Lecture 12:

An Introduction to SystemC

Sharif University of Technology
Computer Engineering Dept.

Winter-Spring 2008

Mehdi Modarressi
Introduction

- **SystemC motivations:**
  - Enhancing the current design methodologies
  - Modeling both hardware and software in system level design
Introduction - motivations

1. Conceptualize
2. Simulate in C/C++
3. Write specification document
4. Hand over specification document
5. Understand
6. (Re)Implement in HDL
7. (Re)Verify
8. Synthesize from HDL

Problems: Written specifications are incomplete and inconsistent.
Translation to HDL is *time consuming* and *error prone*.
Introduction - motivations

Current Methodology

- **Manual Conversion** Creates Errors
- Disconnect Between System Model and HDL Model
- Multiple System Tests

```
Refine
C/C++ System Level Model

Analysis

Result

VHDL/Verilog

Simulation

Synthesis
```
Introduction - motivations

1. Conceptualize
2. Simulate in C++
3. Write specification document
4. Hand over
   - Executable specification
   - Testbench
   - Written specification
5. Understand
6. Refine in C++
7. Verify reusing testbenches
8. Synthesize from C++
Introduction - motivations

- In System Level Design we should model both hardware and software.

- Need a unified environment for developing hardware as well as software.
  - Providing software support in an HDL: SystemVerilog
  - Providing hardware support in an SDL such as C++
Introduction

- C/C++ have no:
  - notion of time
    - No event sequencing
  - Concurrency
    - But H/W is inherently concurrent
  - H/W Data Types
    - No ‘Z’ value for tri-state buses
  - Some languages try to solve these problems
    - SpecC, ArchC, HandleC, HardwareC, and SystemC
SystemC

- A C++ based class library and design environment for system-level design
  - The library adds timing, concurrency, and hardware data types to the standard C++ language.
  - Does not modify or extend C++
- Suitable for functional description that might eventually be implemented as either HW or SW
- Open standard
  - Language definition is publicly available
  - Libraries are freely distributed
    - Synthesis tools are an expensive commercial product
SystemC consists of a
- Library of C++ classes, global functions, data types
- Simulation kernel that can be used for simulating hardware architecture.

SystemC Provides
- an executable specification of the System
  - An executable specification is a program that exhibits the same behavior as the system when executed.
- System-Level Modeling
  - A unified environment for modeling HW and SW
Design Methodology

SystemC Model

Simulation

Refinement

Synthesis

Rest of Process
SystemC History

- SystemC v.2 in 2001.
  - By OSCI (Open SystemC initiative)

- The simulation kernel in SystemC v.2 was much faster than version 1.
  - Event-based vs. Cycle-based

- SystemC v.2.1 in 2005
  - IEEE 1666 standard.

- www.systemc.org
SystemC Highlights

- Features as a codesign language
  - Modules
  - Concurrent processes
  - Ports
  - Signals
  - Rich set of port and signal types
  - Rich set of data types
  - Clocks
  - Cycle-based simulation
  - Multiple abstraction levels
  - Communication protocols
  - Debugging support
  - Waveform tracing
**SystemC**

- **class library and simulation kernel**
  - header files
  - libraries

- **your standard C/C++ development environment**
  - compiler
  - linker
  - debugger

- **source files for system and testbenches**
  - DSP
  - Interface
  - IP-Core
  - ASIC

- **make**
  - sim.x

- **"executable specification"**

- **executable = simulator**
SystemC Programming Model (cont’d)

- SystemC is C++
  - Any C++ statement is allowed
    - `cout`, `cin`, file I/O, etc
  - In principle, any C++ compiler can be used
    - MS VC++ 6.0 for windows
    - GCC for Linux
SystemC programming model

- Module
  - Like ‘Module’ in Verilog
    - actually a C++ class
  - Basic building block for structural partitioning
  - Contains ports, processes, data
  - Other modules

- Process
  - Actually a member function of a module
  - Basic specification mechanism for functional description
  - The concurrent part of the program
SystemC programming model

- Standard C++ classes coexist with SystemC modules
- In a module standard C++ function coexist with SystemC processes
SystemC Data-types

- C++ built in data types may be used but are not adequate to model Hardware.
  - long, int, short, char, unsigned long, unsigned int, unsigned short, unsigned char, float, double, long double, and bool.

- SystemC™ provides other types that are needed for System modeling.
  - Scalar boolean types: `sc_logic, sc_bit`
  - Vector boolean types: `scbv<length>, sclv<length>`
  - Integer types: `sc_int<length>, sc_uint<length>, sc_bign<length>, sc_biguint<length>`
  - Fixed point types: `sc_fixed, sc_ufixed`
A set of modules interacting through signals.

Module functionality is described by processes.
Signal connecting modules (declared in module)

Ports bound directly to each other (no signal)

Process can read/write ports and signals

Other methods can read/write ports and signals

Other methods can read/write ports and signals

SC_MODULE
{
  □ Ports
  □ Signals
  □ Variables
  □ Constructor
  □ Processes
  □ Modules
};
SystemC Basic Building Block

\texttt{SC\_MODULE}( \texttt{<module\_name>}) \{ \\
  // declaring port types \\
  \texttt{sc\_in<\texttt{int}> in;} \\
  \\
  // definition of processes \\
  \texttt{void func1()} \{ \\
    // circuit functionality \\
  \} \\
  \texttt{void func2()} \{ \\
    // functionality \\
  \} \\

\texttt{SC\_CTOR}( \texttt{<module\_name>}) \{ \\
  // declaring processes \\
  \texttt{SC\_METHOD(func1);} \\
  \texttt{sensitive<<in;} \\
\} \\
\}

• \texttt{SC\_CTOR()} is the creator of a module
  • registering a function as a process and determine its sensitivity list
  • Initializing module components

• Processes have a list of events of which they are sensitive.

• It specifies that process \textit{func} will execute when \texttt{in} changes
A Simple Example: Defining a Module

- Complex-number Multiplier
  \[(a+bi)(c+di) = (ac-bd)+(ad+bc)i\]

```
SC_MODULE(cmplx_mult) {
  sc_in<int> a, b;
  sc_in<int> c, d;
  sc_out<int> e, f;
  ...
}
```
A Simple Example: Defining a Module (cont’d)

```cpp
SC_MODULE(cmplx_mult) {
    sc_in<int> a, b;
    sc_in<int> c, d;
    sc_out<int> e, f;
    void calc();
    SC_CTOR(cmplx_mult) {
        SC_METHOD(calc);
        sensitive << a << b << c << d;
    }
};

void cmplx_mult::calc()
{
    e = a*c - b*d;
    f = a*d + b*c;
}
```
SystemC main function

- Every C/C++ program has a `main()` function.
- In SystemC the `main()` function is not used. Instead the function `sc_main()` is the entry point of the application.
- Before execution of the application code we should (calling `main()`)
  - initialize of the simulation kernel
  - initialize the SystemC structures

- Access to all SystemC classes and functions is provided in a single header file named “systemc.h”.
For the next session

- How to run SystemC with MS visual c++ 6
- How to view waveforms in Modelsim
- Data types
- Process types
- SC_main() structure
- ....