Ports & Interfaces

P. Marwedel*
Informatik 12, U. Dortmund

* Partially using slides prepared by Tatjana Stankovic from the University of Nis (Serbia and Montenegro), visiting the University of Dortmund under the TEMPUS program.

Contents

- Introduction
- Data types
- A Notion of Time
- Modules
- Concurrency
- Structure
- Communication, Channels
- Ports & Interfaces
- Advanced Topics

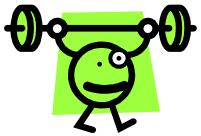
Definition of interfaces and channels

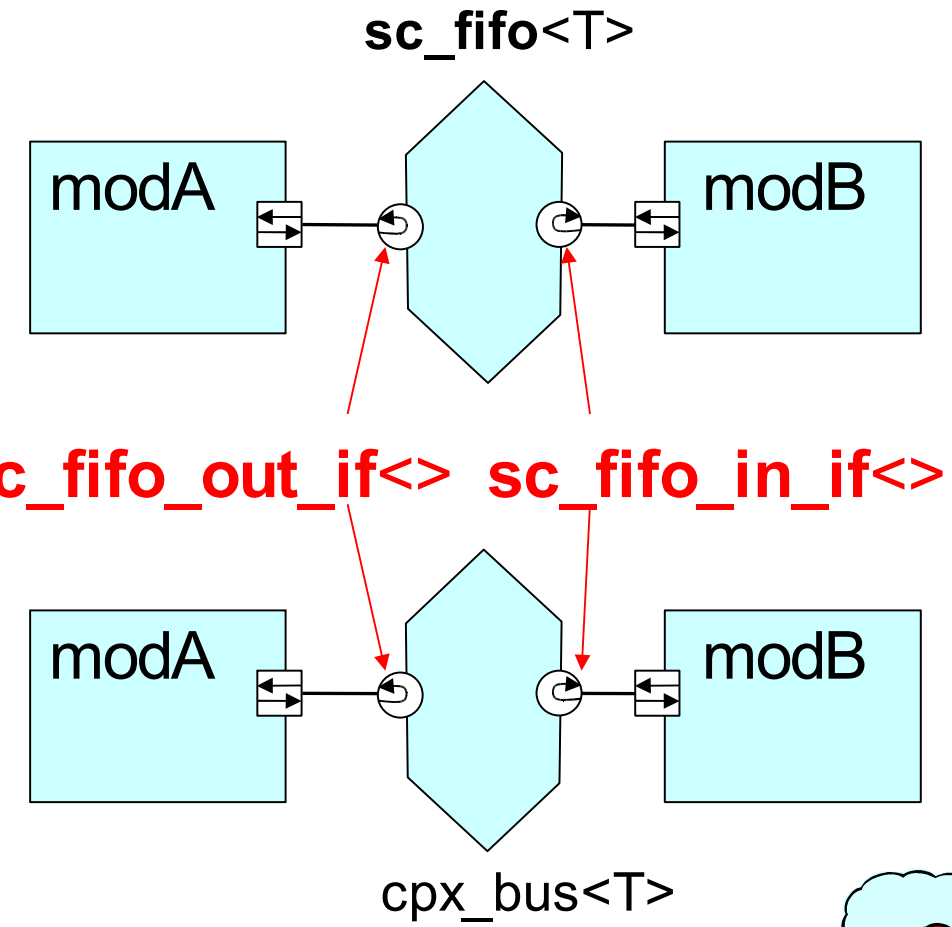
SystemC interface: abstract class inheriting from **sc_interface,** providing only pure virtual declarations of methods referenced by SystemC channels and ports.
No implementations or data are provided in an interface.

SystemC channel: class implementing ≥ 1interface classes, inheriting from **sc_channel** or **sc_prim_channel**.
Channel implements all methods of inherited interface classes.

By using interfaces to connect to channels, we can implement modules *independently* of the implementation details of the communication channels.

Power of interfaces





In one design, modules are connected via a FIFO, in a second design with a complex bus. If the interfaces, in this case **sc_fifo_out_if**<> and **sc_fifo_in_if**<> remain the same, no change to modA and modB is required if we replace the channel.

Key for re-use of intellectual property (IP).

Simple SystemC Port Declarations

Definition: A **SystemC port** is a class templated with and inheriting from a SystemC **interface**.
Ports allow access of channels across module boundaries.

Syntax:
sc_port<*interface*> *portname*; //used @module class definit.

Example:
```
SC_MODULE(stereo_amp) {
 sc_port<sc_fifo_in_if<int> > soundin_p;
 sc_port<sc_fifo_out_if<int> > soundout_p;
 …};
```

Do no omit blank!

Ports

Ports have a **mode** (direction) and a **type**

- **mode:** in, out, inout
- **type**: C++ type, SystemC type, user-defined type
- Examples:
  - **sc_in**<*type*> *in_port_name*; //input port declaration
  - **sc_out**<*type*> *out_port_name*; //output port declaration
  - **sc_inout**<*type*> *inout_port_name*; //bidirectional port
  - **sc_out<int>** result [32]; //vector port/port array

Where to put the ports?

Example:

```
SC_MODULE(adder) {
  sc_in<int> a;
  sc_in<int> b;
  sc_out<int> c;
  // processes, etc
  SC_CTOR(adder) {
        // body of constructor
        // process declaration, sensitivities etc
        c.initialize(0);
  }
};
```

Ports

User can derive specialized ports from the **sc_port** class;
Signal ports are provided by SystemC:

```
template<class T>
  class sc_in : public sc_port<sc_signal_in_if<T> > …;

template<class T>
  class sc_inout : public sc_port<sc_signal_inout_if<T> >
    ..
```

Accessing ports from within a process

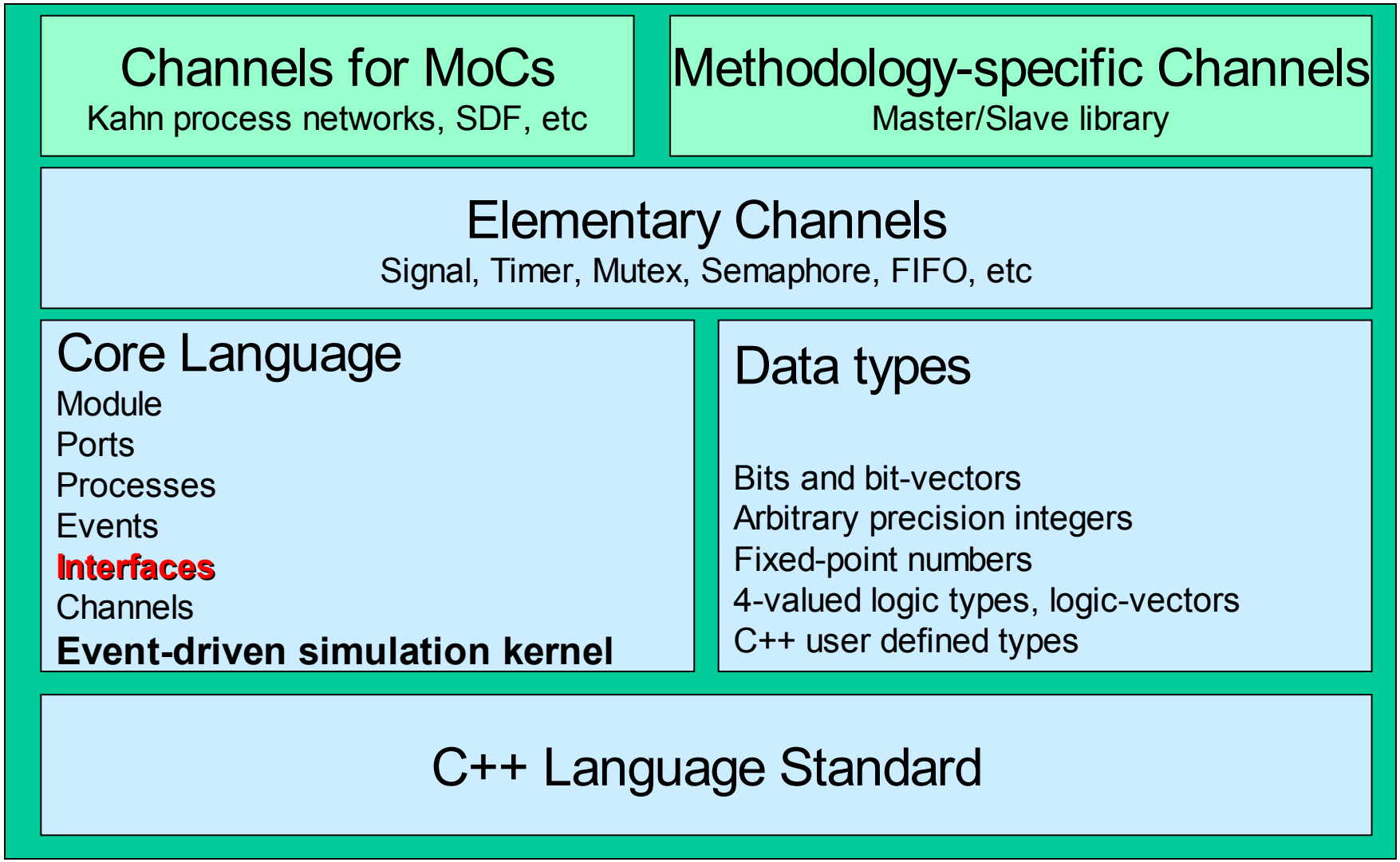
sc_port overloads the -> operator to allow access to ports.

Example:

```
void Video_Mixer::Mixer_thread() {
  …
  switch (control->read()) {
    case MOVIE: K.write(0.0f);    break;
    case MENU:  K.write(1.0f);    break;
    case FADE:  K.write(0.5f);    break;
    default: status->write(Error); break;
   }
…
}
```

Any interface method can be invoked through the port. **read**() and **write**() are interface method calls.

SystemC language architecture



Interfaces

Consist of a set of operations.

Specify only the signature of an operation

- name, parameter, return value

All SystemC interfaces derived from **sc_interface**

- Contains a virtual function called **register_port**()
  - Connects a port to a channel through the interface
  - Arguments: port object and type name of the interface that port accepts

sc_mutex_if

sc_mutex also provides interfaces to be used with ports

```
struct sc_mutex_if: virtual public sc_interface {
  virtual int lock()    = 0;
  virtual int trylock() = 0;
  virtual int unlock()  = 0;
};
```

No method for event sensivity.

sc_semaphore_if

sc_semaphore also provides interfaces to be used with ports

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

Again no method for event sensivity.

sc_fifo interfaces (1)

Two interfaces are provided for **sc_fifo**:

1. **sc_fifo_out_if<>**
2. **sc_fifo_in_if<>**

Together, they provide all methods for **sc_fifo**.

sc_fifo interfaces (2): sc_fifo_out_if<>

```
template <class T>
struct sc_fifo_out_if: virtual public sc_interface {
  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;
  …
};
```

sc_fifo interfaces (3): sc_fifo_in_if<>

```
template <class T>
struct sc_fifo_in_if: virtual public sc_interface {
  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;
  …
};
```

sc_signal interfaces

Interfaces are provided for **sc_signal**:

1. **sc_signal_out_if<>**
2. **sc_signal_in_if<>**
3. **sc_signal_inout_if<>**

Together, they provide all methods for **sc_signal**.

E.g.:

```
template <class T>
struct sc_signal_inout_if: virtual public
sc_signal_in_if<T>
   { virtual void write(const T&) =0;   … };
#define sc_signal_out_if sc_signal_inout_if
```

sc_signal_inout_if provides all methods for the other two interfaces

Ports and static sensivity

It is frequently desirable to describe processes which are sensitive to events on ports.
Ports are essentially pointers, which may be undefined until the end of elaboration (IEEE 1666 insists on this).

- Ports are undefined, when static sensitivity clauses are analyzed.
- Connection to events has to be deferred, events have to be located “later”.
- **Event finders** are methods to be used in static sensivity lists to establish sensitivity to ports.
- In order to avoid the burden of writing event finders, some port specializations provide methods already including event finders.

sc_in<T>

sc_in is a specialization **sc_port<sc_signal_in_if**<T> >
Provided methods include event finders:
sc_in<T> name_sig_ip;
sensitive << name_sig_ip.**value_changed**();
// for T= **bool** and T= **sc_logic**:
sc_in<bool> name_bool_sig_ip;
sc_in<sc_logic> name_log_sig_ip;
sensitive << name_log_ip_sig.**pos**();
sensitive << name_bool_ip_sig.**neg**();

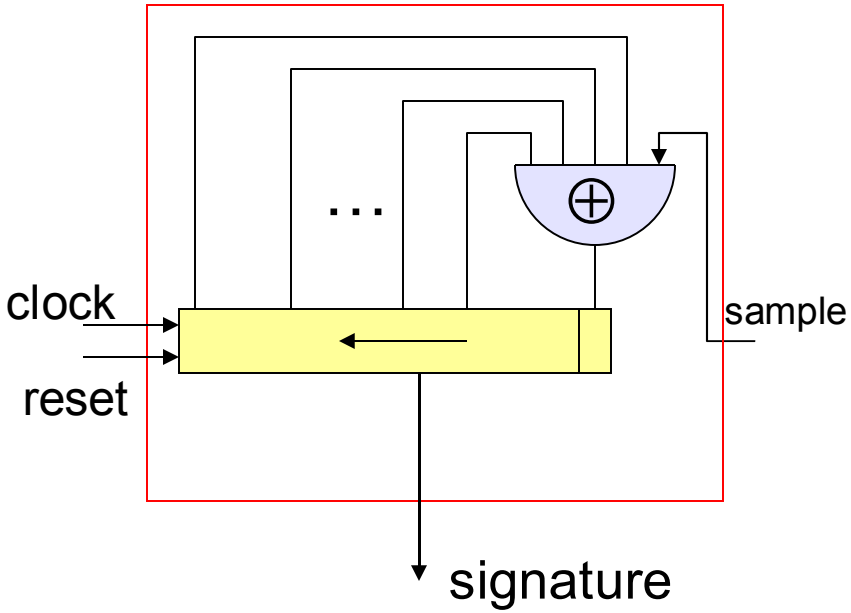
"."-notation discouraged but needed in declaration
"->" - notation preferred.

sc_out<T>

Similarly, **sc_out** provides methods **value_changed()** and **initialize()**

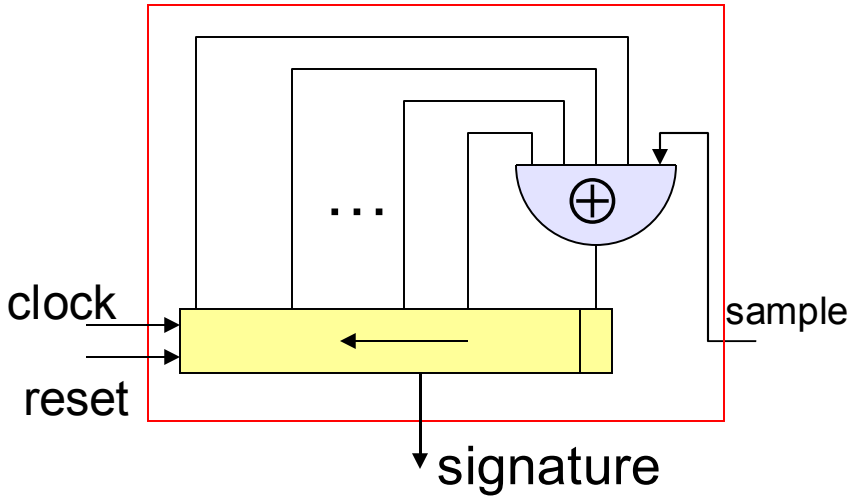
Example: header

```
//file LFSR_ex.h
#ifndef LFSR_EX_H
#define LFSR_EX_H
SC_MODULE (LFSR_ex) // ports
  sc_in<bool> sample;
  sc_out<sc_int<32> > signature;
  sc_in<bool> clock;
  sc_in<bool> reset;
  SC_CTOR(LFSR_ex){
    SC_METHOD(LFSR_ex_method);
    sensitive << clock.pos() << reset;
    signature.initialize(0);
   }
  void LFSR_ex_method();
  sc_int<32> LFSR_reg;  };
#endif
```

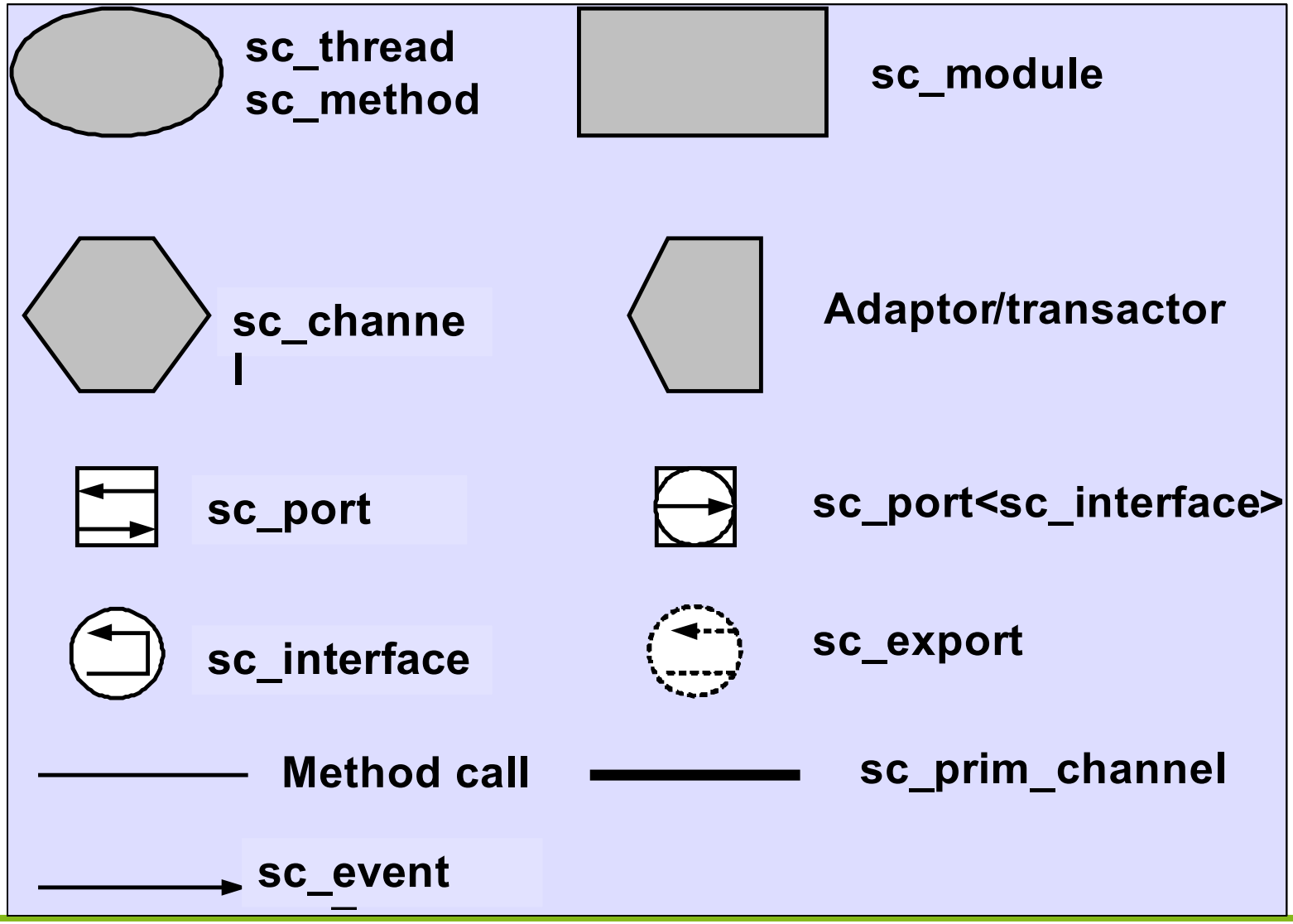


Example: Implementation

```
//file LFSR_ex.cpp
#include <systemc.h>
#include "LFSR_ex.h"
void LFSR_ex::LFSR_ex_method() {
  if (reset->read() == true) {
     LFSR_reg = 0;
      signature->write(LFSR_reg);
  } else {
    bool lsb = LFSR_reg[31] ^ LFSR_reg[25] ^ LFSR_reg[22]
    ^ LFSR_reg[21] ^ LFSR_reg[15] ^ LFSR_reg[11] ^ LFSR_reg[10]
    ^ LFSR_reg[ 9] ^ LFSR_reg[ 7] ^ LFSR_reg[ 6] ^ LFSR_reg[ 4]
    ^ LFSR_reg[ 3] ^ LFSR_reg[ 1] ^ LFSR_reg[ 0] ^ sample->read();
  LFSR_reg.range(31,1) = LFSR_reg.range(30,0); LFSR_reg[0] = lsb;
  signature->write(LFSR_reg);
 }//endif
}
```



Standard Graphical Notation



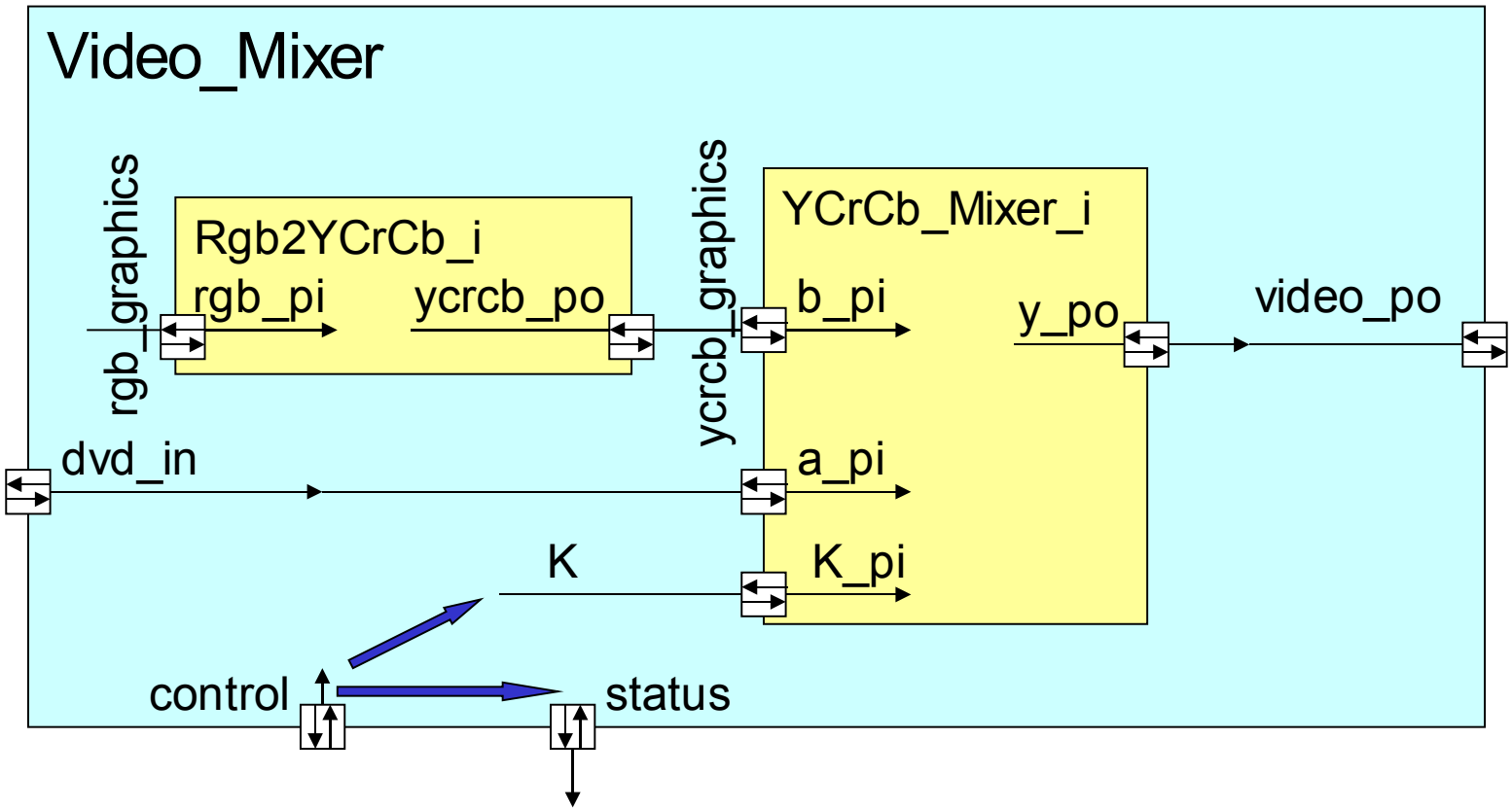
Port connection mechanics

Modules are connected to channels after modules and channels have been instantiated.

Syntax:

mod_instance(*channel_instance*, …); // positional (risky)
mod_inst.portname(*channel_instance*); // named (safer)

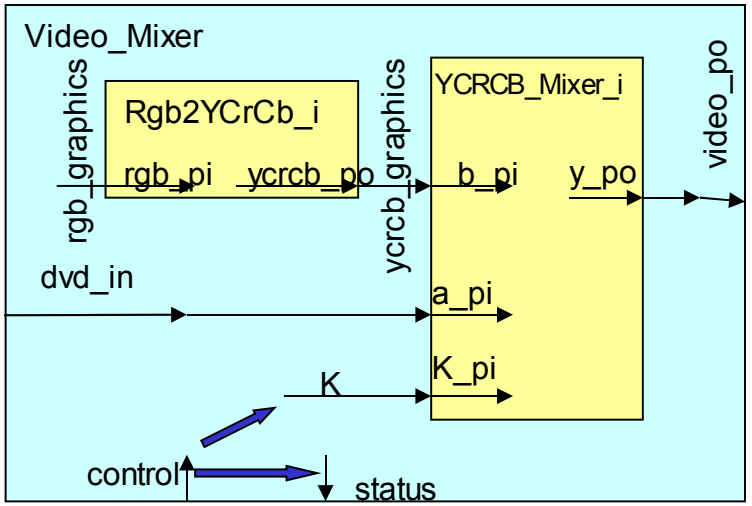
Example: Video mixer



as (in)complete as in "the source"

Example: Video mixer (2)

```
//File: Rgb2YCrCb.h
SC_MODULE(Rgb2YCrCb) {
 sc_port<sc_fifo_in_if<RGB_frame> > rgb_pi;
 sc_port<sc_fifo_out_if<YCRCB_frame> >
ycrcb_po;
};
```

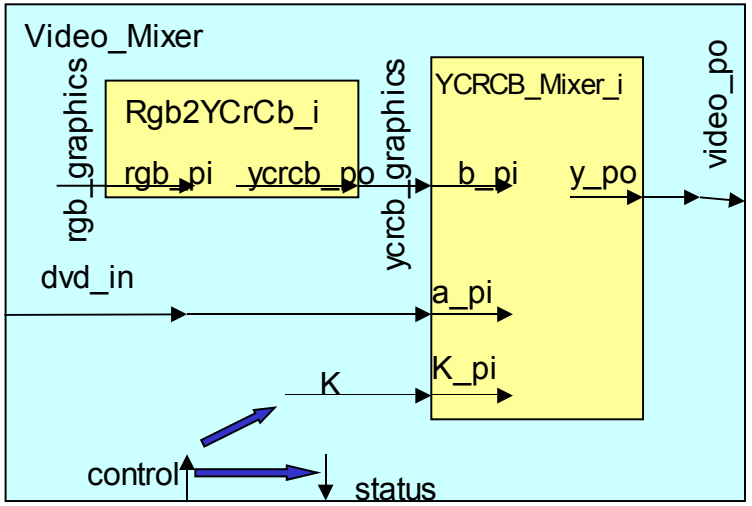



```
//File: YCRCB_Mixer.h
SC_MODULE(YCRCB) {
 sc_port<sc_fifo_in_if<float> > K_pi;
 sc_port<sc_fifo_in_if<YCRCB_frame> > a_pi, b_pi;
 sc_port<sc_fifo_out_if<YCRCB_frame> > y_po;
};
```

Example: Video mixer (3)

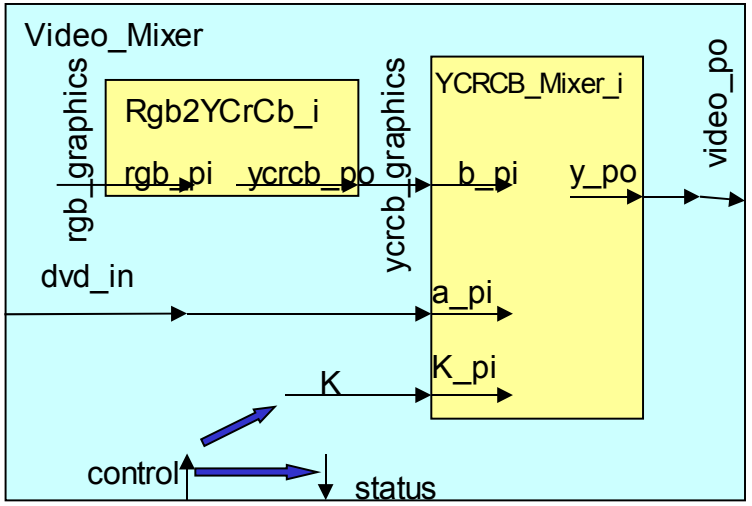
```
//File: Video_Mixer.h
SC_MODULE (VIDEO_MIXER) {
// ports
sc_port<sc_fifo_in_if<YCRCB_frame> > dvd_in;
sc_port<sc_fifo_out_if<YCRCB_frame> >
video_po;
sc_port<sc_fifo_in_if<MIXER_ctrl> > control;
sc_port<sc_fifo_out_if<MIXER_state> > status;
//local channels
sc_fifo<float> K;
sc_fifo<RGB_frame> rgb_graphics;
sc_fifo<YCRCB_frame> ycrcb_graphics;
// constructor
SC_HAS_PROCESS(VIDEO_MIXER);
VIDEO_MIXER(sc_module_name nm);*
void Mixer_thread();
};
```

*__sc_module_name__ is the type of the module name (internally managed strings), which must be the 1st parameter if SC_CTOR is not used.



Example: Video mixer (4)

```
VIDEO_Mixer::VIDEO_Mixer(sc_module_name nm)
:sc_module(nm)
{
  //Instantiate
  Rgb2YCrCb Rgb2YCrCb_i("Rgb2YCrCb_i");
  YCRCB_Mixer YCRCB_Mixer_i(" YCRCB_Mixer_i
");
  //Connect
  Rgb2YCrCb_i.rgb_pi(rgb_graphics);
  Rgb2YCrCb_i.ycrcb_po(ycrcb_graphics);
  YCRCB_Mixer_i.K_pi(K);
  YCRCB_Mixer_i.a_pi(dvd_in);
  YCRCB_Mixer_i.b_pi(ycrcb_graphics);
  YCRCB_Mixer_i.y_po(video_po);
};
```

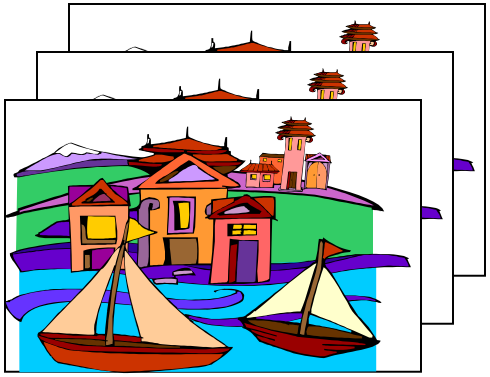


Port arrays

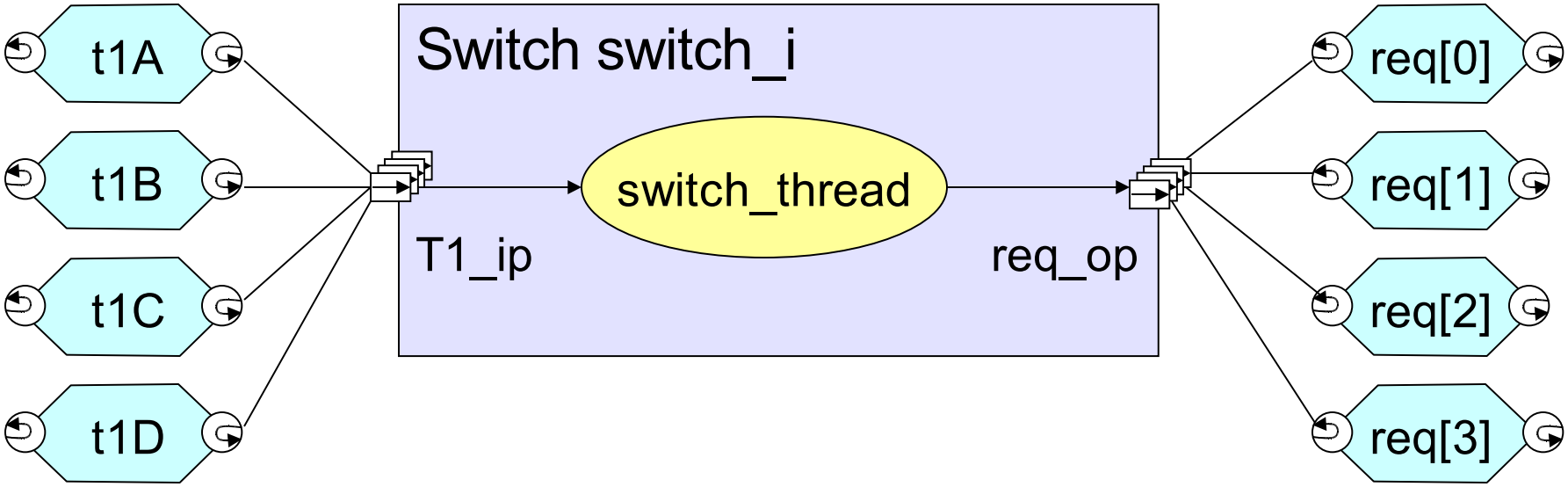
Facility for describing an array of ports.
Can be used, for example, for describing multiplexer inputs.

Syntax:

sc_port <*interface*[,N]> *portname*; // N=0..max, default=1
// if N=0, an unlimited number of interfaces may be
// connected to the port.



Example



```
//File Switch.h
SC_MODULE(Switch) {
  sc_port<sc_fifo_in_if<int>, 4> T1_ip;
  sc_port<sc_fifo_out_if<int>, 0> req_op; …}
```

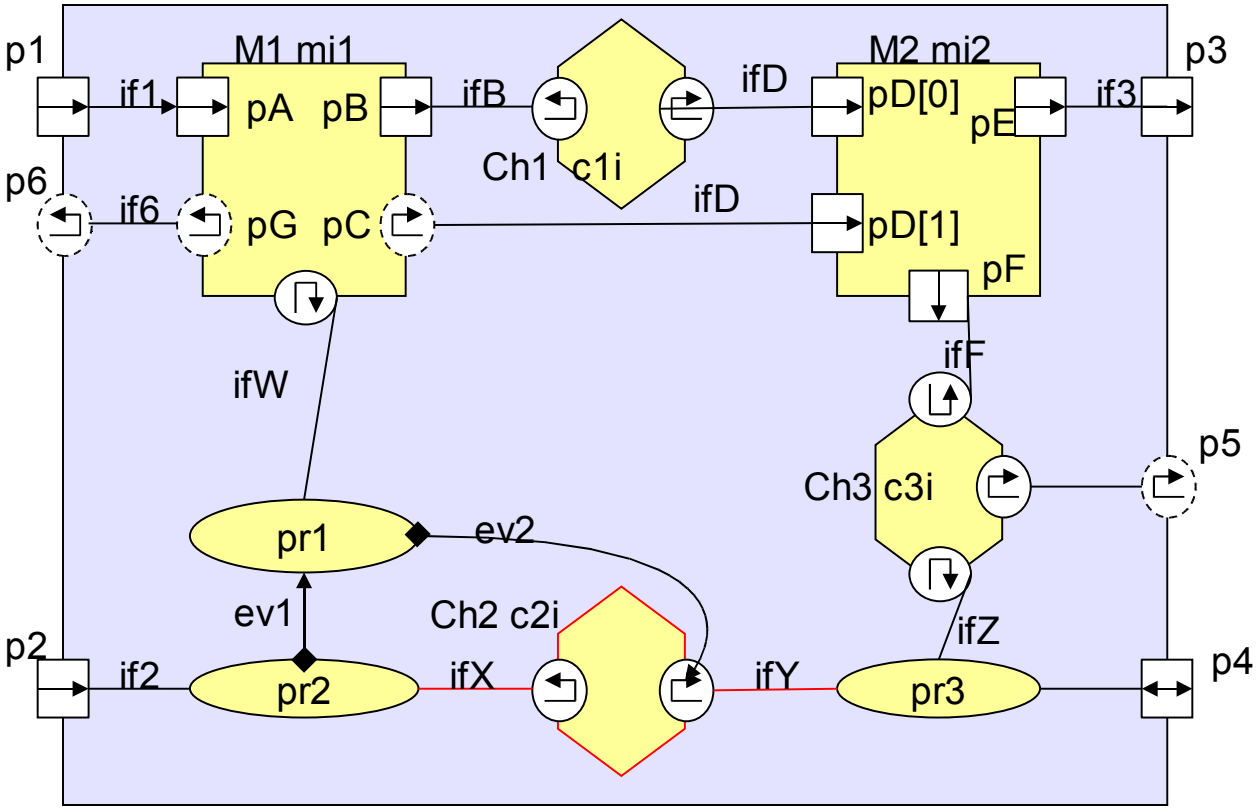
Array Connections

```
// File: board.h
#include "Switch.h"
SC_MODULE(Board) {
 const unsigned REQS;
 Switch switch_i;
 sc_fifo<int> t1A, t1B, t1C, t1D;
 sc_signal<bool> req[9];
 SC_CTOR(Board): switch_i("switch_i");
   {//connect 4 channels to the switch
   switch_i.T1_ip(t1A); switch_i.T1_ip(t1B);
   switch_i.T1_ip(t1C); switch_i.T1_ip(t1D);
   for (unsigned i=0;i!=4;i++) {
      switch_i.req_op(req[i]);
} } };
```

Process code

```
//File: Switch.cpp
void Switch::switch_thread() {
   for (unsigned i=0;i!=req.size();i++) {
     req[i]->write(true);
   }
   //startup after first port is activated
  wait(T1_ip[0]->data_written_event() //returns reference to event
        | T1_ip[1]->data_written_event() //that occurs when data is
        | T1_ip[2]->data_written_event() //written (method of sc_fifo).
        | T1_ip[3]->data_written_event() );
  for (;;) {
    for (unsigned i=0; i!=T1_ip.size();i++) {
       // process each port ..
       int value = T1_ip[i]->read();…
}}}
```

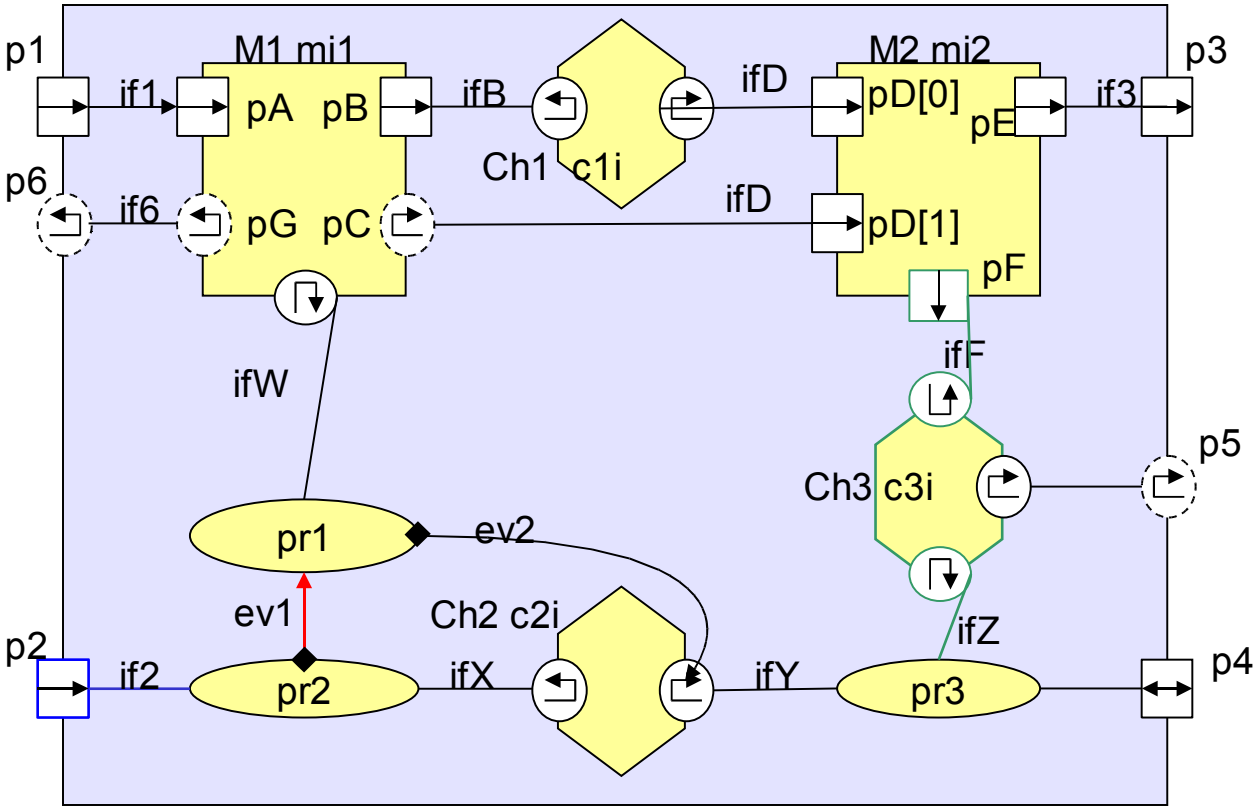
Ways to connect (1)



- pD is an array port
- mi1 implements an interface ifW (☞ custom channels, not explained in course)
- processes may communicate with other processes at the same level using channels (see pr2 & pr3)
- pC is of type **sc_export** (SystemC 2.1: channel moved into module, port can be used like a channel)

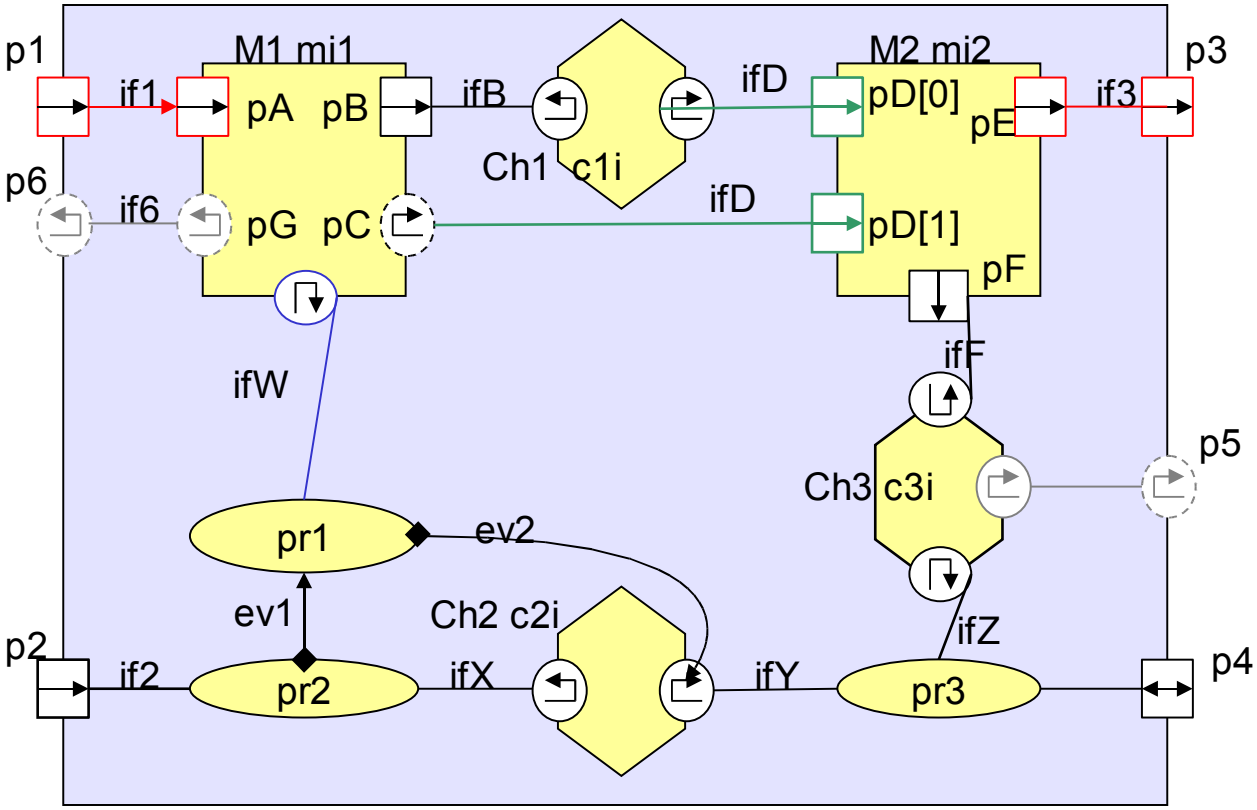
- Channel c1i implements interfaces ifB, ifD,
- channel c2i implements interfaces ifX, ifY,
- channel c3i implements interfaces ifF, if5, ifZ.

Ways to connect (2)



- Processes may synchronize with other processes at the same level using event (see pr1 & pr2)
- Processes may communicate upwards in the hierarchy via ports using interfaces (see pr2 & p2).
- Processes may communicate with sub-modules via channels connected to sub-module ports (see pr3 & mi2).

Ways to connect (3)



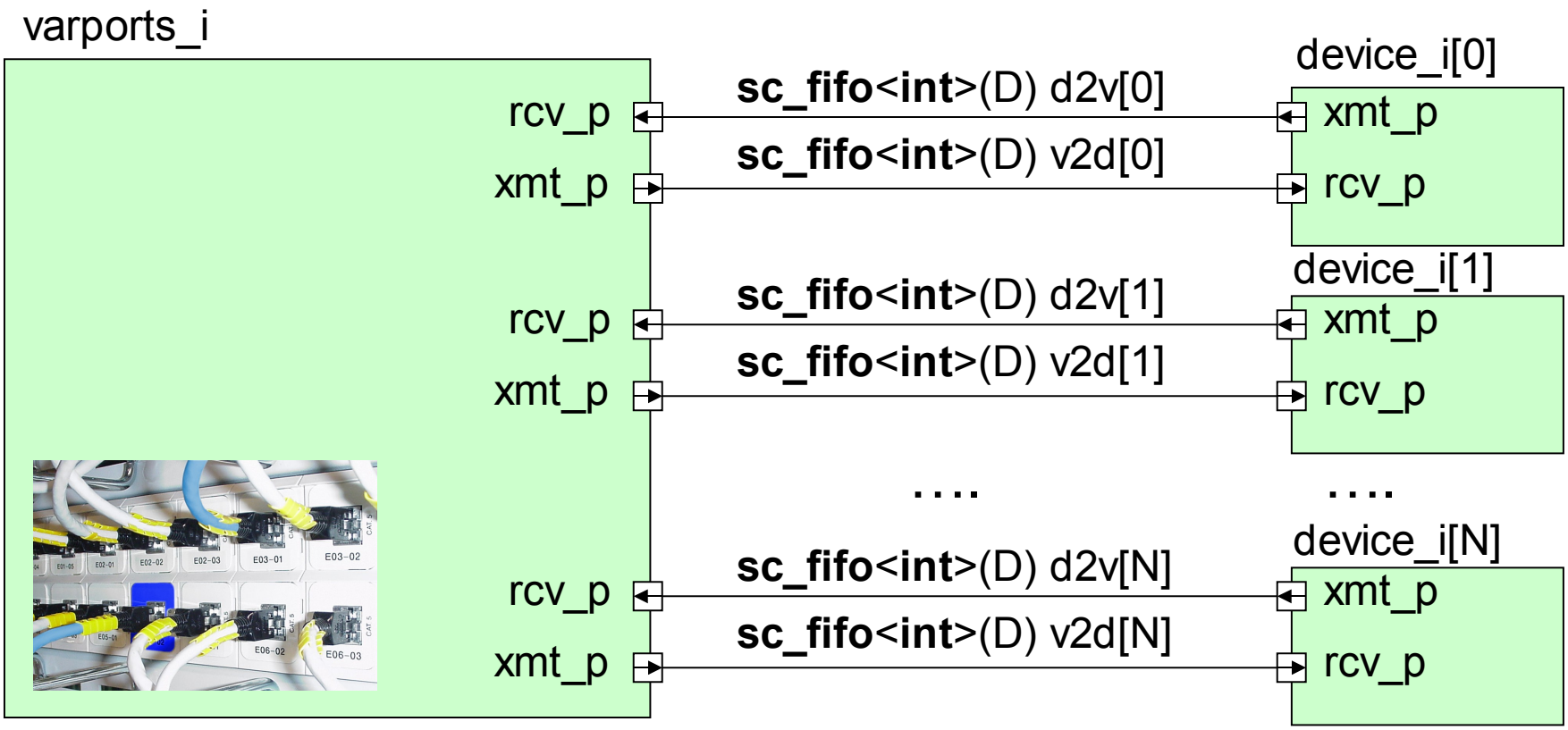
- Processes may communicate with sub-modules by letting the process access the port interface (see pr1 & mi1).
- **sc_ports** may connect directly to ports of sub-modules (see p1 & mi1)
- port arrays can share the same interface (see pD).
- **sc_exports** may connect to **sc_exports** of submodules and channels.

Ways to connect (4)

From	To	Method
Port	Sub-module	Direct connect via **sc_port**
Process	Port	Direct access by process
Sub-module	Sub-module	Local channel connection
Process	Sub-module	Local channel connect or via **sc_export** or interface implemented by sub-module (hierarchical channel)
Process	Process	Events or local channel
Port	Local channel	Direct connect via **sc_export**

Programmable Hierarchy

Feasible to use control structures like **for** or **if then else** etc during generation of structures. Example:



Configura-ble code

```
#include <systemc.h>
#include "varports.h"
#include "device.h"
int sc_main(int argc, char* argv[]) { // get N from cmd line ..
  varports* varports_i;
  device* device_i[N];
  sc_fifo<int>* v2d[N];
  sc_fifo<int>* d2v[N];
  varports_i = new varports(…init parameters…);
  for (unsigned i=0; i<N; i++) {
    sc_string nm;
    nm = sc_string::to_string("device_name_i[%d]",i);
    device_i[i] = new device(nm.c_str());
    nm = sc_string::to_string("v2d[%d]",i);
    v2d[i] = new sc_fifo<int>(nm.c_str(),fifo_depth);
    nm = sc_string::to_string("d2v[%d]",i);
    d2v[i] = new sc_fifo<int>(nm.c_str(),fifo_depth);
    device_i[i]->rcv_p(*v2d[i]);
    device_i[i]->xmt_p(*d2v[i]);
    varports_i->rcv_p(*d2v[i]);
    varports_i->xmt_p(*v2d[i]); }
```

Summary

- Ports
- Interfaces
- Ports and static sensivity
- **pos** and **neg** methods
- LFSR example
- Connecting ports
- Video mixer example
- Port arrays
- Ways to connect
- Programmable hierarchy