Using SystemC for SoC modelling and design

Marc Pauwels - Ates Berna
Fatma Özdemir - Yves Vanderperren

marc.pauwels@mie.alcatel.be

15.01.2002
Outline

- I. Introduction to SystemC
- II. The AME System Design Methodology
- III. The SystemC Language and its features in practice
- IV. The AME System Design Methodology in practice
- V. AME’s Experience with SystemC based design
- VI. Summary and Conclusions
Part I. Introduction to SystemC
Part I. Introduction to SystemC

- What is SystemC? Target and basic features
- OSCI: Open Source SystemC Initiative
- Roadmap and evolution
- The AME User perspective
- Summary and Conclusion
System design language for HW/SW co-design

- A library of C++ classes
  - Processes (for concurrency)
  - Clocks (for time)
  - Hardware (finite) data types [bit vectors, 4-valued logic, fixed-point types, arbitrary precision integers] and infinite data types
  - Waiting and watching (for reactivity)
  - Modules, ports, signals (for hierarchy)
  - Abstract ports and protocols (abstract communication)

- A light-weight simulation kernel

- Allows to make an “Executable Specification”
SystemC enables System design and verification

- System Level IP
- Soft IP
- Hard IP
- C-Compiler

(Courtesy: OSCI)
Levels of Abstraction:

- UTF
- TF
- BCA
- CA

SystemC

- Functional decomposition
- Untimed Functional
- Assign ‘execution time’
- Timed Functional
- HW / SW Partition
- Architectural mapping
- Refine communication

Design Exploration
- Performance Analysis
- HW/SW partitioning

Task Partitioning
- Abstr. RTOS
- BCA

Target RTOS/Core
- RTOS
- RTL

Software
- BCA
- Cycle Accurate

Hardware
- RTL
- Bus Cycle Accurate

(Courtesy: OSCI)
Gradual Refinement

Key of methodology is: design may be refined in gradual step-wise fashion, rather than one giant step. It is not “all or nothing”.

- UTF
- TF
- BCA
- CA

Details added to portions of the system.

(Courtesy: OSCI)
SystemC is “Open Source Initiative”
- Open Source License: code & user's Guide
- Language and Reference Simulator: www.systemC.org
- Platforms: Unix (gcc v2.95) - Linux - WinNT/2000

OSCI: became non-profit organization
- OSCI board and steering group
- OSCI working groups
  - language, reference implementation, dataflow, IP integration, verification
- User Discussion Forum: Q&A via www.systemC.org
- User Meetings
  - 4th ESCUG (Oct 5 2001 - Copenhagen)
  - www-ti.informatik.uni-tuebingen.de/~systemc
Open SystemC Initiative Steering Group

(Courtesy: OSCI)
♦ SystemC User Community: Alcatel amongst the early adopters

(Courtesy: OSCI)
Continued Strong SystemC Adoption: Big momentum...

SystemC v1.0/1.1beta Released

(Courtesy: OSCI)
History and evolution of SystemC

- SystemC v0.90 (Sep. 99)
- SystemC v1.0 (Apr. 00)
- SystemC v1.1 (Jun. 00)
- SystemC v1.2 (Feb. 01)
- SystemC v2.0 (Sep. 01)

Backwards Compatible

Roadmap (Evolution)

- 1997 DAC Paper
- Sep. 01
- SystemC v2.0
- SystemC v1.2
- April 00
- SystemC v1.1
- Jun. 00
- SystemC v1.0
- Apr. 00
- SystemC v0.90
- Sep. 99

- Abstract Protocols - Master Slave Ports
- Fixed Point Types
- RPC

Frontier Design

Scenery

CoWare

Synopsys
1.0 - Hardware Design Flow
  - RTL and Behavioral Hardware Modelling

1.x - Master-Slave Communication Library
  - Extension to system-level Modelling and different levels of abstraction
  - RPC-based untimed & timed functional modelling down to RTL for bus protocol based systems

2.0 - System Design Flow
  - General purpose communication and synchronization
  - Communication Refinement
  - Multiple, customizable models of computation
Foreseen for the future ...

♦ 2.X - Extensions to System Design Flow
  - Dynamic thread creation, fork / join
  - Interrupt / abort for behavioral hierarchy
  - Performance modelling support
  - Timing specification and constraints

♦ 3.X - Software Design Flow
  - Abstract RTOS modelling
  - Scheduler modelling

♦ 4.X - Analog / Mixed Signal Systems Modelling
AME User Perspective
(Used versions and aspects)

1.0 - Hardware Design Flow

1.x - Master-Slave Communication Library
- Extension to system-level Modelling and different levels of abstraction
- RPC-based untimed & timed functional modelling down to RTL for bus protocol based systems

2.0 - System Design Flow

2.X - Extensions to System Design Flow
- Interrupt / abort for behavioral hierarchy
- Performance modelling support
- Timing specification and constraints

3.X - Software Design Flow

SystemC v1.2β

interrupts
performance
timing
Firmware
AME User Perspective
(Design Flow)

- Used Levels of Abstraction:
  - TF
  - CA
Summary

From OSCI

- SystemC language & ref.simulator available
- Long-term Roadmap: continuity in versions

At Alcatel Microelectronics

- Early adoption of SystemC v1.1β since 2000
- Started with no other tools than reference simulator (but C++...
At Alcatel Microelectronics

- Language is not sufficient: one needs vision on Methodology
- Used v1.2β in real life SoC design project (OWL, since Jan 2001) from which Methodology was derived
- Target: Executable model of system under design as reference and baseline for design phase
- Tools supporting the design flow are yet to follow

SystemC and OSCI are the seed for:

- System Design Methodology
- Tools (simulation, co-simulation, synthesis, verification...)
- Standardization (IEEE, SpecC, VSIA, ... )
Part II.
Alcatel Microelectronics’
System Design Methodology

Ofdm
Wireless
Lan

The WLAN-OWL project
Part II. AME’s System Design Methodology

- Evolution from ASICs towards SoC
- SoC example: AME’s OWL project
- AME’s System-on-Chip Design Methodology
Evolution towards SoC
( In general ... )

♦ Evolution of IC (Manufacturing) Technology:
  - getting smaller
  - with possibility of larger functions on the die

♦ Evolution of System Designs
  - get System on ChipSet or ultimately single chip (SoC) for reasons of
    - cost saving
    - compactibility

♦ SoC are complex and contain many components: Hw,Fw,Sw...
  - all must inter-operate correctly and
  - must fit to Customer specifications
  - must be developed with faster-time-to-market

⇒ Need for System-on-Chip (SoC) Design Methodology
Evolution towards SoC

(In general ...)

(Source: IMEC)

http://www.imec.be/desics
Example: evolution within Alcatel Microelectronics:

- **1985**: ASIC Design
- **1995**: Start of ASSP

Start of *System design* (bottom-up)

- ADSL: going from “chips”... [ic’s]
  up to “boards” and... [demonstrators: hw + fw + bom]
  up to “PC-drivers”... [sw]

Start of *System conception* (top-down)

- WLAN-O WL Project of today
  - initially just ‘matlab’ coding
  - introduction of ‘SystemC’ and the ‘System Design Methodology’
High Speed Wireless LAN: Wireless networking at speeds > 10 Mbps

Applications
- home environment: wireless video and multi-media (TV-set, VCR, external video link), wireless LAN, VoIP, access to internet, ...
- business environment: wireless LAN, video-conferencing, multi-media,...

Units and communication
(DL, DiL, UL)

MAC-Frame(2ms):
The OWL Project
(The product in general ...)

- The ETSI HiperLAN-2 Standard
  - Network Layers
    
    \[ \text{E.g.: Ethernet} \quad \text{FireWire} \quad \ldots \]

    - Core Network
    - Core Network
    - Core Network

    CL - Specific Part
    
    Network Convergence Layer
    CL - Common Part

    Data Link Control Layer
    [ MAC - DLC ]

    Physical Layer
    [ PHY ]

  - Different flavours
    
    - Business-environment
    - Home-environment

    ➔ Hardware
    ➔ Firmware
    ➔ Software
The OWL Project
(AME’s view on the product ...)

♦ AME’s HiperLAN-2 SoC (The OWL Chipset)

<table>
<thead>
<tr>
<th>Analog</th>
<th>Digital</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF Layer</td>
<td>PHY Layer</td>
</tr>
<tr>
<td>BaseBand</td>
<td>MAC / DLC Layer</td>
</tr>
<tr>
<td>RF frontend</td>
<td>CL Layer</td>
</tr>
<tr>
<td></td>
<td>Transmit Data-Path &amp;</td>
</tr>
<tr>
<td></td>
<td>Local Controller</td>
</tr>
<tr>
<td></td>
<td>Receive Data-Path &amp;</td>
</tr>
<tr>
<td></td>
<td>Local Controller</td>
</tr>
<tr>
<td></td>
<td>Hardware</td>
</tr>
<tr>
<td></td>
<td>FirmWare/SoftWare</td>
</tr>
<tr>
<td></td>
<td>SDRAM (Fw/Sw)</td>
</tr>
<tr>
<td></td>
<td>(E)PROM (Fw/Sw)</td>
</tr>
</tbody>
</table>

Application Software

PCB Board and Components
Part II. AME’s System Design Methodology

- Evolution from ASICs towards SoC
- SoC example: AME’s OWL project
- AME’s System-on-Chip Design Methodology
  - Methods and tools: merging HW and SW worlds...
  - Iterative and model-based design
  - SystemC based executable model
  - SystemC based testbench reuse
  - SystemC reference for Hw and Fw/Sw design (Co-simulation)
SoC Design Methodology
( Methods and tools ... )

System Design
- Iterative Design
- Matlab modelling
- SystemC modelling

Sw Design
- Iterative Design
- Requir. capture
- Use Cases
- Modelling

Hw Design
- Classical Design
  - Vhdl Synthesis
  - Simulation
  - P&R
- Re-use to reduce complexity
  - uCore platform
  - IP-block re-use

System Validation & Qualification
- Co-simulation
- Sc/Iss/Vhdl
OWL Case: R&D Trajectory is ITERATIVE & MODEL based

- Re-use of SW methodology coming from RUP in project context
- Main target: reduce Risks in order to get ASAP Working product
- Early feedback from Executable SystemC model in First iteration
- Models are not “throw-away” but re-used throughout iterations
- SystemC is baseline for next iterations: testbench reusage
- keep models well-documented, high-quality and consistent!

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>It.0</th>
<th>It.1</th>
<th>It.2</th>
<th>It.3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>matlab</td>
<td>systemC</td>
<td>VHDL\ISS</td>
<td>FPGA</td>
<td>Silicon</td>
</tr>
</tbody>
</table>
OWL - Iteration 0: Big SystemC modelling effort

- From Matlab [algorithms]
- To SystemC [behaviour, system/hw/fw, architecture]

Objectives of the SystemC model

- The model is the executable specification
- The model is a tool during iteration 0 (definition phase) to:
  - verify the over-all system behaviour and architecture
  - detail the architecture for critical blocks
  - do finite-precision design
  - generate test-benches
- The model is a reference for the next iterations:
  - test benches are the starting point for all other test benches (RT-level,...) and system test plans (Lab qualification)
  - The model is a tool during debugging to analyze “where it goes wrong”
AME’s HiperLAN-2 SoC (The OWL Chipset)
OWL HW and FW partitioning
SoC Design Methodology
( OWL SystemC modelling ... )

- SystemC model of “Hw + Fw/Sw + System”

- Spec to start Hw-Design and Fw-Design ...
Used Levels of Abstraction:

- TF
- CA
OWL - Iteration 1: start of “Design”
- Start of HW design (vhdl)
- Start of FW design (C on ARM-processor platform)

SystemC as Specification and Golden Reference

Co-simulation as validation
- SystemC co-simulation with ISS TARGET: FW validation
- SystemC co-simulation with vhdl TARGET: HW validation
SystemC model as reference for "Hw (Vhdl) design"

SystemC and Vhdl Co-simulation
SoC Design Methodology
(OWL design and co-simulation...)

- SystemC model as reference for "FW (C-code) design"

- SystemC and ISS Co-simulation
SoC Design Methodology
(OWL design and co-simulation ...)

- SystemC and vhdl and ISS Co-simulation
Cf. Part-IV : The Complex-MPY section as Tutorial example
Summary and Conclusion

- **SoC Design Methodology**
  - Merging HW and SW worlds...
    - From conception till Lab-qualification
  - Iterative
  - Model based
  - Derived from OWL: a real-life project

- **SystemC model in Iteration-0**
  - Executable spec (with early feedback)
  - Baseline for next Iterations (HW and FW design)
  - Co-simulation with re-use of test-benches
Part III.
SystemC language in practice
Part III. SystemC Language in practice

♦ Context
  - SystemC from AME-user perspective
  - SystemC is extendable (library of C++ classes)

♦ SystemC as from the web

♦ Alcatel Microelectronics extensions

♦ Summary
Part III. SystemC Language in practice

♦ Context

♦ SystemC as from the web
  - System design
  - Modules and hierarchy
  - Processes and concurrency
  - Data types

♦ Alcatel Microelectronics extensions
  - The C++ behind SystemC
  - Matlab front-end
  - Fixed and floating point models

♦ Summary
Abstract view on a system:

- SystemC adds specific features on top of C++

- behaviour (modules/processes) and communication (ports/signals) are split
- concurrent and sequential behaviour
- different time-models (abstraction levels)
- mapped on single thread of execution
- implementation specific features (e.g. fixed-point data-types)
SystemC design flow:
- Procedural language with path towards implementation
- Executable Specification
- 4 Levels of Abstraction:
  - UTF: System level
  - TF: System level
  - BCA: Behavioural
  - CA: RTL
Gradual Refinement

- Key of methodology is: design may be refined in gradual step-wise fashion, rather than one giant step. It is not “all or nothing”.

- UTF
- TF
- BCA
- CA

Details gradually added to portions of the system.
Part III. SystemC Language in practice

♦ Context

♦ SystemC as from the web
  • System design
  • **Modules and hierarchy**
  • Processes and concurrency
  • Data types

♦ Alcatel Microelectronics extensions
  • The C++ behind SystemC
  • Matlab front-end
  • Fixed and floating point models

♦ Summary
Modules

- Module is a structural entity
  - helps to split complex designs
- Module can contain
  - ports
  - processes
  - user defined C++ functions
  - internal member data
  - constructor
  - modules and signals

Note: valid for
- Hardware
- Firmware/Software
- System

Accumulator

```
SC_MODULE(accumulator) {
    // ports (input)
    sc_in<int>  In;
    sc_in<bool> Rst;
    sc_in_clk   Clk;
    // ports (output)
    sc_out<int> Out;
    // local member functions
    void accumulate();
    void display(ostream& = cout);
    // local member data
    int Acc_
    // constructor
    SC_CTOR(accumulator) {
        SC_METHOD(accumulate);
        sensitive_pos << Clk;
        Acc_ = 0;
    }
};
```

.h file
Module Ports

- Pass data between module-boundary and internal module-processes
- Direction
  - in : read
  - out : write
  - inout : read & write
- Type
  - <int>, <bool>, etc.
  - special : clock

```c
SC_MODULE(accumulator) {
  // ports (input)
  sc_in<int> In;
  sc_in<bool> Rst;
  sc_in_clk Clk;
  // ports (output)
  sc_out<int> Out;
  // local member functions
  void accumulate();
  void display(ostream& = cout);
  // local member data
  int Acc_;
  // constructor
  SC_CTOR(accumulator) {
    SC_METHOD(accumulate);
    sensitive_pos << Clk;
    Acc_ = 0;
  }
};
```
Module Member functions

- Processes
  - functions registered with the SystemC kernel (see next)
- User defined functions
  - can be called within processes

```cpp
SC_MODULE(accumulator) {
  // ports (input)
  sc_in<int> In;
  sc_in<bool> Rst;
  sc_in_clk Clk;
  // ports (output)
  sc_out<int> Out;
  // local member functions
  void accumulate();
  void display(ostream& = cout);
  // local member data
  int Acc_;  
  // constructor
  SC_CTOR(accumulator) {
    SC_METHOD(accumulate);
    sensitive_pos << Clk;
    Acc_ = 0;
  }
}

void accumulator::accumulate()
{
  if (Rst.read())
    Acc_ = 0;
  else
    Acc_ += In.read();
  display();
  Out.write(Acc_);
}

void accumulator::display(ostream& os)
{
  os << "Acc_ = " << Acc_ << endl;
}
```
Module Member data
- to keep local state variable
- not to be used for communication between processes

Constructor
- registers processes with the SystemC kernel
- initializes whatever is needed (local member data)
- to create and initialize an instance of a sub-module (connections, instance)

```c++
SC_MODULE(accumulator) {
  // ports (input)
  sc_in<int> In;
  sc_in<bool> Rst;
  sc_in_clk Clk;
  // ports (output)
  sc_out<int> Out;
  // local member functions
  void accumulate();
  void display(ostream& = cout);
  // local member data
  int Acc_;
  // constructor
  SC_CTOR(accumulator) {
    SC_METHOD(accumulate);
    sensitive_pos << Clk;
    Acc_ = 0 ;
  }
};
```
Modules

- Hierarchy of Modules
  - Module instances
  - Signals
    - carry data between modules and between processes

```
SC_MODULE(top) {
  // ports
  sc_in<int> In;
  ...

  // module instances
  accumulator A;
  comparator C;

  // signals
  sc_signal<int> Intern;

  // constructor
  SC_CTOR(top): A("A"), C("C") {
    A.In(In);
    A.Out(Intern);
    ...
  }
};
```

connections
submodules
submodule name
internal signal

Hierachy of Modules

- Module instances
- Signals
  - carry data between modules and between processes

Diagram showing submodules and connections.
Top level “sc_main()”
- instances of modules
- connectivity at top-level
- clock object
Part III. SystemC Language in practice

- Context
- SystemC as from the web
  - System design
  - Modules and hierarchy
  - Processes and concurrency
  - Data types
- Alcatel Microelectronics extensions
  - The C++ behind SystemC
  - Matlab front-end
  - Fixed and floating point models
- Summary
Processes

- Basic unit of execution
- Contained inside modules
- Functions with a specific type
  - method: \texttt{SC\_METHOD(name)}
  - thread: \texttt{SC\_THREAD(name)}
  - clocked thread: \texttt{SC\_CTHREAD(name,clock)}
  - RPC
- Not hierarchical
- communication done via signals, module ports
- SystemC takes care of scheduling the concurrent processes
  - sensitivity list
  - event on a signal in the list triggers the process

\[ \Rightarrow \text{SystemC = Language (C++ Library) + Scheduler} \]
Method

- entire process executed once when a signal in its sensitivity list changes
- can not be suspended
- local variables loose their values between successive calls
- most similar to a usual C++ function
- fastest

void comparator::compare() {
  if (Val.read() > Th.read())
    Result.write(true);
  else
    Result.write(false);
}

SC_MODULE(comparator) {
  // ports
  sc_in<int> Val;
  sc_in<int> Th;
  sc_out<bool> Result;
  // processes
  void compare();
  // constructor
  SC_CTOR(comparator)
  SC_METHOD(compare);
  sensitive << Val << Th;
...}
Thread

- infinite loop limited by event boundaries: execution suspended by a wait() statement(s)
- (re-)activated when any of the signals in the sensitivity list changes
- local variables are saved (similar to static variables in C++ functions)
- slower than method with module data members to store the ‘state’

```cpp
void decision::fsm() {
    int a, b;    // local variables
    state_ = ...;  // initial state

    while(true) {
        wait();
        a = a_in.read();
        b = b_in.read();
        if (...) 
            state_ = ...;
        elseif (...) 
            state_ = ...;
        else 
            state_ = ...;
    }
}
```

```cpp
SC_MODULE(decision) {
    // ports
    sc_in<bool> Sensor_in;
    sc_in<int>  a_in;
    sc_in<int>  b_in;

    // processes
    void fsm();
    // constructor
    SC_CTOR(decision) {
        SC_THREAD(fsm);
        sensitive << Sensor_in;
    }
};
```
Clocked thread

- particular thread:
  - infinite loop,
  - sensitive only to one edge of one clock
- suspended by
  - wait() or
  - wait(int) or
  - wait_until()
- only synchronous systems
- slowest
Remote procedure call (RPC)
- abstract communication and execution semantics for functional level
- master/slave ports and processes
- RPC chain

```
SC_MODULE(M1) {
    sc_outmaster<int> Out;
    ...
    void process1();
    SC_CTOR(M1) {
        SC_METHOD(process1);
        sensitive << ...;
    }
};

SC_MODULE(M2) {
    sc_inslave<int> In;
    ...
    void process2();
    SC_CTOR(M2) {
        SC_SLAVE(process2, In);
    }
};

SC_MODULE(top) {
    sc_link_mp<int> link;
    ...
};
```
Remote procedure call (RPC)

- Equivalent to function call but without function pointer
- Structure is key for re-use (split behaviour of modules)
- Concurrent RPC chains

![Diagram showing concurrent and slave processes with RPC chains between processes A1, A2, B1, B2, and B3.](image)
Remote procedure call (RPC)

- Equivalent to function call but without function pointer
- Structure is key for re-use (split behaviour of modules)
- Concurrent RPC chains

Diagram:
- Concurrent processes
- Slave processes
- Producer() → Write() ; Read() → Consumer()
Part III. SystemC Language in practice

♦ Context

♦ SystemC as from the web
  ● System design
  ● Modules and hierarchy
  ● Processes and concurrency
  ● Data types

♦ Alcatel Microelectronics extensions
  ● The C++ behind SystemC
  ● Matlab front-end
  ● Fixed and floating point models

♦ Summary
Data types

- **C++ data types** (faster)
  - int, double, etc...

- **SystemC data types** (slower, but ... link to implementation)
  - scalar types
    - sc_bit \{0,1\} - sc_logic \{0,1,X,Z\}
  - bit and logic vector types
    - sc_bv<n> - sc_lv<n>
  - integer types
    - sc_int - sc_uint : 64 or 32-bit
    - sc_bigint - sc_biguint : > 64-bit
  - fixed point types
    - sc_fixed - sc_ufixed - sc_fix - sc_ufix

- **User defined data types**
  - e.g. Symbol64C = { 64 complex samples of sc_fixed }
Fixed point types

Parameters

\[ \text{sc\_fixed ( } w\ell \text{, } i\ell \text{, } q\_mode \text{, } o\_mode \text{ [\text{, } n\_bits]} \text{ ) } \text{sig} ; \]

Example

\[ \text{sc\_fixed ( 3, 2, SC\_TRN, SC\_WRAP ) } \text{Cnt} ; \]

<table>
<thead>
<tr>
<th>Cnt =</th>
<th>CntF</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1,5 = &quot;01.1&quot;</td>
<td>1,500</td>
</tr>
<tr>
<td>-0,5 = &quot;11.1&quot;</td>
<td>-0,500</td>
</tr>
<tr>
<td>+0,5 = &quot;00.1&quot;</td>
<td>0,832</td>
</tr>
<tr>
<td>+1,5 = &quot;01.1&quot;</td>
<td>1,500</td>
</tr>
</tbody>
</table>

\[ \text{Cnt = Cnt + Cnt} \]

-2,0 = "10.0"  "01.1"+"00.1"
Part III. SystemC Language in practice

♦ Context

♦ SystemC as from the web
  - System design
  - Modules and hierarchy
  - Processes and concurrency
  - Data types

♦ Alcatel Microelectronics extensions
  - The C++ behind SystemC
  - Matlab front-end
  - Fixed and floating point models

♦ Summary
SystemC is based on C++

- Standard Template Library (vector, map, strings, etc.)
- streams
- templates
- constructors and destructors
- inheritance and composition
- function and operator overloading
- exception handling
- C++ data type: bool
- casting operators
- ...

Easy to add features to SystemC
Example 1: template modules

```cpp
// template module: definition of a mux with any desirable type of signal
template <class SignalType> class mux: sc_module {

    // inputs
    sc_in<SignalType> Input1;
    sc_in<SignalType> Input2;
    sc_in<bool> Selector;

    // outputs
    sc_out<SignalType> Output;

    void process() {
        if (Selector.read())
            Output = Input2.read();
        else
            Output = Input1.read();
    }

    SC_CTOR(mux) {
        SC_METHOD(process);
        sensitive << Input1 << Input2 << Selector;
    }
};
```

```
mux<int> MuxInstance;
```
Example 2: inheritance

```cpp
SC_MODULE(ResetFF) : public FF {
    // ports
    sc_in<bool> Rst;

    // process
    void process_reset() {
        if (Rst.read())
            Q.write(0);
    }

    // constructor
    SC_CTOR(ResetFF) : FF("FF") {
        SC_METHOD(process_reset);
        sensitive << Rst;
    }
};
```

SC_MODULE(FF) {
    // ports
    sc_in<int> D;
    sc_in_clk clk;
    sc_out<int> Q;

    // process
    void process() {
        if (clk.read())
            Q.write(D.read());
    }

    // constructor
    SC_CTOR(FF) {
        SC_METHOD(process);
        sensitive_pos << clk;
    }
};
Example 3: destructor

- useful in testbench
  - dump simulation results
- useful for high-level models
  - dynamic memory allocation
Part III. SystemC Language in practice

- Context

- SystemC as from the web
  - System design
  - Modules and hierarchy
  - Processes and concurrency
  - Data types

- Alcatel Microelectronics extensions
  - The C++ behind SystemC
  - Matlab front-end
  - Fixed and floating point models

- Summary
Matlab front-end

Facts:
- Matlab is used for algorithmic development
- SystemC from the web has no GUI

Use Matlab to create:
- as main goal: **Seamless** design development from Matlab (system:algorithmic level) to SystemC (system:architectural level)
- as side-effect: a **mathematical & graphical analysis** environment in SystemC

→ Use Matlab API-functions in SystemC testbenches
Matlab front-end

- from Matlab (system:algorithmic level) to SystemC (system:architectural level)
Matlab front-end

- Matlab used as mathematical & graphical analysis environment
Example:
Part III. SystemC Language in practice

♦ Context

♦ SystemC as from the web
  - System design
  - Modules and hierarchy
  - Processes and concurrency
  - Data types

♦ Alcatel Microelectronics extensions
  - The C++ behind SystemC
  - Matlab front-end
  - Fixed and floating point models

♦ Summary
Fixed and floating point

- Features of fixed point representation:
  - numerical range
  - required precision
  - for operations: quantization and overflow behaviour

A common representation of a binary fixed-point number (either signed or unsigned) is

<table>
<thead>
<tr>
<th>$b_{ws-1}$</th>
<th>$b_{ws-2}$</th>
<th>...</th>
<th>$b_5$</th>
<th>$b_4$</th>
<th>$b_3$</th>
<th>$b_2$</th>
<th>$b_1$</th>
<th>$b_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MSB

radix point

LSB

where:

- $b_j$ are the binary digits (bits).
- The size of the word in bits is given by $ws$.
- The most significant bit (MSB) is the leftmost bit, represented by location $b_{ws-1}$.
- The least significant bit (LSB) is the rightmost bit, represented by location $b_0$.
- The radix (binary) point is shown four places to the left.
Fixed and floating point

**Features of fixed point representation:**
- **slope/bias scaling:** fixed and floating point values are linked via a scaling factor and a bias

A fixed point number can be represented by a general slope/bias encoding scheme

\[ V = \tilde{V} = SQ + B \]

where:
- \( V \) is an arbitrarily precise real-world value.
- \( \tilde{V} \) is the approximate real-world value.
- \( Q \) is an integer that encodes \( \tilde{V} \).
- \( S = F \cdot 2^E \) is the slope.
- \( B \) is the bias.

The slope is partitioned into two components:
- \( 2^E \) specifies the radix point. \( E \) is the fixed power of two exponent.
- \( F \) is the fractional slope. It is normalized such that \( 1 \leq F < 2 \).

**Note:** \( S \) and \( B \) are constants and do not show up in the computer hardware directly — only the quantization value \( Q \) is stored in computer memory.
Example:
- sensor measuring water temperature
- range is limited: 0 to 100 °C
- bias is 0
- to be mapped onto 8 bits unsigned
  - 1 bit ⇔ 1 °C leads to a waste of precision
  - 1 bit ⇔ 100/255 °C is better

Fixed and floating point

<table>
<thead>
<tr>
<th>fixed point value (bits)</th>
<th>fl. point value (degrees)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>0°C</td>
</tr>
<tr>
<td>00000001</td>
<td>1°C</td>
</tr>
<tr>
<td>00000010</td>
<td>2°C</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>01100100</td>
<td>100°C</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>11111111</td>
<td>imposs</td>
</tr>
</tbody>
</table>
Fixed and floating point

- SystemC enables fixed point or floating point models
Fixed and floating point

Example:

```
SC_MODULE(accumulator) {
    // input ports
    sc_in<int> In;
    sc_in<bool> Rst;
    sc_in_clk Clk;

    // output ports
    sc_out<int> Out;

    // processes
    void process();

    // constructor
    SC_CTOR(accumulator) {
        SC_METHOD(process);
        sensitive_pos << Clk;
    }

    // internal data member
    int Acc_;}
```

```
SC_MODULE(accumulator) {
    // input ports
    sc_in<sc_fixed<8,1>> In;
    sc_in<bool> Rst;
    sc_in_clk Clk;

    // output ports
    sc_out<sc_fixed<8,1>> Out;

    // processes
    void process();

    // constructor
    SC_CTOR(accumulator) {
        SC_METHOD(process);
        sensitive_pos << Clk;
    }

    // internal data member
    sc_fixed<8,1> Acc_;}
```
Our goal: have single model which can easily simulate both precisions

Easy solution: pre-processor
Fixed and floating point

Easy solution: pre-processor

- either floating point or fixed point;
  selection by means of preprocessor

```c
SC_MODULE(SR_shift_reg) {
    // inputs
    sc_in<complex<D_SHIFT_REG> > Sample_In;

    // outputs
    sc_out<vector<complex<D_SHIFT_REG>> > SamplesForSR_1_Out;
    sc_out<vector<complex<D_SHIFT_REG>> > SamplesForSR_2_Out;
    ...
};
```

But ...

- and but ...

AME has a proprietary solution:
extend systemC with new data type

- if FINITE
  `cout << "counter = " << Counter_.to_string(SC_BIN) << endl;`
- else
  `cout << "counter = " << Counter_ << endl;`
- endif
AME solution: new data type
- carrying BOTH floating and fixed point value, and the associated slope
- conditions evaluated using the fixed or the floating point value, depending on a precision mode flag
Fixed and floating point

Definition

<table>
<thead>
<tr>
<th>fx_double</th>
</tr>
</thead>
<tbody>
<tr>
<td>floating point value</td>
</tr>
<tr>
<td>scale</td>
</tr>
<tr>
<td>fixed point value</td>
</tr>
<tr>
<td>fixed point parameters</td>
</tr>
<tr>
<td>precision mode</td>
</tr>
</tbody>
</table>

Example of arithmetic operation

\[
\begin{align*}
25.36 \ [°C] & \left(\frac{100}{255} \ [°C/bit]\right) \ b#01000000 \ (⇔ 25.1 \ [°C]) \\
<8,8,SC\_TRN,SC\_WRAP> & \text{Floating mode} \\
+ & \\
45 \ [°C] & \left(\frac{100}{255} \ [°C/bit]\right) \ b#01110010 \ (⇔ 44.7 \ [°C]) \\
<8,8,SC\_TRN,SC\_WRAP> & \text{Floating mode} \\
= & \\
70.36 \ [°C] & \left(\frac{100}{255} \ [°C/bit]\right) \ b#10110011 \\
<8,8,SC\_TRN,SC\_WRAP> & \text{Floating mode} \\
\end{align*}
\]

note: \( 70.36 \ [°C] / \left(\frac{100}{255} \ [°C/bit]\right) = b#10110011 \)
Fixed and floating point

Example of relational operation

<table>
<thead>
<tr>
<th>25.36 °C</th>
<th>100/255 °C/bit</th>
<th>b#01000000 (⇔ 25.1 °C)</th>
<th>&lt;8,8,SC_TRN,SC_WRAP&gt;</th>
<th>Floating point mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.2 °C</td>
<td>100/255 °C/bit</td>
<td>b#01000000 (⇔ 25.1 °C)</td>
<td>&lt;8,8,SC_TRN,SC_WRAP&gt;</td>
<td>Floating point mode</td>
</tr>
</tbody>
</table>

> Yes

<table>
<thead>
<tr>
<th>25.36 °C</th>
<th>100/255 °C/bit</th>
<th>b#01000000 (⇔ 25.1 °C)</th>
<th>&lt;8,8,SC_TRN,SC_WRAP&gt;</th>
<th>Fixed point mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.2 °C</td>
<td>100/255 °C/bit</td>
<td>b#01000000 (⇔ 25.1 °C)</td>
<td>&lt;8,8,SC_TRN,SC_WRAP&gt;</td>
<td>Fixed point mode</td>
</tr>
</tbody>
</table>

> No
Fixed and floating point

Example:

```cpp
SC_MODULE(SR_shift_reg) {
    // inputs
    sc_in<complex<class fx_double>> Sample_In;

    // outputs
    sc_out<vector<complex<class fx_double>> > > SamplesForSR_1_Out;
    sc_out<vector<complex<class fx_double>> > > SamplesForSR_2_Out;
    ...
};
```

both floating point and fixed point

precision independent operations

```cpp
cout << "counter = " << Counter_ << endl;
```
Pros of AME’s solution:

- Single executable model for both precisions
- Dynamic and automatically computed slopes (e.g., multiplication: slopes are multiplied)
- Strong type checking (e.g., addition: forbidden to have a scaling factor mismatch)
- Single data type → single operations for fixed and floating point
- No discrepancy in code while using debugger (since pre-processor is not used)
- Clean module interfaces, all based on the AME proprietary type
- Single function in the testbench dumping any received signal
Summary

- **SystemC as from the web**
  - has necessary features to describe Systems (Hw, Fw, Sw)
  - procedural language with path to implementation
  - different levels of abstraction
  - together with the language also reference simulator
  - C++ based and therefore easily extendable

- **Alcatel Microelectronics extensions**
  - The C++ behind SystemC
  - Matlab front-end
  - Fixed and floating point models
Part IV.
AME’s System Design Methodology in Practice
Part IV. AME’s System Design Methodology in Practice

- Simplified section tutorial example
  - The complex multiplier example

- SystemC executable spec: modelling and simulation

- SystemC co-simulation

- Summary
Tutorial Example
The Complex MPY Section

Complex Multiplier

Master/Slave Link

Coeff

DataPath

Result

FCoeff

LCtrl

Processor

Operational FW & TestBench
The Complex MPY Section

Operational FW & TestBench

SC_MODULE(CMPY_LCtrl) {
    // ports
    sc_inmaster<bool> UpdateConfig;
    sc_outmaster<bool> ConfigUpdated;
    sc_outmaster<bool> Update;
    sc_inmaster<bool> notEmpty, notFull;
    sc_outmaster<bool> Enable;
    ...
    // local member data
    Symbol64C coef_;   
    // configuration access via pointer
    config_CMPY& configClass_;
    ...
    void updCoef();   // process
    void multiply();  // process

    SC_CTOR(CMPY_LCtrl) {
        SC_THREAD(control);
        sensitive_pos << CLK;
    }
};

SC_MODULE(CMPY_DPath) {
    // ports
    sc_inmaster<Symbol64C> D_In;
    sc_outmaster<Symbol64C> D_Out;
    sc_inslave<bool> Enable;
    sc_inslave<bool> Update;
    ...
    // local member data
    Symbol64C coef_;   
    // configuration access via pointer
    config_CMPY& configClass_;
    ...
    void updCoef();   // process
    void multiply();  // process

    SC_CTOR(CMPY_DPath) {
        SC_SLAVE(updCoef, Update);
        SC_SLAVE(multiply, Enable);
    }
};

.h file
Part IV. AME’s System Design Methodology in Practice

- Simplified section tutorial example
- **SystemC executable spec: modelling and simulation**
  - Top Level Integration
  - Master/Slave Library
  - Documented methodology
  - Testbench Strategy
  - Different levels of abstraction
- **SystemC co-simulation**
- Summary
The Complex-MPY as one of the sections at top level

**Complex Multiplier Example**

- Master/Slave Link
- DataPath
- LCtrl
- UpdateConfig
- ConfigUpdated
- Through Pointers
- Parameter Passing through Pointers (invisible at the block boundaries)

**Processor**
The communication is always with M/S ports.

The communication is based on a configuration which is a C++ class.

FW accesses HW using C++ pointers. Configuration access is based on C++ pointers.
- Soft reset is a command sent to HW by FW and has the highest priority among other commands.

- “watching” mechanism can also be used for IRQ modelling, FW to FW and/or HW to FW.
  - Check first if your cosim tool supports this feature!
Integration Methodology
M/S Ports-1

- HW-FW interface at the top-level
  - Always M/S communication is used

**Advantage!**

COMMUNICATION IS SEPARATED FROM BEHAVIOR

- Advantage of using different abstraction levels
  - All abstraction levels communicate using M/S communication
Integration Methodology
M/S Ports-2

- Well suited to FW-FW, HW-FW and HW-HW interfaces
- Facilitates the communication between modules. Hides unnecessary implementation details.
  - Allows quick model development
  - Trade-off analysis for HW-FW partitioning
  - Resource allocation decisions
Integration Methodology
M/S Ports- 3

- Ability to refine functional communication at the later stages of the design to BCA
- Benefits of separation of communication from behaviour
  - Embedding of IP blocks
  - Refining communication
- Only “sc_link_mp” is supported by our CoSim tool
SC_MODULE(CMPY_LCtrl) {
    // ...
}
SC_MODULE(CMPY_DPath) {
    // ...
}
SC_MODULE(CMPY) {
    // ports
    sc_inmaster <bool> Full_In;
    sc_inmaster <bool> Empty_In;
    sc_inmaster <Symbol64C> Mult_In;
    sc_outmaster <Symbol64C> Mult_Out;
    sc_inmaster <bool> UpdateConfig_In;
    sc_outmaster <bool> ConfigUpdated_Out;
    // ...
}
SystemC Executable Spec
Documented Methodology

- **SystemC Coding Style**
  - A design guidelines and rules document to be used by the SystemC teams.
    Naming conventions, directory structure, code layout, etc.

- **Block Diagram Conventions**
  - Standardizes the drawing conventions.

- **SystemC Coding Generics**
  - Includes coding generics to facilitate the top-level integration at a later phase.

- **Test Plan and Test Status**

- **README files**

- **Comments in the code**
...if we did not have **SystemC Coding Style**?

- Everyone would use a different naming convention.
- There will be no common directory structure when releasing the models.
- When the codes are passed to the implementation team it would be a puzzle for them to figure out the followed conventions.
- Problems would arise when sharing codes.

...if we did not have **Block Diagram Conventions**?

- Everyone would use a different drawing convention.
- This would create misunderstandings between the modelling teams and later for the implementation team.
- As a result drawing a block diagram would lose its meaning as its aim is to facilitate of understanding the model later by someone else.
Processes in SC_CTOR like SC_CTHREAD, SC_THREAD, SC_METHOD. (Don't place process bubbles for SC_SLAVE and other methods!)

Classes with thick lines

Internal named signal

Internal signal

Latency of the process

Throughput of the process

ConfigSR ShiftReg

Instance Name

Module Name

process_name()

master port name

slave port name

PortName_In

PortName_Out

Class_In

LAT

THRMClk

Hierarchical port connection. This port connects to upper level module.

Write the external port name inside this.
...if we did not have SystemC Coding Generics?

- Top level integration would be a nightmare!
- A lot of things like common definitions, duplicate port, signal, module names had to be changed in the codes.
- The changes would be done by the top level integrators, which is error prone, as they don’t know the details of the codes.
- Or the section coders would make the changes to their codes, which would consume the project time.
- There won’t be a reference document for the section coders, to consider top-level integration issues.
...if we did not have Test Plan documents?

- Supervisors would not easily understand how the model will be tested. They would not easily add or remove tests.
- There would not be a track list during the testing of a model. This would probably result in some of the tests to be forgotten.
- There would not be a reference to see the results of the tests.
- If later someone wanted to rerun the tests, he/she would go into the details of the testbench to learn how to run it. **Time Consuming!**
Part IV. AME’s System Design Methodology in Practice

- Simplified section tutorial example
- SystemC executable spec: modelling and simulation
  - Top Level Integration
  - Master/Slave Library
  - Documented methodology
  - Different levels of abstraction
    - TF#1 - TF#2 - CA#1 - CA#2
  - Testbench Strategy
- SystemC co-simulation
- Summary
SystemC Executable Spec.
TF#1 Model of ComplexMPY

Complex Multiplier

Parameter, Passing through Pointers (invisible at the block boundaries)
UpdateConfig
ConfigUpdated
Master/Slave Link
FIFO

Processor
Operational FW & TestBench

SC_MODULE(CMPY_LCtrl) {
    // ports
    sc_inmaster<bool> UpdateConfig;
    sc_outmaster<bool> ConfigUpdated;
    sc_outmaster<bool> Update;
    sc_inmaster<bool>  notEmpty, notFull;
    sc_outmaster<bool> Enable;
    ...
    void control();   // process
    SC_CTOR(CMPY_LCtrl) {
        SC_THREAD(control);
        sensitive_pos << CLK;
    }
};

SC_MODULE(CMPY_DPath) {
    // ports
    sc_inmaster<Symbol64C>  D_In;
    sc_outmaster<Symbol64C> D_Out;
    sc_inslave<bool>        Enable;
    sc_inslave<bool>        Update;
    ...
    // local member data
    Symbol64C coef_;        
    // configuration access via pointer
    config_CMPY& configClass_;        
    ...
    void updCoef();  // process
    void multiply(); // process
    SC_CTOR(CMPY_DPath) {
        SC_SLAVE(updCoef,  Update);
        SC_SLAVE(multiply, Enable);
    }
};
SystemC Executable Spec.
TF#1 Model of ComplexMPY

- Multiply <Symbol64C> (64 complex samples) when Enable is triggered
  - Using “Slow” Functional clock CLK
  - Process execution time $\Rightarrow 1 \times \text{wait()}$ for a functional CLK tick

```cpp
void CMPY_LCtrl::control() {
    while(true) {
        if (UpdateConfig.read()) {
            Update.write(TRUE);
            ConfigUpdated.write(TRUE);
        }
        if (notEmpty.read() && notFull.read()) {
            Enable.write(TRUE);
        }
        wait();  // Functional clock CLK
    }
}

void CMPY_DPath::updCoef() {
    if (Update.read()) {
        coef_ = configClass_.Coef();
    }
}
```

SC_MODULE(CMPY_LCtrl) {
    // ports
    sc_inmaster<bool> UpdateConfig;
    sc_outmaster<bool> ConfigUpdated;
    sc_outmaster<bool> Update;
    sc_inmaster<bool>  notEmpty, notFull;
    sc_outmaster<bool> Enable;
    ...

    void control();   // process
    SC_CTOR(CMPY_LCtrl) {
        SC_THREAD(control);
        sensitive_pos << CLK;  // Funct clk
    }
}

SC_MODULE(CMPY_DPath) {
    // ports
    sc_inmaster<bool> Update;
    ...

    void updCoef();   // process
    SC_CTOR(CMPY_DPath) {
        SC_THREAD(updCoef);
        sensitive_pos << clk;  // Funct clk
    }
}
```

Result ready here!
“Computed in ‘NoTime’“

Output ready here!
Multiply <Symbol64C> (64 complex samples) when Enable is triggered

- Using “Slow” Functional clock CLK
- Process execution time $\Rightarrow$ 1 x wait() for a functional CLK tick

```c
void CMPY_LCtrl::control() {
    while(true) {
        if (UpdateConfig.read()) {
            Update.write(TRUE);
        }
        if (notEmpty.read() && notFull.read()) {
            Enable.write(TRUE);
        }
        wait(); // Functional clock CLK
    }
}
void CMPY_DPath::updCoef() {
    if (Update.read()) {
        coef_ = configClass_.NewCoef.value();
        ConfigUpdated.write(TRUE);
    }
}
```

```c
SC_MODULE(CMPY_LCtrl) {
    // ports
    sc_inmaster <bool> UpdateConfig;
    sc_outmaster <bool> ConfigUpdated;
    sc_outmaster <bool> Update;
    sc_inmaster <bool> notEmpty, notFull;
    sc_outmaster <bool> Enable;
    ...
    void control();   // process
    SC_CTOR(CMPY_LCtrl) {
        SC_THREAD (control);
        sensitive_pos << CLK;
        // Funct clk
    }
}
```

```c
void CMPY_Dpath::multiply() {
    Symbol64C in_ , tmp_ ;
    in_ = D_In.read();
    for(i=0 ; i<64 ; i++) {
        RealCoef_   = coef_.sample[i].data.real();
        ImaginCoef_ = coef_.sample[i].data.imag();
        RealIn_     = in_.sample[i].data.real();
        ImaginIn_   = in_.sample[i].data.imag();
        RealResult_ = (RealIn_ * RealCoef_) - (ImaginIn_* ImaginCoef_);
        ImaginResult_ = (RealIn_* ImaginCoef_) +(ImaginIn_* RealCoef_);
        // Putting result samples in a temporary symbol
        tmp_.sample[i] = complex (RealResult_, ImaginResult_);
    }
    // Writing the result to the output port
    D_Out.write( tmp_ );
}
```

```c
SC_MODULE(CMPY_DPath) {
    // ports
    sc_inmaster<Symbol64C> D_In;
    sc_outmaster<Symbol64C> D_Out;
    sc_inslave<bool>        Enable;
    sc_inslave<bool>        Update;
    ...
    // local member data
    Symbol64C coef_;
    // configuration access via ptr
    config_CMPY& configClass_;
    ...
    void updCoef();  // process
    void multiply(); // process
    SC_CTOR(CMPY_DPath) {
        SC_SLAVE(updCoef,  Update);
        SC_SLAVE(multiply, Enable);
    }
}
```

```
```
Multiply <Symbol64C> (64 complex samples) when Enable is triggered

- Using “Fast” Functional clock CLK64
- Process execution time $\Rightarrow 1 \times \text{wait(64)}$ with functional CLK64 ticks

```c
void CMPY_LCtrl::control() {
    while(true) {
        if (UpdateConfig.read()) {
            Update.write(TRUE);
            ConfigUpdated.write(TRUE);
        }
        if (notEmpty.read() && notFull.read()) {
            Enable.write(TRUE);
        }
    }
}

void CMPY_DPath::updCoef() {
    if (Update.read()) {
        coef_ = configClass_.Coef();
    }
}
```

```
SC_MODULE(CMPY_LCtrl) {
    // ports
    sc_inmaster<bool> UpdateConfig;
    sc_outmaster<bool> ConfigUpdated;
    sc_outmaster<bool> Update;
    sc_inmaster<bool> notEmpty, notFull;
    sc_outmaster<bool> Enable;
    ...

    void control(); // process

    SC_CTOR(CMPY_LCtrl) {
        SC_THREAD(control);
        sensitive_pos << CLK64; // Fast clk
    }
};
```

```
`time`
```
Multiply <Symbol64C> (64 complex samples) when Enable is triggered
- Using “Fast” Functional clock CLK64
- Process execution time => 1 x wait(64) with functional CLK64 ticks

```cpp
void CMPY_LCtrl::control() {
    while(true) {
        if (UpdateConfig.read()) {
            Update.write(TRUE);
        }
        if (notEmpty.read() && notFull.read()) {
            Enable.write(TRUE);
        }
    }
}
void CMPY_DPath::updCoef() {
    if (Update.read()) {
        coef_ = configClass_.NewCoef.value();
        ConfigUpdated.write(TRUE);
    }
}
```
Multiply \(<\text{Symbol64C}>\) (64 complex samples) when Enable is triggered

- Using “Fast” Functional clock CLK64
- Process execution time $\Rightarrow 1 \times \text{wait(64)}$ with functional CLK64 ticks

```c
void CMPY_Dpath::multiply_1() {
    Symbol64C in_ , tmp_; 

    in_ = D_In.read();
    for(i=0 ; i<64 ; i++) {
        RealCoef_   = coef_.sample[i].data.real();
        ImaginCoef_ = coef_.sample[i].data.imag();

        RealIn_     = in_.sample[i].data.real();
        ImaginIn_   = in_.sample[i].data.imag();

        // multiplying data samples by coeff samples
        RealResult_ = (RealIn_ * RealCoef_) - (ImaginIn_* ImaginCoef_);
        ImaginResult_ = (RealIn_* ImaginCoef_) +(ImaginIn_* RealCoef_);

        // Putting result samples in a temporary symbol
        tmp_.sample[i] = complex (RealResult_ , ImaginResult_);
    }

    // Writing the result to the output port
    wait(63); // Fast clock CLK64
    D_Out.write( tmp_ );
    wait(); // Fast clock CLK64
}
```

```c
SC_MODULE(CMPY_LCtrl) {
    // ports
    sc_inmaster<bool> UpdateConfig;
    sc_outmaster<bool> ConfigUpdated;
    sc_outmaster<bool> Update;
    sc_inmaster<bool>  notEmpty, notFull;
    sc_outmaster<bool> Enable;
    ...

    void control(); // process

    SC_CTOR(CMPY_LCtrl) {
        SC_THREAD(control);
        sensitive_pos << CLK64; // Fast clk
    }
};
```
**Tutorial Example**

**CA#1 Model of ComplexMPY**

**Operational FW & TestBench**

---

```
SC_MODULE(CMPY_LCtrl) {
// ports
sc_inmaster<bool> UpdateConfig;
sc_outmaster<bool> ConfigUpdated;
sc_outmaster<bool> Update;
sc_inmaster<bool> notEmpty, notFull;
sc_outmaster<bool> Enable;
...
void control(); // process
SC_CTOR(CMPY_LCtrl) {
SC_THREAD(control);
sensitive_pos << CLK64;
// Fast clk
}
}
```

```
SC_MODULE(CMPY_DPath) {
// ports
sc_inmaster<SampleC> D_In;
sc_outmaster<SampleC> D_Out;
sc_inslave<bool> Enable;
sc_inslave<bool> Update;
...
// local member data
SampleC coef_; // configuration access via pointer
config_CMPY& configClass_;
...
void updCoef(); // process
void multiply_CA1(); // process
SC_CTOR(CMPY_DPath) {
SC_SLAVE(updCoef, Update);
SC_SLAVE(multiply_CA1, Enable);
}
```

---

On the diagram, the Complex Multiplier is shown with a FIFO linked to a Processor. The diagram illustrates the flow of data and control signals between the components.
void CMPY_LCtrl::control() {
  while(true) {
    if (UpdateConfig.read()) {
      Update.write(TRUE);
      ConfigUpdated.write(TRUE);
    }
    if (notEmpty.read() && notFull.read()) {
      Enable.write(TRUE);
    }
    wait();  // Fast clock CLK-64
  }
}

void CMPY_DPath::updCoef() {
  if (Update.read()) {
    coef_ = configClass_.Coef();
  }
}

void CMPY_DPath::multiply_CA1() {
  SampleC in_, tmp_;
  in_ = D_In.read();
  RealCoef_ = coef_.sample[i].data.real();
  ImaginCoef_ = coef_.sample[i].data.imag();
  RealIn_ = in_.sample[i].data.real();
  ImaginIn_ = in_.sample[i].data.imag();
  // multiplying data samples by coeff samples
  RealResult_ = (RealIn_ * RealCoef_) - (ImaginIn_* ImaginCoef_);
  ImaginResult_ = (RealIn_* ImaginCoef_) + (ImaginIn_* RealCoef_);
  // Putting result samples in a temporary symbol
  tmp_.sample[i] = complex(RealResult_, ImaginResult_);
  // Writing the result to the output port
  D_Out.write(tmp_);
}
One complex multiplication per clock tick
- FIFO on sample basis
- Using “Fast” clock CLK-64

```c
void CMPY_DCtrl::control() {
    while(true) {
        if (UpdateConfig.read()) {
            Update.write(TRUE);
            ConfigUpdated.write(TRUE);
        }
        if (notEmpty.read() && notFull.read()) {
            Enable.write(TRUE);
        }
        wait();  // Fast clock CLK64
    }
}

void CMPY_DPath::updCoef() {
    if (Update.read()) {
        coef_ = configClass_.Coef();
    }
}
```

```c
void CMPY_Dpath::multiply_CA1() {
    SampleC in_ , tmp_ ;
    in_ = D_In.read();
    RealCoef_ = coef_.sample[i].data.real();
    ImaginCoef_ = coef_.sample[i].data.imag();
    RealIn_ = in_.sample[i].data.real();
    ImaginIn_ = in_.sample[i].data.imag();
    RealResult_ = (RealIn_ * RealCoef_) - (ImaginIn_* ImaginCoef_);
    ImaginResult_ = (RealIn_* ImaginCoef_) +(ImaginIn_* RealCoef_);
    // Putting result samples in a temporary symbol
    tmp_.sample[i] = complex (RealResult_ , ImaginResult_);
    // Writing the result to the output port
    D_Out.write( tmp_ );
}
```
Functional specification to HW-designers

- one complex multiplication per fast CLK-64 cycle ($THR=1$, $LAT=1$)
- $(a+jb) \times (c+jd) = (a*c - b*d) + j(c*b + a*d)$

Latency!
+ Throughput!

Reference Clk
= Input for HW designers

CLK-64

Result of Real part

Result of imaginary part
Possible implementation:

- 1 Multiplier, 1 Adder/Subtractor;
- use CLK-HW, 4 times faster than CLK64

Latency!
+ Throughput!
Reference Clk
= Input for HW designers
One multiplication per clock tick
- Using “Very Fast” clock CLK-HW
- 4 CLK-HW cycles in order to perform a complex multiplication

SC_MODULE(CMPY_DPath) {
    // ports
    sc_inmaster<SampleC>    D_In;
    sc_outmaster<SampleC>   D_Out;
    sc_inslave<bool>        Enable;
    sc_inslave<bool>        Update;

    // local member data
    SampleC coef_;

    void updCoef();  // process
    void multiply_CA2(); // process

    SC_CTOR(CMPY_DPath) {
        SC_SLAVE(updCoef,  Update);
        SC_SLAVE(multiply_CA2, Enable);
    }
};

SC_MODULE(CMPY_LCtrl) {
    // ports
    sc_inmaster<bool> UpdateConfig;
    sc_outmaster<bool> ConfigUpdated;
    sc_outmaster<bool> Update;
    sc_inmaster<bool>  notEmpty, notFull;

    // local member data
    SampleC coef_;

    void control();   // process

    SC_CTOR(CMPY_LCtrl) {
        SC_THREAD(control);
        sensitive_pos << CLK-HW; // Fast clk
    }
};

.h file
One multiplication per clock tick

- Using “Very Fast” clock CLK
- 4 CLK-HW cycles in order to perform a complex multiplication

```c
void CMPY_LCtrl::control() {
    while (true) {
        if (UpdateConfig.read()) {
            Update.write(TRUE);
            ConfigUpdated.write(TRUE);
        }
        if (notEmpty.read() && notFull.read()) {
            Enable.write(TRUE);
        }
    }
}

void CMPY_DPath::updCoef() {
    if (Update.read()) {
        coef_ = configClass_.Coef();
    }
}
```

```c
void CMPY_Dpath::multiply_CA2() {
    SampleC in_, tmp_;
    in_ = D_In.read();
    RealCoef_ = coef_.sample[i].data.real();
    ImaginCoef_ = coef_.sample[i].data.imag();
    RealIn_ = in_.sample[i].data.real();
    ImaginIn_ = in_.sample[i].data.imag();
    // multiplying data samples by coeff samples
    RealResult1_ = (RealIn_ * RealCoef_);
    wait();  // Hardware clock CLK-HW
    RealResult_ = (RealResult1_ - (ImaginIn_* ImaginCoef_));
    wait();  // Hardware clock CLK-HW
    ImaginResult1_ = (RealIn_* ImaginCoef_);
    wait();  // Hardware clock CLK-HW
    ImaginResult_ = (ImaginResult1_ + (ImaginIn_* RealCoef_));
    // Putting result samples to a temporary symbol
    tmp_.sample[i] = complex(RealResult_, ImaginResult_);
    // Writing the result to the output port
    D_Out.write(tmp_);
    wait();  // Hardware clock CLK-HW = 4 x faster than CLK64
}
```
SystemC Executable Spec.
Latency / Throughput in Block Diagram

Processes in SC_CTOR like SC_CTHREAD, SC_THREAD, SC_METHOD. (Don't place process bubbles for SC_SLAVE and other methods!)

- ConfigSR ShiftReg
- Internal named signal

Classes with thick lines

Write the external port name inside this. Hierarchical port connection. This port connects to upper level module.

Latency of the process

Throughput of the process

Clock Name

Signal connection

Master port name

Process name()

Slave port name

Latency of the process

Throughput of the process

Clock Name

Write the external port name inside this. Hierarchical port connection. This port connects to upper level module.

Latency of the process

Throughput of the process

Clock Name
Latency + Throughput + Ref. Clk

Latency / Throughput

```
instate[LAT] //0.....LAT-1
init() : instate[i]=0;
inptr = 0;

while(true) {
    a = Data_In.read();
inptr = index % LAT;
instate[inptr] = a;
outptr = (inptr + 1) % LAT;
b = calc(instate[outptr]);
wait(THR-1);
Data_Out.write(b);
index++;
wait();
}
```

```
instate[LAT] //0.....LAT-1
init() : instate[i]=0;
inptr = 0;

while(true) {
    a = Data_In.read();
inptr = index % LAT;
instate[inptr] = a;
outptr = (inptr + 1) % LAT;
b = calc(instate[outptr]);
wait(THR-1);
Data_Out.write(b);
index++;
wait();
```
Part IV. AME’s System Design Methodology in Practice

- Simplified section tutorial example
- SystemC executable spec: modelling and simulation
  - Top Level Integration
  - Master/Slave Library
  - Documented methodology
  - Different levels of abstraction
  - Testbench Strategy
    - Horizontal - Vertical
- SystemC co-simulation
- Summary
SystemC Executable Spec.
Testbench Strategy (horizontal re-use)

Processor
Operational Firmware

Load  Evaluate  Get/Analyze
Load Evaluate Get/Analyze

Complex Multiplier

Master/Slave
Link
Coeff
Result
DataPath
LCtrl
ConfigUpdated
UpdateConfig
Parameter Passing through Pointers (invisible at the block boundaries)
Through Pointers
Parameter Passing through Pointers

Processor

FIFO

SystemC Executable Spec.
Testbench Strategy (horizontal re-use)
Matlab as the algorithmic reference

SystemC Testbench

Reused in all stages of the design!

Algorithmic Modelling

SystemC Modelling

Implementation

Prototype Tests

Matlab

HW

FW

VHDL

C-code
SystemC Executable Spec.
Testbench Strategy (vertical re-use)

Complex Multiplier

Load Evaluate Get/Analyze

Processor

FIFO

Master/Slave Link

Coef

Result

DataPath

LCtrl

ConfigUpdated

UpdateConfig

Parameter Passing through Pointers (inval is at the block boundaries)

SystemC Executable Spec.
Testbench Strategy (vertical re-use)

Algorithmic Reference

.mat

Equal?

.mat

SystemC Result

Load Evaluate Get/Analyze

Processor

FIFO

Master/Slave Link

Coef

Result

DataPath

LCtrl

ConfigUpdated

UpdateConfig

Parameter Passing through Pointers (inval is at the block boundaries)

SystemC Executable Spec.
Testbench Strategy (vertical re-use)

Algorithmic Reference

.mat

Equal?

.mat

SystemC Result

Load Evaluate Get/Analyze

Processor

FIFO

Master/Slave Link

Coef

Result

DataPath

LCtrl

ConfigUpdated

UpdateConfig

Parameter Passing through Pointers (inval is at the block boundaries)
SystemC Executable Spec.

Execution of the Model

<table>
<thead>
<tr>
<th>Rx_Section5 module test starts!</th>
</tr>
</thead>
</table>

Test_reference list:
1 - TopLevel_Push_Tag_Only
2 - TopLevel_BPSK_1_2
3 - TopLevel_QPSK_3_4
4 - TopLevel_16QAM_9_16
5 - TopLevel_64QAM_2_3
6 - TopLevel_BPSK_1_2_4OFDM

>> Select one of the test_reference numbers: 2
>> Do you want debug-info?(0=No,1=Yes) 0

(SOFTCODING) : soft reseted at 0 0 clk cycle
(DEINTERLEAVER) : soft reseted at 0 0 clk cycle
(DEPUNCTURER) : soft reseted at 0 0 clk cycle
(VITERBI) : soft reseted at 0 0 clk cycle
(DESCRAMBLER) : soft reseted at 0 0 clk cycle

(SCHEDULER) : process_scheduler() started at 0 0 clk cycle

(TESTBENCH) : Matlab input file = ./input_mat_files/TopLevel_BPSK_1_2_in.mat
(TESTBENCH) : Matlab output file = ./output_mat_files/TopLevel_BPSK_1_2.out.mat
(SCHEDULER) : config signals updated at 50 1 clk cycle
(SOFTCODING) : config updated at 50 1 clk cycle
(DEINTERLEAVER) : config updated at 50 1 clk cycle
(DEPUNCTURER) : config updated at 50 1 clk cycle
(VITERBI) : config updated at 50 10 clk cycle
(DESCRAMBLER) : config updated at 50 1 clk cycle

(SCHEDULER) : current_config_DeScramblerMode() -->2 at 5350 107 clk cycle

(DEINTERLEAVER) : enabled for OFDM symbol 1 at 100 2 clk cycle
(DEPUNCTURER) : enabled for OFDM symbol 1 at 1000 22 clk cycle
(VITERBI) : enabled for OFDM symbol 1 at 1300 26 clk cycle
(DESCRAMBLER) : enabled for OFDM symbol 1 at 5350 107 clk cycle

(TESTBENCH) : Section is done!
(TESTBENCH) : Files are closed!

*****************************************************************************
* START COMPARING OUTPUT WITH GOLDEN REFERENCE
*****************************************************************************
* First file : ./output_mat_files/TopLevel_BPSK_1_2_out.mat
* Second file : ./golden_output_mat_files/TopLevel_BPSK_1_2_golden_out.mat
* Arrays in two files:
  * RxS5TopLevelOut_systemc 2 160x1 real numbers
  * TOPLEVELOutput 2 160x1 real numbers

*****************************************************************************
* REPORT:
* Number of bits compared : 160
* There is no difference between arrays in two files
* Files are closed successfully.
*****************************************************************************

Comparison results are written to "results.log" file.
**START COMPARING OUTPUT WITH GOLDEN REFERENCE**

First file: ./output_mat_files/TopLevel_BPSK_1_2_out.mat
Second file: ./golden_output_mat_files/TopLevel_BPSK_1_2_golden_out.mat

Arrays in two files:
- RxS5TopLevelOut_systemc 2 160x1 real numbers
- TOLEVELOutput 2 160x1 real numbers

Comparing 1. column (OFDM symbol except descrambler)
- 0 ............................................... …
- 50 ............................................... …
- 100 ............................................... …
- 150 ............................................... …

Comparing 1. Column
- SYSTEMC MATLAB
- 0. 1 1
- 1. 1 1
- 2. 1 1
- 3. 1 1
- 4. 0 0
- 5. 0 0
- 6. 0 0
- 7. 0 0

REPORT:
- Number of bits compared: 160
- There is no difference between arrays in two files
- Files are closed successfully.
Part IV. AME’s System Design Methodology in Practice

- Simplified section tutorial example
- SystemC executable spec: modelling and simulation

- **SystemC co-simulation**
  - AME needs and CoWare N2C as a solution
  - Complex Multiplier for SystemC/VHDL cosim
  - Complex Multiplier for SystemC/ISS cosim
  - Complex Multiplier for SystemC/ISS/VHDL cosim

- Summary
Co-simulation
AME needs and N2C as a solution

Coware’s N2C flow

1 System Specification
- System specification & environment model
- C simulation

2 Partitioning & interface design
- Partitioning & interface design
- Simulation - C - Assembly

3 Hardware implementation and software optimization
- Hardware implementation
- Software optimization
- Simulation - Multi-level mixed mode

AME Flow
- Matlab
- Algorithmic Design and Validation

SystemC
- BCA
- BCA
- RTOS
- RTO
- RTL
- C

CoSim Environment
- Software Design and Validation
- Design and Validation
- Firmware Design and Validation

Sign-off system functionality
- Sign-off between system developer, hardware developer and software developer
- Sign-off for processing

SystemC Tutorial © 2002 - Alcatel Microelectronics
Co-simulation
SoC SystemC / VHDL CoSim

- SystemC model as reference for "HW (VHDL) design"

- SystemC and VHDL Co-simulation
Complex Multiplier

Parameter Passing through Pointers

Through Pointers

SystemC I/F

UpdateConfig

ConfigUpdated

Processor

BlackBox

FIFO

Master/Slave Link

Processor (Invisible at the block boundaries)
Wrapper around VHDL: to convert generic C / SystemC types to VHDL data types

```c+
SC_MODULE(CoSimWrapper_BCA_version) {
    cwr_clock_inslave Clk_20M;
    sc_inslave <sc_uint<24>, sc_noHandshake<sc_uint<24> > > Mult_In;
    sc_outmaster <bool, sc_noHandshake<bool > >              ReadFIFOEn_Out;
}
SC_CTOR (CoSimWrapper_BCA_version) {
    // location of the source file
    cwr_add_attribute (this, "import", "source = ("./CoSimWrapper.Entity.vhd");");
    // replace "." to "_" when referring to signals in ports in this context
    // for example clk.clk to clk_clk
    cwr_add_context (this,
    "component CoSimWrapper
    port (\n    "    Clk_20M : in std_logic;\n    "    Mult_In : in std_logic_vector(23 downto 0);\n    "    ReadFIFOEn_Out : out std_logic;\n    // Connect SystemC ports to VHDL ports
    "    CoSimWrapper_1 : CoSimWrapper\n    port map (\n    "    Clk_20M => Clk_20M_clk,\n    "    Mult_In => Mult_In_d,\n    "    ReadFIFOEn_Out => ReadFIFOEn_Out_d,
```

sc_uint

VHDL component in SystemC
Co-simulation
SystemC/VHDL

SystemC I/F Wrapper: to convert C++ data types to generic C or SystemC data types

SC_MODULE(SystemCInterface) {
    // SystemC Interface port declarations
    sc_in_clk Clk20;
    sc_outmaster<Symbol64C> Mult_Out;
    sc_inmaster<Symbol64C> Mult_In;
    sc_inslave<bool , sc_noHandshake<bool > > ReadFIFOEn_In;
    sc_outmaster<sc_uint<24> , sc_noHandshake<sc_uint<24> > MultSerialData_Out;
    sc_inslave<sc_uint<24> , sc_noHandshake<sc_uint<24> > MultSerialData_In;
    ...
    void read_fifo_out();
}

SCCTOR(SystemCInterface) {
    SC_THREAD(read_fifo_out);
    sensitive_pos << Clk20;
    void SystemCInterface::read_fifo_out() { while(true){
        if (ReadFIFOEn_In.d.read() == false)
            counterread = 0;
        else
            if (counterread == 65)  counterread = 0;
            else{ RealMultData_ = Mult_In.read();
                ImagMultData_ = Mult_In.read()[counterread-1].real();
                SerialMultData_.range(23,12) = RealMultData_*NFloatBits;
                SerialMultData_.range(11,0) = ImagMultData_*NFloatBits;
                MultSerialData_Out.write(SerialMultData_);
                counterread = counterread + 1;}
        wait();}
ComplexMPY Abstract Levels

1 CA or TF

Complex Multiplier

Master/Slave Link

FIFO

Coef

Result

DataPath

LCtrl

ConfigUpdated

UpdateConfig

Processor

1 CA or TF

2 TF

2 TF

Coef

SystemC

Matlab

Algorithmic Design and Validation

Function

1 CA or TF

2 TF

1 CA or TF

Hardware

Firmware

Software

Robot Operating System (ROS)

BCA

RTOS

C

Return To Order (RTO)

RTL

Software Design and Validation

Hardware Design and Validation

Firmware Design and Validation

TF

UTF

System Design and Validation

30.01.2002
Co-simulation
SystemC/VHDL

1RTL

Complex Multiplier
Wrapper Around VHDL

SystemC I/F

Processor

2TF

Parameter Passing through Pointers
UpdateConfig
ConfigUpdated

std_logic_vector
coeff. Register
LCtrl

sc_uint
DataPath

std_logic

FIFO

FIFO

FIFO

SystemC

Algorithmic Design and Validation
Function

System

Design and Verification

Software

Hardware

Firmware

Target

Matlab

Abstr. RTOS

BCA

RTL

C

Software Design and Validation
Hardware Design and Validation
Firmware Design and Validation

Parameter Passing through Pointers

2TF

1RTL

Parameter Passing through Pointers
♦ SystemC model as reference for “FW (C-code) design”

♦ SystemC and ISS Co-simulation
N2c compliant System-C Wrapper around not N2c compliant System-C

Through Pointers

Complex Multiplier

IF

Complex Multiplier

IF

Through Pointers

C++ classes (may be hierarchical)

SystemC I/F (N2C)

PROCESSOR

C- code

Clock and Reset generators for processor

Basic C types
Non-hierarchical C structs
Indexed ports

SC_MODULE
mapped to a processor

sc_indexed port

new_fifo_config
struct

fifo_config class

Black Box
Wrapper for ISS

```cpp
// fifo_config class
template <class itemT>
class fifo_config {
    public:
        void fc_cmd(int command);
        int fc_cmd();
        void fc_index(int idx);
        int fc_index();
        void fc_value(itemT val);
        itemT fc_value();
        void fc_status(fifo_status_s sts);
        fifo_status_s fc_status();

    // Define the = operator
    inline void operator = (const fifo_config& rhs){
        fc_cmd_ = rhs.fc_cmd_;  
        fc_index_ = rhs.fc_index_;  
        fc_value_ = rhs.fc_value_;  
        fc_status_ = rhs.fc_status_;  
    }

    private:
        int fc_cmd_;  
        int fc_index_;  
        itemT fc_value_;  
        fifo_status_s fc_status_;};
```

```cpp
// new_fifo_config struct
struct fifo_config {
    int fc_cmd_;  
    int fc_index_;  

    // do not use hierarchical structs
    // divide fifo_status_s into members
    int fs_buffer_size;
    int fs_wrptr;
    int fs_rdpotr;
    int fs_nbrerrwr;
    int fs_full;
    int fs_empty;
};
```

```cpp
sc_indexed port for itemT fc_value_

sc_inoutmaster<int, sc_indexed<int, 128> > FIFOData;

// 128 comes from:
// 64 real part of an OFDM symbol
// 64 imaginary part of an OFDM symbol
```
Co-simulation
SoC SystemC / ISS / VHDL CoSim

- SystemC and VHDL and ISS Co-simulation

![Diagram of SystemC and ISS co-simulation](image)

- Co-simulation of SystemC, VHDL, and ISS
- Integration of hardware (HW) and firmware (FW)
- Operational and testbench firmware

**Key Components:**
- DPRam
- Processor
- ISS: Instruction Set Simulator
- ARM Processor
- SRAM
- CPU Core

![Illustration of SystemC and ISS co-simulation](image)
Complex Multiplier

Wrapper Around Vhdl with SystemC

SC_MODULE mapped to a processor

C++ classes

SystemC I/F (N2C)

PROCESSOR C-code

System Bus
Co-Simulation
ISS / VHDL

Complex Multiplier

SC_MODULE mapped to a processor
PROCESSOR C-code

SystemC I/F (N2C)

Wrapper Around Vhdl with SystemC I/F

std_logic_vector
Coeff. Register
LCtrl
DataPath

FIFO

FIFO

30.01.2002 SystemC Tutorial © 2002 - Alcatel Microelectronics Part IV - slide 55
SystemC is the GOLDEN reference throughout the design phases (iterative and model-based)

- Matlab as an algorithmic reference for SystemC.

Integration methodology based on

- Master/slave ports: M/S communication has advantages.
- Classes and pointers

Testbench strategy:

- Horizontal
- Vertical

Co-simulation:

- SystemC / VHDL
- SystemC / ISS
- SystemC / ISS / VHDL
Part V
AME’s Experience with SystemC Based Design Methodology
Part V. AME’s Experience

- The OWL Project
- Introducing SystemC in AME
- Design Engineers’ Involvement
- SystemC Approach
- Conclusion
Part V. AME’s Experience

- The OWL Project
  - Overview
  - Project Phases
  - Project Team Setup
- Introducing SystemC in AME
- Design Engineers’ involvement
- SystemC Approach
- Conclusion
AME’s HiperLAN-2 SoC (The OWL Chipset)

SystemC execution output:
>(Testbench) : Starting Test1.
>(Testbench) : Transmission started at time 500 ns.
>(PHY) : System reseted at time 300 ns
>(PHY) : System running for 3 frames......
>(PHY) : Transmitting frame 1...
>(PHY) : Transmitting frame 2...
>(PHY) : Transmitting frame 3...
>(PHY) : Transmission Completed!!
>(Testbench) : Comparison started
>........
>........
>(Testbench) : Comparison completed
>(Testbench) : see Test1.LOG file
OWL Project Phases
(Iterative design methodology ...)

- OWL Case: R&D Trajectory is ITERATIVE & MODEL based
  - Re-use of SW methodology coming from RUP in project context
  - Main target: reduce **Risks** in order to get ASAP **Working Product**
  - Early feedback from Executable SystemC model in First iteration
  - Models are not “throw-away” but **re-used throughout iterations**
  - SystemC is baseline for next iterations: testbench re-usage
  - keep models well-documented, high-quality and consistent!

<table>
<thead>
<tr>
<th>S</th>
<th>It.0</th>
<th>It.1</th>
<th>It.2</th>
<th>It.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>matlab</td>
<td><strong>systemC</strong></td>
<td>VHDL|ISS</td>
<td>FPGA</td>
<td><strong>Silicon</strong></td>
</tr>
</tbody>
</table>
## OWL Project Phases

(Iterative design methodology ...)

<table>
<thead>
<tr>
<th>Step</th>
<th>Risk Covered</th>
<th>Model/Platform Used</th>
<th>Test Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Feasibility</td>
<td>Matlab Function</td>
<td></td>
</tr>
<tr>
<td>It.0</td>
<td>Definition</td>
<td>SystemC Simulated</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specification</td>
<td>Timing &amp; Bit level</td>
<td></td>
</tr>
<tr>
<td>It.1</td>
<td>Development</td>
<td>SystemC as golden</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reference for SystemC+ISS+VHDL cosimulation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>environment</td>
<td></td>
</tr>
<tr>
<td>It.2</td>
<td>Development</td>
<td>RTL Implementation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SystemC as golden</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reference for FPGA + Integration boards</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Simulated Timing &amp; Bit level</td>
<td></td>
</tr>
<tr>
<td>It.3</td>
<td>Validation</td>
<td>Backend &amp; production</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SystemC as golden</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reference for real Silicon on Integration boards</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>and System Qualification tests</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Full system</td>
<td></td>
</tr>
</tbody>
</table>

### Phases

- **Matlab**: Matlab Modelling
- **systemC**: SystemC Modelling
- **VHDL\ISS**: HW design, FW design, Cosimulation
- **FPGA**: Check implementation versus SystemC golden reference
- **Silicon**: SystemC as golden reference for real Silicon on Integration boards and System Qualification tests Full system
Teams and Interactions (in/out AME)

- Tools
- FPGA
- IP Blocks
- SystemC
- VHDL
- SW
- CAD Method.
OWL Project Team Setup
(SystemC teams)

Tools

FPGA

VHDL

CAD Method.

Task 1

Task 2

Task 3

Task 4

Task 5


IP Blocks

SW
Part V. AME’s Experience

- The OWL Project
- Introducing SystemC in AME
- Design Engineers’ involvement
- SystemC Approach
- Conclusion
Introducing SystemC in AME

Experience/Activities

Start  week1  week5  Time

ManPower

SystemC language
AME SystemC Methodology
SystemC Methodology

VHDL Implementation & Firmware Implementation
Part V.  AME’s Experience

- The OWL Project
- Introducing SystemC in AME
- Design Engineers’ involvement
  - Phase 1: SystemC Modelling
  - Phase 2: Implementation, Verification and Testing
- SystemC Approach
- Conclusion
Design Engineers’ Involvement
(Phase 1: SystemC modelling)

Phase 1:
SystemC Modelling

HW designers
**HW designers in Phase 1**

- HW designers are familiar with SystemC concepts like:
  - ports
  - signals
  - wait() concept
- HDL background is an advantage while learning/using SystemC to model a HW block.
- HW design experience makes it easier for a designer to make timing assumptions for TF models in SystemC.
- C++ and Object-Oriented background helps HW-designers to get used to SystemC syntax.
# Design Engineers’ Involvement

(Phase 1: SystemC modelling)

<table>
<thead>
<tr>
<th>Phase 1: SystemC Modelling</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW designers</td>
</tr>
<tr>
<td>SW designers</td>
</tr>
</tbody>
</table>
SW designers in Phase 1

- SW designers are familiar with the syntax of SystemC, since it is a C++ based language.
- UTF models of SystemC are very similar to SW coding.
- SW designers are less familiar with HW-concepts like ports, signal timings and wait().
- Knowledge of abstract timing concepts such as throughput and latency makes it possible for SW designers to develop CA Models.
Design Engineers’ Involvement
(Phase 1: SystemC modelling)

Phase 1: SystemC Modelling

| HW designers |
| SW designers |
| System designers |
System designers in Phase 1

- Facilitate making executable spec. out of Matlab algorithms.
- Matlab usage (Algorithms + GUI) facilitates communication between HW/SW and system engineers.
- They decide on model abstraction level (BCA/CA vs. UTF/TF) to guide HW/SW engineers.
- SystemC language is easy to use by System engineers.
## Design Engineers’ Involvement

(Phase 2: Impl., verification and testing)

<table>
<thead>
<tr>
<th>Phase 1: SystemC modelling</th>
<th>Phase 2: Implementation, Verification and Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW designers</td>
<td>HW/SystemC Co-Simulation</td>
</tr>
<tr>
<td>SW designers</td>
<td>ISS/SystemC CoSimulation</td>
</tr>
<tr>
<td>System designers</td>
<td>SystemC reuse in system tests</td>
</tr>
<tr>
<td></td>
<td>- FPGA</td>
</tr>
<tr>
<td></td>
<td>- IC/ lab</td>
</tr>
</tbody>
</table>
Design Engineers’ Involvement

- SystemC creates a suitable platform for:
  - SW,
  - HW and
  - system designers

to contribute to

Executable specification of the design

- Mutual interaction between different teams/disciplines:
  - improves quality of the designed product
  - enhances the development time.
  - increases technical competence and related skills

This makes up an ideal TEAM!
Design Engineers’ Involvement
( Modelling and Implementation )

Classical approach

System designers

HW spec
- funct.
- perf.

HW designers

SW spec

SW designers

SystemC approach

System designers

HW spec
- funct.
- perf.

HW designers

SW spec

SW designers

System level understanding
Part V. AME’s Experience

- The OWL Project
- Introducing SystemC in AME
- Design Engineers’ involvement
- SystemC Approach
- Conclusion
SystemC Approach
(Testbench re-usage)

Classical approach

System designers

HW spec
- funct.
- perf.

SW spec

HW designers

SW designers

Testbench 1

Testbench 2

Testbench 3

Testbench 4

SystemC approach

System designers

HW spec
- funct.
- perf.

SW spec

System level understanding

HW designers

SW designers

Testbench

Testbench

Testbench

Testbench

30.01.2002
Matlab integration

- Matlab files are read by SystemC Models.
- SystemC dumps results to Matlab files.
- SystemC testbench compares the SystemC spec outputs to Matlab reference outputs and generates reports.

System designers

- HW spec
  - funct.
  - perf.

SW spec

SW designers

SystemC Approach
(Examples from OWL-experience)

Matlab model

Matlab stimuli

Matlab output

SystemC output

Reports

Analyze in Matlab environment
The Model is executable

Early bug detection!!

- alarm conditions of the system are detected at SystemC top level integration phase
- HW/SW interface is designed and bugs are fixed at SystemC design phase
SystemC Approach  
(Design complexity and time)

- Mastering design complexity
- Reducing design time (and first-time-right!)

### Examples of some sub-blocks:

<table>
<thead>
<tr>
<th></th>
<th>CA</th>
<th>VHDL code size (#lines)</th>
<th>Hardware size (#Gates)</th>
<th>VHDL development time (man-days)</th>
<th>SystemC development time (man-days)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SystemC code size</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HWObservations</td>
<td>400</td>
<td>2083</td>
<td>37295</td>
<td>80</td>
<td>65</td>
</tr>
<tr>
<td>Processor</td>
<td>1789</td>
<td>2221</td>
<td>10272</td>
<td>150</td>
<td>50</td>
</tr>
<tr>
<td>Rotor</td>
<td>686</td>
<td>770</td>
<td>60999</td>
<td>25</td>
<td>15</td>
</tr>
<tr>
<td>Correlator</td>
<td>102</td>
<td>210</td>
<td>24900</td>
<td>35</td>
<td>10</td>
</tr>
<tr>
<td>Counter</td>
<td>220</td>
<td>137</td>
<td>1183</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>TF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deinterleaver</td>
<td>200</td>
<td></td>
<td></td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>Viterbi</td>
<td>1100</td>
<td></td>
<td></td>
<td>40</td>
<td>5</td>
</tr>
</tbody>
</table>
Part V. AME’s Experience

- The OWL Project
- Introducing SystemC in AME
- Design Engineers’ involvement
- SystemC Approach
- Conclusion
Errors detected in early stage

System specification is generated:
- without thinking about HW/SW first.
- As specification evolves, parts to be implemented in SW and HW become clearer.

The methodology development done for this project
- smoothened integration phase (SW/HW SystemC model integration)
- may be reused in other projects

Testbenches come with the Spec.

These testbenches can be reused in implementation and testing phases.

Matlab is integrated to SystemC testbenches.
Part VI.
Summary and Conclusions
Need for System-on-Chip (SoC) Design Methodology

- OSCI SystemC provides solution:
  - Language & reference simulator
  - Suitable for system (hardware, firmware, software) level design
  - Supports different levels of abstraction with gradual and partial refinement
  - Executable specification
  - Open Source SystemC Initiative
  - Big momentum, world-wide
  - Roadmap: new versions evolve while being backwards compatible
  - Seed for SystemC-based tools and methodologies

Besides the language, one needs also a Methodology ...

- AME used the SystemC language to develop an own SystemC-based SoC-Design Methodology
Introducing SystemC in Alcatel Microelectronics

- Early adoption of SystemC language and reference simulator
- Fast learning curve
- In context of real-life project: OWL Wireless LAN Project
- Resulted in: OWL Executable System model (Hardware, Firmware, Software)
- Resulted in: SoC Design Methodology

The AME SoC Design Methodology

- Iterative based design: reducing risks, model based
- Iteration-0: SystemC model development
  - executable specification as golden reference
  - tool for early (designers/customers) feedback and design decisions
- Next iterations (design implementation, test and qualification):
  - SystemC model as reference & test-bench re-usage
  - SystemC based co-simulation during hardware and software design
Experience from the OWL Project

- SystemC is easy to learn and use
- SystemC is extendable: C++, Matlab integration, fixed point types
- Fast model development for complex and large systems
  - Abstraction level: mainly TF, some CA
  - Master/Slave communication, separated from behaviour
- Model integration methodology is required
- Methodology needs to be well documented
- Allows early involvement of several multi-disciplinary teams (system, hardware, software engineers):
  - Cross-fertilization of ideas
  - The ideal team!
- Executable model results in
  - Fast feedback from designers / customers / marketing
  - Early bug-detection
  - Faster and higher-quality design cycles

Experience from the OWL Project

SystemC is easy to learn and use
The OWL Project team at Alcatel Microelectronics gained large experience and expertise in System-on-Chip development and SoC Design Methodologies by using SystemC and making an executable specification of a real-life Wireless LAN product.

!! Hip hip hurray for SystemC !!
Appendix

References

- http://www.systemC.org
  SystemC source code
  UserGuide.pdf
  systemc-forum@systemc.org
- http://www-ti.informatik.uni-tuebingen.de/~systemc
  ESCUG
- http://www.ittf.no/aktiviteter/2001/dakforum/program
  DAK2001 Forum 2001
- http://www.alcatel.com/microelectronics
  marc.pauwels@mie.alcatel.be  ates.berna@alcatel.com.tr
  yves.vanderperren@mie.alcatel.be  fatma.ozdemir@alcatel.com.tr