

ECE199JL: Introduction to Computer Engineering **Fall 2012**
Notes Set 3.4

Summary of Part 3 of the Course

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).

Note that the homeworks in this part of the course have been more challenging in terms of time, so you may expect that the problems on the exam will be similar in nature but designed to require less actual time to solve (assuming that you have been doing the homeworks).

We'll start with the easy stuff. We have pulled in the register terminology from Part 2 (marked with "+"), since we did not cover those on the last midterm. 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 mux, for example—provided that you know what a mux does and can derive a gate diagram correctly for one in a few minutes. Higher-level skills are much more valuable.

Since we didn't deliver notes on the memory lecture (yet!), some items are *'d—these will not appear on the midterm.

- digital systems terms
  - parallel/serial load of registers+
  - glue logic+
  - logical/arithmetic/cyclic shift+
  - modules
  - fan-in
  - fan-out
  - machine models: Moore and Mealy
  - parity*
  - Hamming distance*
- simple state machines
  - synchronous counter
  - ripple counter
  - serialization (of bit-sliced design)
- finite state machines (FSMs)
  - states and state representation
  - transition rule
  - self-loop
  - meaning of don't care in input combination
  - meaning of don't care in output
  - unused states and initialization
  - completeness (with regard to FSM specification)
  - list of (abstract) states
  - next-state table/state transition table/state table
  - state transition diagram/transition diagram/state diagram
- memory
  - design as a collection of latches
  - number of addresses
  - addressability
  - read/write logic
  - serial/random access memory (RAM)*
  - volatile/non-volatile (N-V)*
  - static/dynamic RAM (SRAM/DRAM)*
  - SRAM cell*
  - DRAM cell*
  - bit lines and sense amplifiers*
- von Neumann model
  - processing unit
    - register file
    - arithmetic logic unit (ALU)
    - word size
  - control unit
    - program counter (PC)
    - instruction register (IR)
    - implementation as FSM
  - input and output units
  - memory
    - memory address register (MAR)
    - memory data register (MDR)
- specific logic circuits
  - (bidirectional) shift register+
  - register supporting parallel load+
  - N-to-M multiplexer (mux)
  - N-to-2^N decoder
  - tri-state buffer
    - meaning of Z/hi-Z output
    - use in distributed mux

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

- Transform a bit-sliced design into a serial design, and explain the tradeoffs involved in terms of area and time required to compute a result.
- Based on a transition diagram, implement a synchronous counter from flip-flops and logic gates.
- Implement a binary ripple counter (but not necessarily a more general type of ripple counter) from flip-flops and logic gates.
- Given an FSM implemented as digital logic, analyze the FSM to produce a state transition diagram.
- Design an FSM to meet an abstract specification for a task, including production of specified output signals, and possibly including selection of appropriate inputs.
- Complete the specification of an FSM by ensuring that each state includes a transition rule for every possible input combination.

When designing a finite state machine, we expect you to be able to apply the following design strategies:

- Abstract design symmetries from an FSM specification in order to simplify the implementation.
- Make use of a high-level state design, possibly with many sub-states in each high-level state, to simplify the implementation.
- Use counters to insert time-based transitions between states (such as timeouts).
- Implement an FSM using logic components such as registers, counters, comparators, and adders as building blocks.

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

- Explain the difference between the Moore and Mealy machine models, as well as why you might find each of them useful when designing an FSM.
- Understand the need for initialization of an FSM, be able to analyze and identify potential problems arising from lack of initialization, and be able to extend an implementation to include initialization to an appropriate state when necessary.
- Understand how the choice of internal state bits for an FSM can affect the complexity of the implementation of next-state and output logic, and be able to select a reasonable state assignment.
- Identify and fix design flaws in FSMs by analyzing an existing implementation, comparing it with the specification, and removing any differences by making any necessary changes to the implementation.