General Introduction to Embedded Systems: Characteristics and Constraints

Pascal Raymond
thanks to Florence Maraninchi

September 30, 2015

What is an Embedded System?

1. What is an Embedded System?
   - Some Examples
   - Classifying Computer Systems
   - Tentative Definition of Embedded Systems (Constraints and Difficulties)

2. Some Industrial Practices

3. Summary
What is an Embedded System?

Embedded Systems: minimal definition

- A computer system dedicated to a particular function
- system: not a software, not a hardware
- particular function: not (really/fully) programmable, does 1 thing

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
What is an Embedded System?

Some Examples

Classifying Computer Systems

Tentative Definition of Embedded Systems (Constraints and Difficulties)

---

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
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.

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.
What is an Embedded System?

1. Some Examples

Classifying Computer Systems

Tentative Definition of Embedded Systems (Constraints and Difficulties)

Transformational Systems

Typical example: a compiler

```plaintext
if Y then return False;
else return False;
```

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.
What is an Embedded System?

- Some Examples
- Classifying Computer Systems
- Tentative Definition of Embedded Systems (Constraints and Difficulties)

External View (1)

The environment (physical world, human beings, other computers)

The system

sensors, inputs, incoming communications, attacks

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 ?)*
- Resistance to attacks *(functional or non functional ?)*
**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...)
1 What is an Embedded System?

2 Some Industrial Practices
   - Simulink in the automotive industry
   - SCADE in the avionics industry
   - SystemC for Systems-on-a-Chip

3 Summary
Some Industrial Practices

Simulink in the automotive industry

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.
Some Industrial Practices

- Simulink in the automotive industry
- SCADE in the avionics industry
- SystemC for Systems-on-a-Chip

A Scade diagram
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.

Some Industrial Practices

- Simulink in the automotive industry
- SCADE in the avionics industry
- SystemC for Systems-on-a-Chip
SystemC: TLM vs RTL

Key idea:
Develop/validate the software before the actual hardware is available. How?

- Provide executable model of the hardware, very abstracted, as a C++ library
- This is the Transaction Level Model (TLM)
- Different from the Register Transfer Level

RTL:
Precise model of the hardware, automatic synthesis, but very slow simulation

TLM:
Abstracted model of the hardware interface, automatic synthesis impossible, but very fast simulation

SystemC/TLM development

Technically:
- SystemC = set of C++ libraries, provides the TLM level
- Application software: written in C
- Develop/debug/validate software at the TLM level
- When the hardware is available, just load the application software on it, it (hopefully) works
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: validity/correctness

**Functional:**
Computes "right", no bugs

**Extra-Functional:**
Meets time constraints, power consumption constraints

---

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

Summary: synonyms / related

- Reactive Systems
- Control Engineering (Systems)
- Cyberphysical Systems
- (Hard) REal Time Systems

This course

Focus on *functional aspects* of Embedded systems:

- How to design (as a whole) an ES? validate? implement?