

# SystemC 2.0

---

Presenter: Betül Buyukkurt

## Presentation Compiled From the Below Sources

---

- SystemC Standard
  - [http://www.cs.ucr.edu/~harry/classes\\_files/papers/Arnout\\_ASPDAC00.pdf](http://www.cs.ucr.edu/~harry/classes_files/papers/Arnout_ASPDAC00.pdf)
- An Introduction to System Level Modelling in SystemC 2.0
  - [http://www.systemc.org/papers/SystemC\\_WP20.pdf](http://www.systemc.org/papers/SystemC_WP20.pdf)
- Simulation Semantics of SystemC
  - [http://www.sigda.org/Archives/ProceedingArchives/Date/Date2001/papers/2001/date01/pdf/02b\\_1.pdf](http://www.sigda.org/Archives/ProceedingArchives/Date/Date2001/papers/2001/date01/pdf/02b_1.pdf)
- C-based Design of Systems-on-Chip: An EDA Perspective
  - <http://www.systemc.org/papers/ITGWorkshop2000HJS.ppt>
- [http://www.anslab.co.kr/download/SystemC/RISC\\_SC\\_All.pdf](http://www.anslab.co.kr/download/SystemC/RISC_SC_All.pdf)
- [http://wwwhome.cs.utwente.nl/~smit/codesign/SystemC\\_Fitch.pdf](http://wwwhome.cs.utwente.nl/~smit/codesign/SystemC_Fitch.pdf)

## Introduction

---

- System complexity is increasing
  - %22 of ASIC designs > 1M gates
  - %01 of ASIC designs > 10M gates
  - how to manage design complexity?
- Shorter development times
  - As low as 3 months on some consumer products
  - Faster designs w/ first time design success?

## Increasing need for SLDL

---

- SLDL: System Level Design Languages
- Hw only designs?
  - Today designs are at the RTL level
  - However, it is slow and complex to simulate larger designs in RTL
  - Event based simulation of billion gates??
- System-on-Chip(SoC) designs
  - SoCs designs are combinations of hw and sw
  - If continued with today's tools, SoC designers will have to struggle with combining hw & sw IPs from various sources described in various incompatible languages

## New Languages

---

- Rosetta
  - In the fall of 1996 the idea was kicked off by EDA – PTAB
  - Also supported by VHDL International
  - Specifics:
    - Integration of multiple domain theories into common semantic framework that supports the ability to budget and decompose system wide capabilities
    - Enable early estimation, co-synthesis and formal verification
  - Problems/Challenges:
    - Not yet complete
    - Adoption by hw, sw & architectural design communities
    - New tools need to be developed for the new language

## New Languages – 2

---

- SuperLog
  - Developed by Co-Design Automation
  - Supported by 12 other EDA companies
  - Specifics:
    - Simplicity of Verilog w/ power of C language
  - Disadvantages:
    - HDL-centric approach
  - Challenges:
    - Adoption

## Extending HDLs

- Verilog
  - In November 1999, Open Verilog International announced its plans for an SLD
  - Efforts were however hw focused; it was announced that the main goal was to enhance the productivity of the Verilog HDL user
- VSPEC
  - Developed at University of Cincinnati & funded by Arpa RASSP contract
  - VSEC is a VHDL-based interface language that can represent both hw and sw as formal properties and constraints by adding annotation to VHDL entries

## Extending HDLs – 2

- Challenges
  - HDLs don't provide effective design reuse
  - By the time the system is specified in an HDL it becomes too implementation specific (i.e. specified at the gate level, established architecture) to work it in another design
- Results:
  - Much of similar work has been abandoned
  - It is widely accepted that HDLs cannot effectively be expanded to cover the full range of required SLD capabilities, since systems are mostly sw

## C/C++ Based Efforts

- Pros:
  - Many hw designers already know the language
  - It is the most commonly taught language in colleges and universities
- Cons:
  - No natural way to represent
    - Hw concurrency
    - Hw Reactivity
    - Distributed Operation
    - Constrained data types and clocks

## C/C++ Based Efforts – 2

- C2Verilog by C-Level Design
  - C based product
  - Translates C language models into synthesizable Verilog code
- CynApps Suite from CynApps
  - Cynthesizer: C++ to Verilog translator
  - Cynchronizer: Verilog to C++ translator
  - Cyn++: Macro language that uses CynLib
  - Cyntax: C++ lint tool

## SystemC Efforts & OSI

- OSI: Open SystemC Initiative
- In September 1999, started by over 55 system, semiconductor, IP, embedded sw and EDA companies
- Endorsed to enable, promote, and accelerate system-level IP model exchange and co-design using a common C++ modeling platform (i.e. SystemC)
- Aim was to create, validate and share models with other companies using SystemC & a commonly agreed dialect of C++

## SystemC

- SystemC is a library of C++ classes
- It defines a modeling platform of C++ class libraries and a simulation kernel
- It's also a methodology that can be used to effectively create cycle-accurate models of
  - system architectures,
  - software algorithms,
  - hardware architectures,
  - interfaces of System On a Chip(SoC)
  - and system-level designs

## SystemC 1.0

---

- Hierarchically definable modules
- Ports and signals enable communication btw modules
- Allows fixed point computations
- Bits, bit vectors, char, int, fixed-point numbers, four state logic signals (1/0/X/Z), vectors of these types
- Concurrency is modeled using processes
- Independent threads; however allowed limited ability for specifying conditions to resume threads

## SystemC 2.0

---

- Objective:
  - Using a general purpose modeling foundation to address wide ranges of
    - models for computation,
    - design abstraction levels and
    - design methodologies

## Model Of Computation

---

- A Model of Computation (MOC) specifies
  - Model of Time:
    - real valued, integer valued, untimed
  - Model of Event Ordering:
    - partial/global
  - Sets of supported methods of communication btw concurrent processes
  - Rules for process activation

## Design Abstraction Level

---

- Untimed Functional (UTF) Level
  - Behavioral Modeling
- Timed Functional (TF) Level
  - Timed but not clocked
  - Performance Modeling
- Bus-Cycle Accurate (BCA) Level
  - Hardware with Bus Architecture
  - Processors synchronized with bus controllers
- Cycle Accurate (CA) Level
  - Synthesizable RTL

## Modeling with SystemC

---

- Currently analog modeling is not available
- But any discrete time system can be well modeled in SystemC
- Examples of models that SystemC can be used to build are:
  - Kahn Process Networks
  - Communicating Sequential Processes
  - Discrete Event(DE) RTL hardware modelling
  - DE Network Modeling
  - DE Transaction Based SoC platform modelling

## SystemC Overview

---

- Modules(sc\_module)
  - container class for processes and other modules; used to build hierarchy
  - have a constructor to instantiate processes and sub modules  
SC\_CTOR(module\_name)

## Ports

---

- Modules have ports
- used to pass data to/from processes
- can be defined as uni- (sc\_in<type>, sc\_out<type>) or bi-directional (sc\_inout<type>)
- ports are provided as templates

## Processes

---

- Where the functionality is defined
- They run concurrently, however code inside a process runs sequential
- Can be defined as
  - method: SC\_METHOD
  - thread: SC\_THREAD
  - clocked thread: SC\_CTHREAD

## Signals

---

- sc\_signal<type>
- Signals are connected to ports and through ports they establish a direct communication link between modules
- Value changes generate events for the event-based simulation
- Two forms of signals are supported in SystemC: resolve & unresolved
- Resolved signals can have more than one driver, while unresolved signals can only have one

## Clocks

---

- sc\_clock<>, sc\_signal<bool>
- time advances at clock edges and the edge can be specified as pos or neg
- multiple clocks with arbitrary phase relationship are allowed

## Datatypes

---

- Bits & bit vectors (sc\_bit, sc\_bv<N>)
- Logic (1/0/X/Z) & logic vectors (sc\_logic, sc\_lv<N>)
- Arbitrary/fixed precision signed unsigned integers with (sc\_bigint<N>, sc\_biguint<N>, sc\_int<N>, sc\_uint<N>)
- Fixed-point numbers

## Waiting and Watching

---

- wait(), wait\_until(...), watching(...)
- Waiting and watching provides hw reactivity
- Waiting refers to a blocking action while waiting for an event to happen, whereas watching refers to a non-blocking action
- watching enables preemption and can be done globally or locally

## SystemC-2.0 Specific Constructs

- Channels: Serves as a container for communication and synchronization
- Interfaces: specifies the set of access methods to the channel, however does not implement them
- Events: low-level synchronization primitives
- The concepts were inspired from SpecC language

## Example

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

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

## Example - cont...

```
class fifo : public sc_channel,
public write_if,
public read_if
{
public:
SC_CTOR(fifo) { // channel constructor
num_elements = first = 0;
}

void write(char c) {
if (num_elements == max)
wait(read_event);
data[(first + num_elements) % max] =
c;
++ num_elements;
write_event.notify();
}

void read(char & c) {
if (num_elements == 0)
wait(write_event);
c = data[first];
-- num_elements;
first = (first + 1) % max;
read_event.notify();
}

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

int num_available() { return
num_elements; }
private:
enum e { max = 10 };
//max is just a constant in class scope
char data[max];
int num_elements, first;
sc_event write_event, read_event;
};
```

## Example - cont...

```
// the producer module
class producer : public sc_module
{
public:
sc_port<write_if> out; // producer output port
SC_CTOR(producer) // module constructor
{
SC_THREAD(main); // start the process
}
void main() // the producer process
{
char c;
while (true) {
out->write(c); // write c into fifo
if (...)
out->reset(); // reset the fifo
}
};

// the consumer module
class consumer : public sc_module
{
public:
sc_port<read_if> in; // consumer input port
SC_CTOR(consumer) // module constructor
{
SC_THREAD(main); // start the process
}
void main() // the consumer process
{
char c;
while (true) {
in->read(c); // read c from the fifo
if (in->num_available() > 5)
... // perhaps speed up processing
}
};
```

## Example - end.

```
// the top module
class top : public sc_module
{
public:
fifo *fifo_inst; // a fifo instance
producer *prod_inst; // a producer
instance
consumer *cons_inst; // a consumer
instance
SC_CTOR(top) // the module constructor
{
fifo_inst = new fifo (Fifo1);
prod_inst = new producer("Producer1");
// bind the fifo to the producer's port
prod_inst->out(fifo_inst);
cons_inst = new consumer("Consumer1");
// bind the fifo to the consumer's port
cons_inst->in(fifo_inst);
};
```