object* rhs) const;
};

UVM_OBJECT_REGISTER(trans_t)

Register class with backplane
class producer : public uvm::uvm_component {
  public:
    sc_port<tlm::tlm_blocking_put_if<trans_t> > put_port;

    producer(sc_core::sc_module_name nm) :
        uvm_component(nm), put_port("put_port"){
        // nothing to do in constructor!
    }

    UVM_COMPONENT_UTILS(producer)
}

void run() {
  trans_t * t;
  sc_core::wait(10, SC_NS);
  ...
  put_port->put(*t); ...

  // SystemC SC_THREAD
  // Synchronized with UVM run_phase

class sctop : public sc_core::sc_module {
public:
    producer prod;
    sctop(sc_core::sc_module_name nm) : sc_module(nm), prod("prod") {
        uvm_ml::uvm_ml_register(&prod.put_port);
    }
    void before_end_of_elaboration() {
        uvm_ml::uvm_ml_connect(prod.put_port.name(),
                                "svtop.consumer.put_export");
    }
};

int sc_main(int argc, char** argv) {
    return 0;
}

UVM_ML_MODULE_EXPORT(sctop)
UVM_COMPONENT_REGISTER(producer)

Alternatively, can call connect function from SV

Note empty sc_main
Transaction and Consumer classes same as UVMC example

```verilog
class svtop extends uvm_env;
  consumer cons;
  function void phase_ended(uvm_phase phase);
    if (phase.get_name() == "build")
      uvm_ml::ml_tlm1#(trans_t)::register(cons.put_export);
  end
endfunction

module topmodule;
initial begin
  string tops[2];
  tops[0] = "SC:sctop";
  tops[1] = "SV:svtop";
  uvm_ml_run_test(tops, "");
end
```

Table contains names of top-level modules

Note: Not UVM run_test()
Agenda

• TLM in UVM and SystemC
• UVM Connect
• UVM-ML
• TLM1 Producer-Consumer
• TLM-2.0
• Future developments?
• Conclusions
TLM-2.0 Blocking Transport

- UVM Connect and UVM-ML both support TLM-2.0

- Converting TLM1 producer-consumer to blocking transport is trivial
  - port/export replaced by initiator/target socket
  - transaction class replaced by generic_payload
  - do_pack/do_unpack for generic_payload built-in
  - TLM1 connect/register functions replaced by TLM-2.0 versions

- Transaction object automatically copied between SC and SV at transaction start and end
TLM-2.0 Non-Blocking Transport

- Transaction lifetime extends over multiple phases of "base protocol" (multiple calls to nb_transport)

- Standard requires a memory manager (not provided by UVM)
  - No memory manager in UVM Connect. SV transactions garbage collected when phase reaches END_RESP
  - UVM-ML memory manager requires full base protocol to be used (may get memory leaks with early completion)

- Complex scenarios (pipelined transactions, payload event queues) require custom code (and great care)
Agenda

- TLM in UVM and SystemC
- UVM Connect
- UVM-ML
- TLM1 Producer-Consumer
- TLM-2.0
- Future developments?
- Conclusions
Future Developments?

• UVM-SystemC
  – A complete implementation of UVM class hierarchy in SystemC
  – Should simplify integration of SC and SV UVM
  – Still need a mechanism to synchronize phases and pass transactions

• Multi-Language Working Group
  – A standard, open-source architecture to integrate SystemVerilog and SystemC UVM?
Agenda

• TLM in UVM and SystemC
• UVM Connect
• UVM-ML
• TLM1 Producer-Consumer
• TLM-2.0
• Future developments?
• Conclusions
Conclusions

• UVM Connect and UVM-ML both provide portable solutions for integration of SystemC and UVM TLM models
  – It is no longer necessary to create your own solution!

• UVM Connect
  – Easiest to learn and create first examples

• UVM-ML
  – More powerful and closer integration with UVM
  – More support for TLM-2.0 models
Further Information

• Additional details given in the paper

• Source code: see Doulos website after DVCon

www.doulos.com/knowhow

• Any Questions?