Chapter 1-2: Embedded Computing

Soo-Ik Chae

Topics

- Design methodologies.
- Methodologies and standards.

Design goals

- Functional requirements: input/output relations.
- Non-functional requirements: cost, performance, power, etc.
- Some project goals may be difficult to measure.

Aspects of performance

- Embedded system performance can be measured in many ways:
  - Average vs. worst/best-case.
  - Throughput vs. latency.
  - Peak vs. sustained.

Energy/power

- Energy consumption is important for battery life.
- Power consumption is important for heat generation or for generator-powered systems (vehicles).

Cost

- Manufacturing cost must be paid off across all the systems.
  - Hardest in small-volume applications.
- Manufacturing cost is incurred for each device.
- Design cost is determined both by the labor and by the equipment used to support the designers: NRE
- Lifetime costs include software and hardware maintenance and upgrades.

Other design attributes

- Design time must be reasonable. May need to finish by a certain date: time-to-market.
- System must be reliability; reliability requirements differ widely.
- Quality includes reliability and other aspects: usability, durability, etc.
  - Difficult to measure
  - User interface

Design methodology

- Design methodology: a procedure for creating an implementation from a set of requirements.
  - It is not simply an abstraction; it must be defined in terms of available tools and resources.
- Methodology is important in embedded computing:
  - Must design many different systems.
  - We may use same/similar components in many different designs.
  - Design time, results must be predictable.

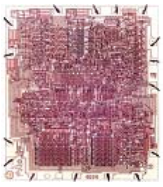
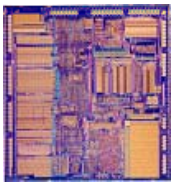
Embedded system design challenges

- Design space is large and irregular. We don't have synthesis tools for many steps.
  - Must rely on analysis and simulation for many design phases
- Can't simulate everything.
  - Limited time
  - The cost of the server: significant portion of the design cost
  - Cycle-accurate simulation is very slow
- Need to develop simulators as soon as possible
  - Simulators must reflect the structure of the application-specific designs
  - System architects need tools to help them construct application-specific simulators.
- Often need to start software development before hardware is finished.
  - To evaluate not just functionality but performance and power as well.

Moore's Law

Gorden Moore (Intel Co-founder)

The number of transistors on a microprocessor would double approximately every 18 months (1965)

	Intel 4004 (1971)	Intel 8086 (1978)	Intel 80386 (1985)	Intel Pentium (1993)	Intel Pentium4 (2000)
					
Technology	10 um	3 um	1.5 um	0.8 um	0.13 um
Tr count	2,300	29K	275K	3.1 M	42M
Op. Freq.	800 KHz	10 MHz	16 MHz	66 MHz	3.4 GHz

ITRS Roadmap

Moore's law will be alive at least for 10 years

Table 1a Product Generations and Chip Size Model Technology Trend Targets—Near-term Years

Year of Production	2005	2006	2007	2008	2009	2010	2011	2012	2013
DRAM ½ Pitch (nm) (contacted)	80	70	65	57	50	45	40	36	32
MPU/ASIC Metal 1 (M1) ½ Pitch (nm)	90	78	68	59	52	45	40	36	32
MPU Printed Gate Length (nm) ††	54	48	42	38	34	30	27	24	21
MPU Physical Gate Length (nm)	32	28	25	23	20	18	16	14	13
ASIC/Low Operating Power Printed Gate Length (nm) ††	76	64	54	48	42	38	34	30	27
ASIC/Low Operating Power Physical Gate Length (nm)	45	38	32	28	25	23	20	18	16
Flash ½ Pitch (nm) (un-contacted Poly)(f)	76	64	57	51	45	40	36	32	28

Table 1b Product Generations and Chip Size Model Technology Trend Targets—Long-term Years

Year of Production	2014	2015	2016	2017	2018	2019	2020
DRAM ½ Pitch (nm) (contacted)	28	25	22	20	18	16	14
MPU/ASIC Metal 1 (M1) ½ Pitch (nm)	28	25	22	20	18	16	14
MPU Printed Gate Length (nm) ††	19	17	15	13	12	11	9
MPU Physical Gate Length (nm)	11	10	9	8	7	6	6
ASIC/Low Operating Power Printed Gate Length (nm) ††	24	21	19	17	15	13	12
ASIC/Low Operating Power Physical Gate Length (nm)	14	13	11	10	9	8	7
Flash ½ Pitch (nm) (un-contacted Poly)(f)	25	23	20	18	16	14	13

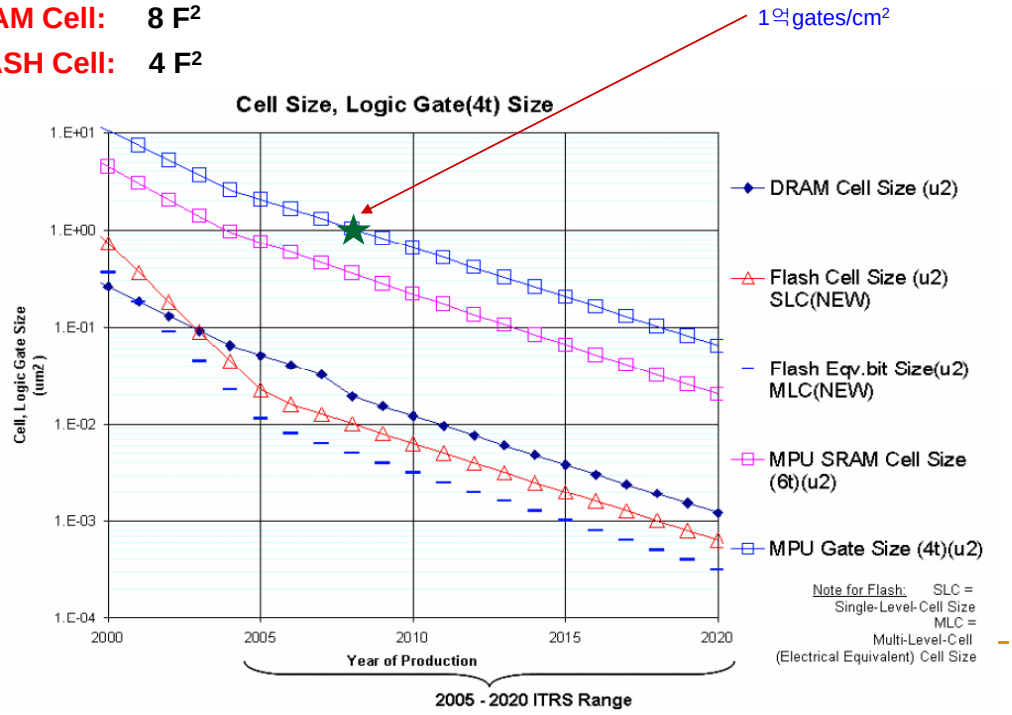
Trends of Memory Size and Gate Size

Gate Logic: **Average size of 4t gate = 320 F^2**

SRAM Cell: **140 F^2**

DRAM Cell: **8 F^2**

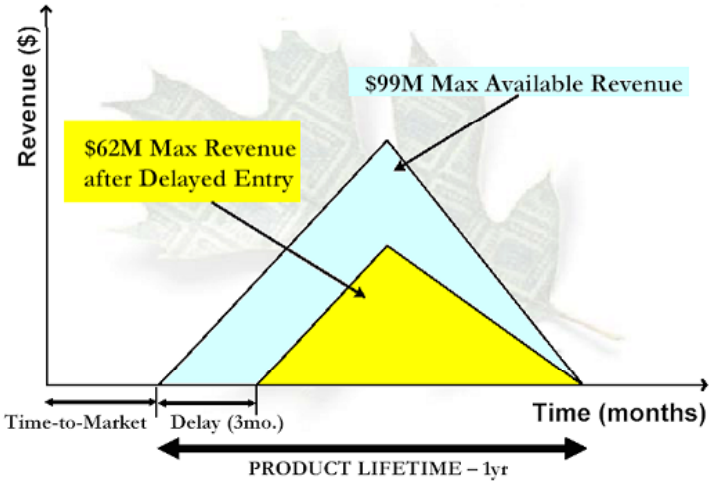
FLASH Cell: **4 F^2**



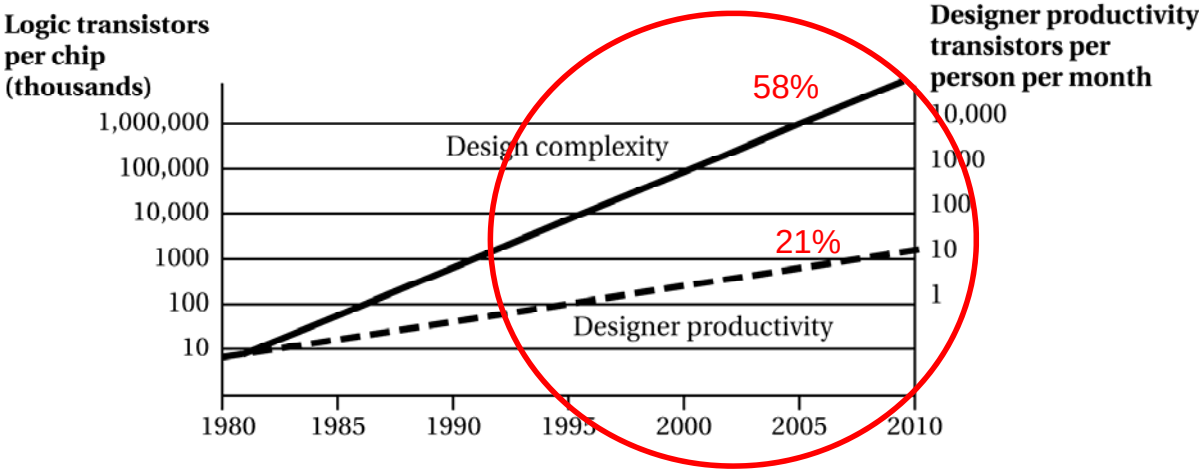
Time-To-Market / Time-In-Market

Managing time-to-market is critical

Two year life-cycle products lost 34% of potential revenue and 50% of profit if they are 3 months late



Design complexity vs. designer productivity



Estimated by Sematech in mid-1990s.

Design complexity vs. designer productivity

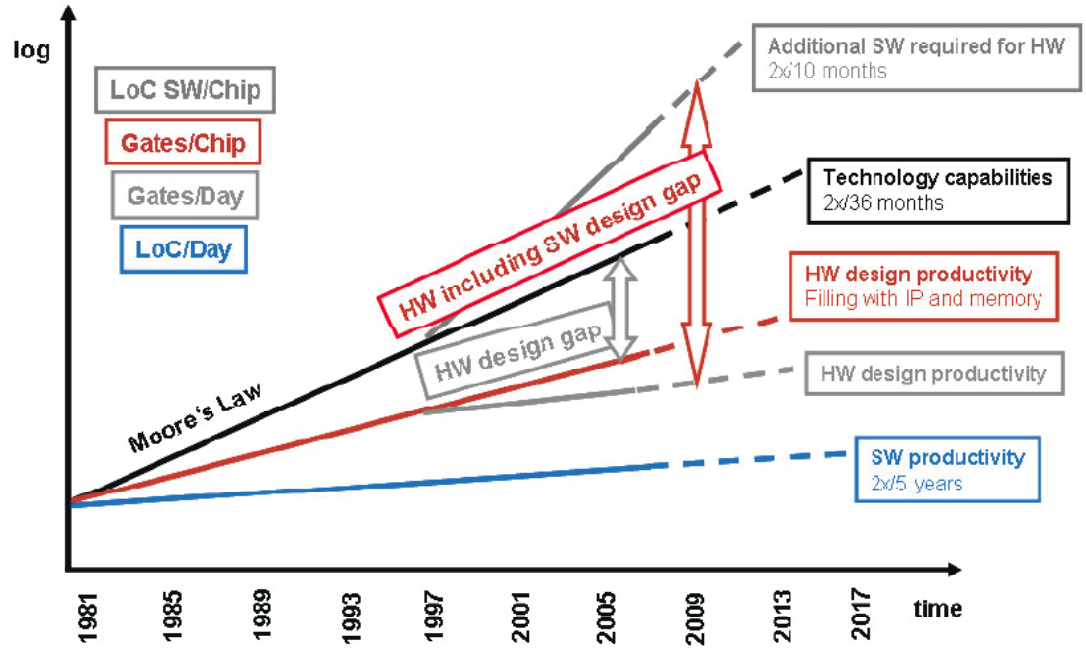
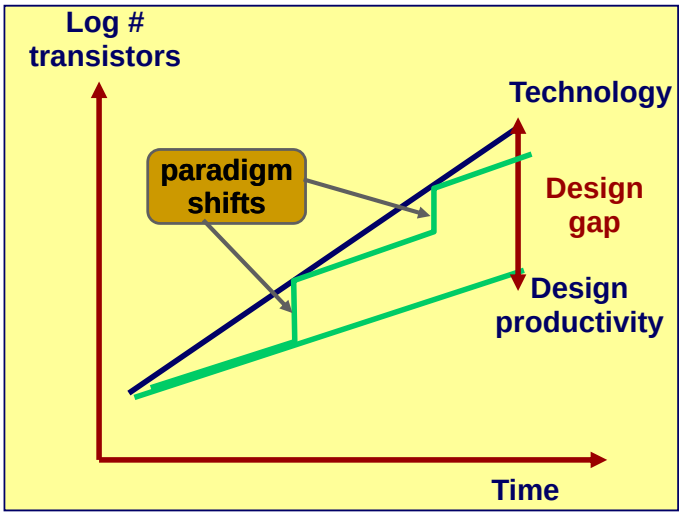


Figure DESN3 Hardware and Software Design Gaps Versus Time[5]

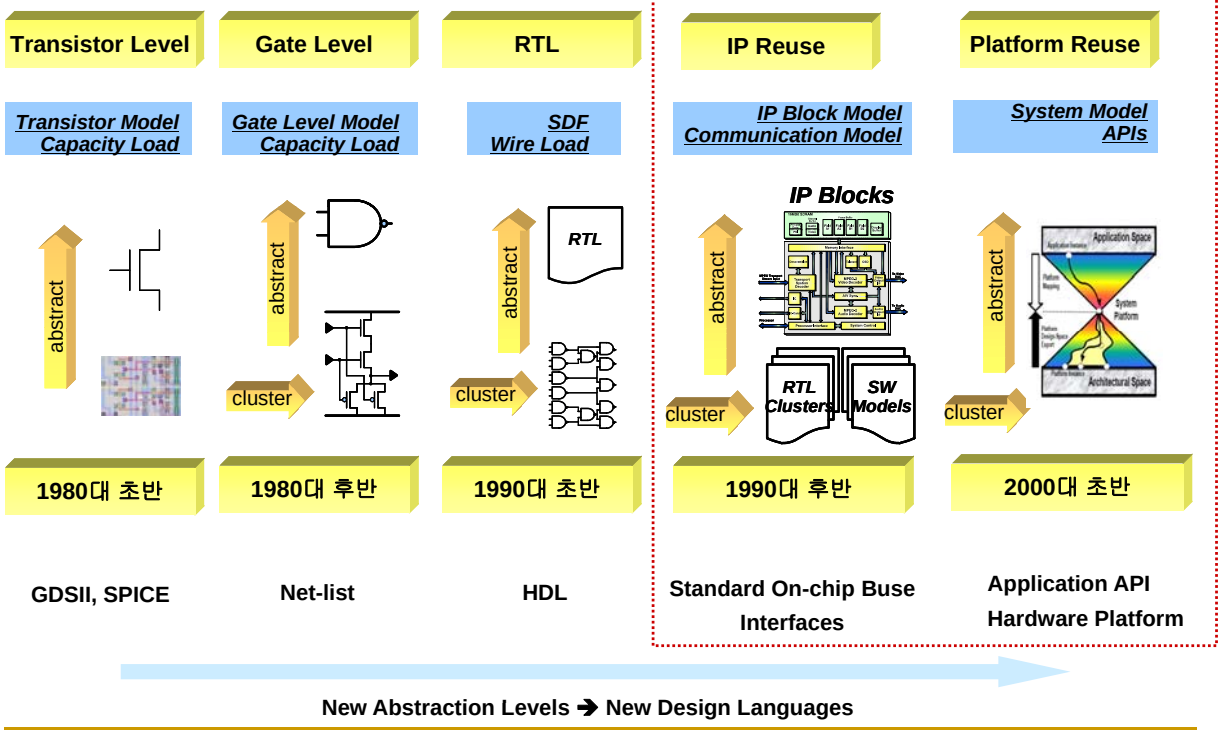
From ITRS2007

Design Paradigm Shifts



➔ Paradigms shifts in design methodology have been the only escape from the design gap.

Levels of Abstraction



State-of-the-art System Design Environment

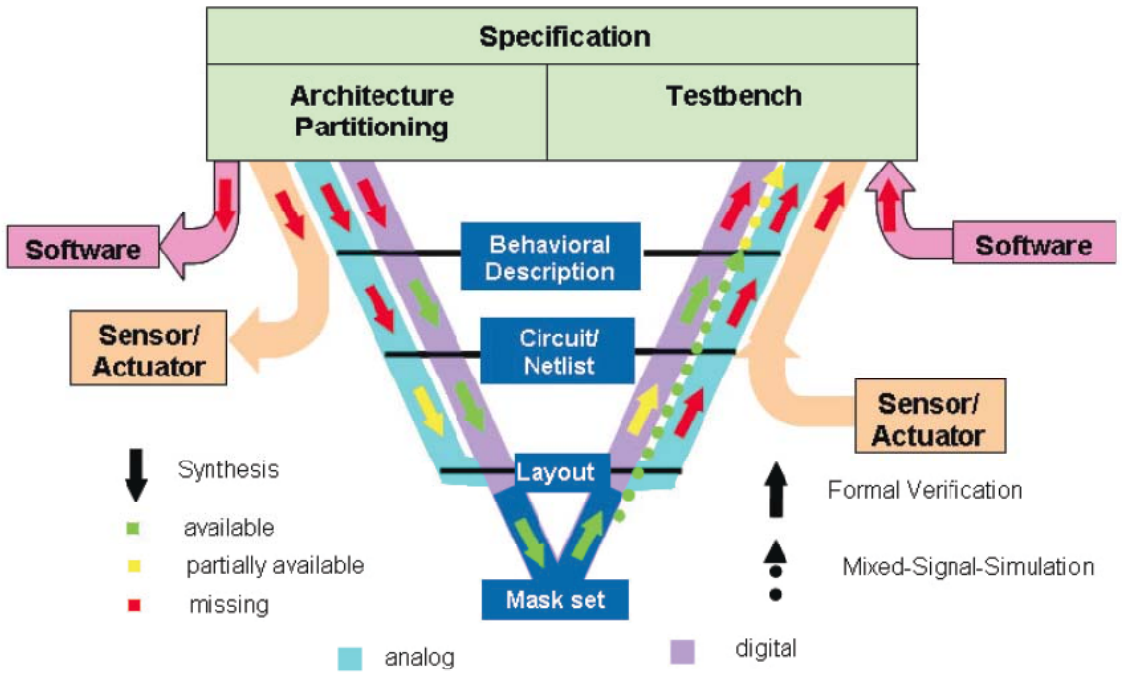


Figure DESN2 The V-Cycle for Design System Architecture[4] From ITRS 2007

Impact of design technology improvement on design productivity

From ITRS 2007

Very large block reuse	2007	+200%	600K	Chip/circuit/PD Verification	Blocks >1M gates; intellectual-property cores
Homogeneous parallel processing	2009	+100% HW +100% SW	1200K	Chip/circuit/PD Design and Verification	Many identical cores provide specialized processing around a main processor, which allows for performance, power efficiency, and high reuse
Intelligent test bench	2011	37.5%	1650K	Chip/circuit/PD Verification	Like RTL verification tool suite, but also with automation of the Verification Partitioning step
Concurrent software compiler	2013	200% SW	1650K	Chip and Electronic System Design and Verification	Enables compilation and SW development in highly parallel processing SOCs
Heterogeneous massive parallel processing	2015	+100% HW +100% SW	3300K	System Electronic Design and Verification	Each of the specialized cores around the main processor is not identical from the programming and implementation standpoint
Transactional Memory	2017	+100% HW +100% SW	6600K	System Electronic Design and Verification	Automates true electronic system design on- and off-chip for the first time, including heterogeneous technologies (Phase 1)
System-level DA	2019	60% HW 38% SW	10557K	System Electronic Design and Verification	Automates true electronic system design on- and off-chip for the first time, including heterogeneous technologies (Phase 2)
Executable specification	2021	200% HW +200% SW	31671K	System Electronic Design and Verification	Automates true electronic system design on- and off-chip for the first time, including heterogeneous technologies (Phase 3)

System Design Cost

From ITRS 2007

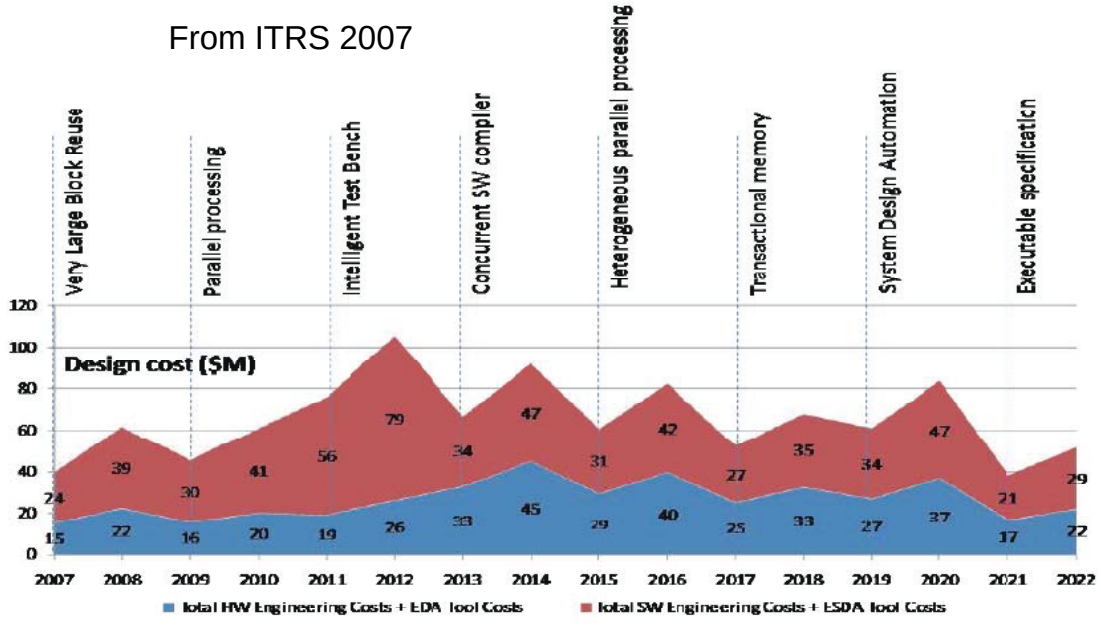


Figure DESN1 Impact of Design Technology on SOC Consumer Portable Implementation Cost

Embedded computing

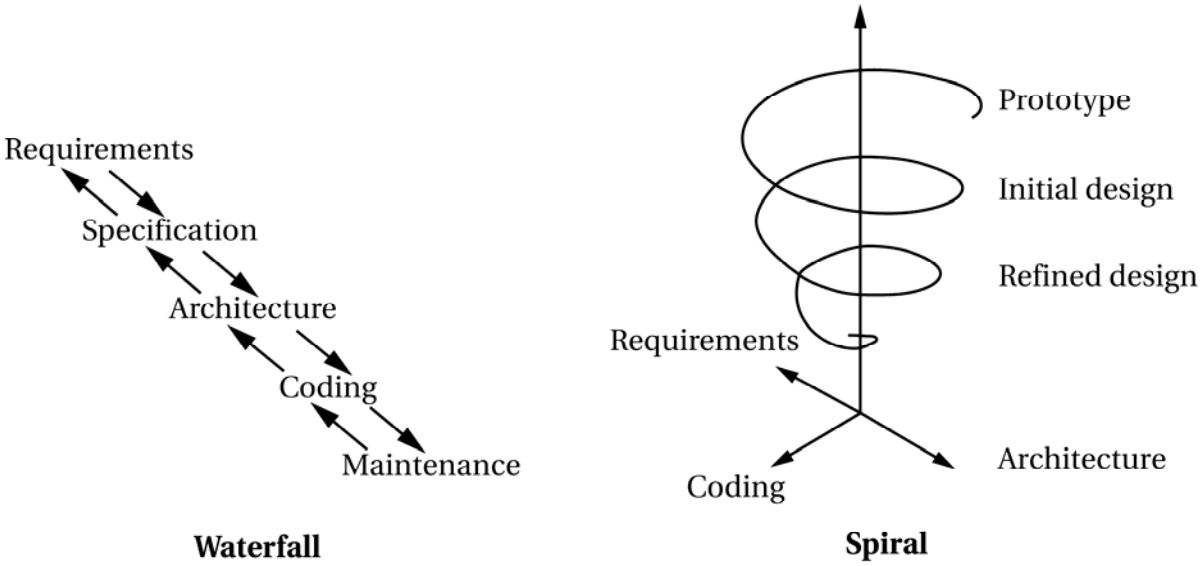
- A partial answer to the design productivity problem, since we move some of the design tasks to software
- But we also need to improve methodologies for embedded computing systems to ensure we can continue to design platforms and load them with useful software.

Design Methodologies

- It is jointly developed by the designers and design technologists.
- It is the sequence of steps by which a design process will reliably produce a design as close as possible to the design target while maintaining feasibility to the constraints.
- Design methodology is distinct from design techniques. All known design methodologies combine
  - Enforcement of system specifications and constraints via top-down planning and search
  - With bottom-up propagation of constraints that stem from physical laws, limits of design and manufacturing technology, and system cost limits

Waterfall and spiral models for software

- Two early models of software development.



Waterfall model

- Five major steps
  - Requirements
  - Specification
  - Architecture
  - Coding
  - Maintenance: delivery and updates and fixes
- Information flow: mostly top-down
  - Unrealistic and undesirable
- In practice, designers can and should use experience from design steps to go back, rethink earlier decisions, and redo some work.

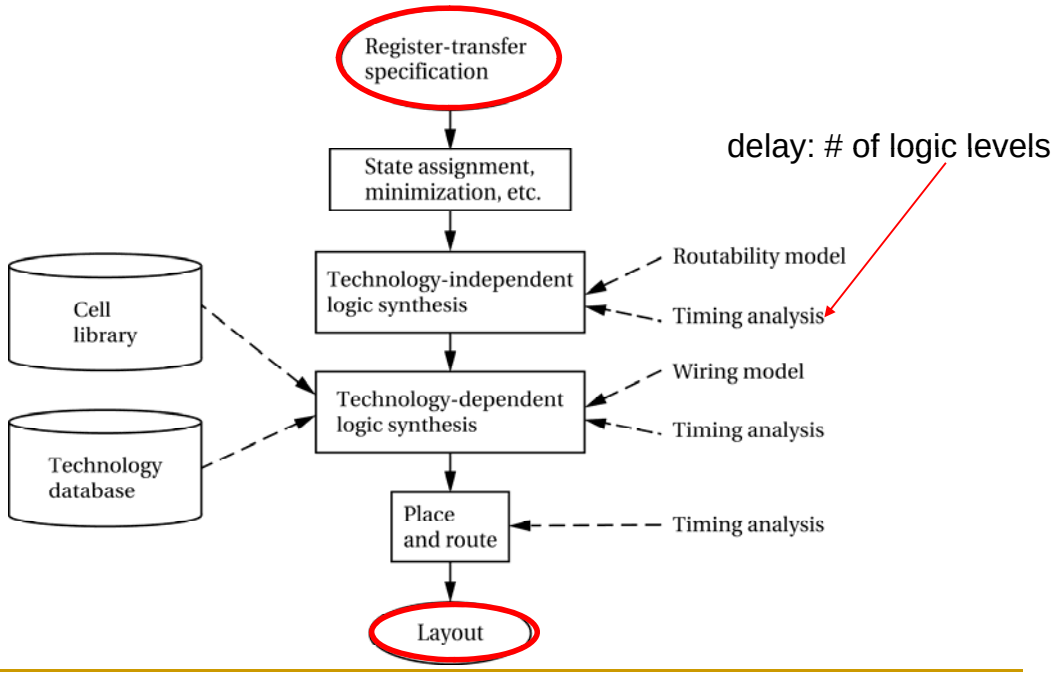
Spiral model

- It was a reaction to and a refinement of the waterfall model.
- It envisions software design as an iterative process in which several versions of the system.
  - Prototype
  - Initial design
  - Refined design
- At each phase, designers go through a requirement-specification-architecture-coding cycle.
- Experience form one stage should help produce a better design in the next design.

Hardware design

- The methodologies for hardware design tend to use a wider variety of tools since hardware design makes more use of synthesis and simulation tools.
- A simplified version of the hardware design flow
  - Figure 1-14: next page
  - Extensive use of several techniques that are not frequently used in software design.
  - Search-based synthesis algorithm
  - Wirling models
  - Estimation algorithms
- Strict cycle-time requirements, power budgets, and area budgets.

Hardware design flow



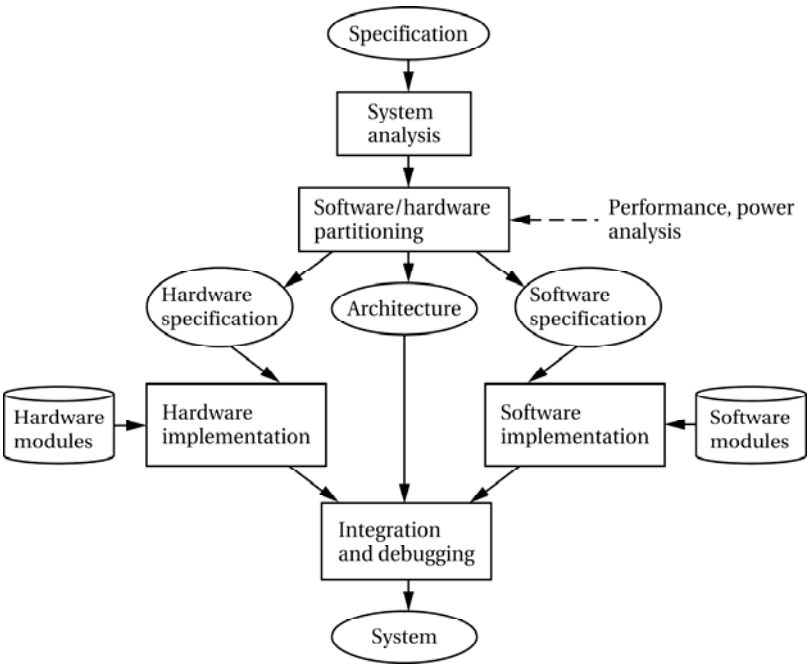
Modeling in hardware

- Technology databases capture manufacturing process information.
- Cell libraries describe the primitive cells used to compose designs.
- Logic synthesis systems use routability and timing models.

Embedded system design flows

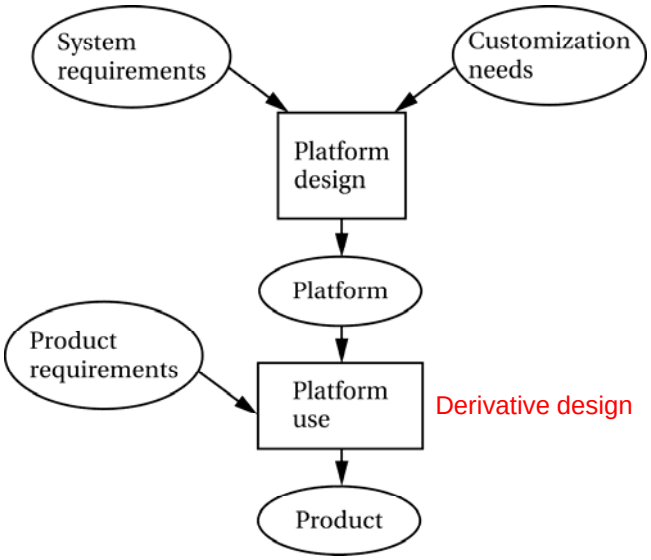
- Embedded system: software components + hardware components
- An embedded system design methodology makes use of the best of both hardware and software traditions.
- Co-design flows: emphasizes the importance of current designs
  - Figure 1-15: a generic co-design methodology from a given executable specification

Hardware/software co-design flow



Platform-based design

- A common approach to designing SoCs
- Platform includes hardware, supporting software.
- Two stage process:
  - Design the platform.
  - Use the platform for designing a derivative.
- Platform can be reused to host many different systems.



Platform design phases

- Transform system requirements and software models into detailed requirements.
  - Use profiling and analysis tools to measure existing executable specifications.
- Explore the design space manually or automatically by evaluating design alternatives
- Develop a hardware platform by optimizing the system architecture based on the results of architecture simulation and other steps.
- Develop its software platform (hardware abstraction layers (HAL) and other software such as OS ports, communications, API, debugging).

Programming platforms

- Programming environment (compilers, editors. Debuggers, simulators) in a single graphic user interface is available for a single CPU.
- The programming environment must be customized to the platform:
  - Multiple CPUs.
  - Specialized memory.
  - Specialized I/O devices.
- Libraries are often used to glue together processors on platforms.
- Debugging environments for multiple processors are a particular challenge.

Standards-based design methodologies

- Standards
  - enable products and create large markets due to inter-operability
  - Large market (several millions chips) justify SoC developments.
  - The designers have much less control over the specification
  - Many standards define that certain operation must be performed , but they do not specify how they are to be performed. (예, ME)
  - Standards tend to become more complex over time.
- Standards generally allow products to be differentiated.
  - Different implementations of operations, so long as I/O behavior is maintained.
  - User interface is often not standardized.
- Standard may dictate certain non-functional requirements (power consumption), implementation techniques.

Reference implementations

- Standard bodies typically provide a reference implementation.
- Executable program that complies with the I/O behavior of the standard.
  - May be written in a variety of language.
  - Often in C or in Java
- In some cases, the reference implementation is the most complete description of the standard.
- Reference implementation is often not well-suited to embedded system implementation:
  - Time-consuming to understand
  - Cannot be used as-is
  - Single process.
  - Infinite memory.
  - Non-real-time behavior.

Reference implementations

- The code must be often restructured in many ways
  - Eliminating features that will not be implemented
  - Replacing heap allocation with custom memory management
  - Improving cache utilization, functions, inlining, and many other tasks.

Designing standards-based systems

1. Design the unspecified parts of the implementation
2. Implement system components not specified by the standard such as user interface.
3. Optimized the reference implementation by performing platform-independent optimizations.
4. Analyze and profile the optimized version of reference implementation.
5. Design hardware platform.
6. Optimize system software based on platform.
7. Further optimize platform.
8. Test for conformity to standard.

H.264/AVC

- Implements video coding for a wide range of applications:
  - Broadcast and videoconferencing.
  - Cell phone-sized screens to HDTV.
- Video codec reference implementation contains 120,000 lines of C code.
- 720p high-profile H.264 Codec
  - 20 ~ 100 Gips (estimated)

Design verification and validation

- Testing exercises an implementation by supplying inputs and testing outputs.
- Validation compares the implementation to the initial specification or requirements.
  - You built the right thing, which refers back to the user's needs..
- Verification may be performed at any design stage; compares design at one level of abstraction to another.
  - You built the thing right, which checks that the written requirements for both as well as formal procedures or protocols for determining compliance.

Where do bugs come from?

- Incorrect specifications
- Misinterpretation of specifications
- Misunderstandings between designers
- Missed cases
- Protocol non-conformance
- Resource conflicts
- Cycle-level timing errors
- …

Design verification techniques

- Simulation uses an executable model
  - relies on inputs.
- Formal methods generate a (possibly specialized) proof.
  - Equivalence checking
  - Model checking: description of properties are tested
  - Theorem proving
- Semi-formal
  - Simulation-based and simulation-assisted
  - Symbolic simulation
  - Directed model checking
- Manual methods, such as design reviews, catch design errors informally.

A methodology of methodologies

- Embedded systems include both hardware and software.
  - HW, SW have their own design methodologies.
- Embedded system methodologies control the overall process, HW/SW integration, etc.
  - Must take into account the good and bad points of hardware and software design methodologies used.

Useful methodologies for ESD

- Software performance analysis: Section 3.4
  - Executable specification
- Architectural optimization.
  - Single CPU : chapter 3
  - Multiple processors: chapter 5
- Hardware/software co-design: chapter 7
- On-chip Network design:
  - On-chip networks: section 5.6
  - Multi-chip networks: section 5.8
- Software verification: section 4.5
- Software tool generation:
  - Compiler generation for configurable processors: section 2.9
  - Software generation for multiprocessors: section 6.3

Joint algorithm and architecture development

- Some algorithm design is necessarily performed before platform design.
- Algorithm => software
  - Algorithm design: specific domain (signal processing, network)
  - Software design: general
  - ESD Goal: Designing an efficient, compact software
- Algorithm development can be informed by platform architecture design.
  - Performance/power/cost trade-offs.
  - Design trends over several generations.
- Algorithm designers can develop software
  - With functional simulators that run as fast as possible
  - Fast turnaround of compilation and simulation is very important to successful software development