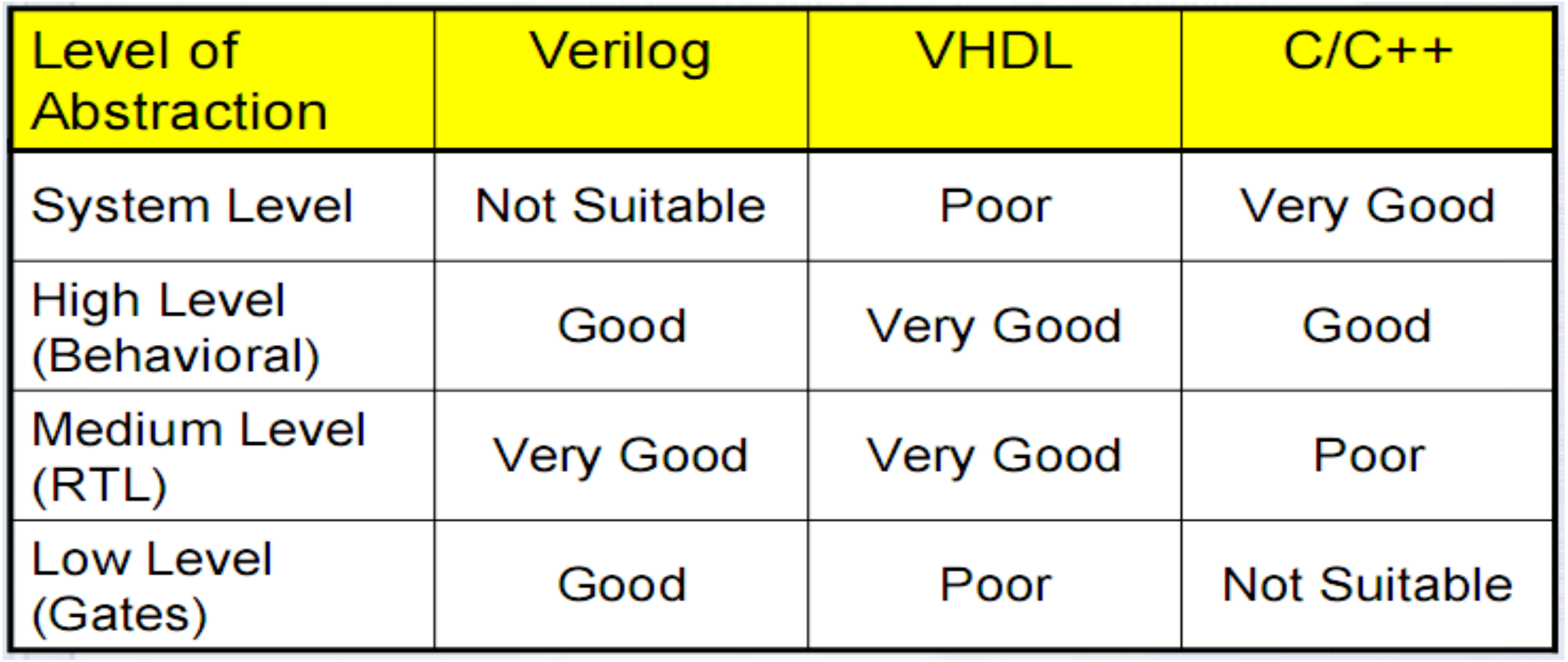
SystemC Language

Outline

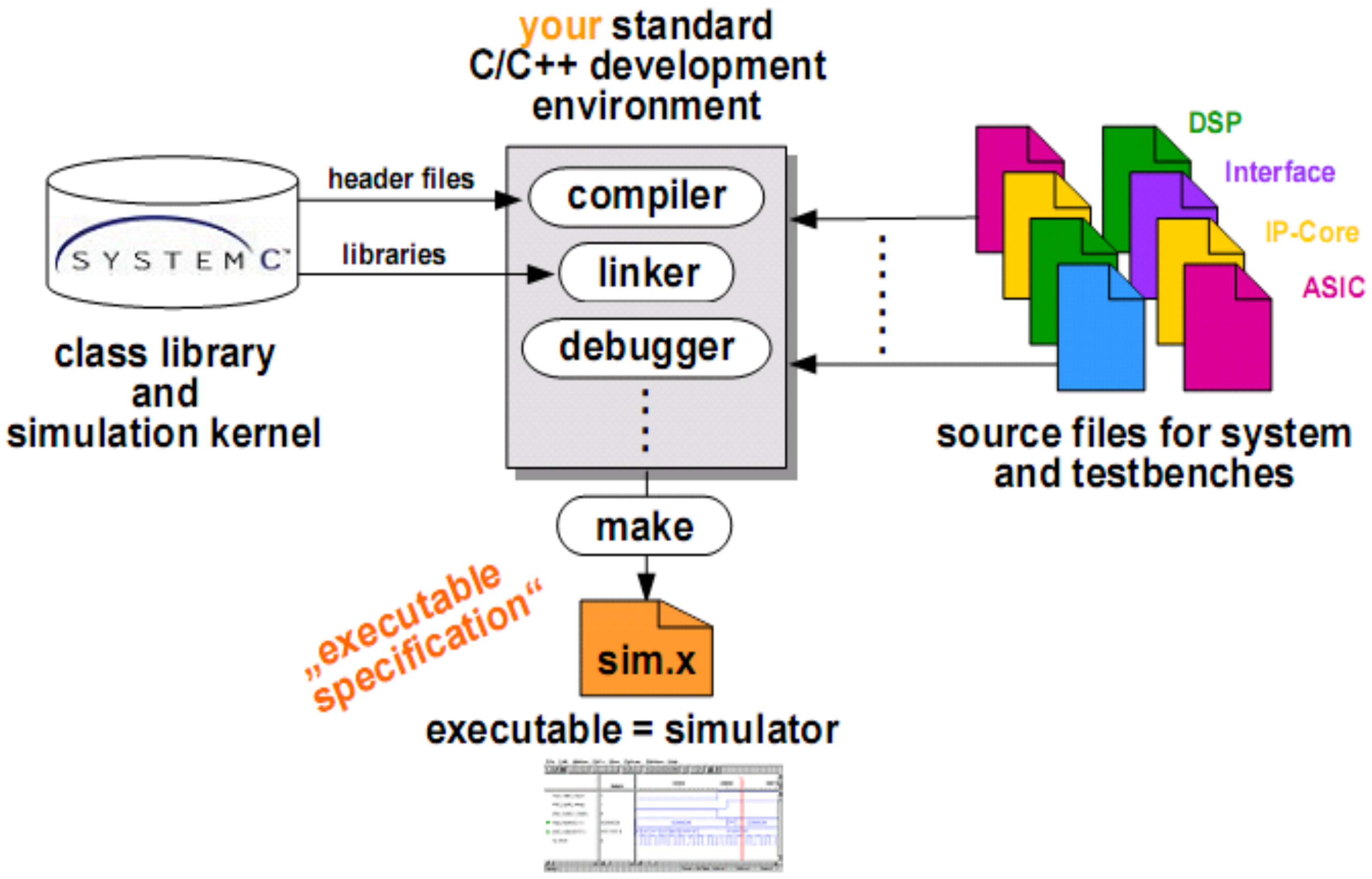
- Introduction
- SystemC Language
- Example
- References

Comparation



Level of Abstraction	Verilog	VHDL	C/C++
System Level	Not Suitable	Poor	Very Good
High Level (Behavioral)	Good	Very Good	Good
Medium Level (RTL)	Very Good	Very Good	Poor
Low Level (Gates)	Good	Poor	Not Suitable

Introduce SystemC Language


your standard C/C++ development environment
header files
libraries
SYSTEMC
compiler
linker
debugger
DSP
Interface
IP-Core
ASIC
class library and simulation kernel
source files for system and testbenches
make
„executable specification“
sim.x
executable = simulator

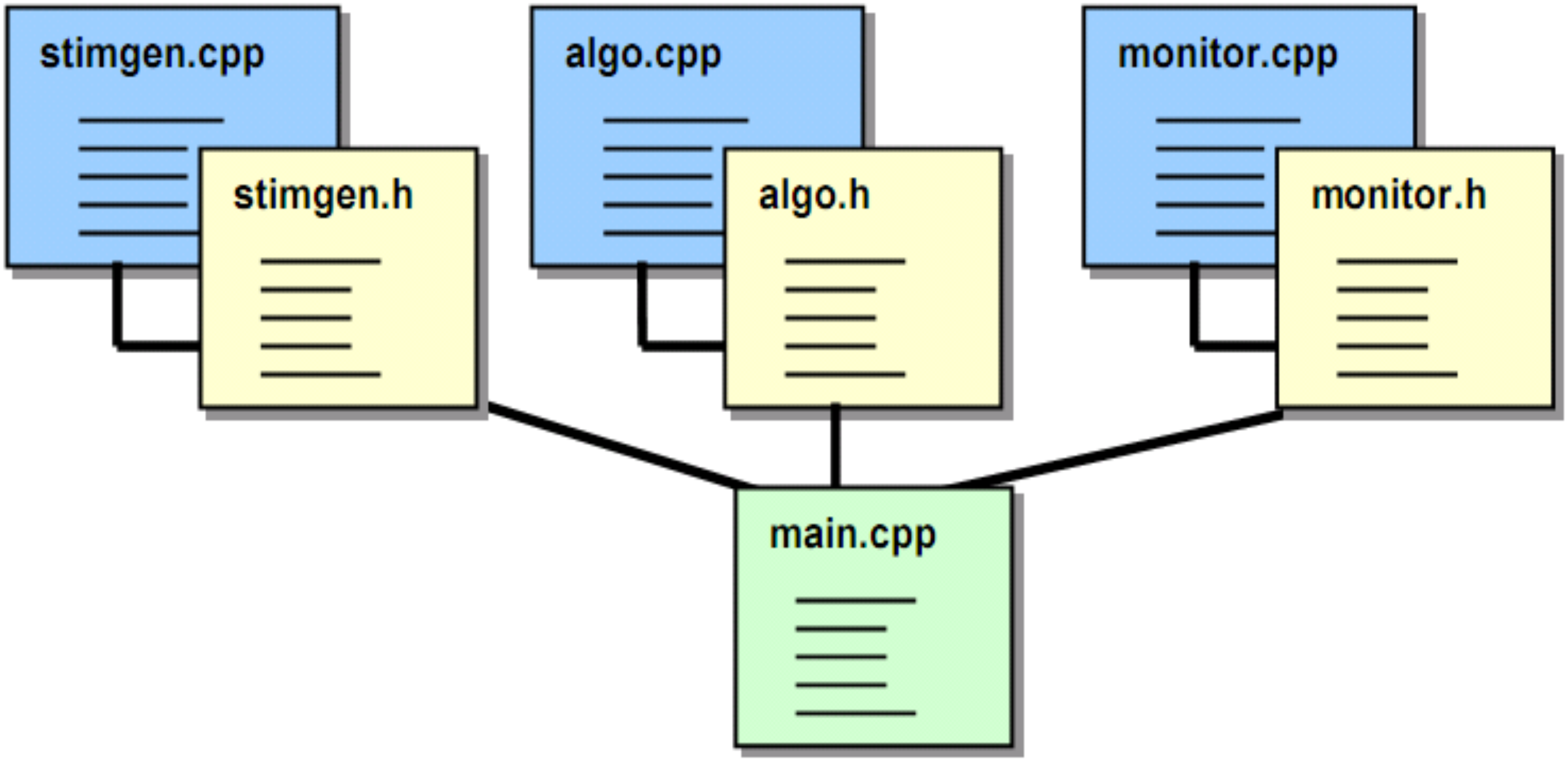
Outline

- Introduction
- SystemC Language
- Example
- References

SystemC Language

- Design Example
- Data type
- Port & Signal
- Module Declaration
- Top Level & Testbench

Design Example



Design Example

file.h

```
#include "systemc.h"
SC_MODULE(updown)
{
  sc_in<bool>  x_din;
  sc_in<bool>  x_clk;
  sc_out<bool> x_dout;
  void doit() ;

  SC_CTOR(updown)
  {
    SC_METHOD(doit);
    sensitive_pos << x_clk;
  }
};
```

file.cpp

```
#include "updown.h"


void dff::doit()
  {
    x_dout = x_din;
  };
```

Design Example

main

```
#include "systemc.h"
#include "updown.h"

int sc_main(int argc, char* argv[]) {

  sc_signal<bool>   din;
  sc_signal<bool>     dout;

 // sc_clock clk("clock", 20);
  sc_clock  clk("my clock" , 20 , 0.5);

 din  = 1;
updown U1 ("updown");
  U1.x_clk(clk);
  U1.x_dout(dout);
  U1.x_din(din);

// trace file

  sc_trace_file *tf =
     sc_create_vcd_trace_file
     ("updown_wave");
  // External Signals
  sc_trace(tf, clk, "clock");
  sc_trace(tf, din, "in");
  sc_trace(tf, dout, "out");

  sc_start(200);

  sc_close_vcd_trace_file(tf);

  sc_stop();

  return (0);
```

SystemC Language

- Design Example
- Data type
- Port & Signal
- Module Declaration
- Top Level & Testbench

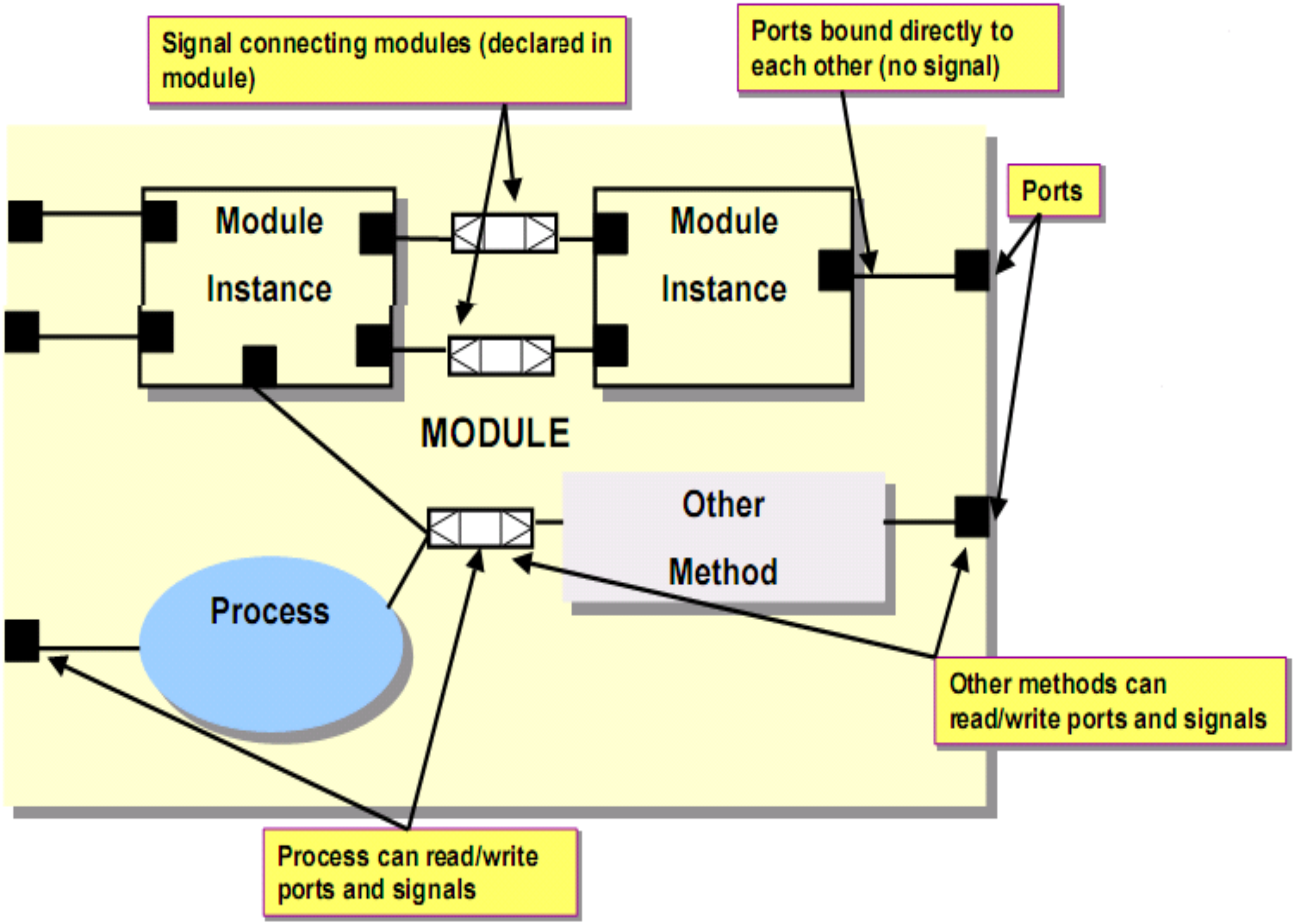
DataType

➢ SC_BIT Data	(sc_bit name)
➢ SC_LOGIC	(sc_logic name)
➢ SC_INT & SC_UINT	(sc_int<5>name)
➢ SC_BIGINT & SC_BIGUINT	(sc_biguint<6>name)
➢ BIT _VECTOR	(sc_bv<7>name)
➢ LENG LOGIC VECTOR	(sc_lv<8>name)

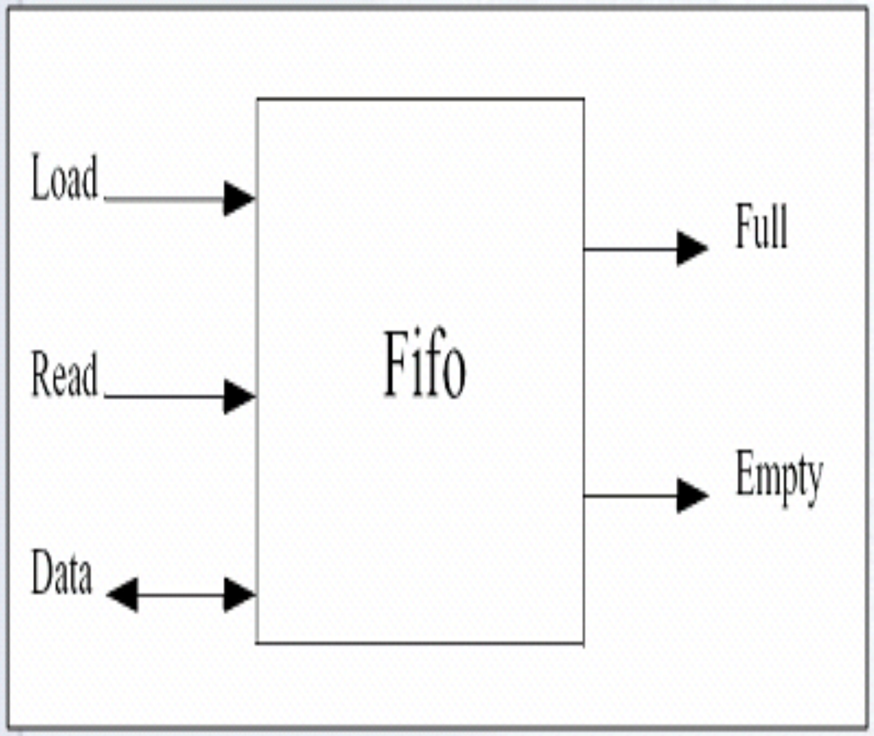
SystemC Language

- Design Example
- Data type
- Port & Signal
- Module Declaration
- Top Level & Testbench

Port & Signal


Signal connecting modules (declared in module)
Ports bound directly to each other (no signal)
Ports
Module Instance
Module Instance
MODULE
Other Method
Process
Other methods can read/write ports and signals
Process can read/write ports and signals

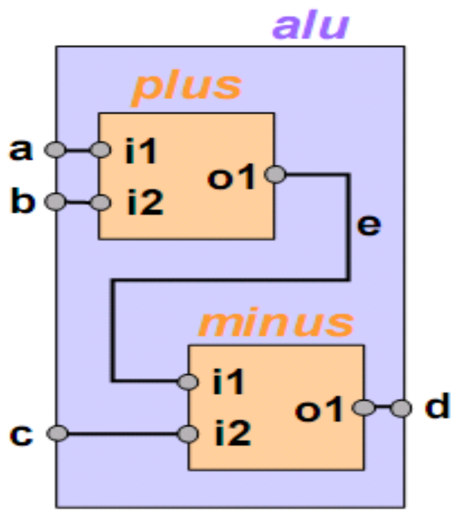
Port & Signal



```
SC_MODULE(fifo) {
    sc_in<bool> load;
    sc_in<bool> read;
    sc_inout<int> data;
    sc_out<bool> full;
    sc_out<bool> empty;
//rest of module not shown
}
```

Port & Signal

Example:



d = (a + b) - c

```
SC_MODULE( plus ) {
    sc_in<int>  i1;
    sc_in<int>  i2;
    sc_out<int> o1;
    ......

};

SC_MODULE( minus ) {
    sc_in<int>  i1;
    sc_in<int>  i2;
    sc_out<int> o1;
    ......
};
```

```
SC_MODULE( alu ) {
   sc_in<int>  a;
   sc_in<int>  b;
   sc_in<int>  c;
   sc_out<int> d;

   plus  *p;
   minus *m;

   sc_signal<int> e;

   SC_CTOR( alu ) {
      p = new plus ( "PLUS“ );
      p->i1 (a);
      p->i2 (b);
      p->o1 (e);

      m = new minus ( "MINUS“ );
      (*m) (e,c,d);
   }
};
```

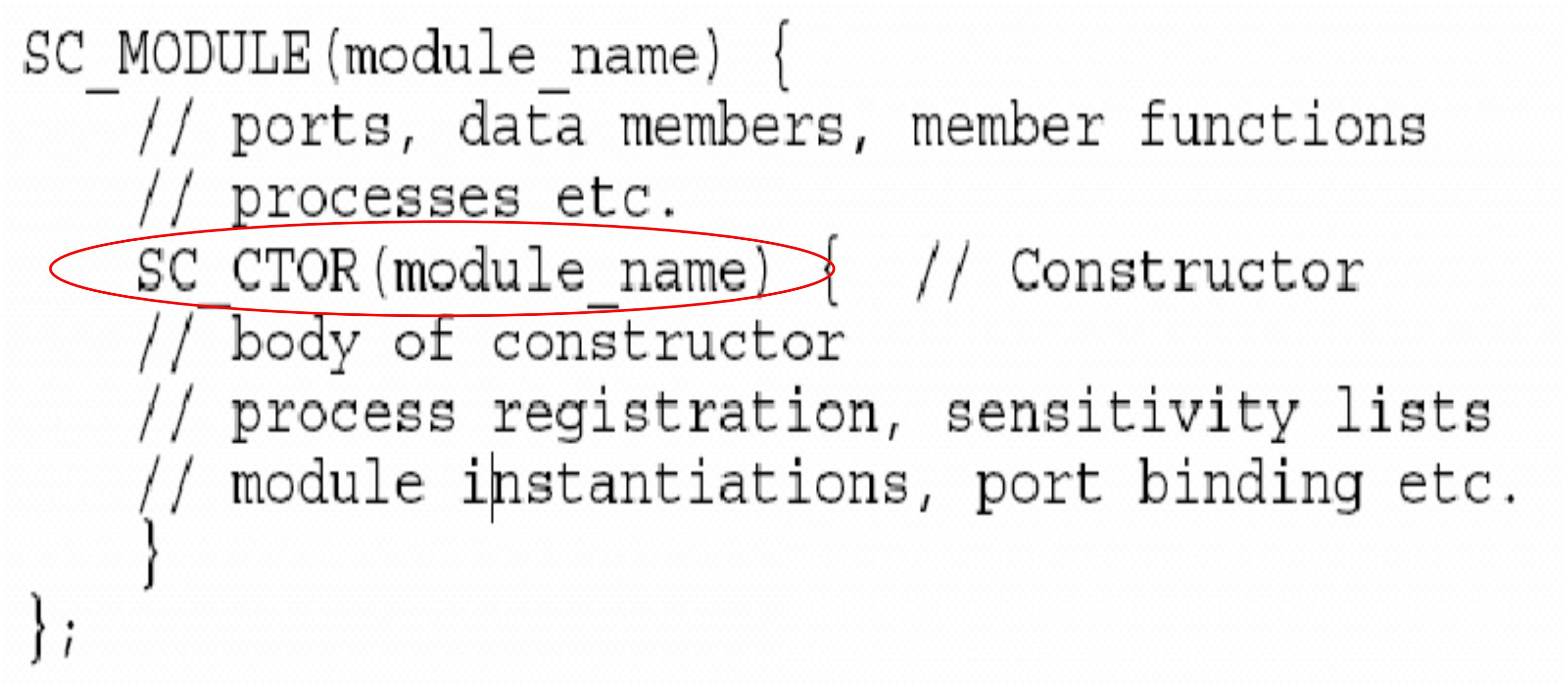
SystemC Language

- Design Example
- Data type
- Port & Signal
- Module Declaration
- Top Level & Testbench

Module Declaration

```
#include "systemc.h"
SC_MODULE(updown)
{
  sc_in<bool>  x_din;
  sc_in<bool>  x_clk;
  sc_out<bool> x_dout;
  void doit() ;

  SC_CTOR(updown)
  {
    SC_METHOD(doit);
    sensitive_pos << x_clk;
  }
};
```

SC_STORE



```
SC_MODULE(module_name) {
    // ports, data members, member functions
    // processes etc.
    SC_CTOR(module_name) {   // Constructor
    // body of constructor
    // process registration, sensitivity lists
    // module instantiations, port binding etc.
    }
};
```

Process

- Method (SC_METHOD)
- Thread (SC_THREAD)
- Clocked Thread (SC_CTHREAD)

Process

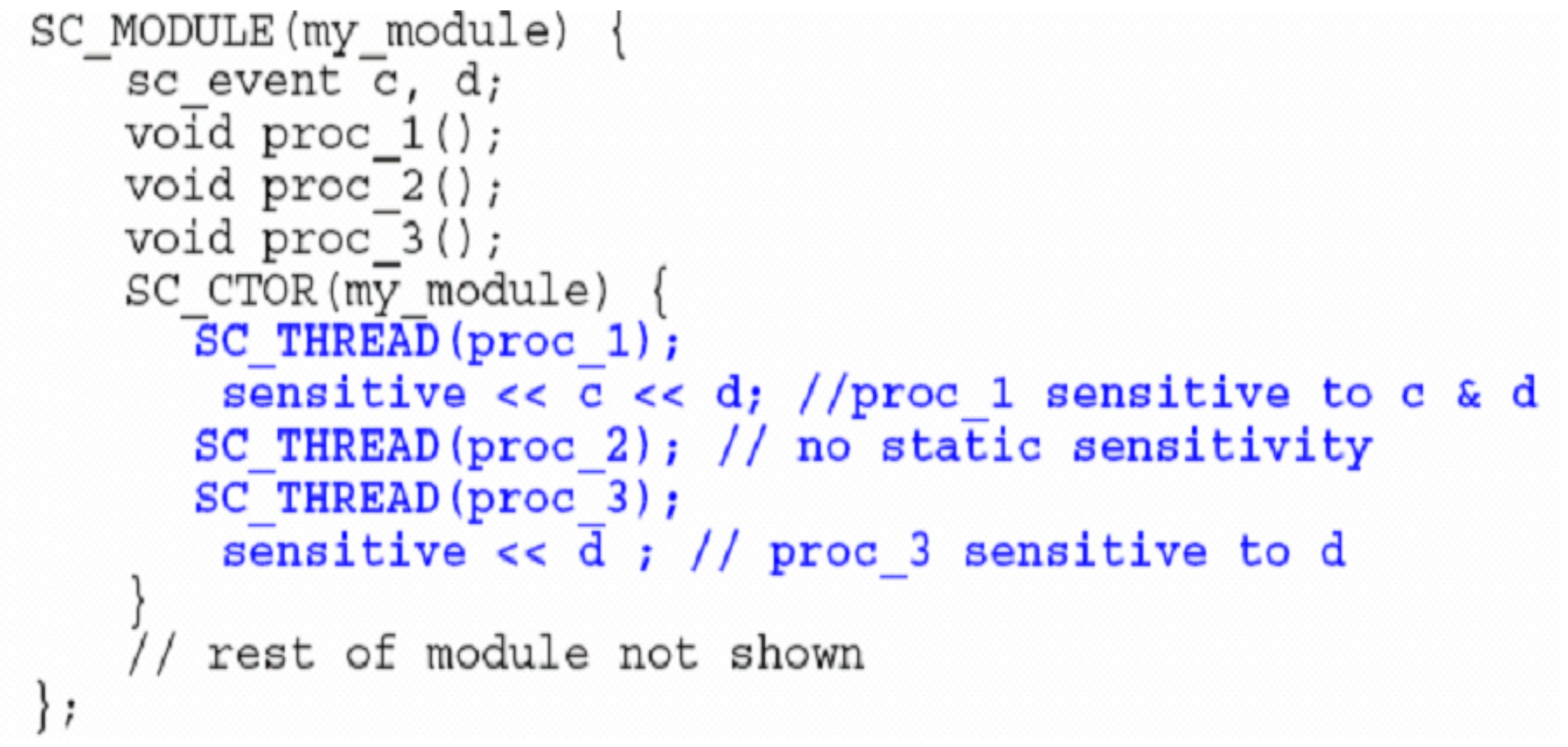
	SC_METHOD	SC_THREAD	SC_CTHREAD
triggered	by signal events	by signal events	by clock edge
infinite loop	no	yes	yes
execution suspend	no	yes	yes

suspend & resume	-	wait()	wait()
			wait_until()

construct & sentisize method	SC_METHOD(*p*);
	
	sensitive(*s*);
	sensitive_pos(*s*);
	sensitive_neg(*s*);
		SC_THREAD(*p*);
		
		sensitive(*s*);
		sensitive_pos(*s*);
		sensitive_neg(*s*);
			SC_CTHREAD (*p*,*clock*.pos());
			
			SC_CTHREAD (*p*,*clock*.neg());

modeling example (hardware)	combinational logic	sequential logic at RT level (asynchronous reset, etc.)	sequential logic at higher design levels

Process



```
SC_MODULE(my_module) {
    sc_event c, d;
    void proc_1();
    void proc_2();
    void proc_3();
    SC_CTOR(my_module) {
       SC_THREAD(proc_1);
        sensitive << c << d; //proc_1 sensitive to c & d
       SC_THREAD(proc_2); // no static sensitivity
       SC_THREAD(proc_3);
        sensitive << d ; // proc_3 sensitive to d
    }
    // rest of module not shown
};
```

SystemC Language

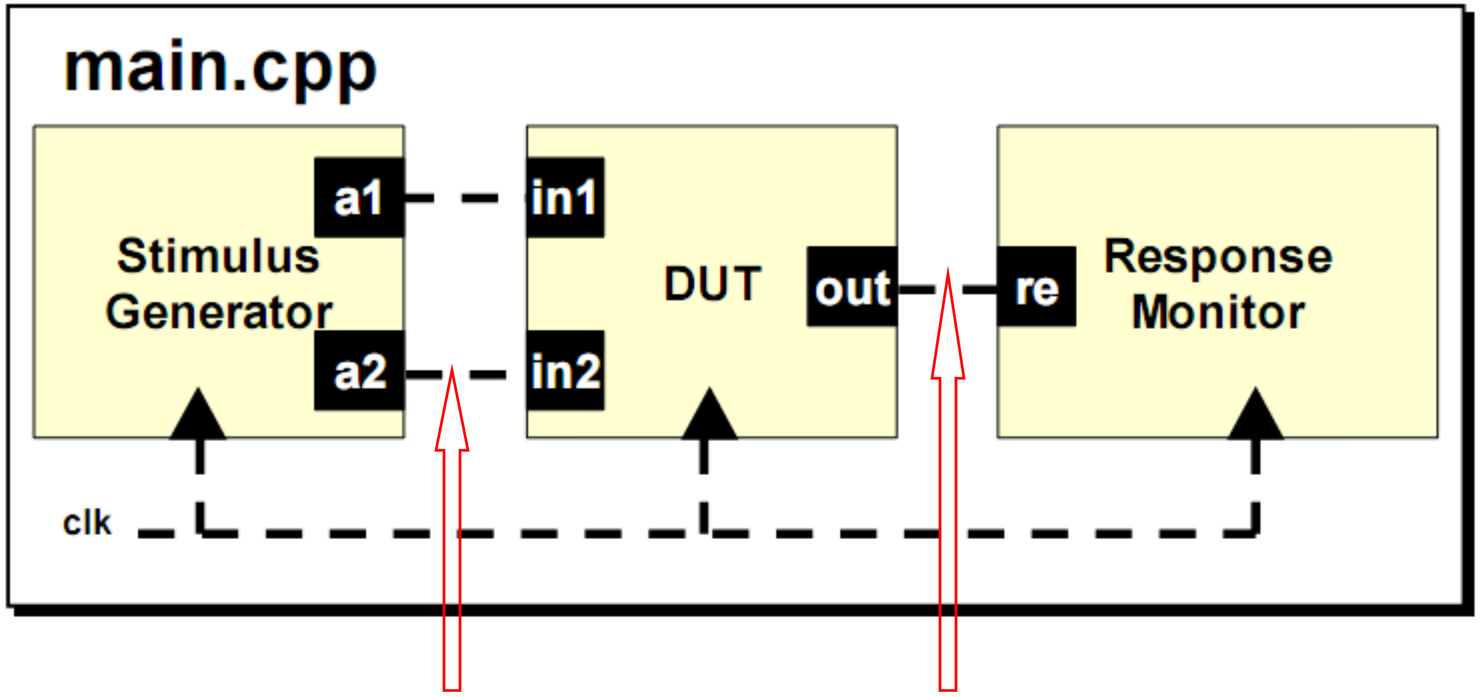
- Design Example
- Data type
- Port & Signal
- Module Declaration
- Top Level & Testbench

Top Level & Testbench

```
// implementation file: main.cc

#include "systemc.h"
#include "process_1.h"
#include "process_2.h"

int  sc_main  (int ac,char *av[])
{
     sc_signal<int> s1 ( “Signal-1“ );
     sc_signal<int> s2 ( “Signal-2“ );
     sc_signal<bool> ready_s1 ( “Ready-1“ );
     sc_signal<bool> ready_s2 ( “Ready-2“ );

     sc_clock clock( "Clock“ , 20 , 0.5 , 0.0 );

     process_1 p1 ( “P1“ );
     p1.clk( clock );
     p1.a( s1 );
     p1.ready_a( ready_s1 );
     p1.b( s2 );
     p1.ready_b( ready_s2 );
```

```
process_2 p2 ( “P2“ );
     p2.clk( clock );
     p2.a( s2 );
     p2.ready_a( ready_s2 );
     p2.b( s1 );
     p2.ready_b( ready_s1 );

     s1.write(0);
     s2.write(0);
     ready_s1.write(true);
     ready_s2.write(false);

     sc_start(100000);

     return 0;
}
```

Top Level



SC_signal SC_signal

DOUT_A (“dout”)

A1.in1(a1)

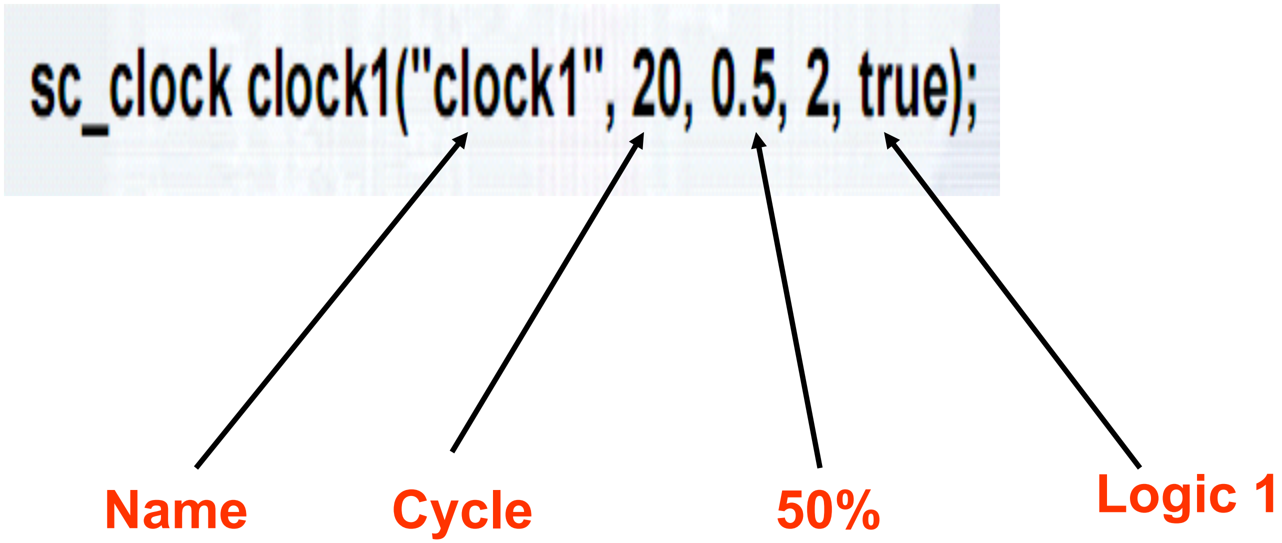
A2.in2(a2)

A3.out(re)

Clock Object

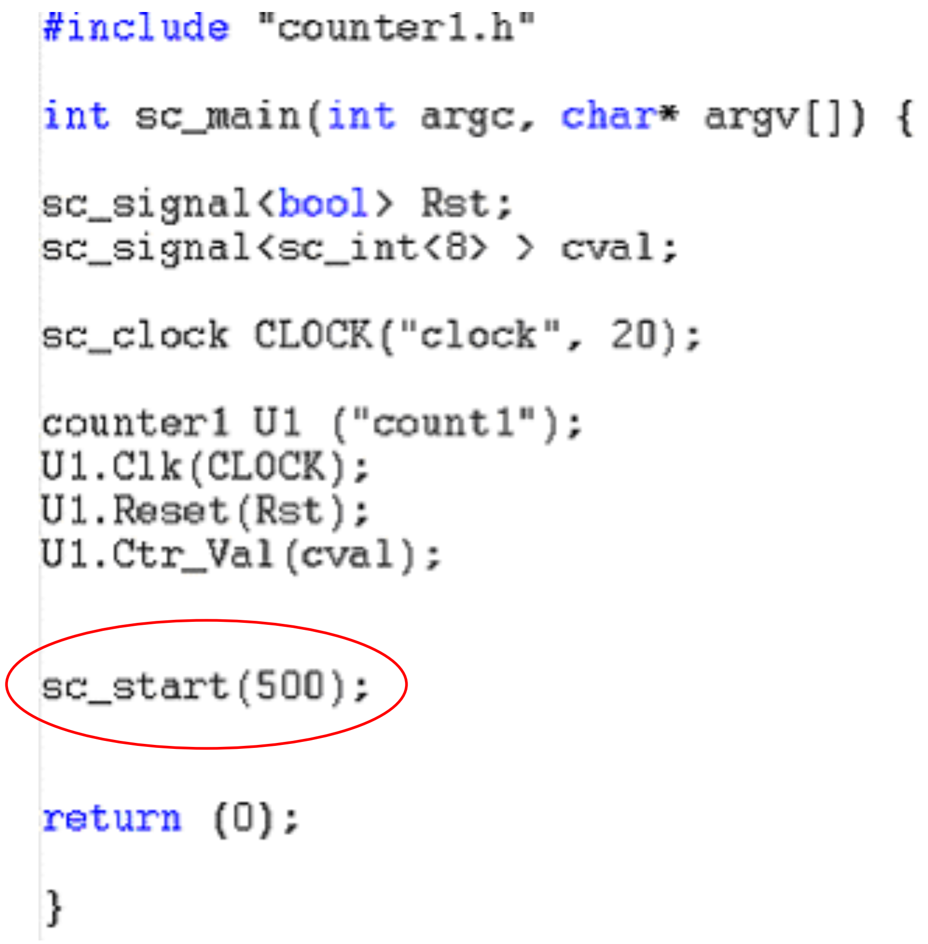
```
int sc_main(int argc, char*argv[]) {
  sc_signal<int>  val;
  sc_signal<sc_logic> load;
  sc_signal<sc_logic> reset;
  sc_signal<int> result;

  sc_clock  ck1("ck1", 20, 0.5, 0, true);

  filter f1("filter");
  f1.clk(ck1.signal());
  f1.val(val);
  f1.load(load);
  f1.reset(reset);
  f1.out(result);

  // rest of sc_main not shown
```

Clock Object



```
sc_clock clock1("clock1", 20, 0.5, 2, true);
```

SC_START Function



```
#include "counter1.h"

int sc_main(int argc, char* argv[]) {

sc_signal<bool> Rst;
sc_signal<sc_int<8> > cval;

sc_clock CLOCK("clock", 20);

counter1 U1 ("count1");
U1.Clk(CLOCK);
U1.Reset(Rst);
U1.Ctr_Val(cval);


sc_start(500);


return (0);

}
```

Graphic Out

Creating a file VCD :

```
sc_trace_file * my_trace_file;
my_trace_file = sc_create_vcd_trace_file(“my_trace”);
```

Simulate by the Funcion :

```
sc_signal<int> a;
float b;

sc_trace(trace_file, a, "MyA");
sc_trace(trace_file, b, "B");
```

Thanks you