EEL 4783: HDL in Digital System Design

Lecture: SystemC Language and Its Usage

Prof. Mingjie Lin
```
#include "systemc.h"

SC_MODULE(add)  // module (class) declaration
{
    sc_in<int> a, b;  // ports
    sc_out<int> sum;

    void do_add()  // process
    {
        sum.write(a.read() + b.read());  // or just sum = a + b
    }

    SC_CTOR(add)  // constructor
    {
        SC_METHOD(do_add);  // register do_add to kernel
        sensitive << a << b;  // sensitivity list of do_add
    }
};
```
SystemC Introduction

• Why not leverage experience of C/C++ developers for H/W & System Level Design?
• But 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
SystemC is ...

• C++ Class Library use for
  – Cycle-Accurate model for Software Algorithm
  – Hardware Architecture
  – Interface of SoC (System-on-Chip)
  – System-level designs
  – Executable Specification

• www.systemc.org
  – All you can use information
SystemC Environment
SystemC History

• SystemC 1.0
  – Provide VHDL like capabilities
  – Simulation kernel
  – Fixed point arithmetic data types
  – Signals (communication channels)
  – Modules
    • Break down designs into smaller parts
SystemC History

• **SystemC 2.0**
  – Complete library rewrite to upgrade into true SLDL
  – Events as primitive behavior triggers
  – Channels, Interfaces and Ports
  – Much more powerful modeling for Transaction Level

• **Future SystemC3.0**
  – Modeling of Oss
  – Support of embedded S/W models
Objectives of SystemC 2.0

- Primary goal: Enable System-Level Modeling
  - Systems include hardware and software
  - Challenge:
    - Wide range of design models of computation
    - Wide range of design abstraction levels
    - Wide range of design methodologies
Objectives of SystemC 2.0

• Introduces a small but very general purpose modeling foundation => Core Language

• Support for other models of computation, methodologies, etc
  – They are built on top of the core language, hence are separate from it
    • Even SystemC1.0 Signals are built on top of this core in SystemC2.0
    • Other library models are provided:
      – FIFO, Timers, ...
SystemC Language Architecture

- SystemC 1.0
  - Modules and Processes are still useful in system design
  - But communication and synchronization mechanisms in SystemC1.0 (Signals) are restrictive for system-level modeling
    - Communication using queues
    - Synchronization (access to shared data) using mutexes

Diagram:
- Methodology-Specific Libraries
  - Master/Slave Library, etc.
- Layered Libraries
  - Verification Library
  - Static Dataflow, etc.
- Primitive Channels
  - Signal, Mutex, Semaphore, FIFO, etc.
- Core Language
  - Modules
  - Ports
  - Processes
  - Interfaces
  - Channels
  - Events
  - Event-driven simulation
- Data Types
  - 4-valued Logic type
  - 4-valued Logic Vectors
  - Bits and Bit Vectors
  - Arbitrary Precision Integers
  - Fixed-point types
  - C++ user-defined types

C++ Language Standard
SystemC vs. Metropolis

- Constructs to model system architecture
  - Hardware timing
  - Concurrency
  - Structure
- Adding these constructs to C
  - SystemC
    - C++ Class library
    - Standard C/C++ Compiler: bcc, msvc, gcc, etc…
  - Metropolis
    - New keywords & Syntax
    - Translator for SystemC
    - Many More features...
System Design Methodology

• Current
  – Manual Conversion from C to HDL Creates Errors
  – Disconnect Between System Model and HDL Model
  – Multiple System Tests
• SystemC (Executable-Specification)
  – Refinement Methodology
  – Written in a Single Language
Modeling Terms (I)

• Untimed Functional (UTF)
  – Refers to model I/F and functionality
  – No time used for regulating the execution
  – Execution & data transport in 0 time

• Timed Functional (TF)
  – Refers to both model I/F and functionality
  – Time is used for the execution
  – Latencies are modeled
  – Data Transport takes time
Modeling Terms (II)

- **Bus Cycle Accurate (BCA)**
  - Refers to model I/F, not functionality
  - Timing is cycle accurate, tied to some global clock
  - Does not infer pin level detail
  - Transactions for data transport

- **Pin Cycle Accurate (PCA)**
  - Refers to model I/F not model functionality
  - Timing is cycle accurate
  - Accuracy of the I/F at the pin Level

- **Register Transfer (RT) Accurate**
  - Refers to model functionality
  - Everything fully timed
  - Complete detailed Description, every bus, every bit is modeled
Model Types (1)

- **System Architectural**
  - Executable specification for H/W & S/W
  - Architecture Exploration, algorithm determination & proof
  - I/Fs are UTF with no pin detail for modeling communication protocols
  - Functionality UTF, sequential since it’s untimed
- **System Performance**
  - Timed executable specification for bithH/W & S/W
  - Used for time budgeting
  - Concurrent behavior modeled
- **Transaction Level (TLM)**
  - Typically describe H/W only
  - Model I/Fs are TF, functionality TF as well (either not cycle accurate)
  - Data Transfers & system Behavior modeled as transactions
Model Types (2)

- **Functional Model**
  - Above TLM, i.e. System Architectural & System Performance
- **System Level**
  - Above RTL
- **Behavioral Synthesis**
  - Architectural Analysis & Implementation
  - I/F cycle accurate with pin level detail
  - Functionality TF and not cycle accurate
- **Bus Functional Model (BFM)**
  - Used for simulation (mainly of processors)
  - Not meant for synthesis
  - I/F pin cycle accurate
  - Transactions for functionality
- **Register Transfer Level (RTL)**
  - Verilog, VHDL
- **Gate Level**
  - not good in SystemC
Current Methodology

- Manual Conversion Creates Errors
- Disconnect Between System Model and HDL Model
- Multiple System Tests
SystemC Methodology
Using Executable Specifications

• Ensure COMPLETENESS of Specification
  – “Create a program that Behave the same way as the system”
• UNAMBIGUOUS Interpretation of the Specification
• Validate system functionality before implementation
• Create early model and Validate system performance
• Refine and Test the implementation of the Specification
SystemC and User Module

User Module #1

User Module #2

.....

User Module #N

Event & Signal IF

C++ Class Library

Events

Hardware Simulation Kernel
(Event Scheduler)

SystemC

Executable Specification
SystemC Highlights (1)

- SystemC2.0 introduces general-purpose
  - Events
    - Flexible, low-level synchronization primitive
    - Used to construct other forms of synchronization
  - Channels
    - A container class for communication and synchronization
    - They implement one or more interfaces
  - Interfaces
    - Specify a set of access methods to the channel
- Other comm& sync models can be built based on the above primitives
  - Examples
    - HW-signals, queues (FIFO, LIFO, message queues, etc)
    - semaphores, memories and busses (both at RTL and transaction-based models)
SystemC Highlights (2)

- Support Hardware-Software Co-Design
- All constructs are in a C++ environment
  - Modules
    - Container class includes hierarchical Modules and Processes
  - Processes
    - Describe functionality
    - Almost all SLDL have been developed based on some underlying model of network of processes
  - Ports
    - Single-directional (in, out), Bi-directional mode
A system in SystemC
A system in SystemC
SystemC Highlights (3)

- Constructs in a C++ environment (continued)
  - Clocks
    - Special signal, Timekeeper of simulation and Multiple clocks, with arbitrary phase relationship
  - Event Driven simulation
    - High-Speed Event Driven simulation kernel
  - Multiple abstraction levels
    - Untimed from high-level functional model to detailed clock cycle accuracy RTL model
  - Communication Protocols
  - Debugging Supports
    - Run-Time error check
  - Waveform Tracing
    - Supports VCD, WIF, ISBD
Data Types

• SystemC supports
  – Native C/C++ Types
  – SystemCTypes

• SystemCTypes
  – Data type for system modeling
  – 2 value (‘0’, ’1’) logic/logic vector
  – 4 value (‘0’, ’1’, ’Z’, ’X’) logic/logic vector
  – Arbitrary sized integer (Signed/Unsigned)
  – Fixed Point types (Templated/Untemplated)
Communication and Synchronization (cont’d)
A Communication Modeling Example: FIFO
class write_if : public sc_interface
{
    public:
        virtual void write(char) = 0;
        virtual void reset() = 0;
};

class read_if : public sc_interface
{
    public:
        virtual void read(char&) = 0;
        virtual int num_available() = 0;
};
Declaration of FIFO channel

```cpp
class fifo: public sc_channel,
    public write_if,
    public read_if
{
    private:
        enum e {max_elements=10};
        char data[max_elements];
        int num_elements, first;
        sc_event write_event,
            read_event;
        bool fifo_empty() {...};
        bool fifo_full() {...};

    public:
        fifo() : num_elements(0),
            first(0);

    void write(char c) {
        if (fifo_full())
            wait(read_event);
        data[<you calculate>] = c;
        ++num_elements;
        write_event.notify();
    }

    void read(char &c) {
        if (fifo_empty())
            wait(write_event);
        c = data[first];
        --num_elements;
        first = ...;
        read_event.notify();
    }
```
void reset() {
    num_elements = first = 0;
}

int num_available() {
    return num_elements;
}
}; // end of class declarations
FIFO Example (cont’d)

- Any channel must
  - be derived from sc_channelclass
  - be derived from one (or more) classes derived from sc_interface
  - provide implementations for all pure virtual functions defined in its parent interfaces
- Note the following wait() call
  - wait(sc_event) => dynamic sensitivity
  - wait(time)
  - wait(time_out, sc_event)
- Events
  - are the fundamental synchronization primitive
  - have no type, no value
  - always cause sensitive processes to be resumed
  - can be specified to occur:
    - immediately/ one delta-step later/ some specific time later
SystemC Highlights (2)

- Support Hardware-Software Co-Design
- All constructs are in a C++ environment
  - Modules
    - Container class includes hierarchical Modules and Processes
  - Processes
    - Describe functionality
    - Almost all SLDL have been developed based on some underlying model of network of processes
  - Ports
    - Single-directional (in, out), Bi-directional mode
A system in SystemC
A system in SystemC
A system in SystemC
A system in SystemC
Final issues

• Please fill out the student info sheet before leaving

• Come by my office hours (right after class)

• Any questions or concerns?