



SystemC / SystemVerilog Interaction

Holger Keding
Staff CAE, System Level Solutions,
Synopsys GmbH

Agenda

- How does SystemC and SystemVerilog compare?
 - application focus, acceptance, strengths, ...
 - why is interaction between SC and SV required
- Technical Issues and Solutions
 - what is the right level of interaction
 - connecting SC to SV and vice-versa
 - is the SV DPI enough to build a SV-SC bridge?
 - synchronizing SC and SV
- Example for using a SV-SC transaction level interface
- Summary

Introduction to SystemC

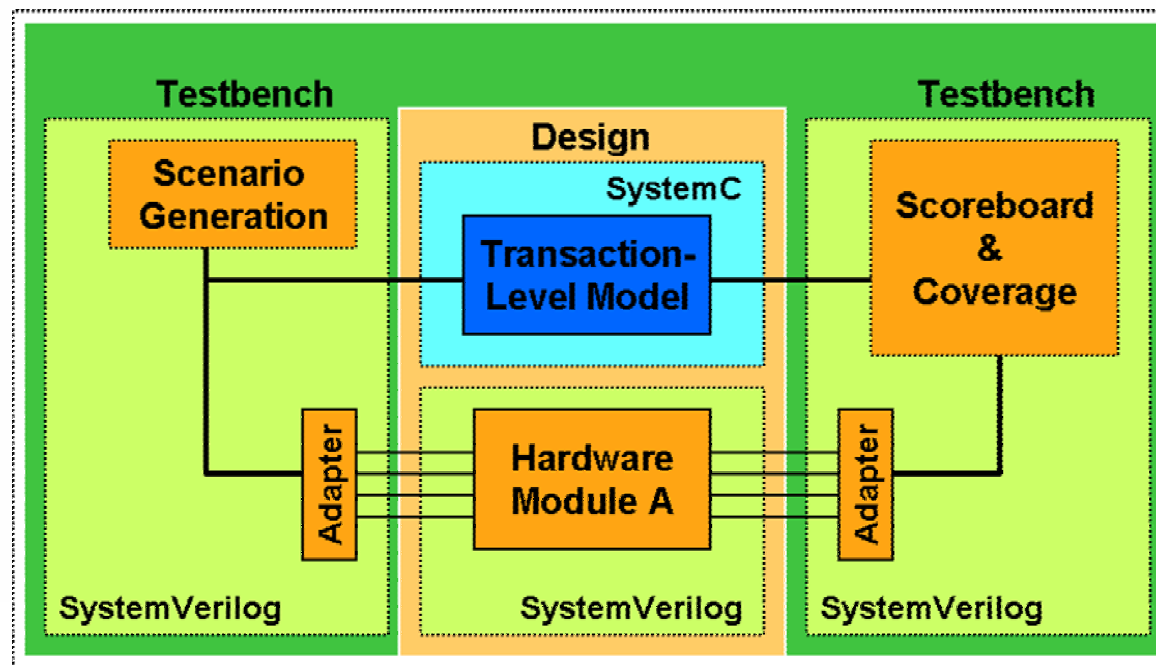
- IEEE 1666-2005
- Excels at architectural exploration and early SW development
- Cleanly supports
 - Transaction level modeling and verification
 - HW/SW co-design
 - SOC architectural analysis and optimization
- Based on C/C++
 - Ideal for integrating models written in C/C++
 - Easy to integrate embedded SW components
 - Easy adoption for people familiar with C++ and system design.
- Widely used

SystemVerilog

- IEEE 1800-2005
- Excels at Verification and RTL Design
- Verification as an integral part of the design language
 - System Verilog Assertions (SVA)
 - Constraint Random Stimuli generation
 - Functional Coverage
 - Classes, queues, strings, dynamic arrays, associative arrays
- Enablement of design Abstraction - high impact on productivity
 - Additional high level constructs such as interfaces
 - Better consistency between synthesis and simulation
- Widely used
- Supported by most EDA vendors

Need for SC-SV Interoperability

- SC and SV are being widely adopted, for different areas though
 - Mixed language usage is already common
 - Main usage models:
 - ◆ (re-)use SystemC TL model as reference model in a SystemVerilog testbench
 - ◆ ensure consistency of RTL and SystemC TL models through common SV testbench



Abstraction Level of Interoperability

- SC modules can instantiate SV and vice-versa
 - Single module hierarchy with mixed language components
- SC and SV schedulers need unified semantics
 - Mixed language systems should work as if they were in a single language
- Pin-Level: SC signals can bind to SV ports and vice-versa
 - Standard set of conversions supported
 - Straightforward to use, but rarely matching design flow requirements
- TL: SC calls SV tasks/functions, SV calls SC methods
 - matches most design flow requirements
 - higher simulation and modeling efficiency

Connecting SC to SV and vice-versa

Different approaches conceivable:

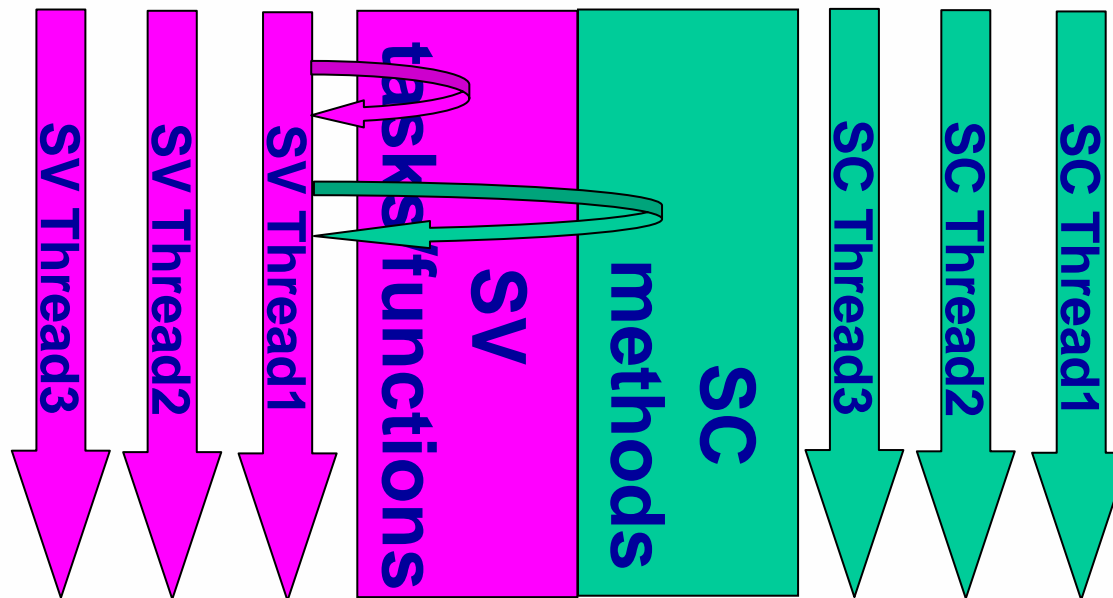
- use fifo-like channels for cross-language connection
 - represented as channel / module in both language domains
 - fifo takes care of data transport and synchronization
 - typical testbench architecture style (e.g. vmm_channels)
 - may cause timing conflicts when used with SC_METHODS
- use common communication concepts in SV and SC
 - both languages have a concept of Interfaces
 - both SV and SC interfaces can contain methods / tasks
 - mapping SV IFs to SC IFs can serve as an easy and intuitive bridge
 - most flexible approach

Is the SV DPI enough as SV-SC bridge?

- Direct Programming Interface (DPI) is part of the IEEE 1800
 - 👍 Allows SV to call C functions (import tasks/functions)
 - 👍 Allows C to call SV functions/tasks
 - 👍 easy to use and most efficient
 - 👎 Only specifies a C interface, not C++, not SystemC
 - 👎 DPI cannot be used to traverse SystemC hierarchy, handles to instance or objects are not easily possible
 - 👎 DPI is build to interface to a single threaded untimed C program

Synchronizing SystemC and SystemVerilog

- DPI works fine as long as you call non-blocking methods / tasks / functions.
- Time consuming tasks / methods must not be called through the DPI.



- You need a service layer on top of DPI that takes care of
 - proper synchronization even for blocking method / task calls

Using a SV-SC transaction level interface

- Example for mapping an SV and SC interface to each other:

SystemVerilog Interface containing tasks

```
interface simple_bus_blocking_if;
...
    task burst_read(input int unsigned priority_
                    , output int data[32]
                    , input int unsigned start_address
                    , input int unsigned length
                    , input int unsigned lock
                    , output int unsigned status);
...
endinterface
```

Using a SV-SC transaction level interface

- Example for mapping an SV and SC interface to each other:

```
// instantiate the SystemC model and the SystemVerilog interface
simple_bus_test          simple_bus_test_instance(clk);
simple_bus_blocking_if   master_blocking();
...
// send constrain random stimuli to the SystemC model
trans0.randomize();
master_blocking.burst_write(trans0.prio,
                             trans0.data,
                             trans0.start_address,
                             trans0.length,
                             trans0.lock,
                             trans0.status);
```

Using a SV-SC transaction level interface

- Corresponding SystemC interface

SystemC Interface containing methods

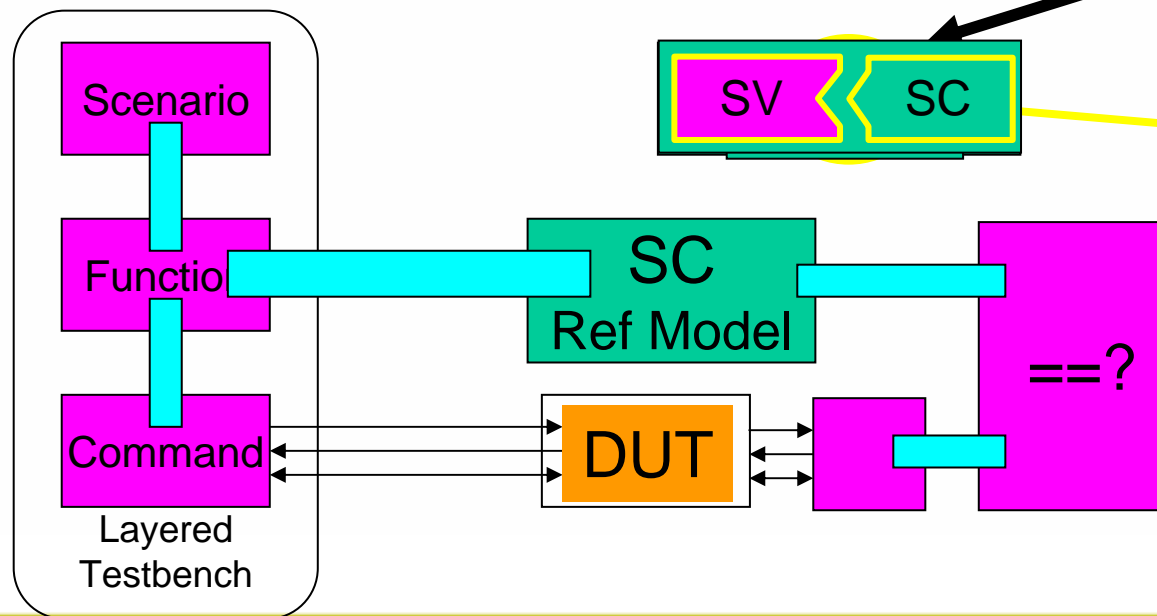
```
class simple_bus_blocking_if : public virtual sc_interface
{
public:
    // blocking BUS interface
    virtual simple_bus_status burst_read(unsigned int unique_priority
                                        , int *data
                                        , unsigned int start_address
                                        , unsigned int length = 1
                                        , bool lock = false) = 0;

    ...
}; // end class simple_bus_blocking_if
```

TLI - Use Model

- Re-use / specify SC interface methods
- Automatically create pair of SV+SC adapters
- Instantiate adapters
- Call IF task/function/method

```
class simple_bus_blocking_if: virtual  
  public sc_interface {  
  status burst_read(...);  
  status burst_write(...);  
};
```



The automatically generated adapters forward call to SC interface method and take care of synchronization between SC and SV

Debugging SC-SV interaction

- Debugging must not fall apart if languages come together
- What do you need to debug a mixed SV + SC design at Transaction Level?
 - Debug SV and SC sources in same frame work
 - Breakpoints, stepping in both domains
 - Follow function calls across language border
 - Combined tracing of signals
 - Trace / visualize transactions

Debugging SV-SC interaction

The screenshot shows a simulation environment with a hierarchy tree on the left, a list of line numbers in the center, and two code windows on the right. The hierarchy tree shows a path from 'root' to 'main_action'. The line numbers list shows a jump from line 105 to 107. The 'Testbench (SV)' window shows code for environment generation and running. The 'Ref-Model (SC)' window shows the implementation of a simple bus main action.

Testbench (SV)

```
91
92 env.gen_cfg();
93 sters to 3
94 env.cfg.masters =
95
96 env.build();
97
98 for (int i=0; i<en
99 my_simple_bus_t
100 simple_bus_tran
101 burst_scenario
102 ref_master_if[i
103 burst_scenario.
104 // env.gen.scen
105 env.gen[i].scen
106
107 end
108
109 env.run();
110 end
111 endprogram
```

Ref-Model (SC)

```
107
108 void simple_bus::main_action()
109 {
110 // m_current_request is cleared aft
111 // single data transfer. Burst requ
112 // select the request again.
113
114 if (!m_current_request)
115 m_current_request = get_next_requ
116 else
117 // monitor slave wait states
118 if (m_verbose)
119 sb_fprintf(stdout, "%g SLV [0x%
120 m_current_request->a
121 m_had_lock = 0;
122 if (m_current_request)
123 handle_request();
124 }
125
126 //-----
127 //-- direct BUS in
128 //-----
129
```

Things that would make life easier (but still are difficult to achieve)

- Avoid redeclaration/definition of parameters or user defined types
 - Reference SC header file directly from SV
 - Reference SV package directly from SC
- support complex/user defined data types/classes over the DPI

Conclusion

- Mixed language SC – SV flows will be common
 - Using each language at its focus
 - Usage of these flows is already starting
 - Integrated simulation and debugging will be key
- Several approaches to integrating SC and SV
- Standardization efforts for some of these approaches may start soon