Introduction
What is this?

- Hardware/Software codesign investigates the concurrent design of hardware and software components of complex electronic systems.
- It tries to exploit the synergy of hardware and software with the goal to optimize and/or satisfy design constraint such as cost, performance, and power of the final product.
- At the same time, it targets to reduce the time-to-market frame considerably.
What is this?

- Digital (Embedded) systems designs consists of hardware components and software programs that execute on the hardware platforms.

What is the definition of HSCD?

Meeting System level objectives by exploiting the synergism of hardware and software through their concurrent design.
Motivation

• Not possible to put everything in hardware due to limited resources
• Some code more appropriate for sequential implementation
• Desirable to allow for parallelization, serialization
• Possible to modify existing compilers to perform the task
History

- Emerged during 1990s as a design of complex integrated circuits (ICs)
- First research community is started in 1992
- International workshop on HSCD has started in 1992
- Now we are in the third generation of the HSCD

(For more details refer to the IEEE Paper:

- Jurgen Teich, Hardware/Software Codesign: The past, the Present and Predicting the Future, Proceedings of the IEEE. Vol 100, May 13, 2012)
Codesign Definition and Key Concepts

Codesign
– The meeting of system-level objectives by exploiting the trade-offs between hardware and software in a system through their concurrent design

Key concepts
– **Concurrent**: hardware and software developed at the same time on parallel paths
– **Integrated**: interaction between hardware and software developments to produce designs that meet performance criteria and functional specifications
Application areas

Automotive electronics:
Application areas

Aircraft Electronics
- Flight control systems
- Anti collision system
- Pilot information system

Trains:
- Situation is similar to the one defined for the cars and airplanes.
- Safety feature are very important

Telecommunication:
- Mobile phone
- RF design, DSP and Low power design are key aspects.
Application areas

**Medical Systems:**
- Pocket ECG Monitor
- Compact Ultrasound systems, etc

**Military Applications:**
- Radar signal processing
- Automated target detection

**Authentication systems:**
- Smart pens, fingerprint identification systems etc.
Application areas

Consumer Electronics:

Fabrication Equipment:
Application areas

Smart Buildings:

Production Systems:
Application areas

Robotics:
Common characteristics: Dependability

- **ES Must be dependable,**
  - **Reliability** $R(t) =$ probability of system working correctly provided that it was working at $t=0$
  - **Maintainability** $M(d) =$ probability of system working correctly $d$ time units after error occurred.
  - **Availability** $A(t)$: probability of system working at time $t$
  - **Safety**: no harm to be caused
  - **Security**: confidential and authentic communication

- Even perfectly designed systems can fail if the assumptions about the workload and possible errors turn out to be wrong. Making the system dependable must not be an after-thought, it must be considered from the very beginning
Common characteristics: Efficiency

- ES must be **efficient**
  - Code-size efficient (especially for systems on a chip)
  - Run-time efficient
  - Weight efficient
  - Cost efficient
  - Energy efficient
Importance of energy efficiency

- Stuck at 200Gop/J
- Lack of $V_{DD}$ scaling

32 bit

 Courtesy: Philips
Connected to the physical environment

A/D converter
sample-and-hold

information processing

display

D/A converter

environment

sensors

actors
Real-time constraints

- Many ES must meet **real-time constraints**
  - A real-time system must react to stimuli from the controlled object (or the operator) within the time interval **dictated** by the environment.
  - For real-time systems, right answers arriving too late are wrong.
  - "A real-time constraint is called hard, if not meeting that constraint could result in a catastrophe" [Kopetz, 1997].
  - All other time-constraints are called **soft**.
  - A guaranteed system response has to be explained without statistical arguments.
Real-Time Systems

Embedded and Real-Time Synonymous?

- Most embedded systems are real-time
- Most real-time systems are embedded
Reactive & hybrid systems

- Typically, ES are reactive systems:
  “A reactive system is one which is in continual interaction with its environment and executes at a pace determined by that environment” [Bergé, 1995]
  Behavior depends on input and current state.
  \[\text{automata model appropriate,}
  \]
  model of computable functions inappropriate.

- Hybrid systems
  (analog + digital parts).
Dedicated systems

- **Dedicated** towards a certain **application**
  Knowledge about behavior at design time can be used to minimize resources and to maximize robustness

- **Dedicated user interface**
  (no mouse, keyboard and screen)
Challenges in ES Design

Dependability?

- Non-real time protocols used for real-time applications (e.g. Berlin fire department)

- Over-simplification of models (e.g. aircraft anti-collision system)

- Using unsafe systems for safety-critical missions (e.g. voice control system in Los Angeles; ~ 800 planes without voice connection to tower for > 3 hrs)
It is not sufficient to consider ES just as a special case of software engineering. EE knowledge must be available, Walls between EE and CS must be torn down.
Challenges for implementation in hardware

- Lack of flexibility (changing standards).
- Mask cost for specialized HW becomes very expensive

[Trend towards implementation in Software]

Importance of Embedded Software and Embedded Processors

“... the New York Times has estimated that the average American comes into contact with about 60 micro-processors every day....” [Camposano, 1996]

Latest top-level BMWs contain over 100 micro-processors
[Personal communication]

Most of the functionality will be implemented in software
Challenges for implementation in software

If embedded systems will be implemented mostly in software, then why don‘t we just use what software engineers have come up with?
Software complexity is a challenge

- Exponential increase in software complexity
- In some areas code size is doubling every 9 months [ST Microelectronics, Medea Workshop, Fall 2003]
- ... > 70% of the development cost for complex systems such as automotive electronics and communication systems are due to software development [A. Sangiovanni-Vincentelli, 1999]
Challenges for Embedded Software

- Dynamic environments
- Capture the required behaviour!
- Validate specifications
- Efficient translation of specifications into implementations!
- How can we check that we meet real-time constraints?
- How do we validate embedded real-time software? (large volumes of data, testing may be safety-critical)
Design flows

Hypothetical design flow

Generic loop: tool chains differ in the number and type of iterations
Design flows
Particular design environment

Example:
SpecC tools

- Specification model
  - simulation
  - architecture exploration
  - validation

- Architecture model
  - simulation
  - communication synthesis
  - evaluation

- Communic. model
  - simulation
  - software compilation
  - evaluation

- Implementat. model
  - simulation
  - hardware compilation
  - validation

- Manufacturing
Design flows (Iterative design)

Example: V-model

- Requirement analysis
- System architecture
- System design
- Software architecture
- Software design
- Integration testing
- Unit tests
- Acceptance & use
- System integration
Design flows (Gajski Y-chart and design path)
Design flows (Gajski Y-chart and design path)

- High level model describing overall behavior
- Information about hardware component
  - High level = Processors
  - Low level = Transistor
- Behavioral
- Structural
- Geometrical
- Design path
- High level of abstraction
- Low level of abstraction
- Layout information such as chips
Synthesis?

Definition: **Synthesis** is the process of generating the description of a system in terms of related lower-level components from some high-level description of the expected behavior.

Manual design steps are more error-prone than automatic synthesis and, therefore, simulation is more important.
Design Representation

Conceptualization of Functions → Manufacturing Blueprint

Marketing → Chief Architect
- Determine requirements
- Convert requirements into Architecture

Technologists
- Selects Possible technology, possible components and suppliers
Design Representation

1. CAD and CAD Software Engineers to acquire the tools
2. Design team prepares the blueprint of the manufacturing process
3. Software Engineers will write the code for the processors
4. Software Engineers will write the code for the processors
5. Test Engineers develop test strategies
Design Representation (Three important representation)

BEHAVIORAL
(Describes the systems functionality in the form of black boxes)

STRUCTURAL
(Define the black box in-terms of set of components and their interconnections)

PHYSICAL
(Implementation dimension, location of each component, describes the weight, height, size, heat dissipation, power consumption and position of each input/output.)
Design representation and Abstraction Levels.

- **Behavioral**
  - Differential eq., Current Voltage diagrams
  - Boolean equations finite-state machines
  - Algorithms flowcharts, Instruction set, Generalized FSM
  - Executable spec., programs

- **Structural**
  - Transistor resistors and capacitors
  - Gates Flipflops
  - Adders, comparators, registers, counters, register files queues
  - Processors, controllers, memories and ASICs

- **Physical Objects**
  - Analog and Digital Cells
  - Modules, Units
  - Micro chips ASICs
  - PCBs, MCMs