

---

# Digital Integrated Circuit Design

## Module SysCIntro SystemC: Introduction

W. Rhett Davis

# History of SystemC

---

- **Synopsys** began development on “Scenic” project (1996)
- Version 2.0.1 released on SourceForge.net (May 31, 2000)
- Version 2.1.v1 released on SystemC.org by **OSCI** (Sept. 19, 2005)
- IEEE Standard
  - » IEEE Std. 1666-2005 published (Apr. 10, 2006)
  - » Version 2.2.05 released on SystemC.org (June 6, 2006)
- OSCI Merged with Accellera (Dec. 5, 2011)
- Revision of IEEE Standard
  - » IEEE Std. 1666-2011 published (Jan. 9, 2012)
  - » Version 2.3 released on accellera.org (July 2, 2012)

# Parts of a Verilog Module

---

- Header: *module* <module\_name> (<port\_list>);
- Parameter, Port, & Internal Variable declarations
- Functionality description
  - » Structural
    - Instantiate basic gates
    - Instantiate lower-level modules
  - » Behavioral
    - Data-Flow (continuous assignments)
    - Procedural (initial & always blocks)
- Terminator: *endmodule*

# Parts of a SystemC Module

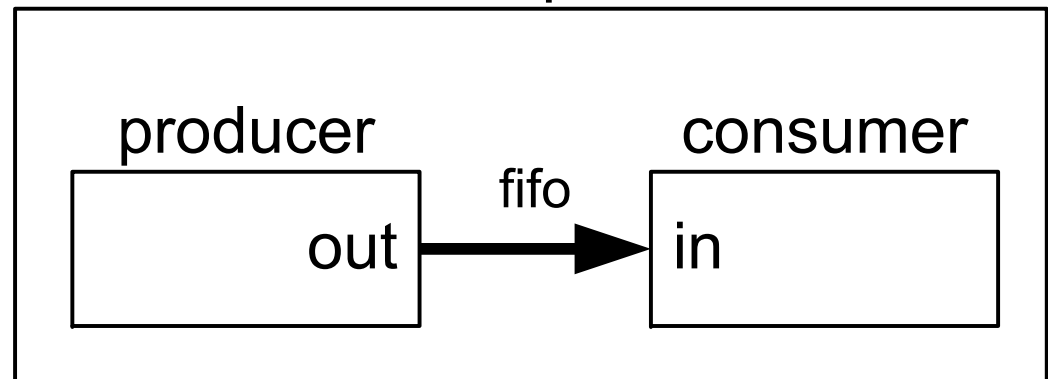
---

- ➔ ● Declaration: `class <module_name>: sc_module`
  - » Port & Internal Variable declarations
  - » Instance & Process declarations
- Constructor
  - » Structural Functionality Description
    - Instantiate lower-level modules, bind ports
  - » Procedural Functionality Description
    - Register thread & method processes with scheduler
- Procedural Functionality Description
  - » Thread & Method Process Definitions

# simple\_fifo Example

---

- Included in the **sc\_tut.tar.gz** file in the Course Resources
- Modified slightly from sec. 7.5 of [Grötter 2002]
- Producer and consumer communicating through a FIFO
  - » Producer endlessly sends the message “Hello, World!\n”
- One level of hierarchy



# Declaring a module in SystemC

---

- Modules in SystemC are classes that are derived from the ***sc\_module*** class

```
class producer : sc_module
{
```

- You may sometimes see the macro SC\_MODULE, which is replaced with the code above at compile time

```
SC_MODULE ( FIR_cascade )
{
```

- » I rarely use this style, because it prevents declaring the sc\_module base class as ***public***

# Declaring Ports

- Ports are member instances of the ***sc\_port*** template class
  - » argument is an interface class that defines the methods used to communicate over the channel
  - » In this case, we'll use the FIFO output interface

```
sc_port<sc_fifo_out_if<char> > out;
```

note the space!

- Interface classes are typically templates themselves, with an argument to represent the type of data that is passed.
  - » In this case, we'll use *char* (8 bits)

# Declaring Internal Variables

---

- Internal Variables (like **wire** and **reg** variables in Verilog) are member instances of other classes
  - » If the variable will be used with procedural descriptions (*i.e.* processes), then the basic data type may be declared.

```
char c;
```

- » If the variable will be used with structural descriptions (*i.e.* instantiations) or sensitivity lists, then it must be a class derived from **sc\_prim\_channel** and have an interface compatible with the port it connects.

```
sc_fifo<char> fifo_inst;
```

- In this case, **sc\_fifo** is derived from **sc\_prim\_channel**, **sc\_fifo\_out\_if**, and **sc\_fifo\_in\_if**.



# Declaring Instances & Processes

---

- Instances of lower-level modules must have member variables to access them

```
producer prod_inst;  
consumer cons_inst;
```

- SystemC processes are like ***always*** blocks in Verilog
  - » They are essentially class methods with no arguments or return value
  - » They must be declared in the class definition

```
void main ( ) ;
```

# Parts of a SystemC Module

---

- Declaration: `class <module_name>: sc_module`
  - » Port & Internal Variable declarations
  - » Instance & Process declarations
- ● Constructor
  - » Structural Functionality Description
    - Instantiate lower-level modules, bind ports
  - » Procedural Functionality Description
    - Register thread & method processes with scheduler
- Procedural Functionality Description
  - » Thread & Method Process Definitions

# Constructor Declaration (3 ways)

---

- (FUNDAMENTAL) Before the constructor is declared, the type `SC_CURRENT_USER_MODULE` is defined to the current module name
  - » This type is used by the process registration macros (described later)

```
typedef producer SC_CURRENT_USER_MODULE;  
producer(sc_module_name)
```

- (SIMPLEST) SystemC defines the macro ***SC\_CTOR*** to represent this code more conveniently

```
SC_CTOR(producer)
```

- (MOST COMMON) SystemC Also defines the macro ***SC\_HAS\_PROCESS*** to represent only the ***typedef*** statement
  - » This is helpful if the constructor needs extra arguments, as is the case with parameterized modules

```
SC_HAS_PROCESS(producer);  
producer(sc_module_name);
```

# Instantiations

- The constructor initializes module and channel instances
  - » The first argument is always the instance name
  - » Some modules or channels may require additional arguments

```
top(sc_module_name name, int size) :  
    sc_module(name),  
    fifo_inst("Fifo1",size),  
    prod_inst("Producer1"),  
    cons_inst("Consumer1")
```

- Ports are **bound** to internal signals by calling the port constructors for the instance, passing channel as argument
  - » Another option is to call “bind” method

```
prod_inst.out(fifo_inst);  
cons_inst.in(fifo_inst);
```

```
prod_inst.out.bind(fifo_inst);  
cons_inst.in.bind(fifo_inst);
```

# Process Registration

---

- Although processes were declared with the class, they must also be registered with the SystemC scheduler.
- SystemC defines the macros SC\_METHOD and SC\_THREAD for registering processes
  - » thread processes are like Verilog **always** blocks **without** a sensitivity list (*i.e.* process may call *wait()*)

```
SC_THREAD(main);
```

- » method processes are like Verilog **always** blocks **with** a sensitivity list (*i.e.* process may *not* call *wait()*)
- » the **sensitive\_pos** function is used to declare the signals in the sensitivity list that trigger the process on their rising edges.

```
SC_METHOD(main);  
sensitive_pos << clock;
```

# Parts of a SystemC Module

---

- Declaration: `class <module_name>: sc_module`
  - » Port & Internal Variable declarations
  - » Instance & Process declarations
- Constructor
  - » Structural Functionality Description
    - Instantiate lower-level modules, bind ports
  - » Procedural Functionality Description
    - Register thread & method processes with scheduler
- ➡ ● Procedural Functionality Description
  - » Thread & Method Process Definitions


# Process Definitions

---

- Finally, write the C++ code to define the processes

```
void main () {  
    const char *str = "Hello, World!\n";  
    const char *p = str;  
  
    while (true) {  
        out->write(*p++);  
        if (!*p) p=str;  
        wait(10, SC_NS);  
    }  
};
```

write method used  
to send a value  
through the fifo  
channel



# Simulating the Design

---

- ***sc\_main*** is the entry-point for user code
- Use ***sc\_start*** to run the simulation for a certain amount of time
  - » If time arguments are omitted, then simulation runs until there are no more events

```
int sc_main (int argc, char *argv[])  
{  
    int size=10;  
  
    top top1("Top1",size);  
    sc_start(100, SC_NS);  
    return 0;  
}
```



---

# Digital Integrated Circuit Design

## Module SysCIntro SystemC: Introduction

W. Rhett Davis

Thanks for watching