SystemC: Main Features

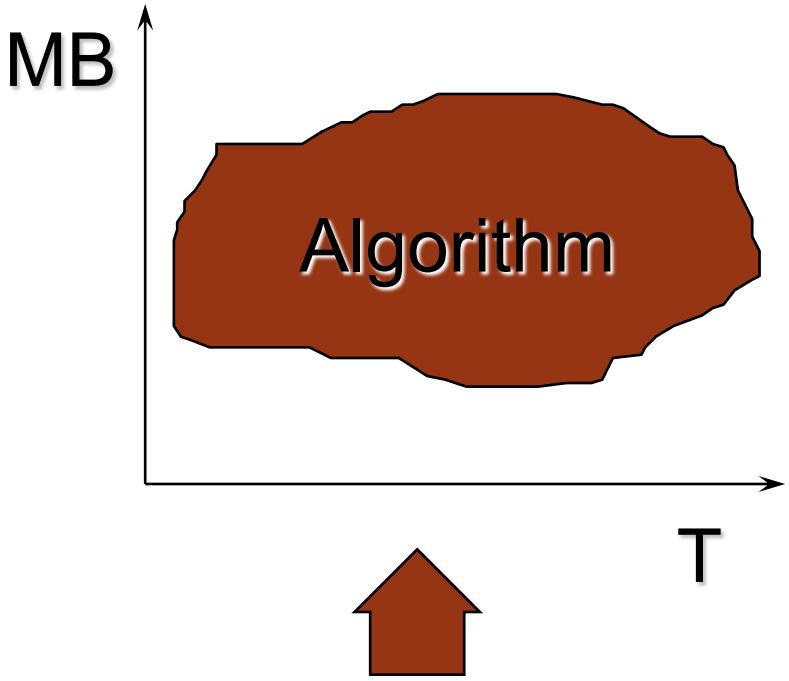
Franco Fummi

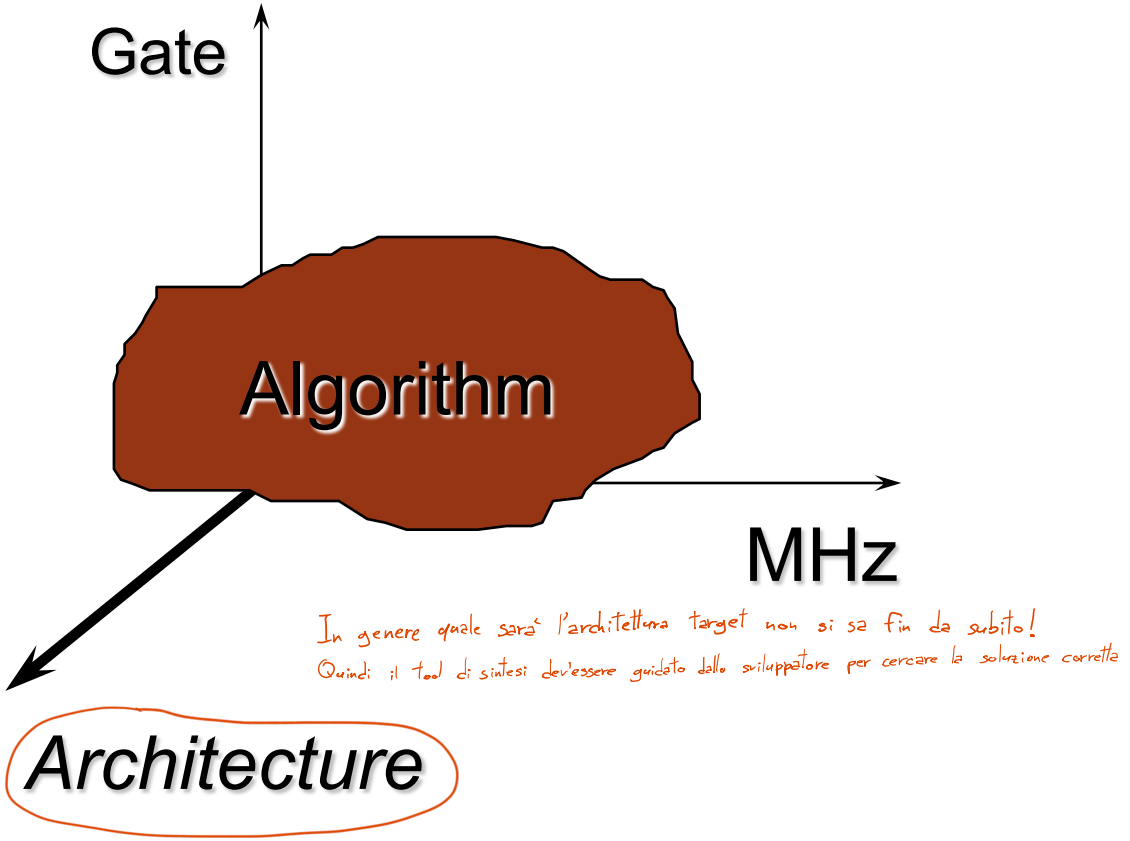
UNIVERSITÀ di VERONA
Dipartimento di INFORMATICA

Version 1.1

Software versus Hardware Design

Trade-off migliore tra memoria e tempo d'esecuzione





In genere quale sarà l'architettura target non si sa fin da subito!
Quindi il tool di sintesi dev'essere guidato dallo sviluppatore per cercare la soluzione corretta

Example: GCD modeled in C

```
#include <stdio.h>

int gcd(int xi, int yi)
{
    int x, y, temp;

    x = xi;
    y = yi;
    while (x > 0){
        if (x <= y){
            temp = y;
            y = x;
            x = temp;
        }
        x = x - y;
    }
    return(y);
}
```

```
main()
{
    int xi, yi, ou;

    scanf("%d %d", &xi, &yi);
    ou = gcd(xi, yi);
    printf("%d\n", ou);
}
```

Hardware requirements (1)

- Input /Output
  - S: printf, scanf...
  - H: component interface must be defined
- Timing
  - S: CPU instructions are executed at the CPU clock speed
  - H: one or more explicit CLOCK signals must be defined

Hardware requirements (2)

- Variables size
  - S: hidden implicit definition (integer 4bytes, char 1byte, …)
  - H: all pre-defined and user-defined types must be translated into bit vectors
- Relationships operands/operators
  - S: all operators in the C libraries are accepted
  - H: explicit mapping of operands on operators

Hardware requirements (3)

- Memory elements identification
  - S: the optimization module of the compiler transparently maps variables onto CPU registers and memory elements
  - H: the synthesis tool identifies memory elements by analyzing the algorithmic semantics
- Modules synchronization
  - S: sequential execution of instructions
  - H: inherently parallel execution of all components

SystemC history

- Open SystemC Initiative (OSCI) (1999-2000)
  - a standard for modeling digital systems
  - founders:
    - Synopsys, CoWare, Frontier Design ... ARM, Cygnus, Ericsson, Fujitsu, Infineon, Lucent, Sony, ST, TI ...
  - free use of the language
  - controlled language extension
  - open market for tools
  - RTL development completed, TLM and AMS in prototypes
- Language moved to AcceLera (2016)
  - maintenance guaranteed
  - Hard further evolutions
  - SystemC 2.3.1 including TLM and AMS

RTL SystemC key features

Register Transfer Level

Hw synth con un tool per avere una completa visione finale (clock cycles)

Concetti da usare in C++ in RTL SystemC

- Concurrency:
- Communication:
- Notion of time:
- Reactivity:
- Hardware data types:

Possiamo descrivere direttamente le dim dei tipi

- Simulation support: Il simulatore è integrato!
- Debugging support:
- Processes (syn and asyn)
- Signals, channel
- Multiple clocks with arbitrary phases
- Waiting for events
- Bit vectors, arbitrary precision integer
- Simulation kernel
- C++ debugging tools

Use of SystemC distribution

- Header files:
  - SystemC class definition
- Libraries:
  - Class library
  - Simulation kernel
- Building strategy:
  - compilation of home-made classes
  - linking of libraries

Executable program with simulation capabilities

L'aver implementato SystemC come una libreria implica che per definire un nuovo tipo devo implementare una nuova classe, che potrebbe sembrare comodo ma se devo ad esempio fare l'override di un operatore e' un gran problema (es: ho definito interi da 17 bit, e devo reimplementare la somma, quindi devo associare al simbolo "+" il mio nuovo metodo)

Ogni volta che posso usare i tipi nativi di C++, li uso ⇒ molto più veloci

SystemC: main characteristics

- SystemC Reference Language manual:
  - Modules and Hierarchy (cap. 3)
  - Processes (cap. 4)
  - Ports and Signals (cap. 5)
  - Data Types (cap.6)
  - Hardware Examples (app. A)

SC_MODULE example

Upper case = macro

```
SC_MODULE(alu) {
// input/output ports
    sc_in<bool>     clock;
    sc_in<sc_int<N> > op1;
    ...
    sc_out<sc_int<N> >    o;
    // method
    void reg_par_par();
    void calcola();
    // internal signals
    sc_signal<sc_int<N> > acc;
    sc_signal<sc_int<N> > t6;
    // constructor
    SC_CTOR(alu) {
        SC_METHOD(reg_par_par);
        sensitive_pos(clock);
        SC_METHOD(calcola);
        sensitive(op1);
        ...
    };
};
```

input port

tipo della port

Tipo systemc_int di N bit

output port

I/O ports of the module

added signal in the module

Nome del modulo

SysC controlla attivazione/disattivazione del metodo reg_par_par

Descrive il comportamento del modello

Uno o più processi concorrenti collegati tramite segnali

SC_THREAD

NB

Process types

- SystemC supports three kinds of process:
  - methods (`SC_METHOD`):
    - executed from the start to the end → Il codice prima o poi DEVE finire, altrimenti la simulazione fallisce
    - sensitive to signals → Viene chiamato quando un segnale varia → Ogni volta che un segnale varia, il simulatore genera un nuovo metodo
  - threads (`SC_THREAD`): ⇒ I thread devono includere ALMENO una wait
    → Quando una thread arriva all'ultima istruzione, NON viene rieseguita
    - executed up to a `wait()`
  - ~~clocked threads (SC_CTHREAD):~~ ⇒ Deprecato
    - ~~sensitive to clocks~~

SC_METHOD Process

- `SC_METHOD` is a process that is executed from the first instruction to the last one every time it is called
- A *sensitivity list* is mandatory; it includes all signals that are able to active it every time they have an event
- The process is executed every time at least one signal in the list changes its value
- They are used to model combinational behaviors or finite state machines
- Example of `SC_METHOD`:
  - `SC_METHOD(reg_par_par);`
    `sensitive_pos(clock);` → Il metodo è sensibile al fronte positivo del segnale (passa da 0 a 1)
  - `reg_par_par` is executed up to its end, every time there is a positive event on the `clock` signal

SC_THREAD Process

- A `SC_THREAD` is a process that can be suspended and restarted
- A *sensitivity list* is mandatory; it includes all signals that are able to active it every time they have an event
- The process is suspended every time a `wait()` function is executed
  - the execution restarts when there is an event on at least one signal of the sensitivity list
  - The execution restarts from the instruction following the `wait()`
- Execution efficiency depends on the number of contest-switches

SC_THREAD Process

- `SC_THREAD` processes are started only once at the beginning of the simulation, while `SC_METHOD` processes are continuously restarted
- `SC_THREAD` processes remain active up to the execution of their last instruction and then they are definitively terminated
- To avoid termination, a `SC_THREAD` is usually based on an infinitive cycle implemented with a `while(true)` instruction; the cycle includes one or more `wait()` functions
- The `watching()` instruction is used to force the restart of the infinitive loop, e.g., to implement an asynchronous reset
- An example of `SC_THREAD`:
  - ```
    SC_THREAD(compute);
      sensitive(trigger);  // (re)executed when trigger changes
      watching(reset.delayed()==1);   // restarted when reset is 1
    ```

quando l'attributo delayed di reset vale 1
    ```

Process example

```
SC_MODULE(my_module) {
 // ports declaration
    sc_in<int>  a;
    sc_in<bool> b;
    sc_out<int> x;

 // signals declaration
    sc_signal<bool> c;

 // process declaration
    void my_method_proc();
 // constructor
    SC_CTOR(my_module) {
    // process record
            SC_METHOD(my_method_proc);
            // sensitivity list declaration
    };
};
```

Simulation kernel

- SystemC scheduler works as follows:

1 clock almeno per poter far partire la simulazione

In 1 ciclo di clock ci aspettiamo che il sistema si sia stabilizzato: i segnali di input non oscillano, quindi non generano altri processi/thread

1. all clock signals are updated
2. all SC_METHOD's and SC_THREAD's with modified input values are executed
3. goto 2 up to a fixed point is reached, then goto 4
4. increase execution time and goto 1

È stabilito un numero max di iterazioni tra i passaggi 2 e 3, dopodiché la simulazione viene interrotta

⇓

D: default è 1000 ma può essere modificato

Port and signal types

- `sc_int<n> sc_uint<n>`
- `sc_bigint<n> sc_biguint<n>`
- `sc_bit` usare bool nativo e non sc_bit
- `sc_logic` → due stati: unknown → importante se il mio stato iniziale non è ben definito o se voglio controllare che il reset funzioni correttamente; don't care
- `sc_bv<n> sc_lv<n>` bit vector, logic vector
- `sc_fixed sc_ufixed`
- `sc_fix sc_ufix`
- end user self defined structures
- Examples:
  - `sc_in<port_type>;`
    - // input port of type `port_type`
  - `sc_out<port_tipe> x[32];`
    - // output port ranging from x[0] to x[31] of type `port_type`
  - `sc_signal<port_type> i[4];`
    - // signal ranging from i[0] to i[3] of type `port_type`

Synchronous D-flip-flop

```
// dff.h
#include "systemc.h"

SC_MODULE(dff) {        // module declaration
    sc_in<bool> clock; //input declaration
    sc_in<bool> din;
    sc_out<bool> dout; //output declaration

    void doit();{
        dout = din;
    }
    SC_CTOR(dff) {
        // declaration of a SC_METHOD
        // process sensitive to clock
        SC_METHOD(doit);
        sensitive_pos(clock);
    };
};
```

Asynchronous D-flip-flop

```
// dffa.h
#include "systemc.h"

SC_MODULE(dffa) {     //module declaration
     sc_in<bool>  clock; //input declaration
     sc_in<bool>  reset;
     sc_in<bool>  din;
     sc_out<bool> dout; //output declaration
     void do_ffa();{
          if (reset){
               dout = false;
          }else if (clock.event()){
               dout = din;}
     }
     SC_CTOR(dffa) {
          SC_METHOD(do_ffa);
          sensitive (reset);
          sensitive_pos(clock);
     };
};
```

Per capire se siamo nel metodo per questo segnale

Parallel-parallel register

```
// reg_par_par.h
#include "systemc.h"
#define N 8
SC_MODULE(reg_par_par) {
    sc_in<bool> clock;
    sc_in<sc_bv<N> >  d;
    sc_out<sc_bv<N> > q;

    //method to build the register
    void register_par_par();

    // constructor declaration
    SC_CTOR(reg_par_par) {
        // declaration of a SC_METHOD
        // process sensitive to clock
        SC_METHOD(register_par_par);
        sensitive_pos(clock);
    };
};
```

```
// reg_par_par.cpp
#include "reg_par_par.h“

// method implementation
void reg_par_par::register_par_par()
{
  // local variable
  static sc_bv<N> reg;

  // input port reading
  reg = d.read();

  // output port writing
  q.write(reg);
}
```

-> Ottengo il valore di una porta input del segnale

-> Scrivero sull'output

Serial/serial register

```
// reg_ser_ser.h
#include "systemc.h"
#define N 8

SC_MODULE(reg_ser_ser) {
    sc_in<bool> clock;
    sc_in<bool> i0;
    sc_out<bool>    o;

    void register_ser_ser();

    SC_CTOR(reg_ser_ser) {

    SC_METHOD(register_ser_ser);
        sensitive_pos(clock);
    };
};
```

```
// reg_ser_ser.cpp
#include "reg_ser_ser.h"
void reg_ser_ser::register_ser_ser()
{
    static sc_bv<N> reg = "00000000";
    bool            i01;

    i01 = i0.read();

    reg.range(N-2,0) = reg.range(N-1,1);
    reg[N-1] = i01;
    i01 = (reg[0] == '1') ? 1 : 0;

    o.write(i01);
}
```

Parallel/serial register

```
// reg_par_ser.h
#include "systemc.h"
#define N 8
SC_MODULE(reg_par_ser) {
    sc_in<bool>     clock;
    sc_in<bool>     i0;
    sc_in<sc_bv<N> >  d;
    sc_in<bool>     ps;
    sc_out<sc_bv<N> > q;
    sc_out<bool>        o;

    void register_par_ser();

    SC_CTOR(reg_par_ser) {

    SC_METHOD(register_par_ser); }
        sensitive_pos(clock);
    };
};
```

```
// reg_par_ser.cpp
void reg_par_ser::register_par_ser()
{
    static sc_bv<N> reg = "00000000
    bool          i01, ps1;
    ps1 = ps.read();
    if (ps1 == 1){
        reg =d.read();
    }else{
        i01 = i0.read();
        reg.range(N-2,0)=reg.range(N-1,1);
        reg[N-1] = i01;
    }
    i01 = (reg[0] == '1') ? 1 : 0;
    o.write(i01);
    q.write(reg);
```

Shifter

```
// shiftrer.h
#include "systemc.h"
#define N 8

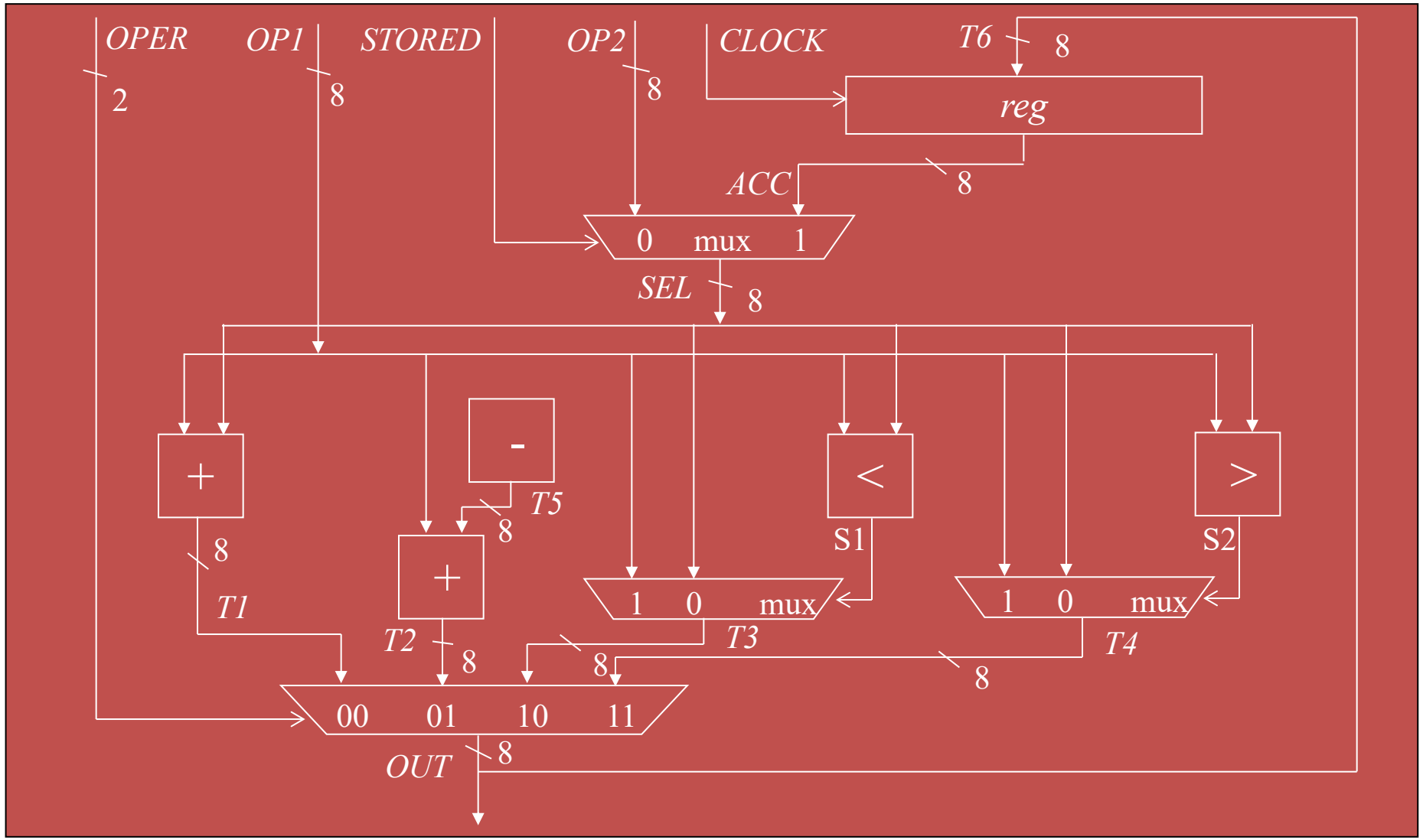
SC_MODULE(shifter) {
    sc_in<bool>      ds;
    sc_in<sc_bv<N> >     a;
    sc_in<bool>      i0;
    sc_out<sc_bv<N> >    o;

    void shift();

    SC_CTOR(shifter) {
        SC_METHOD(shift);
        sensitive(ds);
        sensitive(a);
        sensitive(i0);
    };
};
```

```
// shifter.cpp
#include "shifter.h"
void shifter::shift()
{
    bool     ds1;
    bool     i01;
    sc_bv<N> a1;
    sc_bv<N> c1;
    i01 = i0.read();
    ds1 = ds.read();
    a1 = a.read();
    // rigth shift
    if(ds1 == 1){
        c1.range(N-2,0) = a1.range(N-1,1);
        c1[N-1] = i01;
    }else{
        c1.range(N-1,1) = a1.range(N-2,0);
        c1[0] = i01;
    }
    o.write(c1);
}
```

ALU description



ALU module

```
// alu.h
#include "systemc.h"
#define N 8
#define P 2
SC_MODULE(alu) {
    sc_in<bool>      clock;
    sc_in<sc_int<N> >   op1;
    sc_in<sc_int<N> >   op2;
    sc_in<bool>      stored;
    sc_in<sc_uint<P> >  oper;
    sc_out<sc_int<N> >  o;

    // methods
    void reg_par_par();
    void calcola();

    // signals
    sc_signal<sc_int<N> >   acc;
    sc_signal<sc_int<N> >   t6;
};
```

```
SC_CTOR(alu) {
    SC_METHOD(registro_par_par);
    sensitive_pos(clock);
    SC_METHOD(calcola);
    sensitive(op1);
    sensitive(op2);
    sensitive(stored);
    sensitive(oper);
    sensitive(acc);
};
```

ALU implementation

```
// alu.cpp
#include "alu.h"
void alu::register_par_par(){..}      // see previous slides
void alu::calcola()
{
     sc_int<N>      op11, op21, acc1, sel, t61;
     bool           stored1;
     sc_uint<P>     oper1;

     op11 = op1.read(); op21 = op2.read();
     acc1 = acc.read(); stored1 = stored.read();
     oper1 = oper.read()

     sel = (stored1 == 1) ? acc1; op21;
     switch(opr1){
          case 0:   t61 = op11+sel; break;
          case 1:   t61 = op11-sel; break;
          case 2:   if(op11 < sel) t61=op11; else t61=sel;
                    break;
          case 3:   if(op11 > sel) t61=op11; else t61=sel;
                    break;
     }
     t6.write(t61); o.write(t61);
}
```

SC_MAIN

- The correct link to the simulation kernel needs to write the `SC_MAIN`, which is called by the systemc library
- It hat has to include:
  - definition of at least one clock signal
  - timing evolution of the clock to allow the simulation time to advance
  - instantiation of the top-level module of the design
  - call to simulation start method
  - optional:
    - monitor of signals to produce a VCD file

↳ Wave forms file