2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

The Proposed Accellera SystemC Synthesizable Subset

Mike Meredith

Vice Chair – Accellera Synthesis Working Group

Cadence Design Systems

cadence®

SystemC Synthesizable Subset Work

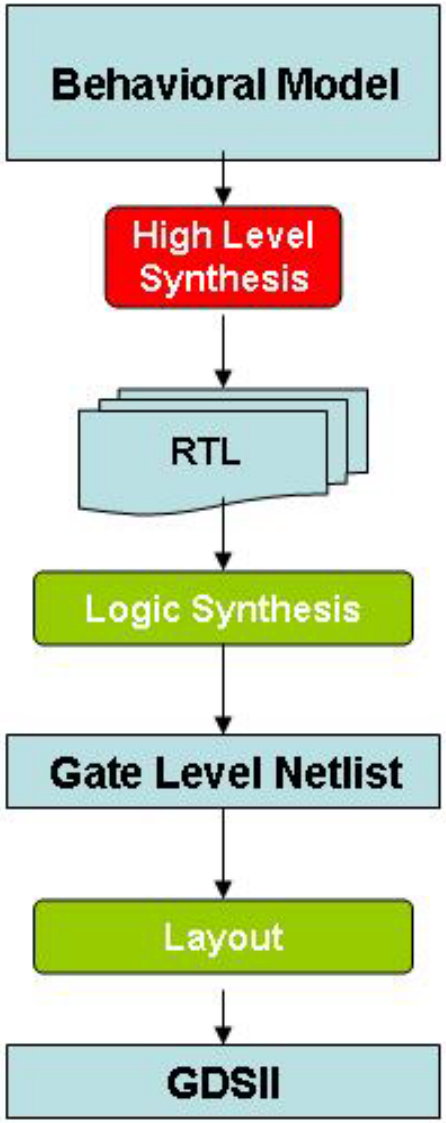
- Development of a description of a synthesizable subset of SystemC
- Started in the OSCI Synthesis Working Group
- Current work is in Accellera Systems Initiative Synthesis Working Group
- Draft has been proposed for approval as a new standard
- Many contributors over a number of years
- Broadcom, Cadence, Calypto, Forte, Fujitsu, Freescale, Global Unichip, Intel, ITRI, Mentor, NEC, NXP, Offis, Qualcomm, Sanyo, Synopsys

General Principles

- Define a meaningful minimum subset
  - Establish a baseline for transportability of code between HSL tools
  - Leave open the option for vendors to implement larger subsets and still be compliant
- Include useful C++ semantics if they can be known statically – e.g., templates

Scope of the Proposed Standard

- Synthesizable SystemC
- Defined within IEEE 1666-2011
- Covers behavioral model in SystemC for synthesis
  - SC_MODULE, SC_CTHREAD, SC_THREAD
- Covers RTL model in SystemC for synthesis
  - SC_MODULE, SC_METHOD
- Main emphasis of the document is on behavioral model synthesizable subset for high-level synthesis



Scope of the Planned Standard

SystemC Elements

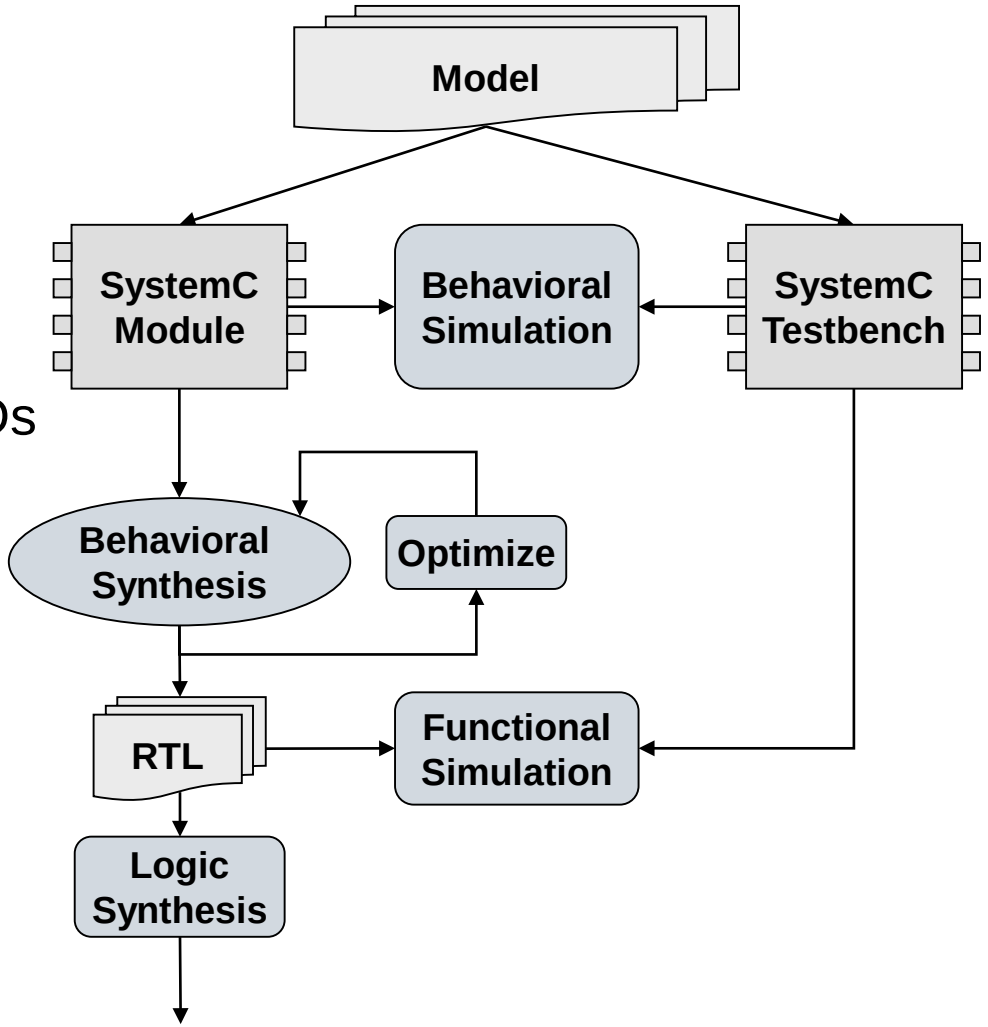
- Modules
- Processes
  - SC_CTHREAD
  - SC_THREAD
  - SC_METHOD
- Reset
- Signals, ports, exports
- SystemC datatypes

C++ Elements

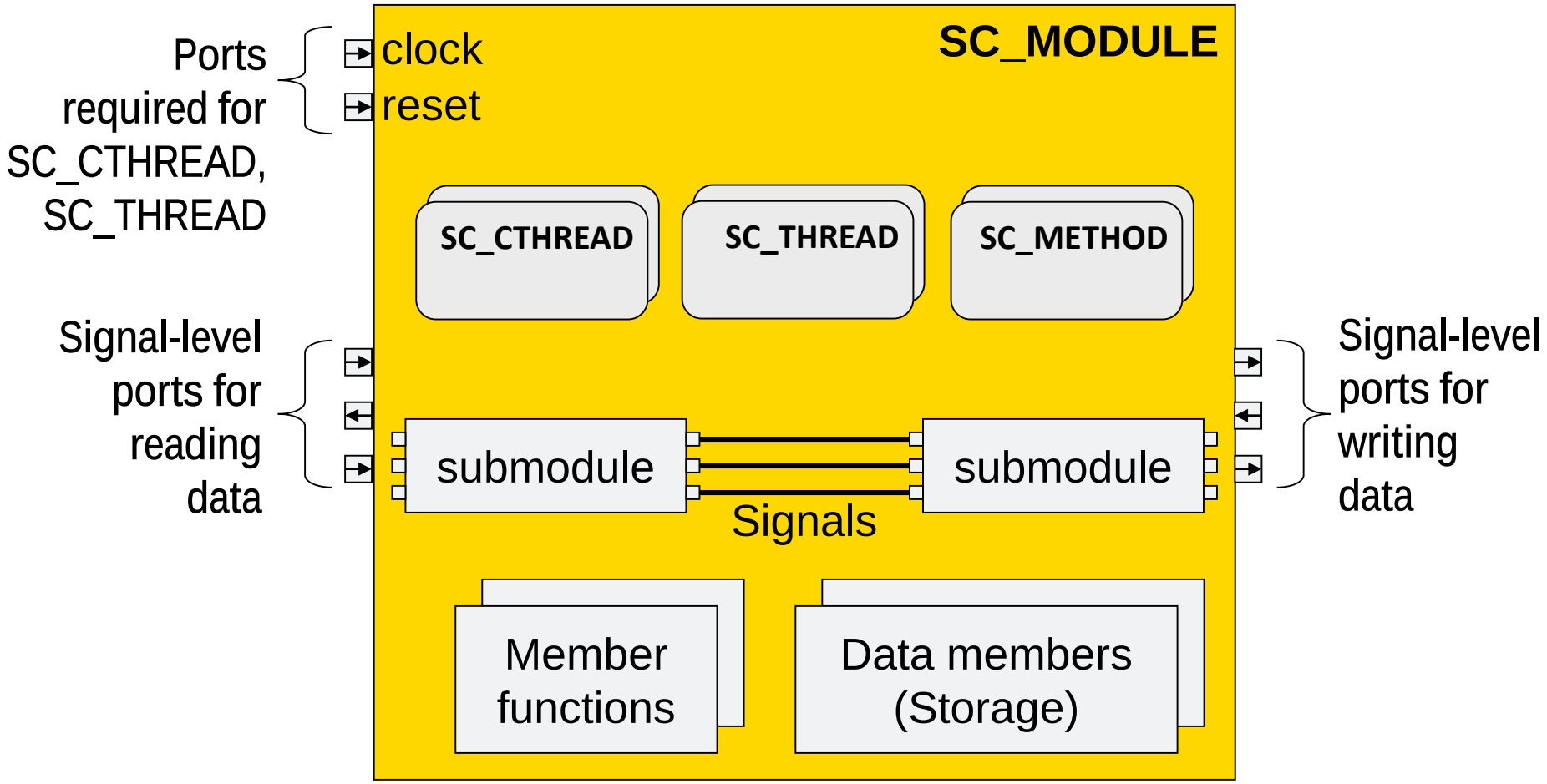
- C++ datatypes
- Expressions
- Functions
- Statements
- Namespaces
- Classes
- Overloading
- Templates

Behavioral Synthesis in the Design Flow

- Design and testbench converted to SystemC modules or threads
- Design
  - Insertion of signal-level interfaces
  - Insertion of reset behavior
  - Conversion to SC_CTHREADs
- Testbench
  - Insertion of signal-level interfaces
  - Reused at each abstraction level
    - Behavioral
    - RTL
    - Gate



Module Structure for Synthesis



Module Declaration

- Module definition
  - SC_MODULE macro
    or
  - Derived from sc_module
    - class or struct
  - SC_CTOR
    or
  - SC_HAS_PROCESS

```
// A module declaration
SC_MODULE( my_module1 ) {
  sc_in< bool> X, Y, Cin;
  sc_out< bool > Cout, Sum;
  SC_CTOR( my_module1 ) {…}
};

// A module declaration
SC_MODULE( my_module1 ) {
  sc_in< bool> X, Y, Cin;
  sc_out< bool > Cout, Sum;
   SC_HAS_PROCESS( my_module1 );
  my_module1(const sc_module_name
    name )
    : sc_module(name)
 {…}
};
```

Derived Modules

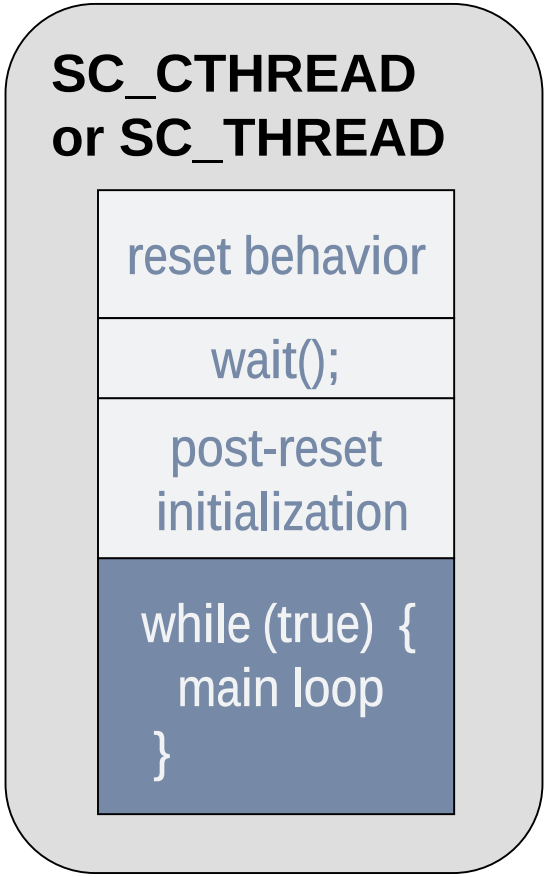
- Derived modules OK

```
SC_MODULE( BaseModule ) {
  sc_in< bool > reset;
  sc_in_clk clock;
  BaseModule ( const sc_module_name name )
    : sc_module( name_ ) {
  }
};

class DerivedModule : public BaseModule {
  void newProcess();
  SC_HAS_PROCESS( DerivedModule );
  DerivedModule( sc_module_name name_ )
    : BaseModule( name_ ) {
    SC_CTHREAD( newProcess, clock.pos() );
    reset_signal_is( reset, true );
  }
};
```

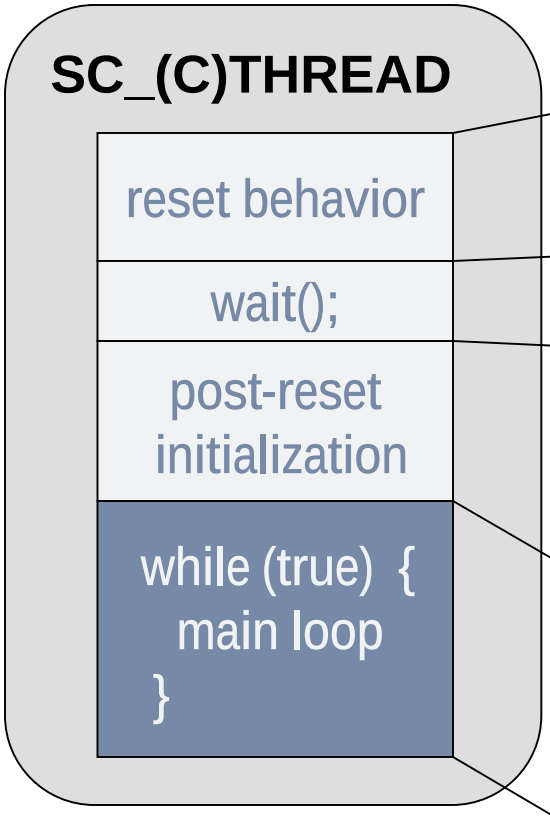
SC_THREAD & SC_CTHREAD Reset Semantics

- At start_of_simulation each SC_THREAD and SC_CTHREAD function is called
  - It runs until it hits a wait()
- When an SC_THREAD or SC_CTHREAD is restarted after any wait()
  - If reset condition is false
    - execution continues
  - If reset condition is true
    - stack is torn down and function is called again from the beginning
- This means
  - Everything before the first wait will be executed while reset is asserted



Note that every path through main loop must contain a wait() or simulation hangs with an infinite loop

SC_THREAD & SC_CTHREAD Process Structure




```
void process() {
 // reset behavior must be
 // executable in a single cycle
 reset_behavior();

 wait();

 // initialization may contain
 // any number of wait()s.
 // This part is only executed
 // once after a reset.
 initialization();

 // infinite loop
 while (true) {
   rest_of_behavior();
 }
}
```

Process Structure Options

- SC_THREAD and SC_CTHREAD processes must follow one of the forms shown
- Note that there must be a wait() in every path of the infinite loops to avoid simulator hangup

```
while( 1 )
{ }

while( true )
{ }

do { }
while ( 1 );

do { }
while ( true );

for ( ; ; )
{ }
```

Specifying Clock and Reset

For synthesis, SC_THREAD can only have a single sensitivity to a clock edge

```
Simple signal/port and level
  SC_CTHREAD( func, clock.pos() );
reset_signal_is( reset, true );
  areset_signal_is( areset, true );

  SC_THREAD( func );
  sensitive << clk.pos();
  reset_signal_is( reset, true );
  areset_signal_is( areset, true );
```

```
reset_signal_is( const sc_in<bool> &port, bool level )
reset_signal_is( const sc_signal<bool> &signal, bool level )
async_reset_signal_is( const sc_in<bool> &port, bool level )
async_reset_signal_is( const sc_signal<bool> &signal, bool level )
```

Use of wait()

- For synthesis, wait(...) can only reference the clock edge to which the process is sensitive
- For SC_CTHREADs
  - wait()
  - wait(int)
- For SC_THREADs
  - wait()
  - wait(int)
  - wait(clk.posedge_event())
  - wait(clk.negedge_event())

For synthesis of SC_THREADs wait(event) must match the sensitivity of the clock edge

Types and Operators

- C++ types
- sc_int, sc_uint
- sc_bv, sc_lv
- sc_bigint, sc_biguint
- sc_logic
- sc_fixed, sc_ufixed

- All SystemC arithmetic, bitwise, and comparison operators supported
- Note that shift operand should be unsigned to allow minimization of hardware

Supported SystemC integer functions					
bit select []	part select (i,j)	concatenate (,)			
to_int()	to_long()	to_int64()	to_uint()	to_uint64()	to_ulong()
iszero()	sign()	bit()	range()	length()	
reverse()	test()	set()	clear()	invert()	

Data Types

- C++ integral types
  - All C++ integral types except wchar_t
  - char is signed (undefined in C++)
- C++ operators
  - a>>b
    Sign bit shifted in if a is signed
  - ++ and -- not supported for bool
- For sc_lv
  - “X” is not supported
  - “Z” is not supported

Pointers

- Supported for synthesis
  - "this" pointer
  - "Pointers that are statically determinable are supported. Otherwise, they are not supported."
  - If a pointer points to an array, the size of the array must also be statically determinable.
- Not supported
  - Pointer arithmetic
  - Testing that a pointer is zero
  - The use of the pointer value as data
    - e.g., hashing on a pointer is not supported for synthesis

Other C++ Constructs

- Supported
  - Templates
  - const
  - volatile
  - namespace
  - enum
  - class and struct
    - private, protected, public
  - Arrays
  - Overloaded operators
- Not supported
  - sizeof()
  - new()
    - Except for instantiating modules
  - delete()
  - typeid()
  - extern
  - asm
  - Non-const global variables
  - Non-const static data members
  - unions

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Thank You!