

ECE199JL: Introduction to Computer Engineering **Fall 2012**
Notes Set 2.8

Summary of Part 2 of the Course

The difficulty of learning depends on the type of task involved. Remembering new terminology is relatively easy, while applying the ideas underlying design decisions shown by example to new problems posed as human tasks is relatively hard. In this short summary, we give you lists at several levels of difficulty of what we expect you to be able to do as a result of the last few weeks of studying (reading, listening, doing homework, discussing your understanding with your classmates, and so forth).

We'll start with the easy stuff. You should recognize all of these terms and be able to explain what they mean. For the specific circuits, you should be able to draw them and explain how they work. Actually, we don't care whether you can draw something from memory—a full adder, for example—provided that you know what a full adder does and can derive a gate diagram correctly for one in a few minutes. Higher-level skills are much more valuable. (You may skip the *'d terms in Fall 2012.)

- Boolean functions and logic gates
  - NOT/inverter
  - AND
  - OR
  - XOR
  - NAND
  - NOR
  - XNOR
  - majority function
- specific logic circuits
  - full adder
  - ripple carry adder
  - $\bar{R}$-$\bar{S}$ latch
  - R-S latch
  - gated D latch
  - master-slave implementation of a positive edge-triggered D flip-flop
  - (bidirectional) shift register*
  - register supporting parallel load*
- design metrics
  - metric
  - optimal
  - heuristic
  - constraints
  - power, area/cost, performance
  - computer-aided design (CAD) tools
  - gate delay
- general math concepts
  - canonical form
  - N-dimensional hypercube
- tools for solving logic problems
  - truth table
  - Karnaugh map (K-map)
  - implicant
  - prime implicant
  - bit-slicing
  - timing diagram
- device technology
  - complementary metal-oxide semiconductor (CMOS)
  - field effect transistor (FET)
  - transistor gate, source, drain
- Boolean logic terms
  - literal
  - algebraic properties
  - dual form, principle of duality
  - sum, product
  - minterm, maxterm
  - sum-of-products (SOP)
  - product-of-sums (POS)
  - canonical sum/SOP form
  - canonical product/POS form
  - logical equivalence
- digital systems terms
  - word size
  - N-bit Gray code
  - combinational/combinatorial logic
    - two-level logic
    - "don't care" outputs (x's)
  - sequential logic
    - state
    - active low inputs
    - set a bit (to 1)
    - reset a bit (to 0)
    - master-slave implementation
    - positive edge-triggered
  - clock signal
    - square wave
    - rising/positive clock edge
    - falling/negative clock edge
    - clock gating
  - clocked synchronous sequential circuits
  - parallel/serial load of registers*
  - glue logic*
  - logical/arithmetic/cyclic shift*

We expect you to be able to exercise the following skills:

- Design a CMOS gate from n-type and p-type transistors.
- Apply DeMorgan's laws repeatedly to simplify the form of the complement of a Boolean expression.
- Use a K-map to find a reasonable expression for a Boolean function (for example, in POS or SOP form with the minimal number of terms).
- More generally, translate Boolean logic functions among concise algebraic, truth table, K-map, and canonical (minterm/maxterm) forms.

When designing combinational logic, we expect you to be able to apply the following design strategies:

- Make use of human algorithms (for example, multiplication from addition).
- Determine whether a bit-sliced approach is applicable, and, if so, make use of one.
- Break truth tables into parts so as to solve each part of a function separately.
- Make use of known abstractions (adders, comparators, or other abstractions available to you) to simplify the problem.

And, at the highest level, we expect that you will be able to do the following:

- Understand and be able to reason at a high-level about circuit design tradeoffs between area/cost and performance (and that power is also important, but we haven't given you any quantification methods).
- Understand the tradeoffs typically made to develop bit-sliced designs—typically, bit-sliced designs are simpler but bigger and slower—and how one can develop variants between the extremes of the bit-sliced approach and optimization of functions specific to an N-bit design.
- Understand the pitfalls of marking a function's value as "don't care" for some input combinations, and recognize that implementations do not produce "don't care."
- Understand the tradeoffs involved in selecting a representation for communicating information between elements in a design, such as the bit slices in a bit-sliced design.
- Explain the operation of a latch or a flip-flop, particularly in terms of the bistable states used to hold a bit.
- Understand and be able to articulate the value of the clocked synchronous design abstraction.