

Parameter Passing From SystemVerilog to SystemC

Bishnupriya Bhattacharya, Samik Das, Zhiting
Duan, Chandra Sekhar Katuri, Pradipta Laha

Cadence Design Systems

cādence®

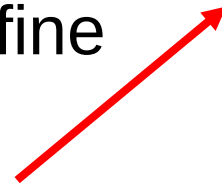
Outline

- Problem Statement
- CCI Parameters in SystemC
- Parameter Passing from SystemVerilog to SystemC
- Results and Example
- Future Work

Problem Statement

- Proliferation of mixed-language designs & test benches
 - Our focus is SystemVerilog-SystemC
- No IEEE/Acellera standard for mixed-language
 - Commercial simulators provide support
- Two primary use models for SV-SC designs
 1. Function calling (DPI based)
 2. **Instantiation**
- For instantiation use model, need to define
 - Port interoperability
 - **Parameter interoperability (for SC instantiated in SV)**

Focus of this work



Our Approach

- Don't invent any ad-hoc solution
- Build upon existing parameter standards
 - IEEE 1800 defines parameter standard in SV
 - IEEE 1666 does **not** define parameter standard in SC
- In SystemC, parameters modeled today as constructor arguments
 - Requires Source code recompilation to change values

Our Approach

- To address this void in SystemC, **Accellera CCI WG** was formed
 - **Configuration, Control and Inspection** Working Group
 - First charter is to define SystemC parameter standard
 - Draft LRM has been published for internal review
 - **Actively working towards Accellera standardization soon**
- Cadence is an early adopter of CCI parameter standard
- **We extend CCI parameter to SV-SC language boundary**
 - Interface SV parameters with CCI parameters in SC
 - Define organic and intuitive semantics for both languages

CCI Parameters – Overview

- Declare in owning module
 - `cci_param<int, elaboration_time_parameter>`
 - specify **default value** during parameter construction
- Assign **override value before construction** using ‘broker’
 - In parent/ancestor modules, use **programmatic API** `set_init_value(<name>, <string_val>)` to configure parameter of child/descendant module
 - Global broker maps {name,value} pair to parameter object
 - Value specified as JSON string
- Assign override value from **external configuration file**
 - Tool will apply with highest precedence
 - **No source code recompilation!**

CCI Parameters – Overview

- During CCI parameter construction
 - Query broker if override value exists
 - If specified, override value wins following precedence rules
 - default value wins otherwise
- To **change value after parameter construction**, use API *set(<value_type>)*
- Specify **callbacks** on a parameter
 - pre_read
 - pre_write/post_write
 - create_param
 -

Passing Parameters from SV to SC – Data Type Mapping

- Declare `cci_param` in SystemC module using normal SC syntax

```
cci_param<int> p1;
```

- We define **data type mapping** from CCI parameter type to appropriate SV type
 - Based on DPI-C mapping
 - Extended to SystemC

SC type	SV type
char	byte
bool	bit
int	int
sc_bv<N>	bit[0:N-1]
....	

Passing Parameters from SV to SC – Restrictions

- Restrictions defined on CCI parameter visibility in SV
- Motivated by logical intersection of SV parameter semantics and CCI parameter semantics
 1. Only CCI parameters created in SC module constructor are visible in SV
 - In SV, parameters are constants, fixed at end of elaboration
 - CCI parameters can also be created during simulation
 - Does not map naturally to SV parameter semantics
 2. CCI parameters created as *toplevel* are not visible in SV
 - Do not belong to parent module scope
 - No natural map to SV parameters that always belong to parent scope

Passing Parameters from SV to SC – Value Propagation

1. Default value is owned by SystemC
2. SV parent can contribute override value when instantiating SC module using normal SV syntax

```
scmod #(.p1(19)) sc1();
```

 - SV instantiation override translates to calling API *set_init_value()* just before SC module construction
 - No JSON string syntax is exposed to SV
 - SV override competes with other overrides as determined by the temporal invocation point, following CCI semantics
 - E.g. override in external configuration file always wins
3. Final value of CCI parameter is propagated back to SV

Passing Parameters from SV to SC – Other Rules

- **Positional vs. named parameter instantiation override**
 - For direct instantiation use model, only support named override
 - For shell-based use model, support both named and positional override
- **No support for *defparam* override**
 - IEEE 1800 deprecates *defparam*
- **CCI parameter construction is the final synch point across SV-SC language boundary**
 - No propagation to SV of later CCI parameter value change from *set()* API
- **SV interoperability does not trigger any CCI callbacks**

Results

Parameter Passing from SystemVerilog to SystemC
has been implemented in the **Cadence Incisive®**
simulator

```
irun sc.cpp top.v -sysc -top top -sconfig param.conf
```

Example

Default value owned by SystemC

```
// top.v
module scmod ()
  (* integer foreign = "SystemC"; *)
  parameter int p1 = 11;
endmodule
module child #(parameter cp = 1);
endmodule
module top;
  parameter vp = 9;
  scmod #(.p1(vp+1)) sc1();
  scmod #(29) sc2();
  scmod sc3();
  child #(top.sc1.p1) v1();
  initial $display("top.sc1.p1 = %d", sc1.p1);
  initial $display("top.sc2.p1 = %d", sc2.p1);
  initial $display("top.sc3.p1 = %d", sc3.p1);
  initial $display("top.v1.cp = %d", cp);
endmodule
```

```
// sc.cpp
SC_MODULE (scmod) {
public:
  SC_CTOR(scmod) :
    scp1("p1", 0)
  { ..... }
protected:
  cci_param<int> scp1;
};
```

```
// param.conf
top {
  sc1 {
    !parameters {
      p1 "17"
    }
  }
}
```

Future Work

- Once Accellera CCI standard is formally published, review and adjust this specification if/as necessary
- Define parameter interoperability when SC instantiates SV
- Map VHDL *generics* to CCI parameters

cā dence®

© 2014 Cadence Design Systems, Inc. All rights reserved worldwide. Cadence and the Cadence logo are registered trademarks of Cadence Design Systems, Inc. in the United States and other countries. ARM and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. All other trademarks are the property of their respective owners.