An Introduction to System Design with SystemC

November 2002

Stuart Swan, stuart@cadence.com

Why do we need a new design language?!

- Verilog and VHDL work very well for HW implementation flows, *but*…
- Systems are becoming more complex, pushing us to design and verify at higher levels of abstraction
  - Enables early exploration of system level design tradeoffs
  - Enables early verification of the entire system, reducing risk
  - Enables much higher speed verification
- Software is playing an increasing role in system design
  - Essential that new design flows support early software development, integration with existing C/C++ code, and HW/SW co-design
- New system level flows require a design language that supports *system level* IP delivery and integration

What are the requirements for the new language?

- *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
- The language must have best-in-class performance.
- It must provide a very general set of modeling constructs to cleanly support the wide range of abstraction levels and models of computation used in system design.
- It must support specification *and* refinement to detailed implementation of both software and hardware.
- It must support verification through all stages of the design process

So what is SystemC?

- The "*de facto*" industry standard language (implemented as a C++ class library) providing both system level and HDL modeling capabilities.
- SystemC is controlled by a board & steering group that includes 10 companies: ARM, Cadence, CoWare, Fujitsu, Mentor, Motorola, NEC, Sony, ST, Synopsys,...
- The SystemC reference simulator can be downloaded free from www.systemc.org.
- Many EDA companies are starting to offer tools and libraries that support the SystemC standard.
- Major electronics companies are using SystemC now and are also planning to extend their use.

SystemC History & Organization

- SystemC History
  - Early development done at Synopsys, CoWare/IMEC, UC Irvine
  - Open SystemC Initiative (OSCI) formed in 2000, many new members joined.
  - OSCI LWG developed SystemC 2.0 specification and implementation in 2000-2001, greatly extending the language capabilities
- SystemC Organization Today
  - Open SystemC Initiative (OSCI) is incorporated as a California non-profit organization with formal bylaws, a significant budget, etc.
  - Licensing model is Open Community Licensing. (Unlike GNU, it is OK to build proprietary products from SystemC code.)
  - Well-organized public relations efforts for SystemC are on-going (e.g. SystemC Tech Forum at DAC 2002).
  - Website and source code maintenance for reference implementations are located on SourceForge, and key developers from different companies can access/modify code.

Alternative Languages to SystemC

- Raw C/C++
  - In wide use, but it's very difficult to build everything yourself.
  - IP exchange is difficult.
- SpecC
  - Not much activity apparent in organization. No RTL modeling. Is neither C nor C++.
- Superlog, SystemVerilog
  - A wide range of extensions to Verilog; many focused on improving RTL designer productivity, some focused on system design & verification. A large amount of work remains for language definition & standardization. For SW design and verification, it is becoming more like C/C++, but it is still not C/C++.
- Accelera System Level Languages Group
  - Not much apparent progress.
- Possibly others: UML, SDL, Vera, Verisity's 'e', etc.

What does SystemC actually provide?

- SystemC 1.0 provided RTL and behavioral HDL modeling capabilities. HW is modeled using zero-delay semantics for combinational logic. Signals are modeled using 01XZ, “C” data types and complex data types can also be used within signals.
- SystemC 1.0 includes good support for fixed point modeling.
- SystemC 1.1 beta and 1.2 beta provided some limited communication refinement capabilities.
- SystemC 2.0 has more general system level modeling capabilities with *channels, interfaces*, and *events*.
- SystemC 3.0 will focus on software and scheduler modeling (more later).

C/C++ Isn't a Magic Bullet for HW Design

- Switching to C/C++ does not necessarily change design flows. Changing the design language does not by itself mean that designers can change the required modeling levels.
- Finding a good hardware or system architecture is just as difficult in C++ as in other languages.
- Behavioral synthesis tools have not eliminated the need for engineers to carefully design hardware architectures.
- At best, simulation performance in C/C++ will be on par with HDL at equivalent levels of abstraction
- For HW RTL design, HDLs are easier to use, more stable, faster, have more mature tools and libraries, better debuggers, etc.

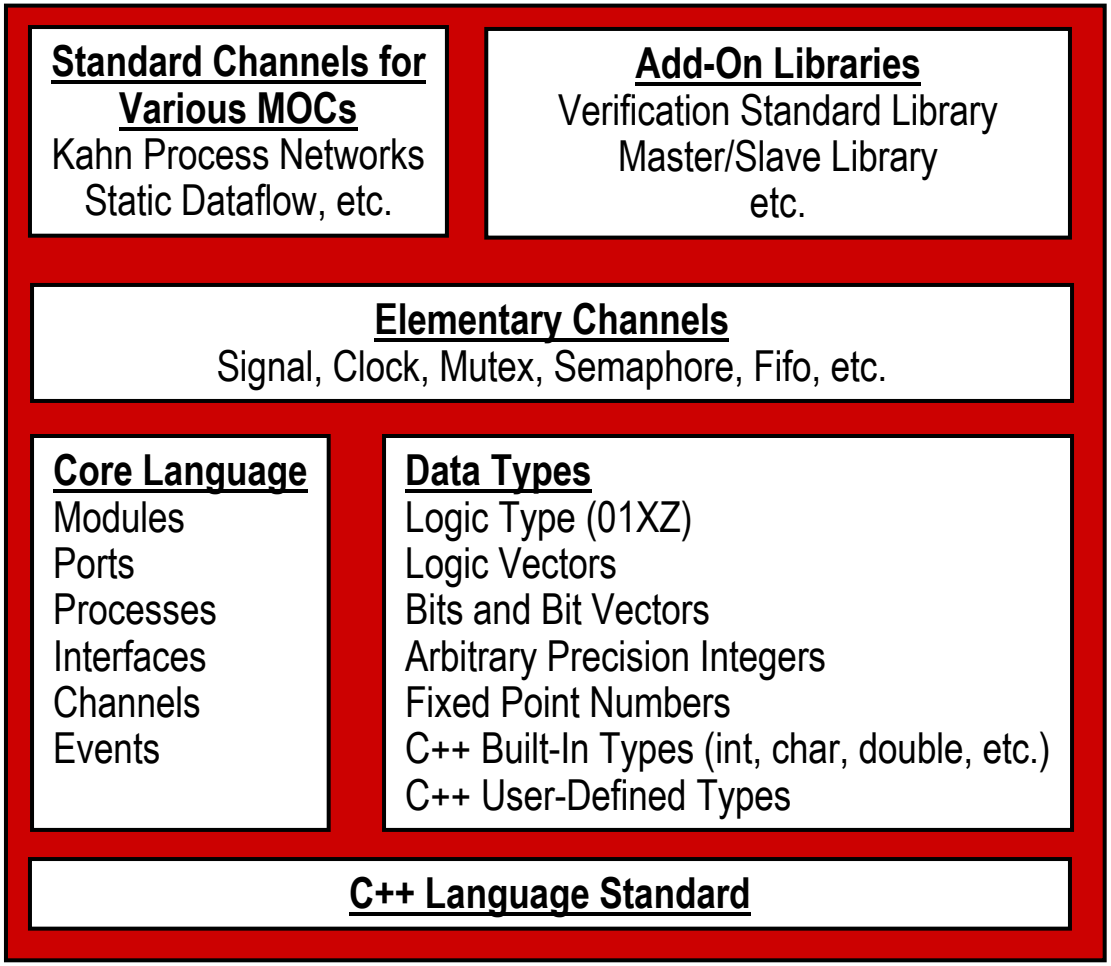
System Level Modeling in SystemC 2.0

- Real potential for SystemC is to be the industry standard language for system level design, verification and IP delivery for both HW and SW.
- Towards this goal, SystemC 2.0 supports generalized modeling for communication and synchronization with *channels, interfaces*, and *events*. Hardware signals are now modeled as a specialization of channels.
- System level extensions in SystemC 2.0 support transaction-level modeling, communication refinement, executable specification modeling, HW/SW co-design.
- Ability to refine HW portions of design to RTL level within a single language is a unique strength of SystemC, as is the fixed point modeling capability, and easy integration of existing C/C++ models.

SystemC 2.0 Language Architecture

Upper layers are built cleanly on lower layers.

Lower layers can be used without upper layers.

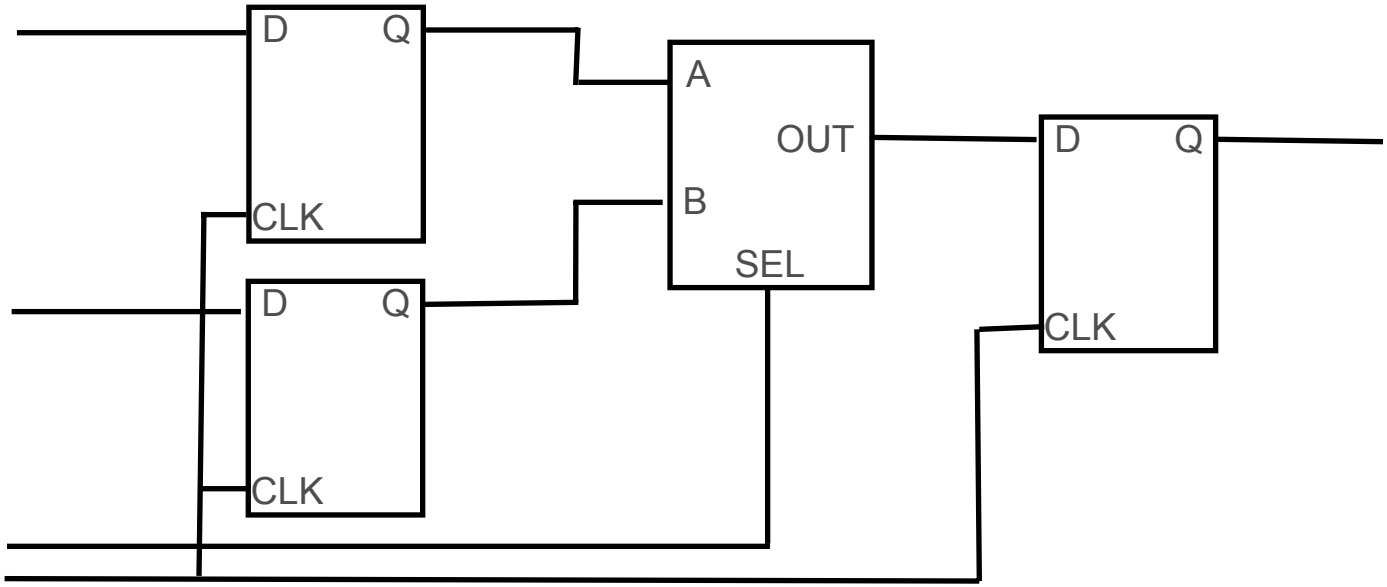


Models of Computation in SystemC 2.0

- A model of computation is broadly defined by:
  - Model of time (real, integer, untimed) and event ordering constraints (globally ordered, partially ordered, etc.)
  - Methods of communication between processes
  - Rules for process activation
- Flexible communication and synchronization capabilities in SystemC 2.0 enable a wide range of MOCs to be naturally modeled.
  - Examples: RTL, Process Networks, Static Dataflow, Transaction Level Models, Discrete Event

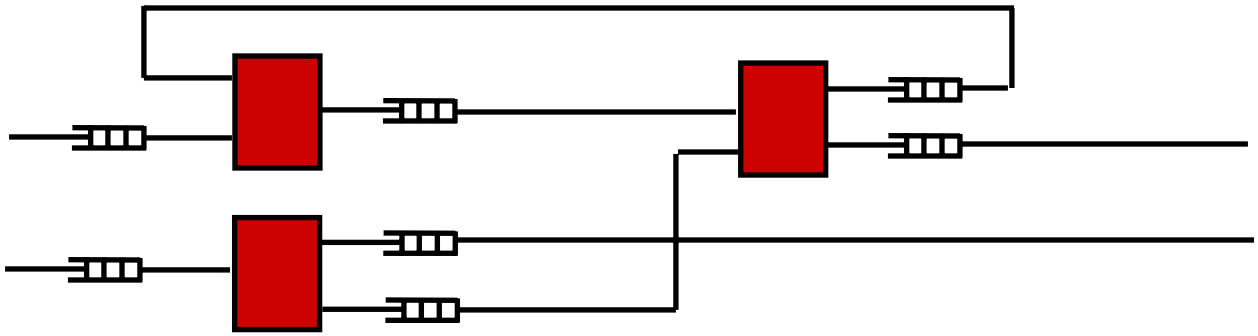
RTL Model of Computation in SystemC

- Models combinational logic and sequential logic triggered by clocks.
- Very similar to RTL modeling in Verilog & VHDL.
- Signals modeled using sc_signal<>, sc_signal_rv<>
- Ports modeled using sc_in<>, sc_out<>, sc_inout<>



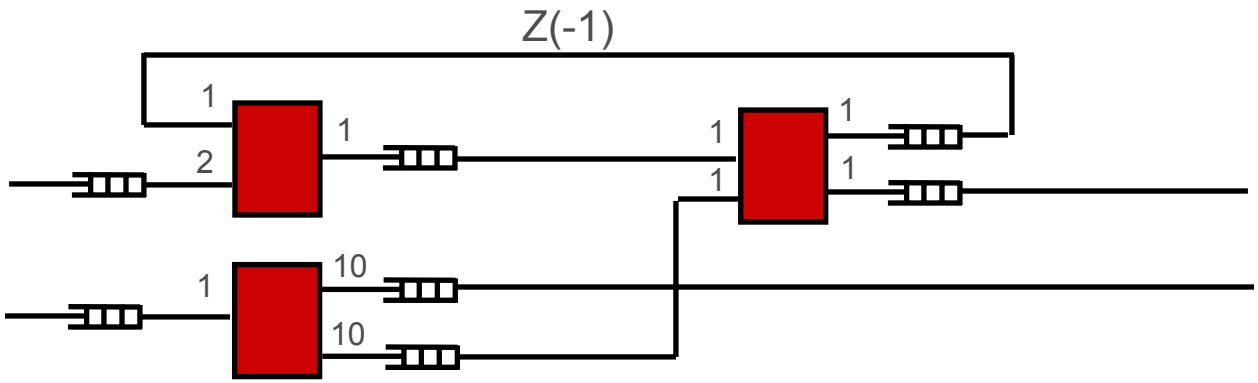
Kahn Process Network MOC in SystemC

- Very useful for high level system modeling
- Modules communicate via FIFOs (sc_fifo<T>) that suspend readers and writers as needed to reliably deliver data items.
- Easy to use and guaranteed to be deterministic
- Pure KPN has no concept of time
- With annotated time delays, becomes *timed functional model* or *performance model*.



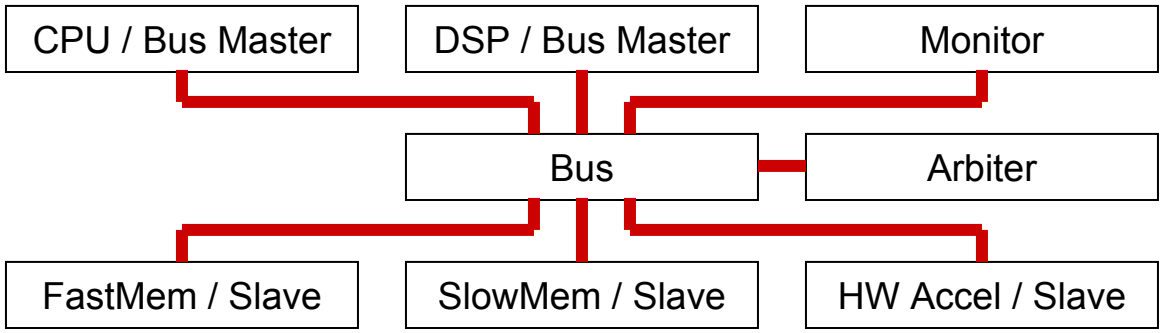
Static Dataflow MOC in SystemC

- A proper subset of the KPN MOC
- Each module reads and writes a fixed number of data items each time it is activated. Sample delays modeled by writing data items into FIFOs before simulation starts.
- Simulators and implementation tools can determine static schedule for system at compile-time, enabling high performance simulation and implementation.
- Commonly used in DSP systems, especially along with SystemC's fixed point types (sc_fixed<>, sc_fix).

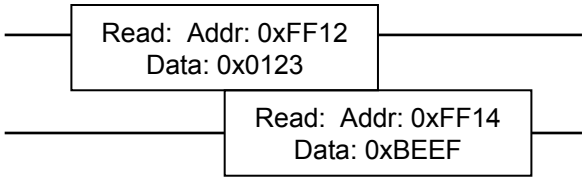


Transaction-Level MOC in SystemC

- Communication & synchronization between modules modeled using function calls (rather than signals)
- Transactions have a start time, end time, and set of data attributes (e.g. burst_read(uint addr, char* data, uint n))
- Two-phase synchronization scheme typically used for overall system synchronization
- Much faster than RTL models (more later…)



Communication between modules is modeled using function calls that represent transactions. No signals are used.

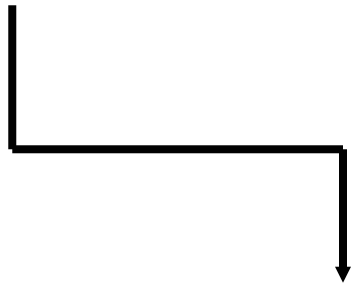


Modeling Example - Interfaces

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

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

Modeling Example - Channel

```
class fifo : public sc_channel, public write_if, public read_if
{
 public:
   fifo() : num_elements(0), first(0) {}

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

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

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

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



```
   void reset() { num_elements = first = 0; }

    int num_available() { return num_elements; }

 private:
   enum e { max_elements = 10 };  // just a constant
   char data[max_elements];
   int num_elements, first;
   sc_event write_event, read_event;
};
```

Modeling Example - Producer / Consumer

```
class producer : public sc_module
{
 public:
  sc_port<write_if> out;   // the producer's output port

  SC_CTOR(producer)     // the module constructor
  {
    SC_THREAD(main);   // start the producer process
  }

  void main()                  // the producer process
  {
      char c;
      while (true) {
        ...
        out->write(c);         // write c into the fifo
        if (...)
          out->reset();        // reset the fifo
      }
  }
};
```

```
class consumer : public sc_module
{
 public:
  sc_port<read_if> in;         // the consumer's input port

  SC_CTOR(consumer)       // the module constructor
  {
    SC_THREAD(main);      // start the consumer process
  }

  void main()                  // the consumer process
  {
      char c;
      while (true) {
        in->read(c);               // read c from the fifo
        if (in->num_available() > 5)
          ...;                     // perhaps speed up processing
      }
  }
};
```

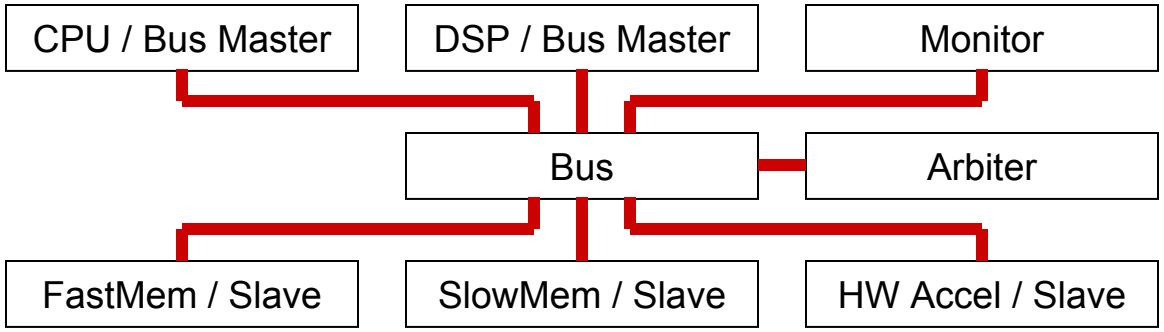
Modeling Example - Top

```
class top : sc_module
{
  public:
    fifo fifo_inst;                  // a fifo instance
    producer *producer_inst;   // a producer instance
    consumer *consumer_inst; // a consumer instance

    SC_CTOR(top)                  // the module constructor
    {
        producer_inst = new producer("Producer1");
        // bind the fifo to the producer's output port
        producer_inst->out(fifo_inst);

        consumer_inst = new consumer("Consumer1");
        // bind the fifo to the consumer's input port
        consumer_inst->in(fifo_inst);
    }
};
```

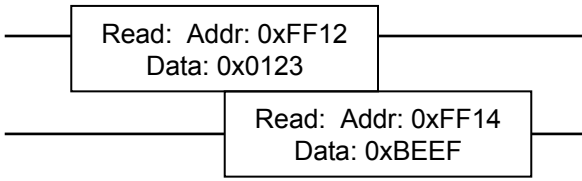
Communication Refinement in SystemC

- Channels may have multiple separate interfaces.
- Ports are bound to a particular interface, not to a channel
- Interfaces can be reused with different channels
- Communication can be refined via channel substitution
- Examples of communication refinement
  - Exploration during functional specification
  - Retargeting abstract communication and synchronization to RTOS API
  - Refining communication to a hardware implementation using *adapters* and hierarchical channels, perhaps followed by “protocol inlining”.

Transaction-Level Modeling in SystemC

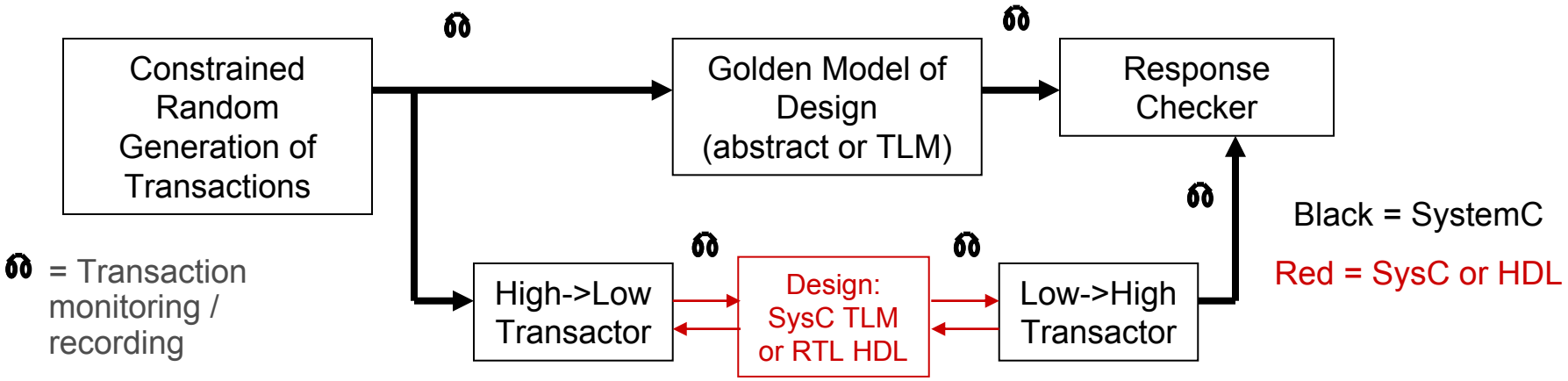


Communication between modules is modeled using function calls that represent transactions. No signals are used.



- Why do transaction-level modeling in SystemC?
  - Models are relatively easy to develop and use
  - HW and SW components of a system can be accurately modeled. Typically bus is cycle-accurate, and bus masters / slaves may or may not be cycle-accurate.
  - Extensive system design exploration and verification can be done early in the design process, before it's too late to make changes
  - Models are fast – typically about 100K clock cycles per second, making it possible to execute significant amounts of the system's software very early in the design process
- Transaction-level modeling is extensively covered in the *System Design with SystemC* book and the code for the *simple_bus* design is provided

Transaction-Based Verification in SystemC



- Why do transaction-based verification in SystemC?
  - Ability to have everything (except perhaps RTL HDL) in SystemC/C++ provides great benefits: easier to learn and understand, easier to debug, higher performance, easy to integrate C/C++ code & models, open source implementation, completely based on industry standards
  - Allows you to develop smart testbenches early in the design process (before developing detailed RTL) to find bugs and issues earlier. Enables testbench reuse throughout the design process.
  - Much more efficient use of verification development effort and verification compute resources
- Transaction-Based Verification in SystemC is described in the *SystemC Verification Standard Specification*, and in the documentation and examples included with the OSCI SCV reference implementation kit.

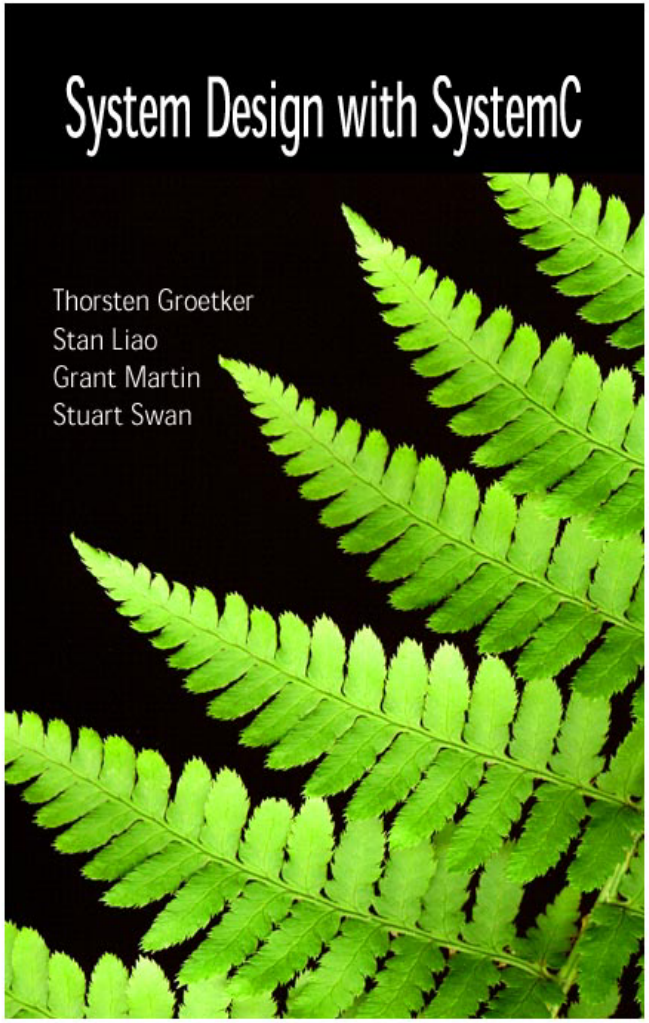
What are Companies Doing Today with SystemC?

- Some companies are using SystemC for RTL modeling, but this is not where the main interest is.
- Many companies are in the process of replacing in-house C/C++ system level modeling environments with SystemC.
- Many companies view SystemC as both a modeling language and a modeling “backplane” (e.g. for ISS integration).
- A number of companies have completed TLM & TBV modeling efforts using SystemC 2.0 and are very enthusiastic. Some of the results are starting to be made publicly available. Some companies are about to announce that they will provide system-level IP using SystemC.
  - For example, see on-line ARM presentation with audio:
    - www.systemc.org/projects/sitedocs/document/ARM1/

What's Happening in the SystemC Committees?

- Language Working Group
  - Working on SystemC 3.0 extensions for software modeling support (e.g., modeling schedulers, abstract RTOS, complex hierarchical FSMs). Code release target Q2/Q3 2003.
- Verification WG
  - Version 1.0 of Verification Standard awaiting final OSCI approval. Open source code should be on OSCI website Nov. or Dec. 2002.
- Mixed Signal WG
  - Considering mixed signal integration
- Synthesis WG
  - Standardize RTL and behavioral "synthesizable subsets" for SystemC
- Probable Future Efforts:
  - IP Integration WG: Develop TLM standards for bus models, various standards for IP packaging and reuse

Learning more about SystemC



- Our new book is available at:
  - www.systemc.org
    - Products & Solutions
    - Books
    - *System Design with SystemC*
- Provides an in-depth discussion of new SystemC features and using SystemC for TLM