Deploying SystemC® for Complex System Prototyping and Validation
CCI WG Update

Trevor Wieman, WG chair
Sr. Member Technical Staff
Intel Corp.

Materials prepared in collaboration with Doulos

** Details of this draft SystemC Configuration standard preview are subject change.**
Configuration, Control & Inspection

Goal: Standardizing interfaces between models and tools
Parameterizing a Typical System

System IPs - DMA, PLL, Power

Peripheral subsystem IPs

Memory subsystem

CPUs

On-chip bus

Accelerator & co-processor IPs

Generally, IPs come from many vendors

Parameter Examples
- system clock speed
- # processor cores
- memory size
- address, data widths
- disabled IP(s)
- address maps
- SW image filename
- IP granularity debug control:
  - logging
  - tracing

Need uniform way to configure simulation w/o recompilation
Key Configuration Components

- **Parameter**
  - Has a name (a string) plus a value
  - Is an instance of cci_param<T> class template (T is a type)
  - Is registered with a broker at construction
  - Provides string-based API for type-independent set/get of value

- **Broker**
  - Provides controlled access to parameters registered with it
  - There is one global (public) broker; many private brokers may also exist
Manage parameter:
- Initialization
- visibility
- access type e.g. read-only

Specify initial value (overrides default)

cci_base_param

cci_param

cci_cnf_broker

Configurator (SystemC module or tool)

SystemC module (parameter owner)
A Parameter Owner Module

```cpp
SC_MODULE(simple_ip) {
  private:
    cci::cnf::cci_param<int> int_param;
  
  public:
    SC_CTOR(simple_ip):
      int_param("int_param", 0) {
      int_param.set_documentation("...");
      SC_THREAD(do_proc);
    }

    void do_proc() {
      for(int i = 0; i < int_param; i++) {
        ...
      }
    }
};
```

- Parameters are usually private members forcing brokered access.
- Default values are optionally supplied using a constructor argument.
- Owner may read parameter value.
Parameter Access via Broker (1)

```c
SC_MODULE(configurator) {
    cci::cnf::cci_cnf_broker_if *m_brkr;
    Handle to broker

    SC_CTOR(configurator) {
        m_brkr = &cci::cnf::cci_broker_manager::get_current_broker(
            cci::cnf::cci_originator(*this));
        Get handle to broker associated with
        this module (otherwise global broker)

        sc_assert(m_cci != NULL);

        SC_THREAD(do_proc);
    }

    ...
```
void do_proc() {
    const std::string int_param_name = "top.sim_ip::int_param";
    if( m_brkr->exists_param(int_param_name)) {
        cci::cnf::cci_base_param *int_param_ptr =
            m_brkr->get_param(int_param_name);
        std::string p_value = int_param_ptr->jsonSerialize();
        ...
        int_param_ptr->jsonDeserialize("2");
        ...
    }
}
Accessing Parameter Value

- When value type is known, call parameter's set or get function
  - Common C++ types
  - SystemC Data types

- When parameter type is unknown or unsupported:
  - Use JavaScript Object Notation (JSON) format strings
  - jsonSerialize() == get(), jsonDeserialize() == set()
Parameter Mutability

- Parameters are mutable by default
- Mutability set by template parameter

```cpp
cxi::cnf::cci_param<int, cxi::cnf::mutable_parameter> p1;
```

- Parameters may also be immutable or locked after elaboration

```cpp
cxi::cnf::cci_param<int, cxi::cnf::immutable_parameter> p2;

cxi::cnf::cci_param<int,
                      cxi::cnf::elaboration_time_parameter> p3;
```
Private Brokers

- Broker association happens during module construction

- A standard private broker class is provided
  - Grants access only to associated module hierarchy
    - No tool access is allowed!
  - Guidelines for creating custom private brokers are also supplied

- Encapsulate “black-box” (pre-compiled) IP configuration using:
  - A private broker, to prohibit unauthorized access
  - A configurator, to apply pre-compiled configuration
Parameter Callbacks

- Callback functions may be registered with a parameter.
- Callback reason reflects nature of parameter access.

```c
enum callback_type { pre_read, reject_write, pre_write, post_write, create_param, destroy_param };
```

- Callbacks usually members of module owning parameter.
- Callback functions return a status.

```c
enum callback_return_type { return_nothing, return_value_change_rejected, return_other_error };
```
Parameter Owner with Callback

```cpp
SC_MODULE(simple_ip) {  
  private:  
    cci::cnf::cci_param<int> P1;  
    cci::shared_ptr<cci::cnf::callb_adapt> P1_cb;  
  
  public:  
    SC_CTOR(simple_ip): P1("P1", 0) {  
      P1_cb = P1.register_callback(cci::cnf::post_write,  
        this, cci::bind(&simple_ip::cb, this, _1, _2));  
      ...  
    }  
    
  callback_return_type cb( cci_base_param& changed_param,  
    const callback_type& cb_reason);  
    ...  
};
```

- Callbacks accessed via shared pointer
- Callback registered in constructor
- Callback function must have this signature
callback_function_definition

callback_return_type simple_ip::cb(
    cci_base_param& changed_param,
    const callback_type& cb_reason)
{
    switch(cb_reason)
    {
        case cci::cnf::pre_write:  ...
            break;
        case cci::cnf::post_write: ...
            break;
        default:
            sc_report_warning("CB", "Unrecognized reason");
    }
    return cci::cnf::return_nothing;
}

Detect callback reason – function could be registered with multiple callbacks!
CCI WG Summary

- **Initial focus is on the Configuration standard**
  - Aligned to requirements that were publicly reviewed
  - Anticipating public review of a draft standard this year
    - Will include LRM and examples

- **Remaining charter will then be prioritized for subsequent work**
  - “Control” and “Inspection”