EEL 5722C
Field-Programmable Gate Array Design

Lecture 14: Introduction to SystemC*

Prof. Mingjie Lin

* SystemC Tutorial, Silvio Veloso

Outline

- Needed tools
- Starting example
- Introduction
- SystemC highlights
- Differences
- Modules, processes, ports, signals, clocks and data types

Needed tools

- SystemC library package v2.0.1
  Download in www.systemc.org
- Linux platform
- GCC compiler
- GTKWave – Waveform tool
- some text editor

Install SystemC

See Course Webpage

Starting Example:Full Adder

FullAdder.h

```
SC_MODULE( FullAdder ) {

   sc_in< sc_uint<16> >   A;
   sc_in< sc_uint<16> >   B;
   sc_out< sc_uint<17> > result;

   void doIt( void );

   SC_CTOR( FullAdder ) {

      SC_METHOD( doIt );
         sensitive << A;
         sensitive << B;
   }

};
```

FullAdder.cpp

```
void FullAdder::doIt( void ) {
   sc_int<16> tmp_A, tmp_B;
   sc_int<17> tmp_R;

   tmp_A = (sc_int<16>) A.read();
   tmp_B = (sc_int<16>) B.read();

   tmp_R = tmp_A + tmp_B;

   result.write( (sc_uint<16>) tmp_R.range(15,0) );
}
```

Introduction

- *What is SystemC ?*
  - SystemC is a C++ class library and methodology that can effectively be used to create a cycle-accurate model of a system consisting of software, hardware and their interfaces.

Introduction

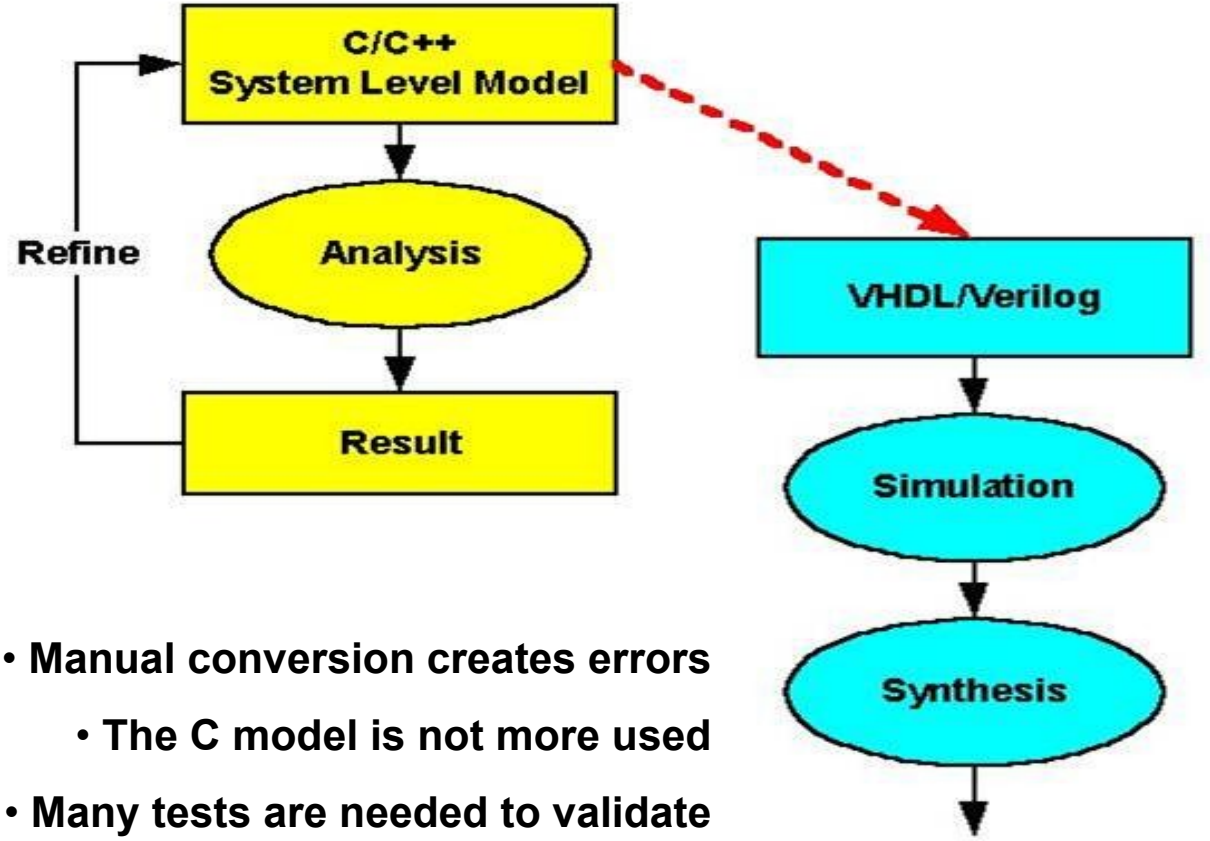
- *Where can I use SystemC ?*
  - In creating an executable specification of the system to be developed.
- *What should I know to learn SystemC ?*
  - Notions of C++ programming and VHDL helps you a lot.

SystemC highlights

- Supports hardware and software co-design
- Developing an executable specification avoids inconsistency and errors
- Avoids wrong interpretation of the specification
- SystemC has a rich set of data types for you to model your systems
- It allows multiple abstraction levels, from high level design down to cycle-accurate RTL level

Why is SystemC different ?

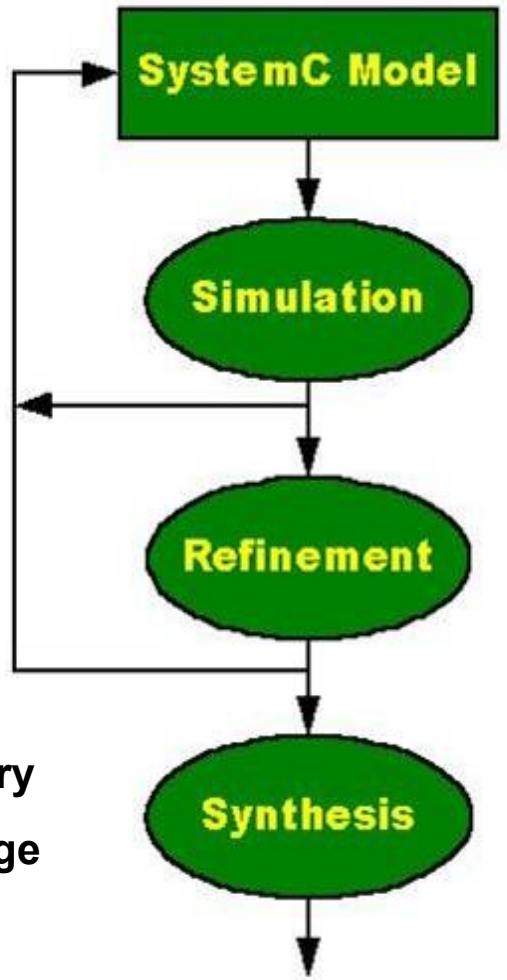
- Current design methodology



- **Manual conversion creates errors**
- **The C model is not more used**
- **Many tests are needed to validate**

Why is SystemC different ?

- SystemC design methodology



•Better methodology, translate is not necessary

•Written in only one language

Modules

- Modules are the basic building blocks to partition a design
- Modules allow to partition complex systems in smaller components
- Modules hide internal data representation, use interfaces
- Modules are classes in C++
- Modules are similar to „*entity*“ in VHDL

Modules

```
SC_MODULE(module_name)
{
    // Ports declaration
    // Signals declaration
    // Module constructor : SC_CTOR
    // Process constructors and sensibility list
    //          SC_METHOD
    // Sub-Modules creation and port mappings
    // Signals initialization
}
```

They can contain ports, signals, local data, other modules, processes and constructors.

Modules

- Module constructor
- Similar to „*architecture*“ in VHDL

Example: Full Adder constructor

```
SC_CTOR( FullAdder ) {

    SC_METHOD( doIt );
      sensitive << A;
      sensitive << B;
  }
```

Modules

- Sub-modules instantiation:
- Instantiate module

  Module_type Inst_module (“*label*”);

- Instantiate module as a pointer

  *Module_type *pInst_module*;

  // Instantiate at the module constructor SC_CTOR

  pInst_module = new *module_type* (“*label*”);

Modules

- How to connect sub-modules ?
  - Named Connection or
  - Positional Connection

Modules

- Named Connection

```
Inst_module.a(s);
Inst_module.b(c);
Inst_module.q(q);

pInst_module -> a(s);
pInst_module -> b(c);
pInst_module -> q(q);♪
```

Modules

- Positional Connection

```
Inst_module << s << c << q;

(*pInst_module)(s,c,q);
```

Modules

- Internal Data Storage
- Local variables: can not be used to connect ports
- Allowed data types
  - C++ types
  - SystemC types
  - User defined types

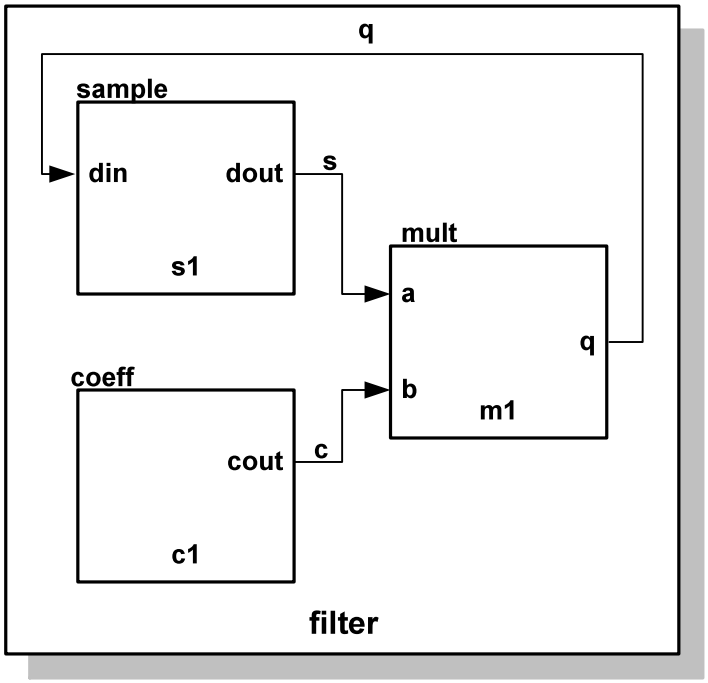
Modules

- Example: Mux 2:1

```
SC_MODULE( Mux21 ) {

   sc_in< sc_uint<8> >   in1;
   sc_in< sc_uint<8> >   in2;
   sc_in< bool >       selection;
   sc_out< sc_uint<8> >  out;

   void doIt( void );

   SC_CTOR( Mux21 ) {

      SC_METHOD( doIt );
         sensitive << selection;
         sensitive << in1;
         sensitive << in2;

   }

};
```

Modules

- Example:




```
SC_MODULE(filter) {
        // Sub-modules : "components"
        sample *s1;
        coeff *c1;
        mult *m1;

        sc_signal<sc_uint 32> > q, s, c; // Signals

        // Constructor : "architecture"
        SC_CTOR(filter) {

            // Sub-modules instantiation and
          mapping
            s1 = new sample ("s1");
            s1->din(q); // named mapping
            s1->dout(s);

            c1 = new coeff("c1");
            c1->out(c); // named mapping

            m1 = new mult ("m1");
            (*m1)(s, c, q); // Positional mapping

        }
}
```

Processes

- Processes are functions that are identified to the SystemC kernel. They are called if one signal of the sensitivity list changes its value.
- Processes implement the funcionality of modules
- Processes are very similar to a C++ function or method
- Processes can be Methods, Threads and CThreads

Processes

- Methods

When activated, executes and returns

- SC_METHOD(process_name)

- Threads

Can be suspended and reactivated

- wait() -> suspends
- one sensitivity list event -> activates
- SC_THREAD(process_name)

- CThreads

Are activated in the clock pulse

- SC_CTHREAD(process_name, clock value);

Processes

Type	SC_METHOD	SC_THREAD	SC_CTHREAD
Activates Exec.	Event in sensit. list	Event in sensit. List	Clock pulse
Suspends Exec.	NO	YES	YES
Infinite Loop	NO	YES	YES
suspended/ reactivated by	N.D.	wait()	wait() wait_until()

Constructor & Sensibility definition	SC_METHOD(*call_back*);
	sensitive(*signals*);
	sensitive_pos(*signals*);
	sensitive_neg(*signals*);
		SC_THREAD(*call_back*);
		sensitive(*signals*);
		sensitive_pos(*signals*);
		sensitive_neg(*signals*);
			SC_CTHREAD(
			call_back,
			clock.pos());
			SC_CTHREAD(
			call_back,
			clock.neg());

Processes

- Process Example

Into the .H file

```
void doIt( void );

   SC_CTOR( Mux21 ) {

      SC_METHOD( doIt );
         sensitive << selection;
         sensitive << in1;
         sensitive << in2;

   }
```

Into the .CPP file

```
void Mux21::doIt( void ) {

   sc_uint<8> out_tmp;

   if( selection.read() ) {
      out_tmp = in2.read();
   } else {
      out_tmp = in1.read();
   }

   out.write( out_tmp );
}
```

Ports and Signals

- Ports of a module are the external interfaces that pass information to and from a module
- In SystemC one port can be *IN*, *OUT* or *INOUT*
- Signals are used to connect module ports allowing modules to communicate
- Very similar to ports and signals in VHDL

Ports and Signals

- Types of ports and signals:
  - All natives C/C++ types
  - All SystemC types
  - User defined types
- How to declare
  - IN : sc_in<***port_typ***>
  - OUT : sc_out<***port_type***>
  - Bi-Directional : sc_inout<***port_type***>

Ports and Signals

- How to read and write a port ?
  - Methods *read( );* and *write( );*
- Examples:
  - in_tmp = in.read( ); //reads the port in to in_tmp
  - out.write(out_temp); //writes out_temp in the out port

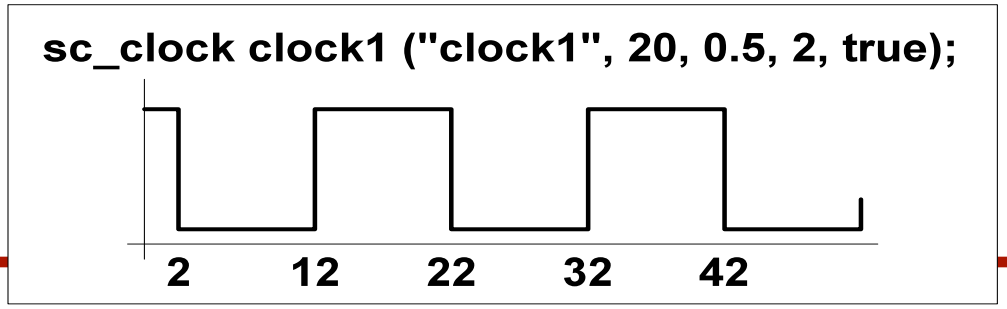
Clocks

- Special object
- How to create ?

  sc_clock ***clock_name*** (

  "***clock_label***", ***period***, ***duty_ratio***, ***offset***, ***initial_value***);
- Clock connection

  f1.clk(clk_signal); //where f1 is a
- Clock example:



Hello World!

```
1  // All systemc modules should include systemc.h header file
2  #include "systemc.h"
3  // Hello_world is module name
4  SC_MODULE (hello_world) {
5    SC_CTOR (hello_world) {
6      // Nothing in constructor
7    }
8    void say_hello() {
9      //Print "Hello World" to the console.
10     cout << "Hello World.\n";
11   }
12 };
13
14 // sc_main in top level function like in C++ main
15 int sc_main(int argc, char* argv[]) {
16   hello_world hello("HELLO");
17   // Print the hello world
18   hello.say_hello();
19   return(0);
20 }
```

counter

```
1  //-----------------------------------------------
2  // This is my second Systemc Example
3  // Design Name : first_counter
4  // File Name : first_counter.cpp
5  // Function : This is a 4 bit up-counter with
6  // Synchronous active high reset and
7  // with active high enable signal
8  //-----------------------------------------------
9  #include "systemc.h"
10
11 SC_MODULE (first_counter) {
12   sc_in_clk     clock ;       // Clock input of the design
13   sc_in<bool>   reset ;       // active high, synchronous Reset input
14   sc_in<bool>   enable;       // Active high enable signal for counter
15   sc_out<sc_uint<4> > counter_out; // 4 bit vector output of the counter
16
17   //------------Local Variables Here---------------------
18   sc_uint<4>        count;
19
20   //------------Code Starts Here-------------------------
21   // Below function implements actual counter logic
22   void incr_count () {
23     // At every rising edge of clock we check if reset is active
24     // If active, we load the counter output with 4'b0000
25     if (reset.read() == 1) {
26       count = 0;
27       counter_out.write(count);
28     // If enable is active, then we increment the counter
29     } else if (enable.read() == 1) {
30       count = count + 1;
31       counter_out.write(count);
32       cout<<"@" << sc_time_stamp() <<" :: Incremented Counter "
33         <<counter_out.read()<<endl;
34     }
35   } // End of function incr_count
36
37   // Constructor for the counter
38   // Since this counter is a positive edge trigged one,
39   // We trigger the below block with respect to positive
40   // edge of the clock and also when ever reset changes state
41   SC_CTOR(first_counter) {
42     cout<<"Executing new"<<endl;
43     SC_METHOD(incr_count);
44     sensitive << reset;
45     sensitive << clock.pos();
46   } // End of Constructor
47
48 }; // End of Module counter
```

Final issues

- Come by my office hours (right after class)
- Any questions or concerns?