Chapter 1:

- Key differences between general purpose computing design and embedded computing design.
  - Why is embedded design more difficult?
  - What techniques have been adopted from general purpose computing and applied to embedded computing design?
- Large design spaces for embedded systems:
  - Why so large?
  - Why is it beneficial?
  - What are the advantages and disadvantages?
- Real time systems
  - What are they?
  - What challenges do they present?
- Hardware/software co-design
  - What is it?
  - What challenges does it present?
  - What benefits does it present?
- Example applications: Radio and networking, multimedia, vehicle control and operation, and sensor networks
  - What unique design challenges exist for each?
  - What unique optimization techniques exist for each?
  - Vehicle control and operation: why are two separate networks necessary?
  - Sensor nodes: why is onboard processing so important? What challenges exist in revealing onboard processing?
- Functional and non-functional requirements
- Design goals
- Why are performance, power, and energy judged in terms of average, peak and work case?
- Why are design methodologies so important for embedded systems?
- Designer productivity gap – What is it and methods to bridge it
- Waterfall vs. spiral design methodology
- During the design process, why are early and accurate estimates so important?
- Platform design vs. hardware/software co-design
  - Challenges for each
  - How are the similar/different
  - Benefits for each
- Techniques used to verify design
- Why is it important to verify a design at each level of abstraction?
- General embedded system design methodologies (page 32)
- Models of computation
  - Why is it important to study models of computation?
  - Are all models appropriate for all systems? How do you choose an appropriate model? Given a type/class of application, what modes are most appropriate?
  - Know the basics for the models discussed in class: FSM, control flow, data flow, parallel models (task graphs, petri nets).
- Sources of parallelism: instruction and data level, task level
  - How can these be exploited?
  - Which models of computation are best for expressing each type of parallelism?
- What does it mean for a system to be reliable, safety critical, and/or secure?
- What unique security challenges to embedded systems have?
- Permanent vs. transient faults
• Different sources of faults
• What is MTTF? What does it tell us?
• What can a system do after a fault? (page 50)
• Key chapter questions:
  o Q1-1, Q1-3, Q1-7, Q1-9, Q1-10, Q1-12, Q1-14, Q1-16, Q1-19, Q1-20, L1-1

Chapter 2:

• How is CPU design for embedded systems different from general purpose processors?
• Metrics used to evaluate processors
• Processor taxonomy
• RISC vs. CISC
• Key architectural features of DSPs
• Static vs. dynamic parallel mechanisms
• VLIW
  o What is it?
  o Advantages and disadvantages: in general and with respect to superscalar processing
  o Register file partitioning
    ▪ Purpose of it
    ▪ What architectural support is needed?
    ▪ Advantages and disadvantages
  o What applications are VLIW good for?
• Superscalar
  o What is it?
  o Advantages and disadvantages: in general and with respect to superscalar processing
• Subword parallelism
• Threadlevel parallelism
  o Hardware multithreading vs. simultaneous multithreading
• Dynamic voltage scaling (DVS) and dynamic voltage and frequency scaling (DVFS)
  o What does it do?
  o How does it work?
  o Benefits?
  o Better than worst case design: Razor architecture
• Register file size vs. application needs. Why is a specialized register file size beneficial?
  o Spilling?
• Why are caches so important?
  o How can they be specialized?
  o How do cache aspects, such as size, line size and associativity affect an application’s performance?
  o Configurable caches
• Scratch pad memories
  o What are they?
  o Why are they good for real time systems?
  o How do they work?
• Code compression
  o How to generate compressed code
  o Architectural layout (i.e. pre-cache vs. post-cache decompression) advantages and disadvantages
  o Difficulties
  o Benefits
  o Compare and contrast basic methods discussed in class (e.g. dictionary, Huffman-based, arithmetic encoding). Advantages and disadvantages of each in general and with respect to each other
  o How does block size affect compression ratio?
- Branches
  - Difficulties
  - Solutions
- Data compression
  - Why is data compression harder than instruction compression?
- Low power bus encoding
  - Basic concept and purpose
  - Bus invert coding
  - Working zone bus encoding
- CPU simulation classification methods (Page 126)
- Basic differences between embedded (i.e. EEMBC) and desktop benchmark (i.e. SPEC) suites
- CPU simulation methods: Trace-based analysis, direct execution, microarchitecture modeling
  - Compare and contrast methods
  - What are each most appropriate for
  - PC sampling techniques
  - Instruction instrumentation
  - Power simulators
- Automated CPU design
  - What is it?
  - What is it used for?
  - Why is it difficult?
  - What special tools are required?
- Different methods to customize processors (page 133)
  - Benefits and purpose for each type
- What are ASIPs?
- Instruction set synthesis
  - Basic concept and motivation
  - What benefits does it afford?
- Key chapter questions:
  - Q2-5, Q2-10, Q2-11, Q2-12, Q2-13, Q2-14

Chapter 3:

- Know the major steps for code generation and the basic concept/idea behind each
- Instruction selection
  - What does it mean for one instruction to “cover” other instructions. Give an example
  - How does instruction selection optimize the program/application?
- Register allocation
  - Given a piece of code, show register lifetimes, draw conflict graph, and allocate the optimal number of registers, showing all register sharing (CH3-1: slides 8, 9, 10)
- Code Placement
  - How does code placement affect performance? Code size?
  - What portions of code are the best target for code placement and why? How can information be gathered to determine these regions of code?
  - Procedure inlining
    - What is it?
    - What are the benefits?
- Memory oriented optimizations
  - Loop transformations
    - Purpose
    - Loop carried dependencies?
    - List potential loop transformations (page 172), define, and do an example
- General strategies for optimizing compilers (page 176)
  - List, define, and motivate
• Worst, average and best case execution times
  o What are they? Why is it important to consider all of them and not just a single one?
• Worst-case execution time analysis
  o What is it? How is it important for real-time systems?
  o What different aspects must be considered?
  o Caches:
    ▪ Why to caches make this difficult?
    ▪ Why is determining the worse case execution path difficult?
  o Loop iteration bounding
    ▪ Given a piece of code, determine the upper and lower loop execution bounds (CH3-2: slide 16)
• Programming languages
  o Why are some languages more appropriate for some programs than others?
    ▪ E.g., Reactive systems and synchronous languages; interrupt-oriented languages; data flow languages;
• End on Ch3-2: slide 30