EEL 5722C
Field-Programmable Gate Array Design

Lecture 17: Describing Synthesizable RTL in SystemC*

Prof. Mingjie Lin

System-Level Design

- Specifying the system
- Verifying its functionality
- Determining optimum system architecture by evaluating design alternatives

- Today's complex systems have significant software content and are integrated into a system on a chip (SoC).

System-Level Design Challenges

- Component Integration
  - widely implemented strategy for handling complexity, but many of the components are provided from various sources.
- Tool Interoperability
  - each tool uses a proprietary model format, which makes a model developed for one tool unsuitable for use with another tool
- Design Team Collaboration
  - Effective architecture design requires participation of the hardware and software design teams for creating models and influencing architectural decisions, which is called hardware-software co-design

Why Synthesis From SystemC?

- Imagine you are a hardware designer, you are given a architectural model contains a variety of models, including processor models, abstract bus models, and peripheral models. You need to implement the peripherals and verify the implementation in the context of the entire system.
- If you use a Verilog or VHDL synthesis tool, you need to rewrite the peripheral models in Verilog or VHDL, which is a time-consuming and error-prone process

4

Why Synthesis From SystemC?

- Or you can synthesize the peripheral models from SystemC
  - Instead of throwing away the work done at the system level and recoding the design, you can take the abstract, non-synthesizable peripheral models and refine them into synthesizable models
- Verification Reuse
  - use the system-level verification environment to check the correctness of your implementation as you refine it.

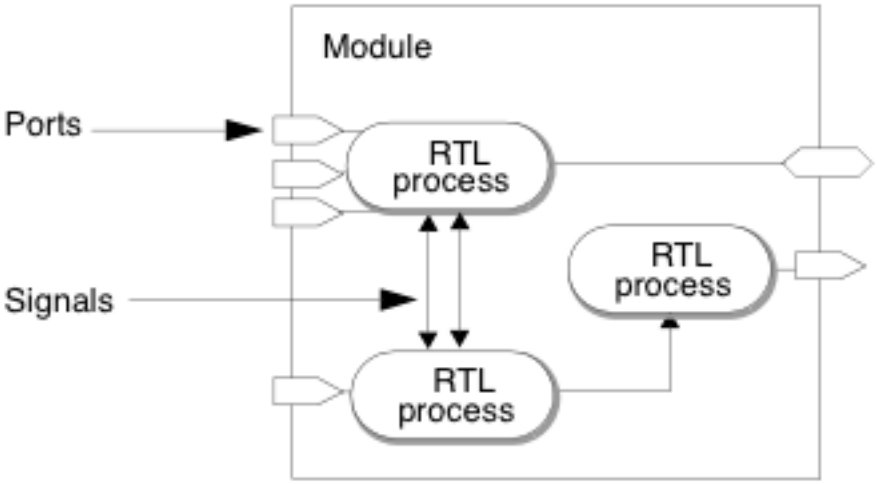
5

Defining Modules and Processes

- The basic building block in SystemC is the module
- A SystemC module is a container in which processes and other modules are instantiated
- A typical module can have:
  - Single or multiple RTL processes to specify combinational or sequential logic
  - Multiple RTL modules to specify hierarchy
  - One or more member functions that are called from within an instantiated process or module

6

Module Example



- Processes
  - describe the parallel behavior of hardware systems
  - processes execute concurrently rather than sequentially like C++ functions
  - The code within a process, however, executes sequentially

  Registering a Process
- Triggering Execution of a Process
- Reading and Writing Processes
- Types of Processes

Creating a Module

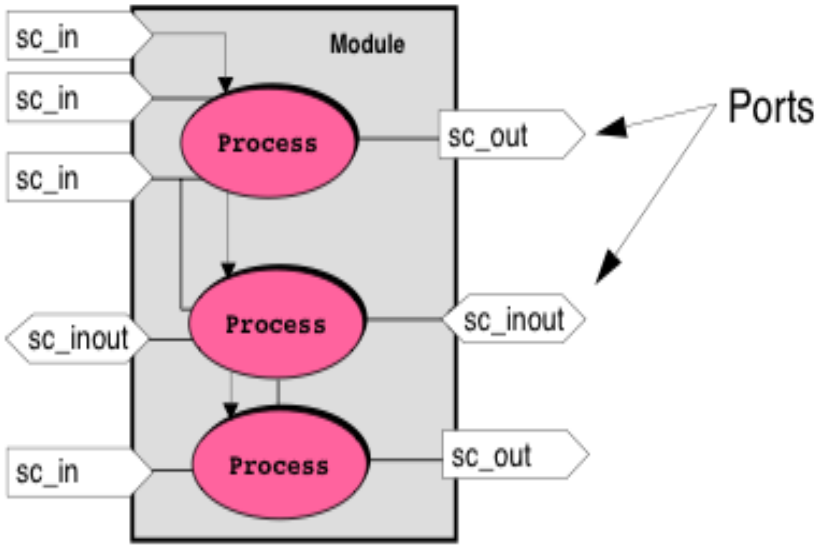
```
#include "systemc.h"
SC_MODULE (module_name) {
     //Module port declarations
     //Signal variable declarations
     //Data variable declarations
     //Member function declarations
     //Method process declarations

     //Module constructor
     SC_CTOR (module_name) {
               //Register processes
               //Declare sensitivity list
     }
};
```

- Coding practice
  - Separate header file (module_name.h)
  - Implementation file (module_name.cpp or module_name.cc)

Module Header File

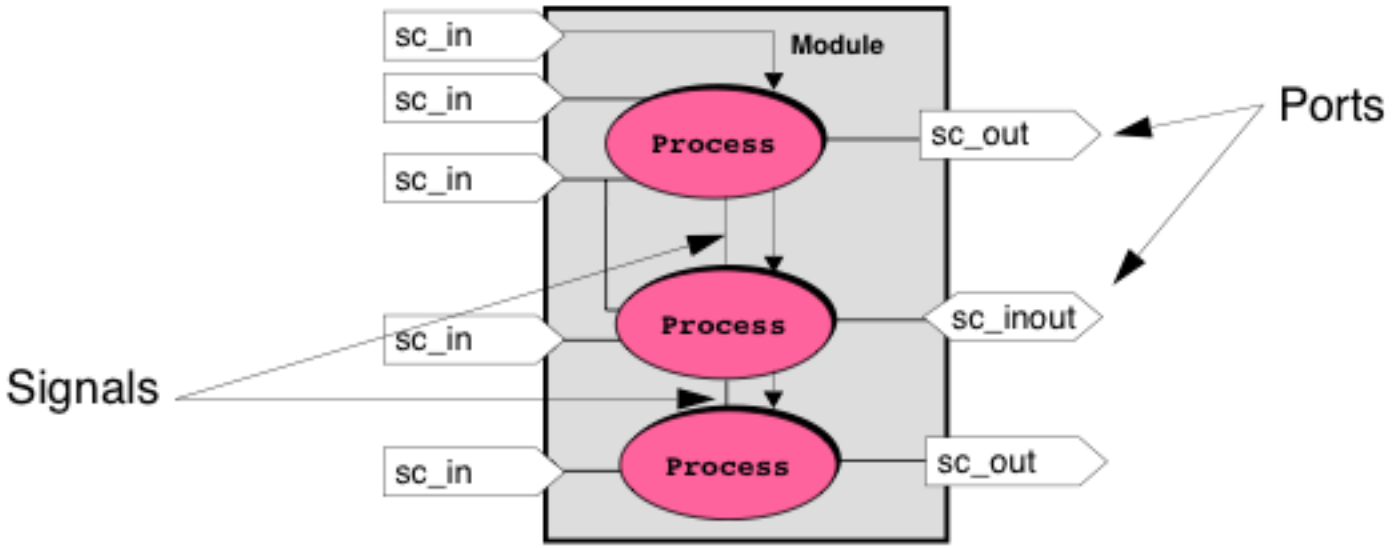
- Port declarations
- Internal signal variable declarations
- Internal data variable declarations
- Process declarations
- Member function declarations
- Module constructor

Module Ports




```
SC_MODULE (module_name) {
     //Module port declarations
     sc_in<port_data_type> port_name;
     sc_out<port_data_type> port_name;
     sc_inout<port_data_type> port_name;
     sc_in<port_data_type> port_name;


     //Module constructor
     SC_CTOR (module_name) {
               //Register processes
               //Declare sensitivity list
     }
};
```

Signals



- Modules use ports to communicate with other modules
  - In hierarchical modules, use signals to communicate between the ports of instantiated modules
  - internal signals for peer-to-peer communication between processes within the same module

signal Syntax

```
SC_MODULE (module_name) {
     //Module port declarations
     sc_in<port_type> port_name;
     sc_out<port_type> port_name;
     sc_in<port_type>port_name;

     //Internal signal variable declarations
     sc_signal<signal_type> signal_name;
     sc_signal<signal_type> signal1, signal2;

     //Data variable declarations
     //Process declarations
     //Member function declarations

     //Module constructor
     SC_CTOR (module_name) {
               //Register processes
               //Declare sensitivity list
     }
};
```

Data Member Variables

```
SC_MODULE (module_name) {
     //Module port declarations
     sc_in<port_type> port_name;
     sc_out<port_type> port_name;
     sc_in port_name;

     //Internal signal variable declarations
     sc_signal<signal_type> signal_name;

     //Data member variable declarations
     int count_val;          //Internal counter
     sc_int<8> mem[1024];  //Array of sc_int

     //Process declarations
     //Member function declaration

     //Module constructor
     SC_CTOR (module_name) {
               //Register processes
               //Declare sensitivity list
     }
};
```

- Do not use data variables for peer-to-peer communication in a module. This can cause pre- and post-synthesis simulation mismatches and nondeterminism (order dependency) in your design.

Creating a Process in a Module

```
SC_MODULE(my_module){
     // Ports
     sc_in<int> a;
     sc_in<bool> b;
     sc_out<int> x;
     sc_out<int> y;
     // Internal signals
     sc_signal<bool>c;
     sc_signal<int> d;
     // process declaration
     void my_method_proc();
        // module constructor
        SC_CTOR(my_module) {
           // register process
           SC_METHOD(my_method_proc);
           // Define the sensitivity list
     }
};
```

Defining the Sensitivity List

- Defining a Level-Sensitive Process

```
SC_MODULE(my_module){
    // Ports
    sc_in<int> a;
    sc_in<bool> b;
    sc_out<int> x;
    sc_out<int> y;
    // Internal signals
    sc_signal<bool>c;
    sc_signal<int> d;
    sc_signal<int> e;
    // process declaration
    void my_method_proc();
    // module constructor
    SC_CTOR(my_module) {
        // register process
        SC_METHOD(my_method_proc);
        // declare level-sensitive sensitivity list
        sensitive << a << c << d; // Stream declaration
        sensitive(b);  //Function declaration
        sensitive(e);  //Function declaration
    }
};
```

Defining the Sensitivity List

- Defining a Edge-Sensitive Process

```
SC_MODULE(my_module){
     // Ports
     sc_in<int> a;
     sc_in<bool> b;
     sc_in<bool> clock;
     sc_out<int> x;
     sc_out<int> y;
     sc_in<bool> reset;
     // Internal signals
     sc_signal<bool>c;
     sc_signal<int> d;
     // process declaration
     void my_method_proc();
     // module constructor
     SC_CTOR(my_module) {
          // register process
          SC_METHOD(my_method_proc);
          // declare sensitivity list
          sensitive_pos (clock); //Function delaration
          sensitive_neg << b << reset; // Stream declaration
     }
};
```

Limitations for Sensitivity Lists

- You cannot specify both edge-sensitive and level-sensitive inputs in the same process for synthesis.

- You cannot declare an sc_logic type for the clock or other edge-sensitive inputs. You can declare only an sc_in<bool> data type.

Module Constructor

- Register processes
- Define a sensitivity list for an SC_METHOD process

Implementing the Module

```
#include "systemc.h"
#include "my_module.h"
void my_module::my_method_proc() {
   // describe process functionality as C++ code
}
```

Reading and Writing Ports and Signals

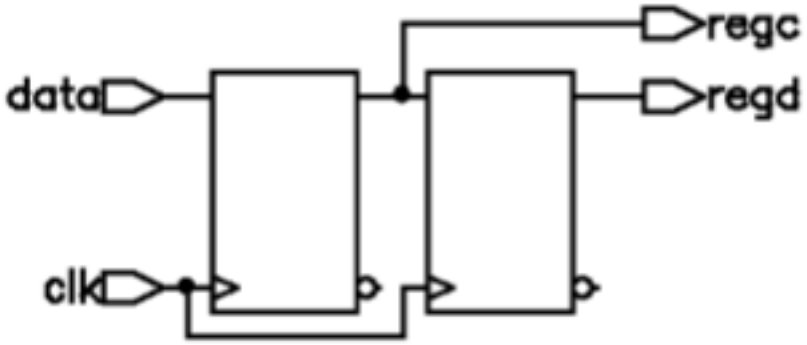
```
//...
// read method
address = into.read();        // get address
// assignment
temp1 = address;              // save address
data_tmp = memory[address]; // get data from memory
// write method
outof.write(data_tmp);        // write out
// assignment
temp2 = data_tmp;             // save data_tmp
//...
```

Reading and Writing Bits of Ports and Signals

```
//...
sc_signal <sc_int<8> > a;
sc_int<8> b;
bool c;
b = a.read();
c = b[0];

// c = a[0]; /Does not work in SystemC
```

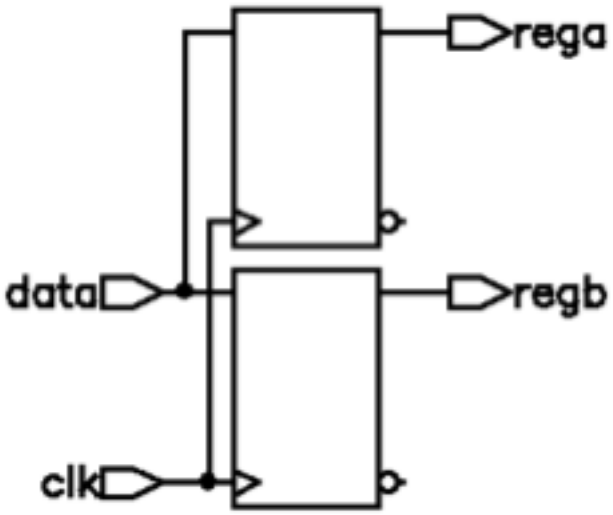
Signal and Port Assignments

```
#include "systemc.h"

SC_MODULE(rtl_nb) {
  sc_in<bool> clk;
  sc_in<bool> data;
  sc_inout<bool> regc, regd;

  void reg_proc() {
    regc.write(data.read());
    regd.write(regc.read());
  }

  SC_CTOR(rtl_nb) {
    SC_METHOD(reg_proc);
    sensitive_pos << clk;
  }
};
```

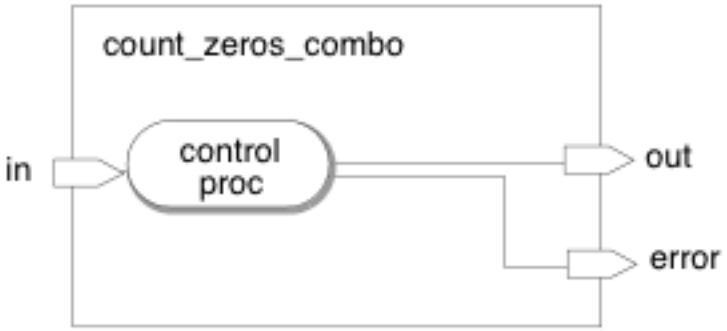


Variable Assignment

```
#include "systemc.h"

SC_MODULE(rtl_b) {
  sc_in<bool> clk;
  sc_in<bool> data;
  sc_out<bool> rega, regb;

  bool rega_v, regb_v;

  void reg_proc() {
    rega_v = data.read();
    regb_v = rega_v;
    rega.write(rega_v);
    regb.write(regb_v);
  }

  SC_CTOR(rtl_b) {
    SC_METHOD(reg_proc);
    sensitive_pos << clk;
  }
};
```

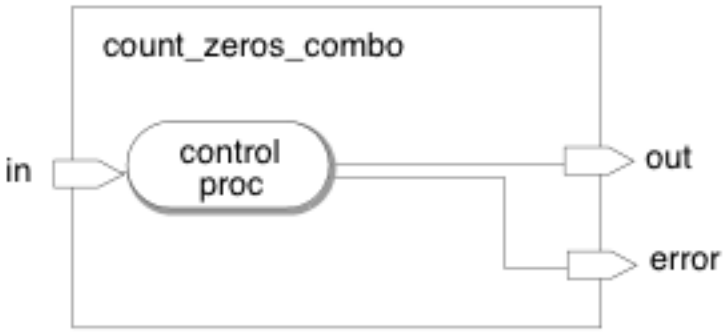


Creating a Module With a Single SC_METHOD Process




```
/****count_zeros_comb.h file***/
#include "systemc.h"

SC_MODULE(count_zeros_comb) {
  sc_in<sc_uint<8> >  in;
  sc_out<sc_uint<4> >  out;
  sc_out<bool> error;

  bool legal(sc_uint<8> x);
  sc_uint<4> zeros(sc_uint<8> x);
  void control_proc();

  SC_CTOR(count_zeros_comb) {
    SC_METHOD(control_proc);
    sensitive << in;
  }
};
```

Creating Module With Single SC_METHOD Process




```
/****count_zeros_comb.cpp file****/
#include "count_zeros_comb.h"

void count_zeros_comb::control_proc() {
  sc_uint<4> tmp_out;
  bool is_legal = legal(in.read());
  error.write(! is_legal);
  is_legal ? tmp_out = zeros(in.read()) : tmp_out = 0;
  out.write(tmp_out);
}

bool count_zeros_comb::legal(sc_uint<8> x) {
  bool is_legal = 1;
  bool seenZero = 0;
  bool seenTrailing = 0;
  for (int i=0; i <=7; ++i) {
    if (seenTrailing && (x[i] == 0)) {
      is_legal = 0;
      break;
    } else if (seenZero && (x[i] == 1)) {
      seenTrailing = 1;
    } else if (x[i] == 0) {
      seenZero = 1;
    }
  }
  return is_legal;
}

sc_uint<4> count_zeros_comb::zeros(sc_uint<8> x) {
  int count = 0;
  for (int i=0; i <= 7; ++i) {
    if (x[i] == 0)
      ++count;
  }
  return count;
}
```

Final issues

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