# UVM Connect

## *Part 4 – UVM Command API*

*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 estblish 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 Command Menu

- **Phasing**
  - uvmc_wait_for_phase
  - uvmc_raise_objection
  - uvmc_drop_objection
- **Configuration**
  - uvmc_set_config_int
  - uvmc_set_config_object
  - uvmc_set_config_string
  - uvmc_get_config_int
  - uvmc_get_config_object
  - uvmc_get_config_string

Most functions have same prototypes as in UVM with uvmc_ prefix

- **Factory**
  - uvmc_print_factory
  - uvmc_set_factory_inst_override
  - uvmc_set_factory_type_override
  - uvmc_debug_factory_create
  - uvmc_find_factory_override
- **Reporting**
  - uvmc_print_topology
  - uvmc_set_report_verbosity
  - uvmc_report_enabled
  - uvmc_report
- **Topology**
  - uvmc_print_topology

## UVM Command API Initialization

- **SV side – Call *uvmc_init* to start service**
  - Will likely be transparent in future release

```
module top;
  import uvmc_pkg::*;
  ...
  initial uvmc_init();
  ...
```

- **SC side – Import uvmc namespace**

```
#include “uvmc.h”
using namespace uvmc;
...
```

All functions in API must be called from SC threads, as they will block until SV is ready

# UVM Command API – Reporting

| | |
|---|---|
| **uvmc_set_report_verbosity** | **(level, context, recurse)** |
| | Configure the specified verbosity *level* at the given *context*, optionally *recursing* into children |

```
uvmc_set_report_verbosity(UVM_HIGH, "top.agent.*", 1);
```

SV context

| | |
|---|---|
| **uvmc_report_enabled** | **(verbosity, severity, id, context)** |
| | Returns true if the report of the given severity and id at the given context would not be filtered based on configured *verbosity* and action at the given *context* |
| **uvmc_report** | **(severity, id, message, verbosity, context, filename, line)** |
| | Issue a report with the given severity, ID, message, verbosity (if INFO), filename and line number. uvmc_report_info\|warning\|error\|fatal also available |

```
if (uvmc_report_enabled(UVM_MEDIUM, UVM_INFO, "SC_TX_RECV", "")) {
  uvmc_report(UVM_INFO, "SC_TX_RECV", $sformatf(...),
                        name(), __FILE__, __LINE__);
}
```

SC context

Can be expensive!

# UVM Command API – Reporting cont…

| | |
|---|---|
| **UVMC_INFO** | **(id, message, verbosity, context)** |
| **UVMC_WARNING**<br>**UVMC_ERROR**<br>**UVMC_FATAL** | **(id, message, context)** |
| | Calls uvmc_report_enabled first. If returns TRUE, call uvmc_report, passing SC-side filename (__FILE__) and line number (__LINE__). |

```
uvmc_set_report_verbosity(UVM_FULL, "", 1);

UVMC_INFO("MY_INFO-NONE","Some none message", UVM_NONE,   name());
UVMC_INFO("MY_INFO-LOW ","Some low  message", UVM_LOW,    name());
UVMC_INFO("MY_INFO-MED ","Some med  message", UVM_MEDIUM, name());
UVMC_INFO("MY_INFO-HIGH","Some high message", UVM_HIGH,   name());
UVMC_INFO("MY_INFO-FULL","Some full message", UVM_FULL,   name());

uvmc_set_report_verbosity(UVM_MEDIUM, "", 1);
```

All SC reports are issued in UVM at uvm_top (“”) context. Context argument is available in UVM report catchers via get_context() function (as of UVM 1.1b)

# UVM Command API – Configuration

| | |
|---|---|
| **uvmc_set_config_int**<br>**uvmc_set_config_string**<br><br>**uvmc_set_config_object** | (context, inst_name, field_name, value)<br><br><br>(type_name, context, inst_name, field_name, value) |
| | Set the integral, string, or uvm_object-based object *value* of the specified *field* at the context: {*context*, ".", *inst_name*}. If value is an object, the first argument specifies the *type_name* of the object. |

```
string s;
uint64 i;
prod_cfg_t cfg;

cfg.min_addr=0x100;
cfg.max_addr=0x200;

uvmc_set_config_int    ("e.prod", "", "some_int", 2);
uvmc_set_config_string ("", "e.prod", "some_str", “Hello from SC”);
uvmc_set_config_object ("prod_cfg_t", "e", "prod", "config", cfg);
```

## UVM Command API – Configuration cont

| | |
|---|---|
| **uvmc_get_config_int**<br>**uvmc_get_config_string**<br>**uvmc_get_config_object** | (context, inst_name, field_name, value)<br>(type_name, context, inst_name, field_name, value) |
| | Get the integral, string, or uvm_object-based value for for the specified field at the specified context |

```
i=0;
s="";
cfg.min_addr=0;
cfg.max_addr=0;

uvmc_get_config_int    ("e.prod",    "", "some_int", i);
uvmc_get_config_string ("e.prod",    "", "some_str", s);
uvmc_get_config_object ("prod_cfg", "e.prod", "", "config", cfg);
```

For set/get_config_object, equivalent object on SV-side must
- extend uvm_object
- be registered with factory by name

# UVM Command API – Factory

| | |
|---|---|
| **uvmc_print_factory** | (uvm_types) |
| | Prints the factory contents. If uvm_types is 1, prints the UVM base types in addition to user types. |
| **uvmc_find_factory_override** | (requested_type, context) |
| | Returns the string type name that would be produced for the given requested_type and context |
| **uvmc_debug_factory_create** | (requested_type, context) |
| | Simulate a factory request. Factory outputs detailed information on what type it would actually produce. |

```
// Print factory info
uvmc_print_factory();

// What type does factory create given a type and context?
actual_type = uvmc_find_factory_override("producer","e.prod");

// Show how factory arrives at that answer
uvmc_debug_factory_create("producer","e.prod");
```

# UVM Command API – Factory cont…

| | |
|---|---|
| **uvmc_set_factory_inst_override** | (requested_type, override_type, context) |
| | Tells factory to produce override_type in place of requested_type at the given context (hier path) |
| **uvmc_set_factory_type_override** | (requested_type, override_type, replace) |
| | Tells factory to produce override_type in place of requested_type. Instance overrides take precedence |

```
// Set a type override
uvmc_set_factory_type_override("producer","producer_ext","e.*");

// Printing factory info will now show new entry
uvmc_print_factory();

// Will return “producer_ext” now
actual_type = uvmc_find_factory_override("producer","e.prod");

// Debug info will show new override taking effect
uvmc_debug_factory_create("producer","e.prod");
```

## UVM Command API – Phasing

| | |
|---|---|
| **uvmc_wait_for_phase** | **(phase, state, op)** |
| | Waits for the phase to reach |

```
// from SC_THREAD in SC_MODULE...

uvmc_wait_for_phase("run", UVM_PHASE_STARTED);
```

# UVM Command API – Phasing cont…

| | |
|---|---|
| **uvmc_raise_objection** | (name, context, description, count) |
| | Raises count (1) number of objections to the phase name on behalf of the given context (uvm_top) using the specified description (“”) |
| **uvmc_drop_objection** | (name, context, description, count) |
| | Drops count (1) number of objections to the phase name on behalf of the given context (uvm_top) using the specified description (“”) |

```
UVMC_INFO("SC_TOP/RAISE_OBJ", (string(name()) +
    " raising objection to run phase").c_str(), UVM_MEDIUM,"");
uvmc_raise_objection("run", name(), "SC waiting 10ns");

// wait some delay to prove we are in control...
wait(sc_time(10,SC_NS));

UVMC_INFO("SC_TOP/DROP_OBJ", (string(name()) +
    " dropping objection to run phase").c_str(), UVM_MEDIUM,"");
uvmc_drop_objection("run", name(), "10ns has passed");
```

# UVM Command API – Printing Topology

| | |
|---|---|
| **uvmc_print_topology** | **( )** |
| | Configure the specified verbosity at the given context, optionally recursing into children |

```
cout << "Waiting for UVM to reach build phase..." << endl;
uvmc_wait_for_phase("build", UVM_PHASE_STARTED);

cout << "Topology before build phase:" << endl;
uvmc_print_topology();


uvmc_wait_for_phase("build", UVM_PHASE_ENDED);

cout << "Topology after build phase:" << endl;
uvmc_print_topology();
```

## The Future

- **Reporting**
  - Register SC-side report catchers
  - Set report actions
  - Recording
- **Events & Barriers**
  - SC-side access to global event pools
  - Reflect some of API to SC
- **Resource DB**
  - Access to resource DB
  - Setting/getting resources by name
- **User Driven**
  - Special requests? Let us know.
  - You have source code. Enhance to suit.

# 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 estblish 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 Connect

## *Part 4 – UVM Command API*

*Adam Erickson*
*Verification Technologist*

*academy@mentor.com*
*www.verificationacademy.com*

VERIFICATION ACADEMY