KUNGL
TEKNISKA
HÖGSKOLAN

Inte
Master Pr
in System-on-Chip Design

Advanced Logic Design

Lecturer

Elena Dubrova
ES/ICT/KTH

dubrova@kth.se
http://www.ict.kth.se/~dubrova

Teaching assistant

Shohreh Sharif Mansouri
ES/ICT/KTH

shsm@kth.se

Office hours

- Friday 12:00 – 13:00
- Send me an email with your questions

Text book

- Course notes
- Helpful, but not necessary
  - G. de Micheli, **Synthesis and Optimization of Digital Circuits**, McGraw-Hill Series**,** 1994, ISBN 0-07-016333-2
  - Eds. S. Husson and T. Sasao, **Logic Synthesis and Verification**, 2002

Course evaluation

- 5 assignments (16%)
- 4 quizzes (4%)
- project (30%)
- final exam (50%)

Lectures

- .pdf files of lecture notes will be available on my home page after each lecture
- Lecture notes are grouped and numbered according to the topic, not to the number of the lecture
- It is normally hard to pass this course without attending the lectures

Exercises

- Exercises will be given by Shohreh
- You will be solving different tasks
- The purpose is to solidify material of the lectures and make you think deeper about concepts
- You can tell me on the lecture which task would you like to do on the exercises (or send email)

Assignments

- 5-7 assignments (16% of the mark)
  - Some smaller (one task only), some larger
  - to be handled on the due date and time
    - Either give me or Shohreh a hard copy on the lecture/exercise
    - Or, send Shohreh an email with your assignment
  - no exceptions; late assignment = 0 points

Quizes

- 4 quizzes (1 point each = 4% of the mark)
  - each consists of a single task
  - easy, but should be done quickly
  - will be done in class, during the first 5 min of the lecture
  - homepage of the course will inform you in advance when the quizzes will be done

Project

- Worth 30% of the final mark
  - Design an algorithm which solves a given problem
  - Program it C or C++
  - Done is groups of 2 people, or individually
  - I will proposes several different topics during the course

PhD students

- PhD students have a choice to do a more challenging individual project
  - You will need to learn some additional material (read some research papers, etc) to make it
- The purpose is to create something original, possibly leading to a publication

Objectives of the course

- To get an idea about the algorithms used in building commercial computer-aided design tools like Synopsis, Cadence, etc.
- To build sufficient background such that one can understand research papers in the area
- To understand why these algorithms are considered computationally expensive and why commercial tools are so costly

Overview

- Introduction to CAD tools
  - design methodology
  - computational complexity
  - functions, Boolean algebra
- Data structures
  - Boolean cubes, logic circuits, BDDs
- Algorithms
  - two- and multiple-level synthesis and optimization
  - technology mapping
  - formal verification

KUNGL
TEKNISKA
HÖGSKOLAN

Inte
Master Pr
in System-on-Chip Design

L1: Introduction to CAD tools

Reading material

- de Micheli pp. 12 - 35

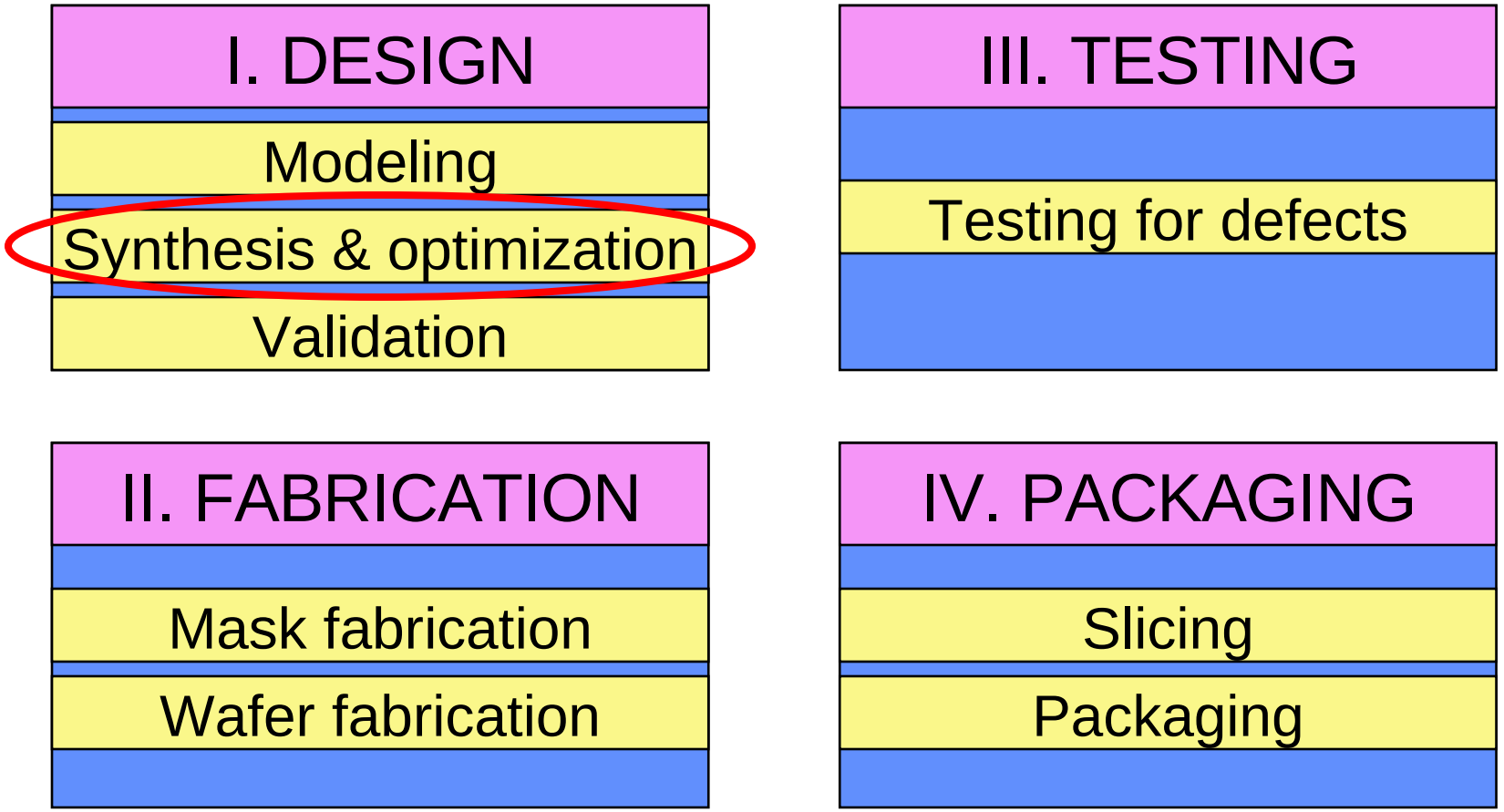
CAD

- Computer-aided design (CAD) of digital circuit has been a topic of great importance in the last two decades
- Many CAD tools are mature. However, since technology changes constantly, new problems arise, thus new solutions are needed
- Examples:
  - Technology mapping for FPGAs with “dual” LUTs
  - Logic synthesis for new types of gates (bio, quantum, ...)

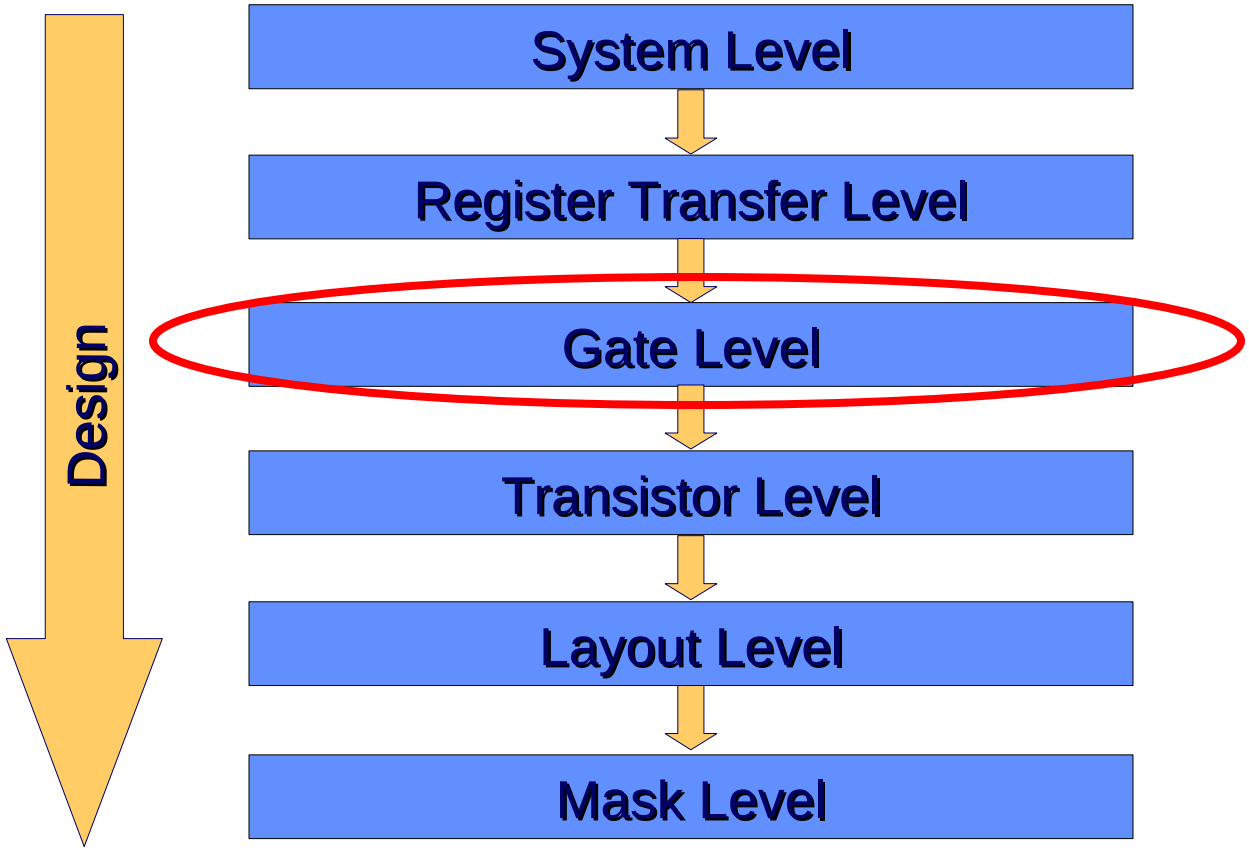
What we can do with CAD tools

- CAD tools allow to design efficiently and successfully large-scale high-performance integrated circuits for a wide spectrum of applications
  - computers
  - telecommunications
  - transportation
  - ...

Four stages in creation of an IC

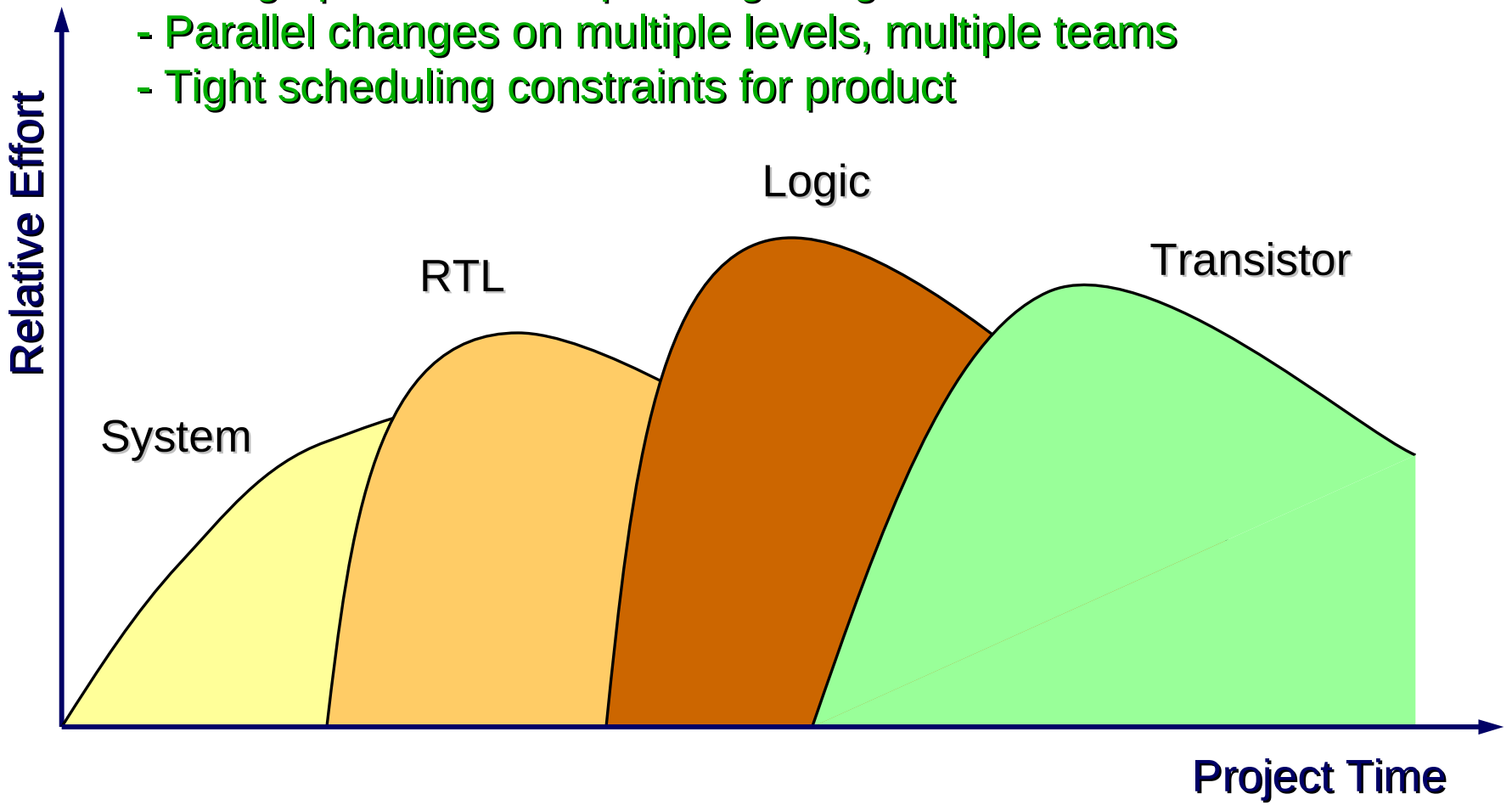


Design of Integrated Systems



Design of Integrated Systems

- Design phases overlap to large degrees
- Parallel changes on multiple levels, multiple teams
- Tight scheduling constraints for product



Design challenges

- Systems are becoming huge, design schedules are getting tighter
  - > 20 Mio gates becoming common for ASICs
  - > 0.4 Mio lines of C-code to describe system behavior
  - > 5 Mio lines of RTL code

- Design teams are getting very large for big projects
  - several hundred people
  - differences in skills
  - concurrent work on multiple levels
  - management of design complexity and communication very difficult

Historical prospective

- The earliest CAD tools targeted automatic layout at physical level of abstraction (1960s, Engineering Design System, IBM).
- As Large-scale integration (LSI) became possible, the CAD-tools shifted to transistor- and logic-levels of abstraction (MINI, IBM, 1970s)
- Logic synthesis tools became popular in the early 1980s, as VLSI technology matured (Multiple-level Interactive System MIS, Berkeley University)
- The early high-level synthesis systems were not coupled with logic synthesis tools (Alert, IBM)

Historical prospective (cont.)

- The first fully integrated systems were IBM's Yorktown Silicon Compiler and the CATHEDRAL system developed at the Catholic University of Leuven, Belgium.
- Fully integrated CAD system are normally developed for a specific application and implementation styles, e.g. CATHEDRAL transforms behavioral model of a particular class of designs (digital signal processors) into circuits with particular design styles

State-of-art: Silicon compiler

- The impact of CAD tools has been extremely positive. Some circuits could never have been designed at the required performance level without CAD tools
- The market growth for CAD tools is very impressive, especially for synthesis tools at higher levels of abstraction
- Analogy with compilers for programming languages: from algorithm to machine instructions
  - in VLSI: from algorithm to mask patterns
- Current practice:
  - this ideal is approached more and more
  - still far away from a single push-button operation

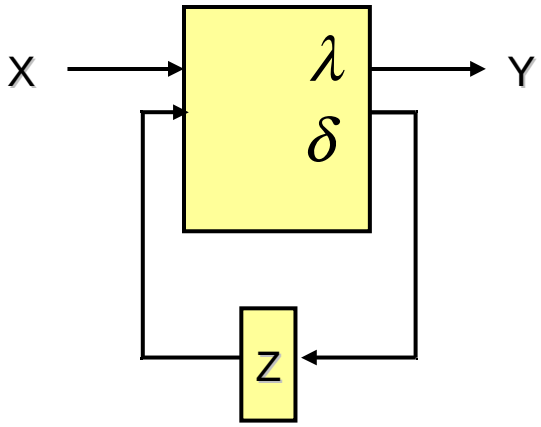
Focus of this course

- CAD tools for synthesis and verification at logic-level of abstraction
- Theory behind: functions representation and manipulation
  - representation ⇔ data structures
  - manipulation ⇔ algorithms
- In-depth course:
  - you should be able create a small CAD-tool

Why logic level?

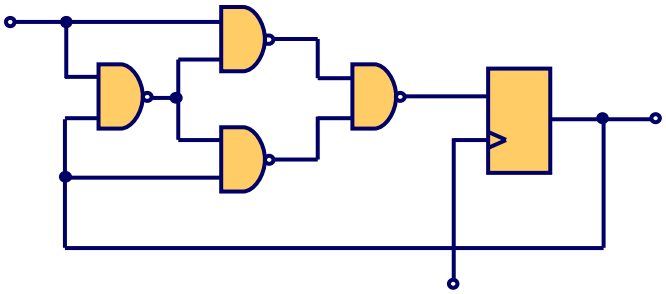
- Logic-level synthesis is the core of today's CAD flows for IC and system design
  - course covers many algorithms and data structures that are used in a broad range of CAD tools
  - basis for other optimization techniques
  - basis for functional verification techniques
- Most algorithms are computationally hard
  - covered algorithms and flows are good example for approaching hard algorithmic problems
  - course covers theory as well as implementation details
  - demonstrates an engineering approaches based on theoretical solid but also practical solutions
    - very few research areas can offer this combination

What is Logic Synthesis?



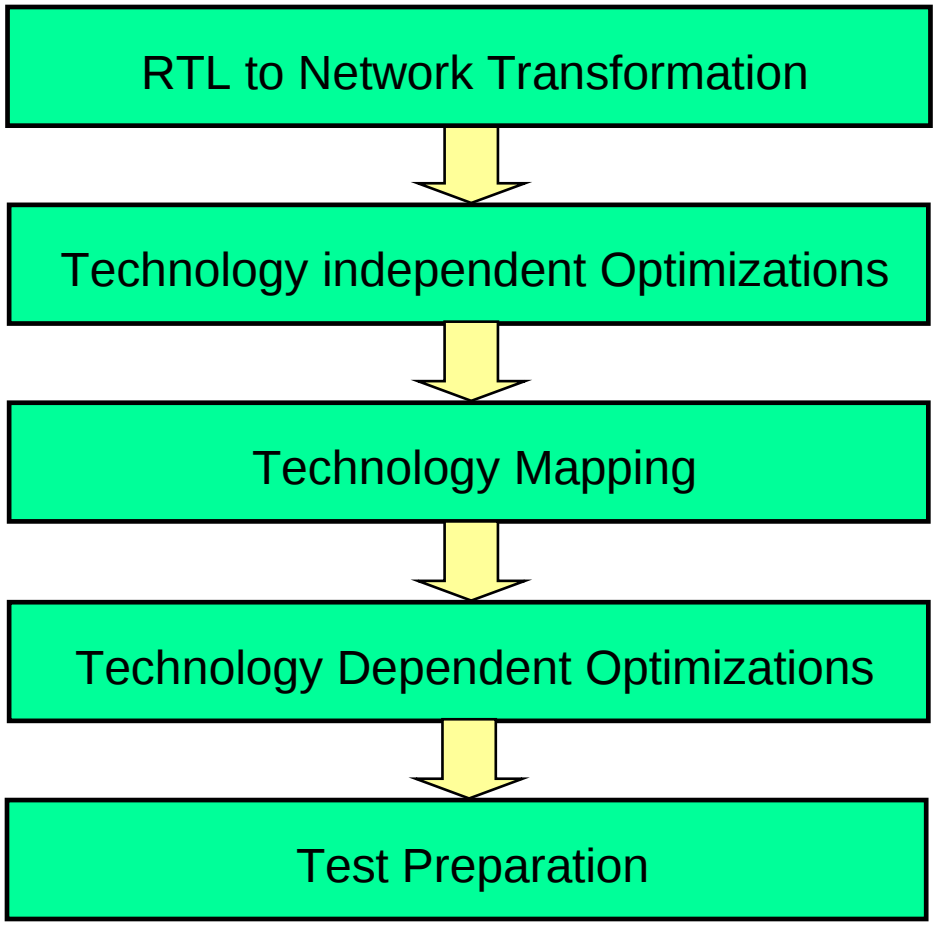
Given: Finite-State Machine F(X,Y,Z,λ ,δ) where:

X: Input alphabet
Y: Output alphabet
Z: Set of internal states
λ : X x Z → Z (next state function)
δ : X x Z → Y (output function)



Target: Circuit C(G, W) where
G: set of circuit components g (Boolean gates flip-flops, etc}
W: set of wires connecting G

Typical logic synthesis scenario



- read Verilog/VHDL

- basic logic restructuring

- use logic gates from target cell library

- timing optimization
- physically driven optimizations

- improve testability
- test logic insertion

Objective function for logic synthesis

- Minimize area
  - in terms of literal count, gate count, register count, etc.
- Minimize power
  - in terms of switching activity in individual gates, blocks, etc.
- Maximize performance
  - in terms of maximal clock frequency of synchronous systems, throughput for asynchronous systems
- Any combination of the above
  - combined with different weights
  - formulated as a constraint problem
    - “minimize area for a clock speed > 300MHz”

Constraints on synthesis

- Given implementation style:
  - two-level implementation (PLA)
  - multi-level logic (ASIC or FPGA)
- Given performance requirements
  - minimal clock speed requirement
- Given cell library
  - set of cells in standard cell library
  - fan-out constraints (maximum number of gates connected to another gate)

Mapping functions into gates

- One of the main steps of logic synthesis is to map a given Boolean functions into a combinational circuit with the minimum “cost”
- This task is VERY hard because:
  - Usually, many different circuits can implement the same function
    - How to search for a “best circuit”: (1) construct one circuit and try to improve it, or (2) directly construct the best
    - How to decide which gates to use (AND, OR, NOT, XOR, …)
    - How to evaluate “cost” (gates, variables, …)

Assignment 1 (2 points, due Oct 29)

- Find a best circuit for following the 4-variable Boolean function:

x_3x_4 \ x_1x_2	00	01	11	10
00	0	1	1	1
01	0	1	1	1
11	1	0	0	0
10	0	1	1	1

- Define as “best” a circuit with the smallest number of 2-input gates
- Define “gate” as AND, OR, or XOR, same cost for all
- Assume that NOT has 0 cost

- Evaluation: 2 for optimal, 1 for non-optimal, 0 for incorrect

Related problems I

- How to *enumerate* all possible “candidates into best“ circuits implementing a given function?
  - We need to put restrictions of the type of gates, otherwise it is not possible
    - For example, we can restrict all gates to be 2-input
    - “Gate” can be defined as on the previous slide
- Suppose that you found an algorithm which solves the above problem and programmed it
  - For how many variables your program will be feasible? (feasible = terminates in < 12 hours)

Related problems I, cont.

- The problem of enumerating all candidates into best circuits implementing a given functions would be a great project topic if you can find an algorithm which is feasible for functions of at least 5 variables
  - Try to decide what is the largest number of gates which will be needed to implement an n-variable function
    - For example, use induction (one gate for n = 2, …)

Related problems II

- Rather then enumerating all possible circuits for a given function *f*, we can start from some circuit and then try to improve it incrementally
- Think about the search space as a connected graph in which each vertex is a different circuit implementing *f*
- Two vertices are connected by an edge if one can be obtained from another by a simple *local transformation*
  - For example, replace a sub-circuit ab + ac by a(b +c)

Related problems II, cont

- The key problem is to get a “complete” set of local transformations, which would allow you to move between any two vertices in the search space
- In this case, you are guaranteed to find the best if you search long enough
  - Guided random search algorithms (simulated annealing, genetic, etc.) can be used
- This strategy can be applied to large functions
- Used in the first IBM logic synthesis tool, LSS

Related problems II, cont

- A possible interesting project topic:

  Design an algorithm which improves a circuit incrementally using local transformations
- Define “gate” as before
- Your algorithm will be interesting if it is feasible for functions of at least 32 variables
- A related paper you may want to read:

  Integrated Logic Synthesis Using Simulated Annealing, P. Farm, E. Dubrova, A. Kuehlmann, *Notes of International Workshop on Logic Synthesis,* 2006.

Next lecture

- Computational complexity