MOSIG DEMIPS/ Embedded Systems

Florence Maraninchi and Barbara Jobstmann

Verimag / Grenoble INP

2011-2012

Teachers and Organization

- Florence.Maraninchi@imag.fr
- Barbara.Jobstmann@imag.fr

Organization, Online Resources

A single place for all documents and slides:
`http://ensiwiki.ensimag.fr/index.php/`
`MOSIG_2_-_Option_DEMIPS_-_UE_Embedded_Systems`

Or, go to http://ensiwiki.ensimag.fr/, and search for DEMIPS.
The page of the course should appear in the list of documents found.

Or...

google "`maraninchi mosig`".

Books?

There's no book that could cover the full range of subjects we will be looking at during the course.
On Barbara's part (model-checking), there will be a book.
On Florence's part (general introduction and notions, abstract interpretation), there will be papers.

VERIMAG Lab

www-verimag.imag.fr

Embedded Systems
(Domain-Specific) Languages, validation methods (automatic test, formal verification), development methods, model-driven design, modeling, components, security, formal models, etc.

Application Domains: embedded control systems, systems-on-a-chip, sensor networks, distributed systems, middleware, component-based systems, robotics, ...

Contents of the course (2011-2012 Edition)

- Formal models that can be used to represent the concurrent and timed aspects of modern computer systems and how they can be used for formal validation
- Modeling principles; synchronous models, asynchronous models; how to use these models to build a programming language; introduction to formal verification
- Principles and applications of model-checking; principles and applications of abstract-interpretation

Prerequisites

- Programming with an imperative language,
- Parallel programming, at least one style
- Basic synchronous circuits,
- Operating systems,
- Low-level software,
- Automata and formal languages

Calendar

See webpage (ensiwiki)

Evaluation

- Barbara's part of the course contains practical exercises (P)
- There is a 3h written exam for the first session (E), about both Florence and Barbara's parts
- The mark for the first session is E, *modified by P*
- There is a 2h written exam for the second session (E'), on both parts
- The mark for the second session is E' (we forget about P)

Part I

General Introduction to Embedded Systems

Outline

Embedded Systems: Computer Systems in Everyday-Life Objects

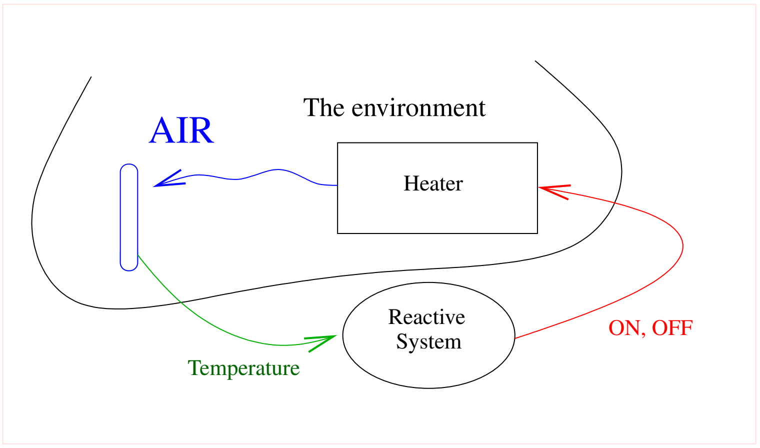
- Smart buildings and Energy
- Trains, subways, cars ...
- Consumer electronics (phones, digital cameras, ...)
- Telecom equipments
- Smart cards
- Computer Assisted Surgery
- Avionics and space

Ex 1: Embedded Control

In trains, cars, aircraft, space objects, nuclear power plants, ...
Systems: ABS, fly-by-wire, automatic flights, security control, ...

- The environment is a physical system, not a human being
- There are quite strong real-time constraints
- They are safety-critical systems
- The computer system is the implementation of a control engineering solution
- The computer system is reactive

Embedded Control - Reactive Systems



Embedded Control - Real-Time Systems

A typical real-time program:

```
initializations
while (true) {
      --- point (1)
      get inputs
           from the sensors
      compute outputs
           and update memory
      write outputs
           on the actuators
      --- point (2)
}
```

The time it takes to execute the code between points (1) and (2) defines the sampling period of the program. This is real time.

The outputs to the environment may have some influence on future inputs. This is reactivity.

Real-Time Programming Problems

- Write code that is sufficiently fast (you're not always allowed to answer: "*try a bigger machine*")
- Be able to tell how fast your program is, *in advance* (Worst-Case-Execution-Time static evaluation)
- It's not always possible to write single-loop code, because of the *intrinsic parallelism* of a reactive system.
  e.g., between two independent sensor-computing-actuator lines

Embedded Control - Safety

Criticity:
A fault is very “expensive”
(human lives, damage to the environment, ...).

HW Fault-tolerance:
Examples: several sensors of the same kind, plus a voter, several copies of the same code, running on several processors
SW Fault-tolerance:
Example: Several distinct versions developed independently from the same specification.

Embedded Control - Centralized or Distributed Systems?

Fact: centralized systems are far simpler than distributed ones.

But:

- Fault-tolerance forces distribution
- Physical position of sensors and computers also forces distribution
- Efficiency sometimes requires distribution

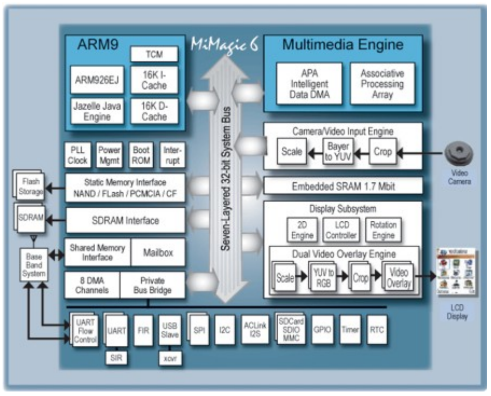
Another fact: distributed real-time programming is very hard.

Ex 2: Consumer Electronics

Digital cameras, set-top boxes, mobile phones, internet tablets, all kinds of portable devices...

- The environment is: a physical system (radio link) + a human being
- There are real-time constraints on the radio part
- They are business-critical systems
- The memory capacity and the processor speed are limited, the size is important, energy consumption is a very important constraint
- The hardware architecture is complex, and dedicated to the device (several processors, a DMA, a MPEG decoder, buses, radio components, ...) and the software is very hard to build.

Consumer Electronics - Systems-on-a-Chip



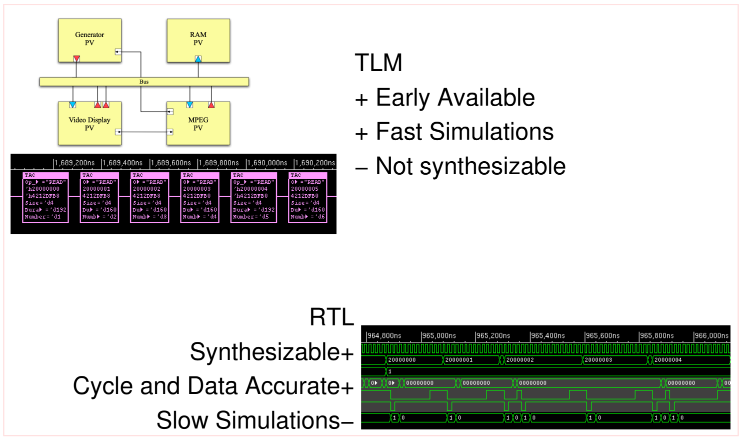
A complete computer system... on a single chip.
Several HW elements, embedded SW on the computing units, (RT)OS on some of the computing units, ...

Systems-on-a-Chip: SW Development Virtual Prototyping

The SW developers should start developing SW for a particular HW platform, long before the circuit corresponding to this HW platform is available.

Solution: virtual prototyping = write a program (SW) able to simulate the behaviours of the HW platform, w.r.t. the SW.

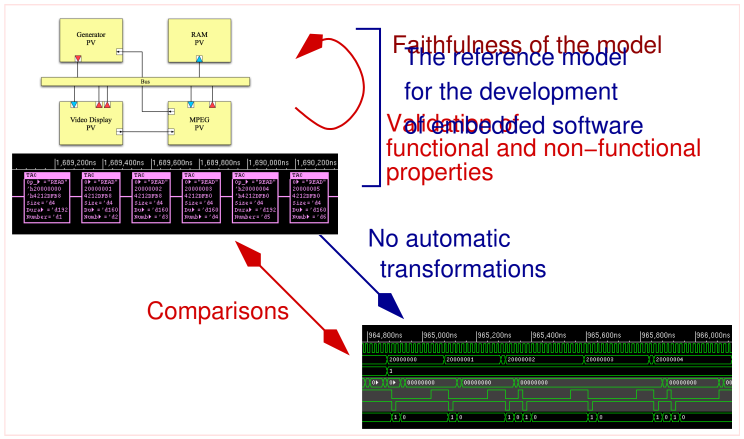
Systems-on-a-Chip: Virtual Prototypes



TLM
+ Early Available
+ Fast Simulations
– Not synthesizable

RTL
Synthesizable+
Cycle and Data Accurate+
Slow Simulations–

Systems-on-a-Chip: SW Development



Systems-on-a-Chip: Summary

Main difficulties:

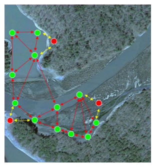
- Choose the right abstraction of the HW behavior
- Ensure that the SW developed on the virtual prototype will work, unchanged, on the real HW
- Define several abstractions, depending on the use (functional validation, time performances, energy consumption, ...)

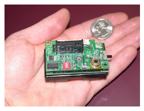
Example: HW and Drivers USB 3.0 Verification Techniques: Testing USB at the System Level

Recent announce for a seminar:
"As designer engineers work to integrate USB 3.0 into their devices, they will have to be cognizant of a number of complex issues. SuperSpeed links introduce 21 new state machines to USB operation - many of which rely on timers for entrance and exit. In addition, consumer devices often require seamless translation into other technologies such as, PCI Express, SATA, Fibre Channel and DDR3. And in mobile applications, vBUS power draw can limit the design's performance."

Ex 3: Sensor Networks

Environment monitoring, logistics, ...



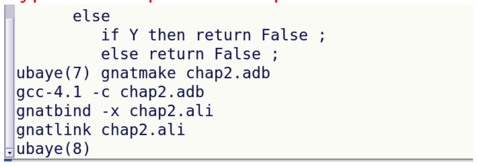


- The environment is: a physical system (radio link + physical inputs on the sensors)
- The memory capacity and the processor speed are very limited, energy consumption is THE key point
- The hardware architecture of a node is quite simple
- The software (MAC and routing protocols, application code) is crucial for energy consumption

The main problem is cross-layer design.

Transformational Systems

Typical example : a compiler



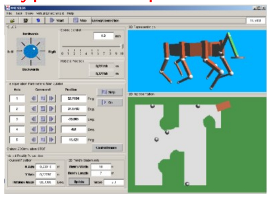
```
      else
         if Y then return False ;
         else return False ;
ubaye(7) gnatmake chap2.adb
gcc-4.1 -c chap2.adb
gnatbind -x chap2.ali
gnatlink chap2.ali
ubaye(8)
```

Inputs at the beginning, then some finite time computation, outputs at the end.

A transformational system has to terminate.

Interactive Systems

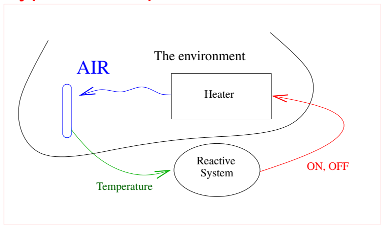
Typical example: a man-machine interface



loop-based behavior (does not necessarily terminate), where inputs come all the time (human actions on buttons, mouse, keyboard) and outputs are produced all the time also (changes of the interface, effects on the underlying computer system).

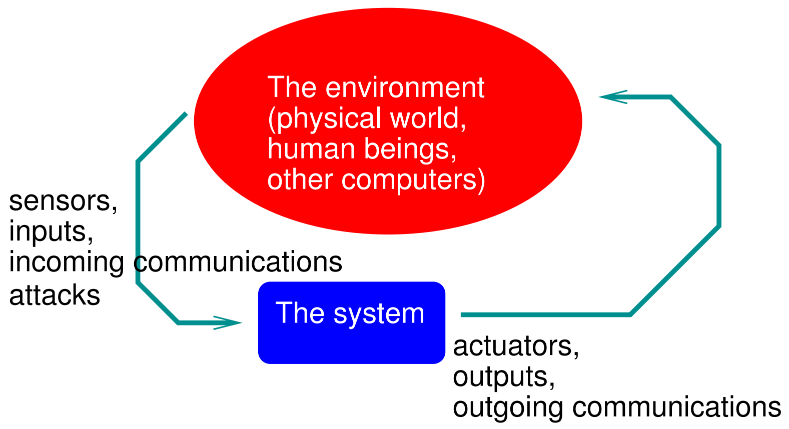
Reactive Systems

Typical example: a heater controller.



The same as interactive systems, but the speed of the interaction is driven by the (physical) environment. The computer system should be sufficiently fast in order not to miss relevant evolutions of the environment.

External View (1)


The environment (physical world, human beings, other computers)
sensors, inputs, incoming communications attacks
The system
actuators, outputs, outgoing communications

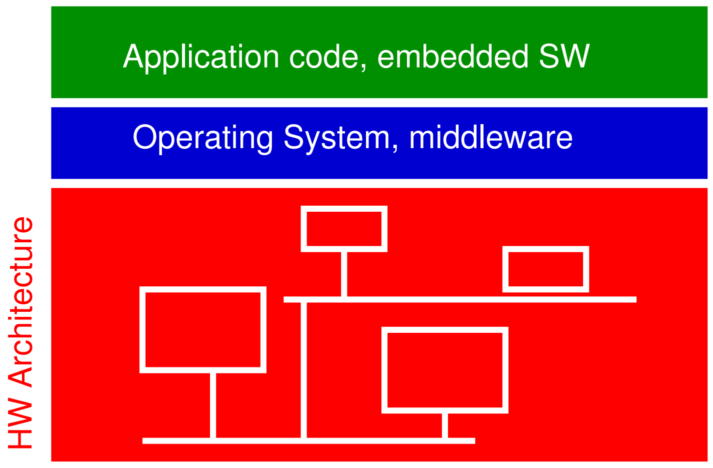
External View (2)

A Communicating Embedded Application is essentially reactive.

Externally observable properties:

- Correctness (functional property), or...
- ... Failure rate (functional property)
- Power consumption (non-functional property)
- Time (functional or non functional ?)
- Resistence to attacks (functional or non functional ?)

Internal View



Constraints

- (very) Scarce resources (memory, CPU, energy, ...)
- Real-time constraints and reactivity
- critical contexts of use (human lives, environment, business, ...) that imply strong and "in advance" validation methods for functional properties
- Importance of power Consumption and other extra-functional properties
- Fault-tolerance

Main Difficulties for the Design of Embedded Systems

- Real-time parallel and distributed programming (choice of a programming language?)
- Relation with control engineering
- Intricate dependency between HW, application SW, and OS or middleware
- Certification authorities
- Several degrees of dynamicity (from simple reconfigurations to mobile code...)

General-Purpose or Domain-Specific Languages?

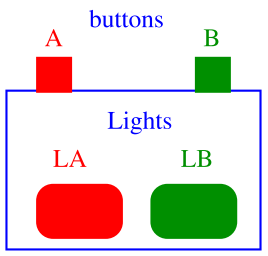
Ada, Java for real-time programming?

A DSL (Domain-Specific Languages) may have specific operations dedicated to reading sensors, writing actuators, synchronizing on time, etc., but it also has less than general-purpose languages.

Example

A programming language for embedded systems should not allow to write programs for which the memory is not statically bounded (implies: no recursion, no dynamic allocation)

A Special Note on Parallel Systems (1)



Behavior to be programmed:
— Each time A is pressed, toggle light A
— Each time B is pressed, toggle light B

How to program such a system?

A Special Note on Parallel Systems (2)

```
StateA := OFF ;
while (true) {
    read button A
    if (buttonA) {
        StateA := not StateA ;
    }
    if (StateA) {
        LightA.ON
    }
    else {
        LightA.OFF
    }}
```

```
StateB := OFF ;
while (true) {
    read button B
    if (buttonB) {
        StateB := not StateB ;
    }
    if (StateB) {
        LightB.ON
    }
    else {
        LightB.OFF
    }}
```

This is a solution for (one button, one light). How to describe two of them in parallel?

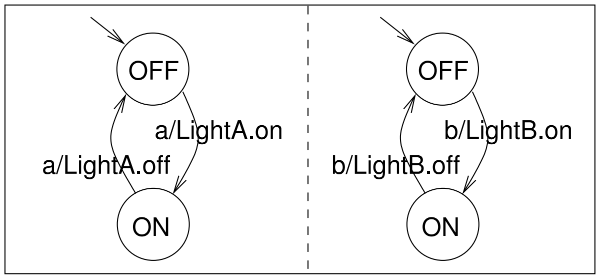
A Special Note on Parallel Systems (3)

```
StateB := OFF ; StateA := OFF ;
while (true) {
    read button B ; read button A ;
    if (buttonB) { StateB := not StateB ; }
    if (buttonA) { StateA := not StateA ; }
    if (StateB)  { LightB.ON } else { LightB.OFF }
    if (StateA)  { LightA.ON } else { LightA.OFF }
}
```

A solution with static scheduling: the code produced is sequential, but the high-level language may be parallel.

A Special Note on Parallel Systems (4)

A Solution in an Automaton-Based Language
(Statecharts, Argos, SCADE, EsterelStudio/SyncCharts, ...)

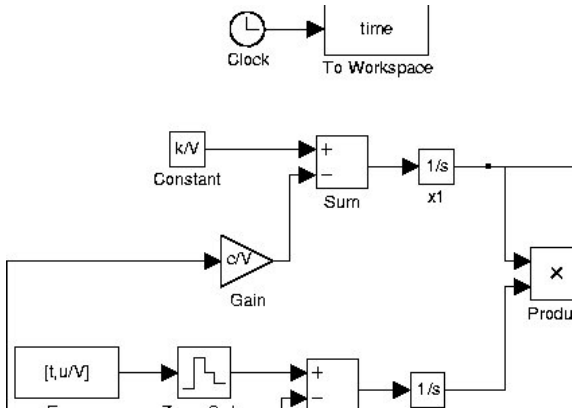


Validation and Certification

Validation:
Simulation, automatic testing, formal verification, ... are methods that help in analysing (functional) properties of a computer system before it is deployed.

Certification:
Examples: the DO178B norm for civil avionics, common criteria for smart cards, ...

A Simulink Diagram



Development from Simulink

- A continuous control problem and solution, including a model of the environment
- A discrete solution for the controller part
- An implementation. *Automatic code generation from Simulink?* or manual encoding, considering the diagrams as a detailed specification?

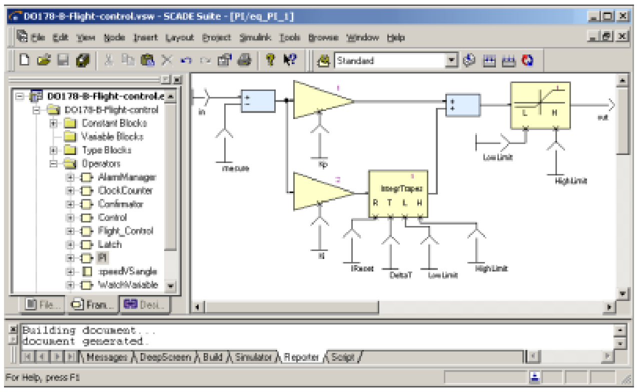
A complete chain from Simulink to embedded code is an instance of the general model-driven approaches.

Recommended readings - Lustre and SCADE

```
@article{ halbwachs91synchronous,
    author = "N. Halbwachs and P. Caspi and
              P. Raymond and D. Pilaud",
    title = "The synchronous data-flow
             programming language {LUSTRE}",
    journal = "Proceedings of the IEEE",
    volume = "79",
    number = "9",
    month = "September",
    year = "1991"}
```

www.esterel-technologies.com/products/scade-suite/overview.html

A Scade diagram

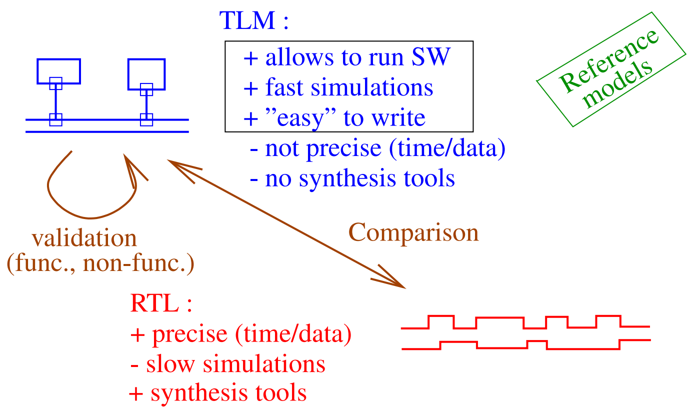


```
Building document ...
document generated
```

Main features

- A formal language
- Powerful bi-directional interface to requirements management tools like DOORS.
- KCG Code Generator qualification eliminates the need for low level testing.
- Verification tools
- *Import and reuse of Simulink block diagrams and Stateflow diagrams into SCADE.*

A new abstraction level: TLM, and SystemC



SystemC example

```
void module1::T1(){
  int a = 0;
  while(true){
   wait(e1);
   a++;e2.notify();
   a++;e3.notify();
}}
void module1::R1(){
  int b = 0;
  while(true){
    b++;wait(e2);
    b++;p2.f2(b);
}}
void module1::
            f1(int x){
   cout << x ;
   e1.notify();
   wait(e3);
}
```

```
void module2::T2(){
 int c;
 while(true){
   c++;   p1.f1(c);
   c++;   wait(e4);
 }
}

void module2::
           f2(int x){
   cout<< x ;
   e4.notify();
}
```

Summary: Programming or Modeling Languages

Software:
C, C++, SystemC, Java or RT Java, Ada, ...
Domain-Specific Languages (DSLs): Lustre/Scade, Simulink, ...

Hardware:
VHDL, Verilog, C, SystemC, ...

Summary: Criticity

Safety-critical systems (e.g., nuclear plants):
Design norms, certification authorities, ...

Business-critical systems (e.g., mobile phones):
Methods to shorten time-to-market (virtual prototyping)

Summary: Intrinsic Difficulties and Methods

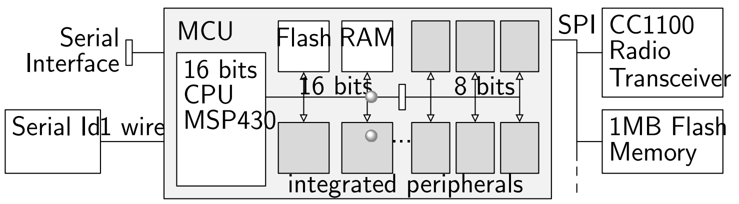
Reactivity, distribution, real-time, fault-tolerance, ...

Use of "*models*"

Motivations

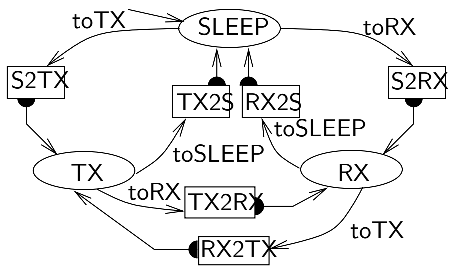
- The radio-chips in sensor nodes are tailored for low-consumption.
- The SW layers should exploit the low-power modes of the radio chip; this implies a quite intricate relation between the HW and the low-level SW.
- Ignoring the HW/SW interface leads to non-significant models (see later).
- Need for evaluating the trade-off between energy consumption and delivery rate or security, for new MAC or routing protocols

The Hardware Of A Node (WSN430)



See: perso.ens-lyon.fr/eric.fleury/Upload/wsn430-docbook/index.html

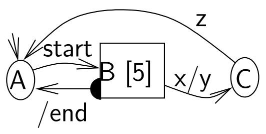
The Radio (Simplified)



Delays:
RX2S=TX2S=0.1 μs
S2RX=S2TX=88.4 μs
TX2RX=21.5 μs
RX2TX=9.6 μs

RX = receive; TX=transmit;
Square states: waiting for some delay;
toRX, toTX, ...: commands from the SW.

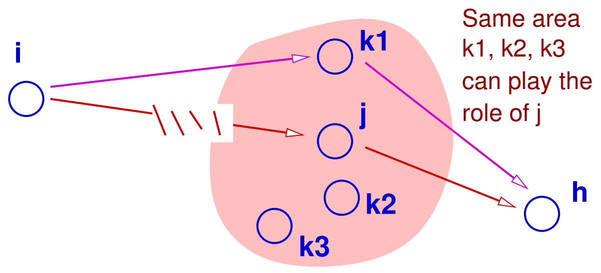
Elements of Syntax for Automata With Timed States (Synchronous Semantics)



- Circles: normal states
- Squares: timed states [delay]
- x/y: input/output

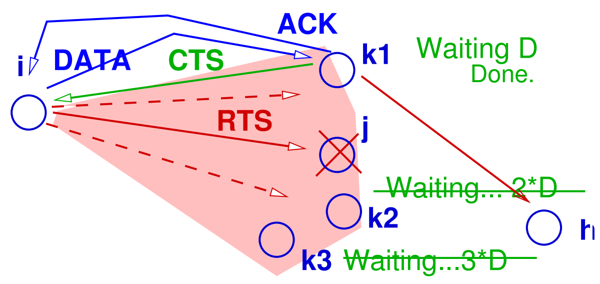
Synchronous semantics: on a discrete time line...

A MAC Protocol - Principles

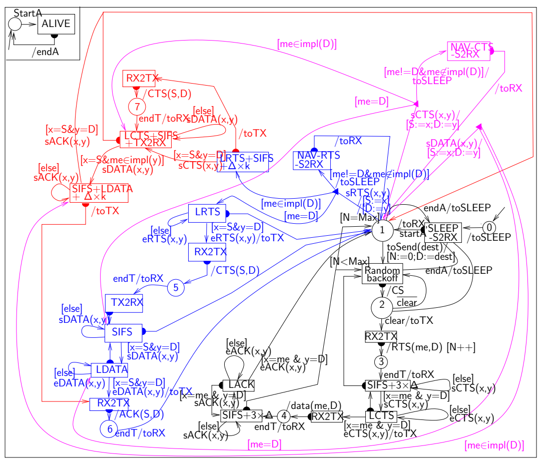


See: *AreaCast: a Cross-Layer Approach for a Communication by Area in Wireless Sensor Networks. K. Heurtefeux, F. Maraninchi, F. Valois. 17th IEEE Int. Conf. on Networks (ICON'11).*

AreaCast Dynamic Behaviour

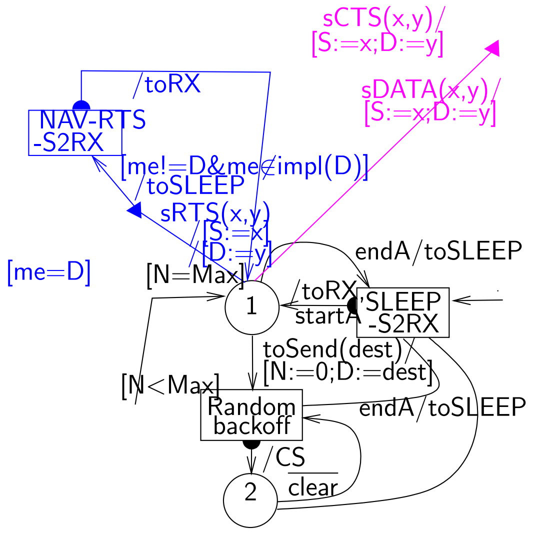


The Protocol - Algorithm

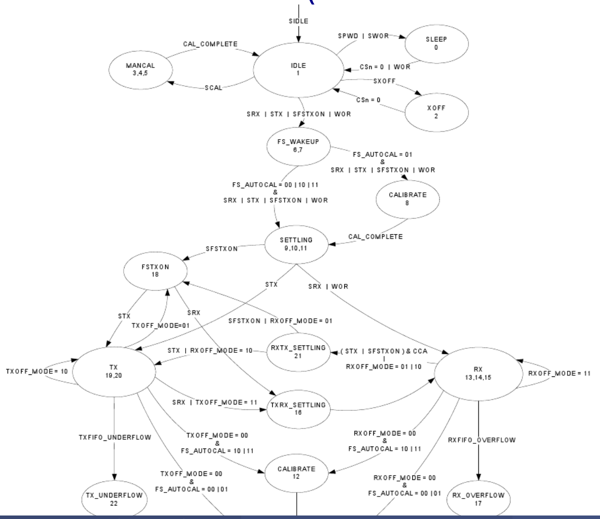


Mode changes issued to the radio: toRX, toTX, toSLEEP;

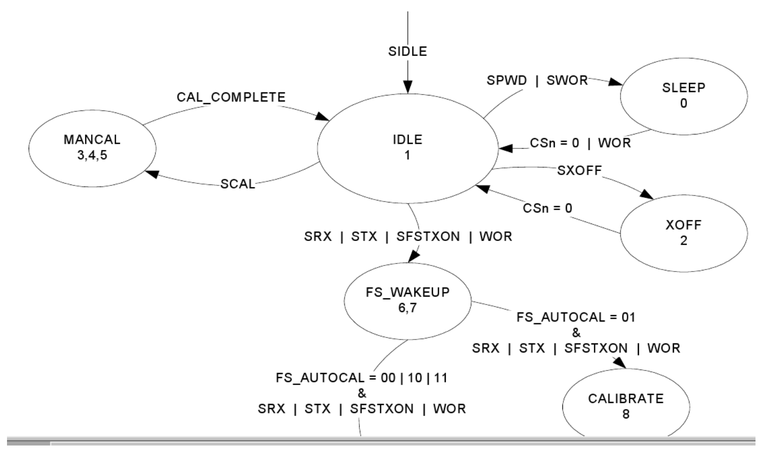
Timed states for counting delays D, 2*D, 3*D and estimated transmission times.



Power-State Models (Texas Instruments CC1100)



Power-State Models (Texas Instruments CC1100)

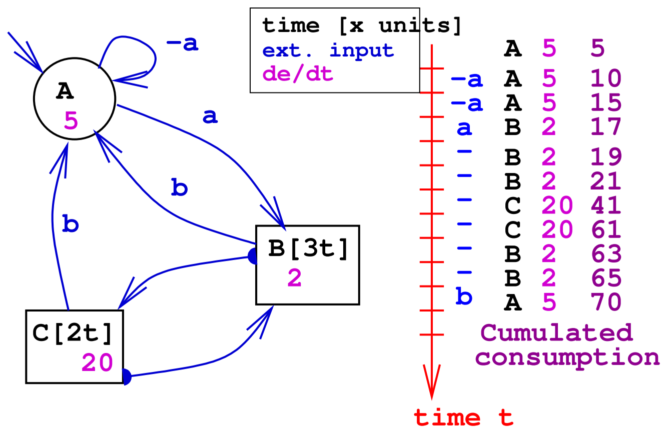

SIDLE
SPWD | SWOR
SLEEP 0
CAL_COMPLETE
MANCAL 3,4,5
IDLE 1
CSn = 0 | WOR
SXOFF
SCAL
CSn = 0
XOFF 2
SRX | STX | SFSTXON | WOR
FS_WAKEUP 6,7
FS_AUTOCAL = 01 & SRX | STX | SFSTXON | WOR
FS_AUTOCAL = 00 | 10 | 11 & SRX | STX | SFSTXON | WOR
CALIBRATE 8

Power-State Models

In a datasheet one can find:
— The automaton (power modes and transitions between them)
— Consumption per unit of time, in each mode
— Time and Consumption penalty for each transition

All this can be expressed by a consumption automaton.
= a discrete view of a linear-priced timed-automaton (LPTA),
or a simple case of hybrid automaton.

Power-State Models: Formal View

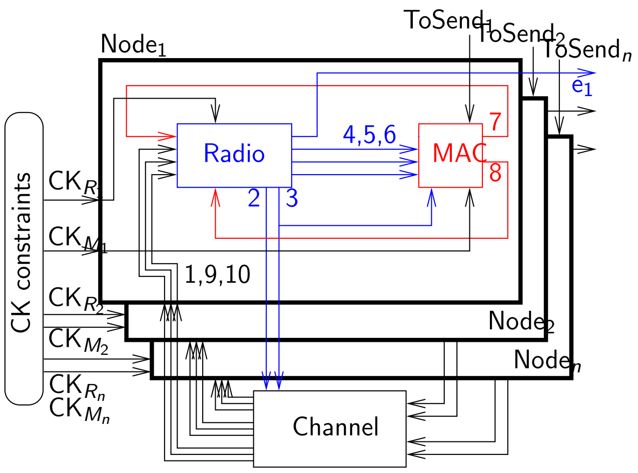


ext. input		de/dt	Cumulated consumption
	A	5	5
-a	A	5	10
-a	A	5	15
a	B	2	17
-	B	2	19
-	B	2	21
-	C	20	41
-	C	20	61
-	B	2	63
-	B	2	65
b	A	5	70

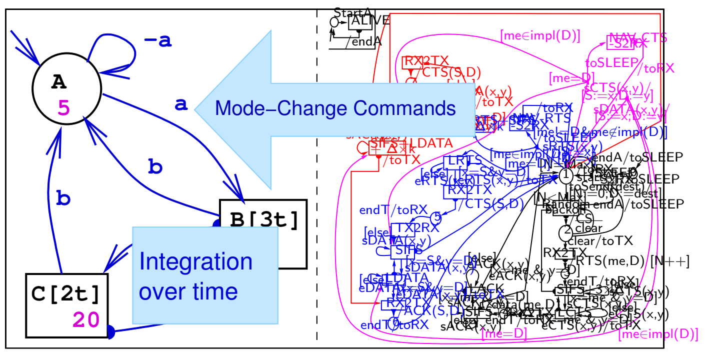
Remarks, Questions, and Discussion...

- The radio device is modeled by an automaton
- The MAC program is given as an automaton (nothing's missing, the automaton syntax is a programming language, you can get the actual running code from this automaton).
- What to do with these two objects? How to combine them?
- What does it mean to "run" the program together with the model of the radio? What can we observe?
- How do we know that the model is faithful to reality?

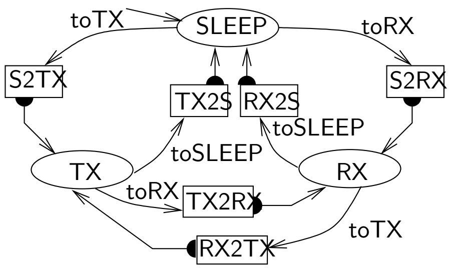
Structure of the Model



The HW/SW Interface



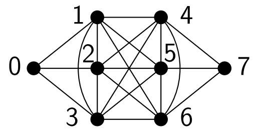
The Radio



Consumptions:
γ(SLEEP)=2
γ(RX)=15
γ(TX)=16
γ(S2TX)=γ(S2RX)=
γ(RX2S)=γ(RX2TX)=
γ(TX2RX)= γ(TX2S)=8

Delays:
RX2S=TX2S=0.1 μs
S2RX=S2TX=88.4 μs
TX2RX=21.5 μs
RX2TX=9.6 μs

A Small Network



Relation between nodes and their implicit relays:
impl(2)={1,3} ;
impl(5)={4,6}

Path requested by the routing level: $0 \to 2 \to 5 \to 7$.

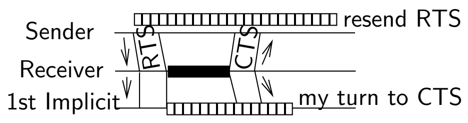
Simulation Results (1)

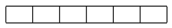
Path requested by the routing level: $0 \to 2 \to 5 \to 7$.
p is the probability for a link to fail.
e0...e7 are the consumptions of nodes 0...7.

	p = 0	p = 0.15	p = 0.15
	$0 \to 2 \to 5 \to 7$	$0 \to 1 \xrightarrow{2} 6 \to 7$	$0 \xrightarrow{2} 2 \to 5 \xrightarrow{5} 7$
e_0	2019	2274	5189
e_1	1831	2248	4436
e_2	1981	2332	4548
e_3	1831	2332	5452
e_4	1857	2145	4626
e_5	2007	2139	5136
e_6	1857	2081	4275
e_7	2032	2107	5227

Simulation Results (2)
Observing the Effect of Clock Drift

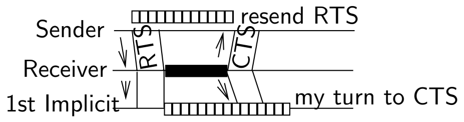
Ideal Behaviour when the receiver is functioning correctly, and the clocks are ok:



Notation:  *represents clocks ticks on one node.*

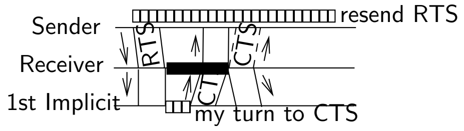
Simulation Results (2)
Observing the Effect of Clock Drift

The sender is too fast: it re-emits its RTS before having waited long enough for a likely arrival of the CTS:



Simulation Results (2)
Observing the Effect of Clock Drift

The first implicit node is faster than others: it decides to play the role of the legitimate node, before that one had a chance to react by a CTS:



Simple Models Help Understand the Complexity of Computer Systems

- Industrial practice for embedded applications : an ever growing wide variety of languages and models C/C++, SystemC, Ada, Simulink, Scade/Lustre, VHDL or Verilog, UML, SysML, AADL, IPXact, Java, ...
- How to understand the relationship between the two ?
- Computer science : a relatively stable and small set of well understood models of computation (for embedded systems: models of time and concurrency)

Modeling Concurrency

- Synchronous models are good at: modeling parallelism between processes that share a common clock; describing synchronous circuits; describing a "logical" parallelism that will be compiled into sequential code;
- Asynchronous models are good at: modeling parallelism when no common clock exists;
- GALS (Globally Asynchronous Locally Synchronous) or LAGS models can mix the two

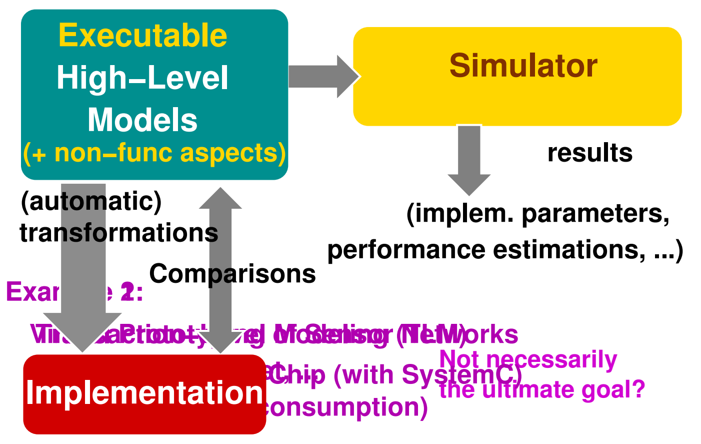
Modeling Time

- Synchronous models are intrinsically timed, because of the existence of a global clock
- Asynchronous models are intrinsically untimed; adding time to an asynchronous model requires ad-hoc modifications.

We will study...

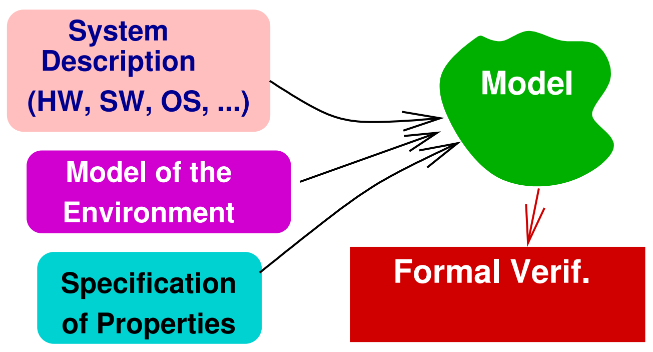
- Synchronous models, and how they can be turned into synchronous programming languages for real-time reactive systems
- Untimed asynchronous models
- How to add time to an asynchronous model (*you already know asynchronous parallel programming*)
- Systems to models to formal verification tools

Model-Based Impl. plus Virtual Prototyping



Executable High–Level Models (+ non–func aspects)
Simulator
results
(implem. parameters, performance estimations, ...)
(automatic) transformations
Comparisons
Implementation
Example 2:
Chip (with SystemC)
consumption)
Not necessarily the ultimate goal?

From Systems to Models to Formal Verification



System Description (HW, SW, OS, ...)
Model of the Environment
Specification of Properties
Model
Formal Verif.

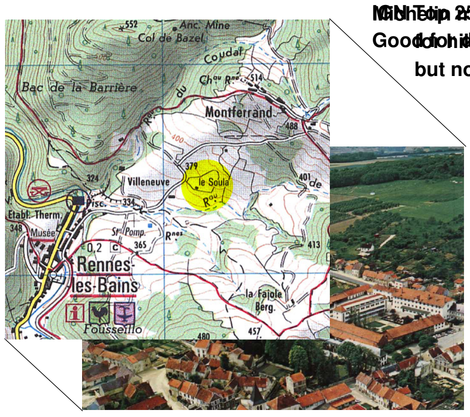
What is a Model?

????

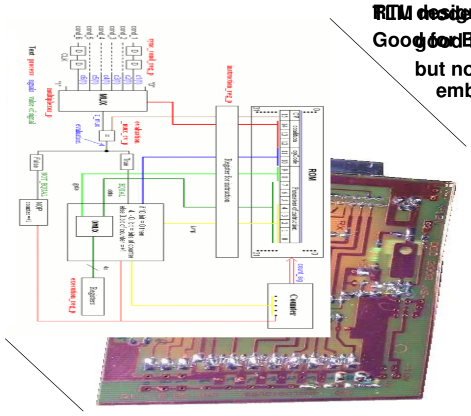
Models are Necessarily Abstract (see J.-L. Borges)

...In that Empire, the craft of Cartography attained such Perfection that the Map of a Single province covered the space of an entire City, and the Map of the Empire itself an entire Province. In the course of Time, these Extensive maps were found somehow wanting, and so the College of Cartographers evolved a Map of the Empire that was of the same Scale as the Empire and that coincided with it point for point. Less attentive to the Study of Cartography, succeeding Generations came to judge a map of such Magnitude cumbersome, and, not without Irreverence, they abandoned it to the Rigours of sun and Rain. In the western Deserts, tattered Fragments of the Map are still to be found, Sheltering an occasional Beast or beggar; in the whole Nation, no other relic is left of the Discipline of Geography.

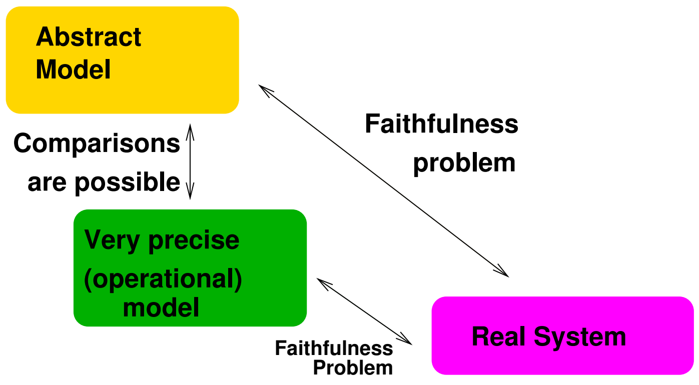
Abstractions are Useful (1)


IGN Top 25 = A Faithful model...
Michelin map, more abstract
Good for hiking
for driving or riding
but not so good for driving
The real system

Abstractions are Useful (2)



Abstraction and Faithfulness


Abstract Model
Comparisons are possible
Faithfulness problem
Very precise (operational) model
Faithfulness Problem
Real System