February 24-26, 2009

Divide and conquer:

Techniques for creating a highly reusable stimulus generation process

Ning Guo

Paradigm Works

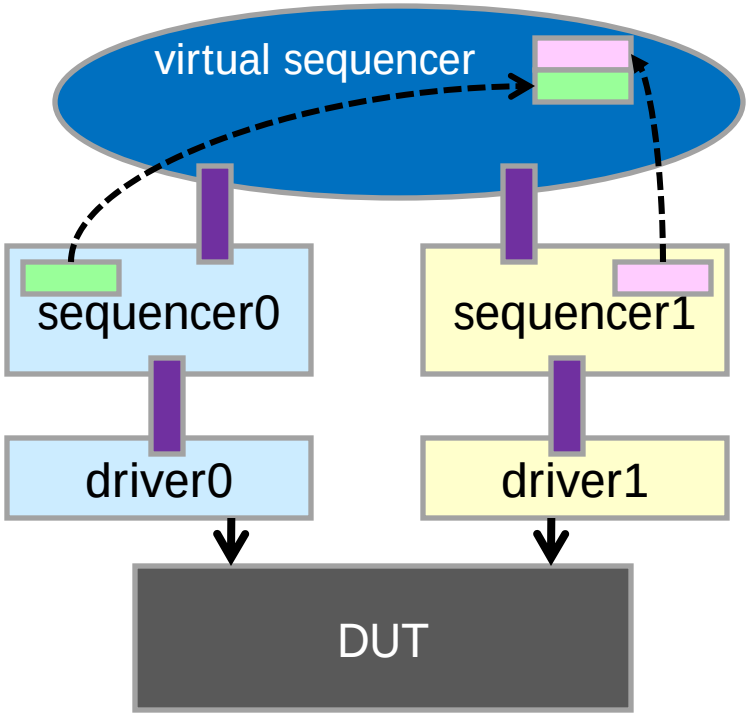
Outline

- Introduction
- Stimulus Representation
- Stimulus Generation
- Implementation
- Conclusion
- Future Work

Introduction

- State-of-the-art Stimulus Generation Methodology
  - OVM Sequence Generation

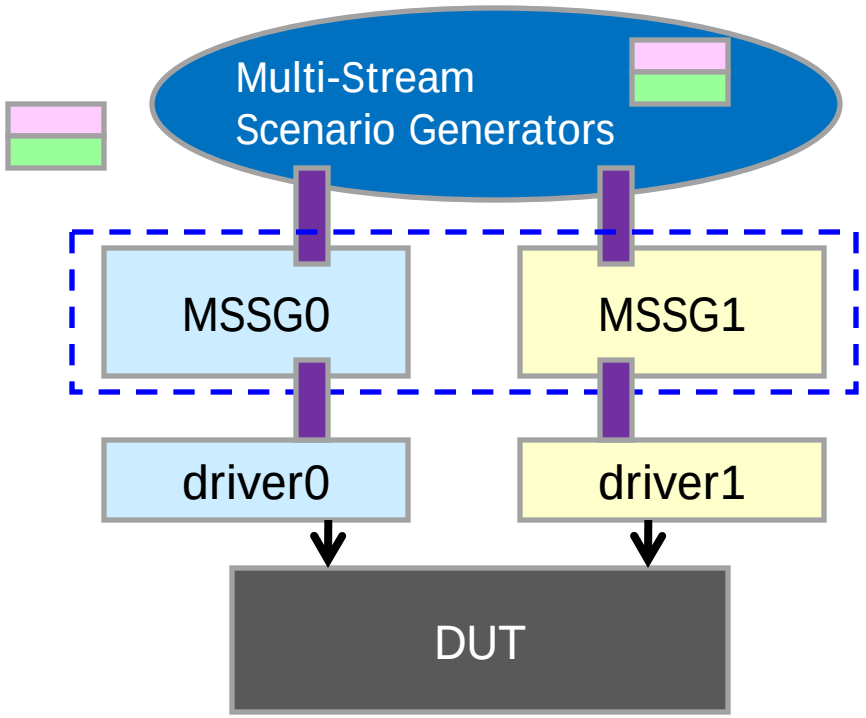
-Create driver(s)
-Create sequencer(s)
-Create sequence(s) for sequencer
-Create virtual sequencer
-Instantiate all sequences in virtual sequencer
-Connect virtual sequencer to sequencer(s)
-Virtual sequencer issues sequence to corresponding sequencer



Introduction

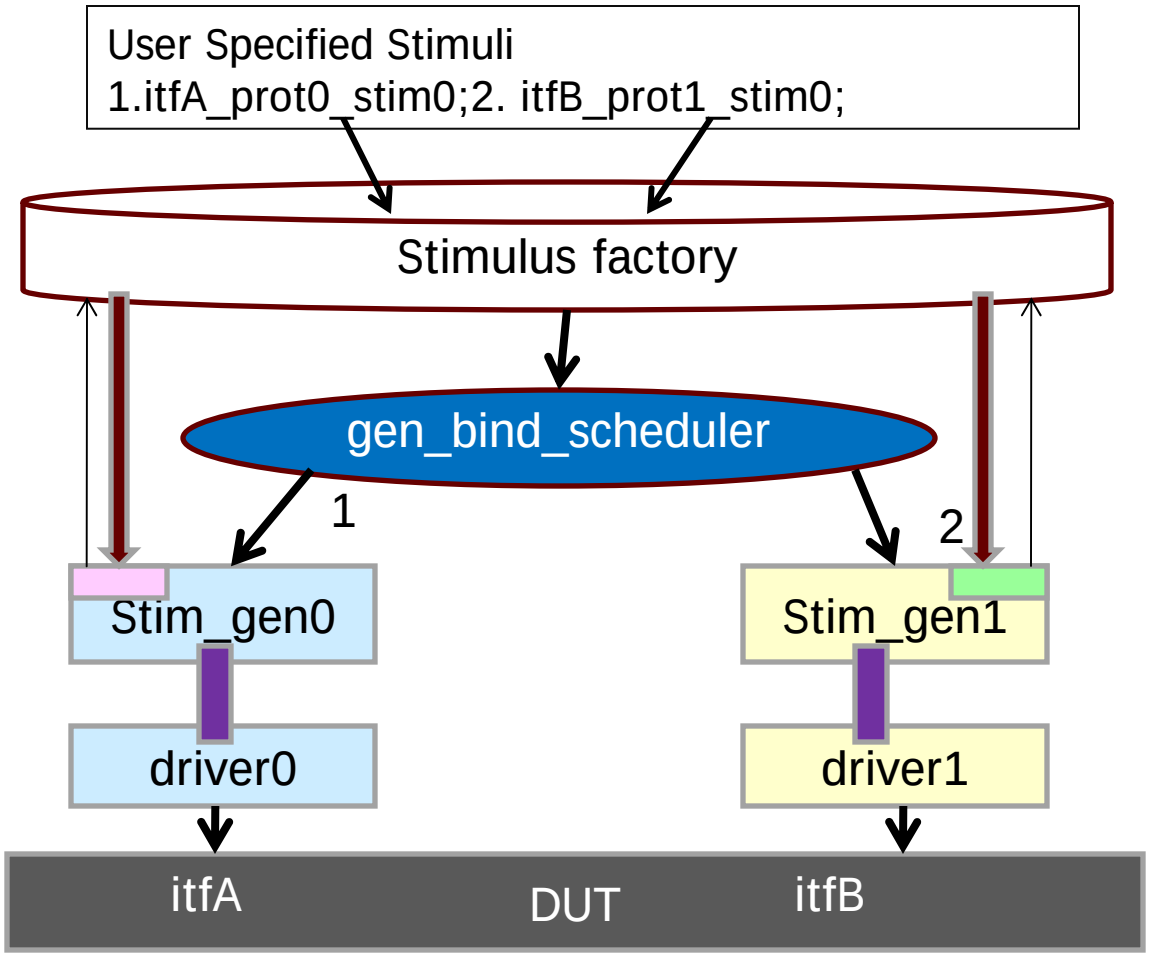
– VMM Multi-Stream Scenario Generation

-Create driver(s)
-Create lower level generator(s)
-Create scenarios
-Create multi-stream scenario generator (MSSG)
-Register scenarios with MSSG
-Register physical channels with MSSG
-MSSG issues scenarios to corresponding generator through proper channel

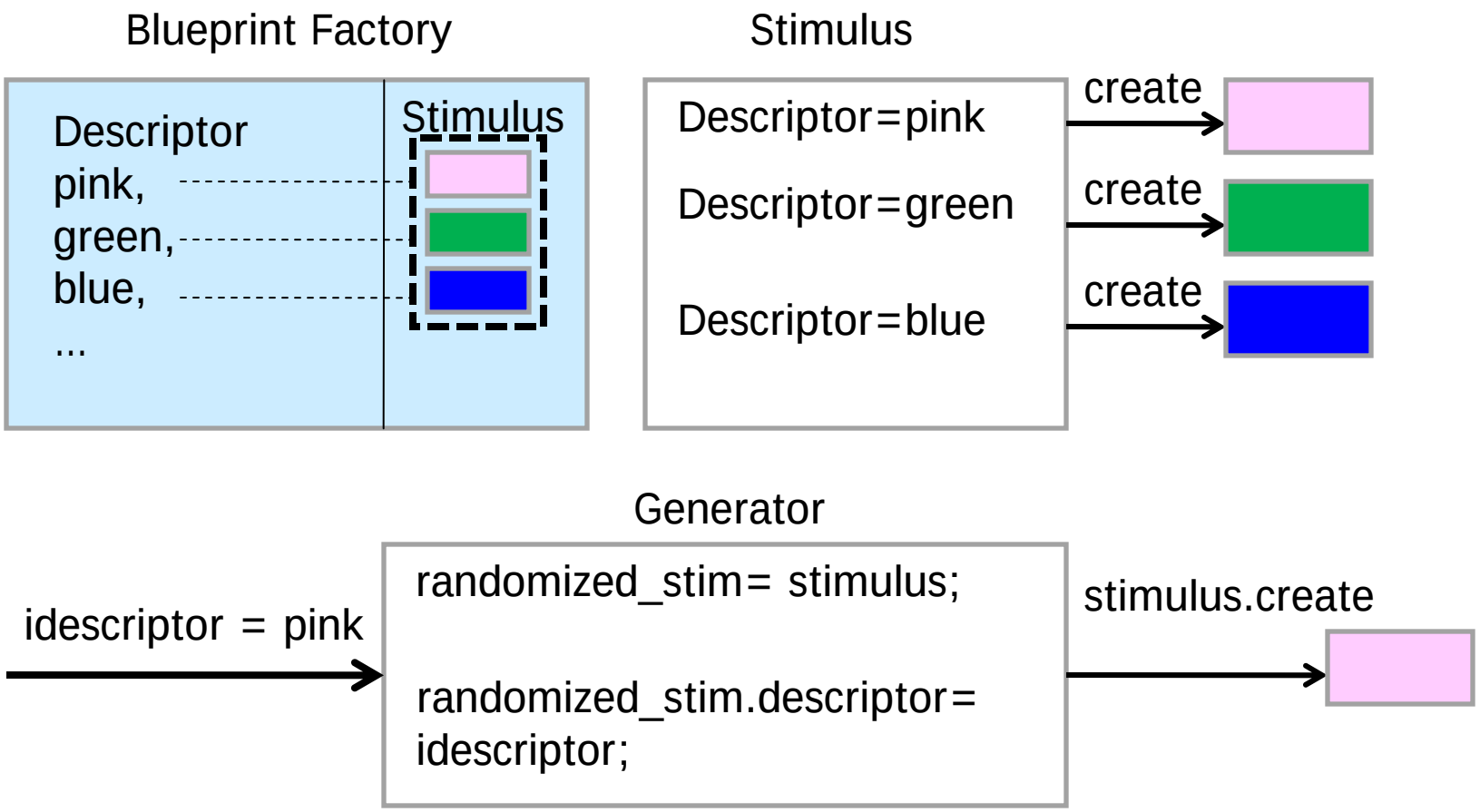


Introduction – *what is this?*

- Proposed Stimulus Generation Structure



Introduction – *what is this?*



Introduction – *summary*

- String based stimulus description
- Per interface Stimulus class
- Stimulus factory contains both descriptors and blueprint stimulus
- Generic factory-based stimulus generator
- Work with any methodology (VMM,OVM,etc.)

Outline

- Introduction
- Stimulus Representation
- Stimulus Generation
- Implementation
- Conclusion
- Future Work

Stimulus Representation

- Stimulus Descriptor

```
class stimulus_descriptor;
    string stim_kind;

    protected string itf_name;
    protected string prot_layer_name;
    protected string stim_name;

virtual function void parse_stim_kind();
function string set_stim_kind(string
ikind);
endclass
```

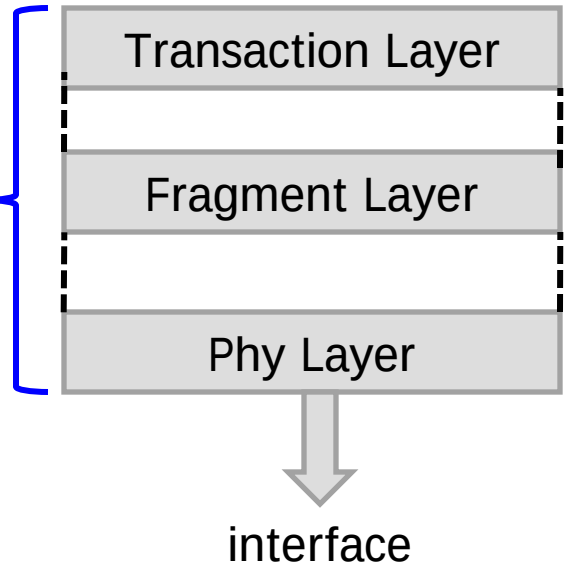
User defined

Protocol Layers

stim_kind=<Intf_ProtLayer_StimName>

itf_name prot_layer_name stim_name

stim_kind =
<Itf_ProtLayer_StimName>



Stimulus Representation *(continue...)*

- Base Stimulus

```
class base_stimulus extends <base_data>;
    stimulus_descriptor stim_descr;
    stimulus_factory    parent;

    <base_data> data4prot_layer[string];

    <base_data> transaction;

    virtual function void create_transaction();
    virtual function int convert2prot_layer(string ola
                                            base_stimulus ost
    virtual function int convert2itf(string oitf,
                                     base_stimulus ostim[$]);
    function void create_descr(stimulus_descriptor bp_descr);
    function void blueprint4prot_layer(string layer,
                                       <

    …
endclass
```

specify stimulus kind

back pointer

To hold blueprint data for each protocol layer

current transaction handle
Determined by create_transaction()

Stimulus transformation. User implement

Register stimulus descriptor

blueprinting

Stimulus Representation *(continue...)*

- Usage Example

```
//Interface A,protocol layer 0
class itfA_prot0_data extends <base_data>;
```

```
//Interface A,protocol layer 1
class itfA_prot1_data extends <base_data>;
```

```
class itfA_stimulus extends base_stimulus;
   itfA_prot0_data   prot0_data;
   itfA_prot1_data   prot1_data;
  …
endclass
```

Instantiate data type for different protocol layer

```
function new(string inst, stimulus factory parent);
   super.new(inst, parent);
   prot0_data = new;
   prot1_data = new;
   create_descr("itfA_prot0_rand");
   blueprint4prot_layer("prot0",prot0_data);
   create_descr("itfA_prot1_rand");
   blueprint4prot_layer("prot1",prot1_data);
endfunction
```

register descriptor with factory

add to blueprint array

Stimulus Representation *(continue...)*

- Usage Example

```
function void create_transaction();
   bit result;

   if (stim_descr.get_prot_layer_name() == “prot0” &
        stim_descr.get_stim_name() == “rand”) begin
      result = data4prot_layer[“prot0”].randomize();
      $cast(transaction, data4prot_layer[“prot0”].copy());
   end
   else if (stim_descr.get_prot_layer_name() == “proti
            stim_descr.get_stim_name() == “rand”) begin
      result = data4prot_layer[“prot1”].randomize();
      $cast(transaction, data4prot_layer[“prot1”].copy());
   end
endfunction
```

Create a transaction for a protocol layer using proper blueprint

Randomize blueprint

Return a copy to transaction

Stimulus Representation *(continue...)*

- Stimulus Factory – Store all stimulus kinds and stimulus blueprints

```
class stimulus_factory;
   string  defined_kind[$];
   itfA_stimulus  itfA_blueprint_stim;
   …
   virtual function base_stimulus create_stimulus(string
ikind);
   …
endclass
```

all defined stimulus kinds

blueprint stimulus

create a stimulus of 'ikind'

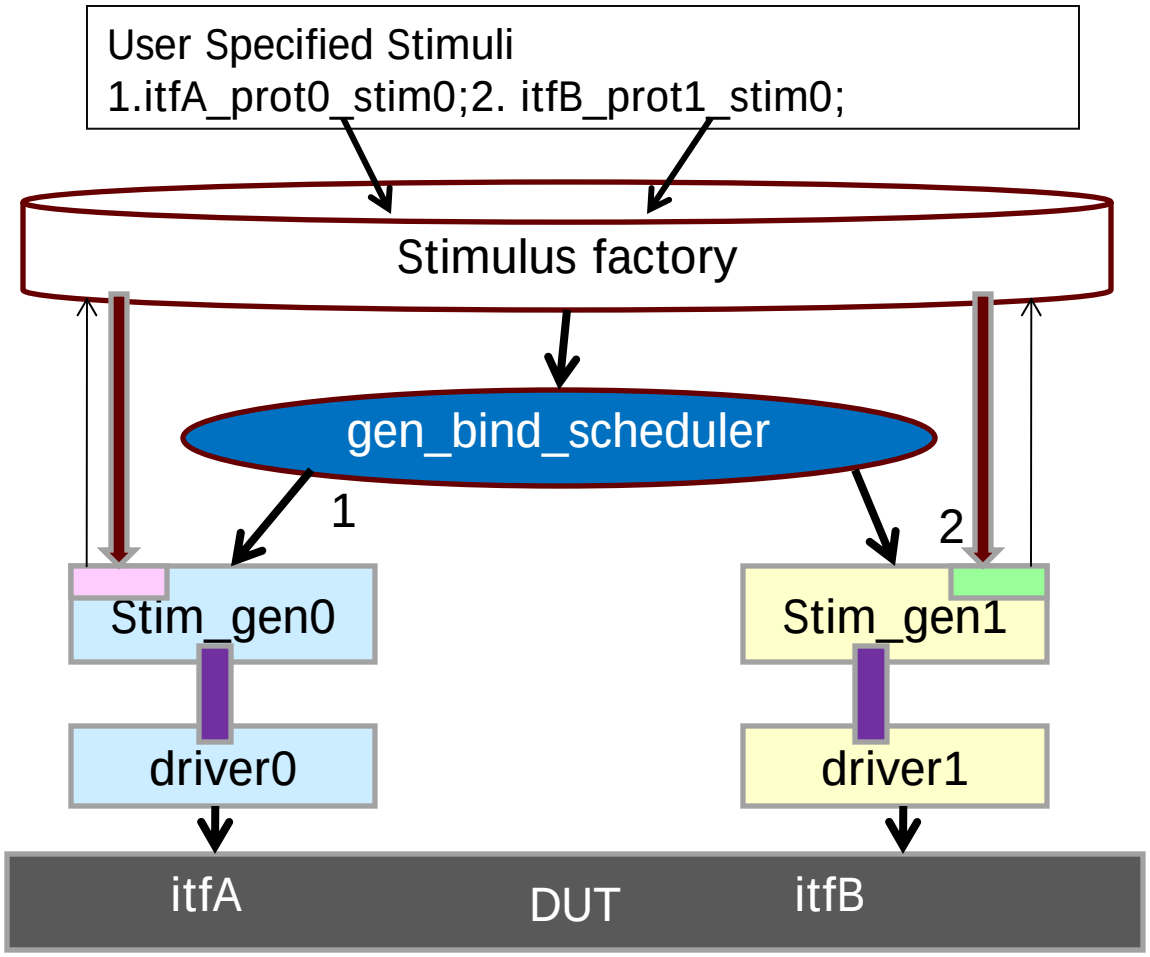
```
function base_stimulus stimulus_factory::create_stimulus(string
ikind);
    // if 'ikind' indicates itfA stimulus,
    itfA_blueprint_stim.stim_descr.set_stim_kind(ikind);
    itfA_blueprint_stim.create_transaction();
    return itfA_blueprint_stim;
endfunction
```

Outline

- Introduction
- Stimulus Representation
- Stimulus Generation
- Implementation
- Conclusion
- Future Work

Stimulus Generation

- Layered Stimulus Generation Structure



Stimulus Generation *(continue…)*

– Base gen_bind_scheduler Class

```
Class gen_bind_scheduler extends <base xactor>;
  stimulus_gen       generators[string];

  function new(string inst);
  virtual function void add_generator(stimulus_gen gen,
                                      string stat="IDLE");
  virtual function void bind_blueprints();
endclass
```

Register generator to generator queue

User to implement. Setup blueprint stimulus for each generator in queue

```
function void gen_bind_scheduler::bind_blueprints();
  generators["itfA"].default_stim_kind = "itfA_prot0_stim0";
  generators["itfB"].default_stim_kind = "itfB_prot1_stim0";
endfunction
```

Stimulus Generation

- Generic stimulus generator

```
class stimulus_generator extends <base xactor>;
   base_stimulus     randomized_stim;
   stimulus_factory  factory;
   string            default_stim_kind;
   int               stimulus_cnt;
   …
   // - Declare methods defined by the <base xactor>
   // New methods
   extern virtual task generate_stimulus(string stim_kind);
   extern virtual function bit is_done();
   extern task main();
   …
endclass
```

Extends base xactor

Handle to factory

Generation Process

Stimulus Generation *(continue...)*

– Generic Methods

```
task stimulus_generator::generate_stimulus(string stim_kind);
   // If needed, add pre_gen callback
   randomized_stim = factory.create_stimulus(stim_kind);
   // If needed, add post_gen callback
   if (gen_kind==0) stimulus_cnt++;
   // process to send out the transaction to the driver,e.g.,
   out_chan.put(randomized_stim.transaction);
endtask
```

```
task stimulus_generator::main();
   // If needed, add condition check before starting the process
   while (!is_done()) begin
     generate_stimulus(default_stimulus_kind);
     // If needed, update default_stimulus_kind
   end // while (1)
endtask
```

Outline

- Introduction
- Stimulus Representation
- Stimulus Generation
- Implementation
- Conclusion
- Future Work

Implementation – Base Classes

VMM	OVM
Standalone *stimulus_descriptor,* *scenario_descriptor,* *stimulus_factory*	Standalone *stimulus_descriptor,* *scenario_descriptor* *stimulus_factory extends* *ovm_sequence*
base_stimulus extends vmm_data	*base_stimulus extends* *ovm_sequence_item*
stimulus_gen extends vmm_xactor	*stimulus_gen extends ovm_sequencer*
gen_bind_scheduler extends *vmm_xactor*	*gen_bind_scheduler extends* *ovm_sequencer*

Implementation – Update VIP

VMM	OVM
Create <vip>_itf_stimulus -Instantiate <vip>_data -Implement create_transaction()	Same as VMM
Replace <vip>_gen with stimulus_gen	Replace <vip>_sequencer with stimulus_gen

```
class pi_itf_stimulus extends base_stimulus;  // VMM & OVM
   pi_transfer  stim_data;   // data class for PI interface
   …
   function void create_transaction();
     case (stim_descr.get_stim_name())
       “SHORT”: stim_data.SHORT_pkt_constr.constraint_mode(1);
       “LONG”: stim_data.LONG_pkt_constr.constraint_mode(1);
     endcase
     if (!stim_data.randomize())
       `vmm_error(log,…); OR ovm_report_error(…);
     $cast(transaction, stim_data.copy()); // use clone() for OVM
   endfunction
```

Implementation – Update TestBench

VMM	OVM
Create <tb>_stimulus_factory -Instantiate <vip>_itf_stimulus -Implement create_stimulus()	Same as VMM
Create <tb>_gen_bind_scheduler -Implement bind_blueprints() -Implement main()	Same as VMM
Update <tb>_env class -Instantiate <tb>_stimulus_factory -Instantiate <tb>_gen_bind_scheduler -Start <tb>_gen_bind_scheduler	Same as VMM (disable virtual sequencer)

Outline

- Introduction
- Stimulus Representation
- Stimulus Generation
- Implementation
- Conclusion
- Future Work

Conclusion

- String based stimulus_descriptor
- Stimulus class that has built in support for
  - Easy blueprinting
  - Transformation. May eliminate the need of some extra componesnts between generator and driver (Future work)
- Generic Generator
- Factory and gen_bind_scheduler also make blueprinting easier
- Work for VMM, OVM and other methodologies
- Highly reusable

Future Work

- Try more examples to
  - enhance the base classes
  - Find out more about its reusability
  - Impact on Scoreboarding and checking

THANK YOU!

Contact: ning.guo@paradigm-works.com