Simplifying UVM in SystemC

Thilo Vörtler[1], Thomas Klotz[2], Karsten Einwich[3], Felix Assmann[2]

[1] Fraunhofer IIS, Design Automation Division, Dresden, Germany
[2]Bosch Sensortec GmbH, Dresden, Germany , [3]COSEDA Technologies GmbH , Dresden, Germany

INTRODUCTION

UVM-SystemC is currently under standardization within Accellera with a first preview release expected in 2015. Although, the UVM standard is getting more and more language-agnostic with implementations available in e, SystemVerilog, and now SystemC, features for transaction-based stimulus and verification environment modeling still strongly rely on the underlying language. For example, packing, copying, and randomization operations are implemented differently in each of these languages; certain features such as aspect-oriented extensions of classes and methods are currently only available in e.

This work therefore presents an add-on library for UVM-SystemC to facilitate the easier creation of a UVM verification environment, with the goal to reduce the amount of code to be written and thereby making creation less error prone and tedious.

Template types for variable registration and randomization

- Variables of an object need to be registered for printing, packing, copying e.g. by the use of field macros
- New template types "uvm_phys_var" and "uvm_var" allow to register the member variables used in the current "uvm" object.
- This allows it to randomize member variables using randomization engines
- By deriving from a specialized base class it is also possible to provide standard implementations for methods such as "do_print" which the user has to implement.

```
class packet: public uvm::uvm_s_sequence_item {
public:
  uvm_enum packet_kind;
  uvm_phys_var<sc_dt::sc_uint<2> > address;
  uvm_phys_var<sc_dt::sc_uint<6> > length;
  uvm_enmum packet_length;
  uvm_phys_var<uvm_vector> payload;
  uvm_phys_var<sc_dt::sc_uint<8> > parity;
  uvm_var<sc_dt::sc_uint<8> > packet_delay;
  packet(const std::string& name = "packet");
};
```

Adding aspect-oriented features to UVM-SystemC

- One of the main features of using the *e* language is the aspect-oriented extension of classes. Aspects cover features or concerns that cut across the system or parts of it, i.e. they do not only effect only one class but multiple ones.
- An aspect can be used to extend existing classes by either introducing new or overwriting existing attributes, methods, etc.
- In order to span up additional aspects later on in the implementation, an extendable type of enumerations is needed.

```
namespace uvm_aspect_enums {
static const uvm_enum_elem SHORT(„SHORT“);
static const uvm_enum_elem LONG(„LONG“);

class uvm_enum {
  bool extend (std::initializer_list<uvm_enum_elem>
);
  std::vector<uvm_enum_elem> elements;
  …
};
…
}
static const uvm_aspect_enums::uvm_enum data_elem;
UVM_ASPECT_CLASS(packet) {
  UVM_ASPECT_CTOR(packet)
  {
    data_elem.extend({SHORT, LONG});
  }
};
```

Adding method calls based on aspects

- The class packet is extended for the case that the randomization selects for the enum "data_elem" the value SHORT.
- Method "my_method" is changed in a way that the method "make_to_short" is executed before the original method

```
UVM_ASPECT_CLASS(packet)
{
  uvm_method<bool(int&,std::string)> my_method;
  bool check(int& val, std::string kind);
  UVM_ASPECT_CTOR(packet) {
    data_elem.extend({SHORT, LONG});
    //point to actual method
    my_method.is(check);
  }
};
// add method extension for SHORT packet only
UVM_EXTEND_CLASS(packet, SHORT){
  bool make_to_short(int& val, std::string kind);
  UVM_EXTEND_CTOR(packet,SHORT) : uvm_aspect(data_elem==SHORT)
  {
    // for SHORT packet execute the make_to_short method first
    my_method.is_first(make_to_short);
  }
};
```

Example

```
int sc_main(int argn,char* argc[])
{
  //instantiates packet with all extensions
  packet* my_packet=UVM_INSTANTIATE(packet);
  my_packet->randomize();  //randomizes variables coresponding to the constraints
  //calls methods corresponding to the randomized data_elem (the aspect)
  my_packet->my_method(23,"ADR");
}
```

- A packet is instantiated, which gets randomized afterwards by calling the function "randomize()"
- When calling the member function "my_method", depending on the aspect it is determined in which order the method calls are executed.

Conclusions

This paper described a C++ library on top of UVM-SystemC In particular, the examples showed the following features:

- Use of template based member variables to simplify randomization
- Extendable enumeration type (as a base for aspects later on)
- Aspect-specific extension of methods
- Aspect-specific adaptions of constraints

This paper describes an ongoing work, the library implementation is subject to change. The proposed library is planned to be donated to the Accellera SystemC Verification Working Group and to be made available as open source.