Accelerating “Concept to RTL” for System-on-Chip Designs with SystemC

Agenda

- 1:00 p.m. Introduction and Seminar Overview
- 1:10 p.m. SOC design challenge and SystemC methodology
- 1:40 p.m. Introduce SystemC language
- 2:30 p.m. Tea Break
- 3:00 p.m. The Concept to RTL flow for the SOHO router
  - Today’s design – SOHO Router
  - Architecture SOC design
  - SW implementation & debug
  - Moving from Transaction Level Models to RTL
  - Reaching golden RTL
- 4:30 p.m. Demo: Multiprocessor ARM design for IP Router
- 5:00 p.m. Q&A and Wrap up
Today’s Chip design challenges.

**Products**
- Multi-domain/application chips integrating HW and SW and analogue
- Conformance to Industry standards is critical for product success
- Re-use of NRE in follow-on projects is a requirement
- ... while shrinking Time To Market (TTM)

Getting it right the first time

IC/ASIC Designs Having One or More Re-spins by Type of Flaw

<table>
<thead>
<tr>
<th>Type of Flaw</th>
<th>0%</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
<th>60%</th>
<th>70%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic or Functional</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analog Circuit</td>
<td>35%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Noise</td>
<td>29%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slow Path</td>
<td>28%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clocking</td>
<td>25%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: Collett International Research (Apr02)

IBS, SOC design Cost study

<table>
<thead>
<tr>
<th>Time To Market counts!</th>
<th>3 Months Late</th>
<th>6 Months Late</th>
<th>9 Months Late</th>
<th>12 Months Late</th>
</tr>
</thead>
<tbody>
<tr>
<td>Market characteristics</td>
<td>Fast Moving market</td>
<td>Slow Moving market</td>
<td>Fast Moving market</td>
<td>Slow Moving market</td>
</tr>
<tr>
<td>Price reduction (%)</td>
<td>7.8</td>
<td>3.8</td>
<td>1.6</td>
<td>51.7</td>
</tr>
<tr>
<td>Volume reduction (%)</td>
<td>2.8</td>
<td>1.4</td>
<td>0.8</td>
<td>4.7</td>
</tr>
<tr>
<td>Power</td>
<td>1.2</td>
<td>0.6</td>
<td>0.4</td>
<td>2.5</td>
</tr>
<tr>
<td>Firmware</td>
<td>0.8</td>
<td>0.3</td>
<td>0.1</td>
<td>2.0</td>
</tr>
<tr>
<td>Other</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: Collett International Research (Apr02)
SOC design applications

Key challenges, ... and solutions

- Time to market, time to volume
  - Reduce design time; achieve first time right!
- Adherence to public standards
  - While standards are still moving!
- Assure promised performance

1. Create an executable, golden architectural model before committing to RTL design
   - Use this model as a virtual system prototype to start Software development early
   - Use this model as a System level test bench to perform RTL module level verification
2. Refine this model into golden RTL through RE-USE and conformance testing before going to silicon

Main design tasks

- What functions should the system perform?
- Through what interfaces does it communicate?
- How is this functionality achieved?
  - Performance goals? What in HW, what in SW, ?
- How is the HW functionality implemented?
  - Does the RTL do what it is supposed to do?
- Realize the SW functionality? Create SW
  - Does the SW + HW implement the spec
- Does the silicon implement the RTL
  - Timing closure, physical effects?
Typical flow and timeline

- Overlap in specification/Architecture phase and RTL design phase; multiple design changes
  - Architecture design done informally
- SW development starting late in the project
- Little SW simulation pre-silicon (takes to long)

Accelerating concept to RTL

Objectives for a new flow

- Introduce the concept of Architecture closure:
  - Achieve a reduction # of RTL iterations
  - Can perform concurrent Hardware and SW design
  - Shorten the time it takes to get to golden RTL
- Run SW on the architecture model

Especially when below 150 nm physical effects impact architecture!
1. Reaching Architectural closure

Goal: Model the entire system (HW & SW) to verify that it meets the performance goals optimally.

- Validate the Architecture
  - Eliminate bottlenecks in bus transactions
  - Refine data buffer structure/management
  - Close on HW/SW partitioning
- Perform Software based testing
  - Verify system setup, peripheral drivers and key application SW features
  - Optimize timing-critical tasks of the embedded software

2. Reaching RTL closure

Goal: Implement and verify the architecture in RTL

- Individual block level
  - Implement/Synthesize RTL blocks, or
  - Import (& re-validate) design IP
  - “Prove” block level functionality and performance
  - Check conformance to specifications/standards
- Full chip level
  - Resolve micro-architecture corner cases (clock domains, FIFOs, handshakes, split bus transactions)
  - Integrate imported IP, show chip level integrity
  - Perform software execution (reset, ......)
Accelerating Concept to RTL flow

Typical flow: Step 1 and 2 performed on RTL model

- Architecture closure
- RTL closure

Incomplete and Slow!
Limited SW

New flow: Step 1 on transaction level, step 2 on RTL model

- Architecture closure
- RTL closure

Complete and Fast!
SW early.

Virtual prototyping helps find bugs

- TLM Execution speed is > 100 times faster than RTL
  - Longer stimuli can be applied
  - HW – SW co-simulation can be performed on large portion of the SW code (not only set up)
- Realistic conditions are applied to the design
  - Traffic (burst length, error density, collision) is injected
  - Spot Bus bandwidth problem and data accumulation during peak utilization
  - Adjust HW blocks performance (add pipelining, increase processing width)
Virtual prototyping helps find bugs

- SW is getting timely and functionally accurate response from the HW:
  - SW tasks interrupt and priority scheme
  - Simulation of Random events (# of table lookup iterations)
  - Adjust Ram-based Buffer size, buffer organization and management task
  - Optimized SW code performance for some time-critical routines
- Critical application code can be executed early

Synopsys’ integrated solution

System Level IP (RDK’s)

System Level
- Algorithm
- Architecture
- SW execution

System Studio™

Smart Verification
- VCS™
- NanoSim™
- MX
- Scirocco™
- VERA™

SystemC

VIP

IIP

VIP

IIP

DesignWare Library + Star IP

Synopsys Professional Services
Synopsys Implementation Platform Solution

Synopsys’ System Level solution
CoCentric System Studio

- SystemC Engine
- GUI Design Entry
- Debugging
- Visualization
- Sim Control
- Co-Simulation
  - Processor ISS
  - VCS™ (Verilog)
  - Scirocco (VHDL)
  - VERA®

Close on the Architecture
Synopsys’ Smart Verification
VCS. Scirocco, Vera

- Fast RTL simulation
- Highest capacity
- Stimulus generation
- Complex scenarios

- Assertion simulation
- Assertion proofs
- Profiling tools
- Coverage tools

Design & Verification IP
The models you need

Synopsys IP
Essential elements for project success

SOC & ASIC
Processor buses
Protocols
Networking
System Level
Wireless
Multimedia

http://www.synopsys.com/products/designware/ipdir
Agenda

- 1:00 p.m. Introduction and Seminar Overview
- 1:10 p.m. SOC design challenge and SystemC methodology
- 1:40 p.m. Introduce SystemC language
- 2:30 p.m. Tea Break
- 3:00 p.m. The Concept to RTL flow for the SOHO router
  - Today’s design – SOHO Router
  - Architecture SOC design
  - SW implementation & debug
  - Moving from Transaction Level Models to RTL
  - Reaching golden RTL
- 4:30 p.m. Demo: Multiprocessor ARM design for IP Router
- 5:00 p.m. Q&A and Wrap up

What is SystemC?

... A language extension to C/C++ to describe and simulate HW/SW systems:
  - Multiple levels of abstraction
  - Concurrency
  - IP reuse methodology
  - For the SOC design challenge
... A C++ class library to standardize C-based system modelling
... A basic simulation kernel
C/C++ out of the box

- C/C++ does not support
  - Hardware style communication
    - Signals, events, … etc.
  - Notion of time
    - Time sequenced operations
  - Concurrency
    - Hardware and systems are inherently concurrent, i.e., components operate in parallel
  - Reactivity
    - HW is inherently reactive, it responds to stimuli and is in constant interaction with its environment, which requires handling of exceptions
  - Hardware datatypes
    - Bit type, bit-vector type, multi-valued logic type, signed and unsigned integer types and fixed-point types

SystemC Goes Beyond C/C++ & OOP

- Separate Communication from Functionality
- Separate Data from Control
- Concurrency

Extends C/C++ for complex HW/SW systems design

Extends computational capabilities of C

SW development

© 2003 Synopsys, Inc. (21)
SystemC Language Architecture

<table>
<thead>
<tr>
<th>Elementary Channels</th>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal, Timer, Mutex, Semaphore, FIFO, etc.</td>
<td>4-valued logic (0, 1, X, Z)</td>
</tr>
<tr>
<td>Time</td>
<td>4-valued logic-vectors</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Bits and bit-vectors</td>
</tr>
<tr>
<td>Modules</td>
<td>Arbitrary-precision integers</td>
</tr>
<tr>
<td>Processes</td>
<td>Fixed-point numbers</td>
</tr>
<tr>
<td>Interfaces</td>
<td>C++ user-defined types</td>
</tr>
<tr>
<td>Ports</td>
<td>C++ built-in types (int, char...)</td>
</tr>
<tr>
<td>Channels</td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td></td>
</tr>
<tr>
<td>Event-driven sim. kernel</td>
<td></td>
</tr>
</tbody>
</table>

C++ Language Standard

SystemC 2.0 for different Models of Computation

- Static Multi-rate Dataflow
- Dynamic Multi-rate Dataflow
- Communicating Sequential Processes/Control Models
- Discrete Event as used for
  - RTL hardware modeling
  - Network modeling (e.g. stochastic or “waiting room” models)
  - Transaction-based SoC platform modeling
- Kahn Process Networks
Open SystemC Initiative roadmap

Modeling - A SystemC "System"

- System consists of a set of modules
- Modules contain concurrent processes
- Process describes functionality
- Processes communicate with each other through channels or events
- Inter-Module communication is through channels
Hierarchy

- Modules are used to create hierarchy
- Top level is not a module but is a SystemC function sc_main()
- May instantiate modules inside sc_main()
- May instantiate modules inside of other modules

Basic Modeling Structure: Module

Module represents:
- system, block, board, chip...
Ports represent:
- interface, pins

Module body
- Module instances
- concurrent processes

Module
- Module Instance
  - Channel
- Process
  - Channel
Basic Modeling Structure: Communication

- **Interfaces**
  - Defines set of access methods for a channel
- **Channels**
  - Implements an interface's methods
- **Ports**
  - Object through which processes access a channel's interface

Inter-Module Communication: Channels, Interfaces, Ports

- **Interface**
  - Defines set of access methods
  - Does not provide the implementation of the methods
  - Purely functional, no data
  - Examples: `sc_fifo_in_if`, `sc_fifo_out_if`
- **Channel**
  - Implements an interface's methods
  - Can implement one or more interfaces
  - Container for communication functionality
  - Example: `sc_fifo<T>`
- **Port**
  - Provides dedicated access methods to a channel
  - Object through which modules (and hence its processes) can access a channel’s interface
  - Process accesses channel by applying interface methods to port
  - Defined in terms of an interface type
  - Can only be used with channels that implement this interface type
  - Example: `sc_port<sc_fifo_in_if<T>>`
Channels: Primitive & Hierarchical

- Primitive channels
  - No visible structure
  - No processes
  - Can not (directly) access other primitive channels
    - `sc_signal<T>`, `sc_signal_rv<N>`
    - `sc_fifo<T>`, `sc_mutex`, `sc_semaphore`
    - `sc_buffer<T>`

- Hierarchical channels
  - Modules
    - Structure
    - Processes
    - Other modules
  - Can (directly) access other channels
  - For modeling complex channels

Channels & Ports

- In general different channels require different interface methods

- Ports are “bound” to an interface
  - To connect a channel to a port
    - Channel must implement the interface the port is bound to
    - Allows for refinement of channel independent of ports

- Ports may be bound to multiple channels
Events and Dynamic Sensitivity

- Event (sc_event) is the basic synchronization object
  - Events are used to synchronize between processes
  - Channels use events to implement blocking

- Dynamic sensitivity for triggering processes
  - Sensitivity list for a process does not need to be defined before simulation starts - can be altered during simulation
  - Enabled through events

- Wait (wait())
  - Processes may wait for an event

- Dynamic sensitivity coupled with ability of process to wait on an event provide for:
  - Efficient simulation (faster)
  - Simple modeling at higher levels of abstraction

Simple Example - 1

![Simple Example Diagram]
Simple Example - 2

```c
#include "systemc.h"
#include "adder.h"
#include "stimgen.h"
#include "monitor.h"

int sc_main(int argc, char *argv[]) {
    // Create fifos with a depth of 10
    sc_fifo<int> s1(10);
    sc_fifo<int> s2(10);
    sc_fifo<int> s3(10);
    // Module instantiations
    // Stimulus Generator
    stimgen stim("stim");
    stim(s1, s2);
    // Adder
    adder add("add");
    add(s1, s2, s3);
    // Response Monitor
    monitor mon("mon");
    mon.re(s3);
    // Start simulation
    sc_start();
    return 0;
}
```

Simple Example - 3

```c
// header file adder.h
SC_MODULE(adder) {
    // Input ports
    sc_port<sc_fifo_in_if<int> > in1;
    sc_port<sc_fifo_in_if<int> > in2;
    // Output ports
    sc_port<sc_fifo_out_if<int> > out;
    // sync process for adder
    void adder_proc() {
        // Module constructor
        SC_CTOR(adder) {
            SC_THREAD(adder_proc);
        }
    }
```

© 2002 Synopsys, Inc. Accelerating “Concept to RTL design Seminar
Simple Example - 4

```c++
#include "systemc.h"
#include "adder.h"

void adder :: adder_proc( ) {
    while (true) {
        out->write(in1->read() + in2->read());
        out->write(in1->read() + in2->read() + 2);
    }
}
```

Model of Time

- Underlying data type for time is an unsigned 64-bit integer
- Time units defined
  - enumeration:
    ```
    SC_FS, SC_PS, SC_NS, SC_US, SC_MS, SC_SEC
    ```
- Type provided to specify time values

Syntax:
```c++
sc_time var_name(time_value, time_unit); // time_value of type uint64
```

type defines the copy constructor, the assignment operator, and the relational and equality operators

Examples
```c++
sc_time t(20, SC_NS);
    // var t of type sc_time with value of 20ns
sc_time r_time( 1000, SC_NS);
```
## Simulation Functions

- **`sc_stop()` function**: (like `$finish` in Verilog)
  - No argument
  - Stops simulation
  - Causes `sc_start()` to return control to `sc_main()` routine

- **`sc_time_stamp()` function**
  - Returns `sc_time` object
  - The current simulation time *in the current default time unit*
  - No argument

- **`sc_simulation_time()` function**
  - Returns value or type `double`
  - The current simulation time *in the current default time unit*
  - No argument

## Agenda

- 1.00 p.m. Introduction and Seminar Overview
- 1:10 p.m. Introduce SystemC language
- 1:50 p.m. SOC design challenge and SystemC methodology
- 2:30 p.m. Tea Break
- 3:00 p.m. The Concept to RTL flow for the SOHO router
  - Today’s design – SOHO Router
  - Architecture SOC design
  - SW implementation & debug
  - Moving from Transaction Level Models to RTL
  - Reaching golden RTL
- 4:30 p.m. Demo: Multiprocessor ARM design for IP Router
- 5:00 p.m. Q&A and Wrap up
SOHO router products on the market

ADSL routing solutions address the SOHO needs

GS-R250S Duo
The GreatSpeed GS-R250S Duo ADSL Router is a two-port ADSL Gateway with built-in LAN connections for supporting two PCs simultaneously sharing an ADSL line, while using only one telephone jack.

Our SOHO router system

ADSL transceiver = MC145650 or equivalent
Ethernet transceiver = TDK 78Q2120B or equivalent

Communication processors based on AMBA bus architectures
SOHO Router
Datasheet

- **Line side interface:**
  - ADSL transceiver (IP over ppp over ATM over ADSL)

- **User side interface:**
  - 4 x 10/100 Mb Ethernet ports (IP over Ethernet)
  - USB 1.1 port

- **Main Networking features**
  - ATM processing (cell delineation, framing)
  - Support of one AAL5 connection UBR ATM cells
  - Support of OAM F4/F5 cells, filtering of empty cells
  - AAL5 SAR (MPOA support)
  - IP packet routing & switching
  - Management protocol
  - Ethernet encapsulation layer 2 and MII

- **Higher level protocol features addressed in SW**

SOHO Router
Suggested architecture

- ARM 926
- Ethernet 10/100
- RAM (SDRAM + Controller)
- AMBA AHB
- ATM TC
- DMA
- CPU and DMA accesses
  - Share a single bus
- USB
- Ethernet MAC
- Timer
- Interrupt controller
- GPIO
- UART

- Some Time-critical Tasks Implemented in SW

- Available in Designware
- Developed by Synopsys Professional Services
- IP from ARM/Synopsys
- Generated from Mempro
**Functional data path description**

- Shown here one direction (ADSL to Ethernet)
- Possible architecture bottle necks
- These blocks share the AMBA bus and the Ram device (single port)
- The time-critical Networking tasks and Interrupt service routines (ATM, Ethernet) are executed on the same processor core as the application SW (upper protocol layers)

**SOHO Router Principle of operations**
*(downstream traffic only)*

- ATM cell received
- ATM FIFO high threshold reached, ATM DMA transfer to RAM buffer
- ATM DMA triggers IT_ATM
- SW reads next ATM buffer from RAM
- AAL5 SW Processing
- SW writes Ethernet buffer to RAM
- Ethernet packets send on MII
- MII Collision detected
- Ethernet FIFO low threshold reached, Ethernet DMA transfer from RAM buffer
- Ethernet DMA triggers IT_ETHERNET
What we want to achieve on the SOHO router design

- Solve architectural challenge
  - Check that a simple bus topology meets the system performance requirements
  - Select FIFO ram buffer size, refine buffer management according to realistic traffic conditions
- Optimize SW timing-critical tasks
  - Executed under critical system conditions
- Get the right RTL
  - Leverage proven IP with......
  - a Smart verification approach

A synopsis 😊 of our approach

- Develop golden SystemC reference model target chip
  - Today’s Focus
- Concurrently refine the Hardware into golden RTL, and develop the Software
- Re-use the verification testbench and test cases from System level to RTL
Design Flow Overview

- **Architectural SOC**
  Executable specifications described at Transaction level (TLM)
  - Catch bugs while there are still cheap to fix
  - Check Chip Level Architecture performance

- **SW development and testing**
  Is performed early on a virtual prototype of the System
  - Get the HW accurate response, with an acceptable simulation speed

- **HW design** uses TLM reference platform as the main verification testbench
  - Reuse the same (successively refined) verification environment
  - Add remaining interfaces
  - Add HDL Assertions
  - Perform Coverage metrics

---

Agenda

- 1.00 p.m. Introduction and Seminar Overview
- 1:10 p.m. Introduce SystemC language
- 1:50 p.m. SOC design challenge and SystemC methodology
- 2:30 p.m. Tea Break
- 3:00 p.m. The Concept to RTL flow for the SOHO router
  - Today's design – SOHO Router
  - Architecture SOC design
  - SW implementation & debug
  - Moving from Transaction Level Models to RTL
  - Reaching golden RTL
- 4:30 p.m. Demo: Multiprocessor ARM design for IP Router
- 5:00 p.m. Q&A and Wrap up
Architecture SOC Design

- Problem Statement
- What is Transaction Level?
- Transaction Level Modeling with SystemC
- Construct a multi-processor platform
  - Use interface method calls to connect block
  - Implement interface method calls
- Model with Designware SystemC library and TLM simulation in CoCentric System Studio

Problems with Prototypes

- Does ARM946 meet the throughput requirements?
- Are the caches big enough?
- Is the Ethernet MAC timing correct?
- Is the bus bandwidth high enough?
- Do the processors communicate efficiently?
- Is the data throughput of the memories high enough?

Fast Answers for these questions is key
**Specification Questions**

- Audio quality?
- Bit error rate?
- Bus bandwidth?
- Cache size?
- Bus protocol?
- Pin behavior?

**Transaction Level for multi-processor platform analysis**

**Requirements for Platform Modeling – Block Level**

- Efficient modeling of interaction between processors, buses, memory and HW co-processors
- Cycle accuracy in crucial
- High simulation speed
- Efficient way of concurrent modeling
- Efficient way to extend systems
Requirements for Platform Modeling – System Level

- System assembling
  - Fast
  - Configurable blocks needed
  - Easy access to configuration
  - Documentation
- Analysis
  - Easy way to change platform configuration
  - Visualization
  - Easy access to results for further processing
  - Interactive simulation and regressions

Architecture SOC Design

- Problem Statement
- What is Transaction Level?
- Transaction Level Modeling with SystemC
- Construct a multi-processor platform
  - Use interface method calls to connect block
  - Implement interface method calls
- Model with Designware SystemC library and TLM simulation in CoCentric System Studio
Data Abstraction

Many designers start with data abstraction

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

Analysis Questions

- What scheduling algorithm do I need?
- What is the scheduler throughput?
- What is the maximum and/or average cell delay?
- Is a buffer size 10 enough for the targeted cell loss ratio?

➢ Model does not answer these questions
Final Model with Data Abstraction

- inefficient way of connecting blocks
- fixed to one implementation (FIFO)
- no flexibility

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
What can be done better?

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

---

What can be done better?

Communication abstraction

- Efficient way of connecting blocks
- Efficient way of develop models
- Efficient way of extend systems and allow successive refinement
What is Transaction Level?

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

Why Transaction Level Communication Abstraction?

- Reduce modeling effort for communication
- Increase simulation speed
- Reduce the effort to develop and extend complex systems
Architecture SOC Design

- Problem Statement
- What is Transaction Level?
- Transaction Level Modeling with SystemC
- Construct a multi-processor platform
  - Use interface method calls to connect block
  - Implement interface method calls
- Model with Designware SystemC library and TLM simulation in CoCentric System Studio

RT-Level SystemC

```c
#include "systemc.h"

SC_MODULE(adder_reg) {
    sc_in<sc_int<8> > a; // input port
    sc_in<sc_int<8> > b; // input port
    sc_out<sc_int<9> > c; // output port
    sc_in<bool> clk; // clock

    // Internal Signals
    sc_signal<sc_int<9> > temp;

    // Adder Process
    void add() {
        temp = a + b;
    }

    // Register update process
    void reg() {
        c = temp;
    }

    SC_METHOD(add);
    sensitive << a << b;

    SC_METHOD(reg);
    sensitive_pos << clk;
    }
};
```

Full RTL Guide
www.synopsys.com/sld
Transaction Level in SystemC: Port, Interface, Channel

```c
#include "my_interface.h"
SC_MODULE(my_master_module)
{
    sc_port<my_if> port1;

    SC_CTOR(my_master_module)
    {
        SC_THREAD(main_function)
    }
}
```

Transaction Level in SystemC: Port, Interface, Channel

```c
#include "my_interface.h"
SC_MODULE(my_master_module)
{
    sc_port<my_if> port1;
}
```

© 2003 Synopsys, Inc. (67)
### Transaction Level in SystemC: Port, Interface, Channel

#### Module

```cpp
#include "my_interface.h"
SC_MODULE(my_master_module)
{
    sc_port<my_IF> port1;

    SC_CTOR(my_master_module)
    {
        SC_THREAD(main_function)
        {
        }
    }
}
```

#### Interface

```cpp
// my_interface.h

class my_IF : public sc_interface
{
    // specify IF method calls here
    virtual int my_write();
    virtual void set_priority();
    virtual my_status get_status();
    ...
};
```
Transaction Level in SystemC: Port, Interface, Channel

module
process

channel

interface

port1

my_master_module::main_function() {
  ...
  port1->my_write();
  ...
}

What’s so cool about SystemC Transaction-Level Modeling?

It is ...
  • Most efficient way of modeling communication between different components in a system
  • easy to use
  • fast
    • use of Interface Method Calls (IMC) ⇒ function calls instead of HW signals and control FSMs
  • easy to develop and extend
Dynamic Sensitivity

- SystemC 1.0
  - Static sensitivity
    - Processes are made sensitive to a fixed set of signals during elaboration
- SystemC 2.0
  - Static sensitivity
  - Dynamic sensitivity
    - The sensitivity (activation condition) of a process can be altered during simulation (after elaboration)
    - Main features: events and extended wait() method

Waiting

```c
wait(); // as in SystemC 1.0
wait(event); // wait for event
wait(e1 | e2 | e3); // wait for first event
wait(e1 & e2 & e3); // wait for all events
wait(200, SC_NS); // wait for 200ns

// wait with timeout
wait(200, SC_NS, e1 | e2);
wait(200, SC_NS, e1 & e2);
```
TLM Multi-Processor Example

μC  μP  DSP

channel

GPIO  Mem

Bus Example – Add Timing

clock

μC  μP  DSP

channel

GPIO  Mem
Extract Arbitration

Key components are the interface method calls (IMC)

Why Interface Files?

// my_interface.h
class my_IF
 : public sc_interface
{
 // specify IF access function here
  virtual int my_write();
  virtual void set_priority();
  virtual my_status get_status();
 ...}

Key language element for communication abstraction

- Virtual declaration -> IMC declaration, but no implementation
  - Separates declaration from implementation
  - Separates communication from functionality
- Key source of information
  - Block -> specifies access methods
  - Channels -> required implementation
Architecture SOC Design

- Problem Statement
- What is Transaction Level?
- Transaction Level Modeling with SystemC
- Construct a multi-processor platform
  - Use interface method calls to connect block
  - Implement interface method calls
- Model with Designware SystemC library and TLM simulation in CoCentric System Studio
Add Master to a Bus-based System

Master: Header File

```c
#include <systemc.h>
#include "simple_bus_types.h"
#include "simple_bus_blocking_if.h"

struct Memory_access // Simple Bus Master Blocking
: public sc_module
{
  // processes
  void main_action();
  // ports
  sc_port<simple_bus_blocking_if, 1> bus_port;

  SC_HAS_PROCESS(Memory_access);
};
```
Simple Bus Interface File

```cpp
#include <systemc.h>
#include "simple_bus_types.h"

class simple_bus_blocking_if : public virtual sc_interface
{
public:
  virtual simple_bus_status burst_read(unsigned int unique_priority,
                                       int *data,
                                       unsigned int start_address,
                                       unsigned int length = 1,
                                       bool lock = false) = 0;

  virtual simple_bus_status burst_write(unsigned int unique_priority,
                                         int *data,
                                         unsigned int start_address,
                                         unsigned int length = 1,
                                         bool lock = false) = 0;
}; // end class simple_bus_blocking_if
```

Master: Source File

```cpp
#include "Memory_access.h"

void Memory_access::main_action()
{
  // storage capacity/burst length in words
  const unsigned int nymlength = 8UL;
  int nymdata[nymlength];
  unsigned int i;
  simple_bus_status status;

  while (true)
  {
    status = bus_port->burst_read(unique_priority,
                                   nymdata,
                                   n_address,
                                   nymlength,
                                   n_lock);
  
    // generate some data
    for (i = 0; i < nymlength; ++i)
    {
      nymdata[i] = i;
    }

    status = bus_port->burst_write(unique_priority,
                                   nymdata,
                                   n_address,
                                   nymlength,
                                   n_lock);

    // wait for a while to start next access, models latency
    wait(Timeout, 0uNS);
  }
}
```

Read - IMC

Write - IMC
Analyze Architecture

Configuration: memory address space, # memory wait statements
bus bandwidth, Master priority
Master latency....

Master: Constructor

```c
// constructor
Memory_access(sc_module_name name,
            unsigned int unique_priority,
            unsigned int address,
            bool lock,
            int timeout)
    : sc_module(name),
    _m_unique_priority(unique_priority),
    _m_address(address),
    _m_lock(lock),
    _m_timeout(timeout),
    bus_port("bus_port")
{
    // process declaration
    SC_THREAD(main_action);
}
private:
    unsigned int _m_unique_priority;
    unsigned int _m_address;
    bool _m_lock;
    int _m_timeout;
}; // end module Memory_access
```
Configure System

Memory_access = new Memory_access("Memory_access", 4, 0x4c, false, 300);

Configure System

// Number of read or write wait states
CCSS_PARAMETER(unsigned int, WaitStates);

// Master Latency
CCSS_PARAMETER(unsigned int, Master_Latency);
Run Architecture Exploration

arch_regressions.scf:

```
#define
set root entity SIM_NAME
set MASTER Memory_access
set MEMORY slow_mem
set BUS single_Doc
set ADDRESS_START [expr 0x90000000]
set BANK_LENGTH 16

# sweep parameters
set size_list E 2 4 8 10 20 30 50
set latency_list E 50 100 150 200 250
set wait_list F 0 1 2 3 4

# loops
foreach size_list [size_list]
foreach latency_list [latency_list]
foreach wait_list [wait_list]

set_value $root/MASTER/Master_latency latency_
set_value $root/MASTER/waitStates wait_
set_value $root/MEMORY/size mem_size

# Simulate
run_iteration
```

```
unix> Sim.x --control arch_regressions.scf
```

Analyze Results
Implementation Steps

1. Include interface header file
   
   ```c
   #include simple_bus_blocking_if.h
   ```

2. Specify port with interface type
   
   ```c
   sc_port<simple_bus_blocking_if> my_new_port
   ```

3. Check interface header file for IMCs

4. Use IMCs
   
   ```c
   my_new_port->burst_read (start_addr,&data,num_value, mode);
   ```
   (for processor integration:
   check Synopsys’ transaction level processor modeling guide)

5. Add parameters to configure system for exploration
   - Constructor
   - Parameters

6. Use Tcl scripts to run simulations with various configurations

7. Analyze architecture performance

DesignWare SystemC Library

![Diagram of DesignWare SystemC Library](image)
DesignWare AMBA AHB Components

- AHB bus with external arbiter and decoder
- Interconnection matrix for AMBA multilayer support
- AHB bus monitor
- Processors (ARM 926, ARM 946)
- Pin Level adaptors
- Example masters and slaves
- Example AHB platforms

DesignWare AMBA APB Components

- AHB-APB bridge with APB bus
- APB bus monitor
- Pin Level adaptors
- APB peripherals
  - Remap and pause controller
  - Interrupt controller
  - APB timer
- Example AHB/APB platforms
Summary – use IMCs

- Easy to connect blocks to the TLM bus infrastructures
- Efficient way to design complex systems
- Easy to use configuration mechanisms via constructor arguments and CCSS_PARAMETERS
- Comfortable configuration, simulation and analyze of your multi-processor architecture with simulation control files
- Use System Studio to interact with the executable and visualize results
- With System Studio information flows nicely from block developer and block user

Architecture SOC Design

- Problem Statement
- What is Transaction Level?
- Transaction Level Modeling with SystemC
- Construct a multi-processor platform
  - Use interface method calls to connect block
  - Implement interface method calls
- Model with Designware SystemC library and TLM simulation in CoCentric System Studio
Connect Slave Blocks

Tasks to implement slaves as interface method calls:

1. Define IMCs
2. Implement IMCs
   - fast mem
   - slow mem
   - slave pin adapter
3. Use IMCs in calling instance
   - Add slave port to bus (see master integration)
     > sc_port<memory_slave_if> new_slave_port;
   - Call IMC in the bus to access slave
     > read( &data, address);
Define Interface Method Calls

Memory Slave Example IMCs

- Read from memory -> read(data*, address)
- Write to memory -> write(data*, address)
- Build address range -> addr_range (start*, end*)

Code example interface file

```cpp
class memory_slave_if
    : public sc_interface
{
public:
    // Slave interface method calls declaration
    virtual status read(int *data, unsigned int address) = 0;
    virtual status write(int *data, unsigned int address) = 0;
    virtual void addr_range(unsigned int *start_ad, unsigned int *end_ad) = 0;
}; // end class memory_slave_if
```

Implement Interface Method Calls - fast mem (header file)

```cpp
class fast_mem
    : public memory_slave_if, public sc_module
{
public:
    SC_HAS_PROCESS(fast_mem)
    // constructor
    fast_mem(sc_module_name name_,
             unsigned int start_address,
             unsigned int end_address)
        : sc_module(name_), m_start_address(start_address),
         m_end_address(end_address)
    {
        sc_assert(m_start_address <= m_end_address);
        unsigned int size = (m_end_address - m_start_address + 1) / 4;
        MEM = new int [size];
        for (unsigned int i = 0; i < size; ++i)
            MEM[i] = 0;
    } // end class fast_mem
```
Implement Interface Method Calls
- fast mem (cont.)

```cpp
// destructor
~fast_mem();

// slave interface method calls
status read(int *data, unsigned int address);
status write(int *data, unsigned int address);
void add_range(unsigned int *start_a, unsigned int *end_a);

private:
int *MEM;
unsigned int m_start_address;
unsigned int m_end_address;
); // end module fast_mem
```

Implement Interface Method Calls
- fast mem (source file)

```cpp
inline status fast_mem::read(int *data, unsigned int address)
{
    *data = MEM[(address - m_start_address)/4];
    return SIMPLE_BUS_OK;
}

inline status fast_mem::write(int *data, unsigned int address)
{
    MEM[(address - m_start_address)/4] = *data;
    return SIMPLE_BUS_OK;
}

inline void fast_mem::range_add(unsigned int *start_a, unsigned int *end_a)
{
    *start_a = m_start_address;
    *end_a = m_end_address;
}

inline fast_mem::~fast_mem()
{
    delete [] MEM;
}
```
Implement Interface Method Calls
- slow mem

```
class slow_mem
    public memory_slave_if, public sc_module
    {
        public:
            // processes
            void wait_loop();
            // ports
            sc_in_clk clock
            SC_HAS_PROCESS(slow_mem);
            // constructor
            slow_mem(sc_module_name name_,
                     unsigned int nr_wait_states)
                 : sc_module(name_), m_start_address(start_address),
                  m_end_address(end_address), m_wait_count(-1)
            {
                SC_METHOD(wait_loop);
                dont_initialize();
                sensitive_pos << clock;
                SC_METHOD(read);
            }
    }
```

Implement Interface Method Calls
- slow mem (cont.)

```
// destructor
~slow_mem();

// slave interface method calls
status read(int *data, unsigned int address);
status write(int *data, unsigned int address);
void add_range(unsigned int *start_a, unsigned int *end_a);

private:
    int * MEM;
    unsigned int m_start_address;
    unsigned int m_end_address;
    unsigned int m_wait_count;
}
```

© 2003 Synopsys, Inc. (105)
Implement Interface Method Calls
– slow mem (cont.)

```cpp
inline void slow_mem::wait_loop()
{
    if (m_wait_count > 0) m_wait_count--;  
}

inline status slow_mem::read(int *data, unsigned int address)
{
    // accept a new call if m_wait_count < 0
    if (m_wait_count < 0)
    {
        m_wait_count = m_nr_wait_states;
        return SIMPLE_BUS_WAIT;
    }
    if (m_wait_count == 0)
    {
        *data = MEM[(address - m_start_address)/4];
        return SIMPLE_BUS_OK;
    }
    return SIMPLE_BUS_WAIT;
}
inline status slow_mem::write(int *data, unsigned int address) ...
inline void slow_mem::range_add(...
inline slow_mem::~fast_mem() ...
```

Return status SIMPLE_BUS_WAIT for n_wait_count cycles

Transaction Level Bus Models

- Synopsys’ Simple Bus
  - Example implementation as starting point
  - Generic bus protocol
  - Limited features, cycle accurate

- Synopsys’ DesignWare AMBA
  - AMBA bus protocol implementation
  - Cycle accurate
  - Pin level adapters, peripherals, ARM, APB and AHB

- OCP (Synopsys, Nokia, TI, Sonics)
  - Defines communication IMCs
  - Specifies levels of abstraction
Summary – Implement IMCs

• Why is the interface method call concept so cool
  ▪ parallels implementation of block and channels in a multi-processor systems
  ▪ Enhance interface functionality by adding functions will not break the current design
  ▪ Elegant refine of communication by adding more detailed timing
  ▪ Customization of channel accesses
  ▪ Compiler already identifies if wrong access methods used

Architecture SOC Design

• Problem Statement
• What is Transaction Level?
• Transaction Level Modeling with SystemC
• Construct a multi-processor platform
  ▪ Use interface method calls to connect block
  ▪ Implement interface method calls
• Model with Designware SystemC library and TLM simulation in CoCentric System Studio
Simulate with System Studio

- System time
- Access parameters
- Model library and graphical entry

System Level Debugging with System Studio

- Memory content
- Bus transactions: which master, which slave, which activity, at which point in time
Architecture Exploration

Packet loss ratio versus number of banks and routing latency

<table>
<thead>
<tr>
<th>latency (SC_NS)</th>
<th>10</th>
<th>8</th>
<th>6</th>
<th>4</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5000</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>6000</td>
<td>2</td>
<td>2</td>
<td>24</td>
<td>24</td>
<td>--</td>
</tr>
<tr>
<td>7000</td>
<td>12</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>8000</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

Generating realistic stimulus

*IP over ATM through AAL5-MPOA*

Generates IP datagrams sequences

Encapsulate in MPOA AAL5, generate ATM cells, insert idle cells
Understanding CPU power requirements, Guide decision for HW / SW partitioning

What about burst traffic conditions?
variable IP packet length?
Address look-up table searching?

TLM Simulations Become Mandatory

TLM Simulation ( AMBA bus bandwidth utilization per module )

- Simulation results of the worst-case downstream traffic
- More that 50 % of the bus bandwidth is available for upstream traffic

Optimization of the RAM-based buffer
Management may reduce the CPU traffic
Ram-based buffer management principle

SW simply compute data accumulation into buffer

TLM Simulation (Ram-based Buffer utilization)

- Architecture Issue: Collisions on Ethernet port #3 produce data accumulation into ram buffer
- When “max buffer size” is reached, data for this port are lost

The buffer management should make used of the space free in other packet buffers
TLM Simulation (buffer management & architecture refinement)

- **Architecture refinement #1**
  - The buffer management should share a pool of IP buffer among 4 Ethernet ports

- **Architecture refinement #2**
  - What must be done when pool is full?
    - discard IP packet?
    - Discard ATM cell?
  - What is the impact of losing data on upper protocol layers

SOHO router design: Reaching the performance

- The number of processor cycles is represented by the # of “loop” iterations

- A rough instruction cycle budget can be defined for each SW processing step:
  - CRC32, address route lookup search, AAL5 cell buffer parsing, pattern matching …

- TLM simulations can help to identify the first bottlenecks … Then for further details …

Execution of the SW on the virtual design

Prototype must now be performed
TLM modeling summary

- We have:
  - Built a high level reference model of the chip functionality made of pre-existing IP and new modules.
  - Used the same language (SystemC) regardless of HW or SW implementation
  - Generated real world stimuli to exercise the chip in critical modes of operations (TWB)
  - Observed and Corrected architectural bottlenecks by Measuring System performances
  - A virtual prototype of the design to be used for refining the HW / SW functional partitioning

Agenda

- 1:00 p.m. Introduction and Seminar Overview
- 1:10 p.m. SOC design challenge and SystemC methodology
- 1:40 p.m. Introduce SystemC language
- 2:30 p.m. Tea Break
- 3:00 p.m. The Concept to RTL flow for the SOHO router
  - Today's design – SOHO Router
  - Architecture SOC design
  - SW implementation & debug
  - Moving from Transaction Level Models to RTL
  - Reaching golden RTL
- 4:30 p.m. Demo: Multiprocessor ARM design for IP Router
- 5:00 p.m. Q&A and Wrap up
SOHO router: 
Creating a virtual prototype

1. Design blocks have a register/bit accurate interface with the SW
2. ARM 926 ISS is wrapped in a SystemC module
3. SW includes peripheral low level drivers and time-critical tasks (networking, data processing, interrupt service routines) SW application code linked in with the target processor startup code.
4. SW execution synchronized with test case controller

SOHO Router
Modeling the processor function

- Simple SW model,
- Functionality coded in C or C++
- SystemC interface to AMBA bus

- Same code can be used to run on ARM 926 when ISS replaced the simple processor model
Processor model
*Simplified SW functionality*

- Focus is not complete SW functionality but processing latency & bus utilization
- **Assumptions:**
  - SW processing is just a delay loop
  - Loop duration represents the number of SW instruction in the main task
  - SW keeps track of data accumulation into buffer
  - ATM buffer reads & IP buffer writes by CPU are done sequentially (not interleaved, with SW processing)
  - Simple buffer management is used

SW router specifications (limited functionality)

*AAL5 processing is made of 4 sub-states*
SW Router (limited functionality)

```c
VOID SW_ROUTER main( MAIN_ARGS )
{
    // Initialization: write first descriptor
    ... 
    while (true) {
        local_nIRQ = nIRQ.read(); // Read interrupt pin 
        if (local_nIRQ == false){ // Service Interrupts
            ... 
            if (intr_status[0] == 1){// Process ATM interrupt
                ok = busport.burst_write(...); // Write Buffer Descriptor
            }
            // Process Ethernet interrupts
            ...
            // Clear Interrupt bits
            ...
        } else { // Main task
            if (in_packet_cnt != 0){
                // Read cells from input buffer, do AAL5 reassembly
                ...
                // AAL5 processing done, write packet to output buffer
                ok = busport.burst_write(...);
            }
        } // Main task
    } // while (true)
}
```

Interrupt service routine takes priority over AAL5 reassembly

AAL5 main task is idle until number of ATM buffer /= 0

Integrate The ARM 926 Processor Model

- High Performance, High Accuracy behavioral model
  - Runs SW as the final core would
  - 100% transaction accuracy
  - Native model executes at >500kHz
- Tight integration to simulation environment
  - Single-process simulation (SystemC kernel + Platform models + Processor Model) => high speed
- Cycle-based for no scheduling restrictions
- Modeled as a bus master
  - Also has a clock port and asynchronous interrupt ports
ARM model Integration with SystemC

Complete ISS consist of ARM 926 Core model + a set of SystemC modules
**SW Debugging Using ARM Debugger**

![ARM SW Debugger](image)

**Direct link**

**Execution of The SOHO Virtual design prototype**

---

**SW modeling summary**

- SW functionality is modeled and simulated in SystemC prior to selecting the target processor.
- Virtual prototyping provides for a fast and accurate SW validation environment.
- The HW/SW glue logic is simulated with real hardware behavior.
- Resultant system is the development platform for software applications
  - Final memory map is available for SW development.
  - Validated startup code is linked to the application software and guaranteed to work the first time.
Agenda

- 1.00 p.m. Introduction and Seminar Overview
- 1:10 p.m. SOC design challenge and SystemC methodology
- 1:40 p.m. Introduce SystemC language
- 2:30 p.m. Tea Break
- 3:00 p.m. The Concept to RTL flow for the SOHO router
  - Today’s design – SOHO Router
  - Architecture SOC design
  - SW implementation & debug
  - Moving from Transaction Level Models to RTL
  - Reaching golden RTL
- 4:30 p.m. Demo: Multiprocessor ARM design for IP Router
- 5 p.m. Q&A and Wrap up

SOHO router: Hardware design: Block level

1. Successive replacement of TLM modules by RTL blocks
2. TLM platform is used as the test bench for module level testing
   - Adapters bridge from transaction to signal level
3. Assertions added to HDL, coverage measured, error injection
4. SW subset focusing at module level verification
Module level RTL design tasks

- Synthesize ATM block
- Replace Ethernet block by DW IP
- Provide transaction to signal adapters
  - For ATM block and Ethernet Block
- Perform Interface error injection
  - Multiple levels
- Add assertions for each RTL module
- Add coverage metrics
ATM block

Getting from TLM to RTL

- TLM refinement and synthesis

**TLM**

SystemC refinement

→

Synthesizable SystemC

Advantages:
- stay in same language & environment
- synthesize from golden model
- can compare TL versus RTL

Disadvantages:
- no formal equivalence checking

Synthesis:

SystemC Compiler

Output:
VHDL/Verilog netlist
and/or .db

Uses:
DC and PC engines

ATM Block

Refinement for synthesis

**ATM TC**

SystemC spec.

→

Synthesizable subset

Design structure
- Partition into blocks to be individually synthesized
- Design interfaces for communication

Manual step

Manually getting to RTL
- I/O protocol
- clock domains
- latency, throughput
- FSM & datapath for RTL

Behavioral SystemC
- Hardware datatypes
- Bit widths

SystemC Compiler

Hardware(gates)
**Ethernet Block**

*Getting from TLM to RTL*

- **Moving from TLM to Verilog**

  - TLM
    - SystemC
  - Created from scratch
  - Imported as RTL IP

  - Verilog
    - RTL
    - Synthesis:
      - Design Compiler or Physical Compiler

  - Verilog Gates

**Advantages:**
- known, supported, comfortable flow

**Disadvantages:**
- Re-coding is always prone to errors.
- Not feasible for large complex designs

---

**SOHO router**

*Ethernet TLM replaced by RTL IP*

- AMBA AHB DMA interface
  - Uses descriptor architecture for minimum CPU intervention
  - two dual-port FIFOs

- Media Independent Interface (MII)
- Compliant with IEEE 802.3
  - Supports Full and Half-duplex operations
  - Supports 10/100 Mbps data transfer rates

- Media Independent Interface (MII)
Verification requires simulation on TLM and RTL level

Adapters translate between different levels of abstraction over an interface

- SystemC TL to SystemC RTL
- SystemC TL to HDL

RTL requires co-simulation

TLM to RTL Adapter

```cpp
class Adapter
  : public sc_module, public bus_slave_if
{ ... }

bus_status Adapter::read(bus_DType *data, bus_AType address) {
  if (!m_selected) {
    m_selected = true;
    Read_o.write(true);
    Addr_o.write(address - _start_address);
    CS_o.write(true);
    return BUS_WAIT;
  } else {
    m_selected = false;
    *data = Data_i.read();
    CS_o.write(false);
    return BUS_OK;
  }
}
```

Inherits from the slave interface of the TLM bus

Implement the read method of the TLM bus

Interface to RTL world = read/write port values
SOHO router:  
*Fault Injection Approach*

1. Cell, packet, and interface errors are controlled by test case controller (SystemC)  
2. Cell and packet-based errors handled by Telecom workbench  
3. Interface signal-based fault injected by TLM adapter  
4. Bus monitor used for tracing bus events

---

**Error injection**  
*Interface signals type of errors*

- **53-byte standard ATM cell**
- **SoC asserted more than one cycle**
- **Incomplete cell**

---

© 2002 Synopsys, Inc.  Accelerating “Concept to RTL design Seminar”
SOHO Router

Adding assertions to the RTL (Verilog)

- Assertions typically define critical properties of the design:
  - E.g. bus protocols
- Assertions are declarative and more concise than HDL
  - Are used in simulation (always on)
  - Can be used in formal tools
    - Proof properties, generate stimulus
- Assertions can easily be reused from block level into chip level verification

Assertion based verification methodology

Block level

Stimulus

Filter

HDL & VERA simulation

Response

Constraint

Proofs & counter examples

input

output

input

output

OVA

RTL

Formal

Dynamic
What are OpenVera Assertions (OVA)?

- OVA describe sequences of events and lets you test for their occurrence
- Declarative form is more concise and readable than procedural code such as Verilog, VHDL, or VERA
  - Example coming in a few slides
- OVA is an effective and efficient way to write
  - Checkers, coverage statements
  - assertions for formal proofs
  - constraints for DUT

OVA’s simulate fast in VCS

---

**OVA Example:**

**AHB Arbiter**

```plaintext
Template definition

template check_bool (expr, msg="", clk=%default%): {
  clock clk {event cb_ev: expr;}
  assert cb:check(cb_ev, msg);
}
template forbid_tool (expr, msg="", clk=%default%): {
  clock clk {event cb_ev: expr;}
  assert cb:forbid(cb_ev, msg);
}

Template usage

module DW_arbiter_2t {
  check_bool(count(grant)==1, "OVA MUTEX FAILED", posedge clk);
}
```

*Only 1 source is selected. It should be: count(grant) <= 1*
OVA Example:  
**AHB Arbiter**

OVA flow in VCS:  
1. Instrument & Monitor OVA  
   - % vcs -ova_debug -ova_report <files.ova>  
2. Analyze OVA report  
   - % more simv.vdb/report/ova.report

```
OVA Example:  
AHB Arbiter

OVA flow in VCS:
1. Instrument & Monitor OVA  
   - % vcs -ova_debug -ova_report <files.ova>  
2. Analyze OVA report  
   - % more simv.vdb/report/ova.report

Ova [0]: "../src/test.ova", 12:  
srs_testbench_test_top.dut.ars_pin1.ars1.DW_ahb_1.U_arb.U_arb2t_ti1_cb:  
started at 25000 failed at 25000, "OVA MUTEX FAILED"  
Offending "count(grant)==1"  

Failed, because  
- count(grant) == 0
```

3. Debug error  
   - % ovadbp

---

OVA Example:  
**Debug OVA with Virsim**

[Image of Virsim debug interface]

- **Passed** at 75,000
- **Failed** at 25,000
Assertion Summary

- Assertions can incrementally be added to the design blocks
  - Assertions are automatically reused in chip level simulation
  - Can be turned on/off globally/selectively
- Assertions constantly monitor RTL behavior
  - Requires low overhead in simulation (VCS supports OVA natively)
- Can be used in formal tools
  - To proof properties
  - To generate effective stimulus

SOHO Router: Block level design

Measuring coverage on RTL

- Code coverage metrics (line, condition, FSM)
  - Built in VCS
- Functional coverage and dynamic feedback
  - Directly supported in VERA
- Assertion coverage (OVA)
  - Using coverage objects to collect data (with API so info can be used dynamically) + reporting
    - Persistent coverage database (regression)
    - Built into VERA
SOHO router
*Adding Code Coverage*

Coverage metrics flow in VCS:
1. Instrument coverage types
   - `% vcs -cm line+cond+fsm`
2. Monitor coverage types
   - `% simv -cm line+cond+fsm`
3. Generate/Analyze coverage report
   - `% cmView`
4. Test Grading
   - `% cmView`

**Code Coverage Results**
*cmView*

- **Lines that are not exercised**
- **GUI for modules selection**
- **Statistic regarding the block**
Functional coverage and assertion coverage

- Instrument design with assertions
  - Link back to spec.
- Define coverage objects to “tally” hit rate
- Create custom reports

Agenda

- 1.00 p.m. Introduction and Seminar Overview
- 1:10 p.m. Introduce SystemC language
- 1:50 p.m. SOC design challenge and SystemC methodology
- 2:30 p.m. Tea Break
- 3:00 p.m. The Concept to RTL flow for the SOHO router
  - Today’s design – SOHO Router
  - Architecture SOC design
  - SW implementation & debug
  - Moving from Transaction Level Models to RTL
  - Reaching golden RTL
- 4:30 p.m. Demo: Multiprocessor ARM design for IP Router
- 5 p.m. Q&A and Wrap up
SOHO router: 
**Chip level design**

1. Create top level RTL, connect to testbench through adapters
2. Complete top level with remaining (pre-verified) modules
3. Add remaining drivers.

**SOHO Router**

**Full chip verification**

- The Verification goal is to check chip level data paths integrity
  - Assure integrity of RTL for complete chip
  - Perform **limited** SW execution
    - Focus on reset issues
  - Perform limited data path simulation
    - Focus on corner cases
- Complete regression suite
  - Requires SystemC/VCS co-simulation
SOHO Router
Creating full RTL model

- Designware AMBA RTL bus IP replaces the transaction level models used so far
- ISS wrapped in SystemC is replaced by ISS wrapped in RTL wrapper, or full functional RTL model
- Add additional modules (from Designware) in design AND testbench:
  - USB controller
  - IP bus and peripherals

*Simulate and build regression suite*

---

Physical design
ARM-Synopsys Reference Methodology

[Diagram showing the process flow from Core Creation to Core Integration into SoC]

*White Paper, Implementation Guide and Scripts Available*
Agenda

- 1:00 p.m. Introduction and Seminar Overview
- 1:10 p.m. SOC design challenge and SystemC methodology
- 1:40 p.m. Introduce SystemC language
- 2:30 p.m. Tea Break
- 3:00 p.m. The Concept to RTL flow for the SOHO router
  - Today’s design – SOHO Router
  - Architecture SOC design
  - SW implementation & debug
  - Moving from Transaction Level Models to RTL
  - Reaching golden RTL
- 4:30 p.m. Demo: Multiprocessor ARM design for IP Router
- 5:00 p.m. Q&A and Wrap up

Concept to RTL Summary

Reach Architecture closure

Hardware dev.

Software dev.

Block level

Chip level
Synopsys’ and ARM’s Tool and IP support for Concept to RTL flow

Summary/conclusions

- SystemC and System Studio are very effective to create an executable golden architecture model
- This model can be used concurrently to drive the SW development and the HW development process
- The testbench created on TLM level can be reused for Block and Chip level
Thank You!