Application Specific Integrated Circuits for Digital Signal Processing
Lecture 1

Oscar Gustafsson

Involved persons

- Oscar Gustafsson, oscar.gustafsson@liu.se, 013-28 40 59
  - Course responsible
  - Lectures
  - Examiner
- Syed Asad Alam, syed.asad.alam@liu.se
  - Exercises
  - Labs
  - Lab responsible
- Carl Ingemarsson
  - Labs

Practical details

- Course homepage:
  `http://www.isy.liu.se/en/edu/kurs/TSTE87/`
- E-mail list (make sure you are signed up)
- Eleven lectures
  - Presents the course contents
  - Final lecture: review and guest lecture (TBD)
  - Slides available on the homepage 24h in advance, if not, there will be printed ones at the lecture
- Eleven exercises
  - Problem solving
  - Problems mainly from book
  - Final session: previous exam

Practical details

- Five laborations (four hours each)
  - Use a Matlab toolbox to implement a DSP algorithm using the techniques covered in the course
  - Download lab manual from course webpage
  - Wait with printing the final lab (possibly updated)
  - Labs in groups of two (or on your own)
  - Groups of two means both students attending and paying attention
  - Possible to work outside of lab hours
  - Do the preparatory exercises (mainly reading up) and read the lab manual before the lab!
  - First labs may take more than four hours if you do not prepare
  - Some slack for the final lab
  - Electronic sign-up

Practical details

- Voluntary oral examination
  - Can give up to 5 + 5 points for the written exam
  - The points can be used to skip (parts of) the first problem
  - Free to solve as much as possible of the first problem
  - Points obtained:
    $\min\{\text{Points on first problem} + \text{Points on oral exam}, 10\}$
  - First on chapters/lectures 1–6, second on the remaining parts
  - Groups of up to six people
  - Lists for sign-up eventually available
- Written exam: 70 points maximum, 30 points for passing
- TEN1: written examination 4.5 HP (U, 3–5)
- LAB1: laboratory series 1.5 HP (pass/fail)

Course material

- Book: L. Wanhammar, DSP Integrated Circuits, Academic Press, 1999
- Out-of-print, but available as e-book (see course webpage)
- Additional material at the course homepage
- Lab compendia to be downloaded from course homepage
- Alternative/additional literature:
  - K. K. Parhi, VLSI Signal Processing
  - U. Meyer-Baese, DSP for FPGAs
- Note that the Wanhammar book is very close to the course contents
- While a similar course can be held with any of the other books, for this course Wanhammar is clearly the best choice

Course motivation

- ASIC for DSP
- ASIC
  - Application specific integrated circuit
  - Covers both traditional "ASICs" and FPGAs
- DSP
  - Digital Signal Processing
  - Computational algorithm operating on sequence of data
- Glue between other courses:
  - Algorithms: digital filters, signal theory, digital communication
  - Implementation: switching theory, digital circuits, VLSI, design of digital systems
- How to analyze and implement (DSP) algorithms in an efficient way
- "Parallel" and follow-up courses: system design, digital arithmetic, computer aided design of electronics, design of embedded DSP processors

Course contents

- DSP system design (today)
- Review of implementation technology and algorithms (possible overlap with other courses)
  - CMOS
  - FPGA
  - Representation of DSP algorithms
  - DFT/FFT
  - Digital filters
  - Multi-rate operation
- Algorithm properties
  - Operation latency
  - Degree of parallelism (minimum sampling period)
  - Computational order
- Algorithm transformations
  - Pipelining and retiming
  - Unfolding
  - Reordering of operations

Course contents

- Mapping of algorithms to hardware
  - Operation execution time
  - Scheduling
  - Resource allocation and assignment
  - Impact of design choices
- Architectures
  - Connection of processing elements and memories
  - Control
- Processing elements
  - Number representations
  - Arithmetic circuits

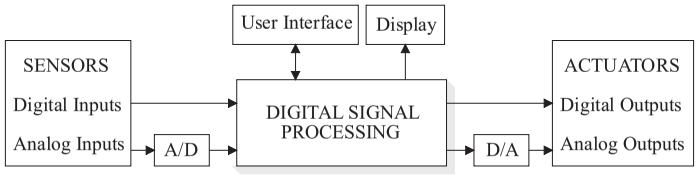
Learning goals

- Analyze computational properties of signal processing algorithms
- Determine resource requirements for implementation of signal processing algorithms
- Determine the suitability of different types of architectures for implementation of signal processing algorithms
- Synthesize nearly optimal architectures
- Realize and evaluate different types of processing elements

More learning goals

- Determine limits on the obtainable data rate
- Determine a computational order of operations
- Compute the latency and execution time of operations and determine the impact of these properties
- Determine computational graphs and optimize schedules
- Define and analyze different types of optimality in parts of the design flow
- Determine and optimize resource allocation and resource assignment
- Synthesize and analyze different architectures for signal processing systems
- Realize different types of processing elements using different arithmetic styles

DSP System Example



Reasons to use application-specific implementations

- Low power
  - Portable communication
- High throughput
  - RADAR
  - Data converter up-/downsampling
- High workload
  - Multimedia
  - Communication
- Wordlength issues
  - Cryptography (long wordlength)
  - Error-correcting codes (short wordlength)
  - DSP (only rarely 16, 24, or 32 bits are the best wordlength)

Computational complexity – system issues

- The total computational complexity is determined by the number of operations per sample times the sample rate

$$\text{operations per second} = \text{operations per sample} \times \text{sample rate}$$

- The number of operators required in an implementation is (roughly) the total computational complexity divided by the clock frequency (assuming one operation per cycle per operator)

$$\text{number of operators} \gtrapprox \frac{\text{operations per second}}{\text{clock frequency}}$$

$$\text{number of operators} \gtrapprox \frac{\text{operations per second}}{\text{operator throughput}}$$

- The clock frequency is primarily determined by the implementation technology

Computational complexity – system issues

- Cases:
  - If less than one operator is needed, the implementation is straightforward (use "processor")
  - If the clock frequency is equal to the sample rate an iso-morphic (one-to-one) mapping of the algorithm to hardware can be performed
- Interesting cases:
  - More than one operator, but fewer than an iso-morphic mapping, are needed, i.e., a time-multiplexed multi-operator architecture
  - The sample rate is higher than the clock rate, i.e., a parallel data stream architecture

Heterogenous systems

- A system typically consists of several different blocks
  - Interface circuits, data converters, etc
  - Processors (DSP and control)
  - Fixed-function units
  - Memory
  - Communication
- Focus on fixed-function units
- Either as stand-alone, SoC building blocks, or accelerators
- Being able to determine when to use what platform

Classification of DSP systems

- Real-time – Nonreal-time
- Resource adequate – Resource limited
- Recursive algorithms – Non-recursive algorithms
- Data dependent – Data independent
- Static scheduling – Dynamic scheduling of operations
- Control dominated – Data dominated
- Fixed throughput – Variable throughput
- Complexity limited – Throughput limited

Design and Implementation Constraints

- Throughput
- Energy consumption
- Size
- Flexibility
- Volume
- Design time

Design and Implementation Issues

- Skilled manpower
- Design resources
- Previous design experience
- Technology independence
- CAD tools
- Design reuse
- Correctness

Some observations

- More and more sophisticated and complex systems
- Research intensive – small step from research to application
- Broad competence in application domain, signal processing, algorithms, arithmetic, and electronics, is required
- Necessary to work in small teams at all levels of a design
- New cost measures – Design and energy efficiency
- Short design time
- Important to make it to work at first try
- Necessary to optimize energy efficiency/power consumption at all levels of the design
- "The best design is the one that works first"

Comparison of available approaches

Comparing the two "extremes"

Standard DSPs	*Feature*	**Application-specific**
Low	Design time	High
Standardized		IP blocks
Easy	Debugging	Hard
Compile-and-run		Time-consuming
		Long turn-around time
General purpose	Flexibility	Dedicated
Limited performance		Design constraint
Limited	Performance	Enough
High	Power	Low
Overhead due to flexibility		Optimized
Low initial cost (COTS)	Cost	High initial cost
High unit cost		Low unit cost

More or less continuous range of platforms

- General DSP
- Application-specific DSPs
- FPGAs
- Coarse-grained arrays
- Structured ASICs
- Standard-cell ASICs
- Full custom ASICs

Basic assumptions

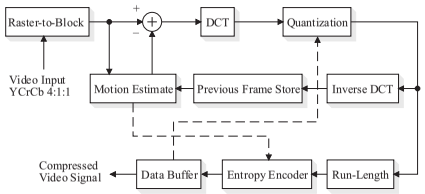
- Standard digital CMOS technology or FPGAs
- Area-speed-power trade-off
- Optimize the resource usage with respect to specifications

Complexity

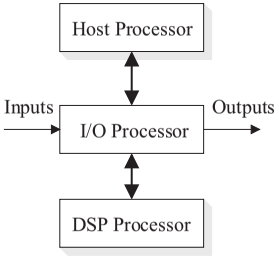
- One major obstacle in DSP system design is dealing with complexity
- The more different aspects and degrees of freedom we have to consider, the harder to make an optimal design
- Guarantee design specifications and find a (near) optimal solution
- Guarantee (a short) design time
- Structured design methodology

Dealing with complexity

- Use different representations/views
  - Behavioral/functional: How does it work?
  - Architectural/structural: What is it composed of?
  - Physical: Which components are used?



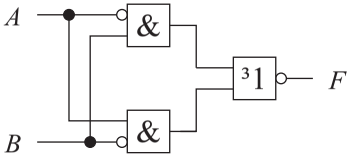
Functional (or structural)

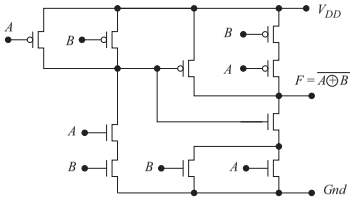


Physical

Dealing with complexity

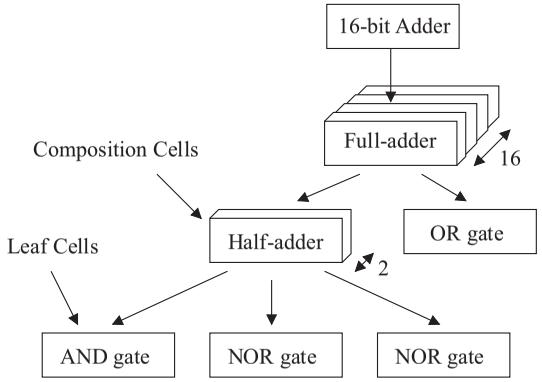
- Representation/view example: XNOR-gate
  - Behavioral/functional: $Y = A \bar{\oplus} B$
  - Architectural/structural:





Dealing with complexity

- Use hierarchies
  - Different hierarchical views useful at different steps
  - Moving down (more details) in hierarchy: synthesis
  - Moving up (less details) in hierarchy: abstraction



Dealing with complexity

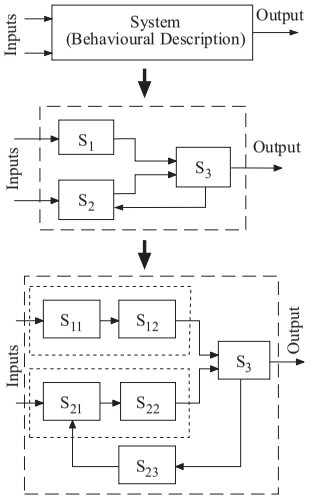
- Reuse
  - Modularity: break into smaller pieces that can be reused
  - Regularity: use identical pieces
  - Standardization: use already available pieces

System design methodology

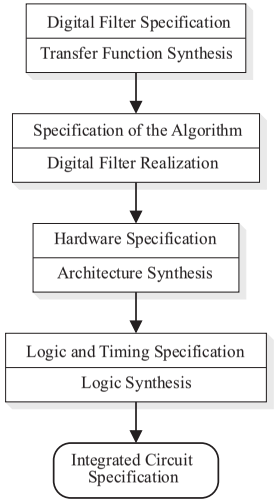
1. Problem understanding
2. Specification
   - What to design?
   - How to test?
     - Verification – formal
     - Validation – informal
3. System partitioning
   - Partitioning techniques
     - Top-down – Start from a high-level description and refine the details
     - Bottom-up – Build and assemble (LEGO)
     - Meet-in-the-middle – Start from a high-level, but eventually use existing blocks
4. More to follow

Top-down/sequence of models

- Start with a high-level model (golden model) capturing functionality
- Refine model (synthesize), introducing more detail and obtaining specifications for later models
- Compare with previous model(s)



Filter design example using top-down



Some notes on complexity

- Most problems we consider are NP-hard
  - NP = Non-deterministic polynomial-time
  - In theory: Solvable in polynomial-time on a non-deterministic Turing machine
  - In practice: not solvable in polynomial-time
  - Typically exponential complexity (exhaustive search)
- Exponential is not always bad
  - Compare the polynomial x^{10} vs the exponential 1.001^x
  - Exhaustive search is not hard for small problems
- May not need the optimal solution
  - "Good enough" is often OK
  - Possible to use heuristics
  - A heuristic is a method that finds a good solution in reasonable time ignoring if it is optimal or not