Embedded systems are everywhere... 

... and control vital functions in our daily life!

Designers have large responsibility!

Between 1985-87 several deaths and serious injuries of cancer patients were due to overdoses of radiation resulting from a race condition between concurrent tasks in the Therac-25 software (1985-87).
How can we avoid future accidents?

Embedded systems
- take over an increasing number of vital functions in our society
- include more and more functionality

Increasingly complex design process
Disciplined methodology is needed to design predictable embedded systems!

Heterogeneous Nature of Embedded Systems
- An embedded system interacts with the physical world and other embedded components
- An embedded system architecture consists of heterogeneous components
Characteristics of Embedded Systems

An embedded system

- is usually designed for one single task. Its functionality will never change.
- is often a mass product. Design cost is critical.
- interacts with the environment at the speed of the environment. Many embedded systems are safety-critical systems and have to fulfill hard real-time requirements.
- is often a hand-held device. Power-efficiency is critical.
- is often a consumer products. Time-to-market is critical.

Design process for embedded systems is very different from general purpose programming!

- Embedded systems can be highly optimized
- All unneeded features are a disadvantage (cost, power)
- Design process must
  - be cost-efficient
  - ensure the correct functionality and timing of the implementation
  - be fast to ensure a short time-to-market

To be fast is not enough!

The system has to react to the environment at the right time instance, otherwise there can be fatal consequences.
How to design an embedded system?

The challenge

How to bridge the abstraction gap?

Top-Down Design Process

Models

The design process requires to express the system at different levels of abstraction. For each level a new model of the system is required.
**Full-custom design methodology**

- A top-down full-custom design methodology will use all levels of abstraction during the design process.
  - Maximal flexibility, all details can be fine-tuned.
  - Maximal performance can theoretically be achieved.

Design-time and thus time-to-market can be very (much too) long!
Platform: Not a new, but a successful concept

Platforms have been used in many different areas! How can we adapt the platform concept for **predictable embedded systems**?

**Reduce Design Time and Time-To-Market**

- The basic idea of platform-based design is to avoid to design a system from scratch.
- Platforms at different levels can be reused for different applications.

![Sales per Quarter Graph](image)

*Source: M. Smith, 1997*

**Time-to-market is critical for many products!**
An industrial platform: OMAP

- The Open Multimedia Application Platform (OMAP) is developed by Texas Instruments. Many mobile phones are using this platform.
- The OMAP 4 platform has been designed to drive smart phones and mobile internet devices (MIDs).

OMAP Hardware Platform Instance

OMAP is not only a hardware platform, but provides several layers of software, which together comprise the OMAP software platform.

OMAP Software Platform

Designer can work at a high-level of abstraction!
Alteras SOPC platform

Altera offers a platform to design a system-on-a-programmable-chip on an FPGA architecture around the Nios II soft processor.

- Designer selects hardware components (processor and peripherals) and specifies the interconnection.

Altera System Interconnect Fabric

Given the specification of the components and the interconnection structure the Altera software generates the System Interconnect Fabric.
Altera Hardware Abstraction Layer Library

In addition to the hardware architecture the Altera software performs an automatic generation of the Hardware Abstraction Layer software libraries, which abstract from the underlying hardware.

Raising the Level of Abstraction

The designer can now design at a higher-level of abstraction using more abstract functions and symbolic address names.

Trade-Off: Design Space vs. Time-to-Market

- A platform limits the design choices.
- The designer needs only to analyze the alternatives that are implemented by the platform.
- The platform can itself be configured to a certain degree.
Platform Benefits

In addition to a shorter time-to-market the platform concept gives more benefits:

- Reuse of platform reduces design costs.
- The platform can be highly optimized since the development costs are shared by several designs.
  - Library of software or IP-blocks
  - Tool support in form of compilers, verification tools, simulators
  - Full-custom design of critical platform components
- Other platforms can be developed on top of a platform.
  - Design entry can be moved to higher levels of abstraction
  - Development of synthesis tools to automatically refine a design from an abstract level to a more detailed level

Mapping of Function to Platform

In order to map a function onto a platform

- functional requirements need to be implemented
- non-functional requirements need to be fulfilled (cost, power, timing, . . . )

Accurate estimates must be provided by the platform!
Current Industrial Design Practice

- It is very difficult to accurately estimate the performance of an embedded system.
- Huge difference between average and worst case execution time

As a consequence

- Industry bases new designs rather on old experiences than on performance analysis
- Industry introduces sufficient safety margins in form of more powerful components and extra communication bandwidth
- Verification costs are extremely high!

Surely, there must be a better way to design systems...

A Dream: Correct-by-Construction Refinement

In an ideal world, the design process would be correct by construction.

- All refinements are correct
  - Functionality is preserved
  - Performance constraints are met

What is needed?

- A predictable platform that provides accurate performance estimates
- Verified transformation rules to map a design from a higher level of abstraction to a lower level of abstraction
Predictable platforms at all levels of abstraction

If the base platform lacks predictability, it is very difficult to build a predictable platform on top of it

Why is it so difficult to estimate software?

There is a huge difference between average and worst case response time

- Many architectures are designed for an optimal average performance!

Worst case performance is critical!

Optimizing average case performance is important for most software systems. However, in particular for safety-critical systems worst case performance is critical!
Caches can significantly improve average performance!

- Cache access time is much less than access time to main memory.

**BUT**

- Memory content is changed dynamically

⇒ Extremely difficult to predict, if data will be in the cache at a certain time instant.

**Very difficult to exploit caches in real-time systems!**

---

**Shared Memory Multiprocessor**

Execution time of programs in a shared memory multiprocessor can in general not be analyzed in isolation.

Execution time for a program on one processor depends on memory access patterns of other processors ($t_{pgm1,a} \leq t_{pgm1,b}$)!
Why can we so accurately predict hardware performance?

- Elegant synchronization mechanism: the hardware clock
  - All input events happen at the same time instance
    ⇒ Communication is predictable!
- Only requirement: all computations are finished within one clock period
  ⇒ Longest computation path determines clock period
- Since communication is synchronous, access to shared resources is predictable

Digital Hardware:
A Small Set of Components, Infinite Possibilities

We can build extremely complex designs with digital hardware!

The foundation for digital hardware is simple!

- All logic functions can be implemented using only NAND or NOR gates
- Digital hardware can be described by boolean equations

Abstract concepts like finite state machines can be directly mapped to hardware!

Keep predictable platforms simple!
Predictable Hardware Platform
Dedicated on-chip memory instead of caches

Memory contents in dedicated on-chip memory (scratchpad memory) is static.

⇒ Predictable access time to main memory
⇒ Predictable access time to on-chip memory

Challenge
Determine, which memory locations should be in scratchpad memory

Predictable Hardware platform
Predictable access to communication network

Give processors guaranteed access to buses or communication links by reserving time slots for each processor

- Possible to predict worst case bus access time for a processor

Challenge
Design of efficient predictable systems with restricted resources.
Towards predictable software

- Embedded systems are inherently parallel
  - Parallelizing a sequential program is very difficult
  - Software language needs to be able to express parallelism (explicit or implicit)
- Time is critical for many embedded systems
  - Time needs to be a first-class citizen
  - Communication should be deterministic

Learn from Hardware Design!

**Synchronous languages** (Esterel, Lustre, Signal, . . .)

- use an implicit synchronous clock
- have been shown very successful for safety-critical applications

**Challenge**

Development of high-level languages for embedded systems. Many approaches for parallel software exist. However, they are often based on an asynchronous communication mechanism and not defined formally.
Modeling at the Functional Level

In the beginning of the design process the functionality of the system has to be understood and captured.

- Decisions about what parts shall be implemented in hardware or software are not taken yet.

⇒ System shall be modeled at high abstraction level

- A system is usually described as heterogeneous concurrent process network
- In order to describe different domains, different modeling techniques are needed

Predictable Modeling Platform

- A model of computation (MoC) specifies the interaction between concurrent processes
  - Synchronous, Untimed, Continuous Time, Discrete Time
- Processes belonging to different models of computation communicate via domain interfaces

Challenge

It is of crucial importance that the semantics of each MoC is well-defined. A big challenge is a meaningful and precise definition of domain interfaces between different MoCs.
ForSyDe

ForSyDe (Formal System Design) is a design methodology for systems-on-chip, which allows to model heterogeneous systems.

- ForSyDe is implemented as domain specific language in Haskell.
- Several libraries for different models of computation exist and can be simulated as integrated model.
- ForSyDe processes are formally defined.
- ForSyDe supports modeling at different levels of abstraction.
- There exists a back-end for hardware design and synthesis (VHDL).
- High-level and synthesizable models can be co-simulated giving access to powerful test benches.

Function Space meets Platform Space

Example: Functional model uses synchronous model of computation

- Easy to map to
  - synchronous software language
  - synchronous hardware language
- Difficult to map to
  - asynchronous communication mechanism

Common semantic base is critical!
Industry?

Industry is in general very conservative

- Huge investments have been made into
  - education of engineers
  - tools
  - design flow
  - components
  - software

- Any change of design process is very costly and a big risk!

⇒ Evolution rather than Revolution . . .

Challenge

How can a new design paradigm find access into industry?

Industry Acceptance: 'Industrial Languages'

- The SYSMODEL project uses SystemC as modeling language.
  - Modeling guidelines are developed to make the SystemC model compliant to a formal model (ForSyDe)
  - Other languages can be imported by means of SystemC-wrappers.
  - The whole model can still be co-simulated.
The SYSMODEL project uses 'refinement-by-replacement'.

- A SystemC block has been refined and replaced by C-code that runs on an instruction set simulator, which belongs to the platform architecture framework.
- The whole model can be co-simulated.

⇒ Existing software and hardware components can be used.

Can platform-based design help us to design predictable systems?

- Platform-based design is a very promising approach to tackle the increasing complexity of embedded system design.
- Challenge is to
  - develop predictable platforms at all levels of abstraction
  - identify predictable mappings between platforms at different levels of abstractions
Can platform-based design help us to design predictable systems?

- Since design of embedded real-time systems is very different from general-purpose computing, we need to
  - rethink architecture of embedded systems
  - rethink software languages
  - introduce time (power) as first-class citizen from the start of the design process
- Formal methods can help a lot
  - Design process needs to be based on a formal foundation!

Platform-based design at KTH

- ForSyDe
  - supports several models of computation and allows co-simulation
  - tool for hardware synthesis (VHDL) exists
- Research on architectures with guaranteed quality of service
- Access to Altera FPGAs, which allow to design and implement predictable architectures

KTH is in an excellent position to take platform-based design for predictable systems a few steps further!

1. Development of a predictable hardware platform for embedded software
2. Mapping of low-level ForSyDe models to platform
Further Information

More information on ForSyDe
http://www.ict.kth.se/forsyde/

More information on the SYSMODEL project
http://www.sysmodel.eu/

More information on platform-based design

More information on predictable architectures

Thanks for your Attention!
Do you have any Questions?