

New Features for Process Control in SystemC

John Aynsley, Doulos

The IEEE P1666 Working Group

CONTENTS

New Features for Process Control in SystemC

- P1666 Update
- Process Control Extensions
- Other Minor Enhancements



P1666 SystemC WG In Coordination With OSCI

Welcome to the IEEE P1666 website

P1666 Charter

The general charter of this project is to provide a C++-based standard for designers and architects who need to address complex systems that are a hybrid between hardware and software.

The specific charter of this standard is to provide a precise and complete definition of the SystemC class library including a Transaction Level Modeling library so that a SystemC implementation can be developed with reference to this standard alone. This standard is not intended to serve as a users guide or to provide an introduction to SystemC, but does contain useful information for end users.

Important Links

- [IEEE Patent Policy](#)
- [IEEE SA Advanced Membership](#)
- [P1666 Policies and Procedures](#)
- [P1666 Roster & Meeting Attendance](#)
- [Meeting Minutes](#)
- [Meeting Schedule](#)
- [Email Reflector Archive](#)
- [Technical Email Reflector Archive](#)
- [Document Repository](#)
- [How to Join the P1666 Email Reflector](#)

P1666 Update

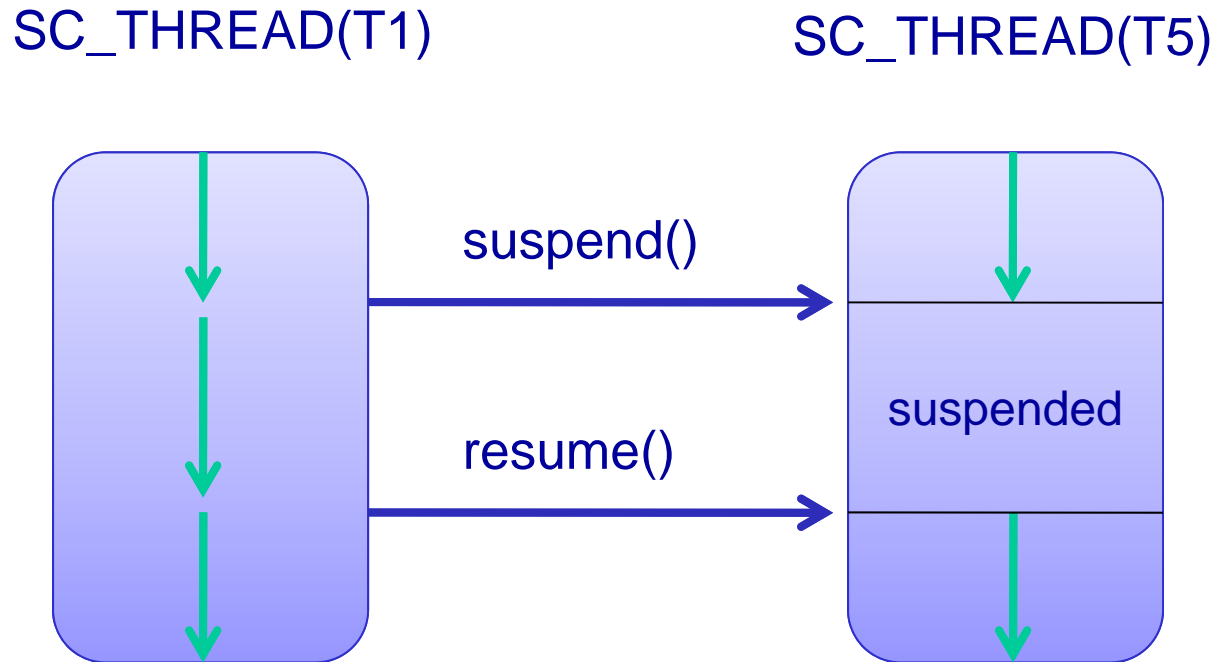
- New SystemC Standard based on
 - Old IEEE 1666-2005 SystemC Standard
 - OSCI TLM-2.0 LRM
- Enhancements already agreed
 - TLM-1 Message Passing Interface
 - Process control extensions
- Goal – IEEE standardization by end 2010

CONTENTS

New Features for Process Control in SystemC

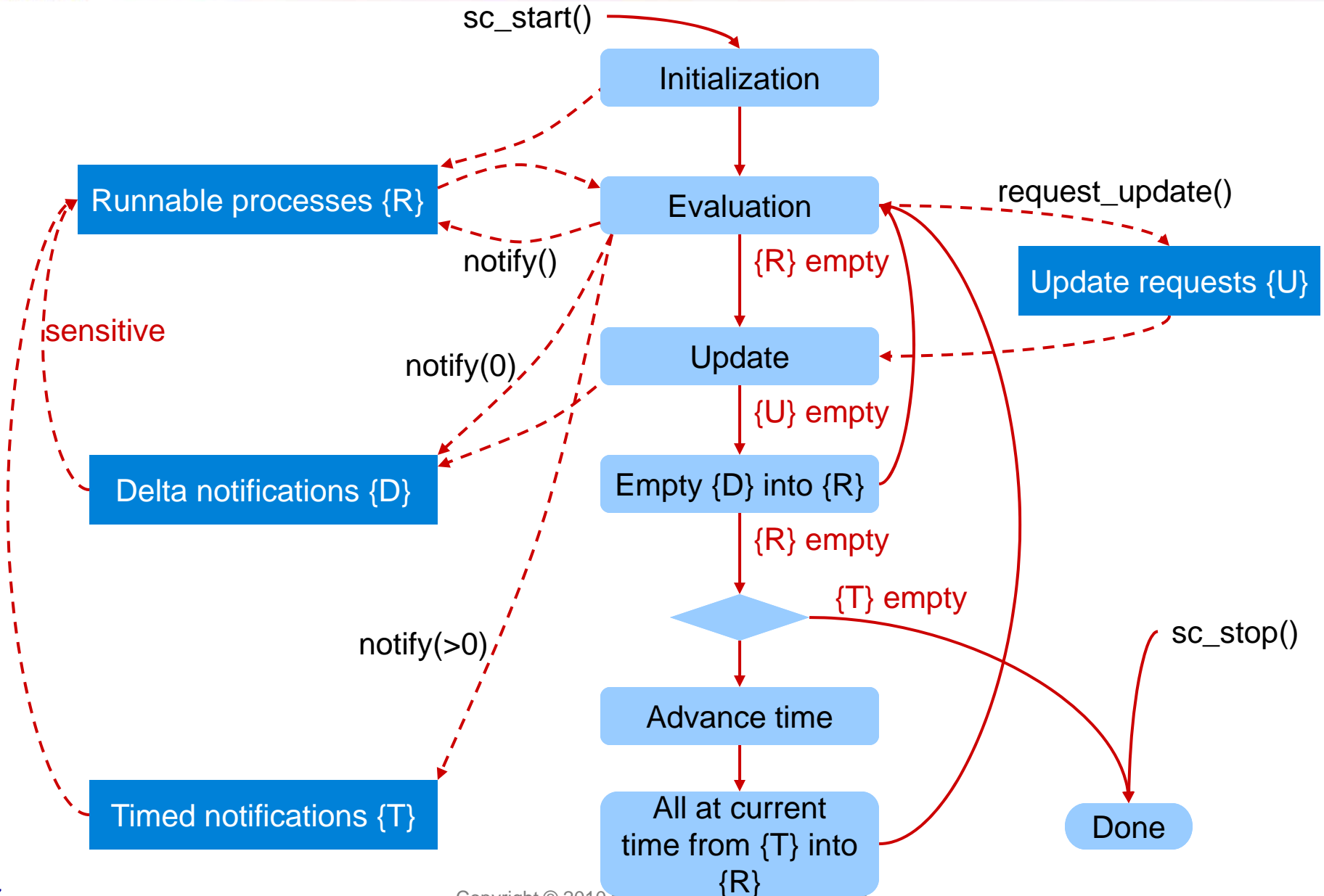
- P1666 Update
- Process Control Extensions
- Other Minor Enhancements

Process Control Extensions

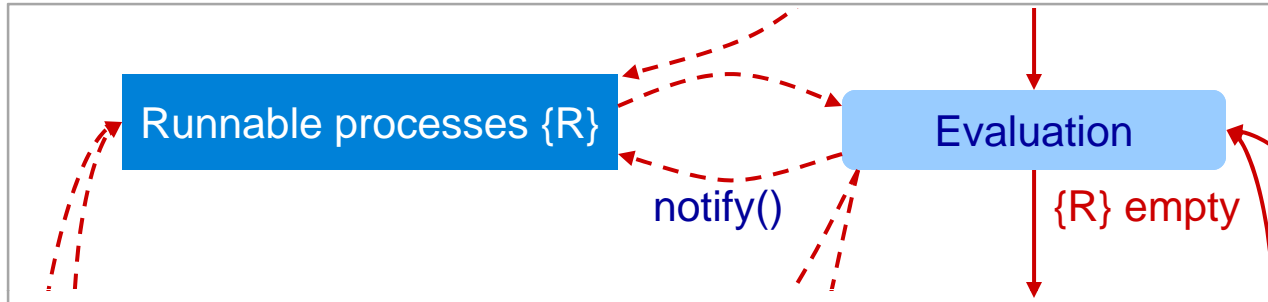


Once a process is suspended, it will not run again until resumed

The Scheduler in Detail



The Keys to the Scheduler



- Select a process from {R} in no particular order
- Run that process until it waits or returns
- Running process cannot be interrupted
- Running process can make other processes runnable

suspend/resume

```
module(sc_module_name _name) {  
    SC_THREAD(T1);  
}  
  
void T1() {  
    sc_process_handle t5 =  
        sc_spawn(sc_bind(&Top::T5, this));  
    wait(25, SC_NS);  
    t5.suspend();  
    wait(20, SC_NS);  
    t5.resume();  
}
```

```
void T5()  
{  
    for (int i = 0; i < 8; i++)  
    {  
        wait(10, SC_NS);  
        cout << ...  
    }  
}
```

T2 awakes at 10ns, 20ns, 45ns, 55ns, ...

Beware races

Use Case – Generation

```
Top(sc_module_name _name)
{
    SC_THREAD(master);
    master_h = sc_get_current_process_handle();

    SC_THREAD(generator1);
    generator1_h = sc_get_current_process_handle();

    SC_THREAD(generator2);
    generator2_h = sc_get_current_process_handle();
}

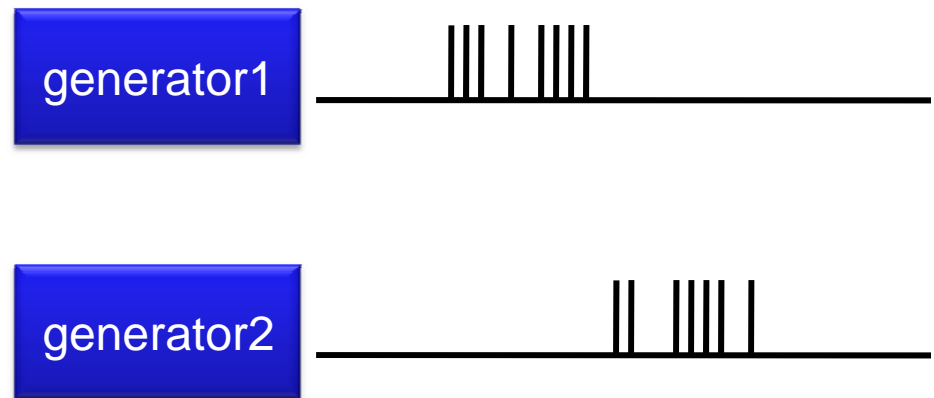
sc_process_handle master_h, generator1_h, generator2_h;
```

Use Case – Generation

```
void master()
{
    generator1_h.suspend();
    generator2_h.suspend();
    wait(...);

    generator1_h.resume();
    wait(...);
    generator1_h.suspend();


    generator2_h.resume();
    wait(...);
    generator2_h.suspend();
}
```



suspend/resume versus disable/enable

```
void T1() {  
    ...  
    t6.suspend();  
    ...  
    t6.resume();  
}
```

```
void T2() {  
    ...  
    t6.disable();  
    ...  
    t6.enable();  
}
```

```
void T6()  
{  
    wait(ev);  
    ...  
}
```

- Suppose sensitivity is satisfied while a process is suspended
 - On resume() process becomes immediately runnable
 - On enable() process does not become runnable

kill versus reset

```
void T1() {  
    ...  
    t7.kill();  
    ...  
}
```

```
void T2() {  
    ...  
    t7.reset();  
    ...  
}
```

```
void T7()  
{  
    // Reset behavior  
    ...  
    for (;;) {  
        wait(...);  
        // Normal behavior  
        ...  
    }  
}
```

- kill() → Never run again
- reset() → Start again from the top

SC_CTHREAD versus SC_THREAD

```
SC_CTHREAD(CT1, clock);  
    reset_signal_is(reset, true);
```

```
void CT1()  
{  
    if (reset)  
        Q1 = 0;  
    for (;;)   
    {  
        wait();  
        Q1++;  
    }  
}
```

```
SC_THREAD(T8);  
    reset_signal_is(reset, true);  
    sensitive << clock.pos();
```

```
void T8()  
{  
    if (reset)  
        Q8 = 0;  
    for (;;)   
    {  
        wait();  
        Q8++;  
    }  
}
```

sync_reset_on/off

```
void T2() {  
    ...  
    t7.sync_reset_on();  
    wait( clock.posedge_event() );  
    t7.sync_reset_off();  
    ...  
}
```

```
SC_THREAD(T7);  
    sensitive << clock.pos();  
  
void T7()  
{  
    // Reset behavior  
    ...  
    for (;;) {  
        wait();  
        // Normal behavior  
        ...  
    }  
}
```

Other Goodies

- SC_CTHREAD and SC_THREAD unified
- Multiple resets
 - reset_signal_is
 - async_reset_signal_is
 - sync_reset_on
 - reset()
- handle.suspend (SC_INCLUDE_DESCENDANTS);
- handle.resume (SC_INCLUDE_DESCENDANTS);
- ...
- handle.throw_it(my_exception);

CONTENTS

New Features for Process Control in SystemC

- P1666 Update
- Process Control Extensions
- Other Minor Enhancements

Unique Process Id

```
sc_process_handle a, b, c;  
a = sc_spawn(...);  
b = sc_spawn(...);  
c = b;
```

```
std::map<sc_process_handle, int> m;  
m[a] = 1;  
m[b] = 2;  
m[c] = 3;
```

$a \neq b$
 $b == c$
 $(a < b) \parallel (b < a)$
 $!(b < c) \ \&\& \ !(c < b)$

$m[a] == 1$
 $m[b] == 3$
 $m[c] == 3$

- `sc_process_handle::operator<`
- `sc_process_handle::swap()`

sc_pause

```
struct M: sc_module {  
    ...  
    void my_process  
    {  
        ...  
        sc_pause();  
        ...  
    }  
};
```

```
int sc_main(...)  
{  
    ...  
    sc_start();  
    ...  
    if (sc_get_status() == SC_PAUSED)  
        sc_start();  
    ...  
}
```

- sc_pause() is like sc_stop() except sc_start() can be called again

sc_get_status

```
if (sc_get_status() & (SC_PAUSED |  
                      SC_STOPPED |  
                      SC_END_OF_SIMULATION) )  
  
...
```

SC_ELABORATION
SC_BEFORE_END_OF_ELABORATION
SC_END_OF_ELABORATION
SC_START_OF_SIMULATION
SC_RUNNING
SC_PAUSED
SC_STOPPED
SC_END_OF_SIMULATION

Other Possible Enhancements

- Macros to get the SystemC version number
- Events as named objects
- Event and/or lists as proper objects
- New container classes to create array-of-port, etc.