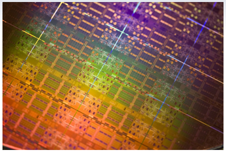


UVM Test Environment for the

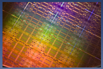
Common Modular Architecture

Static Limitations of the UVM Class Library

Marcel Alsdorf

14. November 2013

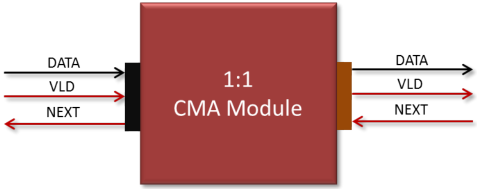
Beams Department, CERN



Chapter 1

Overview of CMA Modules

1:1 CMA Module

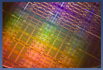


CMA Interface (CMI)

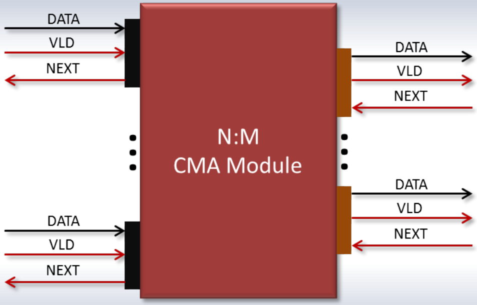
- **DATA** – the actual information
- **VLD/NEXT** – handshake between modules

Parameters

- **inp_data_size** – size of the input data
- **out_data_size** – size of the output data



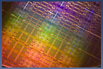
N:M CMA Module



Parameters

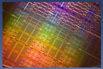
- **inp_data_size**[] – diff. sizes of the input data
- **out_data_size**[] – diff. sizes of the output data
- N – # of CMI Inputs
- M – # of CMI Output

UVM Environment for CMA Modules



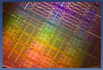
Chapter 2

UVM Environment for CMA Modules

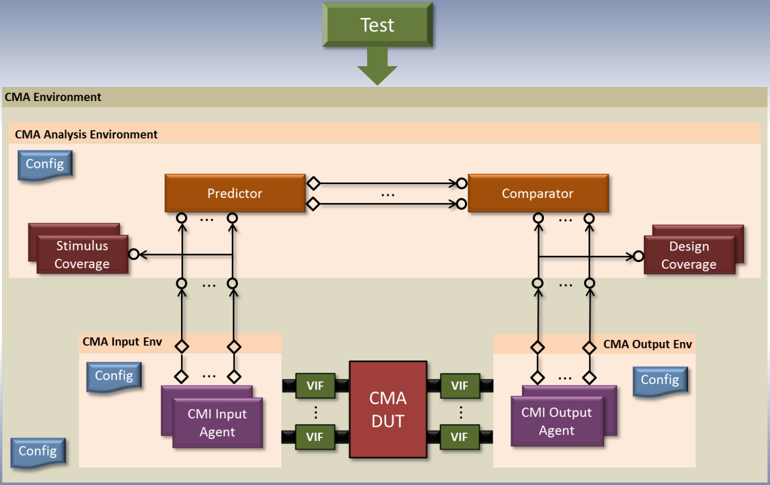


Goals

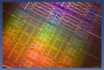
1. Able to verify any CMA module independent of internal functionality using the **same** Test Environment
2. Include the degrees of freedom of CMA modules (**parameters**) as part of the Test Environment
3. Decrease the necessary **changes/adaptations** in extended classes with every module test to a mimimum



General Overview

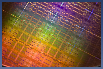


UVM Environment for CMA Modules – Virtual Interfaces

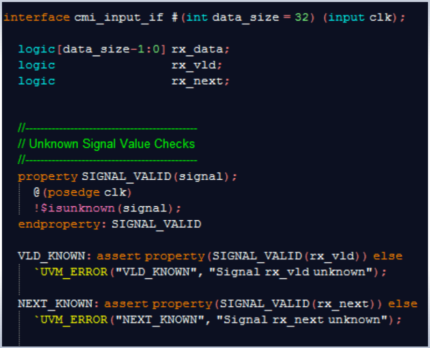


Chapter 2.1

Virtual Interfaces



CMI Input Virtual IF



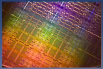
```
interface cmi_input_if #(int data_size = 32) (input clk);

  logic[data_size-1:0] rx_data;
  logic                rx_vld;
  logic                rx_next;


  //---------------------------------------------
  // Unknown Signal Value Checks
  //---------------------------------------------
  property SIGNAL_VALID(signal);
    @(posedge clk)
    !$isunknown(signal);
  endproperty: SIGNAL_VALID

  VLD_KNOWN: assert property(SIGNAL_VALID(rx_vld)) else
    `UVM_ERROR("VLD_KNOWN", "Signal rx_vld unknown");

  NEXT_KNOWN: assert property(SIGNAL_VALID(rx_next)) else
    `UVM_ERROR("NEXT_KNOWN", "Signal rx_next unknown");
```

General

- connects the class world with the module world
- is an actual SV interface

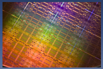
Protocol Assertions

- Unknown Signal Values Checks
- Invalid States Checks
- Timing Relationship Checks



Chapter 2.2

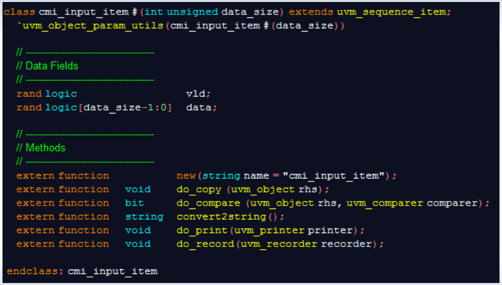
Sequence Items



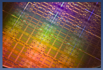
CMA Input Item

Specifics

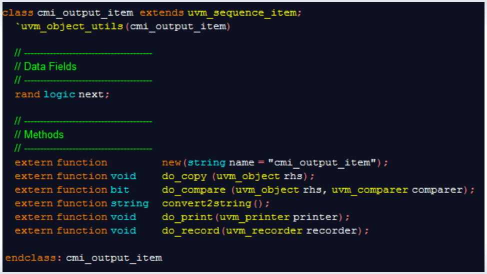
- used by the input agent's sequencer and driver
- is a parameterized class (factory registration different)
- defines the typical sequence item methods



```
class cmi_input_item #(int unsigned data_size) extends uvm_sequence_item;
  `uvm_object_param_utils(cmi_input_item #(data_size))

  // ----------------------------------------
  // Data Fields
  // ----------------------------------------
  rand logic                     vld;
  rand logic[data_size-1:0]      data;

  // ----------------------------------------
  // Methods
  // ----------------------------------------
  extern function           new(string name = "cmi_input_item");
  extern function   void    do_copy (uvm_object rhs);
  extern function   bit     do_compare (uvm_object rhs, uvm_comparer comparer);
  extern function   string  convert2string();
  extern function   void    do_print(uvm_printer printer);
  extern function   void    do_record(uvm_recorder recorder);

endclass: cmi_input_item
```



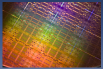
CMA Output Item



```
class cmi_output_item extends uvm_sequence_item;
  `uvm_object_utils(cmi_output_item)

  // ------------------------------------
  // Data Fields
  // ------------------------------------
  rand logic next;

  // ------------------------------------
  // Methods
  // ------------------------------------
  extern function        new(string name = "cmi_output_item");
  extern function void   do_copy (uvm_object rhs);
  extern function bit    do_compare (uvm_object rhs, uvm_comparer comparer);
  extern function string convert2string();
  extern function void   do_print(uvm_printer printer);
  extern function void   do_record(uvm_recorder recorder);

endclass: cmi_output_item
```

Specifics

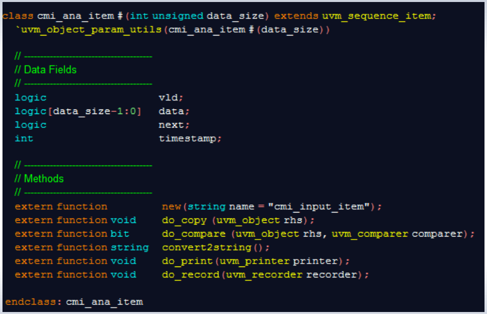
- used by the output agent's sequencer and driver
- is actually not a parameterized class (no control over data lines)
- defines the typical sequence item methods



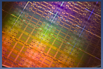
CMA Analysis Item

Specifics

- used by the input/output agent monitors and by the Analysis Environment
- is a parameterized class
- defines the typical sequence item methods

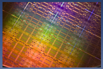


```
class cmi_ana_item #(int unsigned data_size) extends uvm_sequence_item;
  `uvm_object_param_utils(cmi_ana_item #(data_size))

  // ----------------------------------------
  // Data Fields
  // ----------------------------------------
  logic                   vld;
  logic[data_size-1:0]    data;
  logic                   next;
  int                     timestamp;

  // ----------------------------------------
  // Methods
  // ----------------------------------------
  extern function          new(string name = "cmi_input_item");
  extern function void     do_copy (uvm_object rhs);
  extern function bit      do_compare (uvm_object rhs, uvm_comparer comparer);
  extern function string   convert2string();
  extern function void     do_print(uvm_printer printer);
  extern function void     do_record(uvm_recorder recorder);

endclass: cmi_ana_item
```

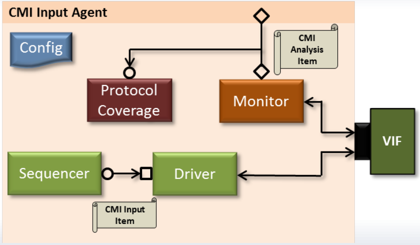


Chapter 2.3

Components

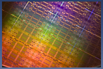


CMI Input Agent



Subclass Tasks

- **Sequencer** – runs sequences based on CMI Input Items
- **Driver** – communicates appropriately with the VIF
- **Monitor** – monitors the VIF and forwards a CMI Analysis Item
- **Protocol Coverage** – covers all State Transitions etc.
- **Config** – configuration object of the agent



UVM Environment for CMA Modules – Components

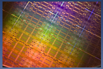
CMI Input Agent

```
class cmi_input_sequencer #(int data_size) extends uvm_sequencer #(cmi_input_item#(data_size));
  `uvm_component_param_utils(cmi_input_sequencer #(data_size))
```

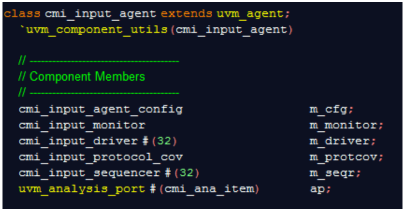
```
class cmi_input_driver #(int data_size) extends uvm_driver #(cmi_input_item#(data_size));
  `uvm_component_param_utils(cmi_input_driver #(data_size))
```

```
class cmi_input_monitor #(int data_size) extends uvm_component #(cmi_ana_item#(data_size));
  `uvm_component_param_utils(cmi_input_monitor #(data_size))
```

```
class cmi_input_protocol_cov #(int data_size)  extends uvm_subscriber #(cmi_ana_item #(data_size);
  `uvm_component_param_utils(cmi_input_protocol_cov #(data_size)
```



CMI Input Agent



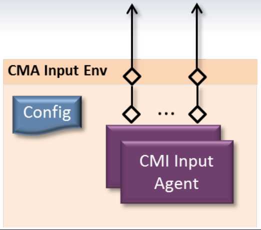
```
class cmi_input_agent extends uvm_agent;
  `uvm_component_utils(cmi_input_agent)

  // ------------------------------------
  // Component Members
  // ------------------------------------
  cmi_input_agent_config                    m_cfg;
  cmi_input_monitor                         m_monitor;
  cmi_input_driver #(32)                    m_driver;
  cmi_input_protocol_cov                    m_protcov;
  cmi_input_sequencer #(32)                 m_seqr;
  uvm_analysis_port #(cmi_ana_item)         ap;
```

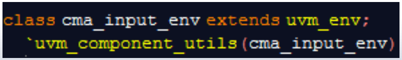
Agent Tasks

- receives parameters through the config object
- factory-creates sub-components (<name>::create ...)
- connects sub-components with each other
- creates ports and connects them with sub-components

UVM Environment for CMA Modules – Components

CMA Input Environment

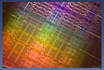




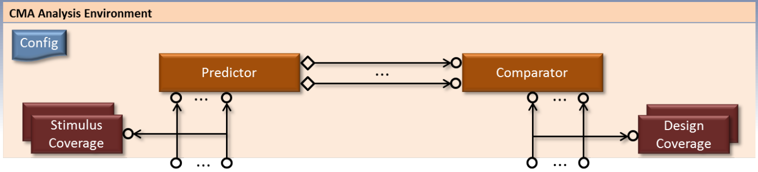
```
class cma_input_env extends uvm_env;
  `uvm_component_utils(cma_input_env)
```

Tasks

- gets the parameters from its config object
- creates config objects for the sub-components (agents)
- factory-creates the agents
- creates and connects ports with agents



CMA Analysis Environment

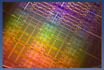


Input Subclass Tasks

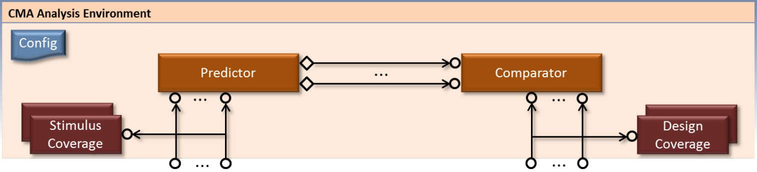
- **Stimulus Coverage** – module dependent coverage points concerning module stimulus
- **Predictor** – user-defined model of the DUT behaviour

Output Subclass Tasks

- **Design Coverage** – module dependent coverage points concerning module results
- **Comparator** – compares projection with reality



CMA Analysis Environment



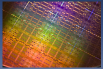
Input Subclass Tasks

- **Stimulus Coverage** – module dependent coverage points concerning module stimulus
- **Predictor** – user-defined model of the DUT behaviour

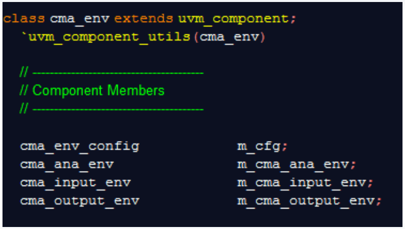
Output Subclass Tasks

- **Design Coverage** – module dependent coverage points concerning module results
- **Comparator** – compares projection with reality

Each class has to be user-extended depending on Module functionality



CMA Environment

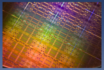


```
class cma_env extends uvm_component;
  `uvm_component_utils(cma_env)

  // -------------------------------------
  // Component Members
  // -------------------------------------

  cma_env_config              m_cfg;
  cma_ana_env                 m_cma_ana_env;
  cma_input_env               m_cma_input_env;
  cma_output_env              m_cma_output_env;
```

Tasks

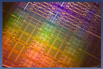
- highest-level environment
- receives parameter through config objects
- creates config objects for sub-enviroments
- builds sub-environments and connects them

UVM Environment for CMA Modules – Sequences



Chapter 2.4

Sequences



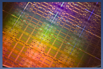
Overview

Standard Sequences

- are objects and therefore they are not part of the initial phasing (build/connect)
- created and destroyed during runtime
- are running on a sequencer
- utilize sequence items as communication objects
- configuring sequences through the config_db is limited and only possible through a sequencer

Virtual Sequences

- are objects as well
- distributes, creates and destroys other sequences
- can run on a virtual sequencer (not recommended)
- their interactions are defined in a body() method
- can not receive informations from the config_db

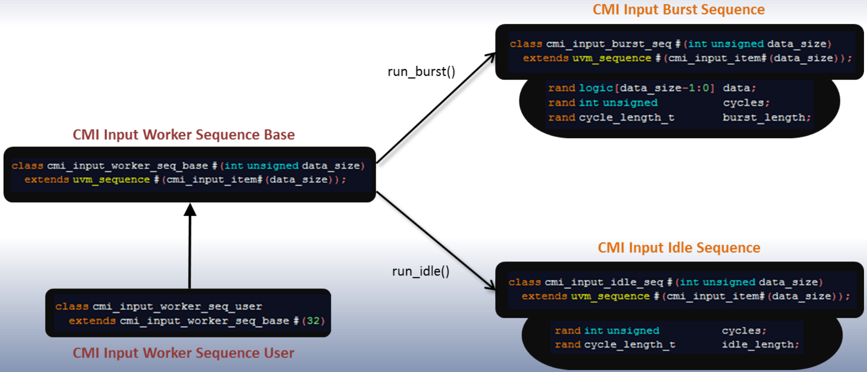


UVM Environment for CMA Modules – Sequences

CMI Input Sequence Hierarchy

Worker Sequences
(part of the Sequence Package)

API Sequences
(part of the Agent Package)



CMI Input Burst Sequence

```
class cmi_input_burst_seq #(int unsigned data_size)
  extends uvm_sequence #(cmi_input_item#(data_size));
    rand logic[data_size-1:0]  data;
    rand int unsigned          cycles;
    rand cycle_length_t        burst_length;
```

CMI Input Worker Sequence Base

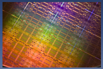
```
class cmi_input_worker_seq_base #(int unsigned data_size)
  extends uvm_sequence #(cmi_input_item#(data_size));
```

CMI Input Idle Sequence

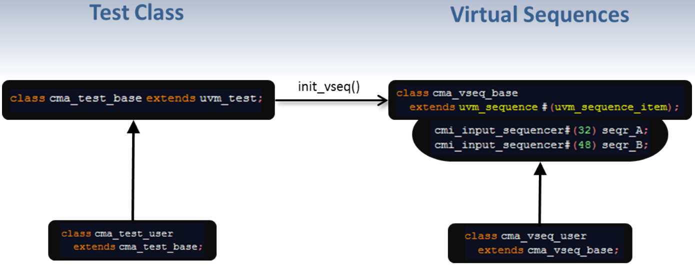
```
class cmi_input_idle_seq #(int unsigned data_size)
  extends uvm_sequence #(cmi_input_item#(data_size));
    rand int unsigned          cycles;
    rand cycle_length_t        idle_length;
```

```
class cmi_input_worker_seq_user
  extends cmi_input_worker_seq_base #(32)
```

CMI Input Worker Sequence User



CMI Input Sequence Hierarchy



```
class cma_test_base extends uvm_test;
```

```
class cma_test_user
  extends cma_test_base;
```

```
class cma_vseq_base
  extends uvm_sequence #(uvm_sequence_item);
```

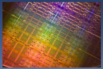
```
cmi_input_sequencer#(32) seqr_A;
cmi_input_sequencer#(48) seqr_B;
```

```
class cma_vseq_user
  extends cma_vseq_base;
```

Tasks
- create vseq and call init_vseq()
- start virtual sequence

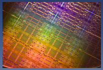
Tasks
- create all sequences
- start sequences on sequencer

UVM Environment for CMA Modules – Configuration



Chapter 2.5

Configuration

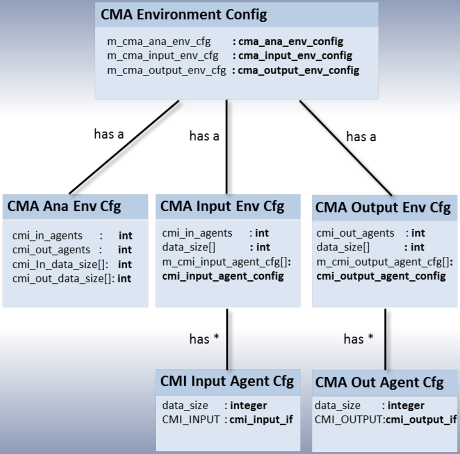


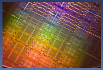
Overview

- convenience layer on top of the resource_db (adds hierarchical path as scope)
- using the resource_db is not recommended
- should be used to transfer virtual interface pointers and configuration information to **components** during initial phasing
- this should be done mainly by using configuration objects
- can be used dynamically during the run_phase (**objects**)
- but calling set() or get() at runtime is expensive and should be avoided
- therefore using the config_db in objects is not recommended

UVM Environment for CMA Modules – Configuration

CMA Component Configuration Tree





Configuring Sequences

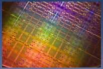
Should be avoided, but if necessary:

get_full_name()

```
uvm_config_db#(TYPE)::get(null, this.get_full_name(), field", field);
```

- sequences do not have hierarchy until they have been started on a sequencer
- once started, **get_full_name()** will return a hierarchy string for your sequence
- this string either includes the parent sequence's hierarchy or the hierarchy of the sequencer that the sequence was started on

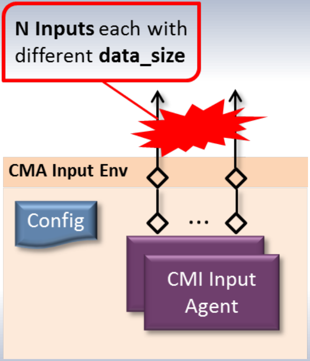
Static Limitations of UVM

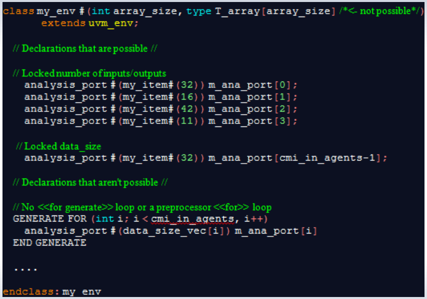


Chapter 3

Static Limitations of UVM

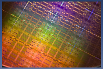
Analysis Items





```
class my_env #(int array_size, type T_array[array_size] /*<- not possible*/)
      extends uvm_env;

  // Declarations that are possible //

  // Locked number of inputs/outputs
    analysis_port #(my_item#(32)) m_ana_port[0];
    analysis_port #(my_item#(16)) m_ana_port[1];
    analysis_port #(my_item#(42)) m_ana_port[2];
    analysis_port #(my_item#(11)) m_ana_port[3];

   // Locked data_size
    analysis_port #(my_item#(32)) m_ana_port[cmi_in_agents-1];

  // Declarations that aren't possible //

  // No <<for generate>> loop or a preprocessor <<for>> loop
  GENERATE FOR (int i; i < cmi_in_agents, i++)
    analysis_port #(data_size_vec[i]) m_ana_port[i]
  END GENERATE

  ....

endclass: my env
```

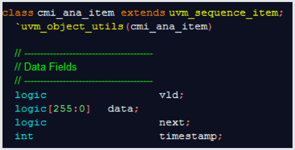
Static Limitations of UVM



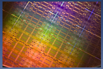
Analysis Items

1. Solution

Remove data_size from Items



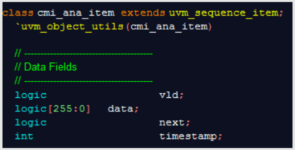
```
class cmi_ana_item extends uvm_sequence_item;
  `uvm_object_utils(cmi_ana_item)

  // ----------------------------------------
  // Data Fields
  // ----------------------------------------
  logic                vld;
  logic[255:0]    data;
  logic                next;
  int                  timestamp;
```



Analysis Items

1. Solution

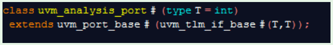
Remove data_size from Items



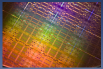
```
class cmi_ana_item extends uvm_sequence_item;
  `uvm_object_utils(cmi_ana_item)

  // ------------------------------------
  // Data Fields
  // ------------------------------------
  logic                   vld;
  logic[255:0]    data;
  logic                   next;
  int                     timestamp;
```

2. Solution

Create our own uvm_analysis_port



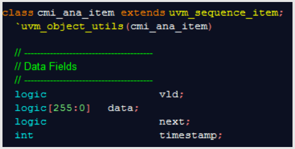
```
class uvm_analysis_port # (type T = int)
 extends uvm_port_base # (uvm_tlm_if_base #(T,T));
```



Analysis Items

1. Solution

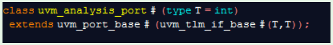
Remove data_size from Items



```
class cmi_ana_item extends uvm_sequence_item;
  `uvm_object_utils(cmi_ana_item)

  // ---------------------------------------
  // Data Fields
  // ---------------------------------------
  logic                 vld;
  logic[255:0]   data;
  logic                 next;
  int                   timestamp;
```

2. Solution

Create our own uvm_analysis_port

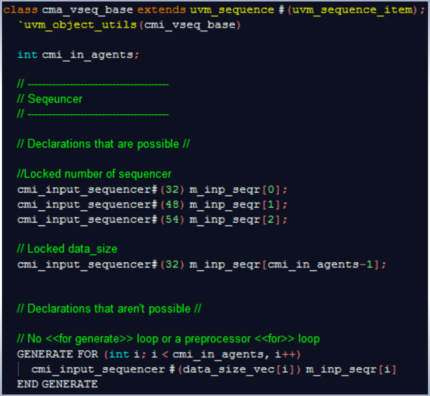


```
class uvm_analysis_port # (type T = int)
 extends uvm_port_base # (uvm_tlm_if_base #(T,T));
```

Both Solution are reasonably unclean and cumbersome

Static Limitations of UVM

Virtual Sequencer

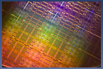

```
class cma_vseq_base extends uvm_sequence #(uvm_sequence_item);
  `uvm_object_utils(cmi_vseq_base)

  int cmi_in_agents;

  // ----------------------------------------
  // Seqeuncer
  // ----------------------------------------

  // Declarations that are possible //

  //Locked number of sequencer
  cmi_input_sequencer#(32) m_inp_seqr[0];
  cmi_input_sequencer#(48) m_inp_seqr[1];
  cmi_input_sequencer#(54) m_inp_seqr[2];

  // Locked data_size
  cmi_input_sequencer#(32) m_inp_seqr[cmi_in_agents-1];


  // Declarations that aren't possible //

  // No <<for generate>> loop or a preprocessor <<for>> loop
  GENERATE FOR (int i; i < cmi_in_agents, i++)
    cmi_input_sequencer #(data_size_vec[i]) m_inp_seqr[i]
  END GENERATE
```

Declaring Sequencers

- leads to same static declaration problem as seen before
- virtual sequence cannot receive informations from the config_db

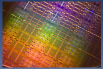
Solution

Rewrite the complete virtual sequence by hand for every test (no base class)



General Solution

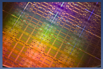
1. Factory ?
   - **Idea**: extend a non-parameterized base class with a parameterezied version
   - but an instance of a class can only be overwritten when it is being factory-created somewhere
   - **uvm_analysis_port** can not be factory created (not part of the hierarchy)
   - **sequences** are also actually not part of the hierarchy, but there are ways to bypass this limitations
   - **still** cumbersome and unclean



General Solution

1. Factory ?
    - **Idea**: extend a non-parameterized base class with a parameterezied version
    - but an instance of a class can only be overwritten when it is being factory-created somewhere
    - **uvm_analysis_port** can not be factory created (not part of the hierarchy)
    - **sequences** are also actually not part of the hierarchy, but there are ways to bypass this limitations
    - **still** cumbersome and unclean
2. Code Generator
    - **Result:** cleaner classes in the test environment
    - bypasses those static limitations of SV
    - possibilities to generate main structures of the user-extended classes

Static Limitations of UVM



General Solution

1. Factory ?
    - **Idea**: extend a non-parameterized base class with a parameterezied version
    - but an instance of a class can only be overwritten when it is being factory-created somewhere
    - **uvm_analysis_port** can not be factory created (not part of the hierarchy)
    - **sequences** are also actually not part of the hierarchy, but there are ways to bypass this limitations
    - **still** cumbersome and unclean
2. Code Generator
    - **Result:** cleaner classes in the test environment
    - bypasses those static limitations of SV
    - possibilities to generate main structures of the user-extended classes

Thank you for your attention!