EE382V: Embedded System Design and Modeling

Lecture 9 – The SystemC Language

Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

Lecture 9: Outline

- ✓ **SystemC tutorial**
  - ✓ Xtreme-EDA
- **SystemC context and comparison to SpecC**
  - Foundation and features
  - Models and methodology
  - Example
- **SystemC TLM 2.0 standard**
  - Coding styles and abstraction levels
  - Advanced techniques

Foundation

- **SpecC language**
  - ANSI C
  - New keywords
  - SpecC compiler (scc)
- **SpecC simulator**
  - Discrete event C++ simulation kernel
- **SpecC standardization**
  - SpecC Technology Open Consortium (STOC)
  - Open source reference compiler and simulator
- **SystemC "language"**
  - C++
  - Class library
  - Standard C++ tools (g++)
- **SystemC simulator**
  - Discrete event C++ simulation kernel
- **SystemC standardization**
  - Open SystemC Initiative (OSCI)
  - Open-source library and simulation kernel
  - IEEE Standard

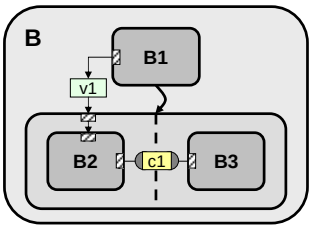
Core Language: Data Types

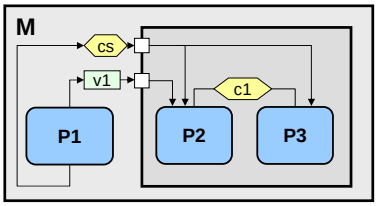
- **SpecC data types**
  - C types & boolean
  - Bit vectors
  - 4-value logic vectors
  - Events
  - Signals*
- **SystemC data types**
  - C++ types
  - Bit vectors
  - 4-value logic vectors
  - Events*
  - Signal channel*
  - Fixed-point
  - Variable-length integers

** SpecC 2.0* ** SystemC 2.0*

Core Language: Hierarchy

- **SpecC structural hierarchy**
  - Behaviors
  - Ports and variables
  - Channels and interfaces
- **SpecC behavioral hierarchy**
  - seq, fsm
  - par, pipe
  - try-trap, -interrupt
- **SystemC structural hierarchy**
  - Modules
  - Ports and variables
  - Channels* and interfaces*
- **SystemC behavioral hierarchy**
  - Parallel leaf processes
    - METHOD (combinatorial)
    - THREAD (behavior)





** SystemC 2.0*

EE382V: Embedded Sys Dsgn and Modeling, Lecture 9 © 2010 A. Gerstlauer 5

Channel Library

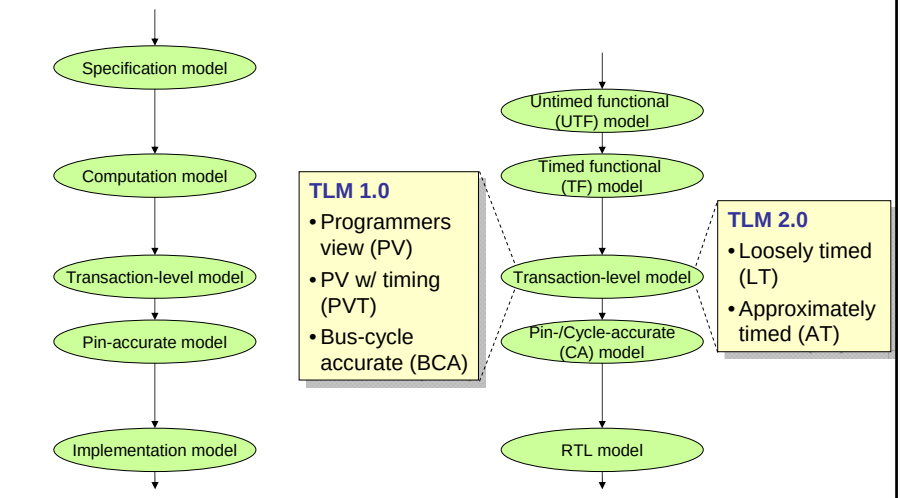
- **SpecC channels***
  - c_queue / c_typed_queue
  - c_double_handshake / c_typed_double_handshake
  - c_handshake
  - c_mutex
  - c_semaphore
  - c_token
  - c_barrier
  - c_critical_section
- **SystemC channels***
  - sc_fifo<T>
  - sc_event_queue
  - sc_mutex
  - sc_semaphore
  - sc_signal<T>
  - sc_buffer<T>
  - sc_clock

** SpecC 2.0*

** SystemC 2.0*

EE382V: Embedded Sys Dsgn and Modeling, Lecture 9 © 2010 A. Gerstlauer 6

Methodology



Lecture 9: Outline

- ✓ SystemC tutorial
  - ✓ Xtreme-EDA
- • SystemC context and comparison to SpecC
  - ✓ Foundation and features
  - ✓ Models and methodology
  - • Example
- • **SystemC TLM 2.0 standard**
  - • Coding styles and abstraction levels
  - • Advanced techniques

FIFO Example: Channel

```
class write_if :
        virtual public sc_interface
{
   public:
     virtual void write(char) = 0;
     virtual void reset() = 0;
};

class read_if :
         virtual public sc_interface
{
   public:
     virtual void read(char &) = 0;
     virtual int num_available() = 0;
};
```

```
SC_MODULE(fifo),
            public write_if, public read_if
{
   public:
     SC_CTOR(fifo):
                        num_elements(0), first(0) {}

     void write(char c) {
       if (num_elements == max)
         wait(read_event);

       data[(first + num_elements++) % max] = c;
       write_event.notify();
     }

     void read(char &c){
       if (num_elements == 0)
         wait(write_event);

       c = data[first]; --num_elements;
       first = (first + 1) % max;
       read_event.notify();
     }

     void reset() { num_elements = first = 0; }
     int num_available() { return num_elements; }

   private:
     enum e { max = 10 };
     char data[max];
     int num_elements, first;
     sc_event write_event, read_event;
};
```

EE382V: Embedded Sys Dsgn and Modeling, Le

Source: S. Swan, Cadence Design Systems

FIFO Example: Behaviors

```
SC_MODULE(producer)
{
   public:
     sc_port<write_if> out;

     SC_CTOR(producer)
     {
       SC_THREAD(main);
     }

     void main()
     {
       char c;

       while (true) {
         …
         out->write(c);
         if (…) out->reset();
       }
     }
};
```

```
SC_MODULE(consumer)
{
   public:
     sc_port<read_if> in;

     SC_CTOR(consumer)
     {
       SC_THREAD(main);
     }

     void main()
     {
       char c;

       while (true) {
         in->read(c);
         if (in->num_available())
           …
       }
     }
};
```

Source: S. Swan, Cadence Design Systems

FIFO Example: Main

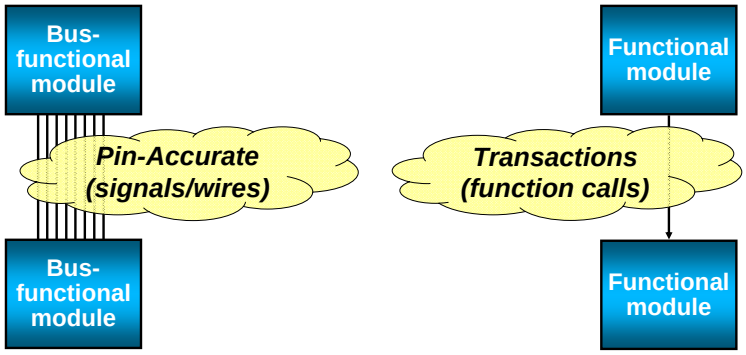
```
SC_MODULE(top),
{
   public:
     fifo *fifo_inst;
     producer *prod_inst;
     consumer *cons_inst;

     SC_CTOR(top)
     {
       fifo_inst = new fifo("Fifo1");

       prod_inst = new producer("Producer1");
       prod_inst->out(*fifo_inst);

       cons_inst = new consumer("Consumer1");
       cons_inst->in(*fifo_inst);
     }
};

int sc_main (int argc , char *argv[])
{
   top top1("Top1");
   sc_start();
   return 0;
}
```

Source: S. Swan, Cadence Design Systems

Lecture 9: Outline

- ✓ SystemC tutorial
  - ✓ Xtreme-EDA
- ✓ SystemC context and comparison to SpecC
  - ✓ Foundation and features
  - ✓ Models and methodology
  - ✓ Example
- **SystemC TLM 2.0 standard**
  - Coding styles and abstraction levels
  - Advanced techniques

Transaction Level Modeling

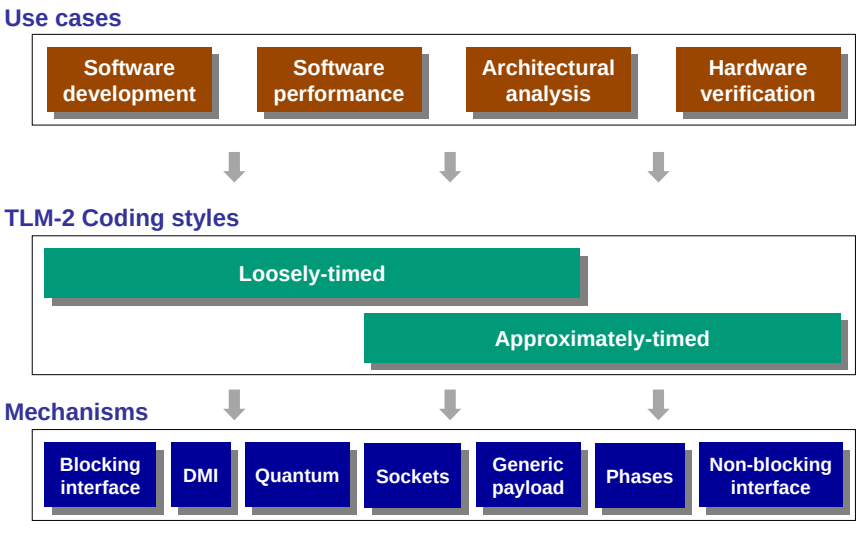


- **Pin-accurate model (PAM)**
  - Simulate every event (protocols)
- **Transaction-level model (TLM)**
  - Communications by transactions (abstract channels)
  - Simulates 100x - 10,000x faster than RTL

Source: OSCI TLM-2.0

EE382V: Embedded Sys Dsgn and Modeling, Lecture 9 © 2010 A. Gerstlauer 13

SystemC/TLM 2.0

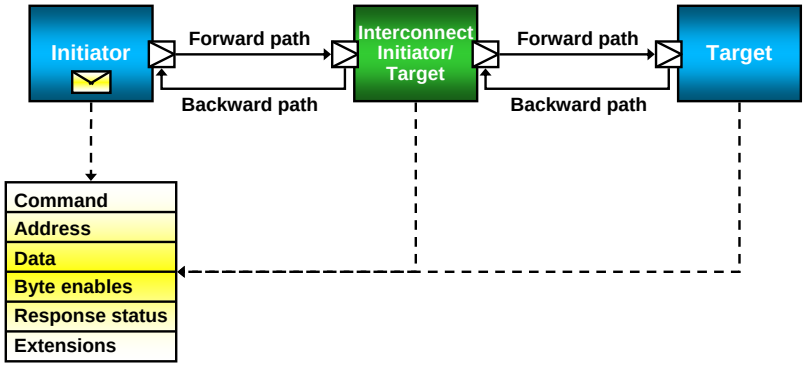


Source: OSCI TLM-2.0

EE382V: Embedded Sys Dsgn and Modeling, Lecture 9 © 2010 A. Gerstlauer 14

Initiators and Targets

- **Pointer to transaction object is passed from module to module using forward and backward paths**
- **Transactions are of generic payload type**

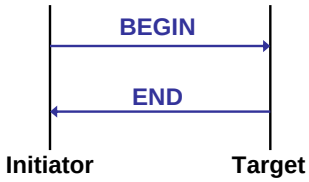


Source: OSCI TLM-2.0

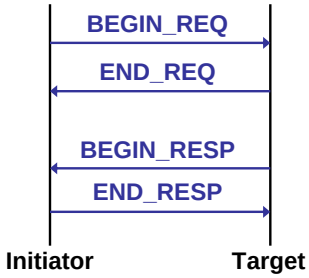
EE382V: Embedded Sys Dsgn and Modeling, Lecture 9 © 2010 A. Gerstlauer 15

Coding Styles

- **Loosely-timed**
  - Sufficient timing detail to boot OS and simulate multi-core systems
  - Each transaction has 2 timing points: *begin* (call) and *end* (return)



- **Approximately-timed**
  - Cycle-approximate or cycle-count-accurate
  - Sufficient for architectural exploration
  - Each transaction has at least 4 timing points



Source: OSCI TLM-2.0

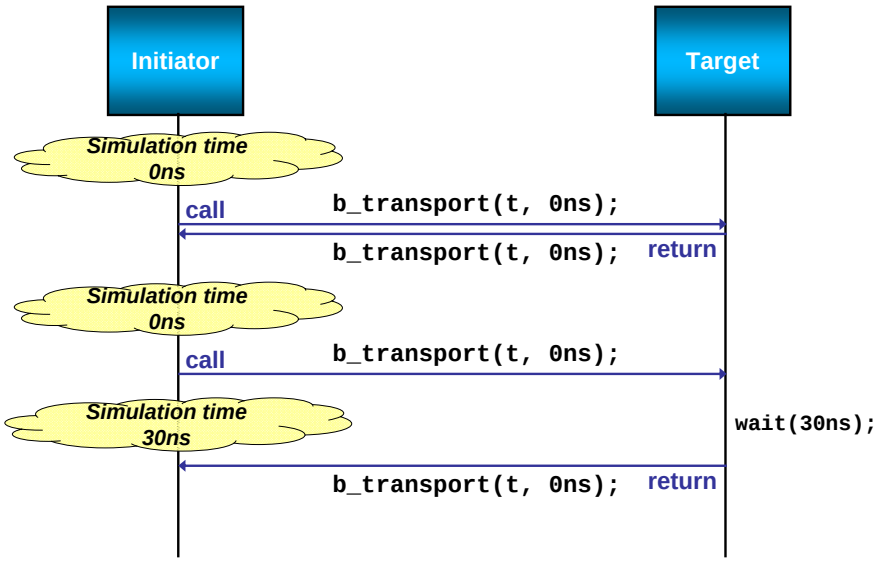
EE382V: Embedded Sys Dsgn and Modeling, Lecture 9 © 2010 A. Gerstlauer 16

Blocking and Non-Blocking Transports

- **Blocking transport interface**
  - Typically used with loosely-timed coding style
  - tlm_blocking_transport_if
    ```
    void b_transport(TRANS&, sc_time&);
    ```
- **Non-blocking transport interface**
  - Typically used with approximately-timed coding style
  - Includes transaction phases
  - tlm_fw_nonblocking_transport_if
    ```
    tlm_sync_enum nb_transport_fw(TRANS&, PHASE&, sc_time&);
    ```
  - tlm_bw_nonblocking_transport_if
    ```
    tlm_sync_enum nb_transport_bw(TRANS&, PHASE&, sc_time&);
    ```

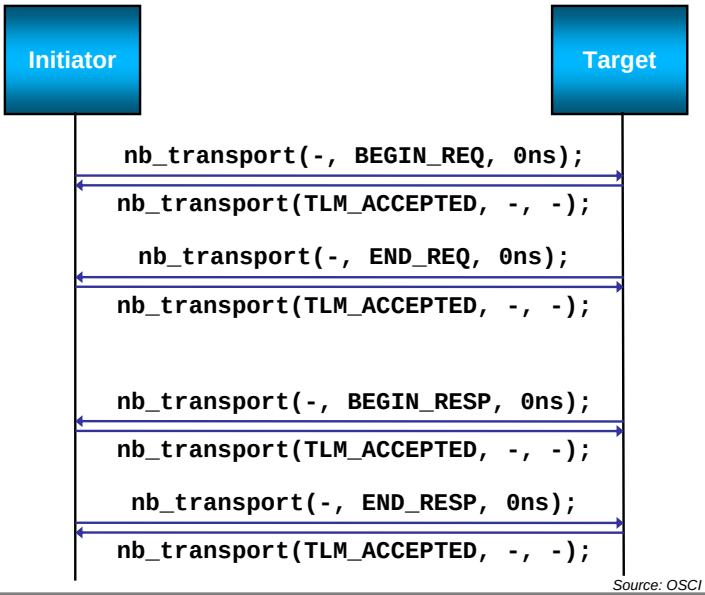
Source: OSCI TLM-2.0

Blocking Transport



Source: OSCI TLM-2.0

Non-Blocking Transport



Source: OSCI TLM-2.0

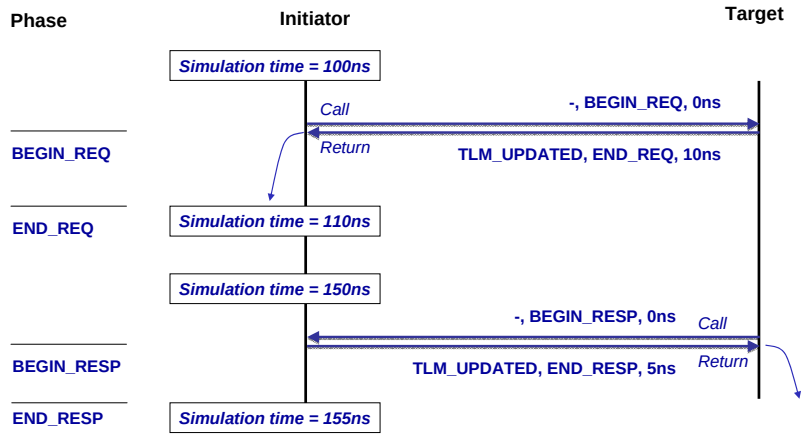
Return Values (tlm_sync_enum)

- **TLM_ACCEPTED**
  - Transaction, phase and timing arguments unmodified (ignored) on return
  - Target may respond later (depending on protocol)
- **TLM_UPDATED**
  - Transaction, phase and timing arguments updated (used) on return
  - Target has advanced the protocol state machine to the next state
- **TLM_COMPLETED**
  - Transaction, phase and timing arguments updated (used) on return
  - Target has advanced the protocol state machine straight to the final phase

Source: OSCI TLM-2.0

Using the Return Path

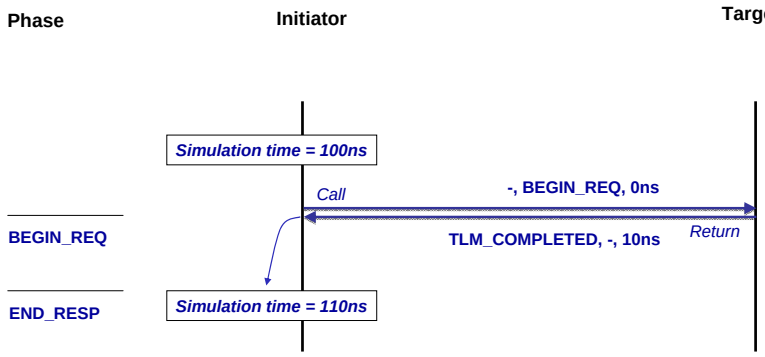
- **Callee annotates delay to next transaction**
  - Caller waits



Source: OSCI TLM-2.0

Early Completion

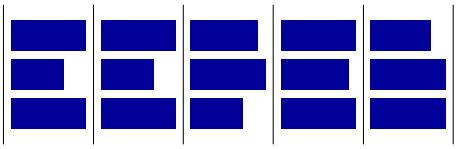
- **Callee annotates delay to next transaction**
  - Caller waits



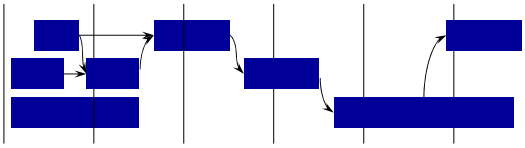
Source: OSCI TLM-2.0

Temporal Decoupling

- **Loose coupling**
  - OS and driver SW development
  - Local process time
  - Every process runs ahead until data is missing or a time quantum boundary was reached (local/global time synchronization)

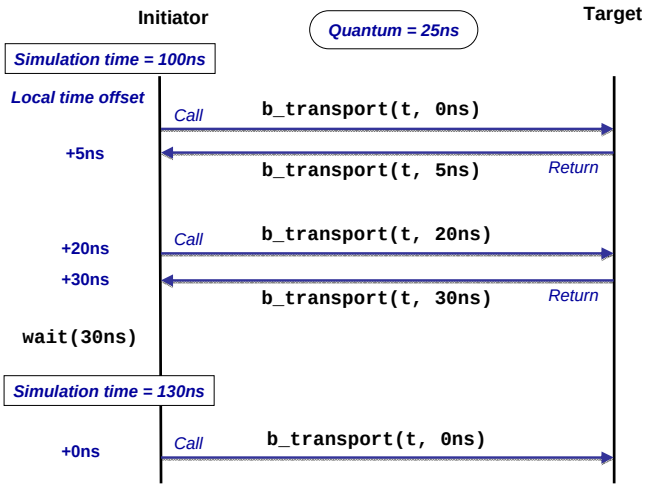


- **Approximate coupling**
  - Architecture trade-off
  - Each process has the global SystemC time, processes synchronize
  - Time may be accurate or estimated



Source: W. Ecker, Infineon

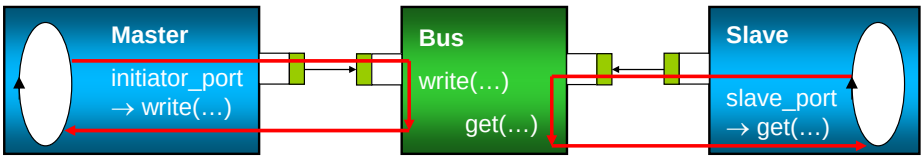
Time Quantum



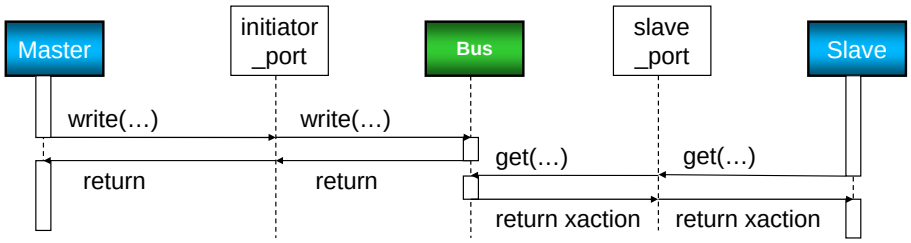
Source: OSCI TLM-2.0

Active Slaves

Slaves have own thread, actively get bus transactions

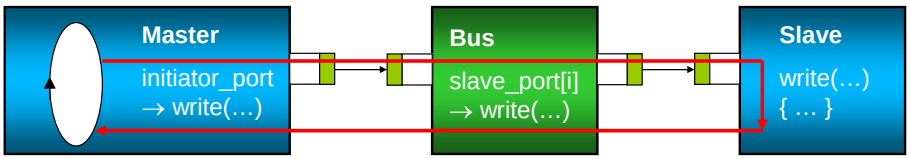


Blocking or non-blocking transport mechanism
May model an active, non-blocking bus with transaction delays

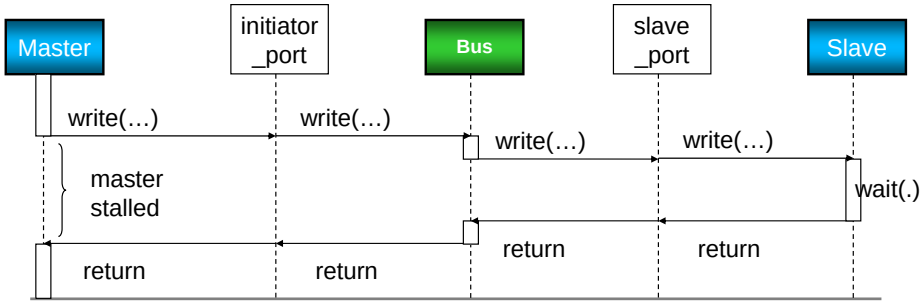


Passive Slaves

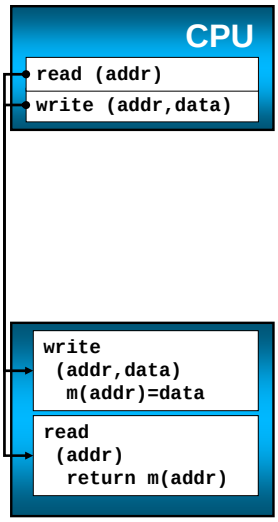
Slave methods executed by the master's thread.



The master is blocked while bus / slave uses its thread.
No context switch overhead



Direct Interface



- **Implementation approach**
  - Direct member function call of target object
    - In master context
  - Pure virtual interface classes defined interface
  - Function call encapsulates and hides all interconnect details
  - Port/export provides connection semantics (incl. restrictions)
- ➢ **Direct memory interface**
  - ➢ Faster (no context switch)
- ➢ **Debug interface**

Source: W. Ecker, Infineon

Lecture 9: Summary

- **The SystemC system-level design language (SLDL)**
  - *Don't invent a new language!* Build on C/C++ so that:
    - Extensive C/C++ infrastructure (compilers, debuggers, language standards, books, etc.) can be re-used.
    - Users' existing knowledge of C/C++ can be leveraged
    - Integration with existing C/C++ code is easy
  - Best-in-class performance
  - General set of modeling constructs to cleanly support the wide range of abstraction levels and models of computation used in system design.
  - Specification *and* refinement to detailed implementation of both software and hardware.
  - Verification through all stages of the design process
- ➢ **The "*de facto*" industry standard language providing both system level and HDL modeling capabilities.**
  - ➢ Controlled by a board & steering group: www.systemc.org

Source: S. Swan, Cadence Design Systems