How to fork threads in SystemC just like in SystemVerilog and Specman-e

Stefan-Tiberiu Petre

Independent verification consultant

SYSTEMS INITIATIVE

SYSTEMC EVOLUTION DAY
OCT 28, 2021 | VIRTUAL WORKSHOP

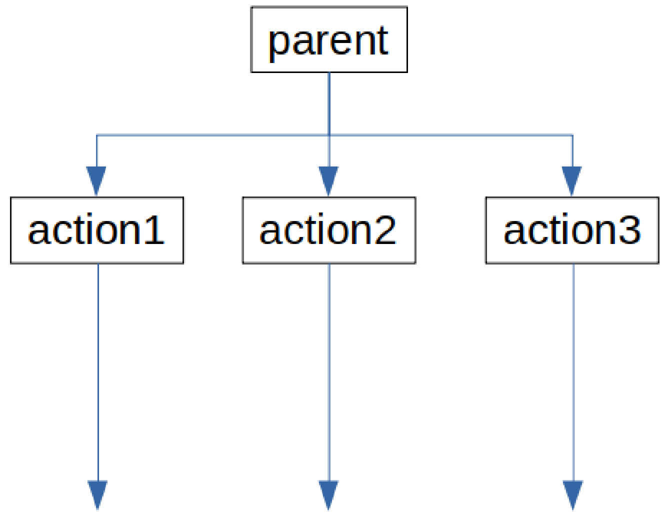
Copyright Permission

About me

- **Name**: Ștefan-Tiberiu Petre
- **Occupation**: Hardware Verification Engineer, 13 years experience
- **Expertise:** Functional Verification
  - SystemVerilog/UVM
  - Specman-e/eRM/UVMe
  - SystemC – for reference models
- **Other interests:**
  - Free and open source EDA tools
  - Simulation
  - Machine learning

Dynamic thread creation

- The creation of new simulation threads
  - after elaboration has finished
  - at simulation times >= 0
- Also known as "forking"



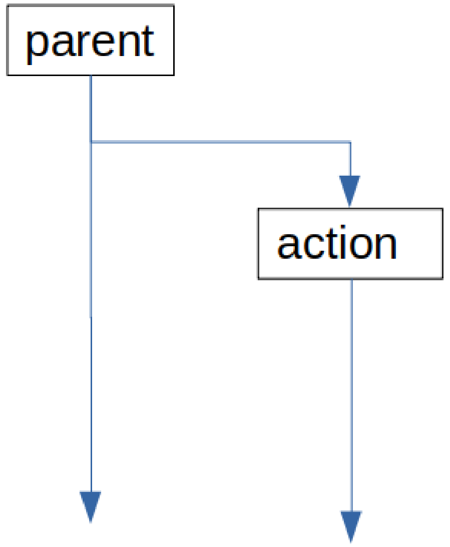
Outline

- Forking threads in SystemC – what's currently supported?
- Forking threads in SystemVerilog and Specman-e
- Forking threads in SystemC just like in SystemVerilog and Specman-e using the sc_enhance library
- Usecases
- Other features of sc_enhance
- Conclusions
- QnA

Dynamic thread creation in SystemC – sc_spawn

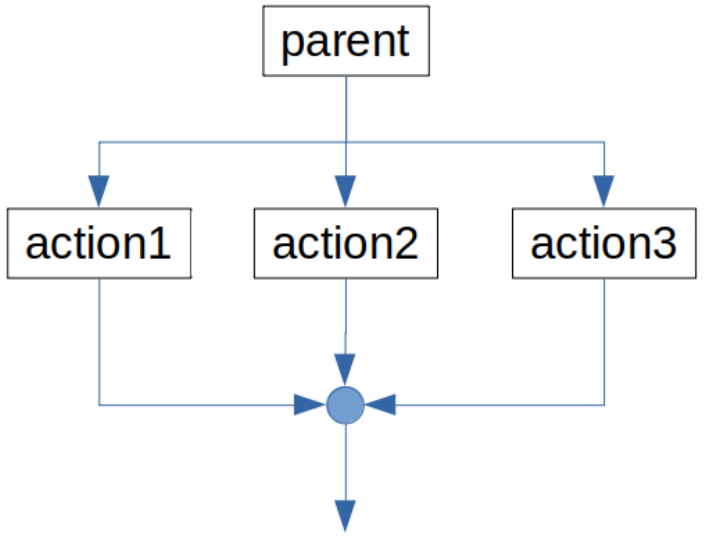
- See IEEE 1666-2011 Section 5.5

```
struct my_mod: public sc_module {

  void action() { ... }

  void master_thread() {
    wait(10, SC_NS);
    sc_spawn( sc_bind(&my_mod::action, this) );
  }

  SC_CTOR(my_mod) {
    SC_THREAD(master_thread);
  }

};
```



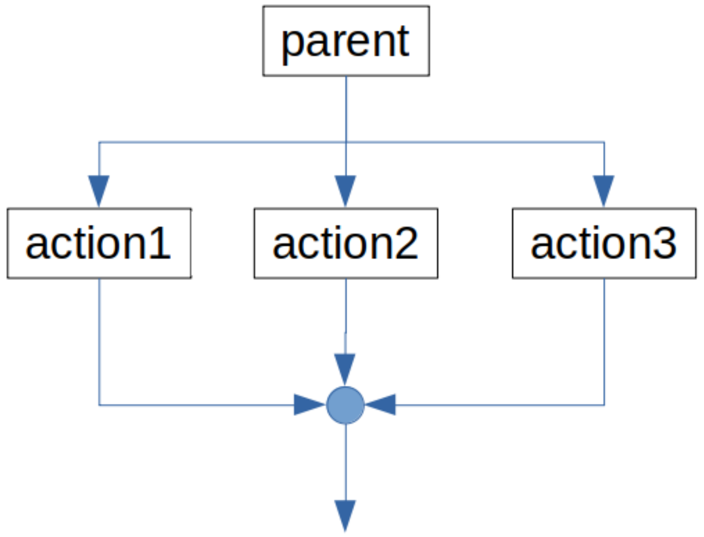
SC_FORK – SC_JOIN (LRM Section 5.5.7)

```
void action3(const bool& in_value, int& out_value) { ... }

struct my_mod: public sc_module {

  void action1() { ... }
  int  action2(int x) { ... }

  void master_thread() {
    int ret_val;
    bool actual_in_value;
    int actual_out_value;

    SC_FORK
      sc_spawn(           sc_bind(&my_mod::action1, this) ),
      sc_spawn( &ret_val, sc_bind(&my_mod::action2, this, 5) ),
      sc_spawn(           sc_bind(&action3,
            sc_cref(actual_in_value),
            sc_ref(actual_out_value) ) )
    SC_JOIN

  }

  SC_CTOR(my_mod) {
    SC_THREAD(master_thread);
  }

};
```



SC_FORK – SC_JOIN with lambdas (C++11 and later)

```
struct my_mod: public sc_module {

  void master_thread() {

    SC_FORK
      sc_spawn(           [&]()        { /* action1 */ } ),
      sc_spawn(           [&]()        { /* action2 */ } ),
      sc_spawn( sc_bind( [&](int arg){ /* action3 */ }, 5 )
    SC_JOIN

  }

  SC_CTOR(my_mod) {
    SC_THREAD(master_thread);
  }

};
```



SystemC – 2 types of fork

- "join none" fork using sc_spawn
  - Parent thread resumes immediately
- "join all" fork using SC_FORK-SC_JOIN
  - Parent resumes only when all forked threads have finished

SystemVerilog and Specman-e – many types of fork

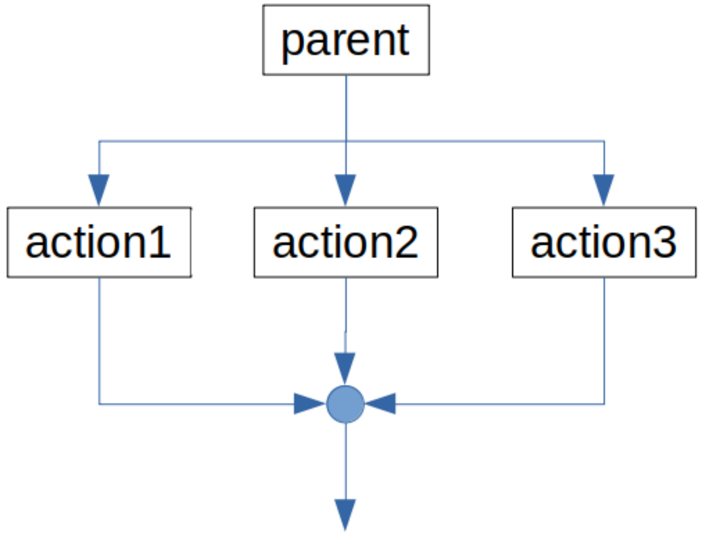
- fork – join / all of
- fork – join_any
- first of
- fork – join_none / start
- all of for each
- first of for each

fork - join / all of

- Already supported by SystemC as SC_FORK-SC_JOIN

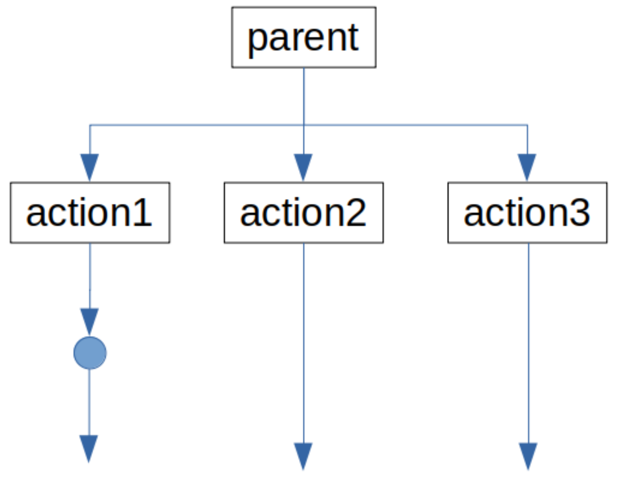
```
// SystemVerilog
fork
  action1();
  action2();
  action3();
join
```

```
// Specman e
all of {
  { action1() };
  { action2() };
  { action3() };
};
```



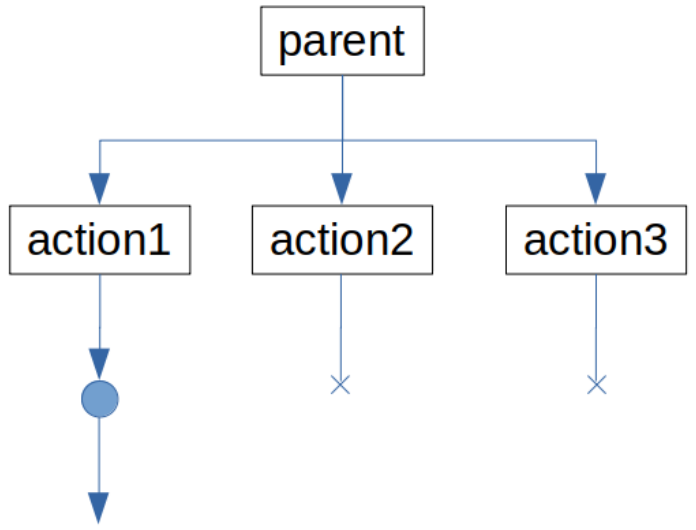
fork – join_any (SV)

```
// SystemVerilog
fork
  action1();
  action2();
  action3();
join_any
```



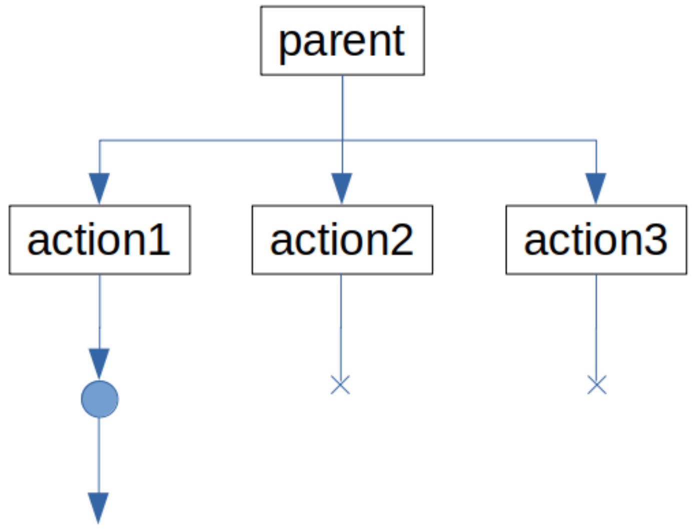
first of (e)

```
// Specman e
first of {
  { action1() };
  { action2() };
  { action3() };
};
```



first of (SV workaround)

```
fork
  begin
    fork
      action1();
      action2();
      action3();
    join_any
    disable fork;
  end
join
```

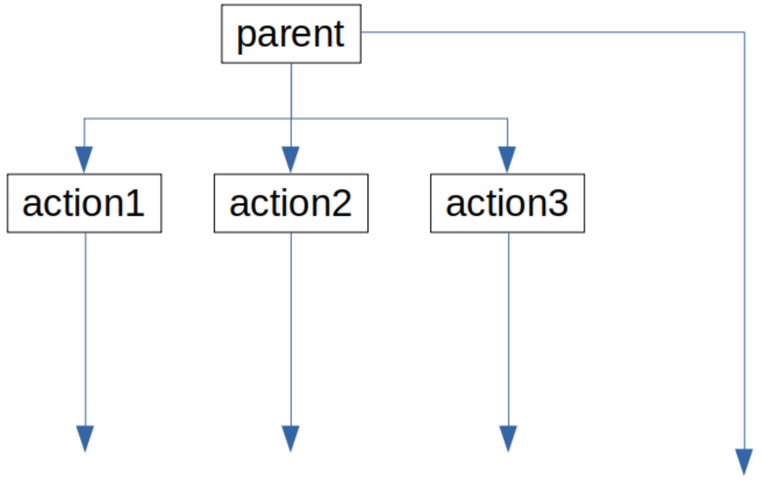


fork – join_none / start

- Already supported by SystemC as sc_spawn

```
// SystemVerilog
fork
  action1();
  action2();
  action3()
join_none
```

```
// Specman e
start action1();
start action2();
start action3();
```



Disadvantages of current SystemC support

- Too verbose (especially the classical C++ variant)
  - Must call sc_spawn everytime
  - In the absence of modern C++ thread functions can't be coded inline (no lambdas)
- No obvious support for join_any
- No obvious support for "first of"
- No obvious support for spawning multiple processes in a loop and joining them in various ways
  - also a problem for SV
  - used to be a problem for Specman-e

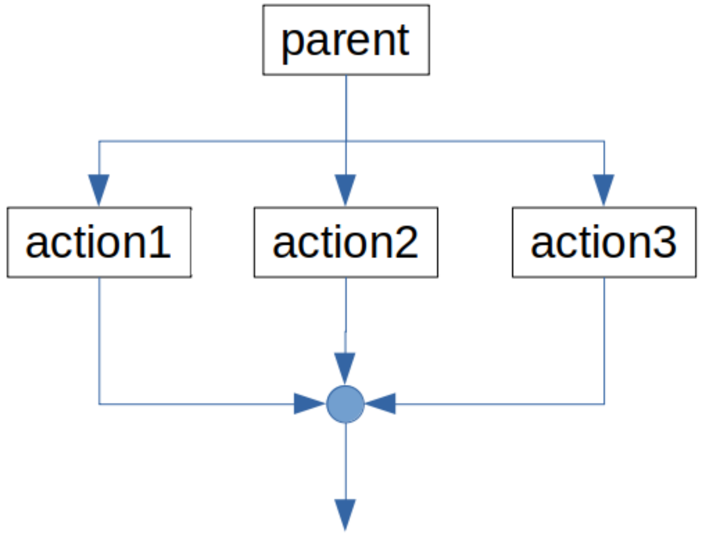
Introducing sc_enhance

- git clone https://github.com/verificationcontractor/sc_enhance.git
- Header only library (#include "sc_enhance.hpp" after <systemc>)
- Uses Modern C++ (-std=c++11 and later)
- A collection of macros and classes meant to simplify the SystemC language
- Can be added to future SystemC standards

SC_FORK – SC_JOIN

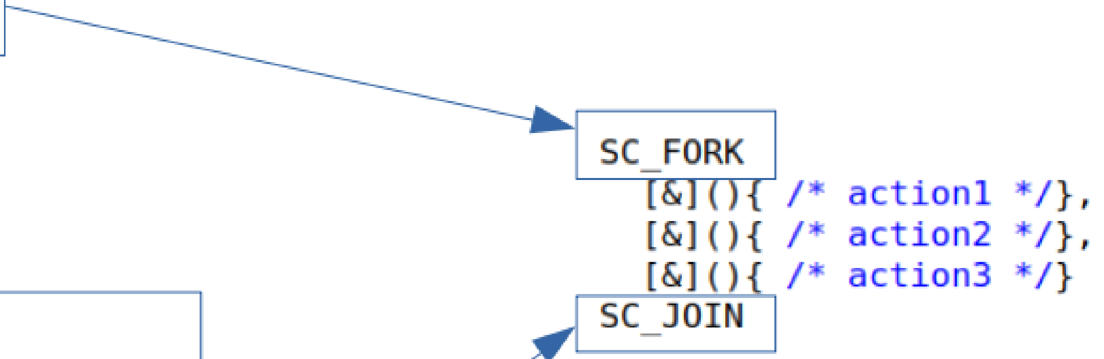
- Simplified version of what is already supported (no need to call sc_spawn)

```
struct my_mod: public sc_module {

  void master_thread() {
    SC_FORK
      [&](){ /* action1 */},
      [&](){ /* action2 */},
      [&](){ /* action3 */}
    SC_JOIN
  }

  SC_CTOR(my_mod2) {
    SC_THREAD(master_thread);
  }

};
```



SC_FORK – SC_JOIN

```
{
  std::function<void(void)> forkees[] = {

    [&](){ /* action1 */},
    [&](){ /* action2 */},
    [&](){ /* action3 */}

  };
  sc_core::sc_join            join;
    sc_core::sc_spawn_options opts;
    std::string s = "";
    for ( unsigned int i = 0; i < sizeof(forkees)/sizeof(std::function<void(void)>); i++ ) {
      s =
        std::string(basename())
        + "_thread_" +
        sc_thread_id_gen::get_id()
        + "_" + std::to_string(__LINE__)
        + "_" + std::to_string(i);
        sc_process_handle handle = sc_spawn(forkees[i], s.c_str(), &opts);
        join.add_process(handle);
    }
  join.wait();
}
```


```
SC_FORK
  [&](){ /* action1 */},
  [&](){ /* action2 */},
  [&](){ /* action3 */}
SC_JOIN
```

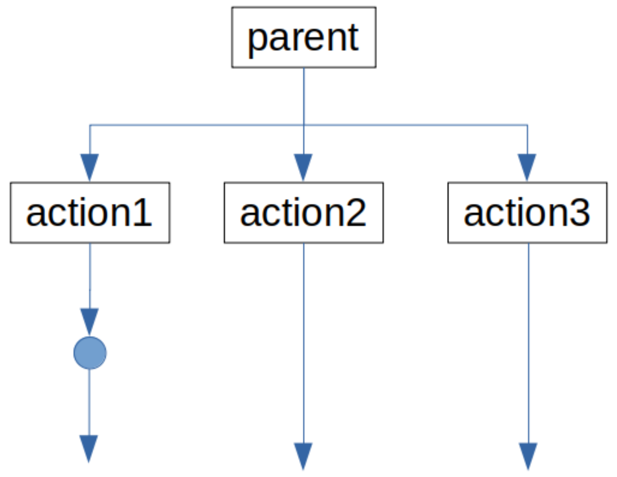
SC_CFORK – SC_CJOIN

- Clocked threads are also supported
- Don't spawn clocked threads from unclocked ones, it won't work

```
struct my_mod: public sc_module {

  sc_clock clk;

  void master_thread() {
    SC_CFORK(clk)
      [&]() { /* action1 */},
      [&]() { /* action2 */},
      [&]() { /* action3 */}
    SC_CJOIN
  }

  SC_CTOR(my_mod) {
    SC_CTHREAD(master_thread, clk)
  }

};
```

SC_FORK – SC_JOIN_ANY

```
struct my_mod: public sc_module {

  void master_thread() {
    SC_FORK
      [&]() { /* action1 */},
      [&]() { /* action2 */},
      [&]() { /* action3 */}
    SC_JOIN_ANY
  }

  SC_CTOR(my_mod) {
    SC_THREAD(master_thread);
  }

};
```



SC_FORK – SC_JOIN_ANY

```
{
  std::function<void(void)> forkees[] = {
```

```
    [&](){ /* action1 */},
    [&](){ /* action2 */},
    [&](){ /* action3 */}
```

```
  };
  sc_core::sc_join_any join_any;
    sc_core::sc_spawn_options opts;
    std::string s = "";
    for ( unsigned int i = 0; i < sizeof(forkees)/sizeof(std::function<void(void)>); i++ ) {
      s =
        std::string(basename())
        + "_thread_" +
        sc_thread_id_gen::get_id()
        + "_" + std::to_string(__LINE__)
        + "_" + std::to_string(i);
        sc_process_handle handle = sc_spawn(forkees[i], s.c_str(), &opts);
        join_any.add_process(handle);
    }
  join_any.wait();
}
```

```
SC_FORK
    [&](){ /* action1 */},
    [&](){ /* action2 */},
    [&](){ /* action3 */}
SC_JOIN_ANY
```

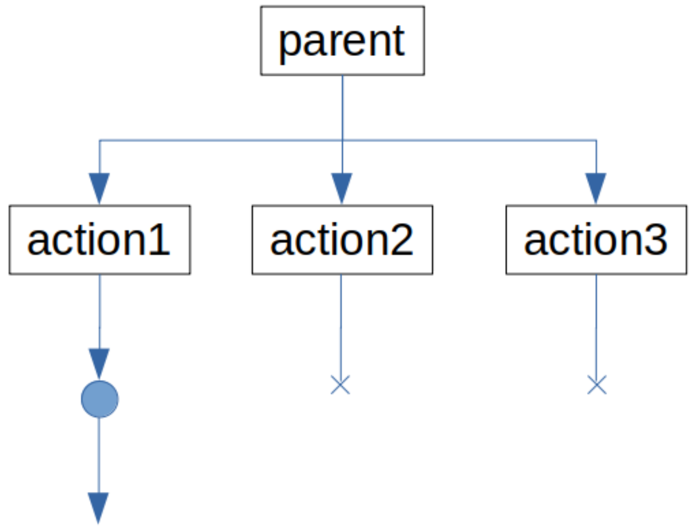
SC_CFORK – SC_CJOIN_ANY

- Clocked threads are also supported
- Don't spawn clocked threads from unclocked ones, it won't work

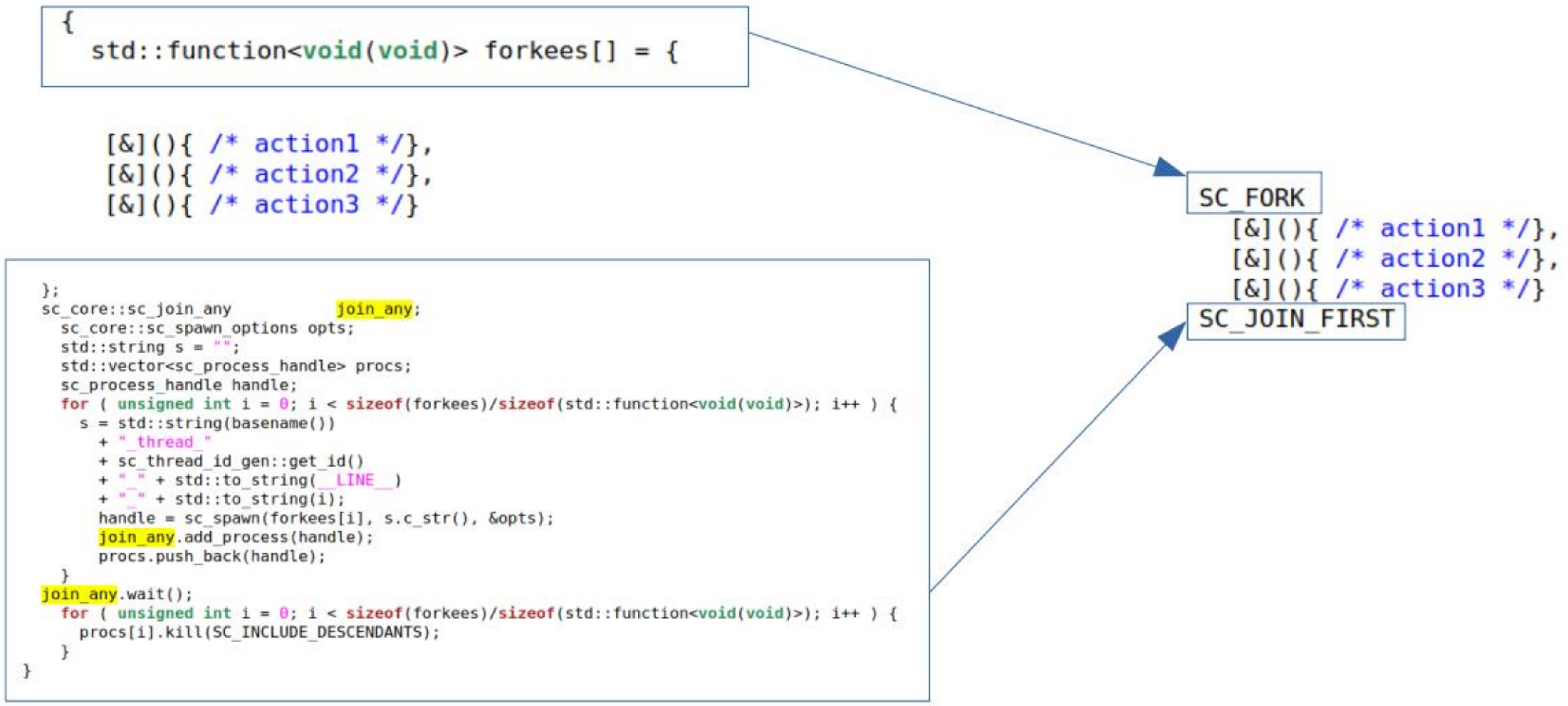
```
struct my_mod: public sc_module {

  sc_clock clk;

  void master_thread() {
    SC_CFORK(clk)
      [&]() { /* action1 */},
      [&]() { /* action2 */},
      [&]() { /* action3 */}
    SC_CJOIN_ANY
  }

  SC_CTOR(my_mod) {
    SC_CTHREAD(master_thread, clk);
  }

};
```

SC_FORK – SC_JOIN_FIRST

```
struct my_mod: public sc_module {

  void master_thread() {
    SC_FORK
      [&]() { /* action1 */},
      [&]() { /* action2 */},
      [&]() { /* action3 */}
    SC_JOIN_FIRST
  }

  SC_CTOR(my_mod) {
    SC_CTHREAD(master_thread);
  }

};
```



SC_FORK – SC_JOIN_FIRST



```
{
  std::function<void(void)> forkees[] = {

    [&](){ /* action1 */},
    [&](){ /* action2 */},
    [&](){ /* action3 */}

  };
  sc_core::sc_join_any           join_any;
    sc_core::sc_spawn_options opts;
    std::string s = "";
    std::vector<sc_process_handle> procs;
    sc_process_handle handle;
    for ( unsigned int i = 0; i < sizeof(forkees)/sizeof(std::function<void(void)>); i++ ) {
      s = std::string(basename())
        + "_thread_"
        + sc_thread_id_gen::get_id()
        + "_" + std::to_string(__LINE__)
        + "_" + std::to_string(i);
        handle = sc_spawn(forkees[i], s.c_str(), &opts);
        join_any.add_process(handle);
        procs.push_back(handle);
    }
  join_any.wait();
    for ( unsigned int i = 0; i < sizeof(forkees)/sizeof(std::function<void(void)>); i++ ) {
      procs[i].kill(SC_INCLUDE_DESCENDANTS);
    }
}
```

```
SC_FORK
  [&](){ /* action1 */},
  [&](){ /* action2 */},
  [&](){ /* action3 */}
SC_JOIN_FIRST
```

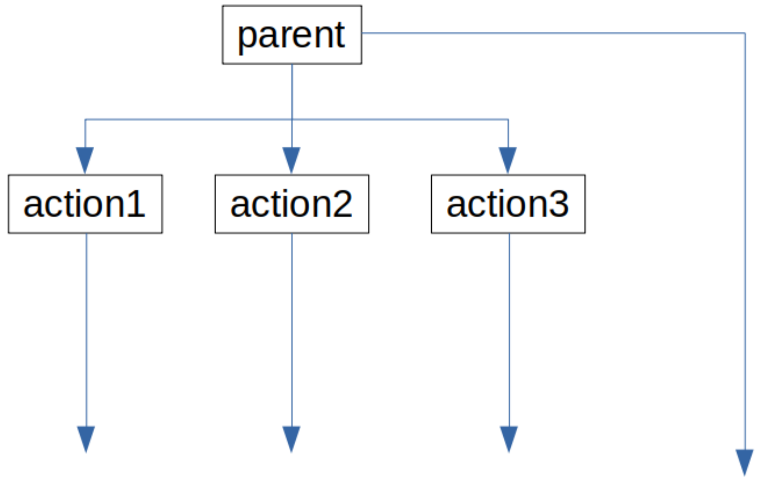
SC_CFORK – SC_CJOIN_FIRST

- Clocked threads are also supported
- Don't spawn clocked threads from unclocked ones, it won't work

```
struct my_mod: public sc_module {

  sc_clock clk;

  void master_thread() {
    SC_CFORK(clk)
      [&]() { /* action1 */},
      [&]() { /* action2 */},
      [&]() { /* action3 */}
    SC_CJOIN_FIRST
  }

  SC_CTOR(my_mod) {
    SC_CTHREAD(master_thread, clk);
  }

};
```

SC_FORK – SC_JOIN_NONE

```
struct my_mod: public sc_module {

  void master_thread() {
    SC_FORK
      [&]() { /* action1 */},
      [&]() { /* action2 */},
      [&]() { /* action3 */}
    SC_JOIN_NONE
  }

  SC_CTOR(my_mod) {
    SC_CTHREAD(master_thread);
  }

};
```



SC_FORK – SC_JOIN_NONE

```
{
  std::function<void(void)> forkees[] = {
```

```
    [&](){ /* action1 */},
    [&](){ /* action2 */},
    [&](){ /* action3 */}
```

```
    };
    sc_core::sc_spawn_options opts;
    std::string s = "";
    for ( unsigned int i = 0; i < sizeof(forkees)/sizeof(std::function<void(void)>); i++ ) {
      s = std::string(basename())
        + "_thread_" + sc_thread_id_gen::get_id()
        + "_" + std::to_string(__LINE__)
        + "_" + std::to_string(i);
      sc_process_handle handle = sc_spawn(forkees[i], s.c_str(), &opts);
    }
  }
```

```
SC_FORK
    [&](){ /* action1 */},
    [&](){ /* action2 */},
    [&](){ /* action3 */}
SC_JOIN_NONE
```

SC_CFORK – SC_CJOIN_FIRST

- Clocked threads are also supported
- Don't spawn clocked threads from unclocked ones, it won't work

```
struct my_mod: public sc_module {

  sc_clock clk;

  void master_thread() {
    SC_CFORK(clk)
      [&]() { /* action1 */},
      [&]() { /* action2 */},
      [&]() { /* action3 */}
    SC_CJOIN_NONE
  }

  SC_CTOR(my_mod) {
    SC_CTHREAD(master_thread, clk);
  }

};
```

Spawn threads in a loop (SV vs. e)

```
for(int i=1; i<=3; i++) begin : for_loop
  fork
    automatic int k = i;
    begin
      thread_template(i);
    end
  join_none
end : for_loop
```

```
unit test {
  children : list of child is instance;

  for each in children {
    start it.thread_template();
  };

  do_something()@sys.any is {
    all of for each in children {
      it.thread_template();
    };

    first of for each in children {
      it.thread_template();
    };
  };

};
```

Spawn threads in a loop (sc_enhance)

```
// Declare the list of forks (it's actually a vector of lambdas)
sc_fork_list proc_list;
// Populate the list of forks
for(int i = 0; i < 10; i++) {
  proc_list.push_back(sc_bind([&](int ii){
        std::cout<< "SC_JOIN_NONE in for-loop: Thread " << ii << " begin" << std::endl;
        wait(1, SC_NS);
        std::cout<< "SC_JOIN_NONE in for-loop: Thread " << ii << " end" << std::endl;
        }, i));
}
// Spawn the list of forks
SC_FORK_JOIN(proc_list);
// or
SC_FORK_JOIN_ANY(proc_list);
// or
SC_FORK_JOIN_FIRST(proc_list);
// or
SC_FORK_JOIN_NONE(proc_list);
```

SC_JOIN - usecases

- Drive/monitor multiple interfaces at the same time
- One interface multiple data streams

```
SC_FORK
  [&]() { UVM_DO_ON(ethernet_traffic_seq, eth_sqr); },
  [&]() { UVM_DO_ON(axis_traffic_seq, axis_sqr); },
  [&]() { monitor_status_signals(); }
SC_JOIN
```

```
SC_FORK
  [&]() { UVM_DO_ON(seq1, eth_sqr); },
  [&]() { UVM_DO_ON(seq2, eth_sqr); }
SC_JOIN
```

SC_JOIN_ANY - usecases

- Legal timeout
- Horse race simulation

```
SC_FORK
  [&]() { wait(10, SC_MS); },
  [&]() {
    for(int i = 0; i < 10000; i++) {
      UVM_DO(transaction);
    }
  }
SC_JOIN_ANY
// Do something interesting afterwards
// ...
```

```
SC_FORK
  [&]() { horse1(); },
  [&]() { horse2(); },
  // ...
  [&]() { horseN(); }
SC_JOIN_ANY
```

SC_JOIN_FIRST - usecases

- Illegal Timeout
- Reset handling

```
SC_FORK
  // Timeout thread
  [&]() {
    wait(10, SC_MS);
    UVM_ERROR("TIMEOUT_ERR", "Transaction timeout");
  },
  // Transaction thread
  [&]() {
    drive_req();
    wait_ack();
  }
SC_JOIN_FIRST
```

```
// Wait first reset
wait_for_reset();
// Collect transactions and restart on future resets
while(1) {
  SC_FORK
    [&]() { wait_for_reset(); },
    [&]() { collect_transactions(); }
  SC_JOIN_FIRST
}
```

SC_JOIN_NONE - usecases

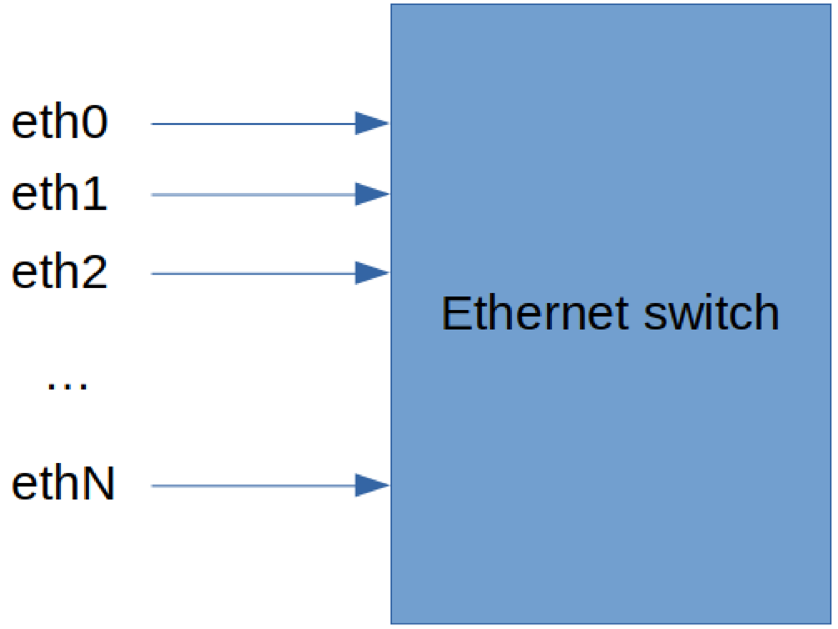
- Atypical join conditions

```
SC_FORK
  [&]() {
    // Do something
    event1.notify();
  },
  [&]() {
    // Do something
    event2.notify();
  },
  [&]() {
    // Do something
    event3.notify();
  }
SC_JOIN_NONE
// Wait join condition
wait( (event1 & event2) | event3 );
```

Spawn threads in a loop - usecases

- Parametrizable number of identical interfaces

```
sc_fork_list thread_list;
// Populate thread list
for(int i = 0; i < NR_ETH_IFS; i++) {
  thread_list.push_back(sc_bind([&](int if_idx) {
        UVM_DO_ON(eth_seq, p_sequencer.eth_sqr[if_idx]);
  }, i));
}
// Spawn thread list
SC_FORK_JOIN(thread_list);
// or
SC_FORK_JOIN_ANY(thread_list);
// or
SC_FORK_JOIN_FIRST(thread_list);
// or
SC_FORK_JOIN_NONE(thread_list);
```



Other features in sc_enhance

- Simplified process declarations

```
SC_METHOD_DECLARE(count_logic)
  sensitive << clk.pos();
SC_METHOD_BEGIN
    if(reset.read() == 1)
        count = 0;
    else
        count++;
SC_METHOD_END
```

```
SC_THREAD_DECLARE(test_thread)
    dont_initialize();
    sensitive << clk.pos();
SC_THREAD_BEGIN
    while(1) {
        if(count % 2 == 0)
          std::cout <<
            "Counter is even." <<
            std::endl;
        wait(clk.posedge_event());
    }
SC_THREAD_END
```

```
SC_CTHREAD_DECLARE(test_cthread, clk.pos())
SC_CTHREAD_BEGIN
    while(1) {
        if(count % 2 == 1)
          std::cout <<
            "Counter is odd." <<
            std::endl;
        wait();
    }
SC_CTHREAD_END
```

Other features in sc_enhance (2)

- Simplified constructors

```
SC_MODULE(demo) {
    SC_CONS(demo) { /* implement constructor here */}

    SC_CONS(demo, int x, int y) { /* implement constructor with arguments here */ }

    SC_CONS_EMPTY(demo, double d);
};

SC_CONS_IMPLEMENT(demo, double d) {
    // Implement constructor with argument outside of module class here
}
```

Other features in sc_enhance (3)

- Method ports

```
struct producer: public sc_core::sc_module {
    SC_HAS_PROCESS(producer);

    sc_out_method_port_declare(send_value, void(int));

    void run() {
        for(int i; i<5; i++)
            send_value(i);
    }

    producer(sc_core::sc_module_name name): sc_core::sc_module(name) {
        SC_THREAD(run);
    }
};
```

```
struct consumer: public sc_core::sc_module {

    int sum = 5;

    sc_in_method_port_declare(get_value, void(int),
    [&](int x) {
        std::cout << "got " << x << std::endl;
        std::cout << "sum is " << sum << std::endl;
        sum += x;
    });

    consumer(sc_core::sc_module_name name): sc_core::sc_module(name) {}

};
```

```
struct tb: public sc_core::sc_module {
    producer prod {"prod"};
    consumer cons {"cons"};

    tb(sc_core::sc_module_name name): sc_core::sc_module(name) {
        prod.send_value.connect(cons.get_value);
    }
};
```

C++ Standard support

- Simplified forks: c++11, c++14, c++17
- Simplified process declaration: c++11, c++14, c++17
- Simplified constructors: c++11, c++14, c++17
- Simplified signal and instance declarations: c++11, c++14, c++17
- Simplified signal connections: c++11, c++14, c++17
- Method ports: c++14, c++17

GDB breakpoints in lambdas

- Lambdas are inlined and optimized at compilation time
- Stepping through the lines in a lambda will have unpredictable effects
  - e.g. jumps at the beginning of the lambda after each line and then to the next line
- g++ -g -Og -std=c++17 -lsystemc -o sim sim.cpp
- Use the -Og optimization flag to fix this

Incorporation into the SystemC standard

- sc_enhance is made of 3 headers:
  - sc_thread_process.h - modified version of file with same name in the SystemC source code
  - sc_method_ports.hpp - method ports classes and macros
  - sc_enhance.hpp - includes the other 2 headers + the rest of the classes and macros
- Incorporation into the SystemC library can be done in one of the following ways:
  - As is + replacing sc_thread_process.h in SystemC
  - Split into multiple headers and add them to the SystemC project
  - Modify existing header files in SystemC by adding the extra classes and macros from sc_enhance

Incorporation into the SystemC standard

- Sections in the standard that might require changes:
  - 5.2 - add extra documentation for the simplified process declarations, simplified constructors and simplified signal/instance declarations and connections
  - 5.5 - add extra documentation for the new types of SC_FORK
  - Add an extra chapter for method ports
- Potential issues
  - SC_FORK and SC_JOIN macros from sc_enhance override those from SystemC – potential backward compatibility issues
  - Method ports are not thread safe, must use mutexes

Conclusions

- sc_enhance offers a more versatile way of forking threads by importing features from SV and e and even overcomming some limitations present in SV and e, thus making SystemC a more powerful language
- Using features from modern C++ (c++11 and later) SystemC code can become more concise, more readable and easier to write.
- sc_enhance can be incorporated into the SystemC library either "as is" or sligthly modified to ensure backward compatibility
- Simplified method declarations make it easier to write RTL code in SystemC
- Method ports make it easier to write SVPs and TLM models (both LT and AT)

Bibliography

- IEEE 1666-2011 Standard for Standard SystemC Language Reference Manual
- IEEE 1800-2017 Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language
- IEEE 1647-2016 Standard for the Functional Verification Language e
- https://sclive.wordpress.com/2008/01/10/systemc-tutorial-threads-methods-and-sc_spawn/
- https://forums.accellera.org/topic/6211-how-can-i-implement-sc_fork-join_any-sc_fork-join_none/

Thank You!