

Hardware-Software Design of Embedded Systems



Luca Benini
DEIS Università di Bologna
AA 2010-2011

Credits: Marwedel 2011, Wolf 2008

Motivation for Course

- Electronics everywhere
 - Disappearing computer,
 - Ubiquitous computing,
 - Pervasive computing,
 - Ambient intelligence,
 - Post-PC era.
- Basic technologies:
 - *Embedded Systems*
 - Communication technologies



Definition

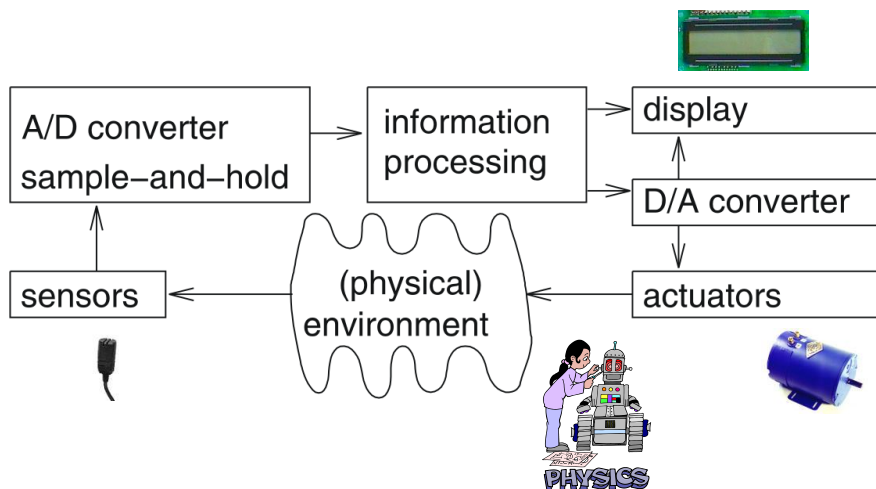
- **Embedded computing system**: any device that includes a programmable computer but is not itself a general-purpose computer.
- Take advantage of application characteristics to optimize the design

© 2008 Wayne Wolf

Overheads for *Computers as Components*, 2nd ed.

Embedded Systems

- The cyber-physical loop



Transportation ES

– Automotive electronics



– Avionics



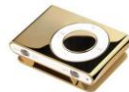
– Trains



Consumer ES

• Mobile

- Apple's trio
- UMPC



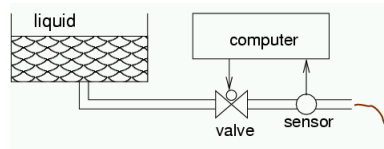
• Home

- DTV (QuadHDTV, UHD, ...)
- Home media center (iTV, ...)
- Game (PS3, Xbox, ...)



Smart Spaces ES

▪ Industrial automation



▪ Smart buildings



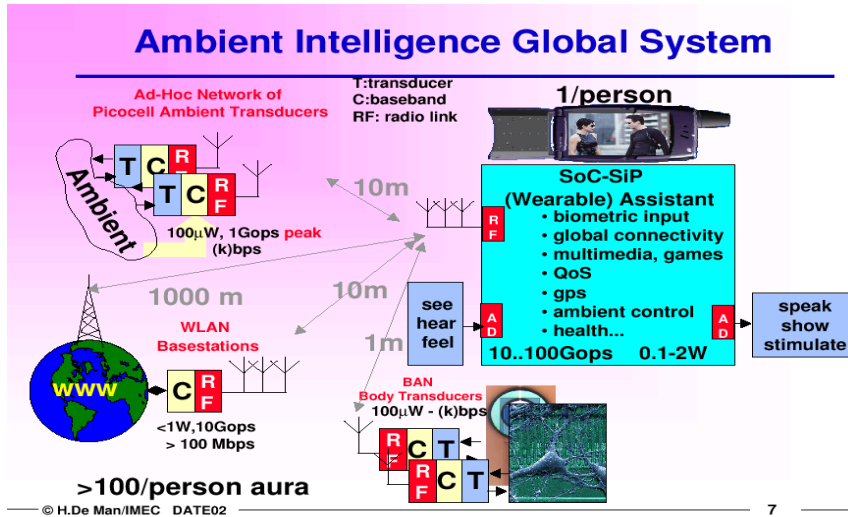
Upcoming Applications

- Picoprojector + picomimio



[Source: J. Chong, ...]

Upcoming Applications



Structure of this course

Architectures

Design flows

High-end
Multicore

GPU

Low-cost
Low-power

μ ctrl

Board
Design

SoC
Design

SW
Design

Notes on the class

- Web: <http://www-micrel.deis.unibo.it/MPHS/>
- Teacher: Prof. Luca Benini luca.benini@unibo.it

- Organization:
 - Lectures - theory and examples
 - Learning on the job: projects

- Examination:
 - In-class tests (3) + Oral Examination
 - In-class tests (3) + Project

- Pre-requisites
 - Basic digital electronics (i.e. CMOS gates, SRAM, DRAM)
 - Basic Computer Architectures (i.e. Cache, pipeline)
 - C programming (i.e. pointers, compiler, debugger)

Examples

Some embedded systems from
real life

Pedometer

- Obvious computer work:
 - Count steps
 - Keep time
 - Averages
 - etc.
- Hard computer work:
 - Actually identify when a step is taken
 - Sensor feels motion of device, not of user feet



© Jakob Engblom

Mobile phones



- Multiprocessor
 - 8-bit/32-bit for UI
 - DSP for signals
 - 32-bit in IR port
 - 32-bit in Bluetooth
- 8-100 MB of memory
- All custom chips
- Power consumption & battery life depends on software

© Jakob Engblom

Telecom Switch



- Rack-based
 - Control cards
 - IO cards
 - DSP cards
 - ...
- Optical & copper connections
- Digital & analog signals

© Jakob Engblom

Cars

Functions by embedded processing:

- ABS: Anti-lock braking systems
- ESP: Electronic stability control
- Airbags
- Efficient automatic gearboxes
- Theft prevention with smart keys
- Blind-angle alert systems
- ... etc ...

- Multiple networks
 - Body, engine, telematics, media, safety
- Multiple processors
 - Up to 100
 - Networked together

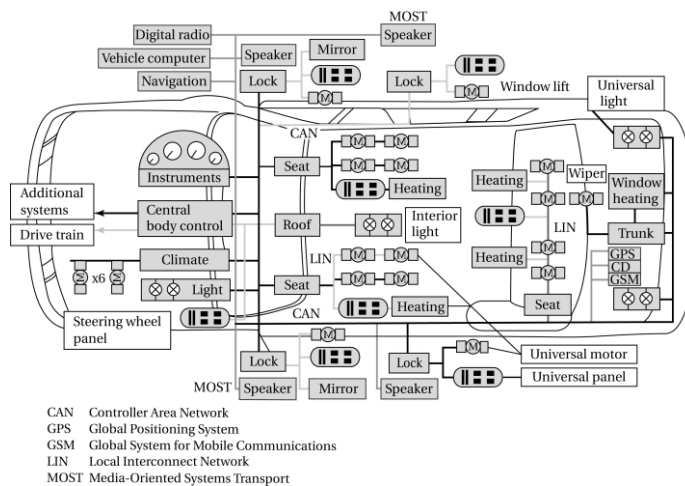


© Jakob Engblom

Networks and embedded systems

- An increasing number of embedded systems connect to the Internet.
 - Resource management.
 - Security.
- Many specialized networks have been developed for embedded systems:
 - Automotive.
 - Device control.

Automobiles as distributed embedded systems



Characteristics



Graphics: © Alexandra Nolte, Gesine Marwedel, 2003

Characteristics of ES

- Sophisticated functionality.
- Real-time operation.
- Low manufacturing cost.
- Low power.
- Designed to tight deadlines by small teams.

Functional complexity

- Often have to run sophisticated algorithms or multiple algorithms.
 - Cell phone, laser printer.
- Often provide sophisticated user interfaces.

Real-time operation

- Must finish operations by deadlines.
 - **Hard real time**: missing deadline causes failure.
 - **Soft real time**: missing deadline results in degraded performance.
- Many systems are **multi-rate**: must handle operations at widely varying rates.

Non-functional requirements

- Many embedded systems are mass-market items that must have low manufacturing costs.
 - Limited memory, microprocessor power, etc.
- Power consumption is critical in battery-powered devices.
 - Excessive power consumption increases system cost even in wall-powered devices.

Design teams

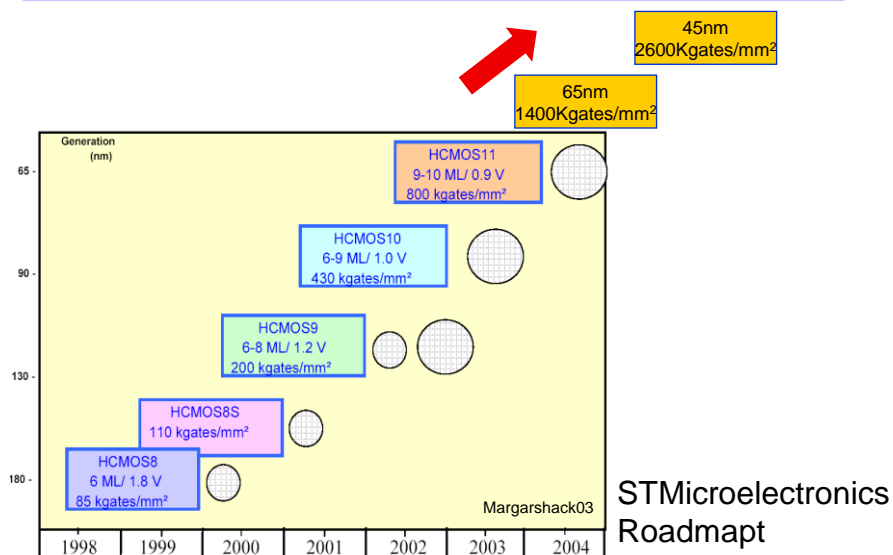
- Often designed by a small team of designers.
- Often must meet tight deadlines.
 - 6 month market window is common.
 - Can't miss back-to-school window for calculator.

Implementation Fabrics



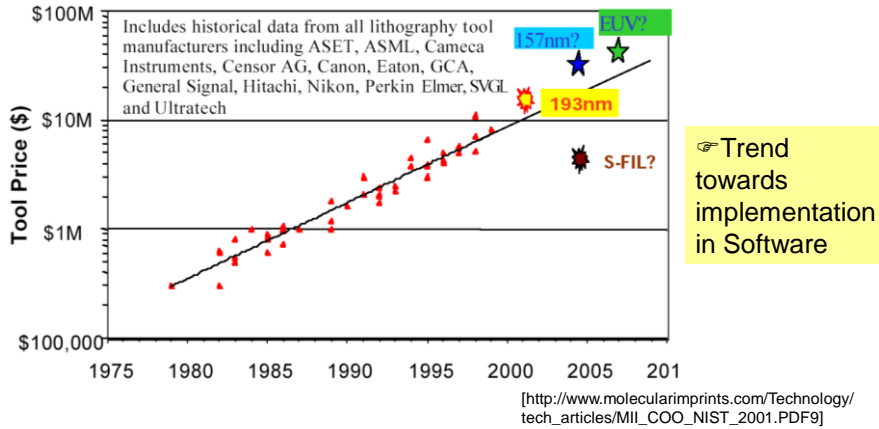
Graphics: © Alexandra Nolte, Gesine Marwedel, 2003

Embedded HW: Moore's Law



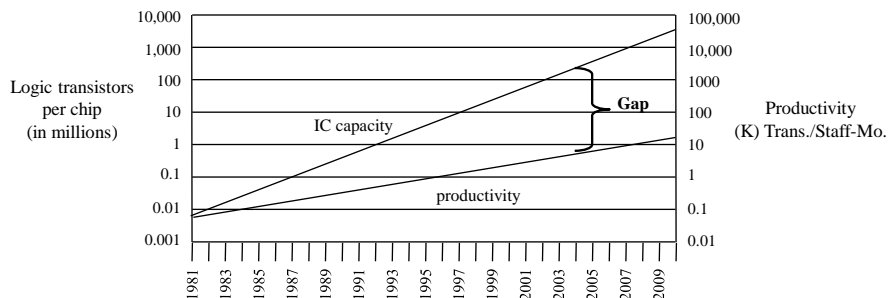
Challenges for implementation in hardware

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



Design productivity gap

- 1981 leading edge chip required 100 designer months
 - 10,000 transistors / 100 transistors/month
- 2002 leading edge chip requires 30,000 designer months
 - 150,000,000 / 5000 transistors/month
- Designer cost increase from \$1M to \$300M



The performance paradox

- Microprocessors use much more logic to implement a function than does custom logic.
- But microprocessors are often at least as fast:
 - heavily pipelined;
 - large design teams;
 - aggressive VLSI technology.

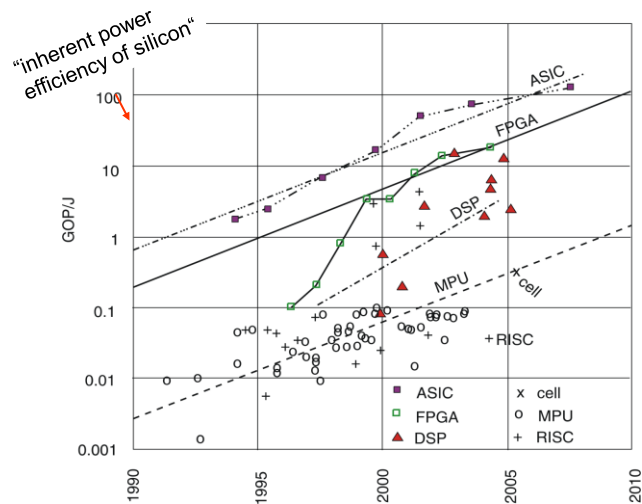
Power

- Custom logic uses less power, but CPUs have advantages:
 - Modern microprocessors offer features to help control power consumption.
 - Software design techniques can help reduce power consumption.
- Heterogeneous systems: some custom logic for well-defined functions, CPUs+software for everything else.

Importance of Energy Efficiency

Efficient software design needed, otherwise, the price for software flexibility cannot be paid.

© Hugo De Man, IMEC, Philips, 2007



Challenges in ES design

- How much hardware do we need?
 - How big is the CPU? Memory?
- How do we meet our deadlines?
 - Faster hardware or cleverer software?
- How do we minimize power?
 - Turn off unnecessary logic? Reduce memory accesses?

Platforms

- Embedded computing platform: hardware architecture + associated software.
- Many platforms are multiprocessors.
- Examples:
 - Single-chip multiprocessors for cell phone baseband.
 - Automotive network + processors.

© 2008 Wayne Wolf

Overheads for *Computers as
Components*, 2nd ed.

Hardware-Software Codesign

An Introduction



Graphics: © Alexandra Nolte, Gesine Marwedel, 2003

Design methodologies

- A procedure for designing a system.
- Understanding your methodology helps you ensure you didn't skip anything.
- Compilers, software engineering tools, computer-aided design (CAD) tools, etc., can be used to:
 - help automate methodology steps;
 - keep track of the methodology itself.

© 2008 Wayne Wolf

Overheads for *Computers as Components*, 2nd ed.

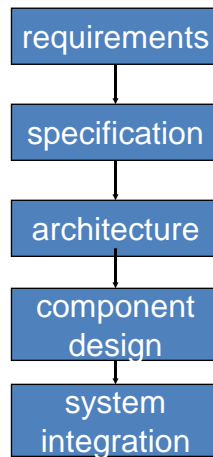
Design goals

- Performance.
 - Overall speed, deadlines.
- Functionality and user interface.
- Manufacturing cost.
- Power consumption.
- Other requirements (physical size, etc.)

© 2008 Wayne Wolf

Overheads for *Computers as Components*, 2nd ed.

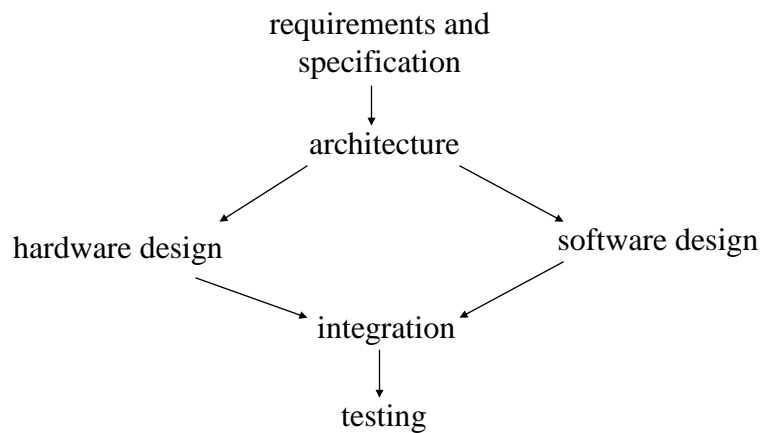
Levels of abstraction



© 2008 Wayne Wolf

Overheads for *Computers as Components*, 2nd ed.

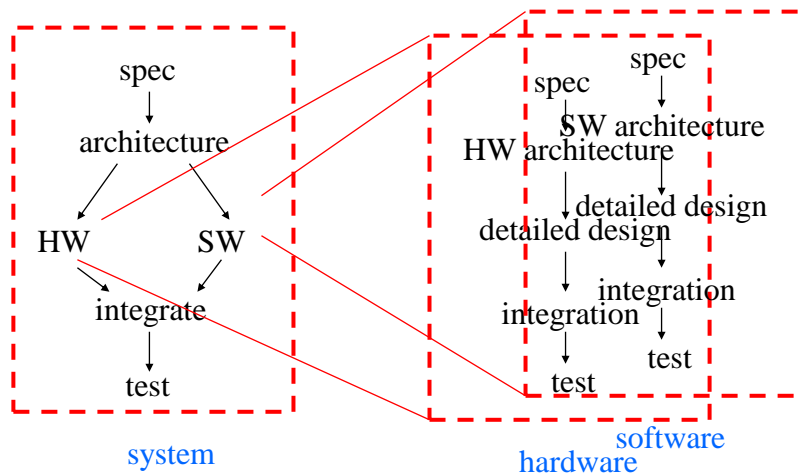
Hardware/software design flow



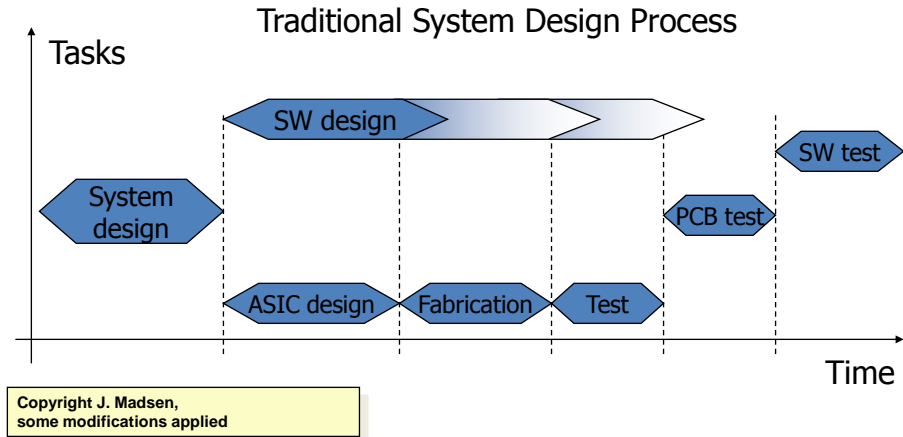
Co-design methodology

- Must architect hardware and software together:
 - provide sufficient resources;
 - avoid software bottlenecks.
- Can build pieces somewhat independently, but integration is major step.
- Also requires bottom-up feedback.

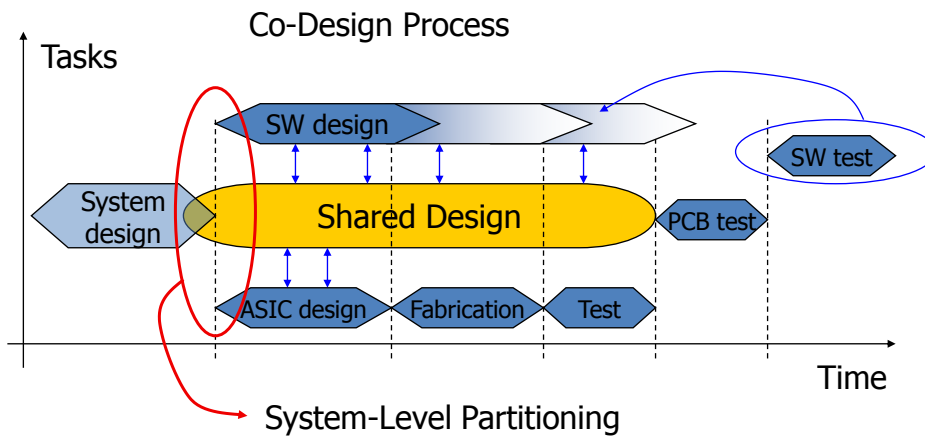
Hierarchical HW/SW flow



Codesign in time



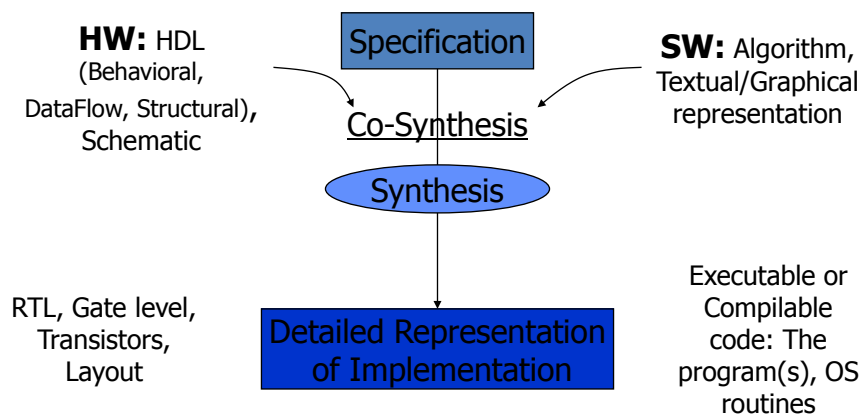
Codesign in time II



Computer-aided hardware/software co-design

- Explore different design alternatives
 - Search for the best solution
- Reduce system design time
 - Reduce product *time to market*
- Support coherent design specification
 - Facilitate hardware and software reuse
- *Provide integrated environment for synthesis and validation of hardware and software components*

Synthesis



Co-synthesis steps

- System-level *modeling* and *partitioning*
- *Hardware synthesis* of dedicated units
 - Research/commercial standard synthesis tools
- *Software synthesis* for processor (core)
 - Specialized compiling techniques
- *Interface synthesis*
 - Hardware-software interface and synchronization
 - Drivers of peripheral devices

System-Level Partitioning

- One extreme: Full HW solution
 - High performance due to *Parallelism*
 - High cost and long time of ASIC fabrication
- Other extreme: Full SW solution
 - High-performance, low-cost processors
 - Operation serialization
 - Lack of support for specific tasks
- Best solution is a mix of HW and SW

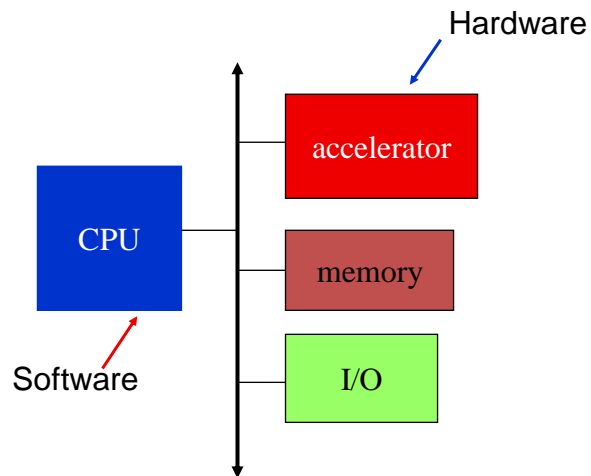
HW/SW partitioning goals

- *Speed-up* software execution
 - By migrating software functions dedicated hardware
- *Reduce cost* of hardware implementation
 - By migrating hardware functions to software
- Achieve system *prototypes*
 - By migrating functions to the prototype medium

Example of partitioning approaches

- *Software speed-up* by using dedicated hardware
- Model system as software program
 - C-like language with performance constraints
- Determine performance bottlenecks
- Migrate critical portions to a hardwired unit

System architecture



Issues in system partitioning

- Object *granularity* in partitioning
 - Operation versus functions
- *Estimation* of performance and cost from graph model
 - Satisfaction of design constraints
- Incorporate designer's experience in biasing a partition

Co synthesis after partitioning

- Software functions identified by program threads
- A single processor requires thread serialization or interleave
- Scheduling of threads and instructions
 - Satisfying performance constraints
- System-level run-time scheduler to synchronize software and hardware functions

Hardware synthesis

- After partitioning, HW behavior is still described at high level of abstraction
- HW synthesis alternatives
 1. Manual translation to RTL
 2. Use Behavioral synthesis

SW synthesis

- SW synthesis \cong Compilation
- Maybe preceded by automatic SW generation steps
- Compilation specialties in embedded systems
 - Code is compiled once \rightarrow compilation time not important, maximum optimization is desired
 - RT constraints \rightarrow code must be tight and fast \rightarrow assembly

Interface synthesis

- Processor interacts with ASIC and peripheral devices via serial or parallel ports
- Device drivers may be in software or in hardware
- Allocate ports to devices
- Schedule processor communication
- Buffer insertion

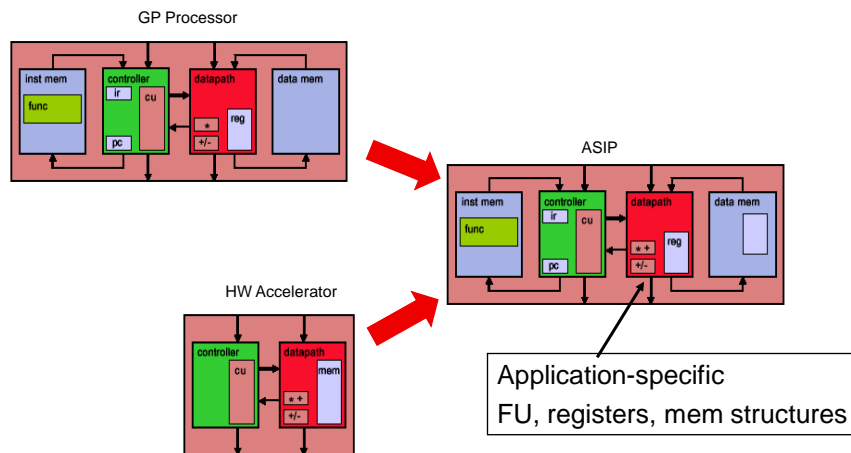
Interface synthesis issues

- Specific problems arise from:
 - Communication and synchronization patterns
 - Memory mapped, DMA, etc.
 - Interfacing processors to peripherals (sensors, actuators)
 - automatic (processor port allocation, deciding to implement device drivers in HW or SW)
 - Scheduling the processor communication
 - Complicated under RT constraints
 - Complicated under data-dependant delays

An alternative approach

- Application-specific hardware is strongly decoupled from the processor
 - It can be viewed as a hardware accelerator
- Communication is expensive
- What if I create an application-specific processor (ASIP)?

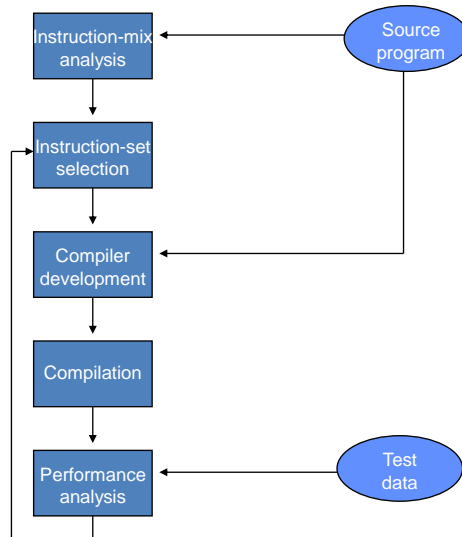
Application specific instruction processors



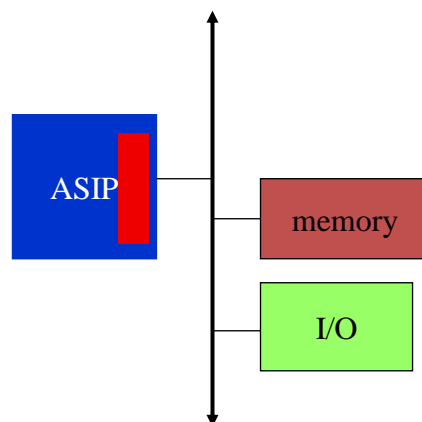
ASIP vs. Accelerator

- Much tighter interaction between software and application-specific hardware
 - ✓ Communicate through registers, on a cycle-by-cycle basis (low overhead)
 - ✗ Instruction fetch bottleneck (loss of parallelism)
- Fine granularity of operations
 - ✗ Cannot migrate arbitrarily complex functions in application specific instructions
 - ✓ Opportunity for fine tuning other microarchitectural parameters

ASIP Co-synthesis flow



System architecture



Software synthesis problems

- Target architecture is an ASIP
 - Develop a specific compiler
- Developing a new compiler for each new ASIP is unreasonable → retargetable compilers
 - Classification according to retargeting time
 - Portable compiler
 - Compiler-compiler
 - Machine-independent compiler

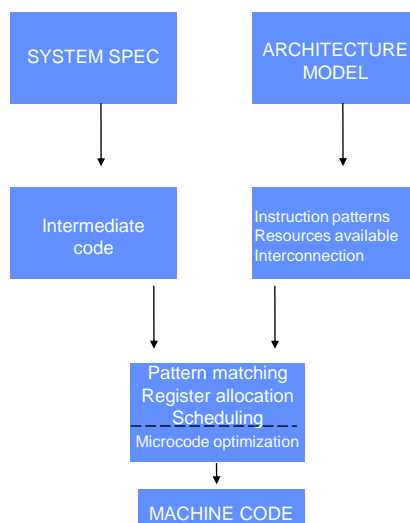
Retargetable compilers

- *Portable compilers*
 - Compiler needs significant rewrite for porting
- *Compiler compilers*
 - Generates compilers from architectural template
- *Machine-independent compilers*
 - Applicable to different architectures

Retargetable compilers

- Compiler technology adaptable to different architectural back-ends
 - ASIPs have specific instruction sets, memory and interconnection resources
- *Code quality* (i.e. execution speed) is important whereas *compilation time* is less critical
- Assembly code programming is still common practice

Retargetable compiler architecture



A third alternative: mapping to a reconfigurable SoC

- Software mapping
 - Onto an embedded core on the SoC
- Hardware mapping
 - Using FPGA or DSP-FPGA fabric
- Same partitioning decisions!
- Difference: resource bounds are fixed by the chosen reconfigurable architecture
 - Reduces degrees of freedom

Today's Reconfigurable System

Xilinx Virtex II Pro

