Course Wrap-up

- Priority Encoder Revisited
- What (We Hope) You Learned
- Design Methodology
- Final Exam Information

Language of logic design
Both map specifications to physical devices
Both must be flawless...the price we pay for using discrete math

What we HOPE you learned in CS 150 ...

- Language of logic design
  - Logic optimization, state, timing, CAD tools
- Concept of state in digital systems
  - Analogue to variables and program counters in software systems
- Hardware system building
  - datapath = control = digital systems
- Hardware system design methodology
  - Hardware description languages: Verilog
  - Tools to simulate design behavior: output = function (inputs)
  - Logic compilers synthesize hardware blocks of our designs
  - Mapping onto programmable hardware (code generation)
- Contrast with software design
  - Both map specifications to physical devices
  - Both must be flawless...the price we pay for using discrete math

Current state of digital design

- Changes in industrial practice
  - Larger designs
  - Shorter time to market
  - Cheaper products
- Scale
  - Pervasive use of computer-aided design tools over hand methods
  - Multiple levels of design representation
- Time
  - Emphasis on abstract design representations
  - Programmable rather than fixed function components
  - Automatic synthesis techniques
  - Importance of sound design methodologies
- Cost
  - Higher levels of integration
  - Use of simulation to debug designs

CS 150: concepts/skills/abilities

- Basics of logic design (concepts)
- Sound design methodologies (concepts)
- Modern specification methods (concepts)
- Familiarity with full set of CAD tools (skills)
- Appreciation for differences and similarities (abilities) in hardware and software design

New ability: To accomplish the logic design task with the aid of computer-aided design tools and map a problem description into an implementation with programmable logic devices after validation via simulation and understanding of the advantages/disadvantages as compared to a software implementation.
Representation of Digital Designs
- Physical devices (transistors, relays)
- Switches
- Truth tables
- Boolean algebra
- Gates
- Finite state behavior
- Register-transfer behavior
- Concurrent abstract specifications

Simulation, Chipscope & Complex System Description (e.g., SDRAM)

Implementation in Software
```
int combination_lock ( ) {
    int v1, v2, v3;
    int error = 0;
    static int i[3] = 3, 4, 2;
    while ((new_value() &&
            v1 = read_value();
            if (v1 != i[1]) then error = 1;
    while ((new_value() &&
            v2 = read_value();
            if (v2 != i[2]) then error = 1;
    while ((new_value() &&
            v3 = read_value();
            if (v3 != i[3]) then error = 1;
    if (error == 1) then return(0); else return (1);
}
```

Implementation as a Sequential Digital System

Encoding:
- How many bits per input value?
- How many values in sequence?
- How do we know a new input value is entered?
- How do we represent the states of the system?

Behavior:
- Clock wire tells us when it's ok to look at inputs (i.e., they have settled after change)
- Sequential: sequence of values must be entered
- Sequential: remember if an error occurred
- Finite-state specification

Sequential Example (cont'd): Abstract Control

Finite-state Diagram
- States 5 states
- Represent point in execution of machine
- Each state has outputs
- Transitions: 6 from state to state, 5 self transitions, 1 global
- Changes of state occur when clock says it's ok
- Based on value of inputs
- Inputs: reset, new, results of comparators
- Output: open/closed

Sequential Example (cont'd): Data-path vs. Control

Internal Structure
- Data-path
- Storage for combination
- Comparators
- Control
- Finite-state machine controller
- Control for data-path
- State changes controlled by clock
Sequential Example (cont'd): Finite-state Machine

- **Finite-state Machine**
  - Refine state diagram to include internal structure

Sequential Example (cont'd): Finite-state Machine

- **Implementation of the Controller**
  - Special circuit element called a register, for remembering inputs when told to by clock

Sequential Example (cont'd): Finite-state Machine

- **Encode State Table**
  - State can be: S1, S2, S3, OPEN, or ERR
  - Needs at least 3 bits to encode: 000, 001, 010, 011, 100
  - And as many as 5: 00001, 00010, 00100, 01000, 010000
  - Choose 4 bits: 0001, 0010, 0100, 1000
  - Output mux can be: C1, C2, or C3
  - Needs 2 to 3 bits to encode
  - Choose 3 bits: 001, 010, 100
  - Output open/closed can be: open or closed
  - Needs 1 or 2 bits to encode
  - Choose 1 bits: 1, 0

Sequential Example (cont'd): Finite-state Machine

- **Generate State Table (much like a truth-table)**

Sequential Example (cont'd): Finite-state Machine

- **Encode State Table**
  - State can be: S1, S2, S3, OPEN, or ERR
  - Choose 4 bits: 0001, 0010, 0100, 1000, 0000
  - Output mux can be: C1, C2, or C3
  - Choose 3 bits: 001, 010, 100
  - Output open/closed can be: open or closed
  - Choose 1 bits: 1, 0

Design Hierarchy

- **Digital System**
  - Data path
  - Control
  - Code registers multiplexer comparator
  - State registers
  - Combinational logic
  - Register
  - Switching networks
Design Methodology

<table>
<thead>
<tr>
<th>Structure and Function (Behavior) of a Design</th>
<th>HDL Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation</td>
<td>Synthesis</td>
</tr>
</tbody>
</table>

Verification: Design Behave as Required?
- Functional: I/O Behavior
- Register-Level (Architectural)
- Logic-Level (gates)
- Timing: Waveform Behavior

Generation: Map Specification to Implementation

Combinational Logic Implementation
- K-map method to map truth tables into minimized gate level descriptions
- Alternative implementation approaches:
  - Two-level logic, multi-level logic, logic implementation with multiplexers
  - Programmable logic in the form of PLAs, ROMs, Muxes, ...
  - Field programmable logic in the form of devices like Xilinx
- Combinational logic building blocks
  - Arithmetic and logic units, including adders/subtractors and other arithmetic functions (e.g., combinational multipliers)

Sequential Logic Implementation
- Models for representing sequential circuits
  - Abstraction of sequential elements
  - Finite state machines and their state diagrams
  - Inputs/outputs
  - Mealy, Moore, and synchronous Mealy machines
- Finite state machine design procedure
  - Deriving state diagram
  - Deriving state transition table
  - Determining next state and output functions
  - Implementing combinational logic
- Sequential logic building blocks
  - Registers, Register files (with multiple read and write ports), Shifters, Counters, RAMs
  - Arbitrators

State Machine Implementation
- Partitioned State Machines
  - Ways to organize single complex monolithic state machine into simpler, interacting state machines based on functional partitioning
  - Time state approach offers one conceptual method
  - Much more relevant is what you likely did in your course project
- Issues of synchronization across independently clocked subsystems
  - Synchronization of signals
  - Four cycle handshake

Final Exam Information
- Exam Group 2
- Friday, May 11, 12:30-3:30 PM
- Room: 145 Dwinelle

Final Exam Information
- (Long) Design Specification in English for an "interesting" digital subsystem
  - Function described in terms of desired input/output behavior
  - You will need to be able to hand generate waveform diagrams to demonstrate that you understand the design specification
- You may have to partition the subsystem into control and datapath
  - Design the control part as one or more interacting Finite State Machines
  - State Diagram as well as Verilog for control
  - Design the datapath blocks
  - Behavioral Verilog mostly, but gate level hand-drawn schematics for some selected parts
- You may have to revise the design to improve its performance
- For an example, see http://hkn.eecs.berkeley.edu/student/online/cs/150/2000/fa-f.html
Past Final Exams

- **Fall 2005: Statistics Coprocessor**
  - Spec for calculation inside processor
  - Handshake for processor-coprocessor communications
  - High Level Design, Datapath Components, Coprocessor State
    Machine, Verilog Description
- **Spring 2004: Content-Addressable Memory**
  - Memory interface and functional specification
  - High Level Design, Datapath Components, Memory Controller State
    Machine, Verilog Description, Performance Improvement
- **Spring 2001: Elevator System**
  - Complex system behavior
  - High Level Design, Datapath Components, Controller State Machine,
    Extend from 1 to 2 elevators, microprogrammed implementation

Final Exam Information

- The Exam is conceptual and DESIGN-skills oriented
- The Exam is not about obscure details of technologies like the
  Xilinx or Actel internal architectures
- The best way to study for The Exam is to review your course
  project and to reflect on the process you went through in
  designing and implementing it!
- The Exam design problem won’t be a video conferencing system
  -- it will be some kind of digital system with control and a
  datapath that can be specified in a couple of pages of English
  text!
- You will need to write a lot for this Exam! Bring multiple pencils,
  erasers, rulers, AND AT LEAST TWO BLUE BOOKS!!! You won’t
  need a computer or a calculator!
- Open course textbook and open course notes. They probably
  won’t help you much :-(}