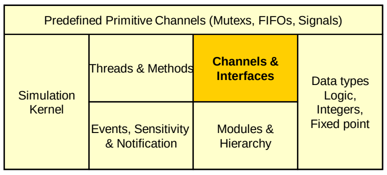
Plan

- Ch1 – Overview of System Design Using SystemC
- Ch2 – Overview of SystemC
- Ch3 – Data Types
- Ch4 – Modules
- Ch5 – Notion of Time
- Ch6 – Concurrency
- Ch7 – Predefined Channels
- Ch8 – Structure
- Ch9 – Communication
- Ch10 – Custom Channels and Data
- Ch11 – Transaction Level Modeling

Ecole polytechnique de l'université de Nice-Sophia Antipolis
Département Electronique

Communication


Predefined Primitive Channels (Mutexs, FIFOs, Signals)
Simulation Kernel
Threads & Methods
Channels & Interfaces
Events, Sensitivity & Notification
Modules & Hierarchy
Data types Logic, Integers, Fixed point

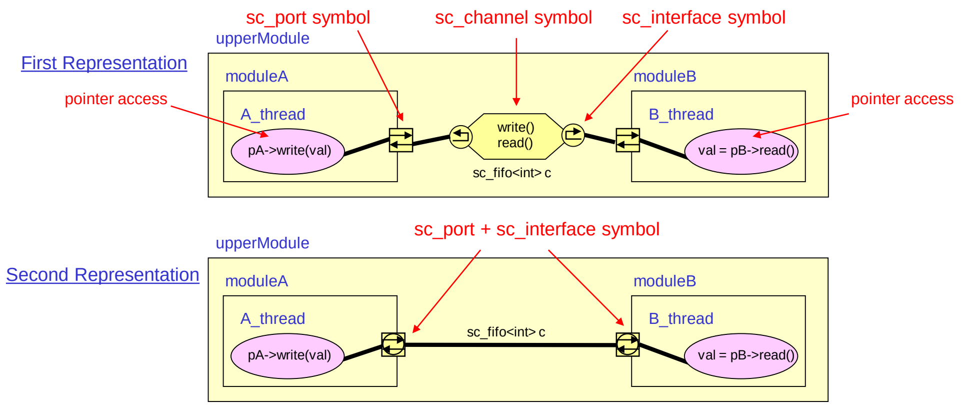
- Port & Interface
- Standard Interfaces
- Static Sensitivity
- Port Array
- SystemC Exports

The Port

- Communication between modules
- Two concerns
  - safety
    - To avoid race condition (anomalous behavior due to unexpected critical dependence on the relative timing of events)
    - Events and channels
  - ease of use
    - Involving global variables (well known as poor methodology)
    - Having a process in an upper-level module. This process would monitor and manage events defined in instantiated modules (awkward !)

The Port

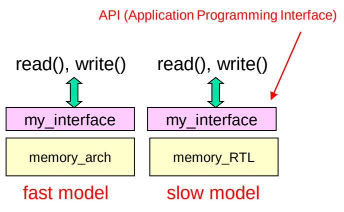
- SystemC approach
  - lets modules use channels inserted between the communicating modules
  - this is a concept called a port
  - a port is a <u>pointer</u> to a channel outside the module



Interfaces
C++

- C++ defines a concept known as an abstract class
  - Pure virtual functions
  - No implementation of the functions

```
struct my_interface
{
    virtual void write(unsigned addr, int data) = 0;
    virtual int read(unsigned addr) = 0;
};
```



2 implementations

```
struct memory_arch: public my_interface
{
    virtual void write(unsigned addr, int data)
    {
        mem[addr] = data;
    }
    virtual int read(unsigned addr)
    {
        return mem[addr];
    }
    private:
    int mem[1024];
};
```

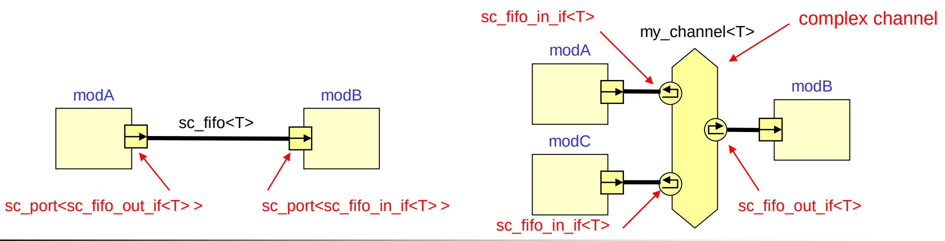
```
struct memory_RTL: public my_interface
{
    virtual void write(unsigned addr, int data)
    {
        // complex details of RTL memory write
    }
    virtual int read(unsigned addr)
    {
        // complex details of RTL memory read
    }
    private:
     // complex details of RTL memory storage
};
```

example

```
void memtest(my_interface *mem)
{
    // complex memory test
    mem->write(0xFF000, 12);
}

memory_arch  fast_model;
memory_RTL  slow_model;

memtest(&fast);
memtest(&slow);
…
```

Interfaces
SystemC

- A SystemC interface
  - is an abstract class that inherits from sc_interface
    - provides only pure virtual declarations of methods
    - no implementation
    - no data
- A SystemC Channel
  - is a class that implements one or more SystemC interface classes
  - inherits from either sc_channel (equal to sc_module) or sc_prim_channel (chapter 7)
    - A channel implements all the methods of the inherited interface classes



Port Declaration

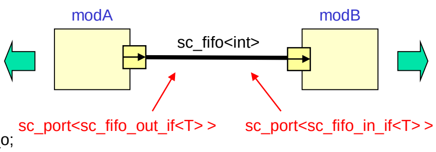
- A SystemC Port is a template class
- Port inherits from SystemC interface
- Syntax

sc_port<*interface*> portname;

sc_port class — sc_interface class — name of the port

first solution
```
SC_MODULE(modA)
{
    sc_fifo_out<int> fifo_o;
    …
```

second solution
```
SC_MODULE(modA)
{
    sc_port<sc_fifo_out_if<int> > fifo_o;
    …
```
blank space !!



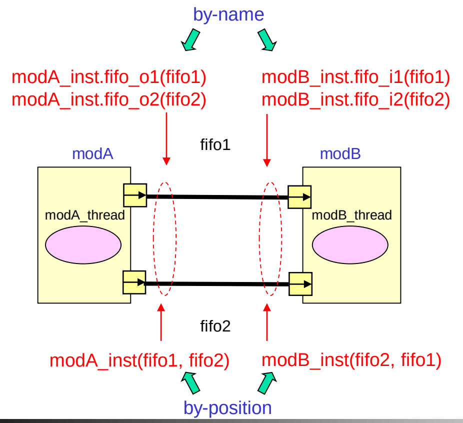
first solution
```
SC_MODULE(modB)
{
    sc_fifo_in<int> fifo_i;
    …
```

second solution
```
SC_MODULE(modB)
{
    sc_port<sc_fifo_in_if<int> > fifo_i;
    …
```
blank space !!

Port Connection

- Two syntaxes
  - by-name (strongly recommended !)
  - by-position

```
SC_MODULE(modA)
{
    sc_port<sc_fifo_out_if<int> > fifo_o1;
    sc_port<sc_fifo_out_if<int> > fifo_o2;
    …

SC_MODULE(modB)
{
    sc_port<sc_fifo_in_if<int> > fifo_i2;
    sc_port<sc_fifo_in_if<int> > fifo_i1;
    …
```



port_ex

Code Example

```
int sc_main(int argc, char* argv[])
{
    sc_fifo<int> fifo1("positive"), fifo2("negative");
    modA modA_inst("modA");
        modA_inst.fifo_o1(fifo1); // by-name
        modA_inst.fifo_o2(fifo2);
        // modA_inst(fifo1, fifo2);
    modB modB_inst("modB");
        // modB_inst.fifo_i1(fifo1);
        // modB_inst.fifo_i2(fifo2);
        modB_inst(fifo2, fifo1);  // by-position

    sc_start(10, SC_MS);

    return 0;
}
```

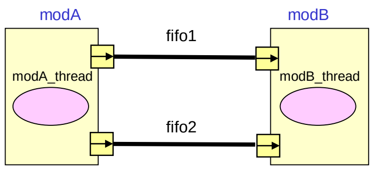
Accessing Ports from within a process

- The sc_port overloads the C++ operator ->()

sc_port<*interface*> portname;

portname->*method*(…);

name of the port   method of the interface class



```
void modA::modA_thread(void)
{
    int val = 0;
    while (true)
    {
        fifo_o1->write(val);
        fifo_o2->write(-val);
        cout << sc_time_stamp() << " ModA : writting value = “;
        cout << val << endl;
        val = val + 5;
        wait(1, SC_MS);
    }
}
```

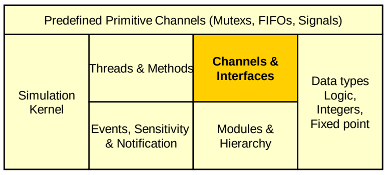
modA_thread

port_ex

```
void modB::modB_thread(void)
{
    int val;
    while (true)
    {
        fifo_i1->read(val);
        cout << sc_time_stamp() << " ModB : reading value 1 = “;
        cout << val << endl;
        fifo_i2->read(val);
        cout << sc_time_stamp() << " ModB : reading value 2 = “;
        cout << val << endl;
    }
}
```

modB_thread

Communication


Predefined Primitive Channels (Mutexs, FIFOs, Signals)
Simulation Kernel
Threads & Methods
Channels & Interfaces
Events, Sensitivity & Notification
Modules & Hierarchy
Data types Logic, Integers, Fixed point

- Port & Interface
- Standard Interfaces
- Static Sensitivity
- Port Array
- SystemC Exports

sc_fifo Interfaces

- Two interfaces
  - sc_fifo_in_if< >
  - sc_fifo_out_if< >

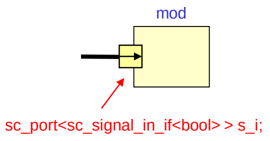
```
template <class T>
class sc_fifo_nonblocking_in_if : virtual public sc_interface
{
public:
    virtual bool nb_read( T& ) = 0;
    virtual const sc_event& data_written_event() const = 0;
};
template <class T>
class sc_fifo_blocking_in_if : virtual public sc_interface
{
public:
    virtual void read( T& ) = 0;
    virtual T read() = 0;
};
template <class T>
class sc_fifo_in_if : public sc_fifo_nonblocking_in_if<T>,
                      public sc_fifo_blocking_in_if<T>
{
public:
    virtual int num_available() const = 0;
protected:
    sc_fifo_in_if();
};
```

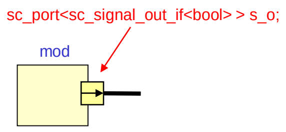
```
template <class T>
class sc_fifo_nonblocking_out_if : virtual public sc_interface
{
public:
    virtual bool nb_write( const T& ) = 0;
    virtual const sc_event& data_read_event() const = 0;
};
template <class T>
class sc_fifo_blocking_out_if : virtual public sc_interface
{
public:
    virtual void write( const T& ) = 0;
};
template <class T>
class sc_fifo_out_if : public sc_fifo_nonblocking_out_if<T>,
                       public sc_fifo_blocking_out_if<T>
{
public:
    virtual int num_free() const = 0;
protected:
    sc_fifo_out_if();
};
```

sc_signal Interfaces

- Interfaces
  - sc_signal_in_if< > equivalent to sc_in
  - sc_signal_inout_if< > equivalent to sc_out
  - sc_signal_out_if< > (deprecated, don't use it ! but …)

```
template <class T>
class sc_signal_in_if : virtual public sc_interface
{
public:
    virtual const T& read() const = 0;
    virtual const sc_event& value_changed_event() const = 0;
    virtual bool event() const = 0;
protected:
    sc_signal_in_if();
};
```





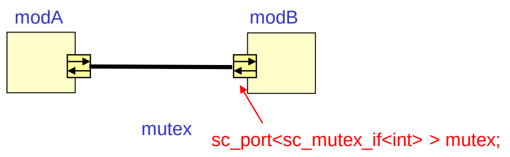
```
template <class T>
class sc_signal_inout_if : public sc_signal_in_if<T>
{
public:
    virtual void write( const T& ) = 0;
protected:
    sc_signal_inout_if();
};
```

sc_mutex & sc_semaphore Interfaces

- sc_mutex and sc_semaphore channels provide interface
  - sc_mutex_if
  - sc_semaphore_if

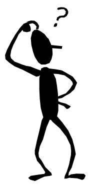
```
class sc_mutex_if : virtual public sc_interface
{
public:
   virtual int lock() = 0;
    // returns -1 if mutex could not be locked
   virtual int trylock() = 0;
   // returns -1 if mutex was not locked by caller
   virtual int unlock() = 0;
protected:
   sc_mutex_if();
};
```

```
class sc_semaphore_if : virtual public sc_interface
{
public:
   virtual int wait() = 0;
   virtual int trywait() = 0;
   virtual int post() = 0;
   virtual int get_value() const = 0;
protected:
   sc_semaphore_if();
};
```



Exercice

- Write the implementation of the sc_mutex class

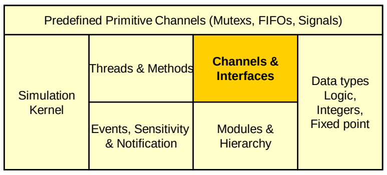


```
class sc_mutex : public sc_mutex_if,  public sc_prim_channel
{
public:
   // constructors
   sc_mutex() : m_owner( 0 ),
         sc_prim_channel( sc_gen_unique_name( "mutex" ) )
   { }
   explicit sc_mutex( const char* name_ ) : m_owner( 0 ),
         sc_prim_channel( name_ )
   { }
   // blocks until mutex could be locked
   virtual int lock();

   // returns -1 if mutex could not be locked
   virtual int trylock();

   // returns -1 if mutex was not locked by caller
   virtual int unlock();

protected:
   bool in_use() const
   { return ( m_owner != 0 ); }

   sc_process_b* m_owner;
   sc_event      m_free;
};
```

```
int sc_mutex::lock()
{
   while( in_use() )
      wait( m_free );
   m_owner = sc_get_curr_process_handle();
   return 0;
}

int sc_mutex::trylock()
{
   if( in_use() )
      return -1;
   m_owner = sc_get_curr_process_handle();
   return 0;
}

int sc_mutex::unlock()
{
   if( m_owner != sc_get_curr_process_handle() )
      return -1;
   m_owner = 0;
   m_free.notify();
   return 0;
}
```

Ecole polytechnique de l'université de Nice-Sophia Antipolis

Département Electronique


Predefined Primitive Channels (Mutexs, FIFOs, Signals)
Simulation Kernel
Threads & Methods
Channels & Interfaces
Events, Sensitivity & Notification
Modules & Hierarchy
Data types Logic, Integers, Fixed point

Communication

- Port & Interface
- Standard Interfaces
- Static Sensitivity
- Port Array
- SystemC Exports

Static Sensitivity with Port

- Chapter “Concurrency”
  - sc_fifo::data_written_event()
  - sc_signal<bool>::posedge_event()
  - use sensitive() method at elaboration time
- Ports are a pointers !
  - undefined at the point in time when sensitive() method needs about them
  - solution : sc_event_finder class

static_port_sensitivity

```
struct my_port_sc_signal_in_if_bool : public sc_port<sc_signal_in_if<bool> >
{
   typedef sc_signal_in_if<bool> if_type;   // typing aid

   sc_event_finder& ef_posedge_event() const
   {
      return *new sc_event_finder_t<if_type> (*this, &if_type::posedge_event);
   }
};
```

trigger on pos-edge event

Static Sensitivity Example

using of my port

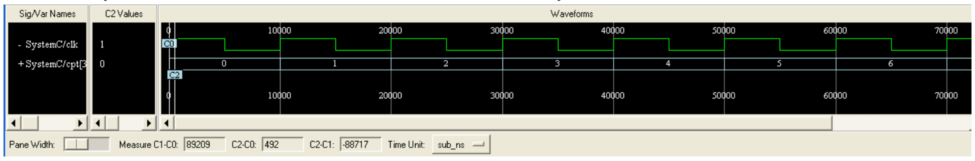
using of my method

```
SC_MODULE(modA)
{
    my_port_sc_signal_in_if_bool my_port;
    sc_out<int> cpt_o;

    int cpt;

    SC_CTOR(modA) : cpt(0)
    {
        SC_METHOD(modA_method);
            dont_initialize();
            sensitive << my_port.ef_posedge_event();
    }
    void modA_method(void)
    {
        cout << sc_time_stamp() << " Event !" << cpt << endl;
        cpt_o->write(cpt);
        cpt++;
    }
};
```

static_port_sensitivity

connection of clk with my port

```
int sc_main(int argc, char* argv[])
{
    sc_set_time_resolution(100, SC_NS);

    sc_clock clk("clk", 1, SC_MS);
    sc_signal<int> cpt;

    modA modA_inst("modA");
        modA_inst.my_port(clk);
        modA_inst.cpt_o(cpt);

    sc_trace_file *tf = sc_create_vcd_trace_file("wave");
    sc_write_comment(tf, "Simulation of Static Sensitivity");
    sc_trace(tf,clk.signal(), "clk");
    sc_trace(tf,cpt,"cpt");

    sc_start(20, SC_MS);
    sc_close_vcd_trace_file(tf);
    return 0;
}
```



Guideline : Dot or Arrow ? Basic Channels – FIFO Example

- Use dot (.)
  - in the elaboration section of the code
  - local channel
- Use arrow (->)
  - process with port channel

sc_port<sc_fifo_in_if<T> >

sc_port<sc_fifo_out_if<T> >

Wrong !

```
SC_MODULE(mod)
{
    sc_fifo_in<double> fifo_in;
    sc_fifo_out<double> fifo_out;

    SC_CTOR(mod)
    {
        SC_THREAD(p_thread);
            sensitive << fifo_in.data_written();
    }

    void p_thread()
    {
        double val;
        while (true)
        {
            wait();
            fifo_in.nb_read(val);
            fifo_out.write(val*val);
        }
    }
};
```

dot

Right !

```
SC_MODULE(mod)
{
    sc_fifo_in<double> fifo_in;
    sc_fifo_out<double> fifo_out;

    SC_CTOR(mod)
    {
        SC_THREAD(p_thread);
            sensitive << fifo_in.data_written();
    }

    void p_thread()
    {
        double val;
        while (true)
        {
            wait();
            fifo_in->nb_read(val);
            fifo_out->write(val*val);
        }
    }
};
```

dot

arrow

Guideline : Dot or Arrow ? Evaluate-Update Channels – Signal Example

- same guideline as basic channels

sc_in<T> ⟺ sc_port<sc_signal_in_if<T> >

```
SC_MODULE(shiftleft)
{
   sc_in<bool> serial_in;
   sc_out<sc_int<32> > q;
   sc_in<bool> clk;
   sc_in<bool> rst;

   sc_signal<sc_int<32> > reg;  // Channel

   SC_CTOR(shiftleft)
   {
      reg.write(0);
      SC_METHOD(p_method);
         sensitive << clk.pos() << rst;
      q.initialize(0);
   }

   void p_method()
};
```

Dot (Channel)

Dot (Port)

```
void shiftleft::p_method()
{
   if (rst->read() == true)
   {
      reg = 0;
      q->write(reg);
   }
   else
   {
      reg.write(reg.read() + 1);
      reg[0] = serial_in->read();
      q->write(reg);
   }
}
```

dot (Channel)

Arrow (Port)

École polytechnique de l'université de Nice-Sophia Antipolis
Département Electronique

Communication

Predefined Primitive Channels (Mutexs, FIFOs, Signals)			
Simulation Kernel	Threads & Methods	**Channels & Interfaces**	Data types Logic, Integers, Fixed point
	Events, Sensitivity & Notification	Modules & Hierarchy	

- Port & Interface
- Standard Interfaces
- Static Sensitivity
- Port Array
- SystemC Exports

Configuration of Multi-port

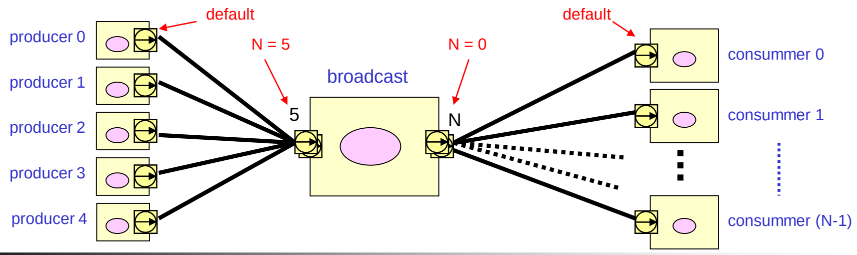
- The sc_port<> class provide a second parameter
  - The array size of the port
  - Optional parameter

sc_port<*interface*, N> portname;

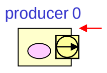
N = 0 ⟹ unlimited number of ports

N = 1 ⟹ default value (sc_port<*interface*>)

N > 1 ⟹ N channels must be connected to the port

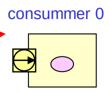


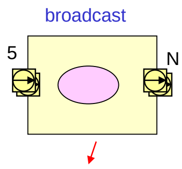
Example: Declarations & Process



```
SC_MODULE(producer)
{
    sc_port<sc_fifo_out_if<int> > mess_out;
    …
```

```
SC_MODULE(consummer)
{
    sc_port<sc_fifo_in_if<int> > mess_in;
    …
```





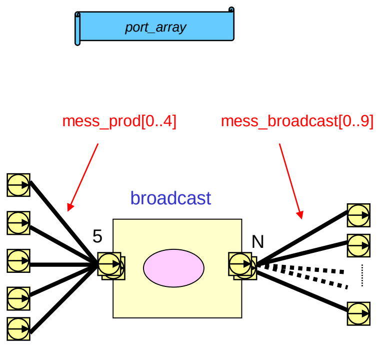
```
SC_MODULE(broadcast)
{
    sc_port<sc_fifo_in_if<int>, 5> mess_in;
    sc_port<sc_fifo_out_if<int>, 0> mess_out;
    …
```

port_array

```
void broadcast::broadcast_thread(void)
{
    int val = 0;
    while (true)
    {
        wait(mess_in[0]->data_written_event()
            | mess_in[1]->data_written_event()
            | mess_in[2]->data_written_event()
            | mess_in[3]->data_written_event()
            | mess_in[4]->data_written_event());      wait on one of mess_in
                                                      data written event

        for (int i=0; i<mess_in->size(); i++)         non blocking read
        {
            if (mess_in[i]->nb_read(val) == true)
            {
                cout << sc_time_stamp() << "  Send from Producer ";
                cout << i << " to all consummer" << endl;
                for (int j=0; j<mess_out->size(); j++)  write on all channels
                    mess_out[j]->write(val);
            }
} } }
```

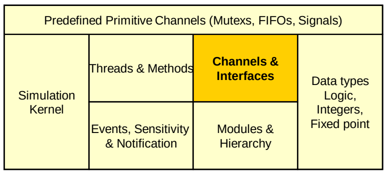
Example: Connections

```
int sc_main(int argc, char* argv[])                MAX = 10
{
   sc_fifo<int> mess_prod[5], mess_broadcast[MAX];

   // Producers Connections
   producer *producer_inst[5];
   for (int i=0; i<5; i++)                          Unique name
   {
      producer_inst[i] = new producer("producer_" + i, i);
         producer_inst[i]->mess_out(mess_prod[i]);
   }

   // Broadcast Connections
   broadcast broadcast_inst("broadcast");
      for (int i=0; i<5; i++)
         broadcast_inst.mess_in(mess_prod[i]);
      for (int i=0; i<MAX; i++)
         broadcast_inst.mess_out(mess_broadcast[i]);

   // Consummer Connections
   consummer *consummer_inst[MAX];
   for (int i=0; i<MAX; i++)
   {
      consummer_inst[i] = new consummer("consummer_" + i, i);
         consummer_inst[i]->mess_in(mess_broadcast[i]);  ← connection
   }
   sc_start(1, SC_US);
   return 0;
}
```



Ecole polytechnique de l'université de Nice-Sophia Antipolis
Département Electronique

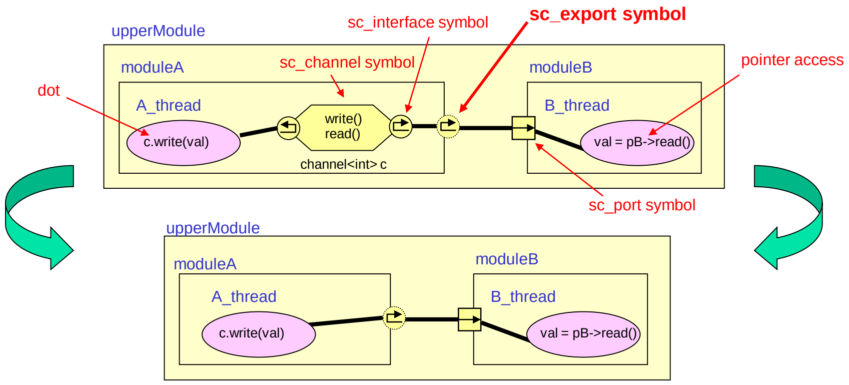
Communication


Predefined Primitive Channels (Mutexs, FIFOs, Signals)
Simulation Kernel
Threads & Methods
Channels & Interfaces
Data types Logic, Integers, Fixed point
Events, Sensitivity & Notification
Modules & Hierarchy

- Port & Interface
- Standard Interfaces
- Static Sensitivity
- Port Array
- SystemC Exports

Introduction

- New type of port called the sc_export class
- Similar to standard ports but differs in connectivity
- Principle
  - move the channel inside the defining module
  - use the port externally as though it were a channel



Why use sc_export ?

- A module can access the internal channel directly
  - Works only if the interior channel is publicly accessible
- For IP Provider
  - export only specific channels
  - keep everything else private
  - allows control over the interface
- sc_export provides multiple interfaces at the top level
  - contains specific interface
  - connection is NOT required
  - allows creation of “hidden” interface
    - a debug or test interface might be used internally by an IP Provider
    - not documented for the end user !
- Limitations
  - not possible to use in a static sensitivity list
    - use wait(xportname->event()) on Threads
  - not possible to have an array of sc_export

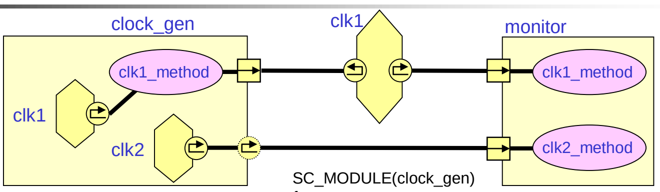
Declaration

```
SC_MODULE(name)                          Declaration of sc_export
{
   sc_export<interface> xportname;

   channel inst;                     Declaration of internal channel (sc_channel)

   SC_CTOR(name)                         Connection of internal channel to sc_export
   {
      xportname.bind(inst);
OR    xportname(inst);
    }
};
```

Methods

```
virtual sc_interface* get_interface();
virtual const sc_interface* get_interface() const;
```

Example

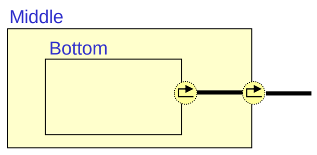


export_ex

```
SC_MODULE(monitor)
{
 sc_in<bool> clk1_p;
 sc_in<bool> clk2_p;

   SC_CTOR(monitor)
   {
      SC_METHOD(clk1_method);
         sensitive << clk1_p;
      SC_METHOD(clk2_method);
         sensitive << clk2_p;
   }

   void clk1_method()
   {
      cout << name() << " clk1=" << clk1_p->read()
         << " at " << sc_time_stamp() << endl;
   }
   void clk2_method()
   {
      cout << name() << " clk2=" << clk2_p->read()
         << " at " << sc_time_stamp() << endl;
   }
};
```

```
SC_MODULE(clock_gen)
{
   sc_port<sc_signal_out_if<bool> >  clk1_p;
   sc_export<sc_signal_out_if<bool> > clk2_p;

   sc_clock clk1;  }
   sc_clock clk2;  }  channels

   SC_CTOR(clock_gen)
     : clk1("clk1",4,SC_NS), clk2("clk2",6,SC_NS)
   {
      SC_METHOD(clk1_method);
         sensitive << clk1;
      clk2_p.bind(clk2);   <-------  export
   }

   void clk1_method()   ------->  clk1_method
   {
      clk1_p->write(clk1);
   }
};
```

Hierarchy



```
struct i_f: virtual sc_interface
{
   virtual void print() = 0;
};

struct Chan: sc_channel, i_f
{
   SC_CTOR(Chan)
   {}

   void print()
   {
      cout << "I'm Channel, name=“;
      cout << name() << endl;
   }
};
```

```
SC_MODULE(Bottom)
{
   sc_export<i_f> xp;
   Chan ch;

   SC_CTOR(Bottom) : ch("ch")
   {
      xp.bind(ch); // Bind export xp to channel ch
   }
};

SC_MODULE(Middle)
{
   sc_export<i_f> xp;
   Bottom* b;

   SC_CTOR(Middle)
   {
      b = new Bottom ("b");
      xp.bind(b->xp); // Bind export xp to export b->xp
   }
   …
   b->xp->print(); // Call method of export within child module
};
```