TDTS30: Laboratory introduction
(Lab 1 and 2)

Soheil Samii
Embedded Systems Laboratory
Dept. of Computer and Information Science
Linköpings universitet

www.ida.liu.se/~sohsa
sohsa@ida.liu.se
Building B, Room 329:202

Outline

§ First some (boring) practical issues
Then:
§ Short introduction
  § Validation
  § Verification
§ Lab 1
  § The HW/SW modeling language “SystemC”
§ Lab 2
  § Modeling and verification in UPPAAL
    § Timed automata
    § Temporal logics
§ Next lesson: Lab 3

(feel free to interrupt and ask questions at any time during the lesson)
Course organization

- Written examination 1.5 p (Petru Eles)
- Laboratory 2 p (Soheil Samii)
- Topic report 1.5 p (Daniel Karlsson)

Lab information

- Assistant: Soheil Samii
- Web page
  - http://www.ida.liu.se/~TDTS30
  - Check the lab pages for information and links to tutorials
- Organization
  - 2 lessons (including this one) (LE in the schedule)
  - 10 two-hour lab sessions (LA in the schedule)
- Consists of 3 labs (one tutorial for each)
  1. SystemC modeling and simulation (new)
  2. Formal verification
  3. Design space exploration
- Deadline: 2 April 2007
Lab registration

§ 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: 25 January 2007
  § do it as soon as possible

Structure

§ SystemC modeling and simulation
  § System-level modeling in SystemC
  § Simulation – checking properties
§ Formal verification
  § Modeling systems in timed automata
  § Simulation and formal verification in UPPAAL
§ Design space exploration
  § MPARM simulator (which is actually written in SystemC)
  § Simulation-based design space exploration
  § Communication in multiprocessor systems
  § Mapping and scheduling

*Each lab has a tutorial. Please read it before the lab session in order to maximize your efficiency in the lab room.*
Reporting

- Demonstration to the lab assistant
  - During lab sessions
- Written report for each lab
  - Present your solution to the lab exercises
  - Explain your design and implementation choices in detail
  - Present your results and discuss your conclusions
- Handing in the report (enclosed in a lab cover). 3 alternatives listed in order of preference:
  - Hand in at a lab session
  - Put in the IN box outside Soheil’s office
  - Put in the post room in front of Cafe Java
- Returned in the UT box outside Soheil’s office
  - You will get an e-mail when your report has been corrected and put in the UT box (assuming you’ve filled in your e-mail addresses on the lab cover)

Introduction

- Embedded systems
  - Composed of heterogeneous HW/SW elements
  - Dedicated functionality
  - Often characterized by real-time behavior
  - High requirements in terms of reliability and correctness
- Need models and methods to design and simulate these systems
- Need models and methods to verify the correctness of embedded systems
Incidents and accidents

- Mistype of one-letter command to the FMS (Flight Management System) led to the crash of a B757 aircraft in Cali in 1995, killing 159 people
- Malfunction of the Therac-25, a radiation machine to treat cancer, maimed and killed several patients
- Pentium floating-point divide bug caused loss of millions and credibility to Intel
- The maiden flight of the Ariane 5 launcher ended in a failure

System validation/verification

- Find out whether the system works according to its specification
- Validation
  - Simulation
  - Testing
- Verification
  - Formal methods
  - Model checking
  - Theorem proving
Validation

§ 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

§ Testing
  § Based on the real implementation of the system
  § Generate input stimuli
  § Very important phase of the design cycle
  § Exercise the system and analyze its response to check whether it functions correctly

Formal methods

For the levels of complexity typical to embedded systems
§ Traditional validation techniques cover a small fraction of the system behavior
§ Bugs found late have a negative impact on time-to-market
§ A failure may lead to a catastrophe
Formal methods (cont’d)

Formal methods can:
§ Overcome some of the limitations of traditional techniques
§ Give a better understanding of the system
§ Help to uncover ambiguities
§ Reveal new insights of the system

Formal methods do have limitations and are to complement, rather than replace, simulation and testing.

Lab 1: SystemC modeling and simulation
A system-level design language

- Designers can use the same language to develop both HW and SW
- Possible to do at different levels of abstraction
  - system-level
  - behavioral
  - structural (RTL,...)

Compare to VHDL, Verilog (hardware description languages)

- Comes with a simulation kernel (discrete-event simulation)
- Designers can evaluate their implementations through simulations

Strictly: not a language but rather a C++ library adding structures needed to model HW and concurrent systems
- Time, modules, processes, events, channels, ports, etc.

**SystemC - Time**

- New 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

§ Basic building blocks in SystemC
  § Contains ports, concurrent processes, internal data structures, channels, etc.
  § Created with the macro `SC_MODULE`
  § Can have 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

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
```
SystemC – Test bench

// Definition of an input generator (next slide)

```c
int sc_main(int argc, char *argv[]) {
    sc_set_default_time_unit(1, SC_SEC);
    sc_set_time_resolution(1, SC_SEC); // must be a power of ten!
    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;
}
```

SystemC – Input generator

```c
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);
        }
    }
    SC_CTOR(Generator) {
        a_p.initialize(0);
        b_p.initialize(0);
        SC_THREAD(gen_thread);
    }
}
```
### Simulation run

```plaintext
sohsa@mina2 ~TDTS30/doc/systemc_lab/examples/adder$ ./adder.x

SystemC 2.1.v1 --- Dec 22 2006 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
...
```

### Try the example

- You can find the example at: `/home/TDTS30/doc/systemc_lab/examples/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 `make` 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 will find the lab assignments as well as the requirements on the deliverables
- Basically:
  - 2 lab assignments
    - Introductory assignment to get familiar with SystemC
    - Design and implement a traffic light controller
  - Write a report
- For details:
  - SystemC Language Reference Manual (LRM), linked from the web page
  - Look into the SystemC source code
    - /home/TDTS30/sw/systemc-2.1.v1/src/

Lab 2: Formal verification in UPPAAL
UPPAAL

- Tool used for modeling, validation, and verification of real-time systems
  - Validation via simulation
  - Verification via model checking
- Developed jointly by Uppsala and Aalborg University
- Systems modeled using timed automata
- Properties to verify specified in CTL
- User-friendly graphical user interface
- www.uppaal.com

Timed automata

- A timed automaton is a finite automaton augmented with a finite set of real-valued variables called clocks (integer values in UPPAAL)
- All the clocks change along the time with the same constant rate
- Timing constraints can be expressed imposing conditions over clocks
- The timed automata model consists of a collection of automata which operate and coordinate with each other through shared variables and synchronization labels
A little more formal:

- A TA is a tuple \((L, L_0, E, \text{Label}, C, \text{clocks}, \text{guard}, \text{inv})\)
- \(L\), a finite set of locations with initial location \(L_0\)
- \(E\) is a set of edges (pairs of elements in \(L\))
- \(\text{Label}\), a function that assign to each location a set of atomic propositions
- \(C\), finite set of clocks
- \(\text{clocks}\), a function that assign to each edge a set of clocks
- \(\text{guard}\), a function that labels each edge with a clock constraint
- \(\text{inv}\), a function that assign to each location an invariant

Timed automata (syntax)

Clocks: \(X, Y\)

- \(X \leq 5 \land Y > 3\) Guard = clock constraint
- \(X = 0\) Reset (action performed on clocks)
- \(Y \leq 2\) Invariant
Timed automata (example)

- a1 \( y=1 \) a2
- b1 \( y=2 \) b2
- a3 \( c_a \leq 3 \)
- b3

Model checking

System Description
Automata model \( N \)

Specification (Req. Properties)
Temporal Logic formula \( f \)

\( AG \neg (p_1 \land p_2) \)
\( EF_{c_2} p_e \)

Model Checker
\( N \models f \)

yes

???

no

Diagnostic Information
Temporal logics

How do we specify properties for timed systems?

- Logic augmented with temporal model operators
- Used to specify desired properties of timed systems
  - Safety: Nothing bad will ever happen
  - Liveness: Something good may eventually happen
  - Bounded-response: Something will happen within a time limit
- CTL: Computation Tree Logic
  - Atomic propositions (states in the TA) and boolean connectors
  - Temporal operators:
    - Path quantifier (A, E)
    - Forward-time (G, F, X, U, R)

Computation tree

- Build (infinite) computation trees from the TA model
**CTL temporal operators**

**Intuition:**
- **AG** \( p \) Invariant
- **EG** \( p \) Potentially global
- **EF** \( p \) Possible
- **AF** \( p \) Inevitable
Two properties of the system

- Possible error: \( \text{EF Error} \)
- Possibly globally OK: \( \text{EF OK} \)
- These constitute the specification of the system

---

UPPAAL has a special syntax for CTL

- \( \text{AF} = A<> \)
- \( \text{AG} = A[] \)
- \( \text{EF} = E<> \)
- \( \text{EG} = E[] \)
- Boolean connectives: and, or, not, imply
- No nested formulas, except:
  - \( A[] (p \text{ imply } A<> q) = p \implies q \)
Lab assignment

§ Study the lab material (available very soon) linked from the course web pages
§ In the document you will find the lab assignments as well as the requirements on the deliverables
§ Basically:
  § Introductory assignments to get familiar with timed automata, CTL, and UPPAAL
  § Model the traffic light controller in timed automata
    § Simulate and verify.
  § Implement a communication protocol
    § Alternating Bit Protocol
  § Write a report