

## ECE 2030: Study Guide

### 1. Computing Systems

- Be able to state Moore's Law and its basic consequences (read the web page)
- Conversion between decimal, octal, hexadecimal, and binary number systems
- Basic components of a computing system
- Abstraction hierarchy: from transistors (switches) to data paths
  - You should know what the components are at each level

### 2. Switch Design

- Can you distinguish between shorts and floats? From a switch diagram?
  - What does high impedance or float mean?
- Translate between Boolean expressions and switch level implementations
- Be able to design a pull-up network or pull-down network from a Boolean expression
- Be able to produce the dual of a switch network built with n-type (p-type) switches
- Produce switch implementations of all of the basic gates: AND, NAND, NOR, OR, INVERTER, XOR, XNOR
  - Including gates with more than two inputs
- Understand the physical properties of transistor devices
  - RC model, rise time and fall time
  - Propagation delay, fan-in, fan-out, concept of load
  - Relationship between load and propagation delay
  - What is switching power and leakage power?

### 3. Boolean Logic

- Be able to translate between Boolean expressions and gate level designs (not using mixed logic)
- Simplify Boolean expressions algebraically using the principal Boolean identities
- Understand the concept of universal gates and be able to demonstrate (algebraically) that NANDs and NORs are universal gates
- Understand standard forms for Boolean expressions
  - Be able to write the sum of minterms or product of maxterms form of a Boolean expression from its truth table representation
  - Be able to convert between sum of minterms form and product of maxterms form for an expression algebraically
  - Algebraically simplify the sum of minterms or product of maxterms form
- Understand sum of product (SOP) and product of sum (POS) forms of Boolean expressions

- How do these forms differ from sum of minterms and product of maxterms forms
- Understand the dual of a Boolean function and be able to compute it
  - Note: taking the dual and complementing each literal is a short cut to computing the complement of a Boolean function
    - Using the preceding observation compute the dual of Boolean expression
- Compare two Boolean expressions and comment on the relative complexity of the gate level or transistor level implementation?
  - Complexity is defined as the number of gates/switches/literals required to compute the function
- Be able to draw basic timing diagrams for small combinational logic circuits.
  - Be able to find the critical path in simple combinational circuits

#### 4. Gate Design

- Understand the terminology of inverting gates and non-inverting gates
  - What is negative logic?
  - What is a mixed logic gate?
- Understand the difference between active-low and active high signals
  - Be able to write the truth table for a basic gate (using 0,1) if one of the inputs is active low
- Know all pairs of DeMorgan's equivalent gates and DeMorgan's square (reference ECE 2030 Reading page handout on mixed logic)
- Using mixed logic, translate from a Boolean expression to a gate level implementation based on only
  - NOR gates
  - NAND gates
  - NOT and Inverter gates
  - NAND and inverter gates
  - AND gates inverter gates (not something we usually want to do in practice)
  - OR Gates and inverter gates (not something we usually want to do in practice)
- Be able to read the Boolean expression given a mixed logic design
- Be able to include active low signals in the translations between Boolean expressions and gate designs using mixed logic notation.
- Determine the number of transistors required to implement a gate level design specified in mixed logic notation
- Understand why we use mixed logic?

#### 5. Simplifying Expressions

- Understand
  - Mapping from minterms and maxterms to the 2,3, and 4 variable K-map
    - Be able to write the minterm and maxterms expressions for each entry in a K-map

- Mapping from the truth table to the 2,3, and 4 variable K-map and vice versa
- Mapping from a truth table with don't care symbols (X) to the 2, 3, and 4 variable K-map
- From an arbitrary Boolean expression create the 2,3, and 4 variable K-map
- Be able to identify and understand the differences between an implicant, prime implicant, and essential prime implicant.
- From any K-map, including one with don't care entries, write
  - The optimized Boolean SOP expression
  - The optimized Boolean POS expression
- Given the preceding skills, now from an arbitrary Boolean expression you should be able to
  - Simplify this expression using k-maps
  - Create a NAND or NOR implementation of the optimized expression using mixed logic
  - Create a switch level implementation of the optimized expression using n- and p-type switches

## 6. Building Blocks

- Understand the operation of Multiplexers and Demultiplexors. Be able to implement both with basic gates or pass gates.
- Understand the operation and implementation of decoders and priority encoders. Be able to implement arbitrary priorities of the inputs.
- Demonstrate ability to implement arbitrary truth tables with multiplexors/decoders.
- Given a network of interconnected building blocks, be able to
  - analyze the connectivity given the value of the control signals
  - determine the values of the control signals to realize a certain connectivity between inputs and outputs
- Construct larger building blocks from smaller building blocks
- Given a PLA/PAL and a Boolean expression to be implemented, show the connections to be blown to realize this connectivity
- Be able to draw a small PLA/PAL/ROM and in the case of a ROM, be able to identify the connections to be made to initialize the contents to a given value.
- Be able to create a memory based implementation of a truth table or simple gate level or switch level circuit (i.e., be able to implement the truth table). The implementation can be a ROM or RAM based implementation.

## 7. Number Systems

- Be able to translate between binary, octal, decimal, and hexadecimal notations.
- Be able to translate between fixed-point binary and decimal notations.
- Know the ranges and precision of n-bit unsigned integer, sign two's complement, fixed and floating point representations

- Translate between floating point and decimal representations
- Know both single and double precision formats as well as ranges
- How do you compare two floating point numbers (for less than or greater than) using only an integer ALU?
- Which of these laws apply to floating point arithmetic: associatively and commutatively. Be able to provide examples
- Given a significant number of digits be able to compute the error for floating point addition or multiplication of two numbers.
- Given a byte stream and the ASCII table be able to decode the byte stream

## 8. Arithmetic

- Addition/subtraction, and overflow detection for two complement and sign-magnitude arithmetic
- Overflow detection: condition and hardware implementation
- Be able to draw the gate level implementations of a full adder
- Define and implement the behavior of a full adder and subtractor. Cascade to a word-wide design.
- Implement a word-wide adder/subtractor using only full adders.
- Be able to draw an implementation of a multiplier using gates and a full adder
- Detection for overflow, negative, and non-zero in a n-bit adder/subtractor
- Calculate the delay (in gate delays) for an n-bit ripple carry adder/subtractor

## 9. Latches and Registers

- Implement a transparent latch using basic gates and pass gates.
- Implement a register with read and write enables using a two-phase non-overlapping clock.
- Understand the difference between a latch and a register.
- Be able to draw the timing diagrams for groups of latches or registers
- Be able to draw the implementation of a shift register capable of left and right shifts, left and right rotate operations, and parallel/serial load. You can use gates and registers as building blocks
- Be able to draw a 10 transistor D-Latch.
- Draw a register with basic gates that includes asynchronous set and preset.
- Draw the implementation of a register file using multi-bit registers, building blocks and basic gates.
- Understand two-phase vs. edge triggered designs. Be able to draw examples of both using basic gates.

## 10. Counters

- Be able to draw a toggle cell and use it to build counters
  - Initialize counters to specific values
  - Construct modulo- $k$  counters
- Design of synchronous up/down counters

- State diagrams for various types of counters
  - Count-by- $k$
  - Ring counter
  - Design of these counters using state machine approaches
- Use register design approaches for serial/parallel load for initialization

## 11. State Machines

- Be able to draw a state diagram from a problem specification (this is admittedly dependent on the problem description). Get in the habit about thinking of the operation of systems as a sequence of states and transitions between states. Specifically from a problem description be able to
  - Define the states
  - Define the state transitions and draw them
    - Describe the conditions for a transition in terms of Boolean variables: state transitions depend on input values and the current state
    - Define the values for each output variable associated with each state transition
    - Label the state transitions with values of input/output variables (in the case of a Moore machine there are no output variables)
  - Pick a binary encoding for the states
  - Pick a binary encoding for the input variables
- Understand the difference between a Moore and a Mealy state machine.
- Translate between state diagrams and state transition tables.
- Implement the combinational logic for each output variable from the state transition table and for the next value of each state element.
- Draw the gate level implementation of the state machine
  - Gate level description of the combinational logic
  - Use latches/flip flops for the state elements
- Given a state machine diagram, be able to figure out what it does, i.e., given an initial state what are the outputs at each clock cycle and what is the state at each clock cycle.
  - Create the state transition table from the state machine diagram
- What is the difference between asynchronous and synchronous state machines?
- What is the impact of different state encodings?
- Handling undefined states and initialization

## 12. Memory

- Understand memory cell behavior and how it is different from a register cell.
- Understand static (6-transistor) and dynamic (1-transistor) implementations of memory cell.

- Know the organization of a memory chip as an array of memory cells: row and column addressing decoding, bidirectional data bus. What do the terms RAS and CAS mean?
- Use memory chips (e.g., 4 M addresses by 4 bits/address) to build larger memory systems. Given a target system design (number of addresses, number of bits per address, and a memory chip building block), define the implementation in terms of number, organization and operation of chips with the addition of external building blocks as necessary (e.g., decoders).
- Identify chips responsible for a specified memory location in a memory system.
  - Given a multi-chip design and an address, what locations (addresses) in which chip(s) are accessed?
- Know byte, half-word, word, double word and  $2^k$  byte boundaries.
  - What does alignment mean?
- Given a set of memory directives, provide the contents of memory (encoded in hex).
- Given a memory system designed with multiple chips (for example a 2M addresses x 16 bits using 1M address x 4 bit chips) and a set of memory directives, provide the hex contents of specific locations in a specific chip.
- Given the memory contents and an interpretation such as little endian or big endian ordering, provide the values that are accessed from memory as a word, half word or double word.

### 13. Datapath Units

- Implement a register file from word-wide registers and decoders.
- Implement a word-wide logical unit using multiplexers. How is the meaning of all 16, possible logical operations defined?
- Implement a 1-bit, 2-bit, and 4-bit left/right shifter from multiplexers. Cascade to a 32 bit barrel shifter. Show how magnitude and direction can be extracted from a six-bit shift value.
- Understand the difference between logical, arithmetic, and rotate shifts.
- Understand the operation of simple three bus datapath that may contain register files, arithmetic units (adder/subtractor, multiplier/divider), logical units, shift units, and memory.
- Be able to translate simple programs into sequences of operations on a single cycle datapath. Define all control signals for each cycle.
- Understand and employ immediate values in programs that require them.
- Be able to translate sequences of operations into microcode

### 14. Controllers & Instructions

- Explain operation of a controller including the instruction pointers, code memory, instruction decode, and next instruction computation.
- Explain how an instruction set architecture separates issues of programming from issues of system implementation.

- Know how fields associated with opcode, register operands, and immediate operands can be decoded into control signals for the single cycle datapath.
- Using a MIPS-like ISA, be able to write simple straight-line assembly code sequences using the expanded single cycle datapath.
- Creating a 32 immediate value in a register.

## **15. Branching and Procedures**

- Use conditional branches are used to implement if-then-else and loop constructs in assembly language programs.
- Use SLT and SLTI to implement all comparative branch types.
- Understand how conditional branches can be implemented in the single cycle datapath.
- Use definition of BEQ and BNE to compute word and byte branch offsets for forward and backward branches.
- Understand the branch range of the branch and jump instructions. Know the difference between absolute and relative branching.
- Explain the need for subroutines in assembly language programs.
- Compute the instruction fields and return address for a jump and link instruction and jump to register instructions.
- Use j, jal, and jr instructions to write subroutines.
- Define and use a stack in memory; know definition of push and pop operations.