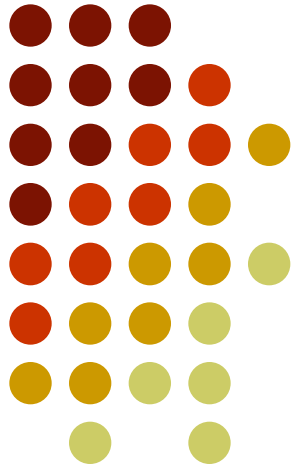
# SISTEME DE CALCUL DEDICATE

Curs 6



# Outline

- SystemC
  - Specialized ports, sc_export
- Bibliography

# Specialized ports, sc_export

- SystemC provides a variety of standard interfaces that go hand in hand with the built-in channels
  - basis for creating custom channels

# Specialized ports, sc_export

- SystemC FIFO Interfaces for the **sc_fifo**<*T*>channel
  - **sc_fifo_in_if**<*T*>
  - **sc_fifo_out_if**<*T*>
    - provide all of the methods implemented by **sc_fifo**<*T*>
- the interfaces were defined prior to the creation of the channel
- the channel simply becomes the place to implement the interfaces and holds the data implied by the functionality of a FIFO

# Specialized ports, sc_export

```
// Definition of sc_fifo<T> output interface
template <class T>
class sc_fifo_out_if: virtual public sc_interface {
public:
  virtual void write(const T& ) = 0;
  virtual bool nb_write(const T& ) = 0;
  virtual int num_free() const = 0;
  virtual const sc_event&
                  data_read_event() const = 0;
};
```

```
// Definition of sc_fifo<T> input interface
template<class T>
class sc_fifo_in_if: virtual public sc_interface{
public:
  virtual void read( T& ) = 0;
  virtual T read() = 0;
  virtual bool nb_read( T& ) = 0;
  virtual int num_available()const = 0;
  virtual const sc_event&
            data_written_event() const = 0;
};
```

# Specialized ports, sc_export

- SystemC Signal Interfaces for the sc_signal<T>channel
  - **Sc_signal_in_if**<*T*>
  - **Sc_signal_inout_if**<*T*>
    - Provide all of the methods provided by **sc_signal**<*T*>

# Specialized ports, sc_export

```
// Definition of sc_signal<T> input/output interface
template<class T>
class sc_signal_inout_if: public sc_signal_in_if<T>
{
public:
  virtual void write( const T& ) = 0;
};
```

```
// Definition of sc_signal<T> input interface
template<class T>
class sc_signal_in_if: virtual public sc_interface {
public:
  virtual const sc_event&
                    value_changed_event() const = 0;
  virtual const T& read() const = 0;
  virtual bool event() const = 0;
};
```

# Specialized ports, sc_export

- sc_mutex and sc_semaphore interfaces

```
// Definition of sc_mutex_if interface
class sc_mutex_if: virtual public sc_interface {
public:
  virtual int lock() = 0;
  virtual int trylock() = 0;
  virtual int unlock() = 0;
};
```

```
// Definition of sc_semaphore_if interface
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;
};
```

# Specialized ports, sc_export

- ports are defined on interfaces to channels
  - allow sensitivity to events defined on those channels
    - Example: process statically sensitive to the data_written_event()
    - Example: monitor an sc_signal<T> for any change in the data using the value_changed_event()
- ***Problem***: Ports are pointers that become initialized during elaboration, and they are undefined at the time when the **sensitive** method needs to know about them

# Specialized ports, sc_export

- solution: a special class **sc_event_finder**
  - defers the determination of the actual event until after elaboration
  - an **sc_event_finder** must be defined for each event defined by the interface

# Specialized ports, sc_export

```
class eslx_port
  :public sc_port<sc_signal_in_if<bool>,1>
{
public:
// Use a typedef to shorten syntax below
  typedef sc_signal_in_if<bool>if_type;
  sc_event_finder& ef_posedge_event() const {
    return *newsc_event_finder_t<if_type>(
                  *this,
                  &if_type::posedge_event
                );
  }//end ef_posedge_event
};
```

```
SC_MODULE(my_module) {
  eslx_port my_p;
  …
  SC_CTOR(…) {
    SC_METHOD(my_method);
    sensitive<< my_p.ef_posedge_event();
  }
  void my_method();
  …
};
```

# Specialized ports, sc_export

- SystemC provides a set of template specializations that provide port definitions on the standard interfaces and include the appropriate event finders

# Specialized ports, sc_export

```
// sc_port<sc_fifo_in_if<T>>
sc_fifo_in<T>name_fifo_ip;
sensitive<<name_fifo_ip.data_written();
value = name_fifo_ip.read();
name_fifo_ip.read(value);
if (name_fifo_ip.nb_read(value))...
if (name_fifo_ip.num_available())...
wait(name_fifo_ip.data_written_event());

// sc_port<sc_fifo_out_if<T>>
sc_fifo_out<T>name_fifo_op;
sensitive<<name_fifo_op.data_read();
name_fifo_op.write(value);
if (name_fifo_op.nb_write(value))...
if (name_fifo_op.num_free())...
wait(name_fifo_op.data_read_event());
```

Don't use dot (.) Use arrow (->) syntax.

**GUIDELINE: Use dot (.) in the elaboration section of the code, but use arrow (->) in processes.**

# Specialized ports, sc_export

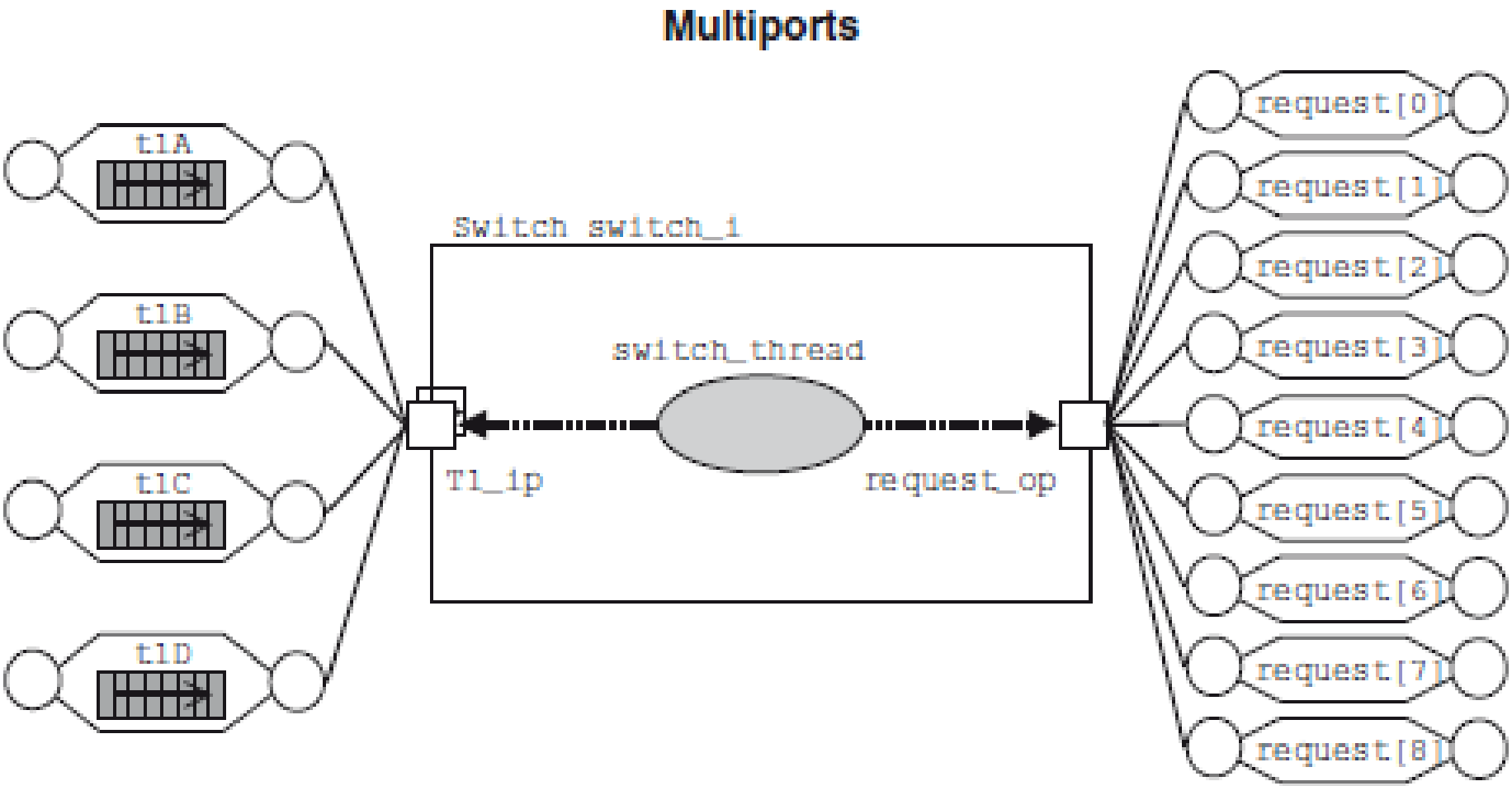
```
// sc_port<sc_signal_in_if<T>>
 sc_in<T> name_sig_ip;
 sensitive << name_sig_ip.value_changed();

 // Additional sc_in specializations...
 sc_in<bool> name_bool_sig_ip;
 sc_in<sc_logic > name_log_sig_ip;
 sensitive << name_sig_ip pos();
 sensitive << name_sig_ip neg();

 // sc_port<sc_signal_out_if<T>>
 sc_inout<T> name_sig_op;
 sensitive << name_sig_op.value_changed();
 sc_inout_resolved<N> name_rsig_op;
 sc_inout_rv<N> name_rsig_op;
 sc_inout<T> name_rsig_op;
 sc_inout_resolved<T> name_rsig_op;
 sc_inout_rv<T> name_rsig_op;
 // everything under sc_in<T> plus the following...
 name_sig_op.initialize(value);
 name_sig_op = value; // <-- DON'T USE!!!
```

# Specialized ports, sc_export

- the **sc_port**<*T*> provides additional template parameters:
  - the array size parameter
    - multi-port or port array
  - the port policy parameter

```
sc_port<interface[,N[,POL]]> portname;
// N=0..MAX Default N=1
// POL is of type sc_port_policy
// POL defaults to SC_ONE_OR_MORE_BOUND
```

# Specialized ports, sc_export



# Specialized ports, sc_export

```
//FILE: Switch.h
  SC_MODULE(Switch) {
    sc_port<sc_fifo_in_if<int>
           ,5
           ,SC_ONE_OR_MORE_BOUND
           > T1_ip;
    sc_port<sc_signal_inout_if<bool>
           ,0
           > request_op;
    ...
  };
```

# Specialized ports, sc_export

```
//FILE: Board.h
#include "Switch.h"
SC_MODULE(Board) {
  Switch switch_i;
sc_fifo<int> t1A, t1B, t1C, t1D;
sc_signal<bool> request[9];
  SC_CTOR(Board): switch_i("switch_i")
  {
    // Connect 4 T1 channels to the switch
    switch_i.T1_ip(t1A);
    switch_i.T1_ip(t1B);
    switch_i.T1_ip(t1C);
    switch_i.T1_ip(t1D);
    // Connect 9 request channels to the
    // switch request output ports
    for (unsigned i=0;i!=9;i++) {
      switch_i.request_op(request[i]);
    }//endfor
    ...
  }//end constructor
  ...
};
```

From preceding example.

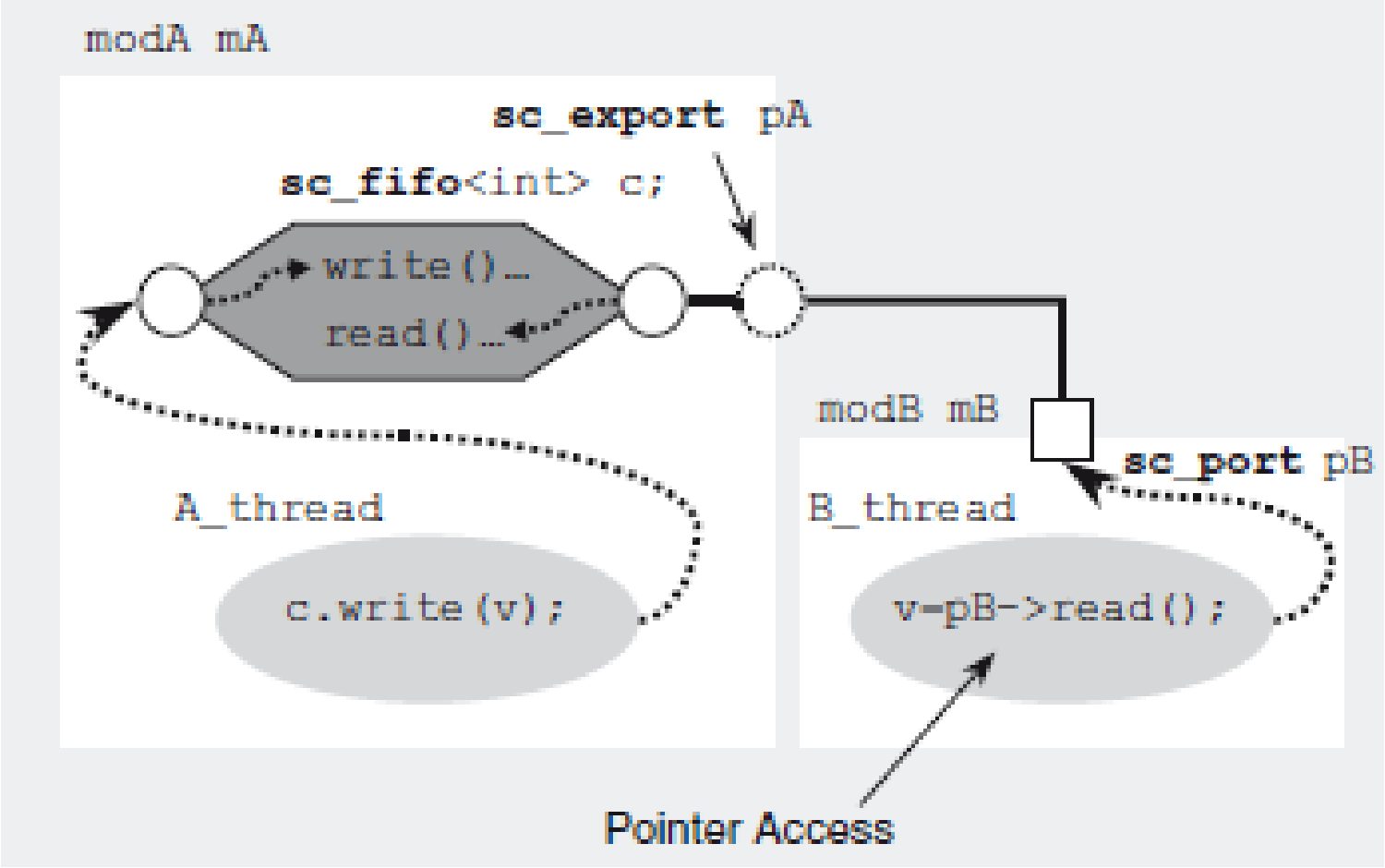
# Specialized ports, sc_export

```
//FILE: Switch.cpp
void Switch::switch_thread() {
  // Initialize requests
  for (unsigned i=0;i!=request_op.size();i++) {
    request_op[i]->write(true);
  }//endfor
  // Startup after first port is activated
  wait(T1_ip[0]->data_written_event()
       |T1_ip[1]->data_written_event()
       |T1_ip[2]->data_written_event()
       |T1_ip[3]->data_written_event()
  );
  while(true) {
    for (unsigned i=0;i!=T1_ip.size();i++) {
      // Process each port...
      int value = T1_ip[i]->read();
    }//endfor
  }//endwhile
}//end Switch::switch_thread
```

# Specialized ports, sc_export

- there is a second type of port called the **sc_export**<*T*>
  - differs in connectivity
  - the idea of an **sc_export**<*T*> is to move the channel inside the defining module
    - hide some of the connectivity details
    - use the port externally as though it were a channel

# Specialized ports, sc_export



# Specialized ports, sc_export

- why use **sc_export**?
  - for an IP provider, it may be desirable to export only specific channels and keep everything else private
    - **sc_export**<*T*> allows control over the interface
  - provide multiple interfaces at the top level
  - communications efficiency down the SystemC hierarchy
    - allows direct access to information (data) without intermediate channels

# Specialized ports, sc_export

```
sc_export<interface> portname;
```

```
SC_MODULE(modulename) {
   sc_export<interface> portname;
   channel cinstance;
   SC_CTOR(modulename) {
      portname(cinstance);
   }
};
```

# Specialized ports, sc_export

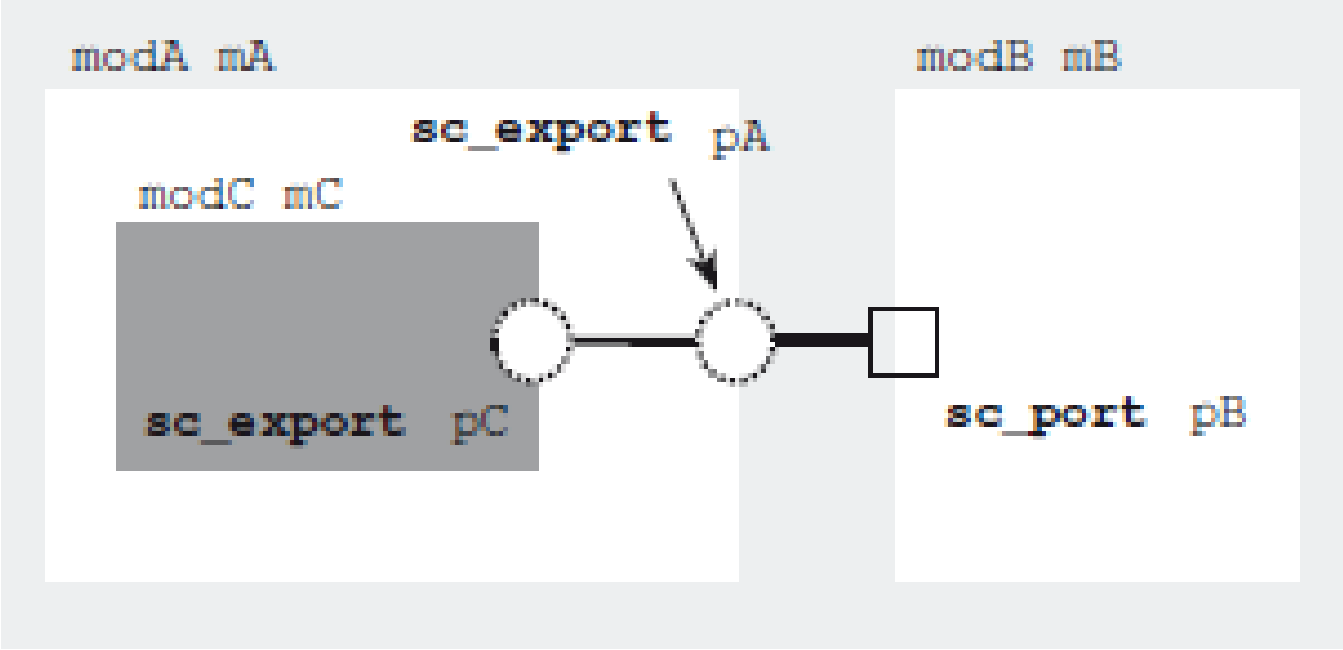
```
SC_MODULE(clock_gen) {
  sc_export<sc_signal<bool>> clock_xp;
  sc_signal<bool> oscillator;
  SC_CTOR(clock_gen) {
    SC_METHOD(clock_method);
    clock_xp(oscillator); // connect sc_signal
                          // channel
                          // to export clock_xp
    oscillator.write(false);
  }
  void clock_method() {
    oscillator.write(!oscillator.read());
    next_trigger(10,SC_NS);
  }
};
```

# Specialized ports, sc_export

```
#include "clock_gen.h"
…
clock_gen clock_gen_i("clock_gen_i");
collision_detector cd_i("cd_i");
// Connect clock
cd_i.clock(clock_gen_i.clock_xp);
…
```

# Specialized ports, sc_export

- **sc_export**<*T*> lets interfaces be passed up the design hierarchy



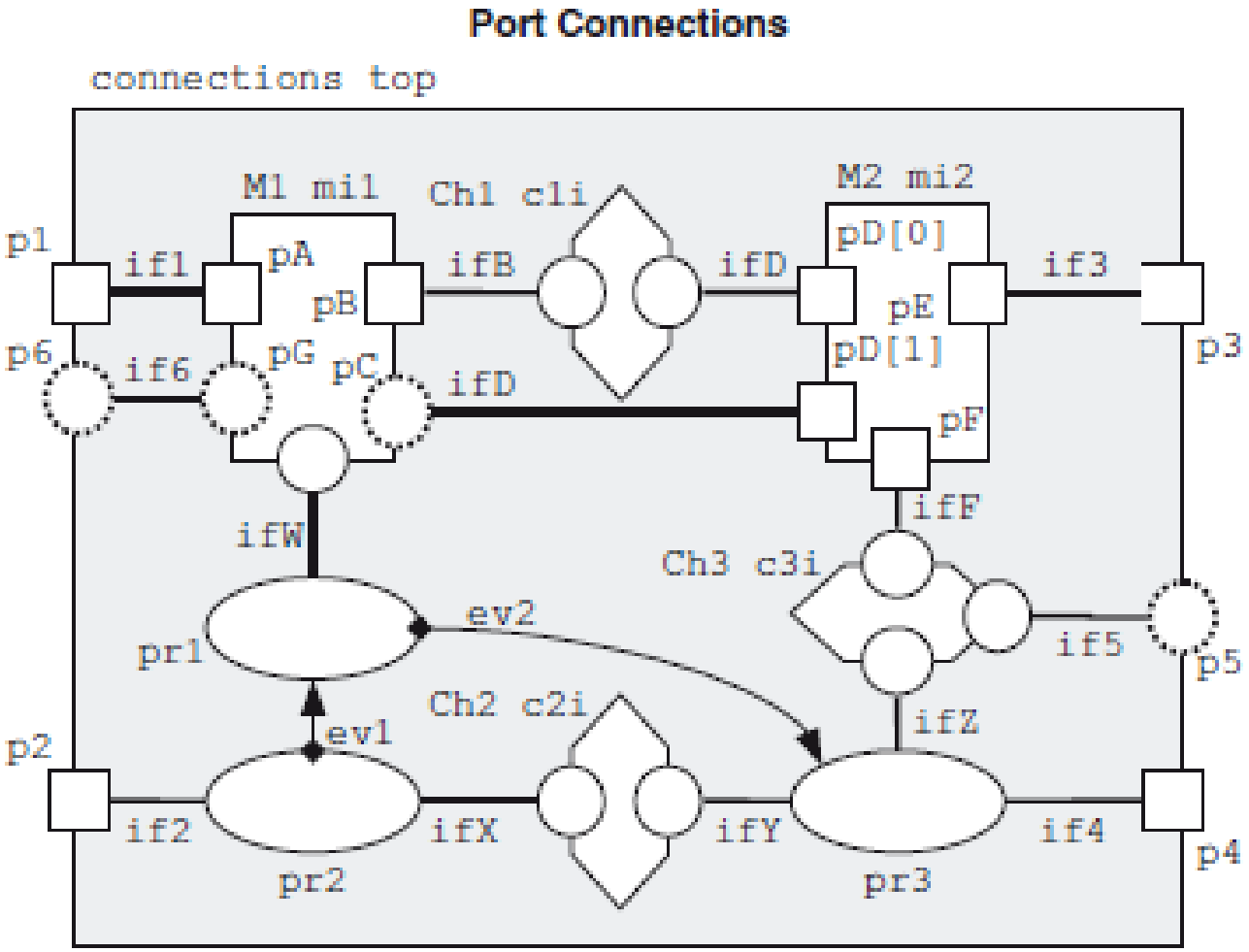
# Specialized ports, sc_export

```
SC_MODULE(modulename) {
   sc_export<interface> xportname;
   module minstance;
   SC_CTOR(modulename)
   , minstance("minstance")
   {
      xportname(minstance.subxport);
   }
};
```

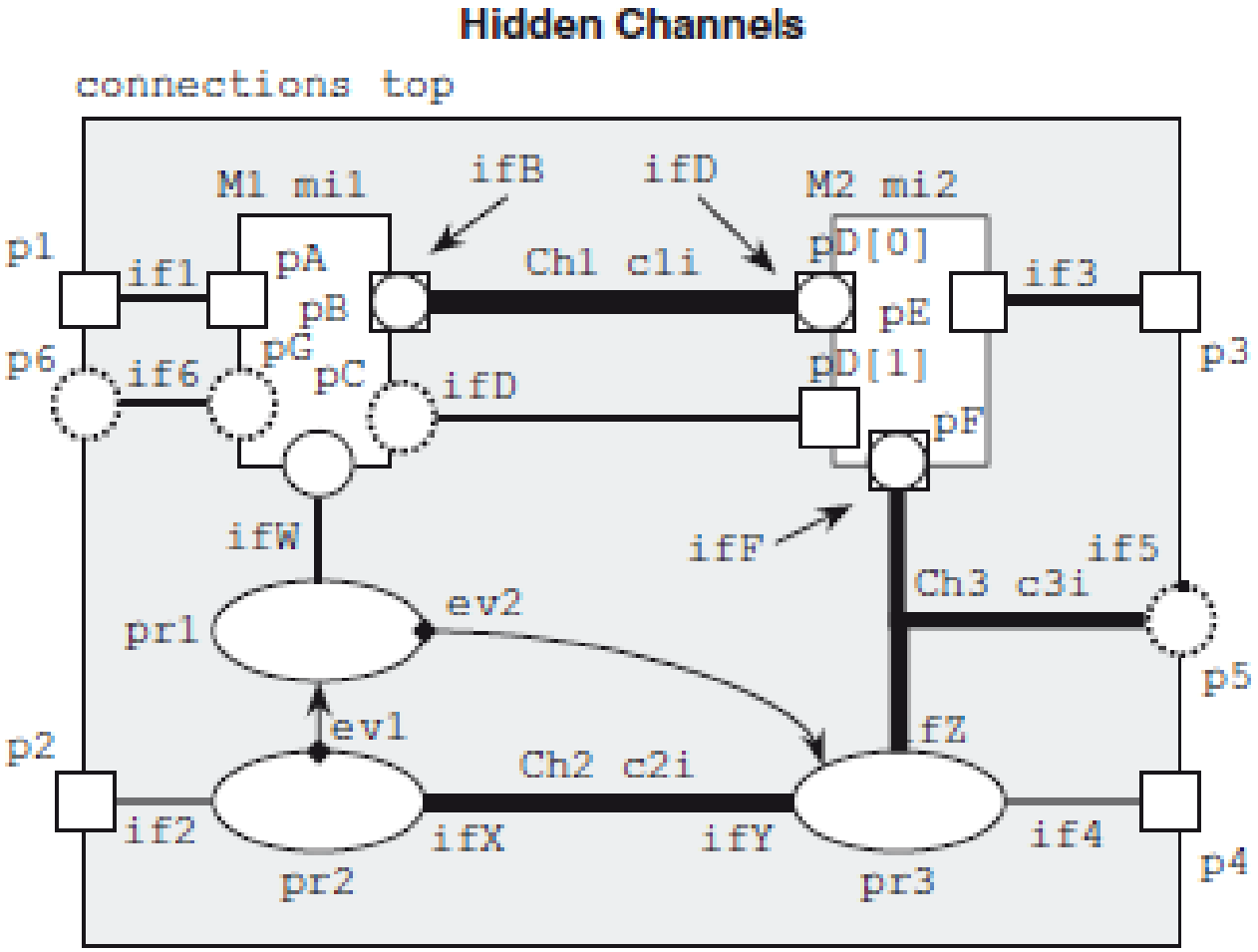
# Specialized ports, sc_export

- **sc_export**<*T*> caveats:
  - it is not possible to use **sc_export**<*T*> in a static sensitivity list
  - it is not possible to have an array of **sc_export**<*T*>
  - it is possible to access the interface via the pointer operator (->)

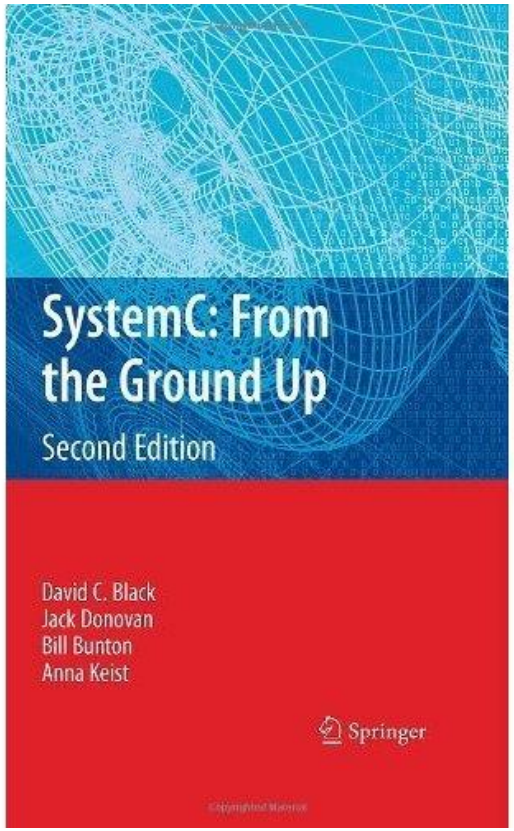
# Specialized ports, sc_export



# Specialized ports, sc_export



# Bibliography




- David C. Black, Jack Donovan, Bill Bunton, Anna Keist, ***SystemC:From the Ground Up,*** Springer Science+Business Media, LLC 2010
  - *"*The authors designed this book primarily for the student or engineer new to SystemC. This book's structure is best appreciated by reading sequentially from beginning to end."