SISTEME DE CALCUL DEDICATE

Curs 5

Outline

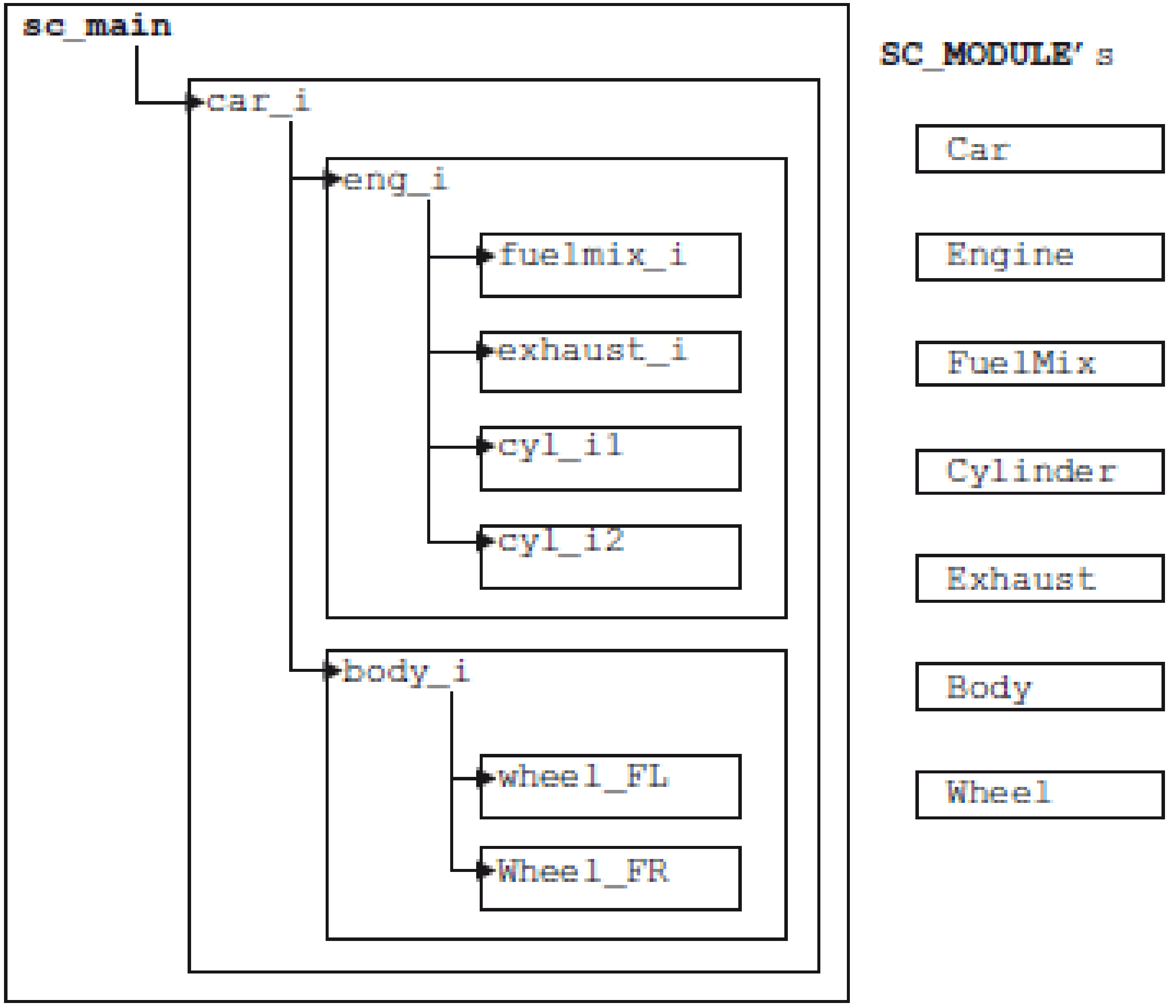
- SystemC
  - Design hierarchy
  - Ports
- Bibliography

Design hierarchy

- design hierarchy
  - hierarchical relationships of modules
  - connectivity that lets modules communicate in an orderly fashion
  - in SystemC uses instantiations of modules as member data of parent modules
    - to create a level of hierarchy, create an **sc_module** object within a parent **sc_module**.

Design hierarchy

Design Hierarchy



sc_main

car_i

eng_i

fuelmix_i

exhaust_i

cyl_i1

cyl_i2

body_i

wheel_FL

Wheel_FR

SC_MODULE's

Car

Engine

FuelMix

Cylinder

Exhaust

Body

Wheel

Design hierarchy

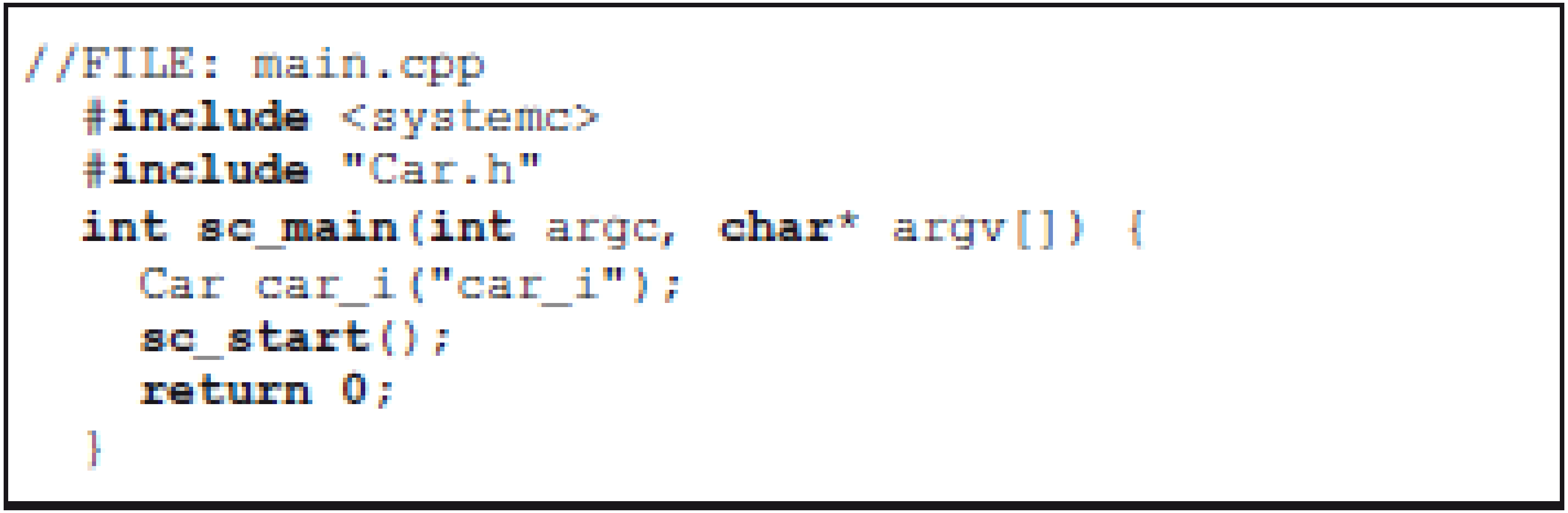
- C++ offers two basic ways to create submodule objects
  - a submodule object may be created ***directly*** by declaration
  - a submodule object may be ***indirectly*** referenced by means of a pointer in combination with dynamic allocation

Design hierarchy

- six approaches
  - Direct top-level (**sc_main**)
  - Indirect top-level (**sc_main**)
  - Direct submodule header-only
  - Direct submodule
  - Indirect submodule header-only
  - Indirect submodule

Design hierarchy

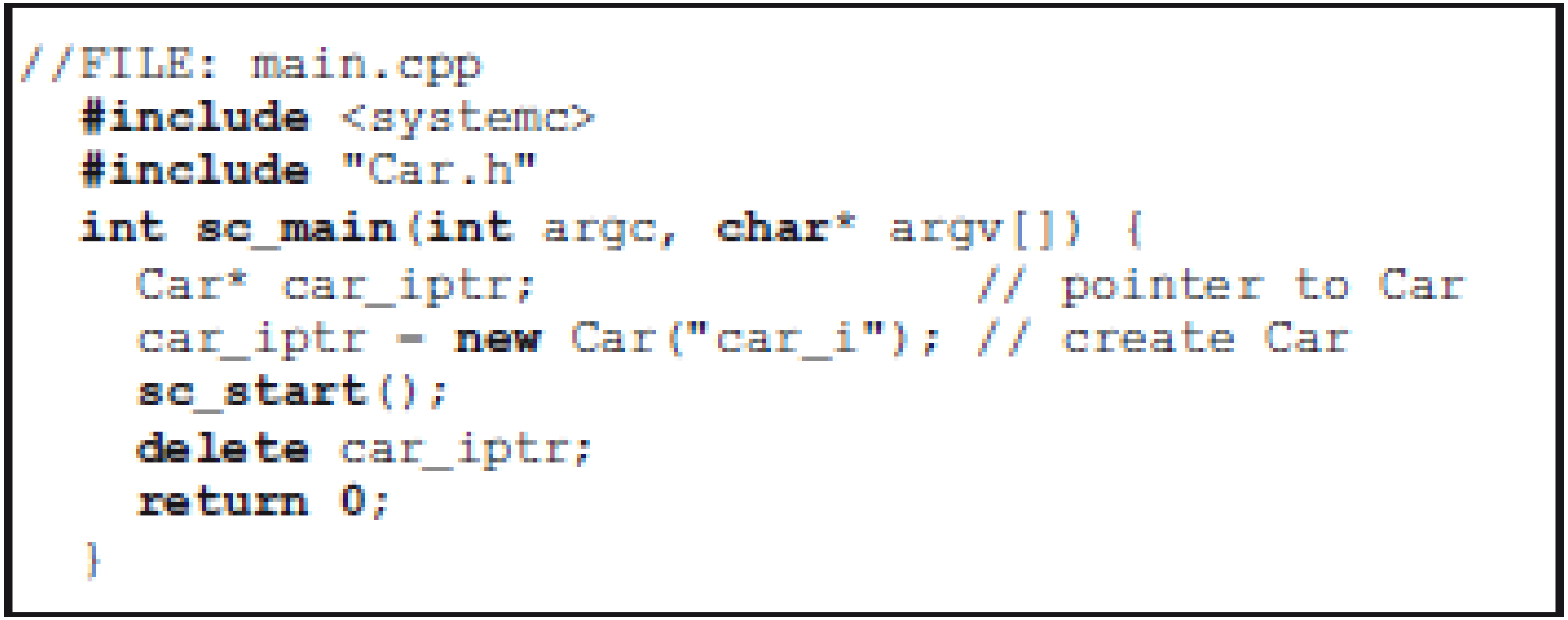
- top-level implementation with direct instantiation



```
//FILE: main.cpp
  #include <systemc>
  #include "Car.h"
  int sc_main(int argc, char* argv[]) {
    Car car_i("car_i");
    sc_start();
    return 0;
  }
```

Design hierarchy

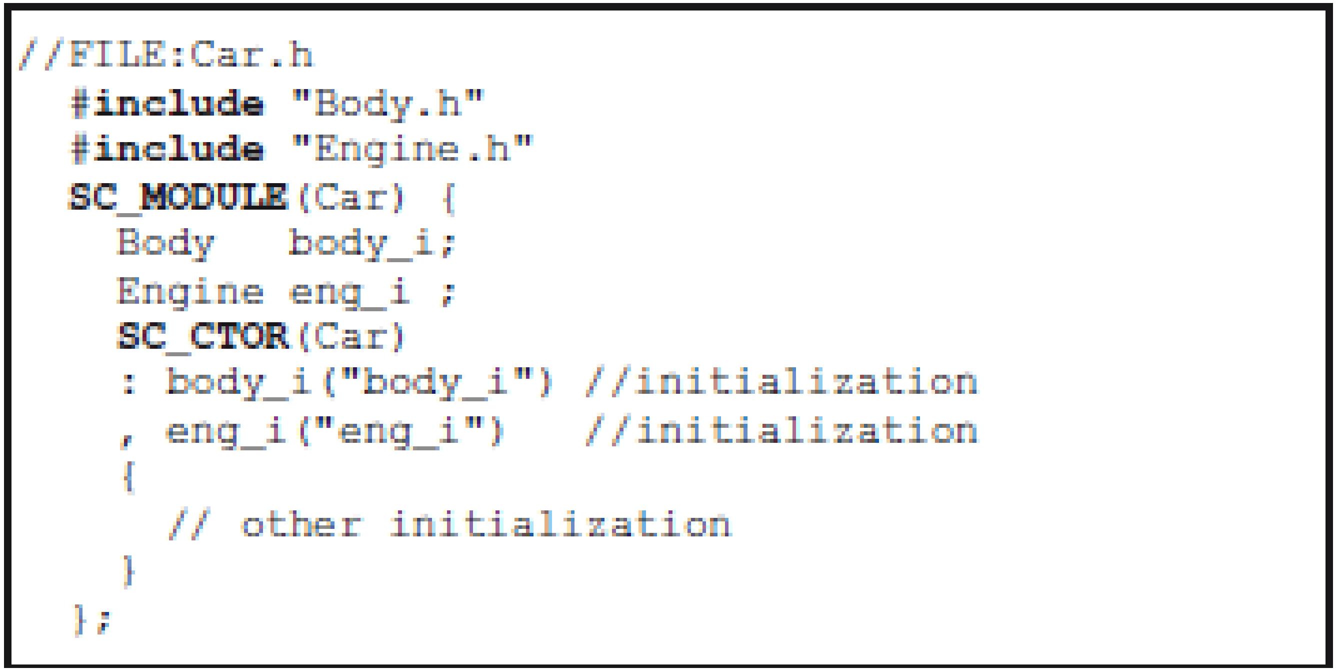
- top-level implementation with indirect instantiation



```
//FILE: main.cpp
  #include <systemc>
  #include "Car.h"
  int sc_main(int argc, char* argv[]) {
    Car* car_iptr;               // pointer to Car
    car_iptr = new Car("car_i"); // create Car
    sc_start();
    delete car_iptr;
    return 0;
  }
```

Design hierarchy

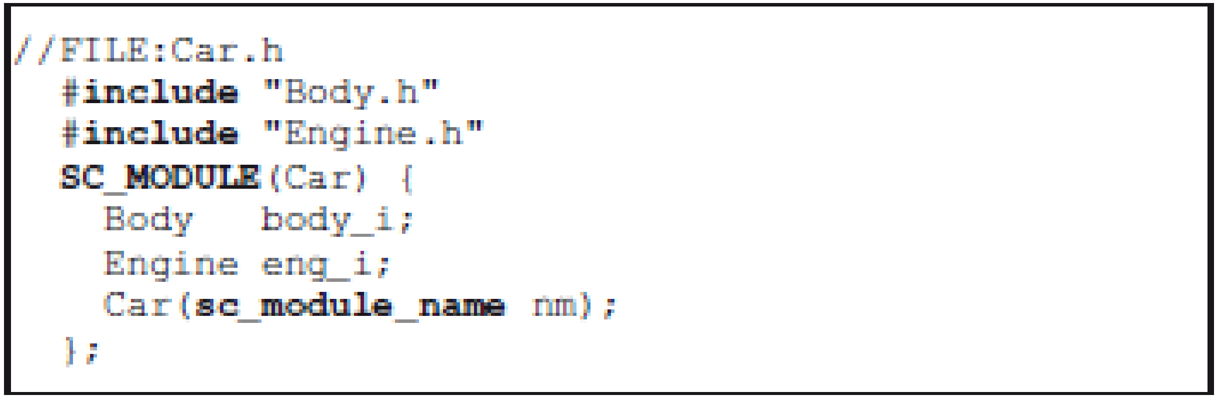
- direct instantiation in the header
  - use of an initializer list



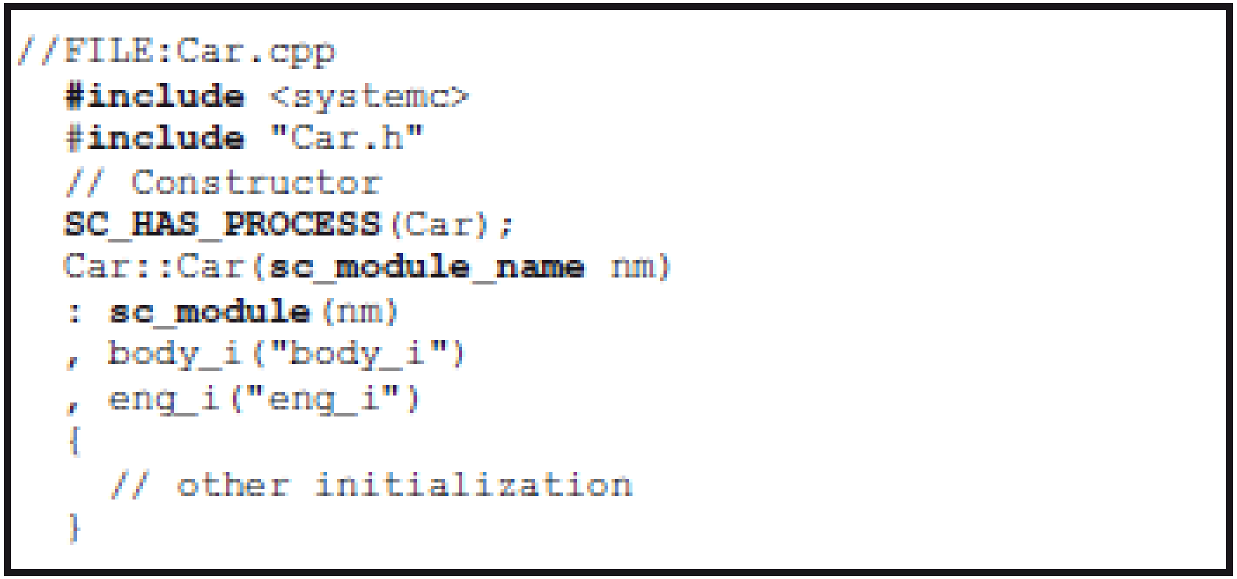
```
//FILE:Car.h
  #include "Body.h"
  #include "Engine.h"
  SC_MODULE(Car) {
    Body   body_i;
    Engine eng_i ;
    SC_CTOR(Car)
    : body_i("body_i") //initialization
    , eng_i("eng_i")   //initialization
    {
      // other initialization
    }
  };
```

Design hierarchy

- direct instantiation and separate compilation



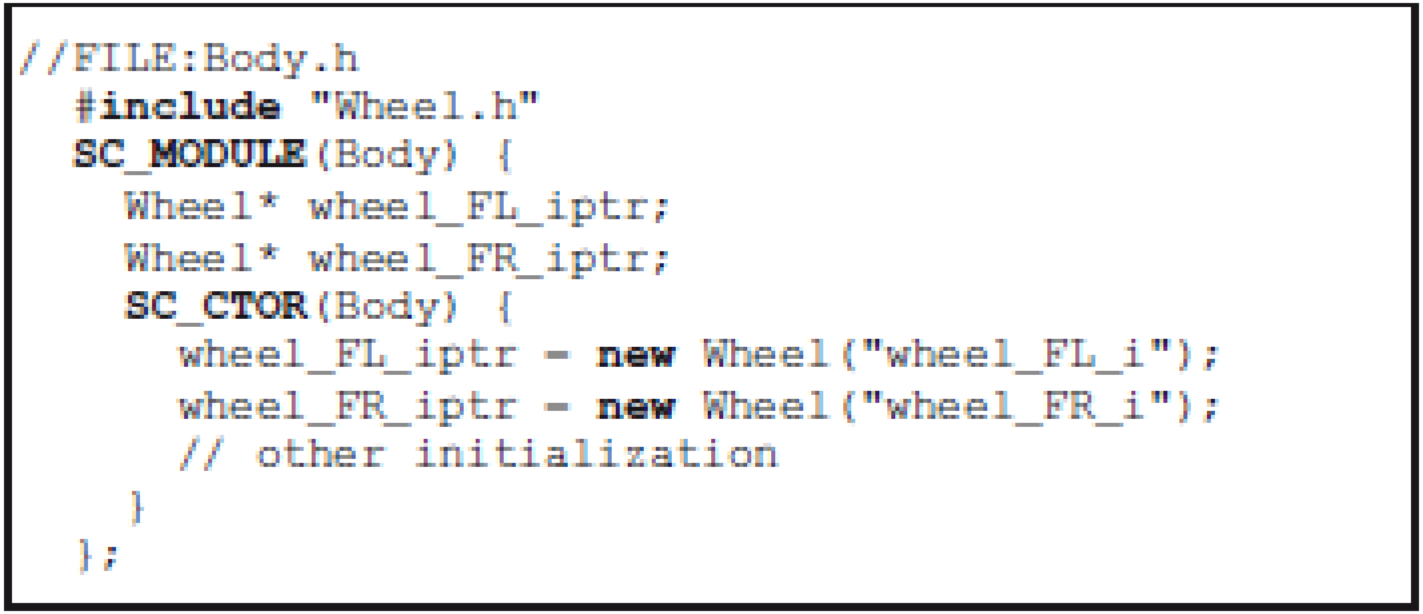
```
//FILE:Car.h
  #include "Body.h"
  #include "Engine.h"
  SC_MODULE(Car) {
    Body   body_i;
    Engine eng_i;
    Car(sc_module_name nm);
  };
```



```
//FILE:Car.cpp
  #include <systemc>
  #include "Car.h"
  // Constructor
  SC_HAS_PROCESS(Car);
  Car::Car(sc_module_name nm)
  : sc_module(nm)
  , body_i("body_i")
  , eng_i("eng_i")
  {
    // other initialization
  }
```

Design hierarchy

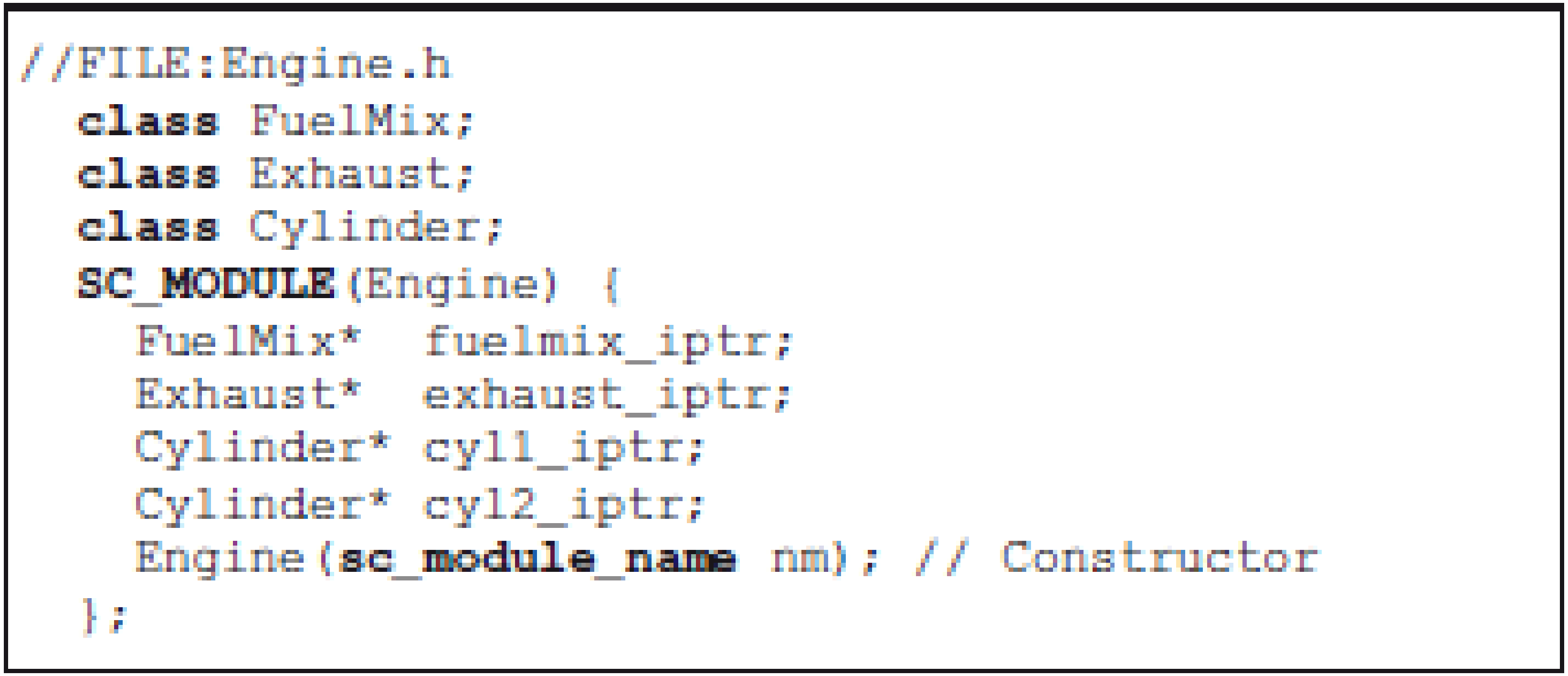
- Indirect Submodule Header-Only Implementation



```
//FILE:Body.h
  #include "Wheel.h"
  SC_MODULE(Body) {
    Wheel* wheel_FL_iptr;
    Wheel* wheel_FR_iptr;
    SC_CTOR(Body) {
      wheel_FL_iptr = new Wheel("wheel_FL_i");
      wheel_FR_iptr = new Wheel("wheel_FR_i");
      // other initialization
    }
  };
```

Design hierarchy

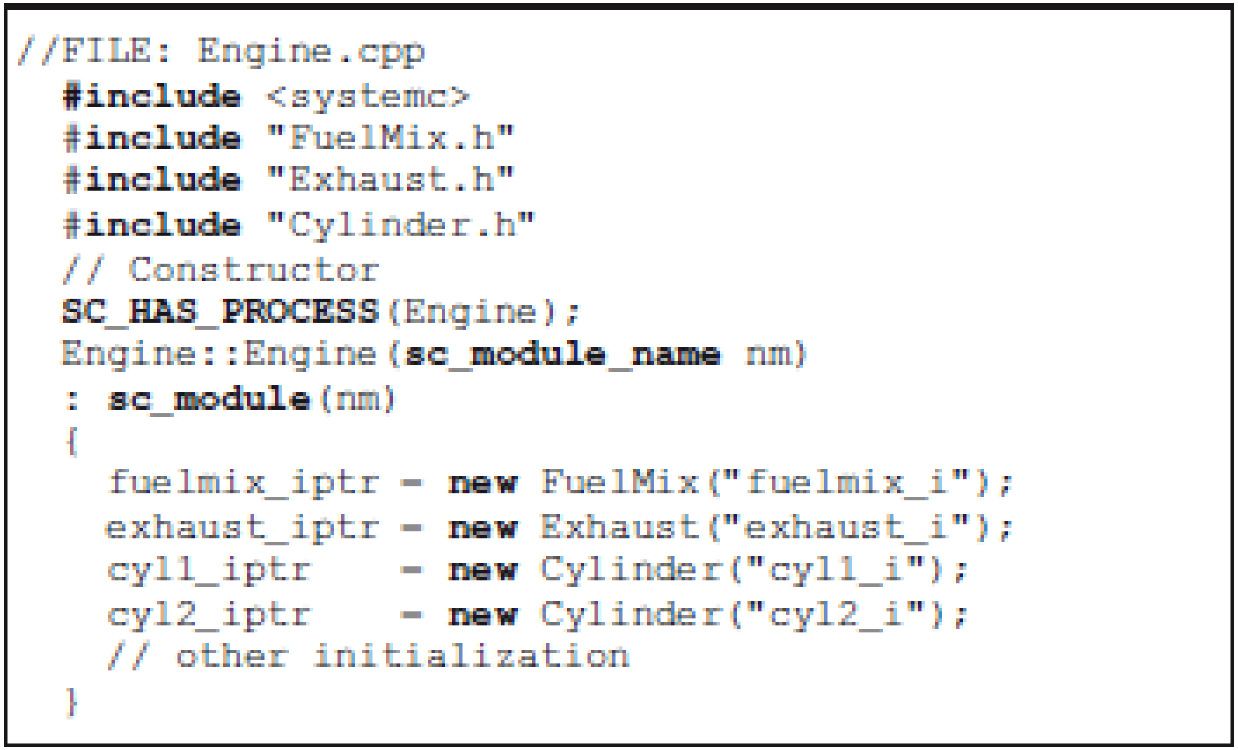
- Indirect Submodule Implementation



```
//FILE:Engine.h
  class FuelMix;
  class Exhaust;
  class Cylinder;
  SC_MODULE(Engine) {
    FuelMix*   fuelmix_iptr;
    Exhaust*   exhaust_iptr;
    Cylinder*  cyl1_iptr;
    Cylinder*  cyl2_iptr;
    Engine(sc_module_name nm); // Constructor
  };
```

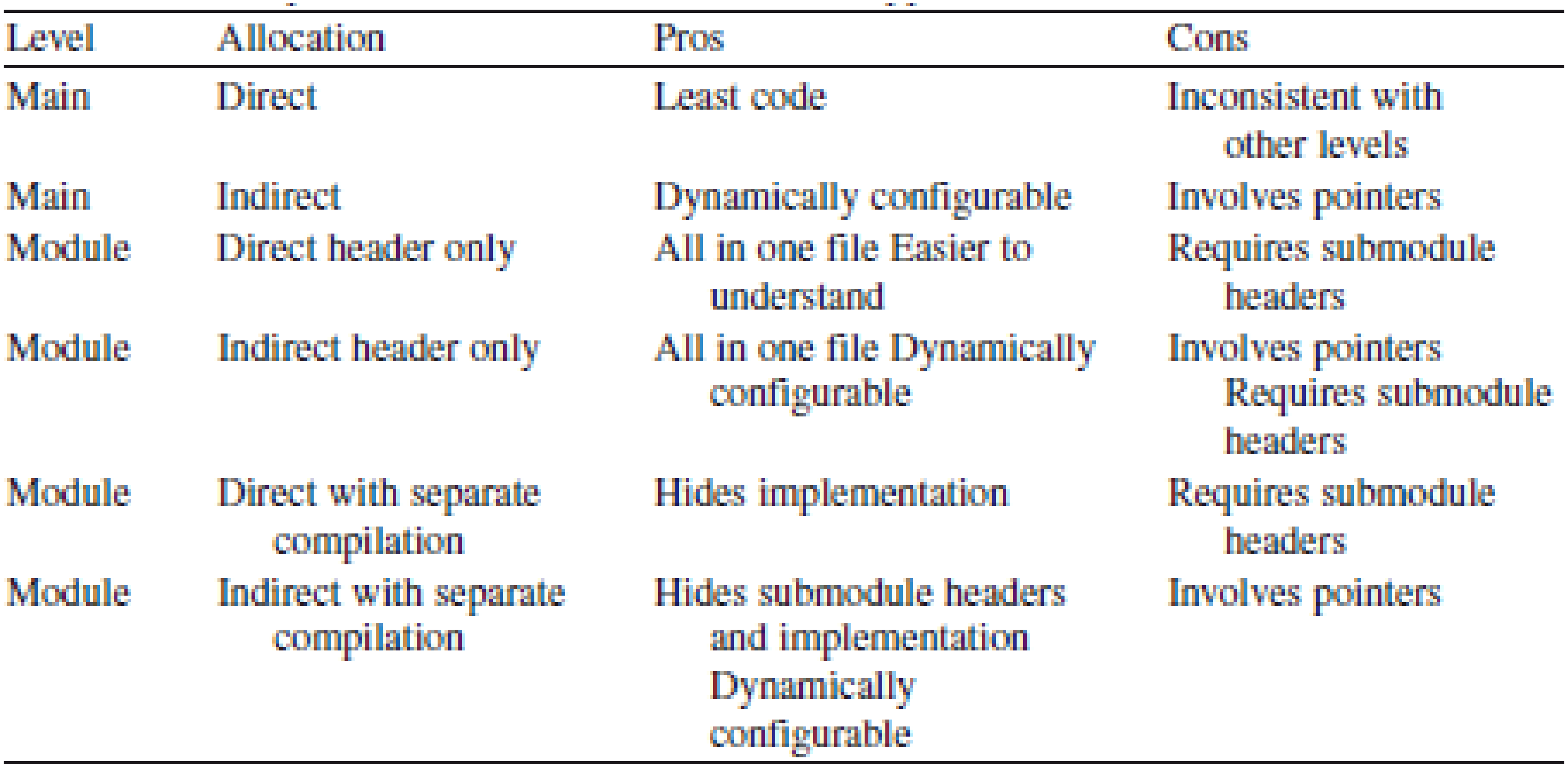
Design hierarchy

- Indirect Submodule Implementation
  - good for IP distribution



```
//FILE: Engine.cpp
  #include <systemc>
  #include "FuelMix.h"
  #include "Exhaust.h"
  #include "Cylinder.h"
  // Constructor
  SC_HAS_PROCESS(Engine);
  Engine::Engine(sc_module_name nm)
  : sc_module(nm)
  {
    fuelmix_iptr = new FuelMix("fuelmix_i");
    exhaust_iptr = new Exhaust("exhaust_i");
    cyl1_iptr    = new Cylinder("cyl1_i");
    cyl2_iptr    = new Cylinder("cyl2_i");
    // other initialization
  }
```

Design hierarchy



Level	Allocation	Pros	Cons
Main	Direct	Least code	Inconsistent with other levels
Main	Indirect	Dynamically configurable	Involves pointers
Module	Direct header only	All in one file Easier to understand	Requires submodule headers
Module	Indirect header only	All in one file Dynamically configurable	Involves pointers Requires submodule headers
Module	Direct with separate compilation	Hides implementation	Requires submodule headers
Module	Indirect with separate compilation	Hides submodule headers and implementation Dynamically configurable	Involves pointers

Ports

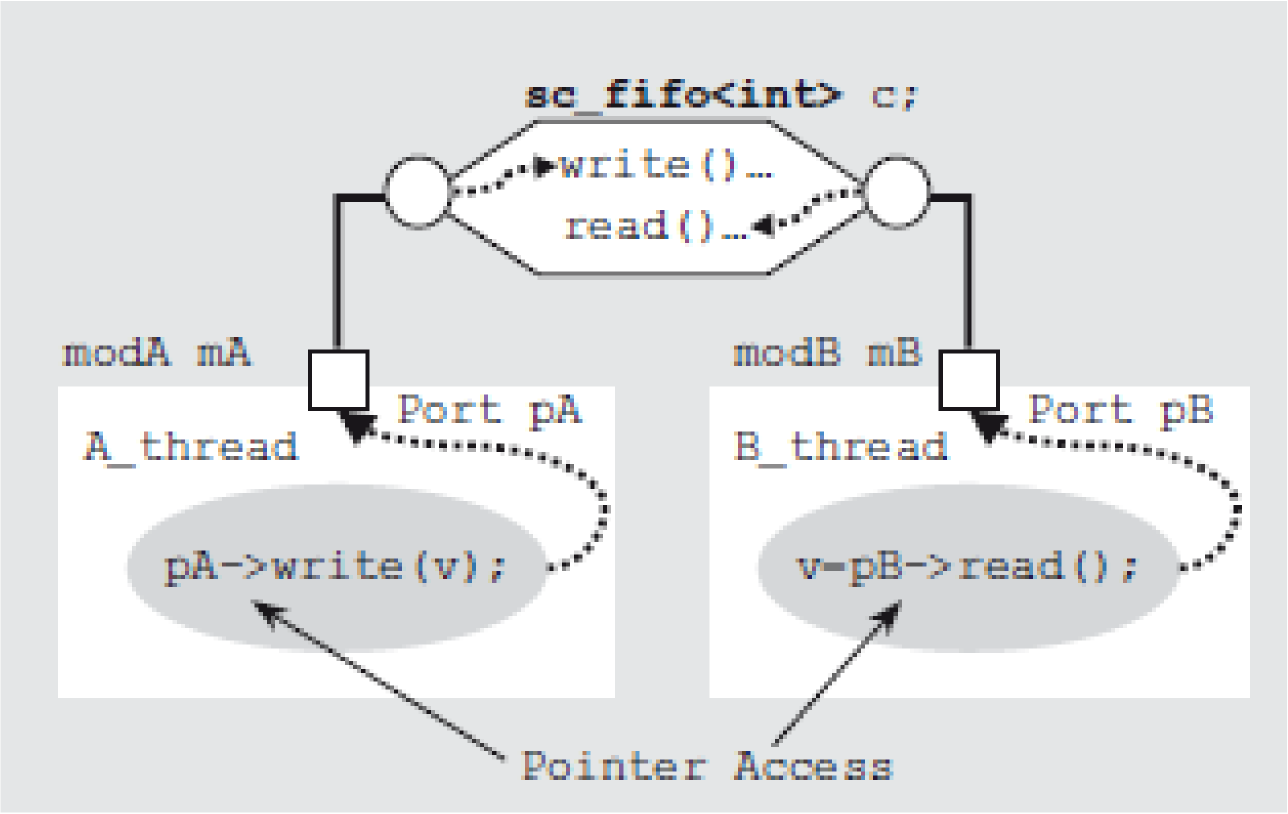
- what is the best way to communicate?
  - safety
    - is a concern because all activity occurs within processes
    - care must be taken when communicating between processes to avoid race conditions.
    - events and channels are used to handle this concern

Ports

- what is the best way to communicate?
  - easy of use
    - dispense with any solution involving global variables
    - a process that monitors and manages events defined in instantiated modules (awkward)
    - SystemC takes an approach that lets modules use channels inserted between the communicating modules
    - a concept called a port
      - a pointer to a channel outside the module

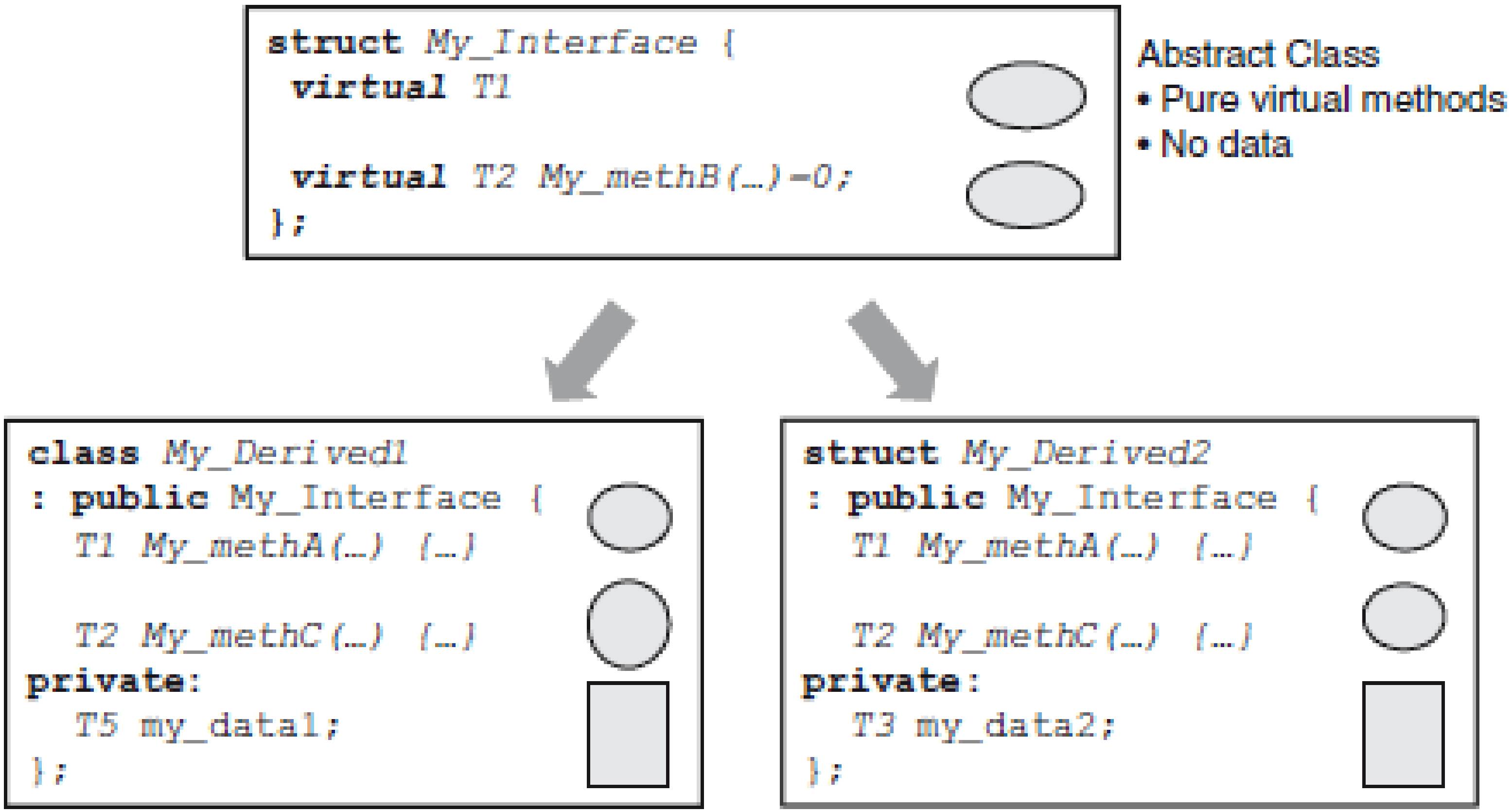
Ports

Communication Via sc_ports



Ports

C++ Interface Relationships



```
struct My_Interface {
 virtual T1

 virtual T2 My_methB(…)=0;
};
```

Abstract Class
- Pure virtual methods
- No data

```
class My_Derived1
: public My_Interface {
   T1 My_methA(…) {…}

   T2 My_methC(…) {…}
private:
   T5 my_data1;
};
```

```
struct My_Derived2
: public My_Interface {
   T1 My_methA(…) {…}

   T2 My_methC(…) {…}
private:
   T3 my_data2;
};
```

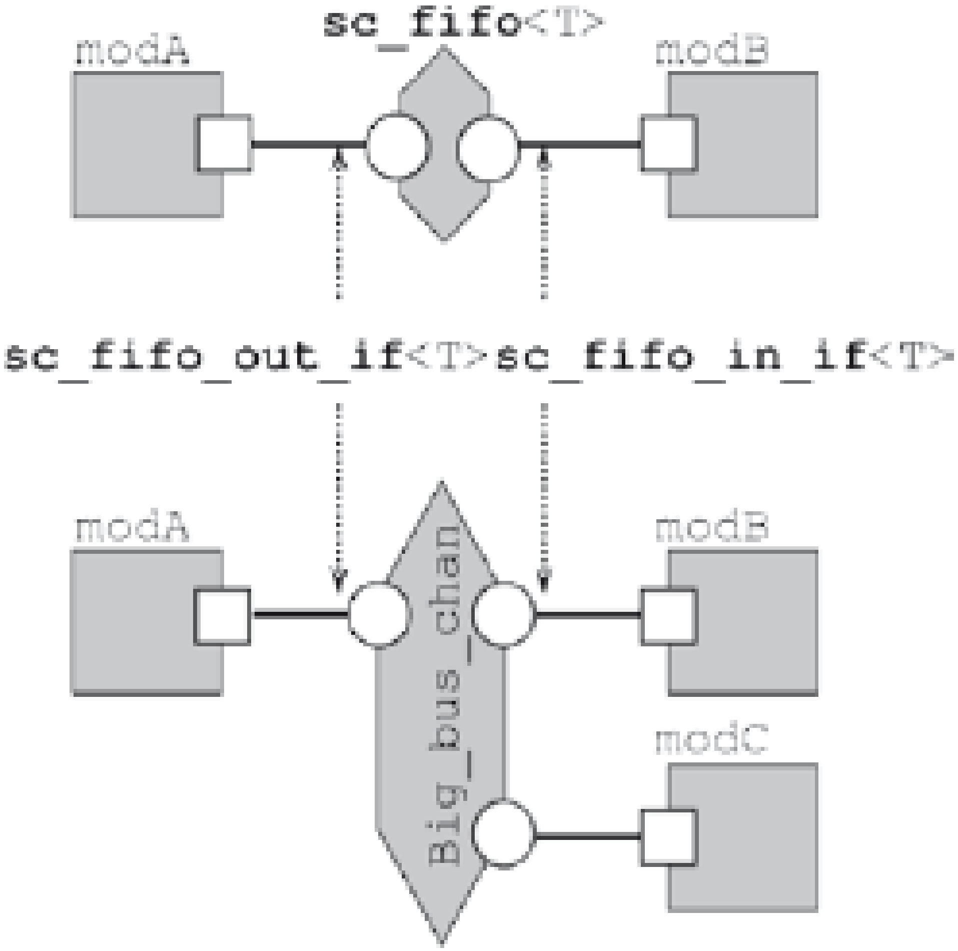
Ports

- **DEFINITION**: A SystemC interface is an abstract class that inherits from **sc_interface** and provides only pure virtual declarations of methods referenced by SystemC channels and ports. No implementations or data are provided in a SystemC interface.

Ports

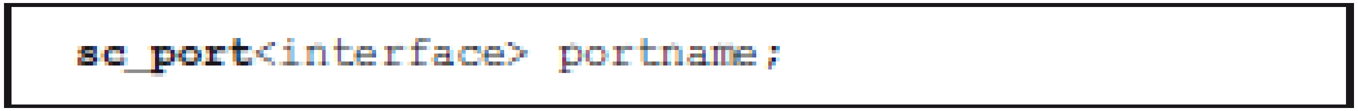
- **DEFINITION**: A SystemC channel is a class that inherits from either **sc_channel** or from **sc_prim_channel**, and the channel should1 inherit and implement one or more SystemC interface classes. A channel implements all the pure virtual methods of the inherited interface classes.

Ports

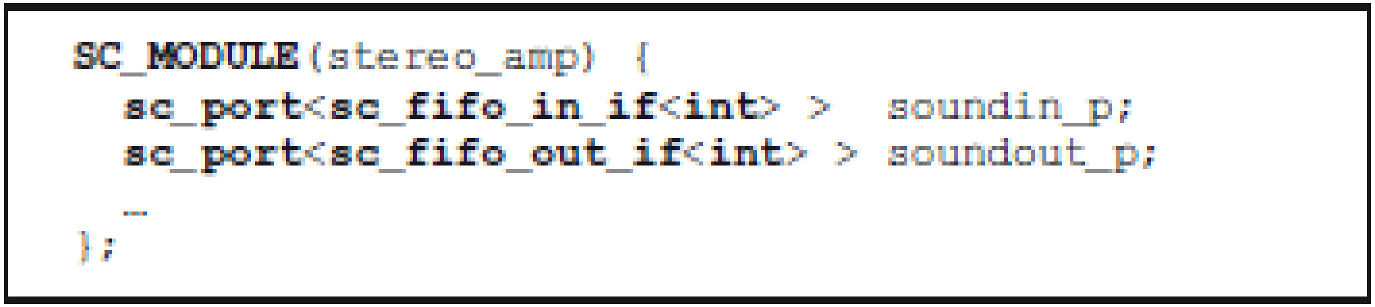

sc_fifo<T>
modA
modB
sc_fifo_out_if<T>
sc_fifo_in_if<T>
modA
Big_bus_chan
modB
modC

Ports

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

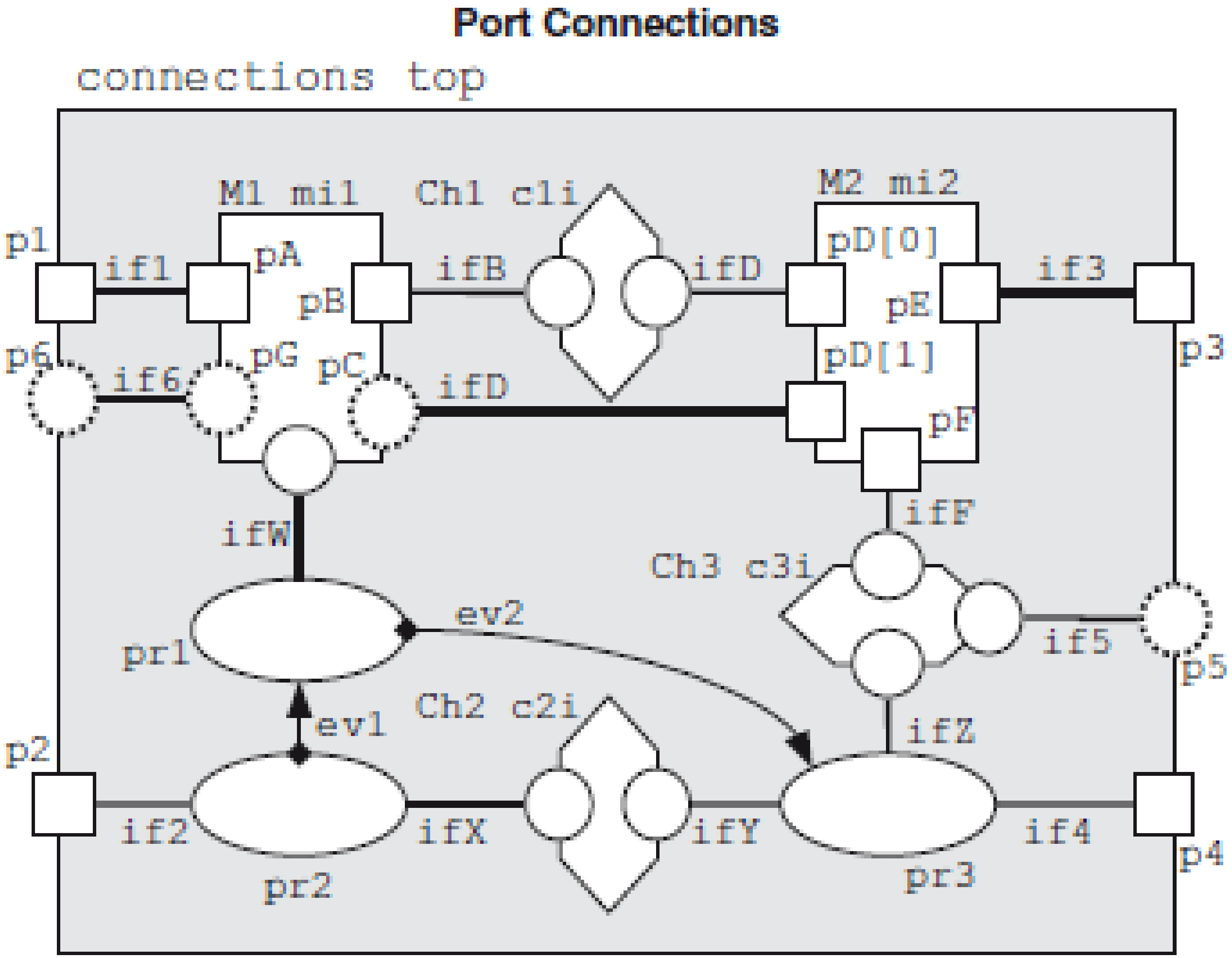


```
sc_port<interface> portname;
```



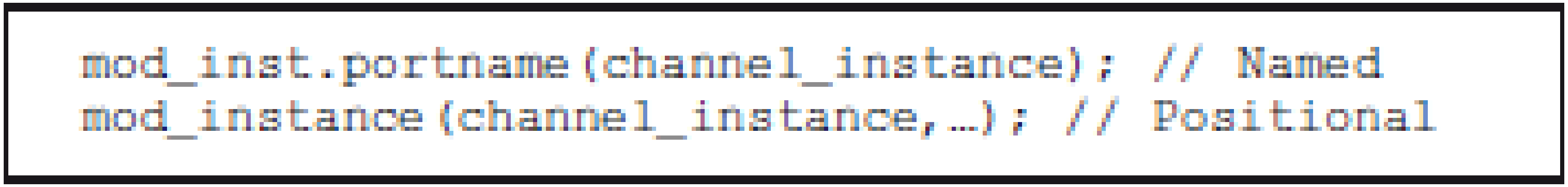
```
SC_MODULE(stereo_amp) {
  sc_port<sc_fifo_in_if<int> >  soundin_p;
  sc_port<sc_fifo_out_if<int> > soundout_p;
  …
};
```

Ports


Port Connections
connections top
M1 mi1
Ch1 c1i
M2 mi2
p1
if1
pA
pB
ifB
ifD
pD[0]
if3
pE
p3
p6
if6
pG
pC
ifD
pD[1]
pF
ifW
ifF
Ch3 c3i
ev2
pr1
if5
p5
ev1
Ch2 c2i
ifZ
p2
if2
ifX
ifY
if4
p4
pr2
pr3

Ports

- modules are connected to channels after both the modules and channels have been instantiated
- two syntaxes for connecting ports
  - by name
  - by position

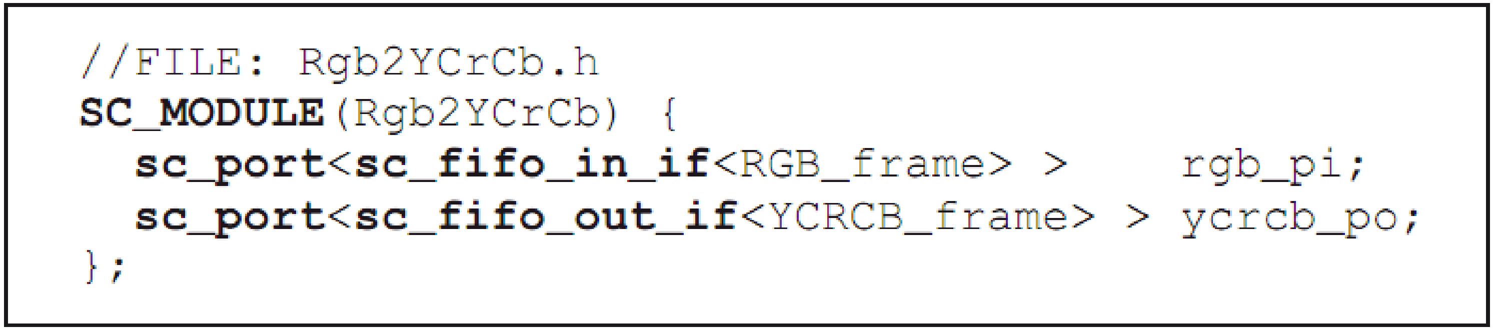


```
mod_inst.portname(channel_instance); // Named
mod_instance(channel_instance,…); // Positional
```

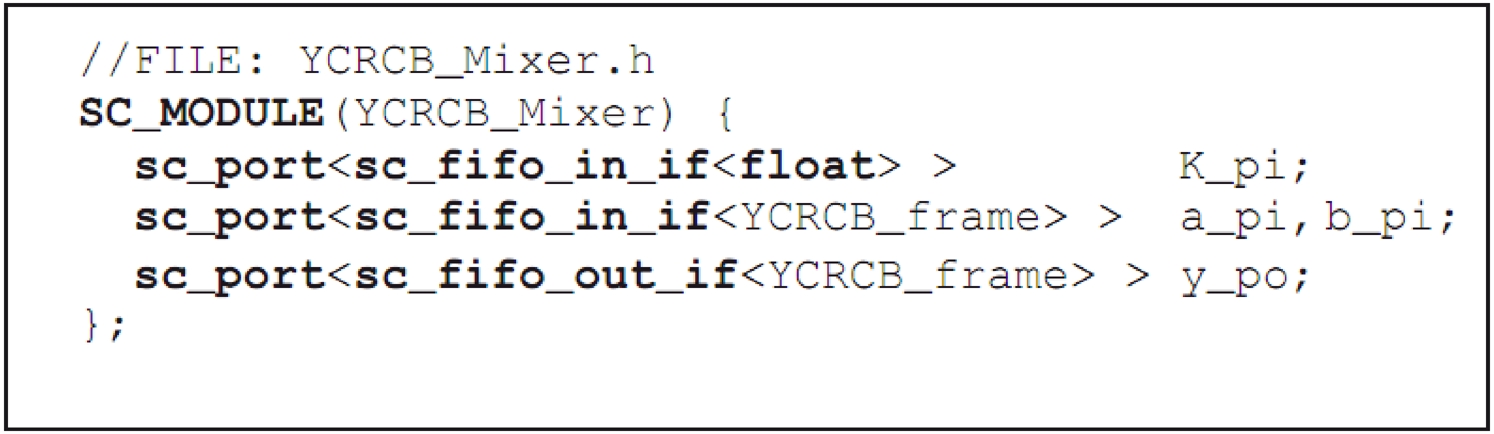
Ports

- When the code instantiating an **sc_port** executes:
  - the **operator()** is overloaded to take a channel object by reference
  - saves a pointer to that reference internally for later access by the port
- a port is an interface pointer to a channel that implements the interface

Ports

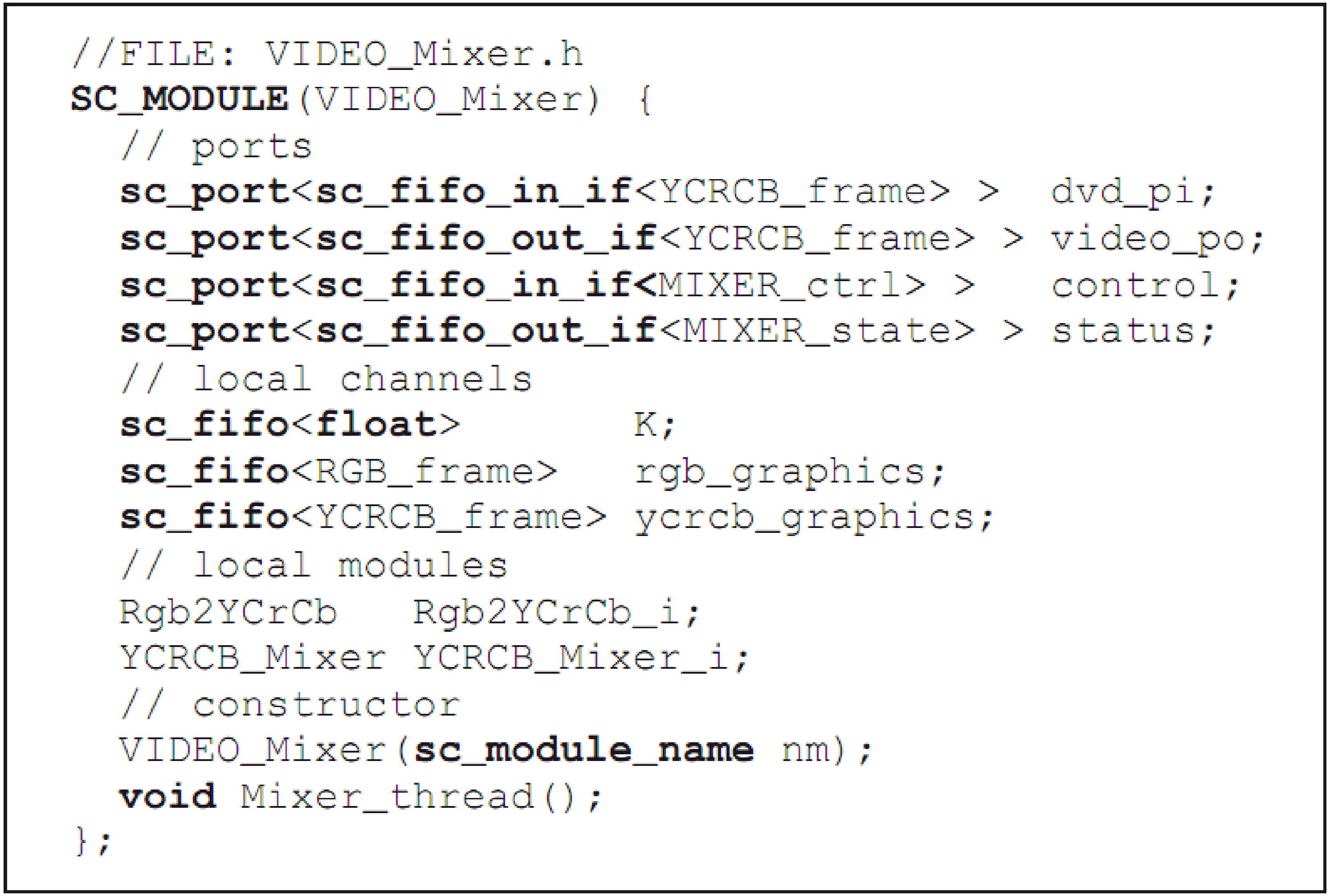


```
//FILE: Rgb2YCrCb.h
SC_MODULE(Rgb2YCrCb) {
  sc_port<sc_fifo_in_if<RGB_frame> >    rgb_pi;
  sc_port<sc_fifo_out_if<YCRCB_frame> > ycrcb_po;
};
```



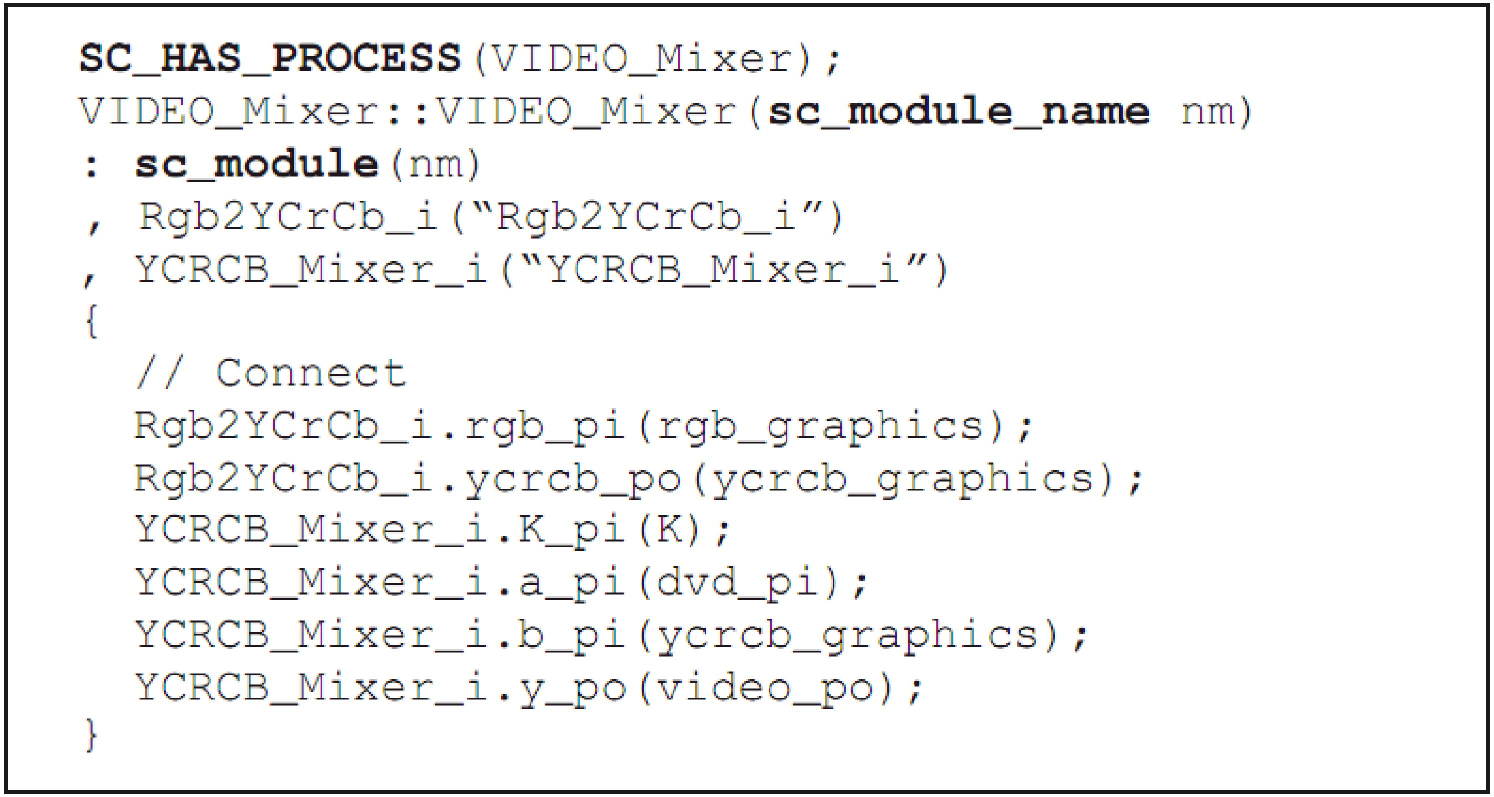
```
//FILE: YCRCB_Mixer.h
SC_MODULE(YCRCB_Mixer) {
  sc_port<sc_fifo_in_if<float> >        K_pi;
  sc_port<sc_fifo_in_if<YCRCB_frame> >  a_pi,b_pi;
  sc_port<sc_fifo_out_if<YCRCB_frame> > y_po;
};
```

Ports



```
//FILE: VIDEO_Mixer.h
SC_MODULE(VIDEO_Mixer) {
  // ports
  sc_port<sc_fifo_in_if<YCRCB_frame> >  dvd_pi;
  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;
  // local modules
  Rgb2YCrCb   Rgb2YCrCb_i;
  YCRCB_Mixer YCRCB_Mixer_i;
  // constructor
  VIDEO_Mixer(sc_module_name nm);
  void Mixer_thread();
};
```

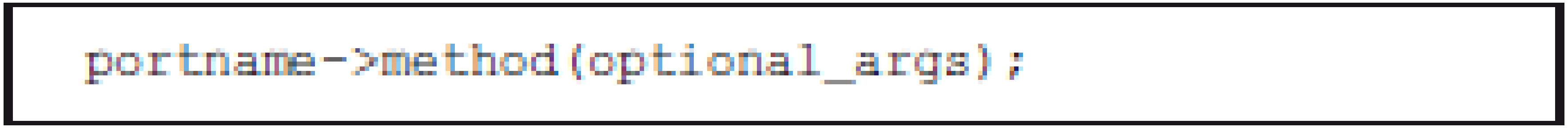
Ports



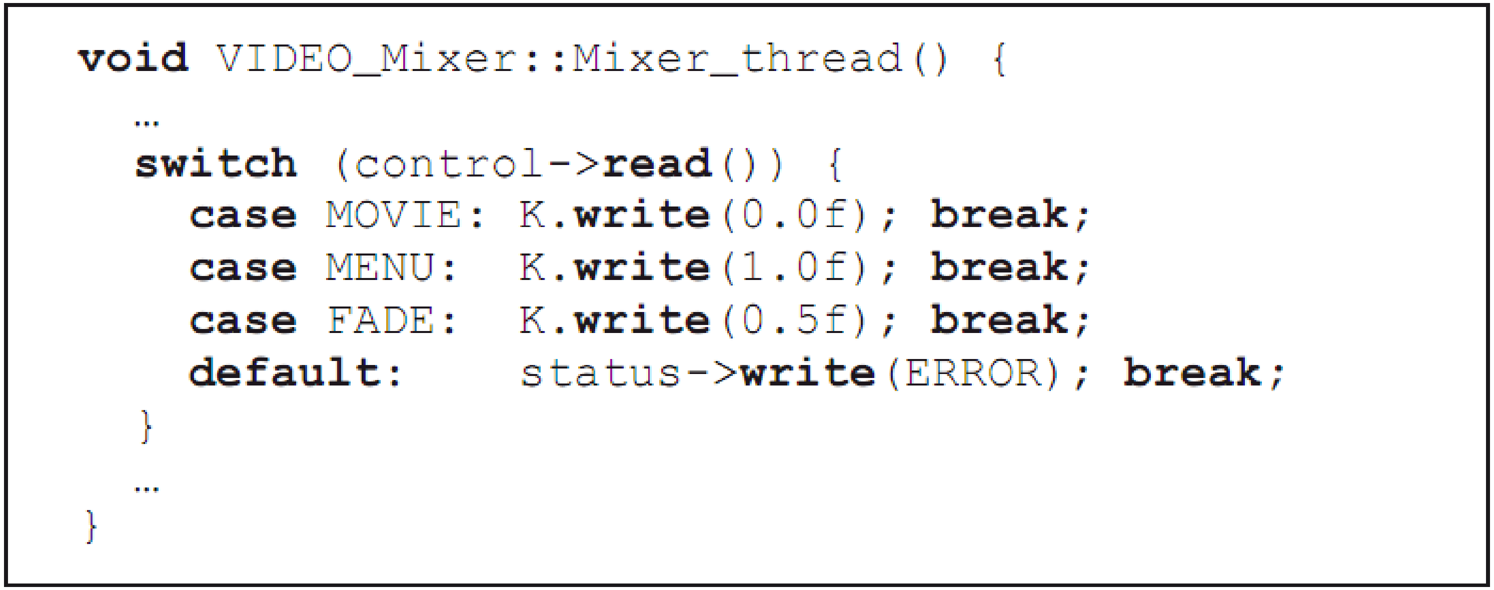
```
SC_HAS_PROCESS(VIDEO_Mixer);
VIDEO_Mixer::VIDEO_Mixer(sc_module_name nm)
: sc_module(nm)
, Rgb2YCrCb_i(“Rgb2YCrCb_i”)
, YCRCB_Mixer_i(“YCRCB_Mixer_i”)
{
  // Connect
  Rgb2YCrCb_i.rgb_pi(rgb_graphics);
  Rgb2YCrCb_i.ycrcb_po(ycrcb_graphics);
  YCRCB_Mixer_i.K_pi(K);
  YCRCB_Mixer_i.a_pi(dvd_pi);
  YCRCB_Mixer_i.b_pi(ycrcb_graphics);
  YCRCB_Mixer_i.y_po(video_po);
}
```

Ports

- the sc_port overloads the C++ **operator->()**, which allows a simple syntax

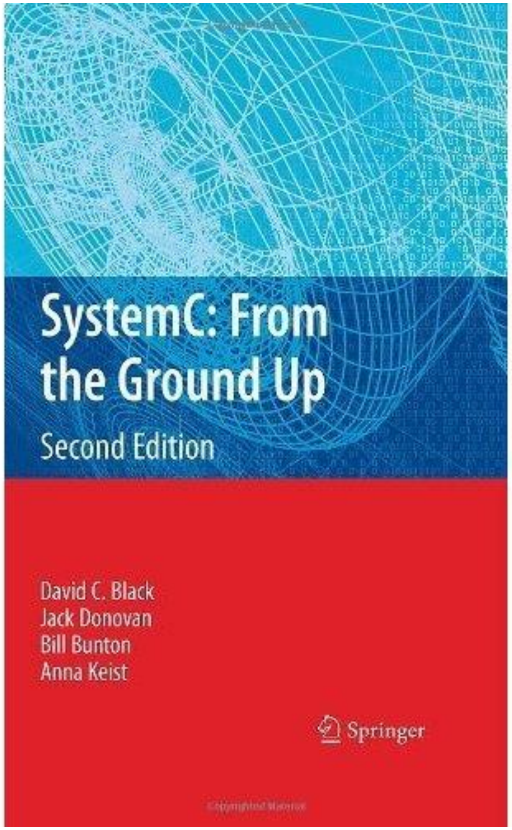


```
portname->method(optional_args);
```



```
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;
  }
  …
}
```

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.”