Arquitectura de Computadores Avançada

MIECT

Nuno Lau (nunolau@ua.pt)

Introduction to SystemC

- SystemC is a set of C++ classes and a methodology for modeling systems

- This document describes some of the most important elements of language
- This introduction is not intended as a complete description of the SystemC

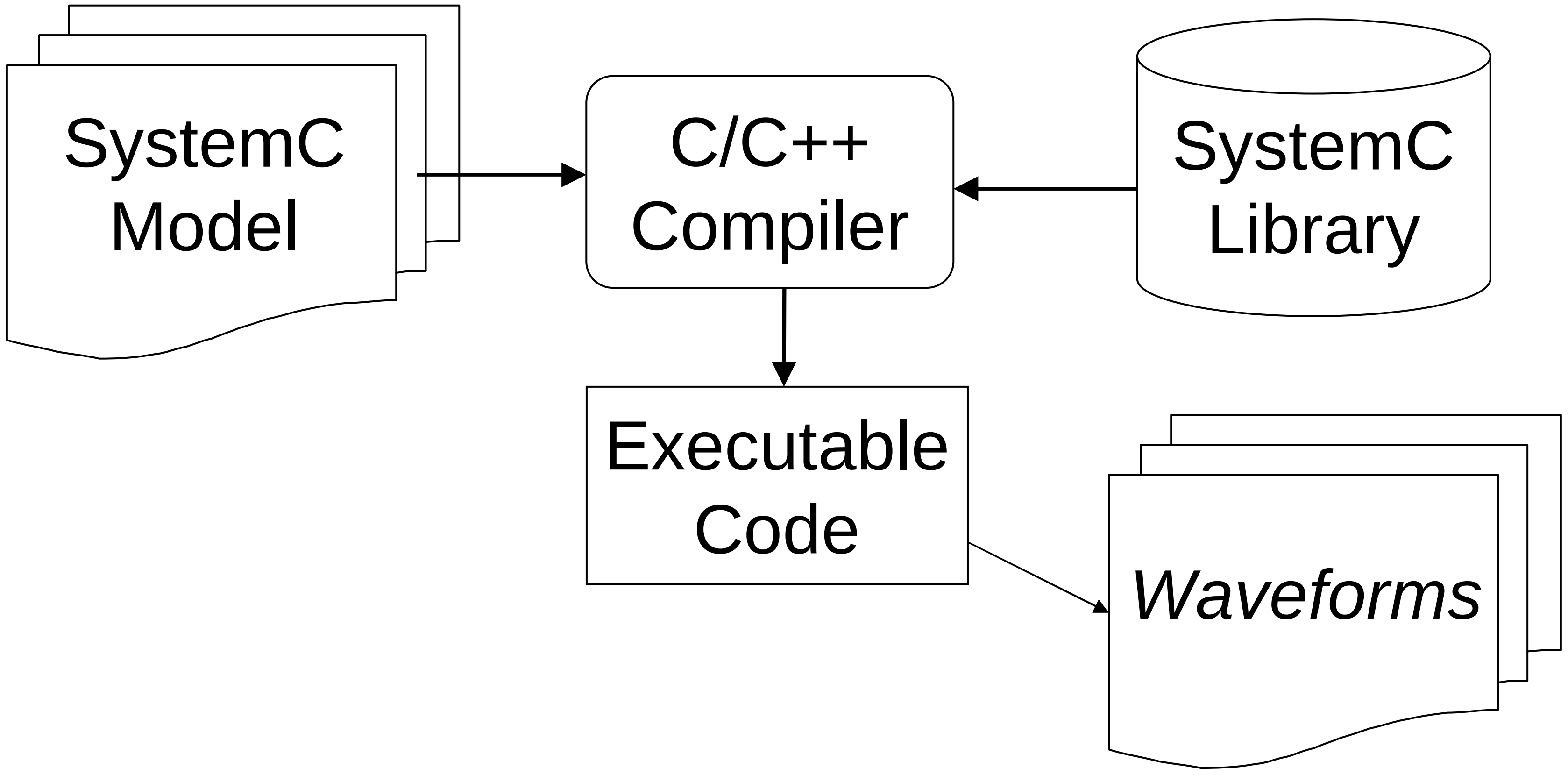
Motivation for SystemC

- Create an executable model of the system to build
- Allows the simultaneous simulation of hardware and software
- The model can be validated and optimized
- There are synthesis tools based on SystemC
- There are other hardware description languages (VHDL, Verilog, ...) and other modeling languages based on C ++ (SpecC, ...)

SystemC provides

- Timed cycle models
- Concurrency
- Reactivity
- Hierarchy
- Hardware oriented data types
- Communication between “hardware type” modules
- Simulation engine
- Waveforms generation

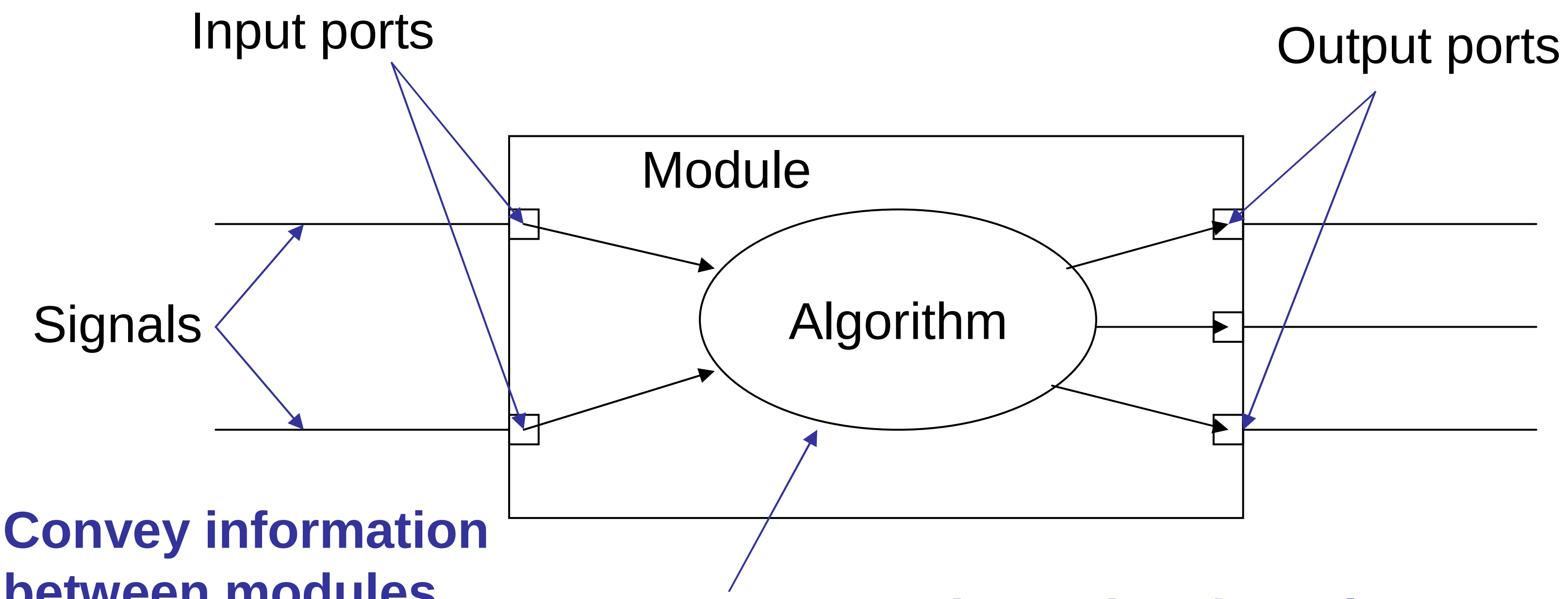
SystemC


SystemC Model
C/C++ Compiler
SystemC Library
Executable Code
Waveforms

Basic language constructions

- **Modules**
  - Represent types of components
  - Basic modeling entities
  - Can contain other modules (instances of other modules)
- **Processes**
  - Used for behavioral description of modules
  - 3 types of processes (**sc_method**, sc_thread, sc_cthread)
- **Ports**
  - Define the Input/Output interface of each module
  - Allow communication between modules
- **Signals**
  - Allow interconnection between modules (through ports)
  - Assigning a value to a signal is not immediate ☹

Model



Convey information between modules (interconnect ports of different modules)

Processes determine the values to write on the output ports taking readings from the input ports (and, possibly, state variables)

Model

Module

```
SC_MODULE(andg) {
   sc_in<bool>  i1,i2;
   sc_out<bool> o;

   SC_CTOR(andg) {
      SC_METHOD(entry);
      sensitive << i1 << i2;
   }
   void entry(void);
};
```

Input ports

Output port

Sensitivity list

Process

(a method of the class)

```
void andg::entry()
{
   o.write(i1.read()
          && i2.read());
}
```

Modules

- Systems are modeled by the interconnection of various modules
  - Complex systems are divided into several simpler modules
- Hides implementation details (communication is only through I/O ports)
- A module may contain:
  - ports, processes, other modules, internal signals, internal variables, internal methods

Modules - syntax

- **SC_MODULE**
  - Module name
  - Declaration of Input and Output ports
- **SC_CTOR**
  - Constructor
  - Behavioral description of the module
- **Good practice**
  - Interface: “.h”
  - Implementation: “.cpp”

```
SC_MODULE(andg) {
   sc_in<bool>  i1,i2;
   sc_out<bool> o;

   SC_CTOR(andg) {
      SC_METHOD(entry);
      sensitive << i1<< i2;
   }
   void entry(void);
};
```

```
void andg::entry()
{
   o.write(i1.read()
          && i2.read());
}
```

Ports

- Define the module interface (**ports are declared within the module**)
- Types of ports
  - Input: `sc_in`
  - Output: `sc_out`
  - Input / Output: `sc_inout`

```
SC_MODULE(andg) {
   sc_in<bool>  i1, i2;
   sc_out<bool> o;

   SC_CTOR(andg) {
      SC_METHOD(entry);
      sensitive << i1<< i2;
   }
   void entry(void);
};
```

Ports

- **All ports must be connected to a signal (or another port)**
- Each port has a **data type**, specified as a template parameter, and a **name** e.g.:
  - `sc_in  <sc_uint<32> > pc;`
  - `sc_out <bool> memRead;`
  - `sc_in  <bool>  i1, i2;`
- Access to ports through:
  - `port_name.read() (e.g. pc.read() )`
  - `port_name.write()(e.g. memRead.write(1))`

Signals

- Carry information between modules or between processes
- Only one type of signal (**`sc_signal`**)
- Signals can be internal to a module or used at the higher level to interconnect modules
- Each signal has a **data type**, specified as a template parameter, and a **name**, e.g.:
  - **`sc_signal <sc_uint <5> > s_rsReg;`**
- **Directionless**

Modules interconnection

- Create instances of modules
- Use signals for interconnection
- Connect ports to signals
  - using names
  - (by order)

```
SC_MODULE(andg3) {
 sc_in < bool > a, b, c;
 sc_out< bool > out;

 sc_signal <bool> s_aux;

 andg *a1, *a2;

 SC_CTOR(andg3) {
   a1 = new andg(“a1”);
   a1->i1(a);
   a1->i2(b);
   a1->o(s_aux);
   a2 = new andg(“a2”);
   a2->i1(s_aux);
   a2->i2(c);
   a2->o(out);
 }
};
```

Modules interconnection

```
SC_MODULE(andg) {
   sc_in<bool>  i1,i2;
   sc_out<bool> o;

   SC_CTOR(andg) {
      SC_METHOD(entry);
      sensitive << i1<< i2;
   }
   void entry(void);
};
```

```
void andg::entry()
{
   o.write(i1.read()
          && i2.read());
}
```

```
SC_MODULE(andg3) {
 sc_in < bool > a, b, c;
 sc_out< bool > out;

 sc_signal <bool> s_aux;

 andg *a1, *a2;

 SC_CTOR(andg3) {
   a1 = new andg(“a1”);
   a1->i1(a);
   a1->i2(b);
   a1->o(s_aux);
   a2 = new andg(“a2”);
   a2->i1(s_aux);
   a2->i2(c);
   a2->o(out);
 }
};
```

Processes

- Allow algorithmic definition of the behavior of a module (behavioral description)
- Similar to C++ methods
- Must be registered in the SystemC simulation engine
- 3 types of processes
  - **Method (sc_method)**
  - Thread (sc_thread) (models testbenches)
  - clocked thread (sc_cthread) (models synchr. FSMs)
- Execute until the end (have also the ability to suspend)

Processes

- **Called when a signal of its sensitivity list changes value**
- Processes use signals to communicate with each other
  - Global variables should not be used
- The value of the signals is not immediately updated
- SC_METHOD processes model combinatorial or sequential circuits

```
//.h
SC_CTOR(andg) {
   SC_METHOD(entry);
   sensitive << i1 << i2;
}

//.cpp
void andg::entry()
{
   o.write(i1.read()
          && i2.read());
}
```

Sensitivity list

- The process is only executed when a value of its sensitivity list changes
- After registration of a process its sensitivity list must be defined:

```
SC_METHOD(entry);
sensitive << i1 << i2;
```

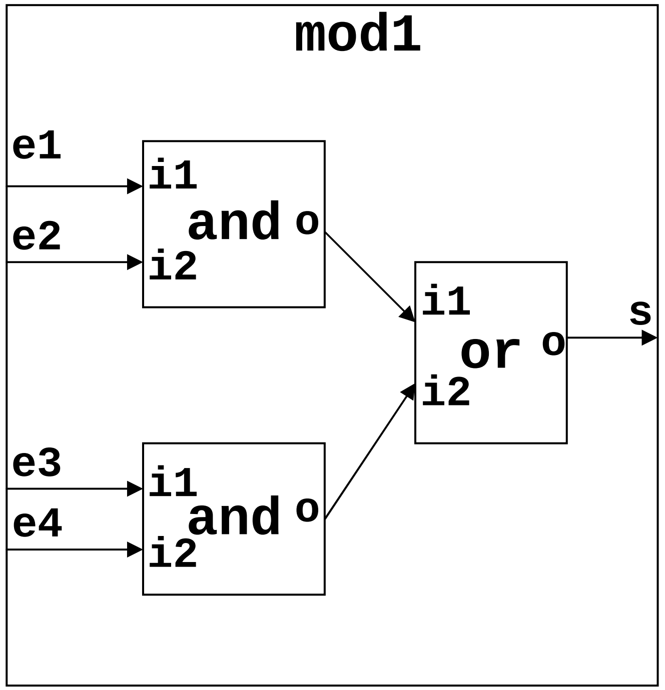
- **Combinatorial modules should include all input ports in its sensitivity list**
- Various types of sensitivity
  - `sensitive << i1 << i2;`
  - `sensitive_pos << clk;`

Data types

- The C++ data types can be used
- Specific SystemC data types
  - **`sc_int <n>, sc_uint <n>`**
    - 1 to 64-bit signed and unsigned integers
  - **`sc_bit`**
    - Single-bit, two-valued ('0', '1')
  - **`sc_logic`**
    - Single bit, four-valued ('0', '1', 'X', 'Z')
  - etc. ...

Hierarchy

- Create a module consisting of several interconnected sub-modules (hierarchy)
  - Declare pointers to sub-modules
  - Define internal signals
  - Create instances of sub-modules
  - Interconnect ports through signals



s = e1.e2 + e3.e4

Initialization

- **`sc_main`** function
- Instantiate modules
- Define waveforms
- Control simulation
  - **`sc_start`**, **`sc_stop`**, **`sc_cycle`**, etc ...

sc_main

```
int sc_main(int argc, char *argv[])
{
    andg andmod; testbench tb;
    sc_signal <bool> a,b,out;

    andmod.i1(a); andmod.i2(b); andmod.o(out);
    tb.o1(a); tb.o2(b);

    sc_trace_file *tf = sc_create_vcd_trace_file("andtest");

    sc_trace(tf,a,“a");sc_trace(tf,b,“b");
    sc_trace(tf,out,"out");

    sc_start(100);

    sc_close_vcd_trace_file(tf);

    return 0;
}
```

Testbench

- Special module used to test the circuit development
- Often uses type processes SC_THREAD
- Execution of the process (or processes) applies test vectors with meaning to the device under test
- Verification through the waveforms of the signals

```
//.h
SC_CTOR(reg_tb)
{
    SC_THREAD(stimuli);
}

//.cpp
void reg_tb::stimuli()
{
    reset.write(true);
    dout.write(2);
    wait(2, SC_NS);

    reset.write(false);
    wait(2, SC_NS);
}
```

Usual errors

- Compilation errors
  - Wrong utilization of templates

```
mux< sc_uint<32> > *mPC;//correct
mux< sc_uint<32>> *mPC; //wrong (needs a space between >>)
```

  - Signals, Modules or Ports undefined

```
mips.cpp:138: no matching function for call to
alu::diin(sc_signal<sc_dt::sc_uint<32> >&)
```

  - Incompatibility between signals and ports (type mismatch)

```
mips.cpp:129: no match for call to
`(sc_out<sc_dt::sc_uint<32> >)
(sc_signal<sc_dt::sc_uint<3> >&)
```

Usual errors

- Execution
  - Port is not connected to a signal

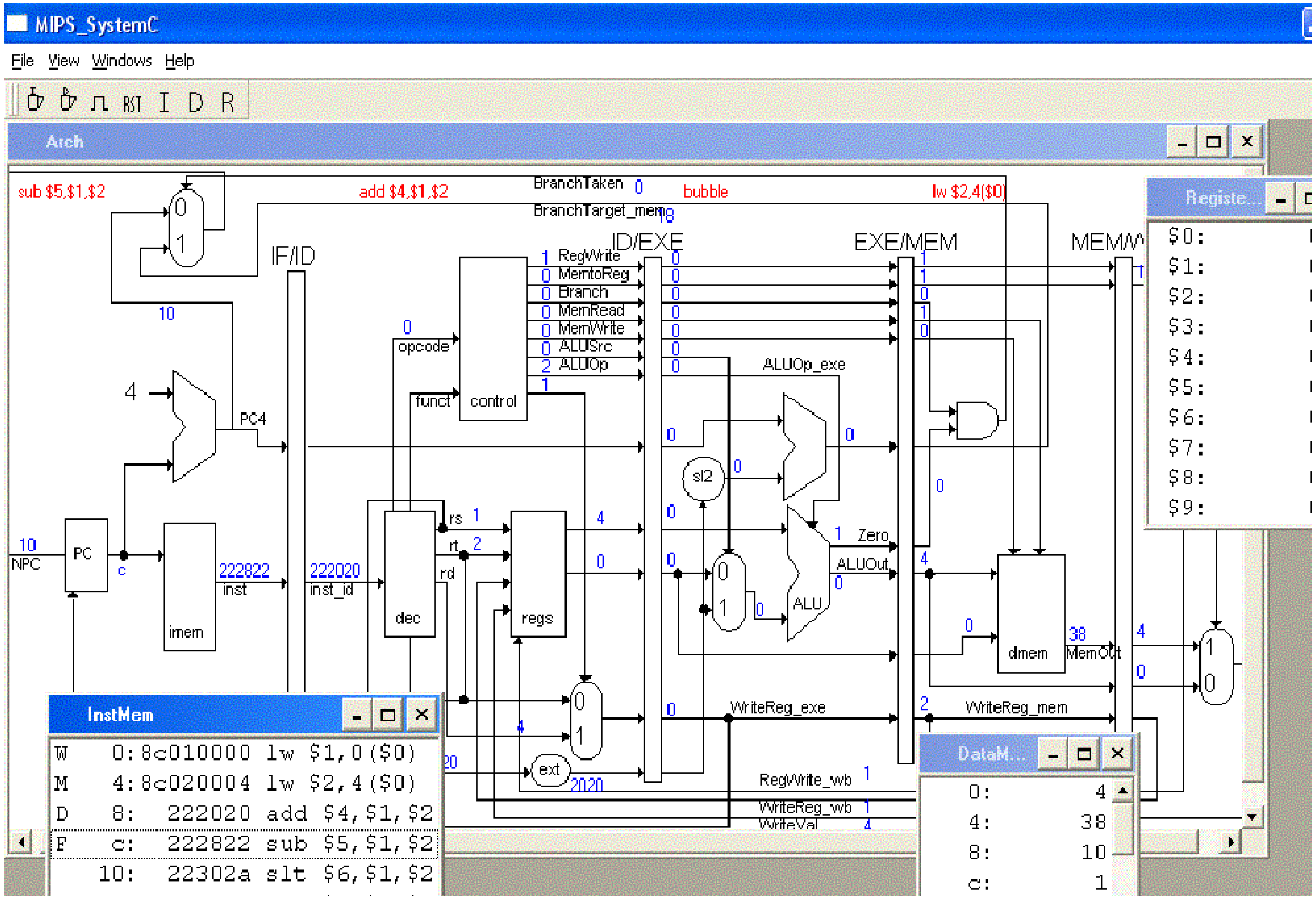
```
Error: (E109) complete binding failed: port
  not
bound: port 'MIPS.alu.port_3' (sc_out)
```

  - Port connected to more than 1 signal

```
Error: (E107) bind interface to port failed:
maximum reached: port 'MIPS.alu.port_3'
  (sc_out)
```

- Specification
  - Sensitivity list incomplete

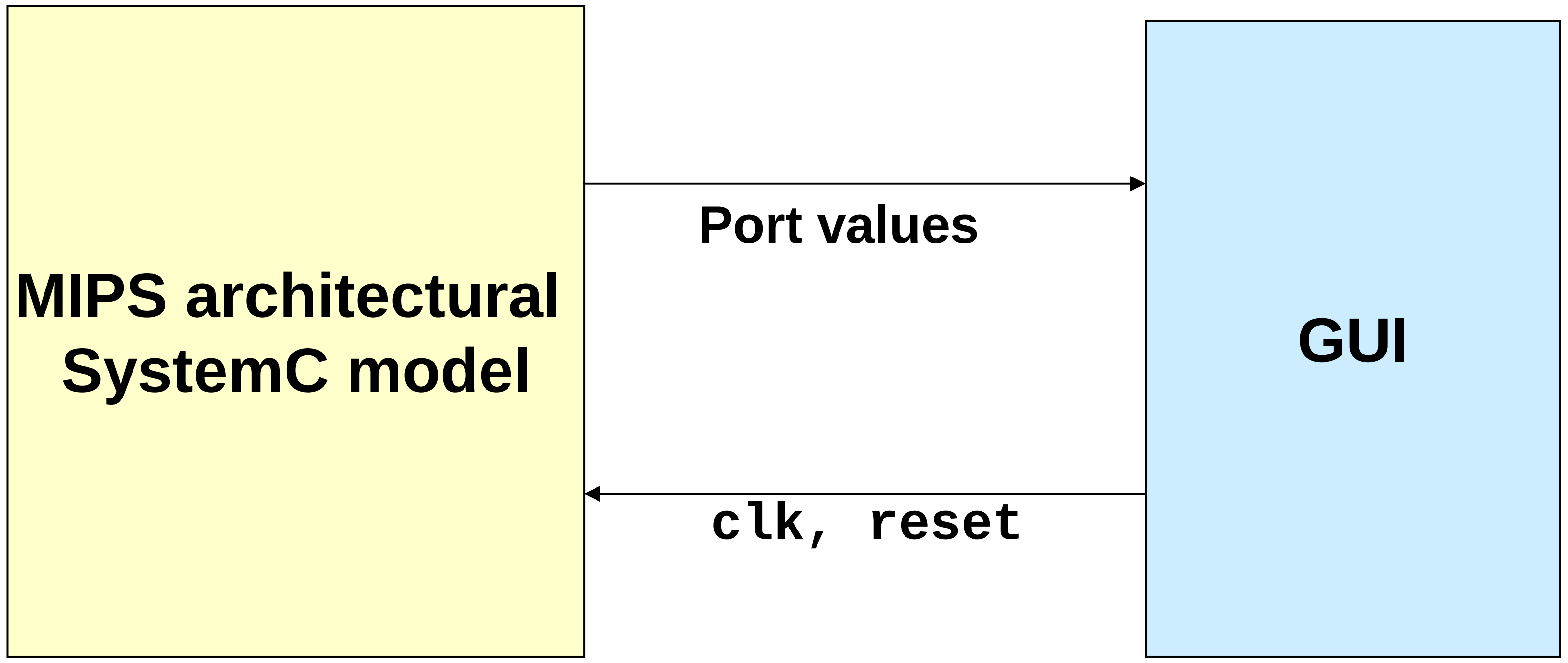
MIPS_SystemC



```
W    0:8c010000 lw $1,0($0)
M    4:8c020004 lw $2,4($0)
D    8:  222020 add $4,$1,$2
F    c:  222822 sub $5,$1,$2
    10:  22302a slt $6,$1,$2
```

```
0:     4
4:    38
8:    10
c:     1
```

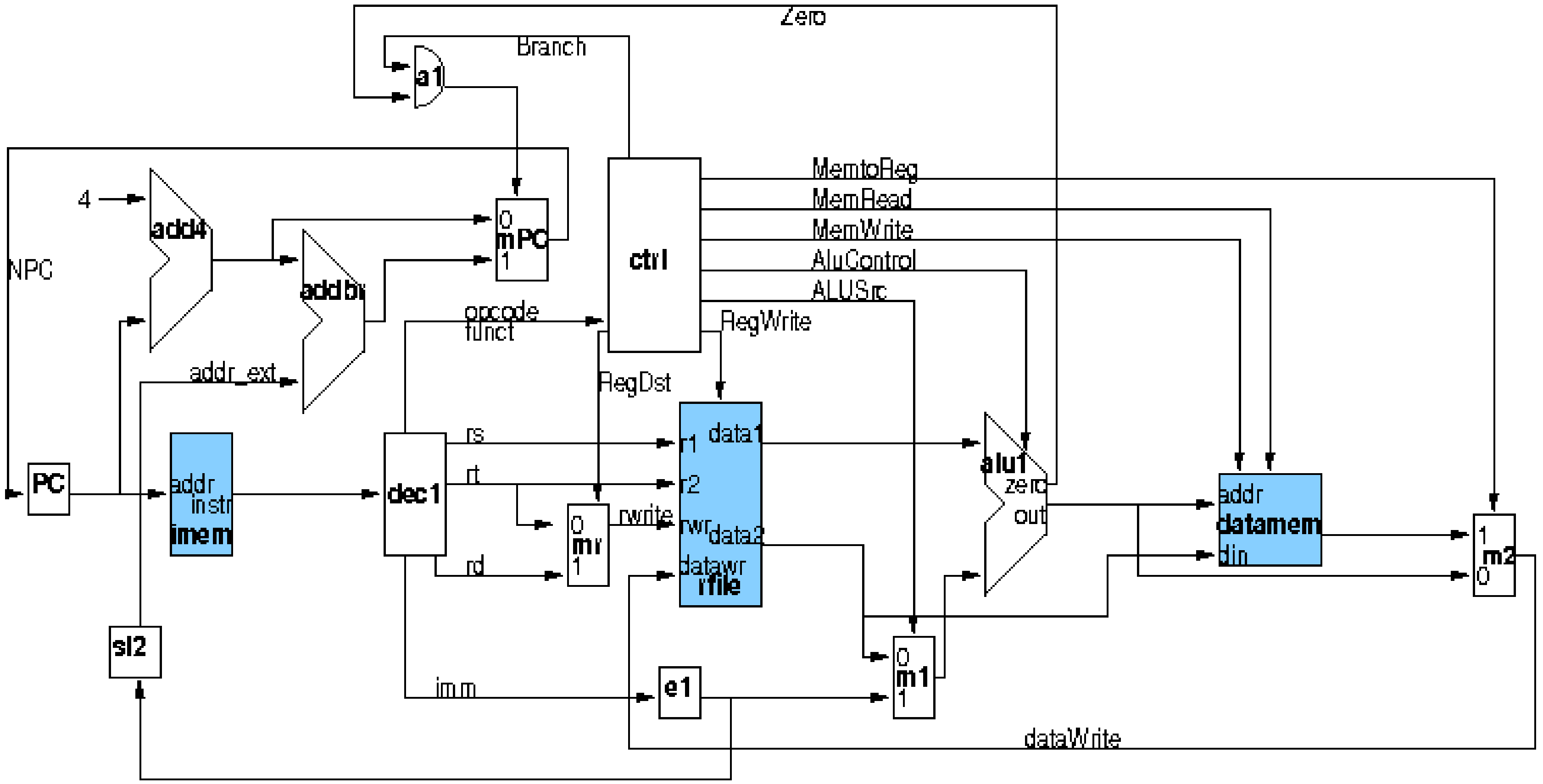
MIPS_SystemC



MIPS_SystemC

- Main modules
  - **`imem`**, **`dmem`**, **`alu`**, **`decode`**, **`rfile`**
  - **`reg_if_id_t`**, **`reg_id_exe_t`**, **`reg_exe_mem_t`**, **`reg_mem_wb_t`**
- Auxiliary modules
  - **`regT`**, **`mux`**, **`andgate`**, **`orgate`**, etc
- Signals connected to the output ports of the pipeline registers are suffixed with the name of the phase where they connect

MIPS_SystemC (single cycle version)



Bibliography

- *SystemC User's Guide*
- *An Introduction to SystemC*, Niemann et al
- MIPS_SystemC reference manual (**`refman.pdf`**) - in Portuguese
- ACA Lab assignements