DOULOS

Preparing the SystemC Language Reference Manual (LRM) for Standardisation

John Aynsley

DOULOS

The SystemC LRM

- What's Happening?
- LRM Goals
- Why The LRM is Necessary
- A Peek at the Changes
- What of the Reference Simulator?

What's Happening

- Draft 2.0.1 LRM first published June 2003
- Being reviewed and rewritten ready for IEEE
  - 2.0.1 LRM largely complete
  - 2.1 LRM to be completed by end 2004

LRM Goals

- Starting point for the IEEE standard
- To be the definitive reference for SystemC
- To differ from the current reference simulator only where it is plainly buggy or inconsistent
- To include only features that will remain in SystemC indefinitely

Why the LRM is Necessary

```
SC_MODULE(mod)
{
  sc_in <int> a, b;
  sc_out<int> f;

  SC_CTOR(mod)
  {
    SC_THREAD(entry);
    sensitive << a << b;
  }
  void entry();
};
```

- Published part of SystemC

```
class sc_runnable
{
public:

  sc_runnable();
  ~sc_runnable();

  void push_method( ... );
  void push_thread( ... );

  ...
};
```

- Private artefact of the reference simulator

Why the LRM is Necessary

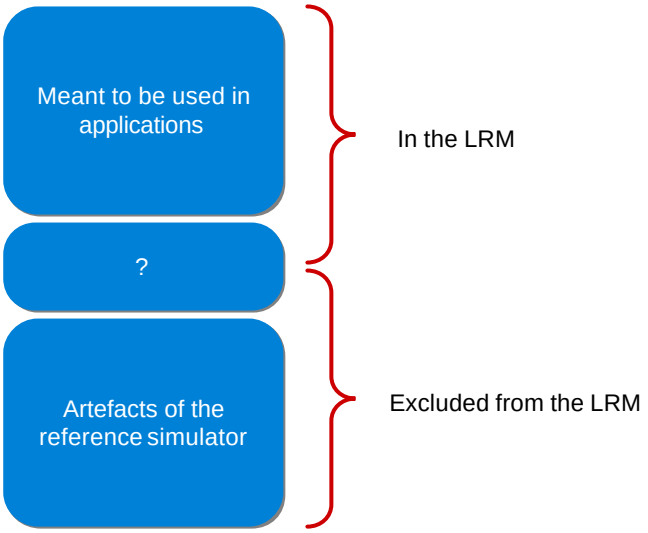
- But what about these?

```
class sc_process_b;
```

```
class sc_sensitive;
```

```
class sc_simcontext
{
  ...
  uint64 delta_count() const;
  bool is_running() const;
  bool update_phase() const:
  ...
};
```

Why the LRM is Necessary



In Fact...

```
class sc_process_b;
```

- Still being debated

```
class sc_sensitive;
```

- In the LRM, but an application shall not create an object of this class

```
class sc_simcontext
{
  ...
  uint64 delta_count() const;
  bool is_running() const;
  bool update_phase() const:
  ...
};
```

- Out of the LRM, but replaced by new function sc_delta_count()

Why the LRM is Necessary

- Binding a port at the point of module instantiation

```
SC_MODULE(mod)
{
  sc_in <int> a, b;
  sc_out<int> f;

  SC_CTOR(mod)
  {
    SC_THREAD(entry);
    sensitive << a << b;
  }
  void entry();
};
```

```
{
  m = new mod("m");
  m->a(p);
  m->b(q);
  m->f(r);
}
```

Why the LRM is Necessary

- But is this SystemC? What was the intent?

```
SC_MODULE(top)
{
  sc_signal<int> a, b, f;
  ...
};
```

```
SC_MODULE(comp)
{
  sc_in <int> p, q;
  sc_out<int> r;

  top *t;

  SC_CTOR(comp)
  {
    t = new top("t");
    p.bind(t->a);
    q.bind(t->b);
    r.bind(t->f);
  }
};
```

In Fact...

- This is legal and correct, but strongly discouraged

```
SC_MODULE(top)
{
  sc_signal<int> a, b, f;
  ...
};
```

```
SC_MODULE(comp)
{
  sc_in <int> p, q;
  sc_out<int> r;

  top *t;

  SC_CTOR(comp)
  {
    t = new top("t");
    p.bind(t->a);
    q.bind(t->b);
    r.bind(t->f);
  }
};
```

What is The LRM?

- The LRM is a contract between tool developer and tool user
  - The tool developer must provided the classes and semantics defined in the LRM
  - The tool user can only assume the classes and semantics defined in the LRM
- The LRM defines the *intent* of the class library
- The LRM is not a user guide

A Peek at the Changes

- Deprecated

```
sc_set_default_time_units(1, SC_NS);
sc_start(10);
```

```
sc_cycle()
sc_initialize()
```

```
sc_bit
```

```
sc_in_clk
sc_inout_clk
sc_out_clk
```

```
timed_out()
```

```
sensitive_pos
sensitive_neg
```

```
watching()
delayed()
```

A Peek at the Changes

```
inst.port(chan);
```

- Named binding - okay

```
inst.port.bind(chan);
```

- Named binding - okay

```
inst(chan1, chan2, chan3);
```

- Positional binding - okay

```
inst(chan1);
inst(chan2);
inst(chan3);
```

- Deprecated

```
inst << chan1 << chan2 << chan3;
```

- Deprecated

What of the Reference Simulator?

- The Reference Simulator is still there!
- There is a Reference Simulator for version 2.1
- Will be brought into line with the LRM
- Will still include all the deprecated features
- Legacy applications will still work

Your Chance!

- Lookout for the new LRM on the OSCI Website for public review