Topic VI

System Level Design and Verification

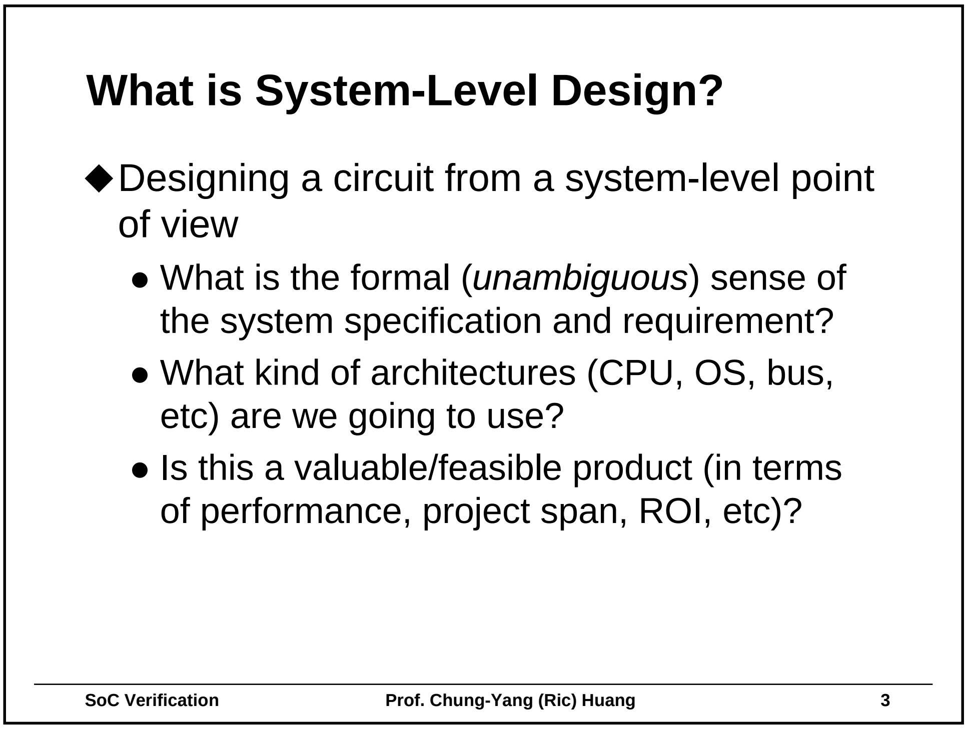
系統晶片驗證

SoC Verification

Sep, 2004

What we will cover in this topic…

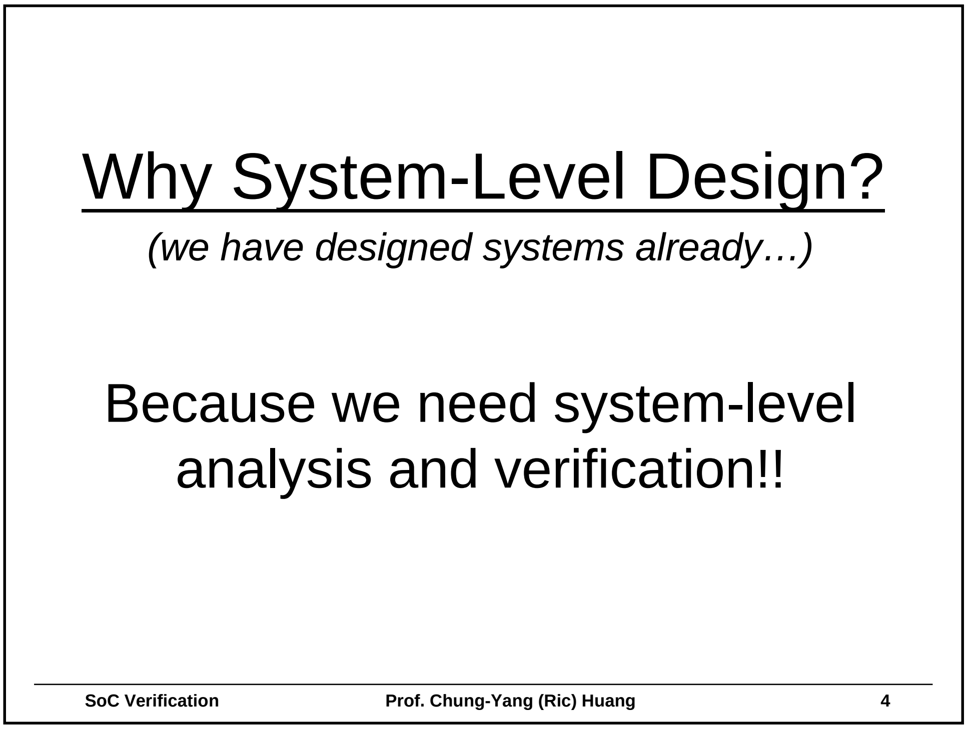
- ◆What is system-level design, and why?
- ◆SystemC basics
- ◆Transaction-Level Modeling (TLM)
- ◆System or unit-level verification
- ◆SystemC verification library
- ◆EDA Tools for System-Level



What is System-Level Design?

- ◆Designing a circuit from a system-level point of view
  - ● What is the formal (*unambiguous*) sense of the system specification and requirement?
  - ● What kind of architectures (CPU, OS, bus, etc) are we going to use?
  - ● Is this a valuable/feasible product (in terms of performance, project span, ROI, etc)?

SoC Verification **Prof. Chung-Yang (Ric) Huang** **3**



Why System-Level Design?

(we have designed systems already…)

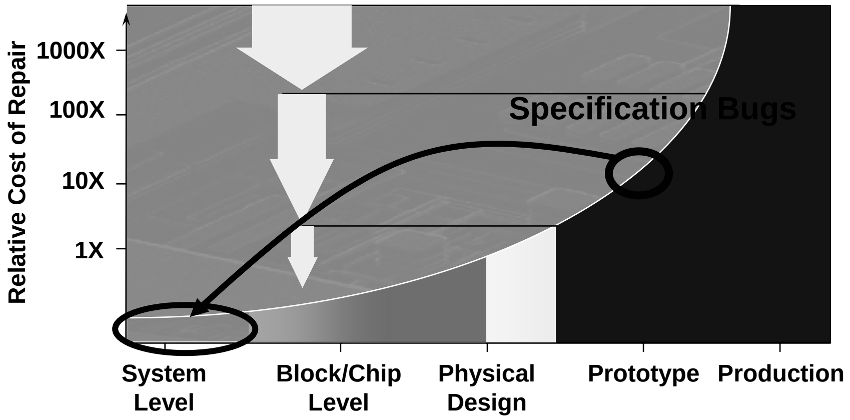
Because we need system-level analysis and verification!!

SoC Verification **Prof. Chung-Yang (Ric) Huang** **4**

Old-Style System Design

- System requirements are translated and partitioned into specs on various design blocks (by system/chip architect)
  - Each HW/SW designer designs on his/her own block(s)
  - System is integrated after all the block designs are done
  - Usually use prototype for integrated system validation
- Why cannot we do better?
  - Deficiency in automatic system design environment
    - HW/SW co-design /co-verification environment
    - Tool supports
    - Design languages
  - Organization problem
    - Chip architect vs. designer vs. verification engineer
- Any problem?
  - Inaccurate (lack of system analysis)
  - Conservative (because of inaccuracy…)
  - Limited system design flexibility

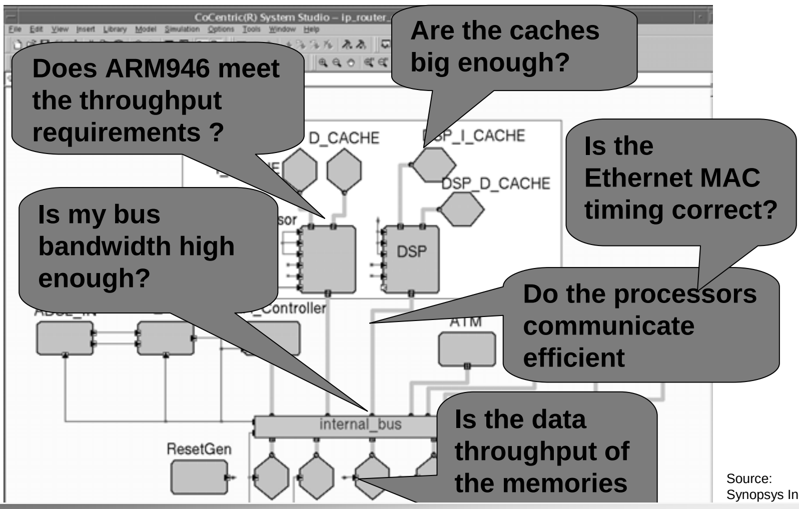
Remember: Escalating Costs to Find Bugs



Finding bugs at system level reduces costs tremendously

Source: Synopsys Inc.

Problems with Prototypes



Does ARM946 meet the throughput requirements ?

Are the caches big enough?

Is the Ethernet MAC timing correct?

Is my bus bandwidth high enough?

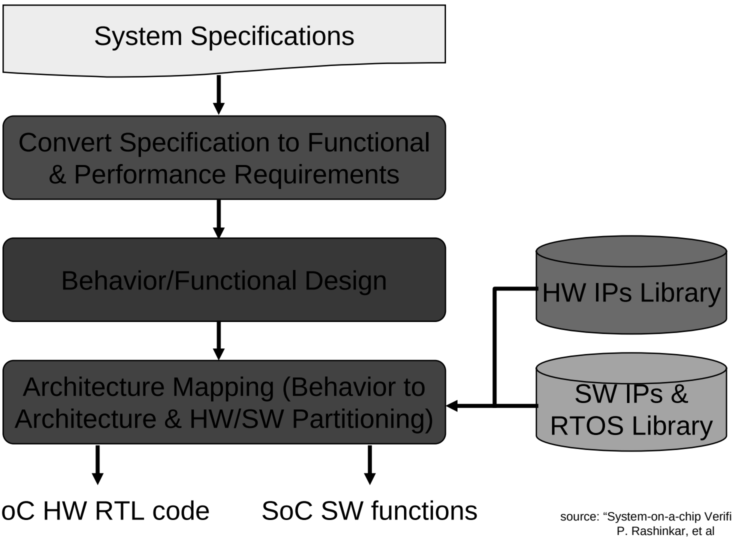
Do the processors communicate efficient

Is the data throughput of the memories

Source: Synopsys Inc.

Fast Answers for these questions is key

System-Level Design Flow



System Specifications

Convert Specification to Functional & Performance Requirements

Behavior/Functional Design

Architecture Mapping (Behavior to Architecture & HW/SW Partitioning)

HW IPs Library

SW IPs & RTOS Library

SoC HW RTL code

SoC SW functions

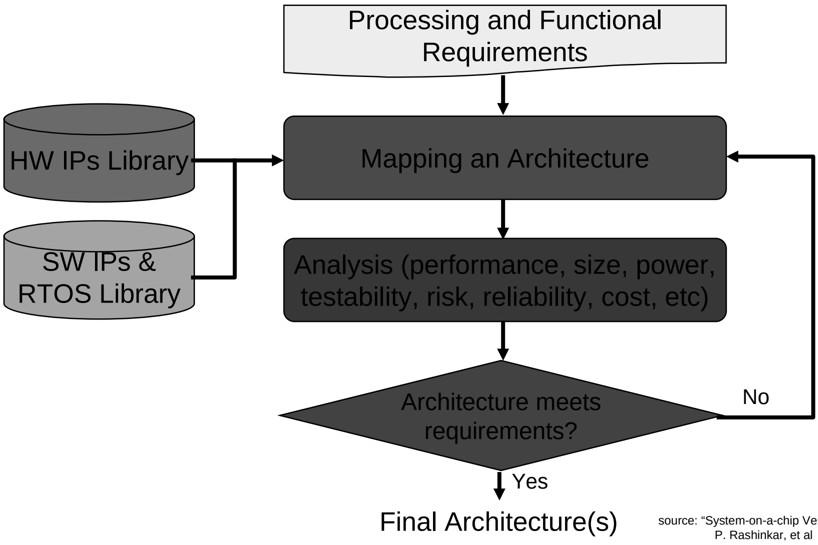
source: “System-on-a-chip Verification”, P. Rashinkar, et al

Behavior and Functional Design

- ◆ Goal
  - ● Be able to formalize the criteria to select the architecture and HW/SW partitioning
- ◆ Algorithms are analyzed to assess the processing/computational, memory, and I/O requirements
  - ● What kinds of architecture are you going to use?
  - ● Can they satisfy the system specifications?

➔ Algorithm and process flows are translated to data and control flows that are independent of architecture

Architecture Mapping and Selection


Processing and Functional Requirements
HW IPs Library
SW IPs & RTOS Library
Mapping an Architecture
Analysis (performance, size, power, testability, risk, reliability, cost, etc)
Architecture meets requirements?
No
Yes
Final Architecture(s)

source: "System-on-a-chip Verification", P. Rashinkar, et al

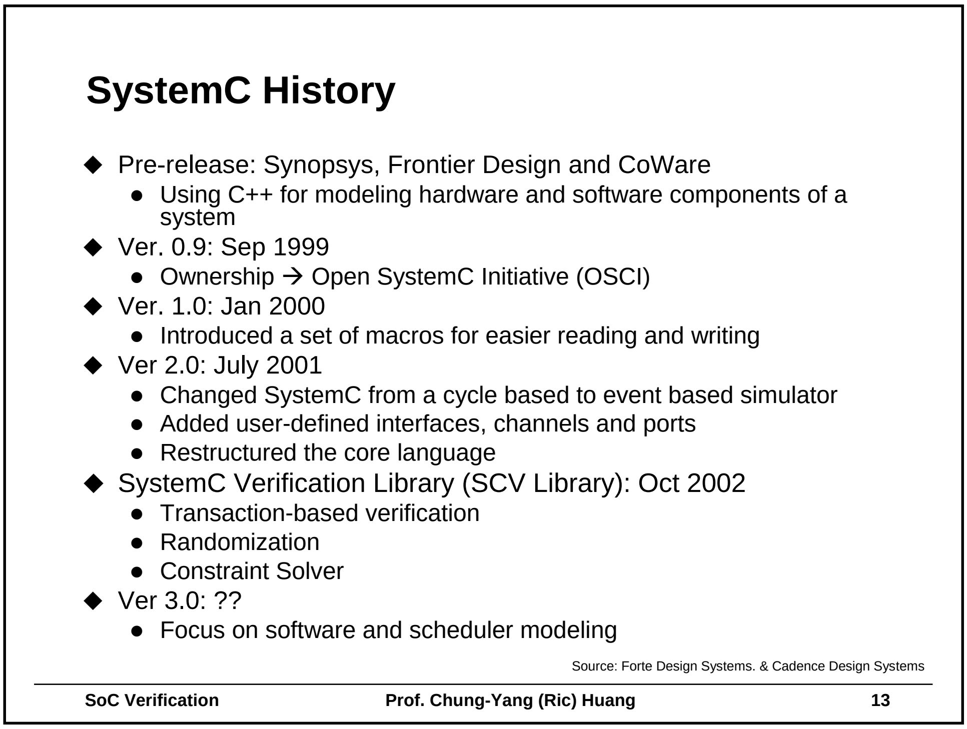
But how?

What kind of languages and tools do we use for system-level design?

SystemC

A System-Level

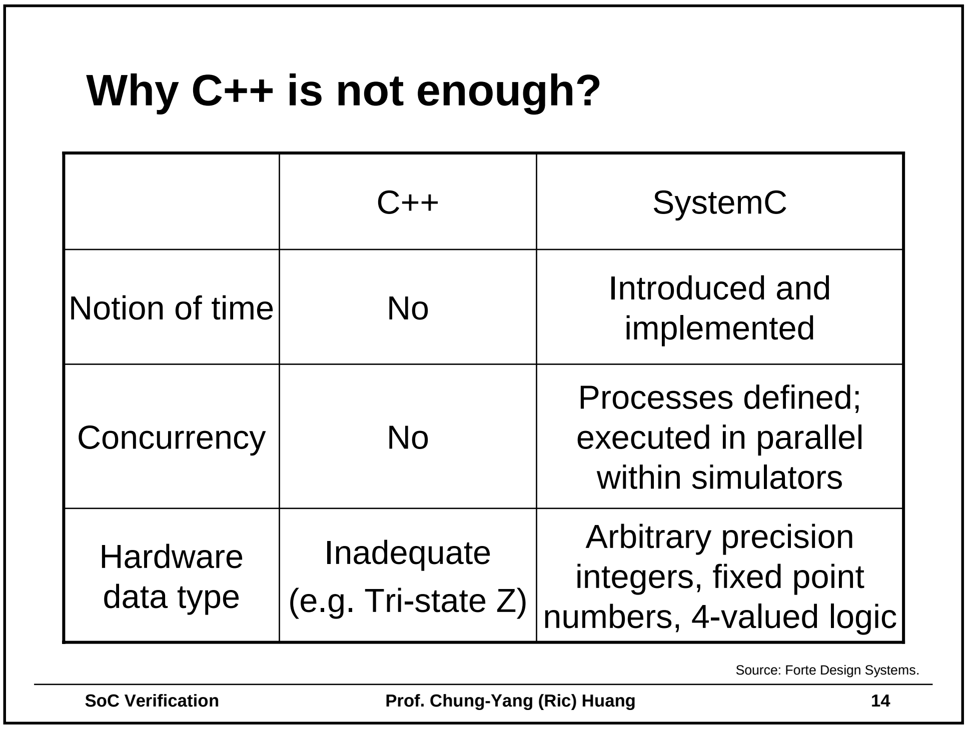
Design and Verification Language



SystemC History

- Pre-release: Synopsys, Frontier Design and CoWare
  - Using C++ for modeling hardware and software components of a system
- Ver. 0.9: Sep 1999
  - Ownership → Open SystemC Initiative (OSCI)
- Ver. 1.0: Jan 2000
  - Introduced a set of macros for easier reading and writing
- Ver 2.0: July 2001
  - Changed SystemC from a cycle based to event based simulator
  - Added user-defined interfaces, channels and ports
  - Restructured the core language
- SystemC Verification Library (SCV Library): Oct 2002
  - Transaction-based verification
  - Randomization
  - Constraint Solver
- Ver 3.0: ??
  - Focus on software and scheduler modeling

Source: Forte Design Systems. & Cadence Design Systems



Why C++ is not enough?

	C++	SystemC
Notion of time	No	Introduced and implemented
Concurrency	No	Processes defined; executed in parallel within simulators
Hardware data type	Inadequate (e.g. Tri-state Z)	Arbitrary precision integers, fixed point numbers, 4-valued logic

Source: Forte Design Systems.

Facts about SystemC

◆A bridge between HW and SW designs

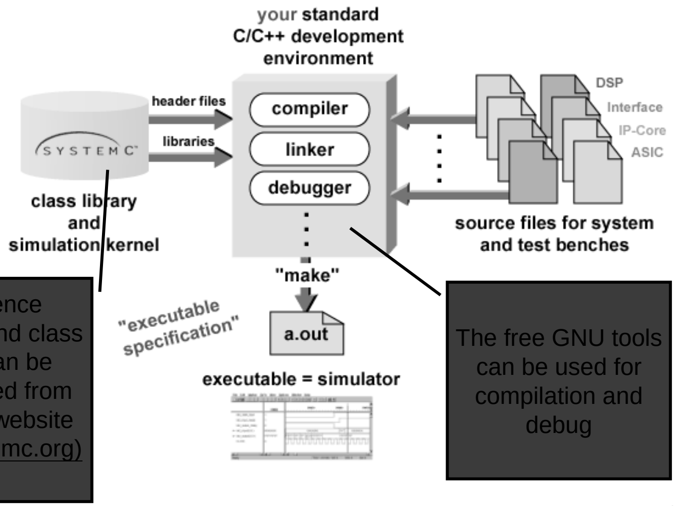
◆Open source C++ library with event-driven simulator

◆“Extends" the C++ language without changing the syntax by using classes

- Can use regular C++ development environment

◆Modules are introduced as a means of encapsulating behavior and describing hierarchy

SystemC Development Environment



A reference simulator and class library can be downloaded from the OSCI website (www.systemc.org).

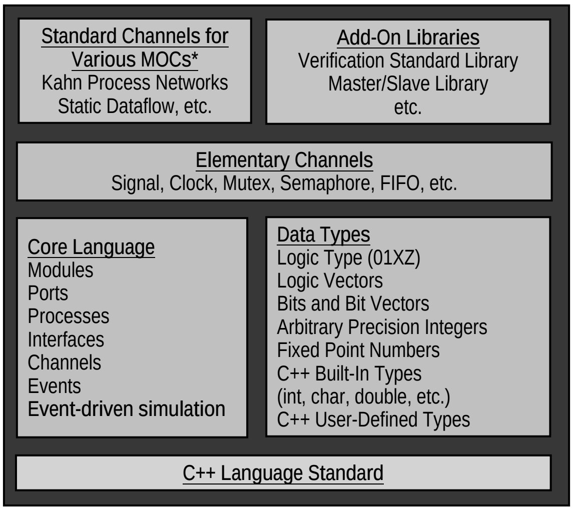
The free GNU tools can be used for compilation and debug

Source: Forte Design Systems.

SystemC 2.0 Language Architecture

Upper layers are built cleanly on lower layers.

Lower layers can be used without upper layers.



Standard Channels for Various MOCs*
Kahn Process Networks
Static Dataflow, etc.

Add-On Libraries
Verification Standard Library
Master/Slave Library
etc.

Elementary Channels
Signal, Clock, Mutex, Semaphore, FIFO, etc.

Core Language
Modules
Ports
Processes
Interfaces
Channels
Events
Event-driven simulation

Data Types
Logic Type (01XZ)
Logic Vectors
Bits and Bit Vectors
Arbitrary Precision Integers
Fixed Point Numbers
C++ Built-In Types
(int, char, double, etc.)
C++ User-Defined Types

C++ Language Standard

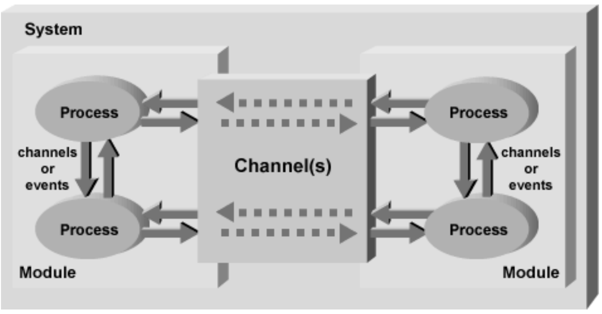
*MOCs: models of computation

Source: Cadence Design Systems

What we will cover for SystemC…

- Modeling structures
  - Module
  - Port
- Communication structures
  - Interface
  - Channel
  - Port
- Events and Processes
- Other items…

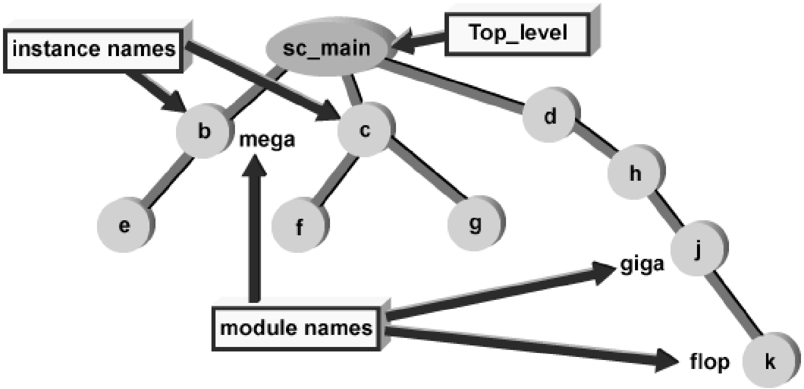
A SystemC System



1. A SystemC system consists of a set of modules
2. Modules contain concurrent processes; they are used to create hierarchy
3. Processes describe functionality and communicate with each other through channels or events
4. Inter-Module communication is through channels

Source: Forte Design Systems.

SystemC Hierarchy

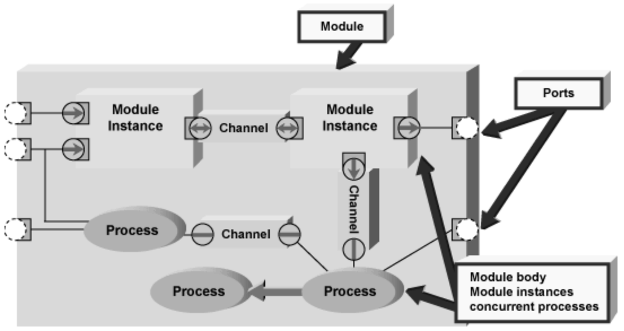


- ◆ Top level: sc_main(); not a module
- ◆ Modules may be instantiated inside of sc_main() as well as inside of other instances of modules (creating hierarchy)

Source: Forte Design Systems.

Basic Modeling Structure

- Module: may represent a system, a block, a board, a chip...etc
- Ports: represent the interface, pins etc., depending upon what the module represents



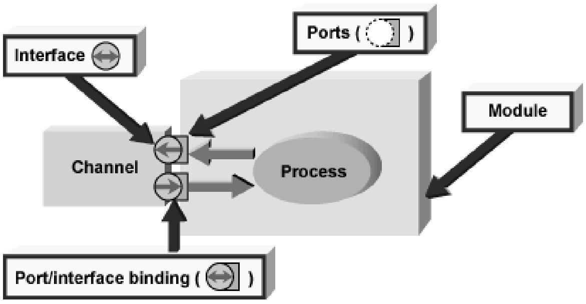
1. Module instances at a peer level are connected through their ports with a channel
2. Parent to child connection of module instances is done port to port
3. Processes communicate through channels or events

Source: Forte Design Systems.

Three Basic Communication Structures

1. Interface
2. Channel
3. Port

Basic Communication Structures (1/3)



Interface

- Provides a set of method declarations, but no implementations and no data fields
  - To define sets of methods that channels must implement.
- Ports are connected to channels through interfaces
  - A port sees only those channel methods that are defined by the interface
  - ➔ Not able to access any other method or data field in the channel
- SystemC 2.0 allows users to define their own interfaces.

Source: Forte Design Systems && OSCI "Functional Spec 2.0"

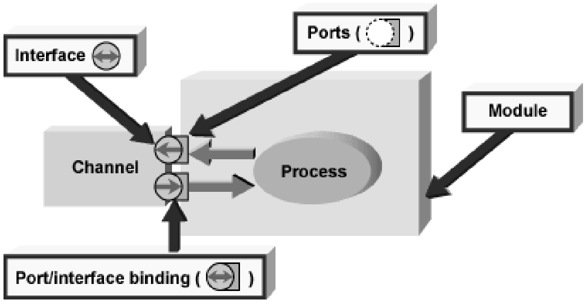
Interface Examples

```
template <class T>
class sc_read_if : virtual public sc_interface
{
  public:
    // interface methods
    virtual const T& read() const = 0;
};
// ------------------------------
template <class T>
class sc_write_if : virtual public sc_interface
{
  public:
    // interface methods
    virtual void write( const T& ) = 0;
};
```

Interface Examples (cont'd)

```
//
template <class T>
class sc_read_write_if
: public sc_read_if<T>,
  public sc_write_if<T>
{};
```

Basic Communication Structures (2/3)



Channel

- Implements one or more interface's methods
- The container for communication functionality
- Not necessarily a point-to-point connection; may be connected to more than two modules.

Source: Forte Design Systems && OSCI "Functional Spec 2.0"

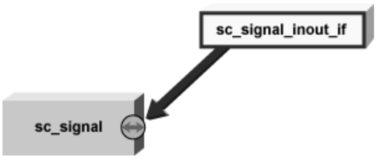
Two Types of Channels

1. Primitive channels
   - No visible structure, they contain no processes and cannot directly access other channels. Examples of primitive channels in SystemC are:
     - sc_signal<T>
     - sc_signal_rv<N>
     - sc_fifo<T>
     - sc_mutex
     - sc_semaphore
     - sc_buffer<T>
2. Hierarchical channels
   - Have structure
   - Similar to modules in they can have processes, ports and instances of other channels or modules
   - Can directly access other channels

Source: Forte Design Systems.

Primitive Channel Example: sc_signal<T>

- Implements the sc_signal_inout_if interface
- Used to describe hardware signals and are typically used in RTL modeling
- Signals are unresolved
  - May only have one writer, but may have multiple readers
  - If more than one process writes to a signal an error occurs



Source: Forte Design Systems.

sc_signal<T>

- *Syntax*

  sc_signal<T> signal_name, signal_name, ... ;

- *Example of syntax:*

```
SC_MODULE (module_name) {
     sc_signal<int> d ;
     sc_signal<char> e ;
     sc_signal<sc_int<10> > f;
     // rest of module not shown
} ;
```

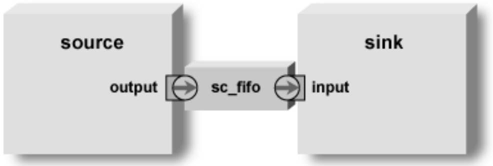
Source: Forte Design Systems.

Example use of methods of sc_signal<T>

```
// declarations
sc_signal<int> sig_a;  // channel used inside module
int a;

// use
a = sig_a.read(); // read local channel
sig_a.write(5); // write local channel
// read() and write() are event methods of sc_signal
```

Source: Forte Design Systems.

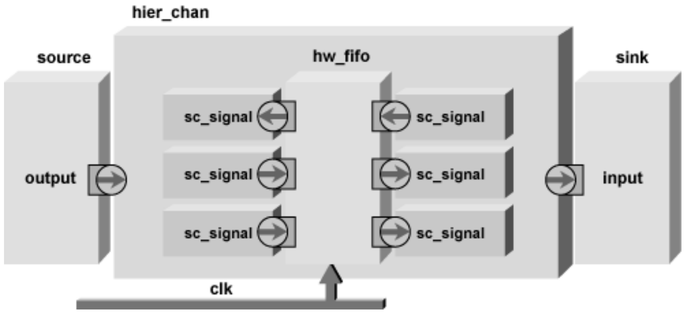
Another Primitive Channel Example



```
// main.cpp
int sc_main(int argc, char **argv) {
    sc_fifo<int> fifo_chan(5); // fifo channel of depth 5
    // module instantiations
    source src("src");
    sink snk("snk");
    // connect src & snk
    src.output(fifo_chan);
    snk.input(fifo_chan);
    sc_start();
}
```

Source: Forte Design Systems.

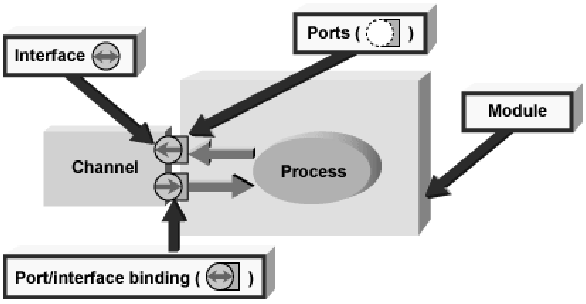
Change It to Hierarchical Channel



- ◆ The same source and sink modules as in the previous example
- ◆ As an adapter which "translates" between the interfaces for the hw_fifo and the source and sink modules
- ◆ The module hw_fifo implements a FIFO

Source: Forte Design Systems.

Basic Communication Structures (3/3)



Port

- Created from the base class sc_port and bound to an interface type
- Syntax
  - sc_port<interface_type, N> port_name, port_name,... ;
    - N is the number of channels that can be connected to the port

Source: Forte Design Systems.

Port Example

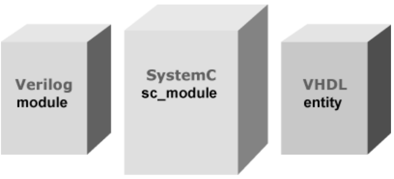
```
SC_MODULE(my_module) {
   // port in_p with 2 channels connected
   sc_port<sc_signal_in_if<int>,2> in_p;
   sc_port<sc_signal_inout_if<int> > out_p;

   // body of module
};
```

Source: Forte Design Systems.

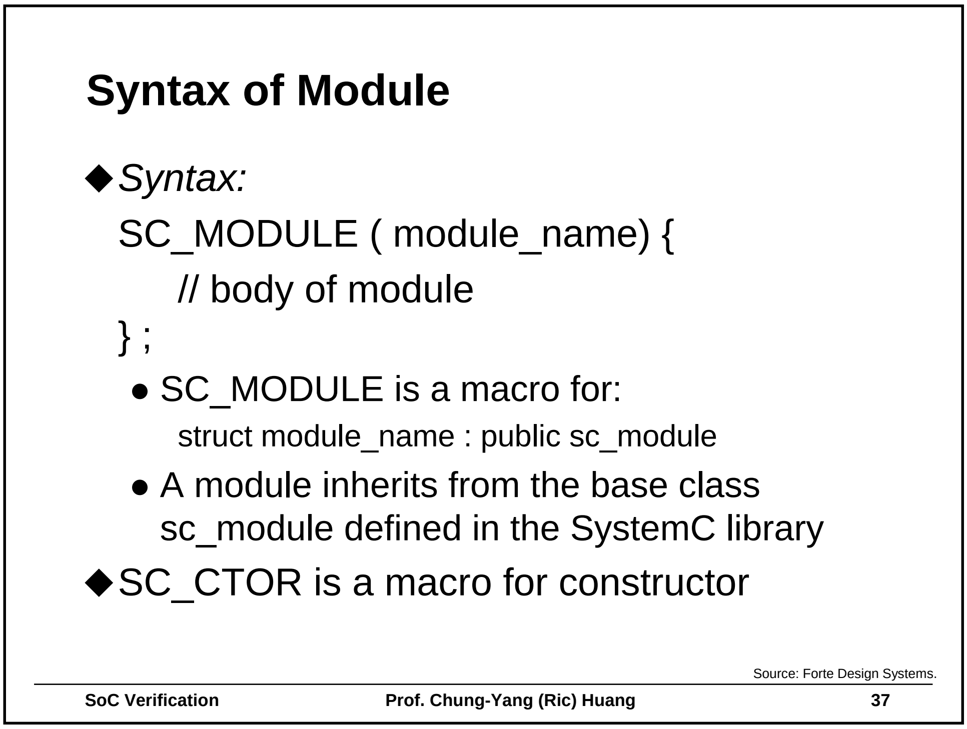
What is modules in SystemC?

Modules in SystemC

- A module is a container
- It is similar to a Verilog "module" or a VHDL "entity.“
- Each module is described using a header file (module_name.h) and an implementation file (module_name.cpp)
- Modules contain
  - Ports
  - Internal channel variables
  - Internal data variables
  - Processes of different types
  - Other methods
  - Instances of other modules
  - Constructor



Source: Forte Design Systems.



Syntax of Module

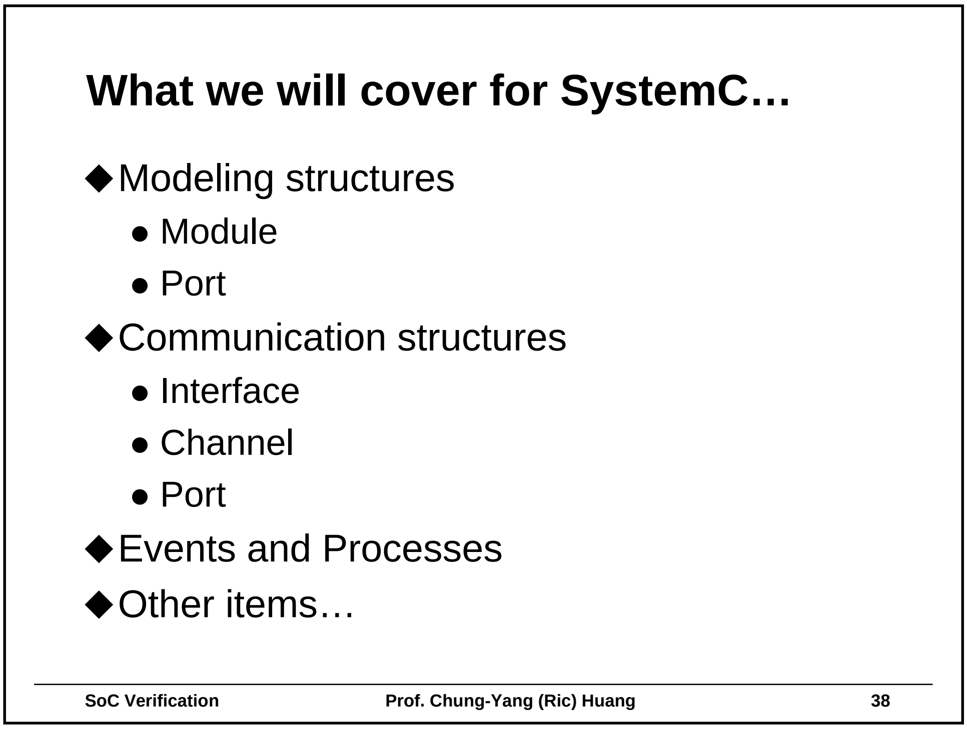
- *Syntax:*

```
SC_MODULE ( module_name) {
     // body of module
} ;
```

  - SC_MODULE is a macro for:

    struct module_name : public sc_module
  - A module inherits from the base class sc_module defined in the SystemC library
- SC_CTOR is a macro for constructor

Source: Forte Design Systems.



What we will cover for SystemC…

- Modeling structures
  - Module
  - Port
- Communication structures
  - Interface
  - Channel
  - Port
- Events and Processes
- Other items…

Events in SystemC

- ◆ Basic synchronization object
  - ● Used to synchronize between processes
  - ● Declared inside a module
- ◆ *Syntax:*

  sc_event event_name, event_name,... ;
- ◆ *Example :*

```
SC_MODULE(ex) {
        // Ports not shown
        // events
        sc_event ev_1, ev_2;
        // Rest of the module not shown
};
```

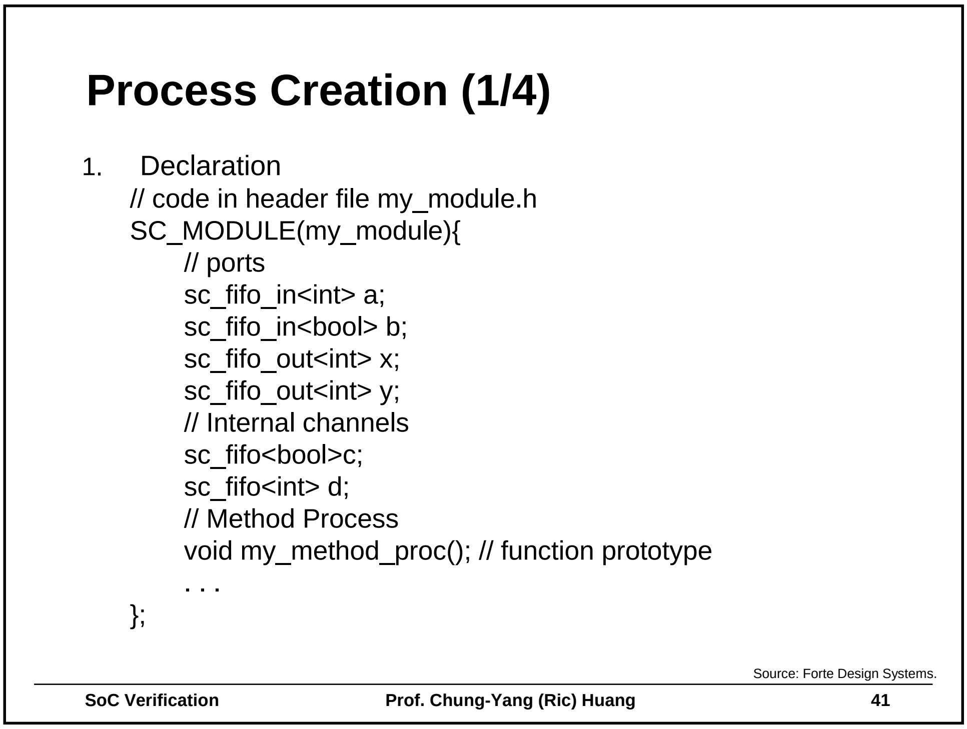
Source: Forte Design Systems.

Processes in SystemC

- ◆ Describes functionality
- ◆ Do not call processes directly in the code

  ➔ Invoked by the kernel based on either its static or dynamic sensitivity lists
- ◆ Not hierarchical - can not have a process inside another process
- ◆ Use channels or events to communicate with each other
- ◆ Registered inside the constructor of the module class
- ◆ Types
  - ● Methods (SC_METHOD)
  - ● Threads (SC_THREAD)
  - ● Clocked Threads (SC_CTHREAD)

Source: Forte Design Systems.

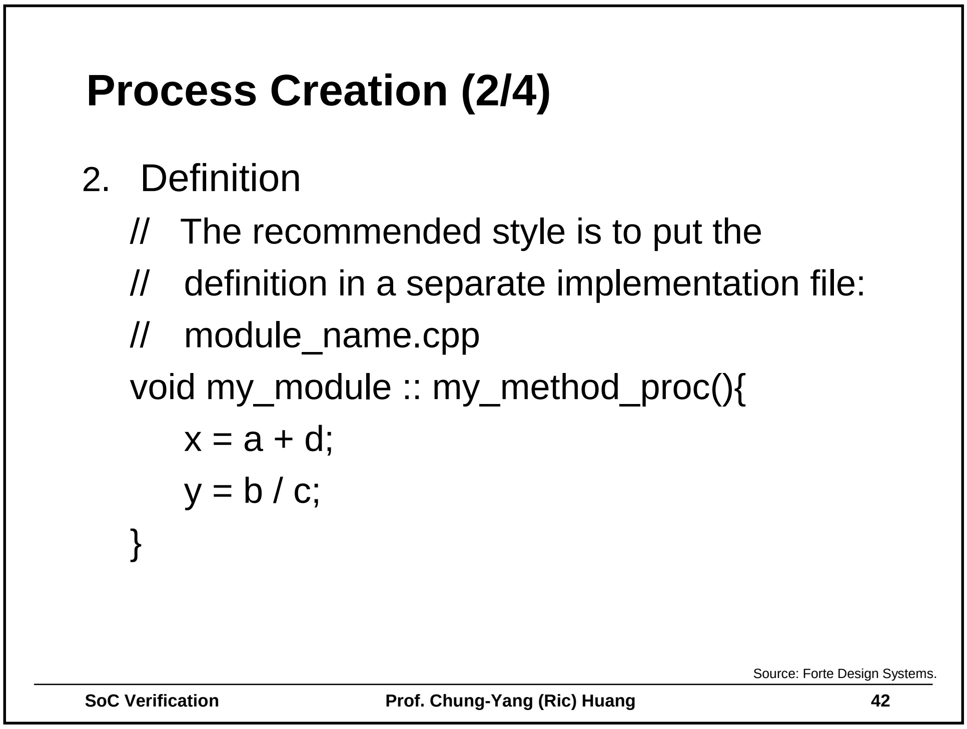


Process Creation (1/4)

1. Declaration

```
// code in header file my_module.h
SC_MODULE(my_module){
    // ports
    sc_fifo_in<int> a;
    sc_fifo_in<bool> b;
    sc_fifo_out<int> x;
    sc_fifo_out<int> y;
    // Internal channels
    sc_fifo<bool>c;
    sc_fifo<int> d;
    // Method Process
    void my_method_proc(); // function prototype
    . . .
};
```

Source: Forte Design Systems.



Process Creation (2/4)

2. Definition

```
//  The recommended style is to put the
//   definition in a separate implementation file:
//   module_name.cpp
void my_module :: my_method_proc(){
    x = a + d;
    y = b / c;
}
```

Source: Forte Design Systems.

Process Creation (3/4)

3. Registration (in module constructor)

```
SC_CTOR(my_module) {
    // register method process
    SC_METHOD(my_method_proc);
    // rest of constructor not shown
    . . .
}
```

Source: Forte Design Systems.

Process Creation (4/4)

4. Static Sensitivity Declaration (optional)

```
my_channel a; // implements my_event
sc_port<my_interface<int> > b; // connected to my_channel
sc_event c;
// process
void my_method_proc();
SC_CTOR(my_module) {
    // register thread process
    SC_METHOD(my_method_proc);
    // declare sensitivity list
    sensitive(a.my_event ); // sensitive to my_event on a
    sensitive << b->my_event(); // sensitive to my_event on b
    sensitive(c); // sensitive to event c
}
```

Source: Forte Design Systems.

Other items…

- ◆Port binding (module)
- ◆Dynamic sensitivity declaration (process)
- ◆Time resolution
- ◆Clocks
  - ● Syntax

```
sc_clock clock_name ("name", period, duty_cycle,
                start_time, positive_first ) ;
```

- ◆sc_start()

sc_start()

- ◆At the bottom of the sc_main() function and before the return statement
  - ● Execution of this statement marks the end of elaboration and the start of simulation
- ◆Have an optional argument: sc_start(arg)
  - ● Specifies the number of time units to simulate (default = forever)

Source: Forte Design Systems.

Online SystemC Trainings

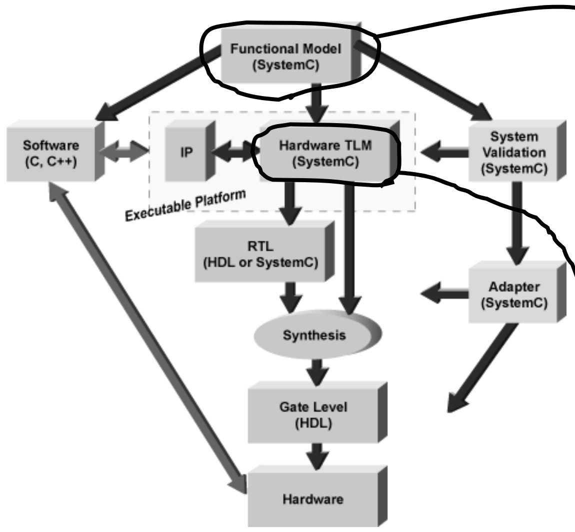
1. http://www.forteds.com/systemc/training/index.asp (Free with registration)
2. http://www.doulos.com/knowhow/systemc/tutorial/ (Free)

Official SystemC website

- ◆ Open SystemC Initiative (OSCI)

  http://www.systemc.org

SystemC in the Design Flow



◆ Funtional Model (optional)

= System architectural + System performance models

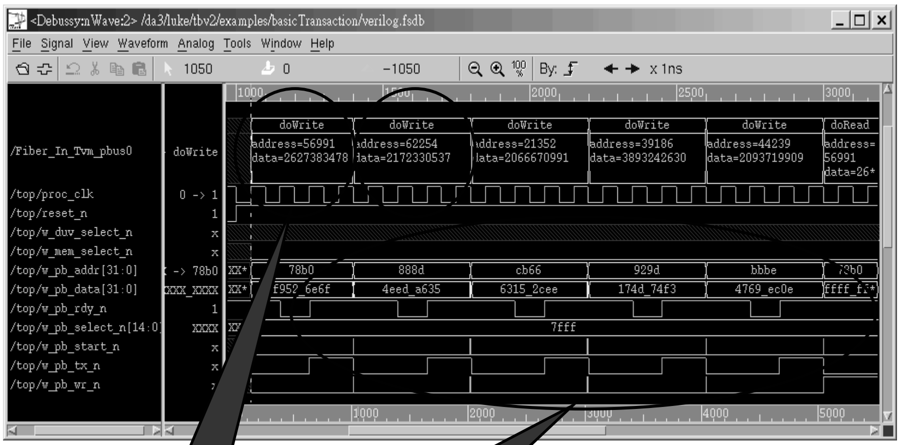
- Mostly untimed model
- No pin level details;
- Modeling communication protocols
- Behavior is modeled algorithmically

◆ TLM

= Transaction-Level Model

Source: Forte Design Systems.

What is Transaction-Level Model?



Transaction Info

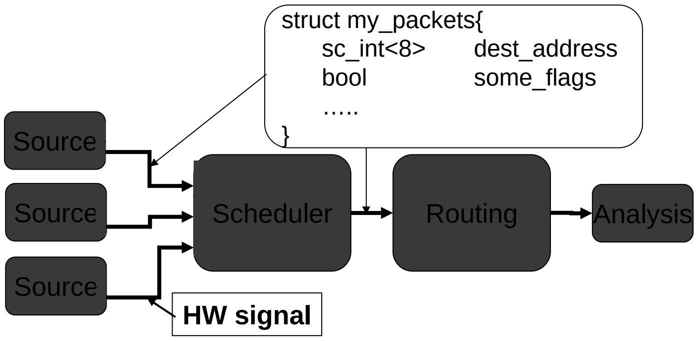
Signal value changes

Source: SprintSoft Inc.

Let’s look at the following example…

Design Planning: Data Abstraction

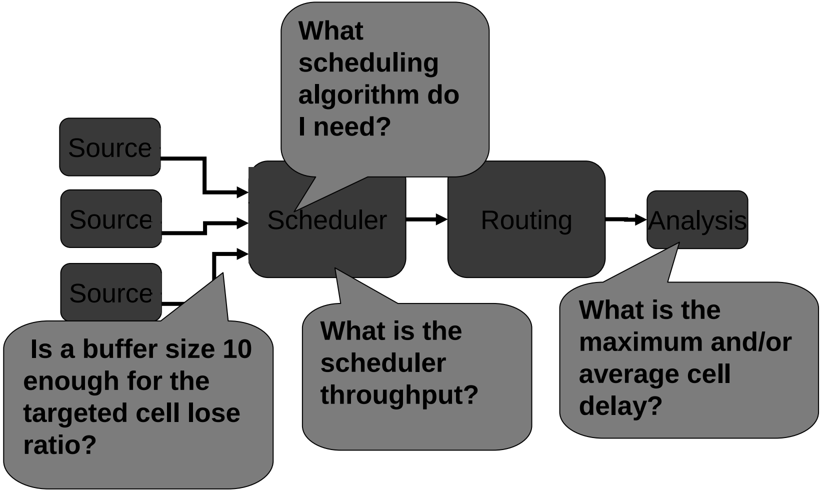
Many designers start with data abstraction



```
struct my_packets{
      sc_int<8>       dest_address
      bool            some_flags
      …..
}
```

Source: Synopsys Inc.

Analysis Questions



What scheduling algorithm do I need?

Is a buffer size 10 enough for the targeted cell lose ratio?

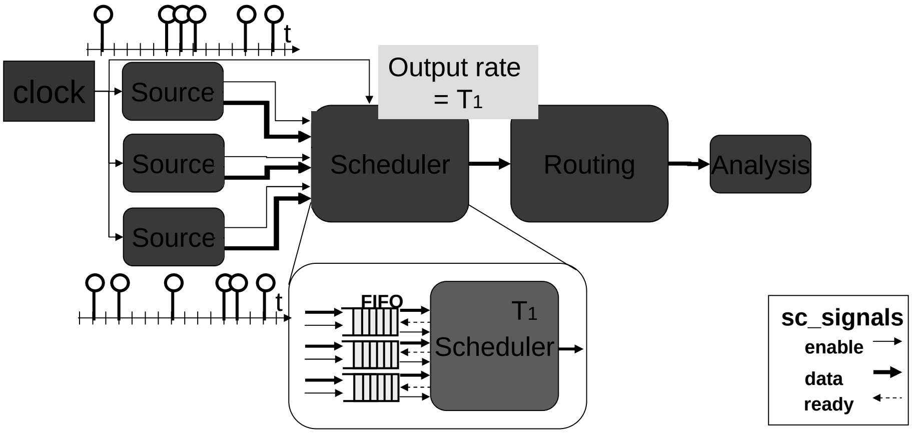
What is the scheduler throughput?

What is the maximum and/or average cell delay?

- **Model does not answer these questions**

Source: Synopsys Inc.

Final Model with Data Abstraction



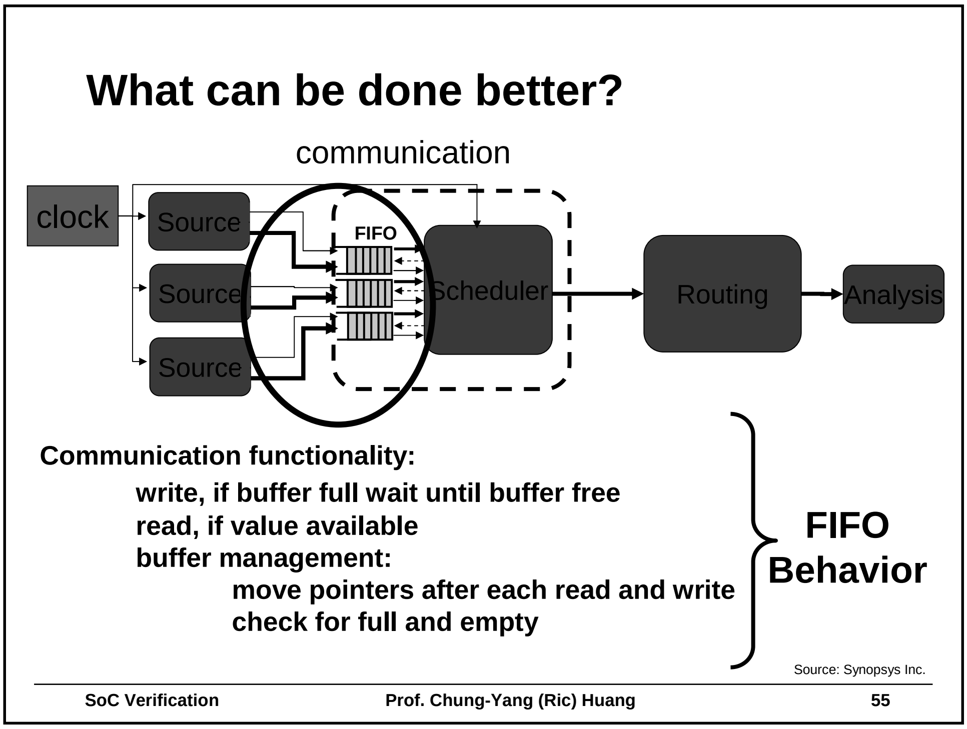
- Inefficient way of connecting blocks
- Fixed to one implementation (FIFO)
- No flexibility

Source: Synopsys Inc.

The Mistakes

- ◆Mistake 1: Data abstraction does not solve the problem
  - ● Model does not allow to answer the specification questions
- ◆Mistake 2: Using HW signals as communication mechanism
  - ● HW signals are specific for reactive HW
  - ● Signal communication mechanism not suitable for high level models

Source: Synopsys Inc.



What can be done better?

communication
clock
Source
Source
Source
FIFO
Scheduler
Routing
Analysis

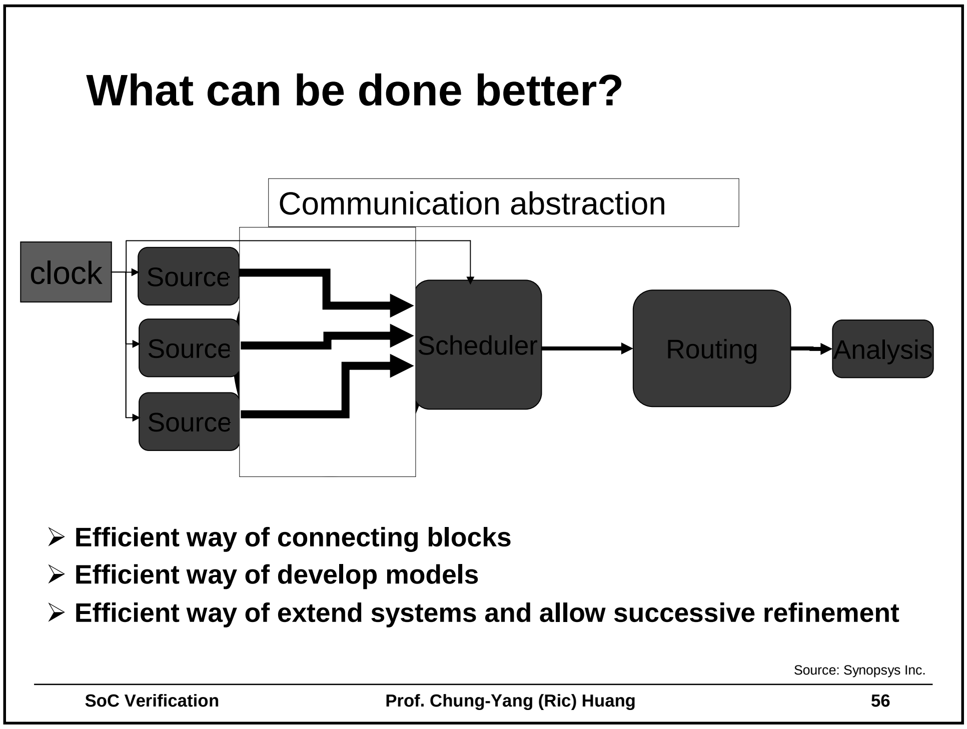
Communication functionality:

write, if buffer full wait until buffer free
read, if value available
buffer management:
move pointers after each read and write
check for full and empty

FIFO Behavior

Source: Synopsys Inc.

SoC Verification Prof. Chung-Yang (Ric) Huang 55



What can be done better?

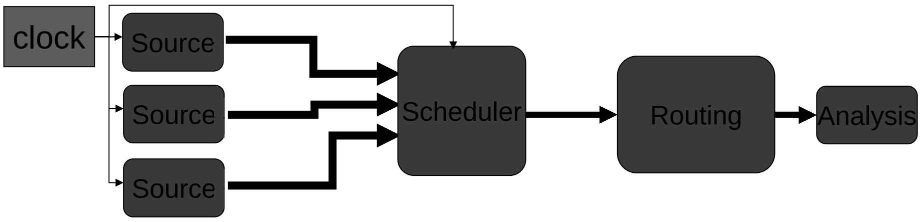
Communication abstraction
clock
Source
Source
Source
Scheduler
Routing
Analysis

- **Efficient way of connecting blocks**
- **Efficient way of develop models**
- **Efficient way of extend systems and allow successive refinement**

Source: Synopsys Inc.

SoC Verification Prof. Chung-Yang (Ric) Huang 56

What is Transaction Level?



clock

Source

Source

Source

Scheduler

Routing

Analysis

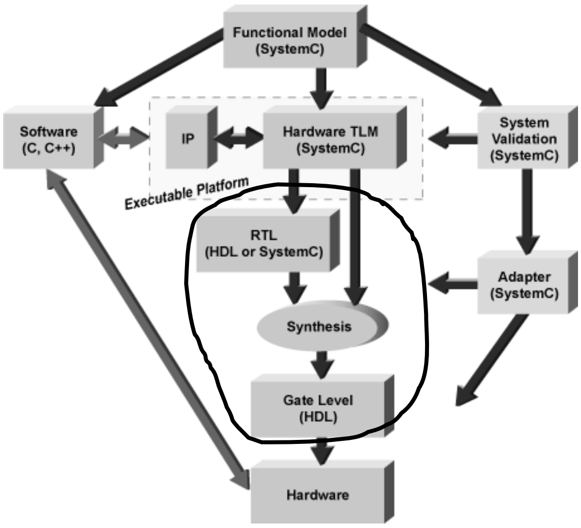
Transaction level is not only data abstraction,
it is also communication abstraction

Source: Synopsys Inc.

In short, TLM ---

- ◆Data + communication abstraction
  - ● Describe how data is communicated between modules
  - ● Can be used to simulate and analyze the system behavior
- ◆No RTL implementation details
  - ● Greater simulation speed
  - ● Shorter development (modeling) time
  - ● Easier for system analysis and debug

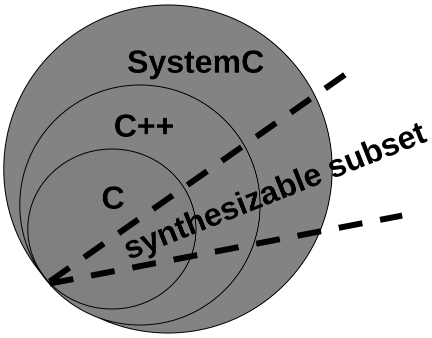
SystemC in the Design Flow



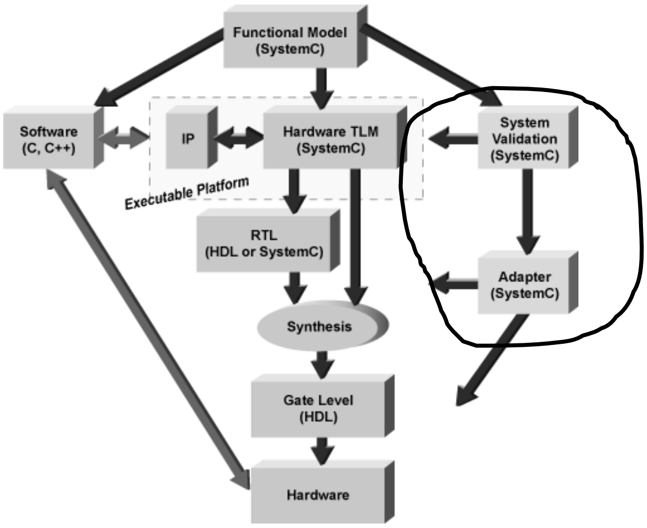
Source: Forte Design Systems.

SystemC Synthesizable Subset

- A subset of SystemC constructs that can be unambiguously translated into RTL or gate HDL



SystemC in the Design Flow



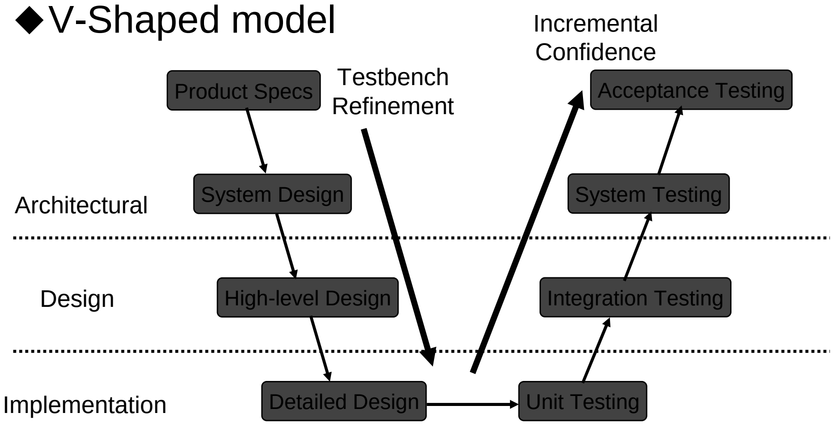
Source: Forte Design Systems.

System-Level Verification

- With current available technologies and tools out there, simulation (testbench) approach seems to be the only viable solution
- Testbench creation strategies: 2 options
  1. System-level testbench first
     - Pro: system-level is verified first; problem can be found earlier
     - Con: takes more planning and forethought; needs to migrate testbench to lower level(s)
  2. Unit-level testbench first
     - Pro: Familiar to designers; testbench can be created quickly
     - Con: Duplicated efforts; system is not well tested

System or Unit?

- ◆System-level is more recommended
- ◆V-Shaped model



source: “System-on-a-chip Verification”, P. Rashinkar, et al

SystemC Verification (SCV) Library

- ◆SCV 1.0 is now ready for download
- ◆A collection of classes that supports various verification in SystemC
  - Data introspection
  - Transaction recording
  - Constrained randomization
  - Utility data types

Data Introspection

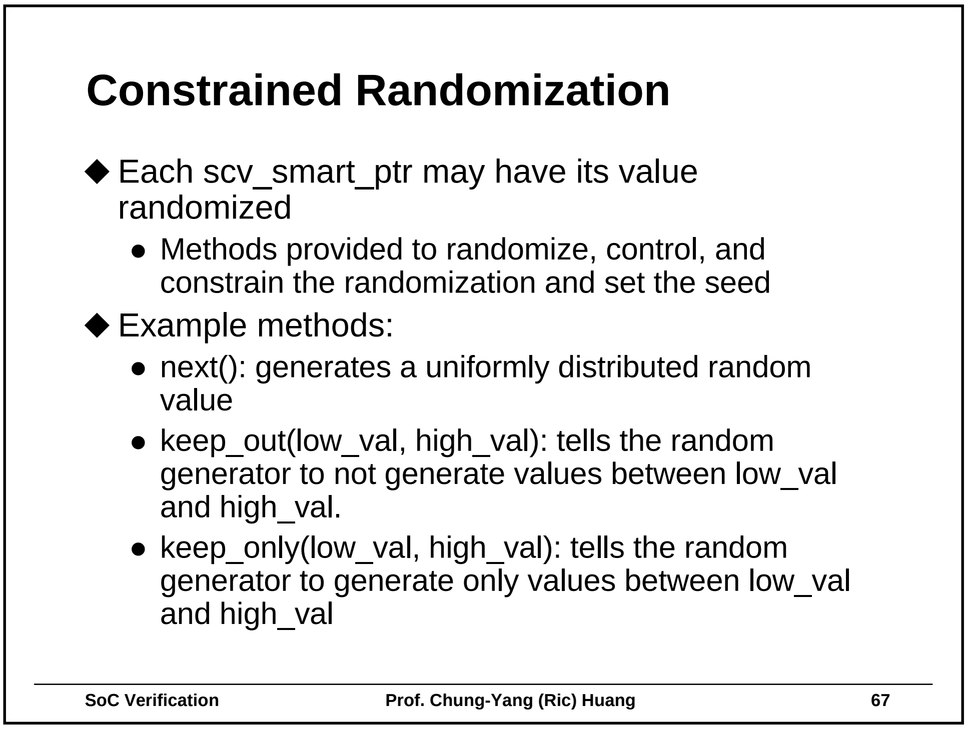
- Allows arbitrary data types to provide information about themselves and to manipulate their contents
  - Provides a "wrapper" class implementation for normal C++ and SystemC types. The wrapper type acts like a pointer, but has extra introspection methods
  - e.g. scv_smart_ptr smart_int;
- Things you can do to svc_smart_ptr
  - What type are you?
  - What is your value?
  - Set the value directly.
  - Set the value through randomization.
  - Register callbacks.
    - When data changes.
    - When accessed.
    - When written

Source: Forte Design Systems.

SCV Testbench Generation Options

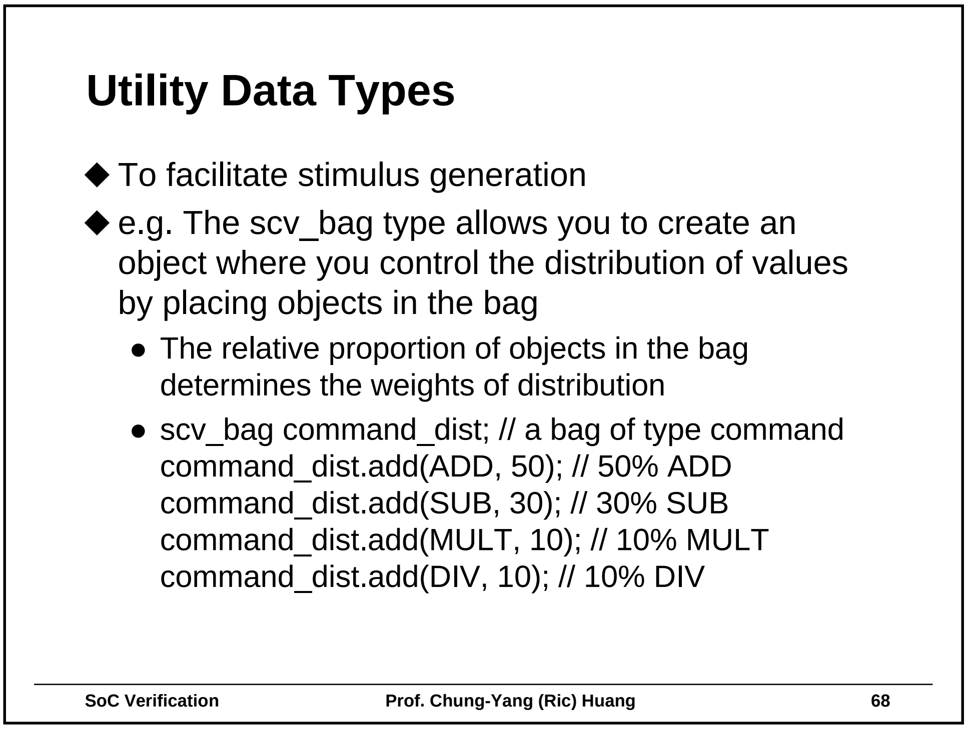
- Directed Tests
  - Traditional method for testbenches
- Weighted Randomization
  - Focuses stimulus on interesting cases
- Constrained Randomization
  - Enables complex and thorough tests to be developed quickly
- Combinations of the above

Source: Forte Design Systems.



Constrained Randomization

- Each scv_smart_ptr may have its value randomized
  - Methods provided to randomize, control, and constrain the randomization and set the seed
- Example methods:
  - next(): generates a uniformly distributed random value
  - keep_out(low_val, high_val): tells the random generator to not generate values between low_val and high_val.
  - keep_only(low_val, high_val): tells the random generator to generate only values between low_val and high_val

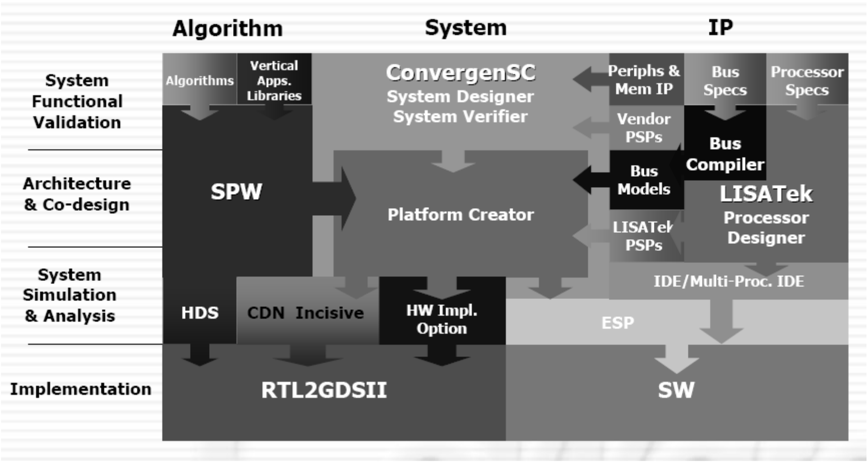


Utility Data Types

- To facilitate stimulus generation
- e.g. The scv_bag type allows you to create an object where you control the distribution of values by placing objects in the bag
  - The relative proportion of objects in the bag determines the weights of distribution
  - scv_bag command_dist; // a bag of type command
    command_dist.add(ADD, 50); // 50% ADD
    command_dist.add(SUB, 30); // 30% SUB
    command_dist.add(MULT, 10); // 10% MULT
    command_dist.add(DIV, 10); // 10% DIV

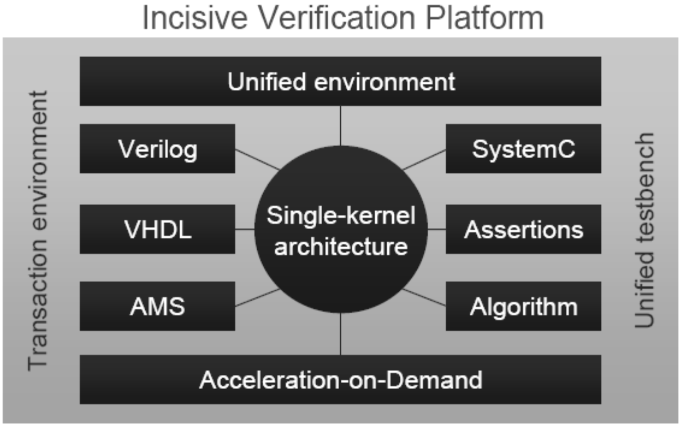
EDA Tools for System-Level (1)

- ◆Cadence / CoWare



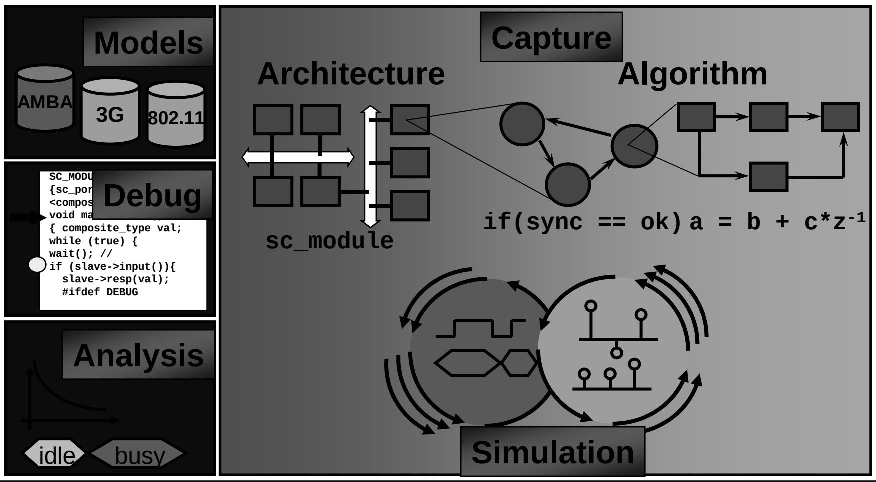
EDA Tools for System-Level (2)

- ◆Cadence Incisive Verification Platform



EDA Tools for System-Level (3)

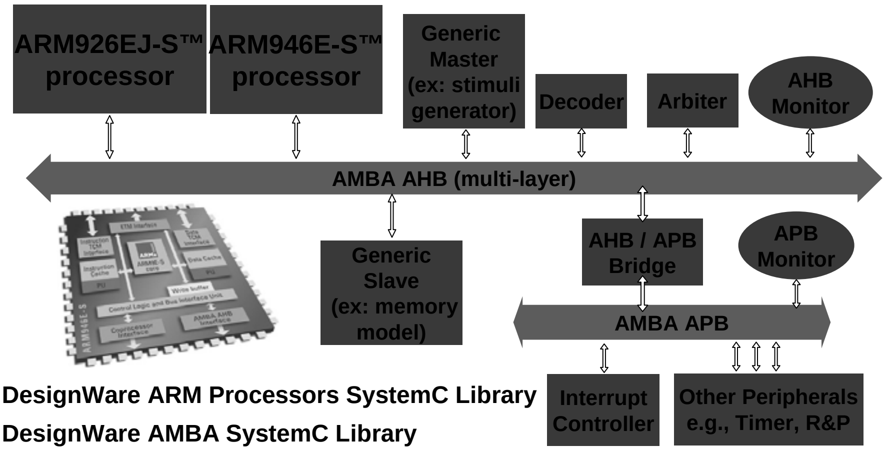
◆Synopsys ConCentric System Studio



```
SC_MODU
{sc_por
<compos
void ma
{ composite_type val;
while (true) {
wait(); //
if (slave->input()){
  slave->resp(val);
  #ifdef DEBUG
```

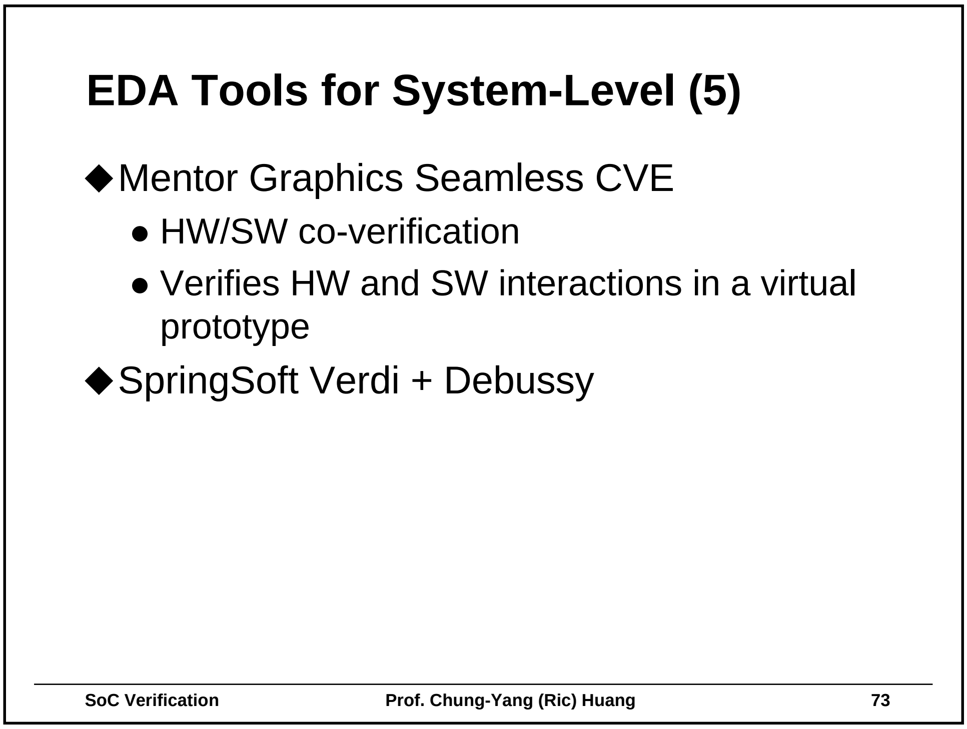
EDA Tools for System-Level (4)

◆Synopsys DesignWare SystemC Libraries



DesignWare ARM Processors SystemC Library

DesignWare AMBA SystemC Library



EDA Tools for System-Level (5)

- Mentor Graphics Seamless CVE
  - HW/SW co-verification
  - Verifies HW and SW interactions in a virtual prototype
- SpringSoft Verdi + Debussy