SystemC – Lecture 1

Prof. Gerald E. Sobelman
Dept. of Electrical and Computer Engineering
University of Minnesota
Minneapolis, MN 55455 USA

火龙果 · 整理
uml.org.cn

Outline

- Overview of SystemC
  - Simulation environment
  - Modules and processes
  - Ports, channels and interfaces
  - Transaction-Level Modeling (TLM)
  - SystemC Verification Library (SCV)
- Simple Design Example
  - Half adder and full adder modules
  - Testbench: DUT, Driver, Monitor, Main
- General Module Structure

火龙果 · 整理
uml.org.cn

Introduction to SystemC

- Complex electronic systems require modeling and verification at higher-levels of abstraction in order to test them over lengthy, realistic sets of input conditions.
- Can also be used to maintain a consistent set of models and testbenches throughout the design refinement process.
- SystemC:
  - Provides a C++ class library for hardware
  - Support for Transaction-Level Modeling (TLM)
  - Support for functional verification

火龙果 · 整理
uml.org.cn

SystemC Overview

- SystemC runs in a C++ development environment, allowing modeling and verification of both hardware and software within a common framework.
- It is well suited for high-level hardware descriptions: RTL and above.
- The necessary libraries can be downloaded from **`http://www.systemc.org`** or they are available within commercial CAD environments (Cadence, Synopsys, etc).
- Compiling and linking the SystemC source files creates an executable file which, when run, performs the simulation.
- Results can be viewed in text files or through a waveform viewer.

火龙果 · 整理
uml.org.cn

Simulation Environment

- The organization of a SystemC simulation is shown below:

火龙果 · 整理
uml.org.cn

Modules

- An **SC_MODULE** is the basic unit of hardware description. Module designs are usually separated into two files:
  - A **.h** header file containing the input/output ports and a declaration of the functions which are used and their sensitivity lists, as well as instances of other modules.
  - A **.cpp** program file containing the implementations of those functions.

火龙果 · 整理
uml.org.cn

Example Header Structure

- A header file **`HalfAdder.h`** contains the following:

```
#include "systemc.h"
SC_MODULE(HalfAdder) {
        sc_in<bool> a, b;
        sc_out<bool> s, c;
        void HalfAdderProcess();
        SC_CTOR(HalfAdder) {
                SC_METHOD(HalfAdderProcess);
                sensitive << a << b;
        }
};
```

Ref: J. Bhasker, *A SystemC Primer*, Star Galaxy Publishing, 2002.

火龙果 • 整理
uml.org.cn

Example Program File Structure

- A program file **`HalfAdder.cpp`** specifies the details of the process declared in the header file. It contains the following:

```
#include "HalfAdder.h"

void HalfAdder::HalfAdderProcess() {
        s = a ^ b;
        c = a & b;
}
```

火龙果 · 整理
uml.org.cn

Processes

- There are two ways of specifying functional behavior: **SC_METHOD** and **SC_THREAD**.
- An **SC_METHOD** process can be used to specify either combinational or sequential logic.
  - The function is called according to its sensitivity list.
  - Once invoked, it executes and then returns control to the simulation kernel.

火龙果 · 整理
uml.org.cn

Processes – Cont.

- An **`SC_THREAD`** process is used to specify more complex behaviors which can be suspended and then reactivated.
  - The function is called when an event occurs on its sensitivity list.
  - A **`wait()`** function is used to suspend execution at various points.
  - The process can be suspended for a specified time interval or until a specified event occurs.
  - Useful for modeling complex behaviors at the system/algorithmic level, as well as for testbenches.

火龙果 · 整理
uml.org.cn

Processes – Cont.

- The sensitivity lists can contain signal names and/or specified events such as rising or falling edges.
- We can also have dynamic sensitivity in either type of process:
  - Add arguments to the **`wait()`** function in an **`SC_THREAD`** so that the process is activated when a specified combination of events occurs.
  - Use a **`next_trigger()`** function with arguments to specify the combination of events that will cause the next activation of an **`SC_METHOD`**.

火龙果 · 整理
uml.org.cn

Ports and Channels

- Connections between objects are created using three types of elements:
  - A *port* of type **`sc_in`**, **`sc_out`**, or **`sc_inout`** is at the edge of a module.
  - A *channel*:
    - specifies the type of link between either:
      - the ports of different modules
      - different processes within a single module
    - may consist of either:
      - one of the built-in primitive types:
        - **`sc_signal`**, **`sc_fifo`**, **`sc_mutex`**, etc.
      - a user-defined type with arbitrarily complex behavior

火龙果 · 整理
uml.org.cn

Interfaces

- An *interface* specifies the way that a given channel is accessed. Each channel has a set of corresponding interfaces. For example, interfaces for the channel **`sc_fifo`**:
  - **`sc_fifo_in_if`**:
    - takes data from an **`sc_fifo`** channel and puts it into a port
  - **`sc_fifo_out_if`**:
    - takes data from a port and puts it into an **`sc_fifo`** channel

火龙果 · 整理
uml.org.cn

Top-Level Function and the Design Hierarchy

- The top-level of a design is placed in a function **`sc_main`**, which is put into the file **`main.cpp`**. It is called by the SystemC environment and this function contains:
  - Instances of top-level design modules (which, in turn, may contain instances of lower-level modules, and so on) and a testbench (with stimulus and checking).
  - Interconnections between modules using positional or named associations.
  - Specification of one or more clock signals using **`sc_clock`**.
  - Simulation control commands (start time, stop time, etc.) using **`sc_start`**.

火龙果 · 整理
uml.org.cn

Top-Level Module Example

```
// include the necessary header files
…

int sc_main(int argc, char* argv[]) {
  // instantiate and connect the top-level
  // modules and testbench
  …
  sc_clock … ( … ); // specify clock parameters
  sc_start( … );    // specify sim. controls
  return(0);
}
```

火龙果 · 整理
uml.org.cn

Transaction-Level Model (TLM)

- A higher-level event-based approach for modeling a set of functional blocks (processors, memories, etc.) communicating with each other over a bus or a network.
  - The bit-level details of the communication between blocks is abstracted away.
  - Blocks communicate with each other via transaction-oriented functions such as **`read()`** and **`write()`**.
  - The transactions may or may not be clock-cycle accurate, depending on the degree of detail that is desired in the simulations.
  - The major benefit of using TLMs is much higher simulation speed so that more extensive verification suites can be run.

火龙果 · 整理
uml.org.cn

TLM - Continued

- Once the functionality of a TLM has been verified, the design can be refined to the register-transfer level by:
  - Using adaptors to provide pin-accurate/cycle-accurate interfaces
  - Converting abstract data types into SystemC data types such as **`sc_logic`** (a 4-valued logic signal that is 0, 1, X or Z).
- The original TLM-level testbench can be kept as the design is refined to an RTL model.
  - The TLM of the design under verification (DUV) serves as the golden model for comparison purposes.

火龙果 · 整理
uml.org.cn

SystemC Verification (SCV) Library

- Originated and adapted from the TestBuilder verification library developed by Cadence.
- Uses the C++ notion of *smart pointers* to create input streams of arbitrary types.
  - The scv_smart_pointer allows *introspection* to be used
  - This means that generally applicable test generation methods can be applied to arbitrary data types.

火龙果・整理
uml.org.cn

SCV Library – Cont.

- The library supports several types of stimulus generation methods:
  - *directed tests*: using a user-specified set of test vectors
  - *random tests*: the system generates pseudo-randomly generated test vectors:
    - constraints may be used to keep the vectors within specified ranges
    - weighting may be used to generate more vectors in interesting ranges

火龙果 · 整理
uml.org.cn

SCV Library – Cont.

- Several built-in functions available to control stimulus generation:
  - **`next()`** creates a uniformly distributed set of random values of the desired type
  - **`keep_only()`** is used to specify an allowed range of values
  - **`keep_out()`** is used to specify a disallowed range of values
- A built-in data type called **`scv_bag`** can also be used to specify the relative weightings of certain input values.
  - Create a “bag” object.
  - Then, add randomly generated elements into the bag using desired percentage amounts.

火龙果 · 整理
uml.org.cn

Working with SystemC

- The SystemC libraries are available for download from:

  `http://www.systemc.org`

- For these lectures, I will be using the following version of SystemC:

  `systemc-2.2.0.`

  running under linux.

火龙果 · 整理
uml.org.cn

Recall the Half Adder Header File

- A header file **HalfAdder.h** contains the following:

```
#include "systemc.h"
SC_MODULE(HalfAdder) {
        sc_in<bool> a, b;
        sc_out<bool> s, c;
        void HalfAdderProcess();
        SC_CTOR(HalfAdder) {
                SC_METHOD(HalfAdderProcess);
                sensitive << a << b;
        }
};
```

火龙果 · 整理
uml.org.cn

Recall the Half Adder Program File

- A program file **`HalfAdder.cpp`** specifies the details of the process declared in the header file. It contains the following:

```
#include "HalfAdder.h"

void HalfAdder::HalfAdderProcess() {
        s = a ^ b;
        c = a & b;
}
```

火龙果 · 整理
uml.org.cn

Using Instances in a Design Hierarchy

- We can construct a hierarchical description of a full adder by using two instances of the half adder module, together with a process to perform an OR function:

Ref: J. Bhasker, *A SystemC Primer*, Star Galaxy Publishing, 2002.

火龙果 · 整理
uml.org.cn

Full Adder Header File: FullAdder.h

```
#include "HalfAdder.h"
SC_MODULE(FullAdder) {
    sc_in<bool> a, b, cin;
    sc_out<bool> s, cout;
    sc_signal<bool> x, y, z;
    void OrProcess();
    HalfAdder *ha1, *ha2;
    SC_CTOR(FullAdder) {
        ha1 = new HalfAdder("ha1"); ha2 = new HalfAdder("ha2");
        ha1->a(a); ha1->b(b); ha1->s(x); ha1->c(y);
        ha2->a(x); ha2->b(cin); ha2->s(s); ha2->c(z);
        SC_METHOD(OrProcess); sensitive << y << z;
    }
    ~FullAdder() {
        delete ha1; delete ha2;
    }
};
```

火龙果 · 整理
uml.org.cn

Full Adder Program File: FullAdder.cpp

```
#include "FullAdder.h"

void FullAdder::OrProcess() {
        cout = y | z;
}
```

火龙果 · 整理
uml.org.cn

Testbench Structure

- One possible way to organize a testbench is to create a driver module (for generating the input patterns) and a monitor module (for analyzing, printing or recording the output patterns.)
- These would be instantiated at the top level function, which is always called **sc_main**, and has the format:

```
int sc_main(int argc, char* argv[]) {
...
return(0);
}
```

Ref: J. Bhasker, *A SystemC Primer*, Star Galaxy Publishing, 2002.

火龙果 · 整理
uml.org.cn

Files Used in the Simulation

- We will build up a simulation for the full adder design using the following set of files:
- The design-under-test (DUT):
  - **`HalfAdder.h, HalfAdder.cpp`**
  - **`FullAdder.h, FullAdder.cpp`**
- The driver for the testbench:
  - **`Driver.h, Driver.cpp`**
- The monitor for the testbench:
  - **`Monitor.h, Monitor.cpp`**
- The top-level main program:
  - **`FullAdderMain.cpp`**

Ref: J. Bhasker, *A SystemC Primer*, Star Galaxy Publishing, 2002.

火龙果 · 整理
uml.org.cn

Driver Header File: Driver.h

```
#include "systemc.h"

SC_MODULE(Driver) {
        sc_out<bool> Da, Db, Dcin;

        void DriverProcess();

        SC_CTOR(Driver) {
                SC_THREAD(DriverProcess);
        }
};
```

火龙果 · 整理
uml.org.cn

Driver Program File: Driver.cpp

```
#include "Driver.h"

void Driver::DriverProcess() {
        sc_uint<3>  InputVector;
        InputVector = 0;

        while(1) {
                Da   = InputVector[2];
                Db   = InputVector[1];
                Dcin = InputVector[0];
                wait(10, SC_NS);
                InputVector++;
        }
}
```

火龙果 · 整理
uml.org.cn

Monitor Header File: Monitor.h

```
#include "systemc.h"

SC_MODULE(Monitor) {
    sc_in<bool> Ma, Mb, Mcin, Ms, Mcout;

    void MonitorProcess();

    SC_CTOR(Monitor) {
        SC_METHOD(MonitorProcess);
        sensitive << Ma << Mb << Mcin << Ms << Mcout;
    }
};
```

火龙果·整理
uml.org.cn

Monitor Program File: Monitor.cpp

```
#include "Monitor.h"

void Monitor::MonitorProcess() {
        cout << "Time is: " << sc_time_stamp();
        cout << " Inputs are: " << Ma << Mb << Mcin;
        cout << " Sum is: " << Ms;
        cout << " Carry out is: " << Mcout << endl;
}
```

火龙果 · 整理
uml.org.cn

Main Program File: FullAdderMain.cpp

```
#include "FullAdder.h"
#include "Driver.h"
#include "Monitor.h"

int sc_main(int argc, char* argv[]) {
        sc_signal<bool> A, B, CIN, S, COUT;

        // instantiate the DUT
        FullAdder a1("DUT");
        a1.a(A); a1.b(B); a1.cin(CIN);
        a1.s(S); a1.cout(COUT);

        // instantiate the DriverBlock
        Driver d1("DriverBlock");
        d1.Da(A); d1.Db(B); d1.Dcin(CIN);
```

火龙果 · 整理
uml.org.cn

Main Program File – Cont.

```
    //instantiate the MonitorBlock
    Monitor m1("MonitorBlock");
    m1.Ma(A); m1.Mb(B); m1.Mcin(CIN);
    m1.Ms(S); m1.Mcout(COUT);

    // stop the simulation after 90 ns
    sc_start(90, SC_NS);

    // we're done
    return(0);
}
```

火龙果 · 整理
uml.org.cn

Linux Make File Used: MakeFullAdder

```
TARGET_ARCH = linux

CC = g++
OPT = -O3
DEBUG = -g
OTHER = -Wall
CFLAGS = $(OPT) $(OTHER)

MODULE = FullAdder.run
SRCS = FullAdderMain.cpp \
       HalfAdder.cpp     \
       FullAdder.cpp     \
       Driver.cpp        \
       Monitor.cpp
OBJS = $(SRCS:.cpp=.o)
include Makefile.defs
```

火龙果 · 整理
uml.org.cn

Linux Procedure to Compile and Run

- Compile the code using the command:

```
make -f MakeFullAdder
```

- Execute the code using the command:

```
./FullAdder.run.x
```

- The output produced is shown on the following page. Note that there are generally several lines of output at each time value. This is because the sensitivity list in the Monitor causes a line to be printed each time an input or output changes. These changes may be at the same simulation time but still treated as sequential events due to the "delta delay."

火龙果 · 整理
uml.org.cn

```
Time is: 0 s Inputs are: 000 Sum is: 0 Carry out is: 0
Time is: 10 ns Inputs are: 001 Sum is: 0 Carry out is: 0
Time is: 10 ns Inputs are: 001 Sum is: 1 Carry out is: 0
Time is: 20 ns Inputs are: 010 Sum is: 1 Carry out is: 0
Time is: 20 ns Inputs are: 010 Sum is: 0 Carry out is: 0
Time is: 20 ns Inputs are: 010 Sum is: 1 Carry out is: 0
Time is: 30 ns Inputs are: 011 Sum is: 1 Carry out is: 0
Time is: 30 ns Inputs are: 011 Sum is: 0 Carry out is: 0
Time is: 30 ns Inputs are: 011 Sum is: 0 Carry out is: 1
Time is: 40 ns Inputs are: 100 Sum is: 0 Carry out is: 1
Time is: 40 ns Inputs are: 100 Sum is: 1 Carry out is: 1
Time is: 40 ns Inputs are: 100 Sum is: 1 Carry out is: 0
Time is: 50 ns Inputs are: 101 Sum is: 1 Carry out is: 0
Time is: 50 ns Inputs are: 101 Sum is: 0 Carry out is: 0
Time is: 50 ns Inputs are: 101 Sum is: 0 Carry out is: 1
Time is: 60 ns Inputs are: 110 Sum is: 0 Carry out is: 1
Time is: 60 ns Inputs are: 110 Sum is: 1 Carry out is: 1
Time is: 60 ns Inputs are: 110 Sum is: 0 Carry out is: 1
Time is: 70 ns Inputs are: 111 Sum is: 0 Carry out is: 1
Time is: 70 ns Inputs are: 111 Sum is: 1 Carry out is: 1
Time is: 80 ns Inputs are: 000 Sum is: 1 Carry out is: 1
Time is: 80 ns Inputs are: 000 Sum is: 0 Carry out is: 1
Time is: 80 ns Inputs are: 000 Sum is: 0 Carry out is: 0
```

火龙果 · 整理
uml.org.cn

SystemC Module Structure

- The basic unit of design is the module:
  - **SC_MODULE**.
- Each module contains a constructor
  - **SC_CTOR**.
  - The constructor gives the names of the processes contained in the module, along with their sensitivity lists.
  - It can also contain instances of other modules.
- Each process is specified to be one of two kinds, i.e. a method or a thread:
  - **SC_METHOD**
  - **SC_THREAD**

火龙果 · 整理
uml.org.cn

Example Outline of a Module Specification

```
SC_MODULE (module_name) {
   // declare inputs, outputs, signals and variables
   // declare processes (return type must be void)
   // point to lower-level modules
   SC_CTOR (module_name ) {
       // list all of the method and thread processes
       SC_METHOD(process_name);
       // sensitivity list for this process
       SC_METHOD(process_name);
       // sensitivity list for this process
       SC_THREAD(process_name);
       // sensitivity list for this process
   }
};
```

火龙果 · 整理
uml.org.cn

A Module Destructor

- A destructor is used when the constructor for a module contains the **new** operator.
  - The **new** operator is used to create instances of other modules in the current module.
  - The **new** operator allocates memory.
  - The destructor is used in order to avoid a "memory leak."
- The format for the destructor is to use the ~ character in front of the name of the module.

火龙果・整理
uml.org.cn

A Module Destructor – Cont.

- For example, if the constructor created pointer_1, pointer_2, ..., then the destructor would be as follows:

```
~module_name() {
  delete pointer_1;
  delete pointer_2;
  ...
}
```

火龙果 · 整理
uml.org.cn

Modeling Logic with SC_METHOD

- An **`SC_METHOD`** process can be used to model either combinational logic or sequential logic.
- To obtain a combinational logic design, use a sensitivity list with the keyword **`sensitive`** which includes the names of all of the input signals
- To obtain a sequential logic design, use a a sensitivity list with the keyword **`sensitive_pos`** or **`sensitive_neg`**, which models rising or falling edge triggered behavior, respectively.
- The format for a sensitivity list uses C++ stream notation:
  - For example:

```
sensitive << a << b;
```

火龙果 · 整理
uml.org.cn

Modeling with SC_THREAD

- A thread process, using **SC_THREAD**, is more general than a method process.
- The major difference is that the thread process can be suspended.
  - This is accomplished by using one or more **`wait()`** statements within the thread process.
- Thread process may have level-sensitive and/or edge-sensitive types of sensitivity list(s).

火龙果 · 整理
uml.org.cn

References

- Open SystemC Initiative (OSCI), http://www.systemc.org
- IEEE Standard SystemC® Language Reference Manual, IEEE, IEEE Std 1666™-2005, 31 March 2006.
- J. Bhasker, *A SystemC Primer*, Star Galaxy Publishing, 2002.
- SystemC Tutorial, Doulos Ltd., http://www.doulos.com/knowhow/systemc/tutorial/
- Getting Started with TLM-2.0, Doulos Ltd., http://www.doulos.com/knowhow/systemc/tlm2/
- SystemC：An Introduction for beginners, electroSofts.com, http://electrosofts.com/systemC/index.html

火龙果 · 整理
uml.org.cn