TDDI08: Embedded Systems Design
- Laboratory introduction -

Adrian Lifa
Embedded Systems Laboratory
Dept. of Computer and Information Science
Linköpings universitet

www.ida.liu.se/~adrli
adrian.alin.lifa@liu.se
Building B, Room 329:206
Lab Assistant:
- Adrian Lifa
  - Office: Building B 329:206
  - Email: adrian.alin.lifa@liu.se
Outline

- Organization (administrative issues)
- Lab 1
  - The HW/SW modeling language “SystemC”

Next time:
- Lab 2
  - The MPARM simulation framework
- Your questions
Lab Information

- **Lab groups**
  - Webreg groups A and B: Adrian Lifa (adrian.alin.lifa@liu.se)

- **Course web page**
  - http://www.ida.liu.se/~TDDI08
  - Check the lab pages for information and links to tutorials

- **Organization**
  - 2 lessons (including this one)
  - 7 two-hour lab sessions

- **2 lab assignments (one tutorial for each)**
  1. SystemC modeling and simulation
  2. Design space exploration in MPARM
Lesson 1 – January 22, 17.15-19.00, S10
  • Administrative issues and SystemC
Lesson 2 – February 6, 15.15-17.00, P36
  • MPARM simulation framework and your questions
Lab Information

- **Deadline for all lab assignments:** March 22, 2015, 23:59
  - Reports that are handed in after the deadline are read at the earliest convenience

- **Rules:** Please read them (linked from the lab pages)

- Choose a lab partner and sign up for the labs through webreg
  - www.ida.liu.se/webreg (or follow the link from the lab web page)

- **Deadline for registration:** January 31
  - do it as soon as possible
1. SystemC modeling and simulation
2. Design space exploration

- Each lab has a tutorial. Please read it and be prepared before you attend the lab session. We want to work efficiently during the supervised lab sessions.
Examination

- Written report for each lab (pdf)
  - Present your solution to the lab exercises
  - Explain your design and implementation choice in detail
  - Present and discuss your results and simulations

- Prepare an archive with the report (pdf) and the code (where any) and email it to your group’s assistant (CC both lab partners)

- Grades:
  - Passed
  - Intermediate: revise your solution and report according to your assistant’s comments
Lab 1: SystemC modeling and simulation
Simulation

- Based on executable models of the system
- Generate input stimuli
- Permits a quick and shallow evaluation of the design quality
- Good for finding bugs
- Not suitable for finding subtle errors
- Compare to VHDL and Verilog
  - Contains structures for modeling HW components and their interaction
  - Comes with a simulation kernel
  - It’s a unified HW-SW design language

- What do we need to model systems?
  - time
  - modules
  - concurrent processes
  - events
  - channels
  - ports
Data type `sc_time` (a C++ class)

Use like an ordinary basic C++ data type (`int`, `double`)

- `sc_time t1(9, SC_MS);`
- `sc_time t2 = sc_time(5, SC_SEC);`
- `if (t1<t2) cout << t1*3 << endl << t2+t2;`
- Many of the standard operators are defined for `sc_time`

The underlying representation is based on 64 bits unsigned integer values

- The representable time is limited (discrete time)
- Depends on the time resolution
  - 1 picosecond
  - Can be set by the user through the function `sc_set_time_resolution`
SystemC - Modules

Modules:
- Basic building blocks in SystemC
  - Contains ports, concurrent processes, internal data structures, channels, etc.
- Created with the macro `SC_MODULE`
- Concurrent processes (`SC_THREAD` or `SC_METHOD`)
  - Use `wait` statements to advance time (or event notification)
  - Sensitive to events (`sc_event`) or value changes in channels
- Input and output ports to communicate with the environment
Example: Adder

Adder

a

b

sum
# SystemC module example

```cpp
#include <systemc.h>
#include <iostream>
using std::cout;
using std::endl;

SC_MODULE(Adder) {
    sc_in<int> a_p;
    sc_in<int> b_p;
    sc_out<int> sum_p;
    sc_event print_ev;

    void add_method() {
        sum_p = a_p + b_p;
        print_ev.notify(SC_ZERO_TIME);
    }

    void print_method() {
        cout << sc_time_stamp() << " :Sum=" << sum_p << endl;
    }

    SC_CTOR(Adder) {
        sum_p.initialize(0);
        SC_METHOD(add_method);
        sensitive << a_p << b_p;
        SC_METHOD(print_method);
        dont_initialize();
        sensitive << print_ev;
    }

}; // END Adder
```
Generate inputs

Generator → Adder

a

b

sum
// Definition of an input generator (next slide)

```c
int sc_main(int argc, char *argv[]) {
    sc_set_default_time_unit(1,SC_SEC);
    sc_signal<int> a_sig, b_sig, sum_sig; // create channels
    Adder adder_module("Adder_1"); // create an instance
    adder_module(a_sig, b_sig, sum_sig); // connect ports to channels
    Generator gen("Generator_1");
    gen(a_sig, b_sig);
    sc_start(30,SC_SEC);
    return 0;
}
```
SC_MODULE(Generator) {
    sc_out<int> a_p;
    sc_out<int> b_p;
    void gen_thread() {
        for (;;) {
            wait(1,SC_SEC);
            a_p = a_p + 1;
            b_p->write(b_p->read() + 1);
        }
    }
}

SCCTOR(Generator) {
    a_p.initialize(0);
    b_p.initialize(0);
    SC_THREAD(gen_thread);
}

}; // END Generator
Simulation run

adrli@mina2 ~TDTS30/doc/systemc_lab/examples/adder$ ./adder.x

SystemC 2.1.v1 --- Dec 22 2014 16:12:32
Copyright (c) 1996-2005 by all Contributors
   ALL RIGHTS RESERVED

 0 s: Sum=0
 1 s: Sum=2
 2 s: Sum=4
 3 s: Sum=6
 4 s: Sum=8
 5 s: Sum=10
 6 s: Sum=12
 7 s: Sum=14
 8 s: Sum=16
 9 s: Sum=18
10 s: Sum=20
11 s: Sum=22
   .
   .
   .
   .


Simulator kernel

1. **Initialize**: each process executed once; it’s possible to disable this phase for methods
2. **Evaluate**: select a ready to run process and execute/resume it; immediate notification may happen – e.notify()
3. repeat 2 until no more processes to run
4. **Update**: values assigned to channels in the previous evaluate cycle
5. stept 2-4 = \textit{delta-cycle}; if 2 or 3 resulted in delta event notifications (e.notify(0) or wait(0)) go to 2 without advancing simulation time
6. Advance to next simulation time with pending events
   Determine processes ready to run and go to 2
//inside a process
sc_signal<int> sig_int;
//assume current value
//of sig_int is 0
sig_int.write(1);
int value =
    sig_int.read();
cout << value << endl;  ←→ 0
wait(SC_ZERO_TIME);
value = sig_int.read();
cout << value << endl;  ←→ 1
Try the example

- You can find the example at:
  - /home/TDTS07/tutorial/systemc/adder
  - Copy it to your home directory
- Two files:
  - adder.cc (implements the two modules + the test bench)
  - Makefile (helps you compile and build the program)
    - Type `gmake` at the command line assuming you are in the correct directory
    - Creates an executable `adder.x`
- After building the example, type `./adder.x` to run it
- Study the source code together with the tutorial
Lab assignment

- Study the lab material linked from the course web pages
- At the end of the document you find the lab assignment
  - Design and implement a traffic light controller
Next Time
Lab 2: Design space exploration in MPARM
Thank you! Questions?