The New IEEE 1666 SystemC Standard

John Aynsley, Doulos

The IEEE P1666 Working Group

IEEE

IEEE P1666 Goals And Charter

- **Charter is to update the 1666-2005 SystemC Standard**
  - **Errata and clarification to 1666-2005 standard**
  - **Better formalization of TLM 1.0 Message Passing Interface**
  - **Inclusion of OSCI TLM-2.0 LRM**
  - **New and enhanced features**
- **Progress**
  - **Draft P1666-2011 standard is complete**
  - **Plan to ballot draft P1666-2011 standard in Q1 2011**
- **Chair: Stan Krolikoski (stanleyk@cadence.com)**

IEEE

2

P1666 Member Entities

Accellera
Cadence
Freescale
Intel
JEITA
Mentor
NXP
OSCI
ST Micro
STARC
Synopsys
Texas Instruments

IEEE

3

New Features!

suspend() resume() reset() kill() async_reset_signal_is()

sc_pause() sc_get_status() sc_time_to_pending_activity()

sc_vector

set_verbosity_level() SC_REPORT_INFO_VERB

IEEE

4

Process Control - suspend & disable

```
SC_THREAD(control);
SC_THREAD(target);
  t = sc_get_current_process_handle();
```

```
void control()
{
  wait(...);
  ev.notify(5, SC_NS);
  t.suspend();
  wait(10, SC_NS);
  t.resume();

  ev.notify(5, SC_NS);
  t.disable();
  wait(10, SC_NS);
  t.enable();
}
```

t awakes

t waits for next event

```
void target()
{
  ...
  for (;;)
  {
    wait(ev);
    ...
  }
}
```

IEEE

5

suspend vs disable

```
void control()
{
  ...
  t.suspend();
  ...
  t.resume();
  ...
}
```

- Target awakes asynchronously
- Target can tolerate being delayed
- Unsuitable for clocked target processes
- Building abstract schedulers

```
void control()
{
  ...
  t.disable();
  ...
  t.enable();
  ...
}
```

- Target awakes synchronously
- Suitable for clocked targets
- Abstract clock gating

IEEE

6

reset and kill

```
SC_THREAD(control);
SC_THREAD(target);
  t = sc_get_current_process_handle();
```

```
void control()
{
  wait(10, SC_NS);
  ev.notify();

  wait(10, SC_NS);
  t.reset();

  wait(10, SC_NS);
  ev.notify();

  wait(10, SC_NS);
  t.kill();
}
```

t awakes

t is reset asynchronously

t awakes

t is terminated immediately

```
void target()
{
  ...
  for (;;)
  {
    wait(ev);
    ...
  }
}
```

IEEE

7

reset and kill are immediate

```
void target()
{
  ...
  for (;;)
  {
    try {
      wait(ev);
      ...
    }
    catch (const sc_unwind_exception& ex)
    {
      sc_assert( sc_is_unwinding() );
      if (ex.is_reset())
        cout << "target() was reset" << endl;
      else
        cout << "target() was killed" << endl;
      throw ex;
    } ...
```

reset()

kill()

IEEE

8

reset_event

```
SC_THREAD(control);
SC_THREAD(target);
  t = sc_get_current_process_handle();

SC_METHOD(reset_handler);
  dont_initialize();
  sensitive << t.reset_event();

SC_METHOD(kill_handler);
  dont_initialize();
  sensitive << t.terminated_event();
```

```
void control()
{
  wait(10, SC_NS);
  t.reset();
  wait(10, SC_NS);
  t.kill();
  ...
```

```
void target()
{
  ...
  for (;;)
  {
    wait(ev);
    ...
  }
}
```

kill(SC_INCLUDE_DESCENDANTS)

IEEE

9

SC_CTHREAD versus SC_THREAD

```
SC_CTHREAD(CT, clock.pos());
  reset_signal_is(reset, true);
  async_reset_signal_is(areset, true);
  ct = sc_get_current_process_handle();

void CT() {
  if (reset)
    Q1 = 0;
  for (;;)
  {
    wait();
    Q1++;
  }
}
```

```
SC_THREAD(T);
  reset_signal_is(reset, true);
  async_reset_signal_is(areset, true);
  sensitive << clock.pos();
  t = sc_get_current_process_handle();

void T() {
  if (reset)
    Q8 = 0;
  for (;;)
  {
    wait();
    Q8++;
  }
}
```

```
void control()
{
  ...
  ct.disable();
  t.disable();
  ...
  ct.enable();
  t.enable();
  ...
}
```

IEEE

10

sync_reset_on/off

```
SC_THREAD(control);
SC_THREAD(target);
  // reset_signal_is(reset_sig, true);
  t = sc_get_current_process_handle();
```

```
void control()
{
  wait(...);
  t.sync_reset_on();
  wait(10, SC_NS);
  ev.notify();

  wait(10, SC_NS);
  t.sync_reset_off();
  wait(10, SC_NS);
  ev.notify();
}
```

t is reset

t awakes

```
void target()
{
  ...
  for (;;)
  {
    wait(ev);
    ...
  }
}
```

IEEE

11

Pausing Simulation

```
int sc_main(...)
{
  Top top("top");
  sc_start();
  ...
  sc_start();
  ...
  sc_start();
  ...
}
```

End of delta

End of delta

End of delta

```
void thread_process()
{
  ...
  sc_pause();
  ...
  wait(...);
  ...
  sc_pause();
  ...
  wait(...);
  ...
  sc_stop();
  ...
}
```

IEEE

12

Simulation Status

```
int sc_main(...)
{
  Top top("top");
  // sc_get_status() == SC_ELABORATION

  sc_start();
  // sc_get_status() == SC_PAUSED
  ...
  sc_start();
  ...
  sc_start();
  // sc_get_status() == SC_STOPPED
}
```

```
void thread_process()
{
  // sc_get_status() == SC_RUNNING
  ...
}
```

IEEE

13

Event Starvation

```
int sc_main(...)
{
  Top top("top");
  sc_time period(10, SC_NS);


  sc_start(period, SC_RUN_TO_TIME);

  sc_start(period, SC_EXIT_ON_STARVATION);


  while (sc_pending_activity())
    sc_start(sc_time_to_pending_activity());
  ...
}
```

Consume the entire 10ns

Return when starved

IEEE

14

sc_vector

```
SC_MODULE(my_module)
{
  sc_vector<sc_inout<int> > port_vec;
  ...
  port_vec.init(8);
```

Vector-of-ports

Set the size of the vector

```
sc_vector<sc_signal<int> > signal_vec;
...
SC_CTOR(...)
{
  signal_vec.init(8);

  m = new my_module("m");
  m->port_vec.bind( signal_vec );
  ...
```

Vector-of-signals

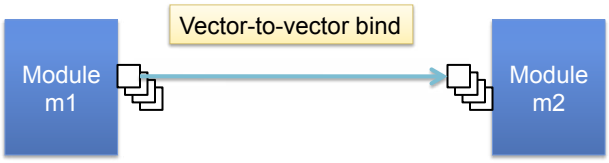
Set the size of the vector

Vector-to-vector binding

IEEE

15

Binding Vectors

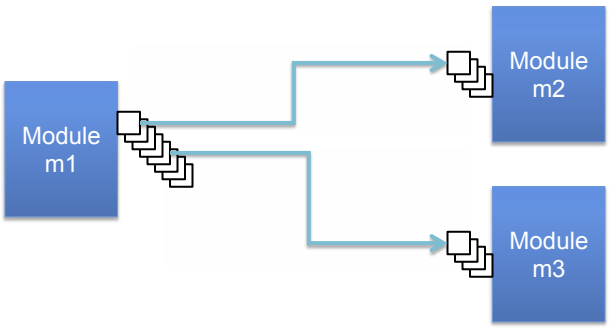



```
m1->port_vec.bind( m2->export_vec );
```

IEEE

16

Partial Binding



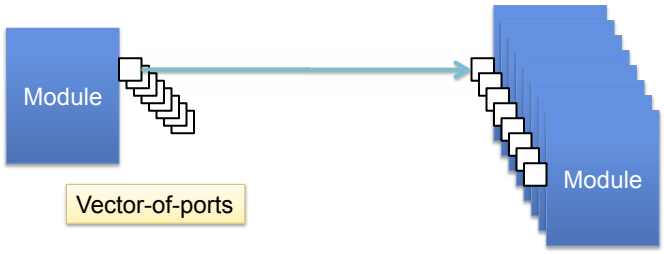

```
it = m1->port_vec.bind( m2->export_vec );

it = m1->port_vec.bind( m3->export_vec.begin(),
                        m3->export_vec.end(),
                        it );
```

IEEE

17

sc_assemble_vector



Vector-of-ports

Vector-of-modules

each with one port

```
init->port_vec.bind(
    sc_assemble_vector(module_vec,
&Target::export) );
```

IEEE

18

Verbosity Filter for INFO Reports

```
enum sc_verbosity {
  SC_NONE = 0,
  SC_LOW = 100,
  SC_MEDIUM = 200,
  SC_HIGH = 300,
  SC_FULL = 400,
  SC_DEBUG = 500
};
```

```
sc_report_handler::set_verbosity_level( SC_HIGH );
```

Sets the global maximum

```
SC_REPORT_INFO_VERB("msg_type", "msg", SC_LOW);
```

Ignored if argument > max verbosity level

cf. UVM

IEEE

19

Event List Objects

```
sc_event_and_list and_list = ev1 & ev2 & ev3;
wait(and_list);
```

```
sc_port<sc_signal_in_if<int>, 0> port;
...

void thread_process()
{
  sc_event_or_list or_list;

  for (int i = 0; i < port.size(); i++)

    or_list |= port[i]->default_event();

  wait(or_list);
  ...
}
```

Multiport

IEEE

20

Summary of New SystemC Features

- Process control – suspend, resume, disable, reset, kill, ...
  - Asynchronous resets
- Pausing simulation – sc_pause
  - Starvation policy for sc_start
  - Function to get simulation status
  - Functions to detect pending activity
  - Function to return maximum simulation time
- Vectors of sc_objects
- Verbosity level for reports

IEEE

21

Technical Enhancements

- Event list objects
- Hierarchically named events
- Process handles can be stored in a container like std::map
- Multiple writer policy for sc_signals
- Asynchronous update requests for primitive channels
- sc_mutex and sc_semaphore no longer primitive channels
- Binding operators for sc_port, sc_export now virtual
- Certain fixed-point constructors made explicit
- Preprocessor macros to return SystemC version

IEEE

22

TLM-2.0 Enhancements

- Definition of TLM-2.0 compliance
  - TLM-2.0-compliant implementation
  - Base-protocol-compliant
  - Custom-protocol-compliant
- Minor generic payload / base protocol changes
  - Option to use full set of GP attributes for DMI & debug
  - More flexibility with TLM_IGNORE_COMMAND
- Binding operators for TLM sockets now virtual
- Include file renamed <tlm> for consistency with <systemc>

IEEE

23