Improving the Usability and Performance of Tracing in SystemC

Rauf Salimi, Philipp Hartmann

Presentation Copyright Permission

– A non-exclusive, irrevocable, royalty-free copyright permission is granted by Intel Corporation to use this material in developing all future revisions and editions of the resulting draft and approved Accellera Systems Initiative **SystemC** standard, and in derivative works based on this standard.

SystemC Traces

- Temporally ordered records of value changes
- Generated for entities selected by **sc_trace**

```
void sc_trace(sc_trace_file*, const object&, const std::string& );
template <class T>
         void sc_trace(sc_trace_file*, const sc_signal_in_if<T>&, const std::string& );
template <class T>
         void sc_trace(sc_trace_file*, const sc_in<T>&, const std::string& );
template <class T>
         void sc_trace(sc_trace_file*, const sc_inout<T>&, const std::string& );
```

- **sc_trace**
  - Is defined for primitive C/C++ and SystemC types.
  - Can be defined for custom types using primitive **sc_trace** .

Tracing in SystemC - Bottlenecks

- Modern, large-scale system-level models:
  - Tens of thousands of traceable objects
  - Thousands of modules
  - Many levels of hierarchy

- The following proposals address
  - Usability and developer efficiency
  - Tracing performance

Usability Bottleneck

- One explicit `sc_trace` needed per traced object.
- User code needed for passing around trace file handles across hierarchy.
- Custom types need to define `sc_trace` even if tracing of objects of the type is not required.
- It can become a major contributor to overall LoC.
- It can become a source of programming errors and maintenance effort.

Performance Bottleneck

- Pulling/polling implementation irrespective of object type:
  - C++ and SystemC primitive data types
  - signals and buffers
- **`sc_trace_file::cycle()`**
  - Iterates over all traced objects at least once per timed notification phase.
  - Checks if the value of the object has changed and records the value if it has.
- Performance affected by the number of inactive signals.
- The cost of object's comparison operator affects performance.
  - especially costly for vectors (e.g. logic vectors) and custom aggregate types
- Redundant comparison operations for signals

Usability Bottleneck - Proposal

- Treat all SystemC objects as potentially *traceable.*

```
virtual void sc_object::trace(sc_trace_file* tf ) const;
```

- This will increase efficiency and flexibility. for example:
  - Tracing any SystemC object by name (e.g. from a configuration file, tool)
  - Tracing multiple SystemC object by traversing the SystemC object hierarchy
  - Tracing local variables can be added to the `trace()` override for `sc_module`
- This is gated by the current standard
  - Annex C.O (deprecated features): Member function `trace()` of classes `sc_object`, `sc_signal`, `sc_clock`, and `sc_fifo` (Use `sc_trace` instead)

Experimental Results

Model	Number of sc_trace() calls
System-level Virtual Prototype A	18000
System-level Virtual Prototype B	6000

The following snippet was used instead of explicit sc_trace calls. A fine-granular control of tracing can be implemented using CCI and parameters.

```
void trace_all(sc_object* object, sc_trace_file* tf) {
    object->trace(tf);
    std::vector<sc_object*> children= object->get_child_objects();
    for(unsigned i=0;i<children.size();i++)
        trace_all(children[i], tf);
}
```

Performance Bottleneck – Solution 1

- An event-driven, push-based `sc_trace_file :`
  - Utilize the **`value_changed_event()`** of **`sc_signal_in_if`**.
  - Spawn a *monitor* **`SC_METHOD`** per traced **`sc_signal_in_if`** derivate.
- Significant performance improvement for *`sc_signals`* of
  - User-defined aggregate types
  - SystemC primitive types with high comparison cost
- Caveats:
  - Traced values are off by one delta cycle.
  - It is *delta cycle tracing* by nature.

Experimental Results

Number of Idle Signals (Aggregate Data Type)	Speedup
0	0.94
100	1.13
500	1.5
1000	1.9
2500	3.15
5000	6.7

Achieved a speed-up of 4x-5x in a large system-level virtual prototype.

Performance Bottleneck – Solution 2

- Improve pull-based tracing using *update sets.*
- An update set maintains a list of traces which *may* change together.
- For signals, update sets can be utilized to reduce the number of comparisons significantly.
- This will have major impact on performance especially for types with high comparison costs.

Performance Bottleneck – Solution 2

The required updates to the standard/LRM:

Section 6.8.4,

```
template <class T>
void sc_trace(sc_trace_file*,const sc_in<T>&,const std::string&);
```

Section 6.10.5

```
template <class T>
void sc_trace(sc_trace_file*,const sc_inout<T>&,const std::string&);
```

*"Function **sc_trace** shall trace the channel to which the port passed as the second argument is bound (see 8.1) by calling function **sc_trace** with a second argument of type ~~**const T&** (see 6.4.3)~~ const **sc_signal_in_if<T>&**"*

This update is required to allow the delegation of the trace call to the **sc_trace** overload for sc_signal (see 8.1.6).

Experimental Results

Number of Idle Signals (Aggregate Data Type)	Speedup
0	1
100	1.12
500	1.4
1000	1.7
2500	2.3
5000	3.8

Conclusion

- Minor changes to the SystemC Standard and the LRM will enable:
  - Improving the usability of SystemC tracing.
  - Implementation of standard-compliant efficient solutions.

- Get involved in the ongoing discussions in the LWG forum!

Backup

sc_update_occurred_if

```
class sc_update_occurred_if : public virtual sc_interface
{
public:
    typedef /*implementation-defined*/ change_stamp;

    virtual bool update_occurred() const = 0;

    virtual bool
    update_occurred_since(change_stamp& last_stamp) const = 0;
protected:
    sc_update_occurred_if()  = default;
    ~sc_update_occurred_if() = default;
};

sc_update_occurred_if::change_stamp
sc_get_current_change_stamp();
```