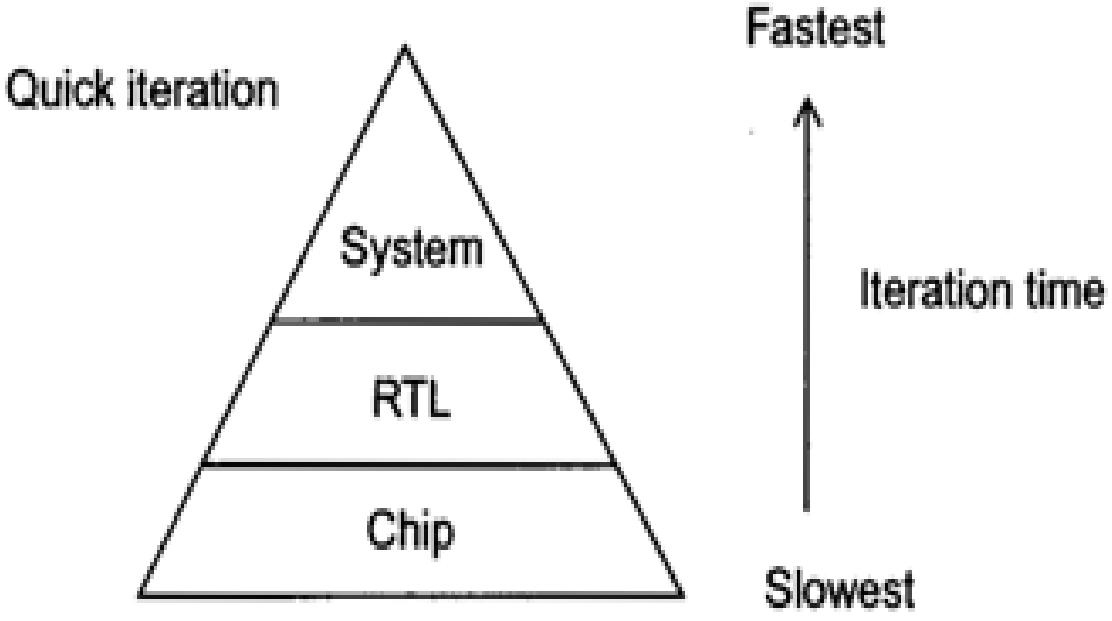
System C-Part-1

BITS Pilani
Hyderabad Campus

Software for Embedded Systems

Motivation

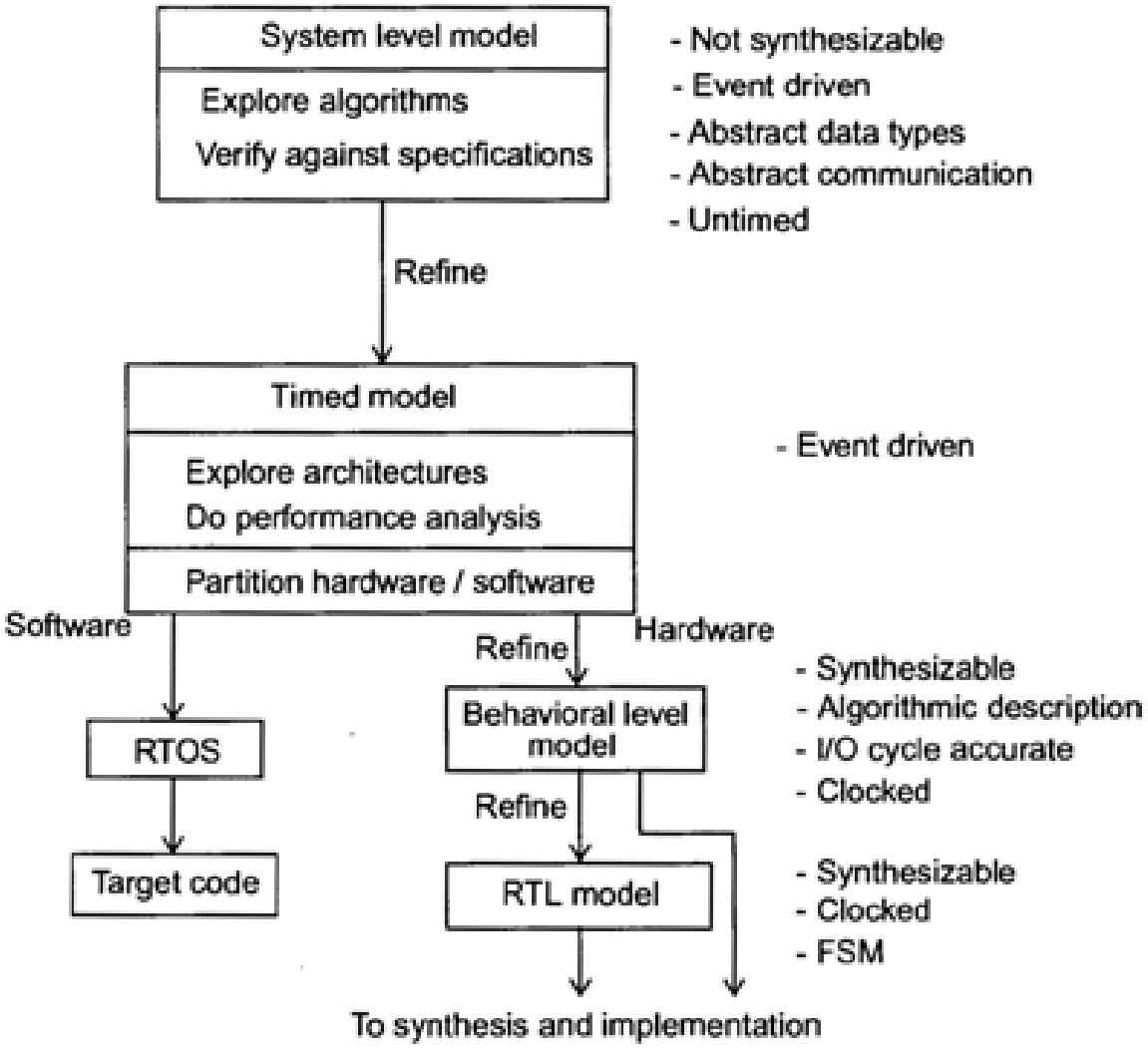
Exploring different architectures at different levels



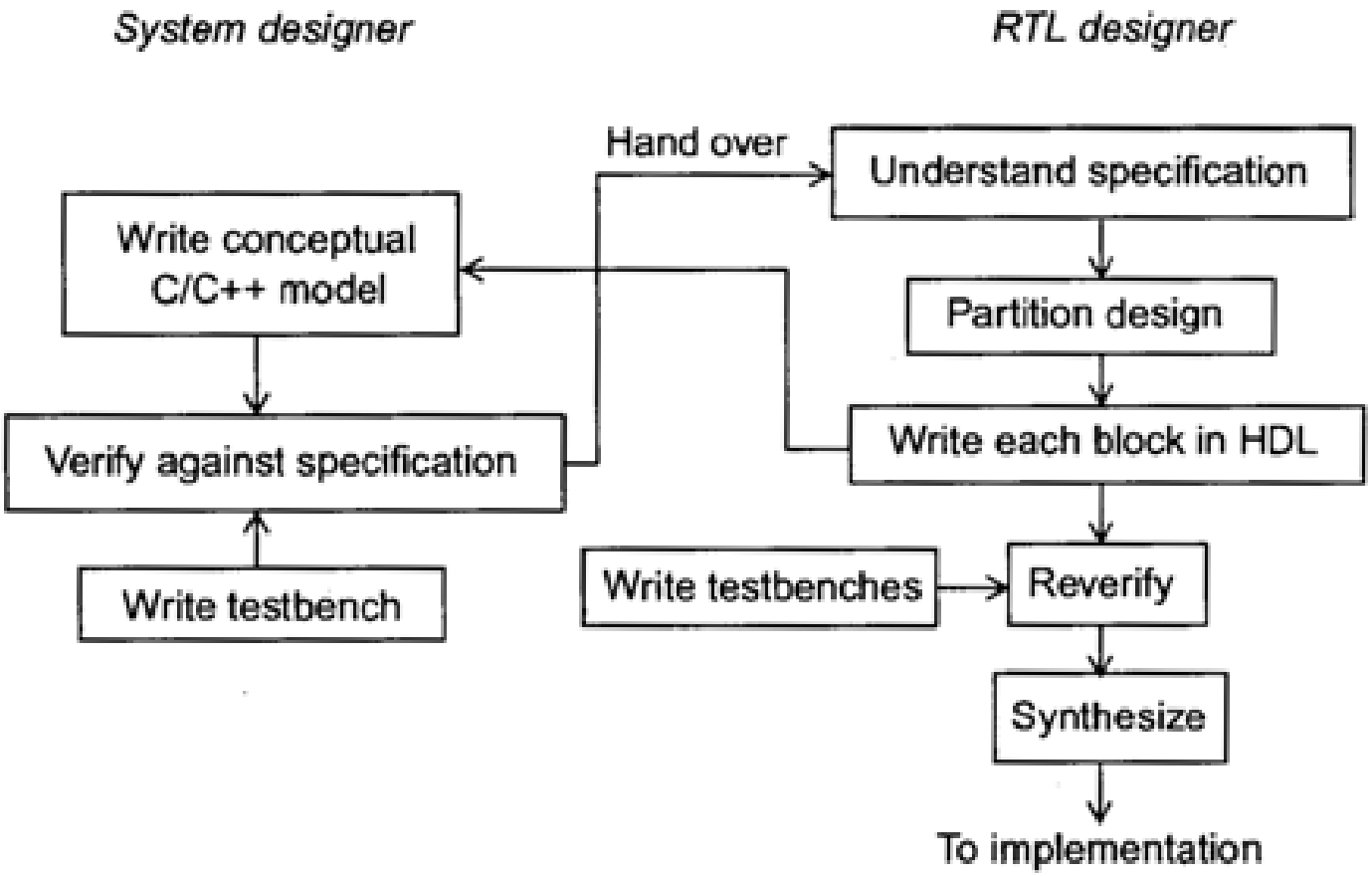
Primary goal

- Enable System-Level Modeling
  - Systems include hardware, software, or both
  - Challenges:
    - Wide range of design models of computation
    - Wide range of design abstraction levels
    - Wide range of design methodologies

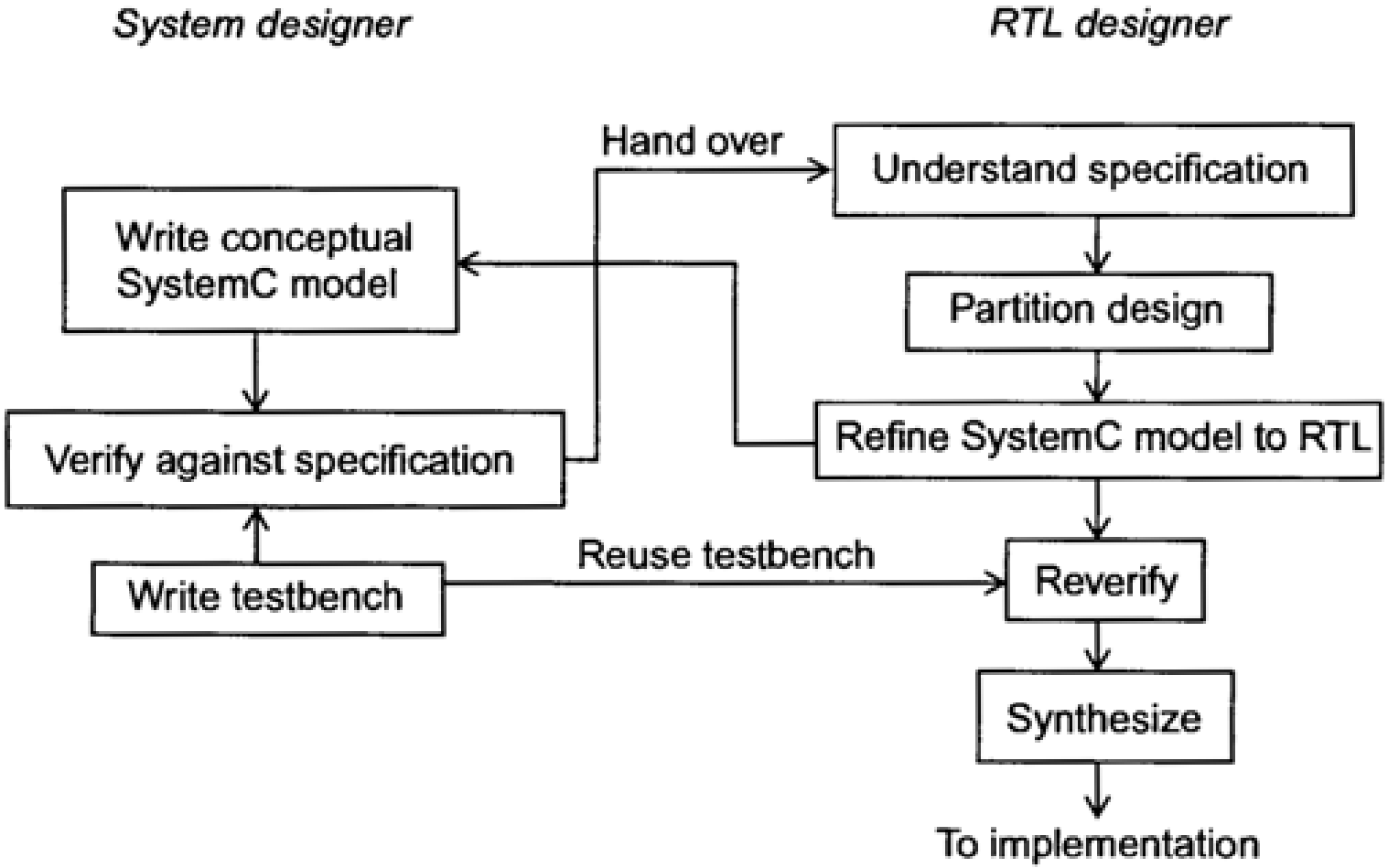
System level design process



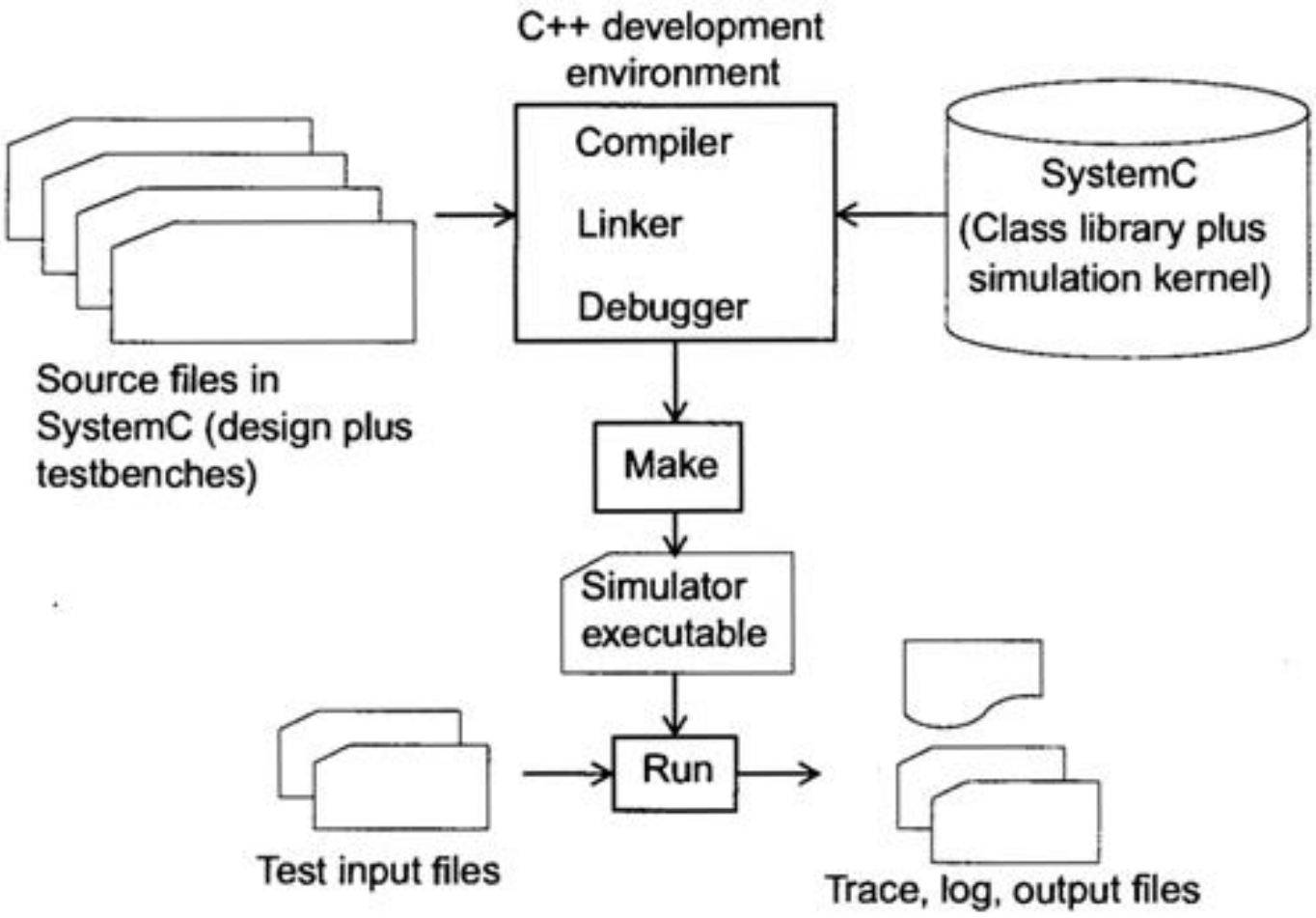
Non System C methodology



System C methodology



System C in C++ deve environment



Solution in System C 2.0

- SystemC is both system level and hardware description language.
- Allows you to model at algorithmic level.
- Introduces a small but very general purpose modeling foundation
- Elementary channels
  - Other library models provided (FIFO, Timers, ...)
- Support for various models of computation, methodologies, etc.
- Built on top of the core language, hence are separate from it

What does sysetm C provide?

SystemC 1.0 provided RTL and behavioral HDL modeling capabilities.

HW is modeled using zero-delay semantics for combinational logic. Signals are modeled using 01XZ, “C” data types and complex data types can also be used within signals.

SystemC 1.0 includes good support for fixed point modeling.

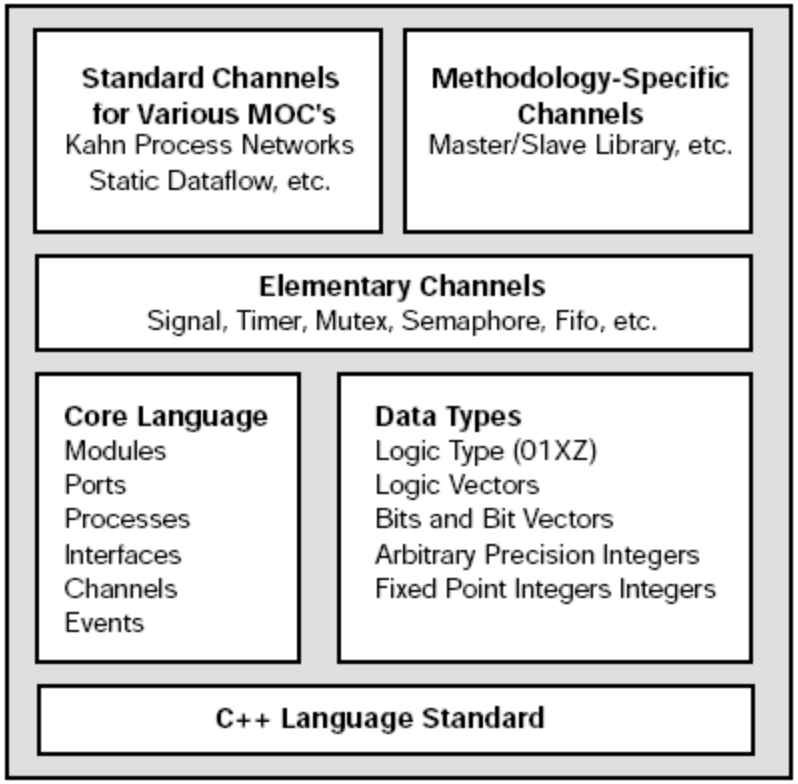
SystemC 1.1 beta and 1.2 beta provided some limited communication refinement capabilities.

SystemC 2.0 has more general system level modeling capabilities with *channels, interfaces, and events.*

SystemC 3.0 will focus on software and scheduler modeling (more later).

SystemC 2.0 Language Architecture

- All built on C++
- Upper layers built on lower ones
- Core language
  - Structure
  - Concurrency
  - Communication
  - Synchronization
- Data types separate from the core language
- Commonly used communication mechanisms and MOC built on top of core language
- Lower layers can be used without upper ones



Modules

Basic entity to represent certain functionality

Modules are interconnected through ports

Modules can contain other modules and processes

Described using SC_MODULE

Process

Describes certain functionality

Processes are inside modues

Can be abstracted as:

- SC_METHOD
- SC_THREAD

Processes are not hierarchical

Port

A module has ports through which it communicates with other modules.

- Input
- Output
- Inout ports

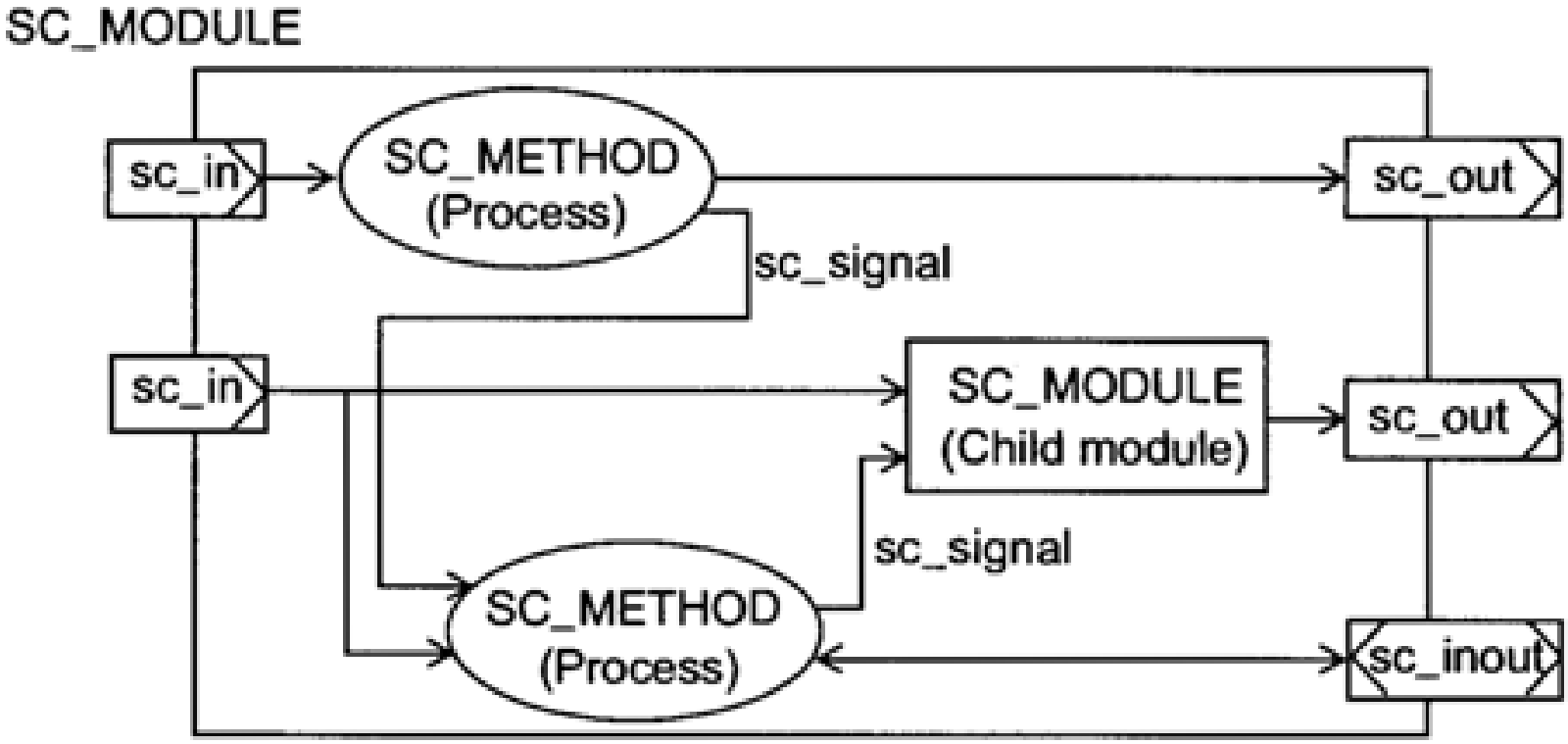
User can describe new kind of port with user defined interface

Signals

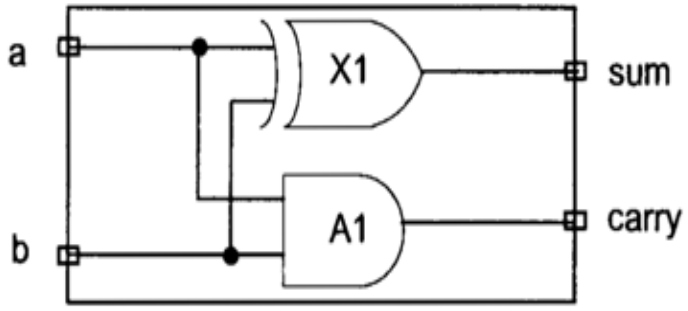
A signal can carry a value

Used to connect multiple processes and module instances.

SC_MODULE


SC_MODULE
sc_in
SC_METHOD
(Process)
sc_out
sc_signal
sc_in
SC_MODULE
(Child module)
sc_out
sc_signal
SC_METHOD
(Process)
sc_inout

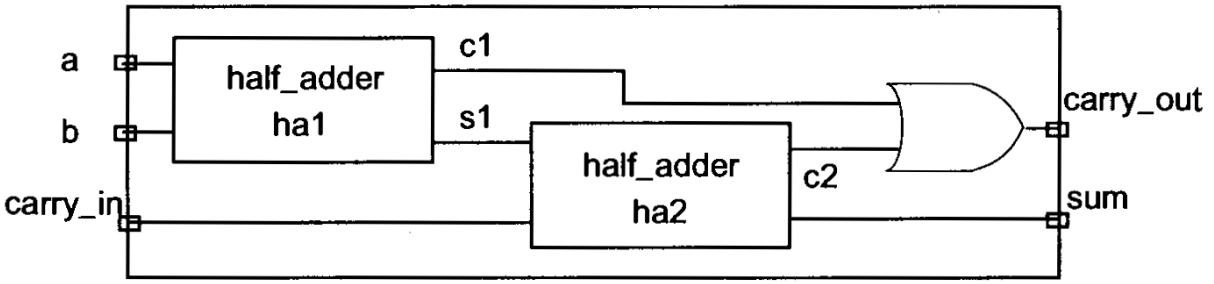
A half adder



- An SC_METHOD is sensitive to a specific set of signals and ports
- Cannot suspend due to wait statement

```
// File: half_adder.h
#include "systemc.h"

SC_MODULE (half_adder) {
  sc_in<bool> a, b;
  sc_out<bool> sum, carry;

  void prc_half_adder ();

  SC_CTOR (half_adder) {
    SC_METHOD (prc_half_adder);
    sensitive << a << b;
  }
};

// File: half_adder.cpp
#include "half_adder.h"

void half_adder::prc_half_adder () {
  sum = a ^ b;
  carry = a & b;
}
```

Full adder




```
// File: full_adder.h
#include "half_adder.h"

SC_MODULE (full_adder) {
  sc_in<bool> a, b, carry_in;
  sc_out<bool> sum, carry_out;

  sc_signal<bool> c1, s1, c2;
  void prc_or ();
  half_adder *ha1_ptr, *ha2_ptr;

  SC_CTOR (full_adder) {
    ha1_ptr = new half_adder ("ha1");
    // Named association:
    ha1_ptr->a (a);
    ha1_ptr->b (b);
    ha1_ptr->sum (s1);
    ha1_ptr->carry (c1);

    ha2_ptr = new half_adder ("ha2");
    // Positional association:
    (*ha2_ptr) (s1, carry_in, sum, c2);

    SC_METHOD (prc_or);
    sensitive << c1 << c2;
  }

  // A destructor:
  ~ full_adder() {
    delete ha1_ptr;
    delete ha2_ptr;
  }
};

// File: full_adder.cpp
#include "full_adder.h"

void full_adder::prc_or () {
  carry_out = c1 | c2;
}
```

Driver module


```
// File: driver.h
#include "systemc.h"

SC_MODULE (driver) {
  sc_out<bool> d_a, d_b, d_cin;

  void prc_driver ();

  SC_CTOR (driver) {
    SC_THREAD (prc_driver);
  }
};
```

```
// File: driver.cpp
#include "driver.h"

void driver::prc_driver () {
  sc_uint<3> pattern;
  pattern = 0;

  while (1) {
    d_a = pattern[0];
    d_b = pattern[1];
    d_cin = pattern[2];
    wait (5, SC_NS);
    pattern++;
  }
}
```

Monitor module

```
// File: monitor.h
#include "systemc.h"

SC_MODULE (monitor) {
  sc_in<bool> m_a, m_b, m_cin, m_sum, m_cout;

  void prc_monitor ();

  SC_CTOR (monitor) {
    SC_METHOD (prc_monitor);
    sensitive << m_a << m_b << m_cin << m_sum << m_cout;
  }
};
```

```
// File: monitor.cpp
#include "monitor.h"

void monitor::prc_monitor () {
  cout << "At time " << sc_time_stamp() << "::";
  cout << "(a, b, carry_in): ";
  cout << m_a << m_b << m_cin;
  cout << "  (sum, carry_out): " << m_sum
       << m_cout << endl;
}
```

The simulation


```
// File: full_adder_main.cpp
#include "driver.h"
#include "monitor.h"
#include "full_adder.h"

int sc_main(int argc, char* argv[]) {
  sc_signal<bool> t_a, t_b, t_cin, t_sum, t_cout;

  full_adder f1 ("FullAdderWithHalfAdder");
  // Connect using positional association:
  f1 << t_a << t_b << t_cin << t_sum << t_cout;

  driver d1 ("GenerateWaveforms");
  // Connect using named association:
  d1.d_a(t_a);
  d1.d_b(t_b);
  d1.d_cin(t_cin);

  monitor mo1 ("MonitorWaveforms");
  mo1 << t_a << t_b << t_cin << t_sum << t_cout;

  sc_start(100, SC_NS);

  return(0);
}
```

Thanks for your attention