SystemC Introduction – Part II

Suryaprasad

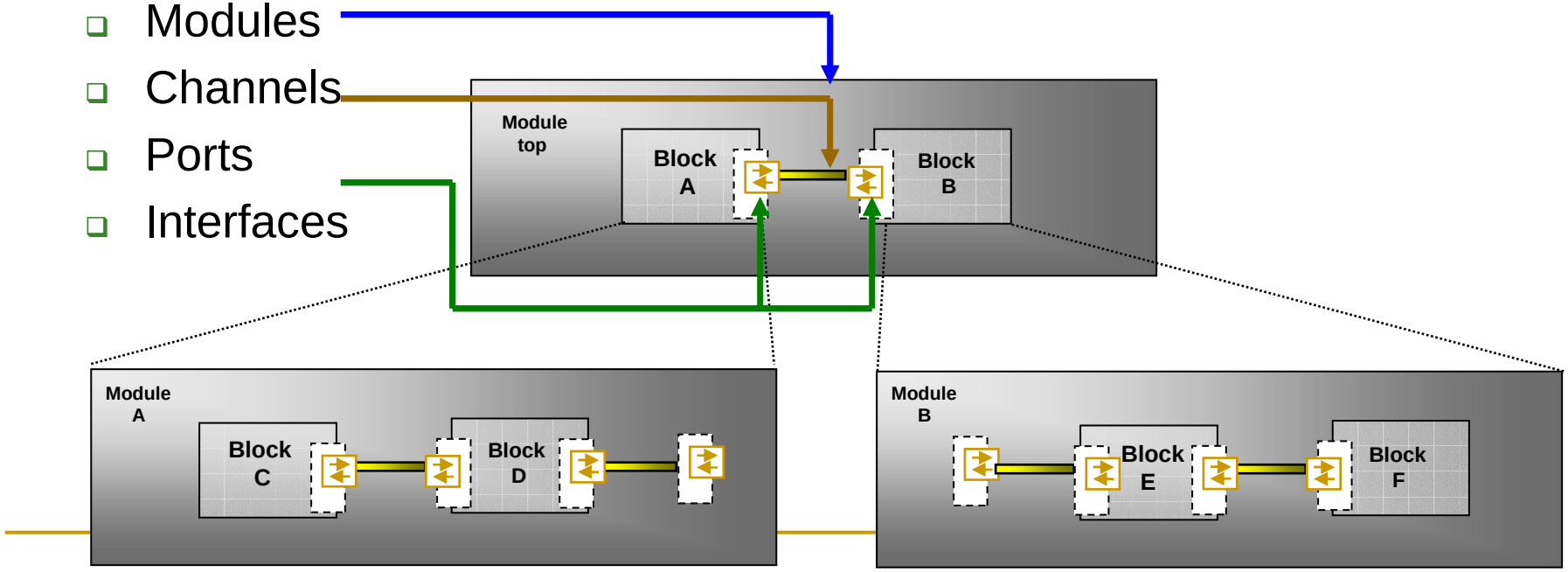
Florida Atlantic University

Presentation Outline

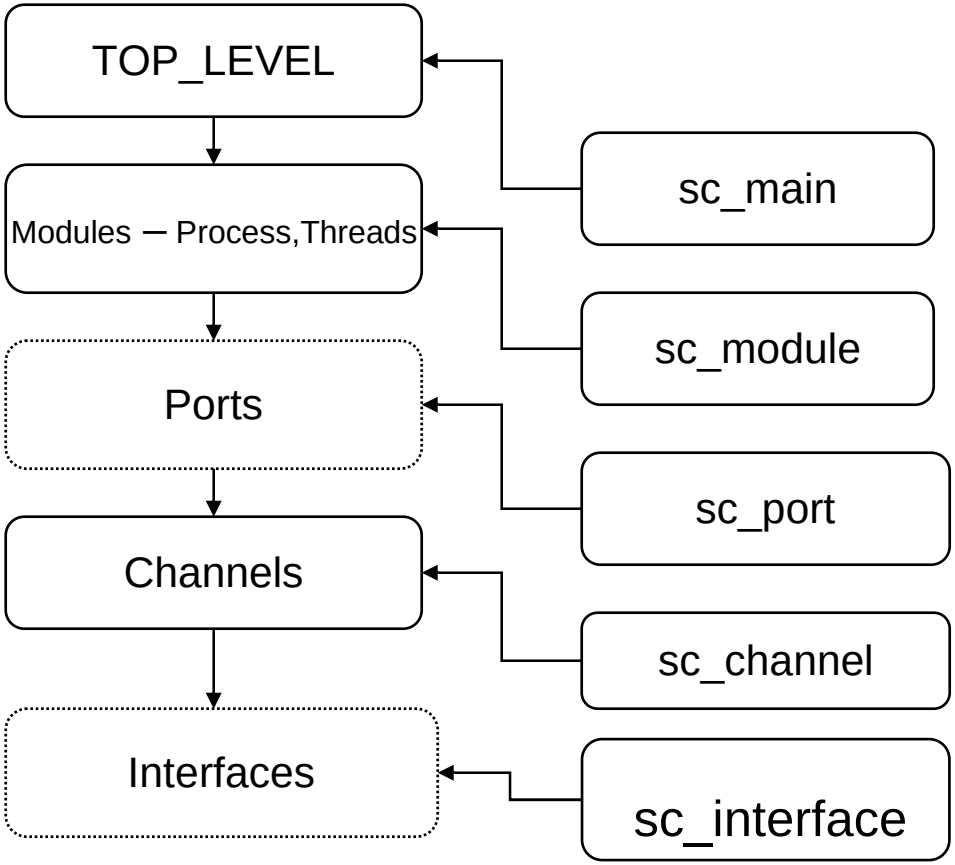
- SystemC Structural Hierarchy
- Graphical Notation
- Communication Modeling -Producer/Consumer Example
- Interfaces
- Channel
- Ports
- Modules

Structural Modelization

- Structure and Hierarchy are used to control complexity by breaking the system into smaller more manageable pieces
- Basic hierarchy in SystemC is supported through the use of:
  - Modules
  - Channels
  - Ports
  - Interfaces



SystemC Structural Hierarchy – A Simplified view



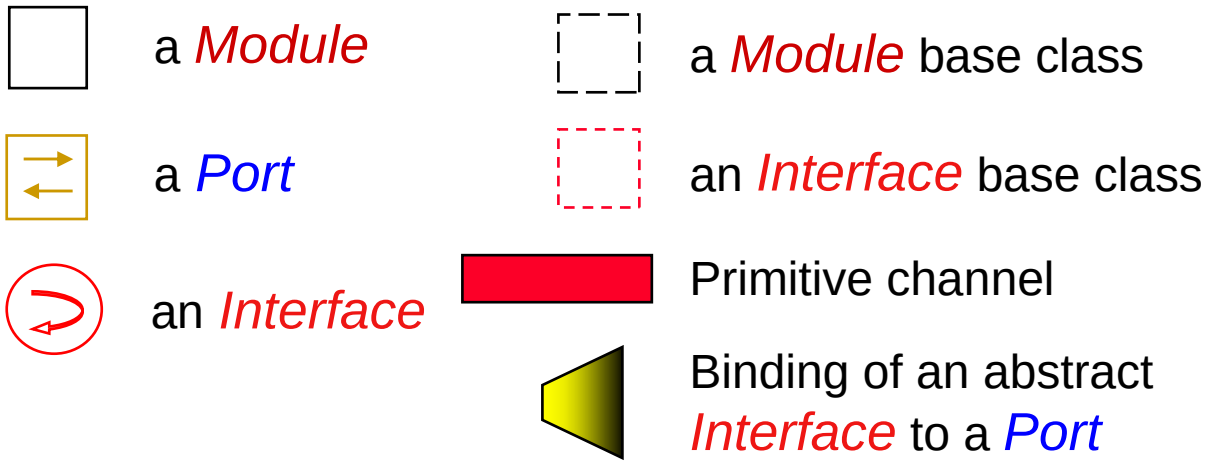
Module Anatomy

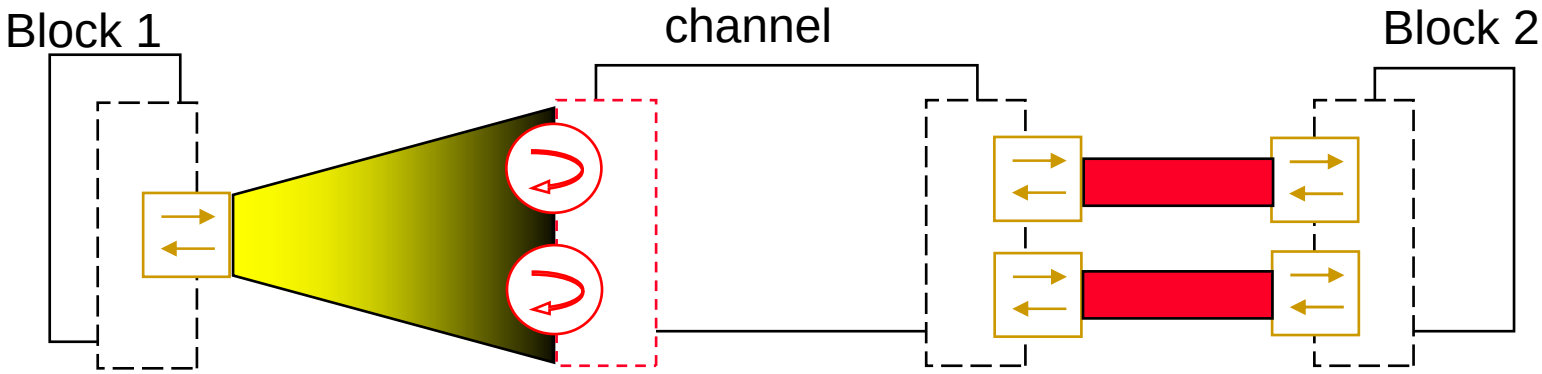
```
SC_MODULE(my_module) {

        /* Port Instances    */
        /* Channel Instances */
        /* Module Instances  */
        /* Processes Declaration and/or
           Definition        */
        SC_CTOR(my_module)
        {
                /* Module Netlist       */
                /* Process Registration */
        }
        /* All allowed C++ constructs */
};
```

CONSTRUCTOR

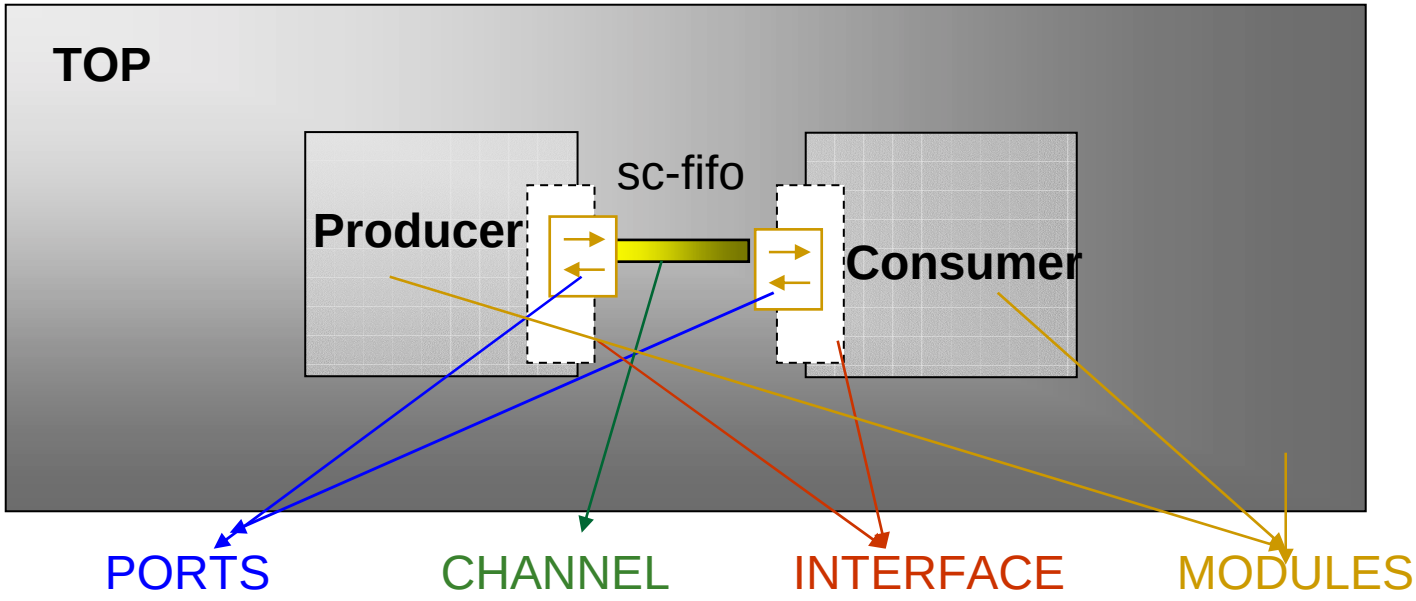
Graphical Notations





Communication Modeling – Producer/Consumer Example

- FIFO can store 10 characters
- Supports blocking read and write



Producer/Consumer - Example

- MODULES
  - TOP
  - Producer
  - Consumer

Producer/Consumer Example

- TOP Module

```
class TOP : public sc_module    {
  public:
    fifo *fifo_inst;
    producer *prod_inst;
    consumer *cons_inst;

    TOP(sc_module_name name) : sc_module(name)
    {
      fifo_inst = new fifo("Fifo1");     //FIFO Channel Instantiation

      prod_inst = new producer("Producer1");  //Producer Instantiation
      prod_inst->out(*fifo_inst);                        //Port Binding

      cons_inst = new consumer("Consumer1");   //Consumer Instantiation
      cons_inst->in(*fifo_inst);                          //Port Binding
    }                          };
```

```
int sc_main (int argc , char *argv[])
 {
   TOP top1("Top1");
   sc_start(-1);
   return 0;
}
```

Producer/Consumer Example

- Producer Module

```
class producer : public sc_module   {
   public:
     sc_port<write_if> out;   //Producer has a port with write interface
     SC_HAS_PROCESS(producer);
     producer(sc_module_name name) : sc_module(name)
     {
       SC_THREAD(main_action);
     }
     void main_action()
     {
       const char *str =
         "Hello How are you! see what SystemC can do for you today!\n";
       while (*str)
         out->write(*str++);
     }                          };
```

Constructor

Producer/Consumer Example

- Consumer Module

```
class consumer : public sc_module  {
   public:
     sc_port<read_if> in;   // Consumer has a port with read interface
     SC_HAS_PROCESS(consumer);
     consumer(sc_module_name name) : sc_module(name)
     {
       SC_THREAD(main_action);
     }
     void main_action( )     {
       char c;
             while (true)       {
         in->read(c);
         cout << c << flush;
         if (in->num_available() == 1)   cout << "<1>" << flush;
         if (in->num_available() == 9)   cout << "<9>" << flush;
                        } /*End of While loop */
 }/*End of main_action */       };
```

Constructor

Producer/Consumer Example

- Channel

```
class fifo : public sc_channel, public write_if, public read_if
{
  public:
    fifo(sc_module_name name) : sc_channel(name), num_elements(0), first(0)
{
}

    void write(char c) {
      if (num_elements == max)
        wait(read_event);

      data[(first + num_elements) % max] = c;
      ++ num_elements;
      write_event.notify();
    }
```

Definition/Implementation of WRITE interface

Producer/Consumer Example

- Channel

```
void read (char &c)   {
      if (num_elements == 0)
        wait(write_event);
      c = data[first];
      -- num_elements;
      first = (first + 1) % max;
      read_event.notify();
    }

    void reset() { num_elements = first = 0; }

    int num_available() { return num_elements;}
```

Definition/Implementation of READ interface

```
private:
      enum e { max = 10 };
      char data[max];
      int num_elements, first;
      sc_event write_event,
                  read_event;
};
```

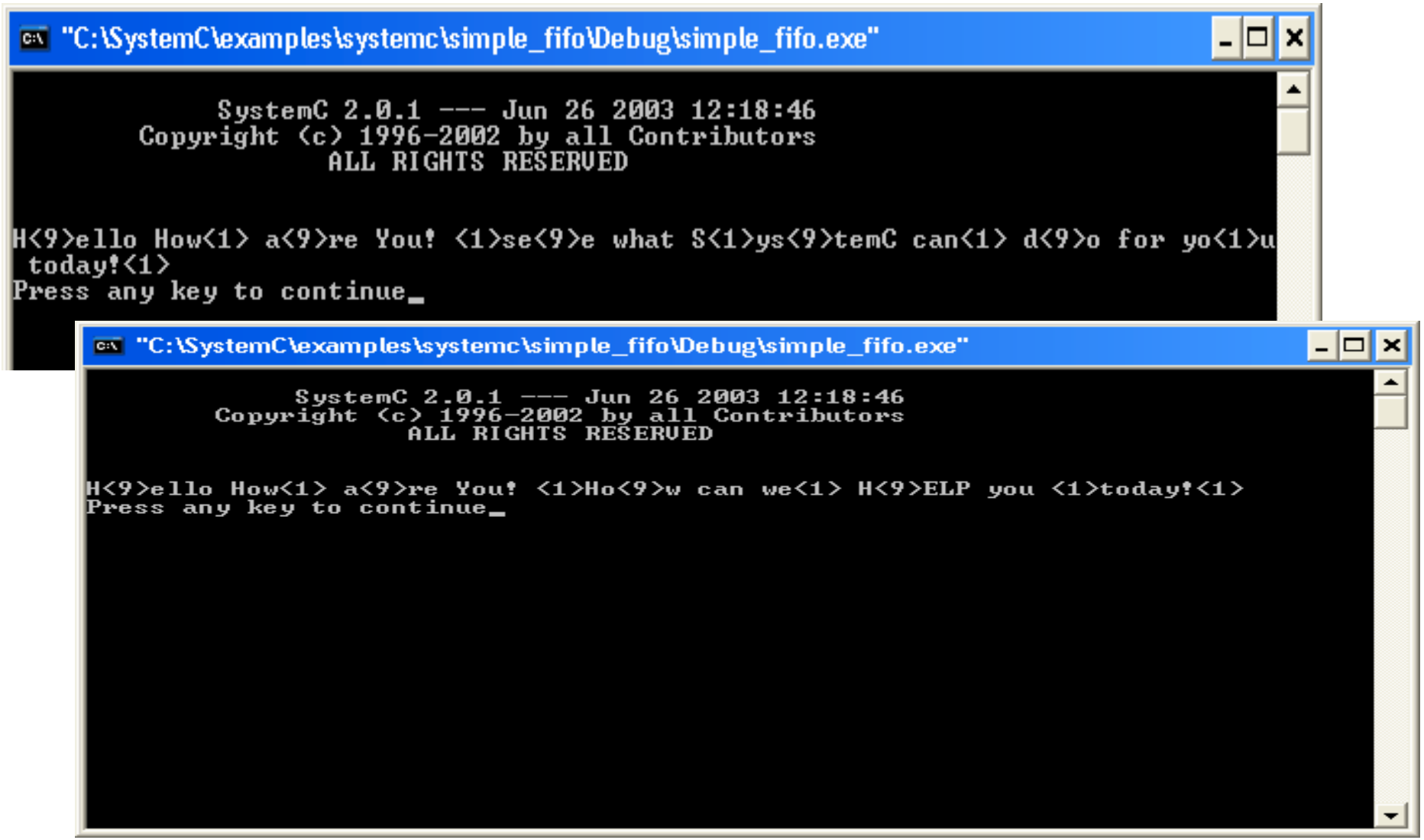
Producer Consumer Example

- Interface

```
class write_if : virtual public sc_interface
{
   public:
     virtual void write(char) = 0;
     virtual void reset() = 0;
};

class read_if : virtual public sc_interface
{
   public:
     virtual void read(char &) = 0;
     virtual int num_available() = 0;
};
```

Output Snapshot

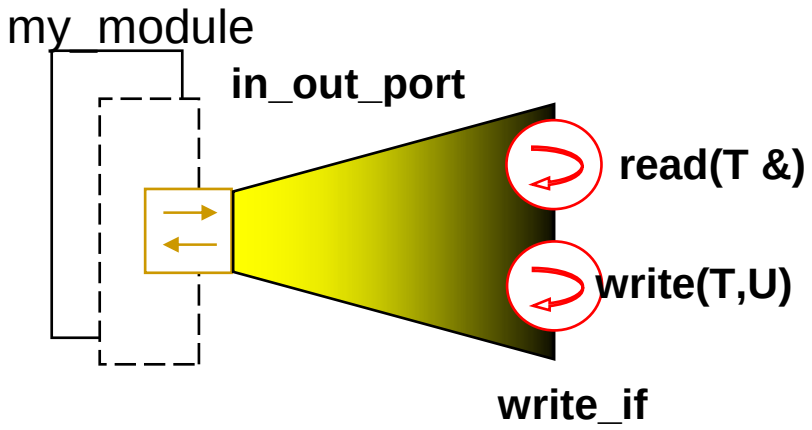


Interfaces

- Interfaces are means of communication between ports and channels
- "*A port is bound to the channel through an interface […]*"
  - Interfaces **declare** sets of method function declarations that channels must **implement**
    - Interfaces don't implement the functions – the *methods are pure virtual*
    - From a C++ viewpoint

      An Interface is an abstract class with only pure virtual methods as members
- SystemC 2.0 allows users to define their own interfaces

Interfaces

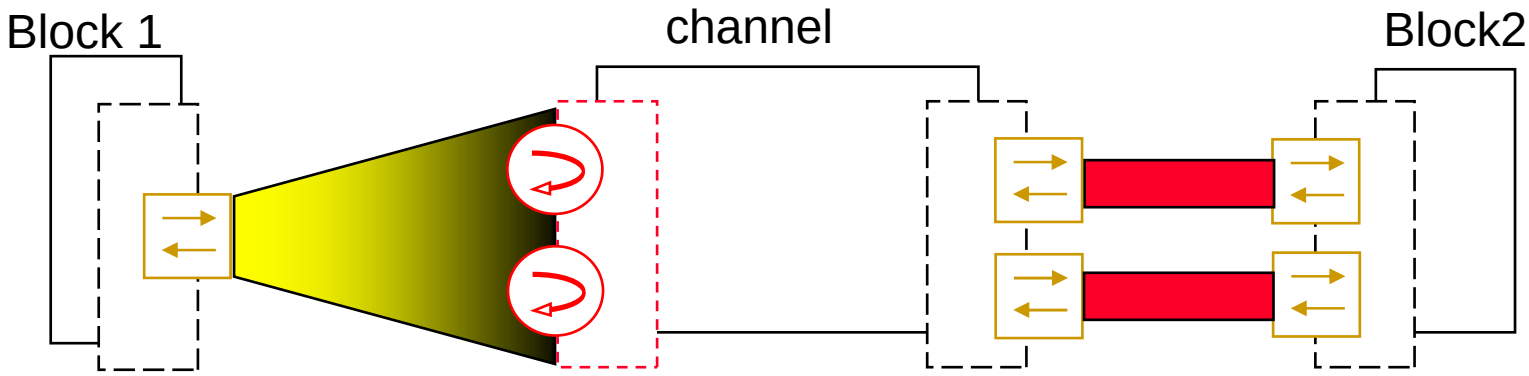


my_interface.h

```
Class write_if : virtual public sc_interface
{
public:
    virtual void write(char) = 0;
    virtual void reset() = 0;
};
```

Channels

- SystemC *Channels* separate communication from functionality
  - *Channels* are containers for communication protocols and synchronization events
  - *Channels* implement one/more *Interface(s)* or *Ports*



  - An *Interface* defines a set of pure virtual methods
  - *Modules* access Channels' *Interface(s)* via *Ports*

PORTS

- Ports allow a Modules to connect to Channels through an *Interface*
  - Pass data to/from the module
  - Trigger actions within the Module
  - Ports are declared with the SystemC keyword sc_port<T>

sc_port < *interface_name* , N>

p*ort_instance_name* ;

PORTS

- Ports must specify the type of interface to which they correspond
  - During elaboration, ports are *bound* to the interfaces of channels
- Note that many times the interfaces are templates with respect to the data type. For example, to declare a port named “my_port” that can access the interface sc_signal_in_if<int>

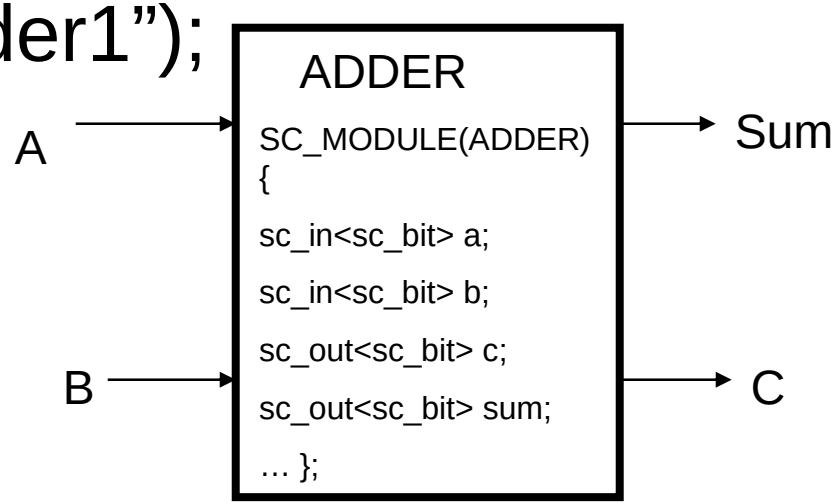
  sc_port<sc_signal_in_if <int> > my_port ;

PORTS

- Port Binding:

  Named Form: Order doesn't matter

```
ADDER adder1("adder1");
adder1.a(in1);
adder1.b(in2);
adder1.sum(sum);
adder1.c(carry_out);
```

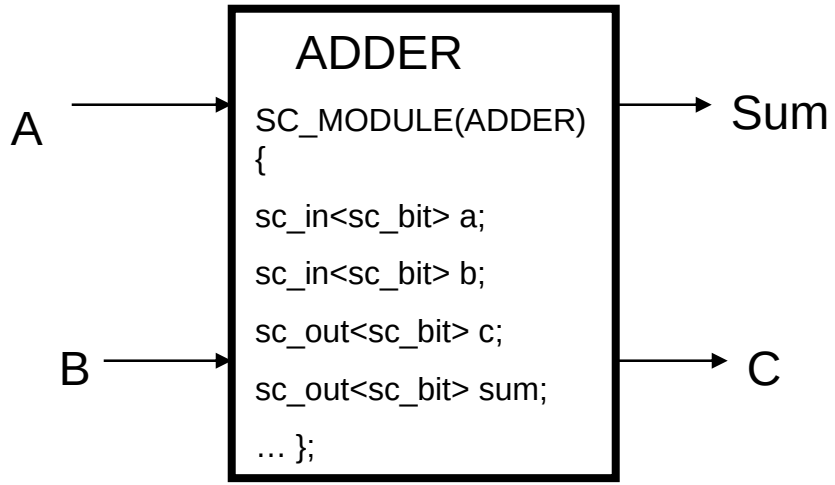


PORTS

- Positional Form : Ordering is important

  Adder2 = new ADDER(“Adder2);

  (*Adder2) (a, b, carry_out, sum)



Modules

- Modules are the basic building blocks for partitioning a design
- Modules helps in
  - Breaking complex systems into smaller manageable pieces
  - Hiding internal data representation
  - Hiding algorithms from other modules
- Module can be described with the SC_MODULE macro or derived explicitly from sc_module, as well as other classes
  - SC_MODULE(Producer) { … };
  - Class Producer: public sc_module { … };

Modules

- A Module must contain a C++ constructor.
  - Instantiate the elements of the module: channels, other modules, ports
  - Declare event sensitivities
  - Register processes (SC_METHOD & SC_THREAD) with the SystemC kernel.
- Constructors should have the explicit SystemC macro SC_CTOR or SC_HAS_PROCESS(module name);

Module Instantiation

- SC_MODULEs can be instantiated to create the hierarchy
- C++ allows two ways to instantiate a module:
  - Using pointer (module *t = new <module name>)
  - Using Member instances (module t)

Module Instantiation (cont…)

top.h

```
SC_MODULE(top) {
  smodule1 *s1 ;
  SC_CTOR(top) {
    s1 = new smodule1("s1") ;
    s1->sport1(sig1) ;
   }
}
```

Create a pointer « s1 » on « *smodule1* »

Allocate memory for the new instance of « *s1* » module

Connect « *sport1* » to « *sig1* » channel

top.h

```
SC_MODULE(top) {
  smodule1 s1 ;
  SC_CTOR(top) :
    s1("s1"){
     s1.sport1(sig1) ;
   }
}
```

Create « *s1* » as member of « top » module

Initialize « *s1* » member

Connect « *sport1* » to « *sig1* » channel

Processes

- SystemC provides 3 forms of processes to fill different needs in expressiveness and simulation performance
  - SC_METHOD
  - SC_THREAD
  - SC_CTHREAD
    - Derived from SC_THREAD to optimize clock-edge sensitivity

NOTE: They all are infinite loops (either implicit in SC_METHOD or user- implemented in SC_THREAD/SC_CTHREAD) synchronized by wait on events and events notifications

SC_METHOD

- A module method with a sensitivity list that does execute and returns control back to the simulation kernel
  - No wait() statement is allowed (Execution from begin to end)
  - No infinite loops allowed
  - Does not keep an *implicit execution state*
    - May be faster than Thread processes

```
void my_method1() ;


SC_CTOR(my_module)
{
  SC_METHOD(my_method1);
  sensitive << my_port1;
}
```

```
void my_module::my_method1() {
…
        // Code of my_method1
} ;
```

SC_THREAD

- A module method which has its own thread of execution, and which can call code that calls wait() statement
  - Normally have infinite loops that continuously execute

```
void my_thread1() ;
void my_thread2() ;

SC_CTOR(my_module)
{
  SC_THREAD(my_thread1);
  SC_THREAD(my_thread2);
  sensitive << my_port1;
}
```

```
void my_module::my_thread2() {
    // Initialization of my_thread2
    …
    while (true) {
        // Code of my_thread2
        wait();
    }
  } ;
```

  - In SystemC2.0 threads are static, Dynamic threads are being considered for future SystemC versions

SC_CTHREAD

- Variant of SC_THREAD process – provided for SystemC1.0 compatibility only
  - Only triggered on one edge of one clock
  - No other sensitivity than the specified clock

```
void my_cthread1() ;


SC_CTOR(my_module)
{
  SC_CTHREAD(my_cthread1,clk.pos())
}
```

```
void my_module::my_cthread1() {
    // Initialization of my_cthread1
    …
    while (true) {
        // Code of my_cthread1
        wait();
    }
  } ;
```

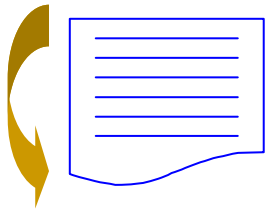
- Does keep the context of the suspension point (locals vars…)

SystemC Processes (Summary)

- SystemC provides 3 forms of process
  - SC_METHOD
  - SC_THREAD
  - SC_CTHREAD

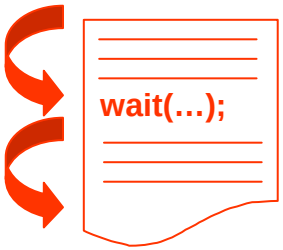
Has a sensitivity list

Execute in 0 time



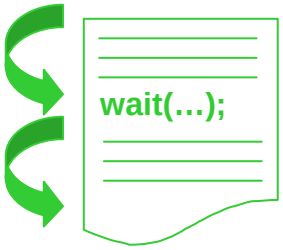
May have a sensitivity list

Can be suspended And reactivated



Sensitive only to a clock

Can be suspended And reactivated




```
…
SC_CTOR(<module_name>)
{
  SC_METHOD(<sc_method_name>)
  sensitive << <port1> << <port2> ;
  SC_THREAD(<sc_thread_name>);
  SC_THREAD(<sc_thread_name>);
  sensitive << <port1> << <port2> ;
  SC_CTHREAD(<sc_cthread_name>,<clk_name>.pos());
  …
 }
```

THANK YOU