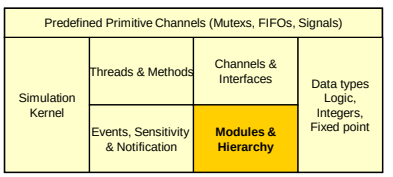
Plan

- 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



Modules

- Main Function
- Module
- Basic Styles

A starting Point : sc_main

- All C/C++ programs need starting point
  - C/C++ : main()
  - SystemC : sc_main()

main.cpp — *starting_point_cpp*

```
int main(int argc, char* argv[])
{
        if (argc == 1)
                cout << "No Argument ! " << endl;
        if (argc >= 1)
                cout << “First Argument : “ << argv[1] << endl;
        if (argc >= 2)
                cout << “Second Argument : “ << argv[2] << endl;
        // Body of Program
        return 0;  // EXIT CODE (0 = success)
}
```

```
> test.exe blue yellow

First Argument : blue
Second Argument : yellow
```

argc = 3 — argv[0] = “test.exe” — argv[1] = “blue” — argv[2] = “yellow” — argv[3] = 0

sc_main() → *main.cpp* — *starting_point_systemc*

```
int sc_main(int argc, char* argv[])
{
    if (argc == 1)                                   // elaboration
        cout << "No Argument ! " << endl;

    // ELABORATION Phase
    if (argc >= 1)
        cout << “First Argument : “ << argv[1] << endl;
    if (argc >= 2)
        cout << “Second Argument : “ << argv[2] << endl;

    sc_start();  // SIMULATION begins and ends      // simulation
                 // in this function

    return 0; // EXIT CODE (0 = success)
}
```


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

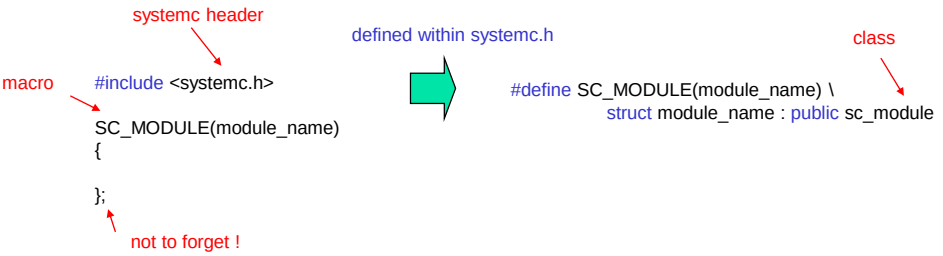
Modules

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

- Main Function
- Module
- Basic Styles

Declaration

- A systemC module is the smallest container of functionality with
  - states
  - behaviors
  - structures for hierarchical connectivity

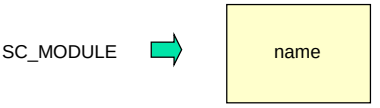


Body of the MODULE

- Included in the body
  - Ports
  - Member channel instances
  - Member data instance
  - Member module instances (sub-designs)
  - Constructor
  - Destructor
  - Process member functions (processes)
  - Helper functions

GUIDELINE: Respects the order of declaration

Graphical notation



Port
Declaration and Functions

- A module has ports to communicate with modules

	IN port	IN/OUT port	OUT port
Graphical notations	i	io	o
	sc_in<*type_name*> i;	sc_inout<*type_name*> io;	sc_out<*type_name*> o;

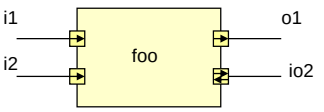
	const T& read() const;
	operator const T& () const;
		const T& read() const;
		operator const T& () const;
		void write(const T&);
		sc_inout<T>& operator= (const T&);
			void write(const T&);
			sc_out<T>& operator= (const T&);

classes
- sc_in<bool> and sc_in<sc_logic>
- sc_inout<bool> and sc_inout<sc_logic>

additional functions

```
const sc_event& posedge_event() const;
const sc_event& negedge_event() const;
bool event() const;
bool posedge() const;
sc_event_finder& pos() const;
sc_event_finder& neg() const;
```


Port
Example




```
SC_MODULE (foo)
{
    sc_in<sc_bit> i1, i2;
    sc_out<bool> o1;
    sc_inout<bool> io2;
    …
}
```

VHDL

```
entity foo is
    port( i1, i2 : in bit;
          o1 : out bit;
          io2 : inout bit);
end entity;
```

Constructor

- the SC_MODULE constructor performs several tasks
  - initializing/allocating sub-design
  - connecting sub-design
  - registering processes with the SystemC kernel
  - providing static sensitivity
  - miscellaneous user-defined setup

```
SC_MODULE (module_name)
{
    sc_in<bool> a …
    …
    SC_CTOR(module_name)
    {
        // subdesign allocation
        // subdesign connectivity
        // process registration
        // miscellaneous setup
    }
};
```

equal

```
SC_MODULE (module_name)
{
    sc_in<bool> a …
    …
    module_name(sc_module_name inst_name, arg1, arg2 … )
        : sc_module(inst_name)
    {
        // subdesign allocation
        // subdesign connectivity
        // process registration
        // miscellaneous setup
    }
};
```

new argument(s)

macro

```
#define SC_CTOR(user_module_name) \
        typedef user_module_name SC_CURRENT_USER_MODULE; \
        user_module_name( sc_module_name )
```


Destructor

- included in C++ language
- A constructor initializes an object
  - file
  - memory (new operator)
  - and so on …
- A destructor
  - release the memory (delete operator)
  - …

```
SC_MODULE (foo)
{
    sc_in<sc_bit> i1, i2;
    sc_out<bool> o1;

    int* table;

    foo(sc_module_name inst_name, int sz)
        : sc_module(inst_name)
    {
        table = new int[sz];
    }

    ~foo()
    {
        delete[] table;
    }
};
```

Member data instance

constructor

dynamic allocation of memory

destructor

dynamic deallocation of memory

Process Member Functions (1/2)

- A SystemC process
  - is a member function or class method of an SC_MODULE
  - is invoked by the scheduler (SystemC simulation kernel)

  void process_name(void) OR void process_name()

- Registering a process
  - must be inside the constructor

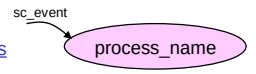
SC_THREAD(process_name)

- similar to VHDL process / Verilog initial block
- using wait method

SC_METHOD(method_name)

- triggers the method and performs it all
- No wait possible !

Graphical notations



sc_method or sc_thread

Process Member Functions (2/2) Declaration : 2 solutions

Constructor with MACRO SC_CTOR

```
SC_MODULE (module_name)
{
    sc_in<bool> a …
    …

    SC_CTOR(module_name)
    {
        SC_THREAD(p1_thread);
        SC_THREAD(p2_thread);
        // …
    }

    void p1_thread();
    void p2_thread ();
};
```

constructor (Macro)

registering

process declaration

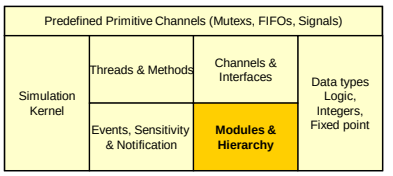
user-defined (arg.) Constructor

```
SC_MODULE (module_name)
{
    sc_in<bool> a …
    …

    SC_HAS_PROCESS(module_name);

    module_name(sc_module_name n)
        : sc_module(n)
    {
        SC_THREAD(p1_thread);
        SC_THREAD(p2_thread);
        // …
    }

    void p1_thread ();
    void p2_thread ();
};
```

Macro

constructor

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



Modules

- Main Function
- Module
- Basic Styles

Two Basic Styles
First Solution : Traditional Template

NAME.h

```
#ifndef NAME_H
#define NAME_H

#include "submodule.h"
…

SC_MODULE(NAME)
{
      // Port declaration
      // Channel/sub module instances

      SC_CTOR(NAME) : init var …
      {
            // Connectivity
            // Registration
      }
      // Process declarations
      void p1_method();
      void p2_thread();

      // Helper Declarations
      int getValue();
};
#endif
```

+

NAME.cpp

```
#include <systemc.h>
#include "NAME.h"

// Process implementations

void NAME::p1_method()
{
      …
}

void NAME::p2_thread()
{
      …
}

// Helper implementations

int NAME::getValue()
{
      …
}
```

Two Basic Styles
Second Solution : Recommended

NAME.h

```
#ifndef NAME_H
#define NAME_H

#include "submodule.h"
…

SC_MODULE(NAME)
{
      // Port declaration
      // Channel/submodule* definitions

      // Constructor declarations
      SC_CTOR(NAME);

      // Process declarations
      void p1_method();
      void p2_thread();

      // Helper Declarations
      int getValue();
};
#endif
```

pointer

+

NAME.cpp

```
#include <systemc.h>
#include "NAME.h"

// Constructor
NAME::NAME(sc_module_name nm)
      : sc_module(nm)
      , init var …
{     …
}

// Process implementations
void NAME::p1_method()
{     …
}

void NAME::p2_thread()
{     …
}

// Helper implementations
int NAME::getValue()
{     …
}
```

GUIDELINE: this approach is more conductive to independent development of modules

Example
Simple Process

simple_process.h

```
#ifndef SIMPLE_PROCESS_H
#define SIMPLE_PROCESS_H

#include <systemc.h>

SC_MODULE(simple_process)
{

      SC_CTOR(simple_process)
      {
            SC_THREAD(my_thread_process);
      }

      void my_thread_process(void);
};
#endif
```

simple_process.cpp

```
#include "simple_process.h"

void simple_process::my_thread_process(void)
{
      std::cout << "my_thread_process executed within "
                << name()
                << std::endl;
}
```

simple_process

main.cpp

```
#include "simple_process.h"

int sc_main(int argc, char* argv[])
{
      simple_process my_instance1("my_inst1");
      simple_process my_instance2("my_inst2");

       sc_start();

      return 0;
}
```

```
SystemC 2.1_oct_12_04.beta --- Apr 10 2005 17:53:24
Copyright (c) 1996-2004 by all Contributors
ALL RIGHTS RESERVED
my_thread_process executed within my_inst1
my_thread_process executed within my_inst2
Press any key to continue
```