CSEP567: Design and Implementation of Digital Systems

- Course staff:
  - Bruce Hemingway, Charles Giefer and Tom Anderl
- Course web:
  - My office: CSE 464 Allen Center, 206 543-6274
- Today
  - Class administration, overview of course web, and logistics
  - What is logic design?
  - What is digital hardware?
  - What will we be doing in this class?

Highlights:

- we'll be reading hand-outs and papers from various sources.
- The course work will be built around an embedded-core processor in an FPGA.
- Tools are Active-HDL from Aldec, Synplify, and Xilinx ISE.
  - Languages are verilog and C.
- Applications in the FPGA will include some audio.
- Lecture-discussion for an hour or so, then into the lab for the rest of the evening.
  - No hardware experience required.
Labs:

- Lab 1- Atmel AVR Microprocessor PWM experiment
- Lab 2 Tutorials:
  1. Introduction to Active-HAL
  2. Using Verilog with Active-HDL
- Lab 3- AFX_XC1000 Blink program in Verilog
- Lab 4- MicroBlaze Blink Program
- Lab 5- MicroBlaze PWM Program
- Lab 6- Verilog String Synthesis Example
- Lab 7- MicroBlaze String Controller

Why are we here?

- **Obvious reasons**
  - this course is part of the PMP requirements
  - it is the implementation basis for all modern computing devices
    - building large things from small components
    - provide a model of how a computer works

- **More important reasons**
  - the inherent parallelism in hardware is often our first exposure to parallel computation
  - it offers an interesting counterpoint to software design and is therefore useful in furthering our understanding of computation, in general
What will we learn in CSEP567?

- The language of logic design
  - Boolean algebra, logic minimization, state, timing, CAD tools
- The concept of state in digital systems
  - analogous to variables and program counters in software systems
- How to specify/simulate/compile/realize our designs
  - hardware description languages
  - tools to simulate the workings of our designs
  - logic compilers to synthesize the hardware blocks of our designs
  - Use of IP (software microprocessor core)
- Contrast with software design
  - sequential and parallel implementations
  - specify algorithm as well as computing/storage resources it will use

Applications of logic design

- Conventional computer design
  - CPUs, busses, peripherals
- Networking and communications
  - phones, modems, routers
- Embedded products
  - in cars, toys, appliances, entertainment devices
- Scientific equipment
  - testing, sensing, reporting
- The world of computing is much much bigger than just PCs!
The Digital Age

- Computing is in its infancy
  - Processing power
    - Doubles every 18 months
    - Factor of 100 / decade
  - Disk capacity
    - Doubles every 12 months
    - Factor of 1000 / decade
  - Optical fiber transmission capacity
    - Doubles every 9 months
    - Factor of 10,000 / decade
- The bases are mathematics and switches
  - How did we get here?

Diophantus of Alexandria  b. ~200 BCE

DIOPHANTI ALEXANDRINI ARITHMETICORVM LIBRI SEX, ET DE NUMERIS MULTANOVILIBRI XV.

Known as the “father of algebra”

Arithmetica is a collection of 130 problems that gives numerical solutions of determinate equations, which have a unique solution, and indeterminate equations.

The Later Alexandrian Age was a time when mathematicians were discovering many ideas that lead to our concept of mathematics today.
850 AD

- Abu Ja'far Muhammad ibn Musa al-Khwarizmi
  - Lived in Baghdad, 780 to 850 AD. One of the first to write on algebra (using words, not letters) and also Hindu-Arabic numbers (1, 2, 3, ...).
  - From his name and writings came the words "algebra" and "algorithm".
  - Book: *Hisab al-jabr w'al-muqabala*

1822

- Charles Babbage
  - Father of computing
- 1822 Difference Engine
  - A calculator
- 1834 Analytical Engine
  - A computer
  - Programmable
### 1854

- George Boole
  - Boolean algebra
  - Number system with 2 values
    - 0/1 ↔ false/true
    - Do math on logic statements
    - 3 operations (NOT, AND, OR)

<table>
<thead>
<tr>
<th>A</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### 1938

- Claude Shannon
  - Implemented Boolean algebra using switches
  - Described information using binary digits (bits)

<table>
<thead>
<tr>
<th>A</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
**Computer Hardware**

- Components
  - Logic
  - Memory

![NOR Circuit](image1.png)

![Latch Circuit](image2.png)

![Adder Circuit](image3.png)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Sum</th>
<th>Carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**1937**

- Alan Turing
  - Turing Machines
  - Simple computer model
    - Can something be computed?

![Turing Machine Diagram](image4.png)

Also pioneered artificial intelligence
1945

- John von Neumann
  - First stored computer program
- A sequence of operations
  - Read from memory
  - Operate using logic gates
  - Store result into memory

Other contributions:
- Quantum Mechanics
- Cellular Automata
- Game Theory

Stored Programs = Software

Bill Gates and Paul Allen, Lakeside, 1968
1946

- ENIAC... the first computer
  - Vacuum tubes for switches

1000x faster than anything before...
- 19,000 tubes
- 200 kilowatts
- 357 multiplies per second
1947

- Bardeen, Brattain, Shockley invent the transistor

1947 Nobel Prize, 1956

2000

Courtesy Mark Bohr, Intel

0.13\mu m

1958

- Kilby and Noyce invent the integrated circuit

1958 Nobel Prize, 2000

2000

Pentium

Courtesy Yan Borodovsky, Intel
1965

- Gordon Moore
  - Moore’s Law: The transistor density of silicon chips doubles every 18 months

1971

- Ted Hoff invents the microprocessor

- Intel 4004
  - 2,300 transistors
  - 3 mm by 4 mm
  - As powerful as the ENIAC
Hardware + Software + Technology

Algorithm → Program

Problem

Memory

Logic (CPU)

Program

Data

1977 and 1981

- Apple II and IBM PC
  - The first microcomputers
A modern example

- **Goal:** Interface a computer to an animal brain
  - Measure brain signals in intact animals

![Tritonia diomedea](image1)

Tritonia and seapen

Brain with implanted chip

Courtesy Jim Beck and Russell Wyeth

More modern examples

- **Computing everywhere**
  - Wireless/wired networking
  - Wearable devices
  - Smart sensors

![Computing everywhere](image2)
What is design?
- given a specification of a problem, come up with a way of solving it choosing appropriately from a collection of available components
- while meeting some criteria for size, cost, power, beauty, elegance, etc.

What is logic design?
- determining the collection of digital logic components to perform a specified control and/or data manipulation and/or communication function and the interconnections between them
- which logic components to choose? – there are many implementation technologies (e.g., off-the-shelf fixed-function components, programmable devices, transistors on a chip, etc.)
- the design may need to be optimized and/or transformed to meet design constraints

What is digital hardware?
- Collection of devices that sense and/or control wires that carry a digital value (i.e., a physical quantity that can be interpreted as a “0” or “1”)
  - example: digital logic where voltage < 0.8v is a “0” and > 2.0v is a “1”
  - example: pair of transmission wires where a “0” or “1” is distinguished by which wire has a higher voltage (differential)
  - example: orientation of magnetization signifies a “0” or a “1”

- Primitive digital hardware devices
  - logic computation devices (sense and drive)
    - are two wires both “1” - make another be “1” (AND)
    - is at least one of two wires “1” - make another be “1” (OR)
    - is a wire “1” - then make another be “0” (NOT)
  - memory devices (store)
    - store a value
    - recall a previously stored value
What is happening now in digital design?

- Important trends in how industry does hardware design
  - larger and larger designs
  - shorter and shorter time to market
  - cheaper and 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
  - simulate and verify before you build

Computation: abstract vs. implementation

- Up to now, computation has been a mental exercise (paper, programs)
- This class is about physically implementing computation using physical devices that use voltages to represent logical values
- Basic units of computation are:
  - representation: "0", "1" on a wire
    set of wires (e.g., for binary ints)
  - assignment: \( x = y \)
  - data operations: \( x + y - 5 \)
  - control:
    - sequential statements: \( A; B; C \)
    - conditionals: \( \text{if} \ x == 1 \ \text{then} \ y \)
    - loops: \( \text{for} \ ( i = 1 ; i == 10, i++; \) \)
    - procedures: \( A; \text{proc}(...); B; \)
- We will study how each of these are implemented in hardware and composed into computational structures
Switches: basic element of physical implementations

- Implementing a simple circuit (arrow shows action if wire changes to “1”):

  close switch (if A is “1” or asserted) and turn on light bulb (Z)

  open switch (if A is “0” or unasserted) and turn off light bulb (Z)

  $Z = A$

Switches (cont’d)

- Compose switches into more complex ones (Boolean functions):

  $Z = A \text{ and } B$

  $Z = A \text{ or } B$
Switching networks

- Switch settings
  - determine whether or not a conducting path exists to light the light bulb
- To build larger computations
  - use a light bulb (output of the network) to set other switches (inputs to another network).
- Connect together switching networks
  - to construct larger switching networks, i.e., there is a way to connect outputs of one network to the inputs of the next.

Transistor networks

- Modern digital systems are designed in CMOS technology
  - MOS stands for Metal-Oxide on Semiconductor
  - C is for complementary because there are both normally-open and normally-closed switches
- MOS transistors act as voltage-controlled switches
  - similar, though easier to work with than relays.
MOS transistors

- MOS transistors have three terminals: drain, gate, and source
  - they act as switches in the following way:
    - if the voltage on the gate terminal is (some amount) higher/lower than the source terminal then a conducting path will be established between the drain and source terminals

\[
\text{n-channel open when voltage at G is low closes when:} \quad \text{voltage}(G) > \text{voltage}(S) + \epsilon
\]

\[
\text{p-channel closed when voltage at G is low opens when:} \quad \text{voltage}(G) < \text{voltage}(S) - \epsilon
\]

MOS networks

what is the relationship between \( x \) and \( y \)?

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 volts</td>
<td>( y )</td>
</tr>
<tr>
<td>3 volts</td>
<td>( y )</td>
</tr>
</tbody>
</table>
What influences the speed of CMOS networks?

- charging and discharging of voltages on wires and gates of transistors
- Capacitors hold charge
  - capacitance is at gates of transistors and wire material
- Resistors slow movement of electrons
  - resistance mostly due to transistors
Representation of digital designs

- Physical devices (transistors, relays)
- Switches
- Truth tables
- Boolean algebra
- Gates
- Waveforms
- Finite state behavior
- Register-transfer behavior
- Concurrent abstract specifications

Digital vs. analog

- Convenient to think of digital systems as having only discrete, digital, input/output values
- In reality, real electronic components exhibit continuous, analog, behavior

  Why do we make the digital abstraction anyway?
  - switches operate this way
  - easier to think about a small number of discrete values

  Why does it work?
  - does not propagate small errors in values
  - always resets to 0 or 1
A simple model of a digital system is a unit with inputs and outputs:

```
  inputs ----| system |---- outputs
         :             :
```

- Combinational means "memory-less"
  - a digital circuit is combinational if its output values only depend on its input values
Combinational logic symbols

- Common combinational logic systems have standard symbols called logic gates
  - Buffer, NOT
    \[ A \rightarrow Z \]
  - AND, NAND
    \[ A \quad B \rightarrow Z \]
  - OR, NOR
    \[ A \quad B \rightarrow Z \]
  - Easy to implement with CMOS transistors (the switches we have available and use most)

Sequential logic

- Sequential systems
  - exhibit behaviors (output values) that depend not only on the current input values, but also on previous input values
- In reality, all real circuits are sequential
  - because the outputs do not change instantaneously after an input change
  - why not, and why is it then sequential?
- A fundamental abstraction of digital design is to reason (mostly) about steady-state behaviors
  - look at the outputs only after sufficient time has elapsed for the system to make its required changes and settle down
Synchronous sequential digital systems

- Outputs of a combinational circuit depend only on current inputs after sufficient time has elapsed
- Sequential circuits have memory even after waiting for the transient activity to finish
- The steady-state abstraction is so useful that most designers use a form of it when constructing sequential circuits:
  - the memory of a system is represented as its state
  - changes in system state are only allowed to occur at specific times controlled by an external periodic clock
  - the clock period is the time that elapses between state changes it must be sufficiently long so that the system reaches a steady-state before the next state change at the end of the period

Example of combinational and sequential logic

- Combinational:
  - input A, B
  - wait for clock edge
  - observe C
  - wait for another clock edge
  - observe C again: will stay the same

- Sequential:
  - input A, B
  - wait for clock edge
  - observe C
  - wait for another clock edge
  - observe C again: may be different
Abstractions

- Some we've seen already
  - digital interpretation of analog values
  - transistors as switches
  - switches as logic gates
  - use of a clock to realize a synchronous sequential circuit
- Some others we will see
  - truth tables and Boolean algebra to represent combinational logic
  - encoding of signals with more than two logical values into binary form
  - state diagrams to represent sequential logic
  - hardware description languages to represent digital logic
  - waveforms to represent temporal behavior

An example

- Calendar subsystem: number of days in a month (to control watch display)
  - used in controlling the display of a wrist-watch LCD screen
  - inputs: month, leap year flag
  - outputs: number of days
Implementation in software

```c
integer number_of_days ( month, leap_year_flag)
{
    switch (month) {
        case 1: return (31);
        case 2: if (leap_year_flag == 1) then return (29)
                      else return (28);
        case 3: return (31);
        ...
        case 12: return (31);
        default: return (0);
    }
}
```

Implementation as a combinational digital system

- **Encoding:**
  - how many bits for each input/output?
  - binary number for month
  - four wires for 28, 29, 30, and 31

- **Behavior:**
  - combinational
  - truth table specification

<table>
<thead>
<tr>
<th>month</th>
<th>leap</th>
<th>d28</th>
<th>d29</th>
<th>d30</th>
<th>d31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0001</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0010</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0011</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0100</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0101</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0110</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0111</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1000</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1001</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1010</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1011</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1100</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1101</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>111-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Combinational example (cont’d)

- Truth-table to logic to switches to gates
  - \( d_{28} = 1 \) when \( \text{month}=0010 \) and \( \text{leap}=0 \)
  - \( d_{28} = m_8' \cdot m_4' \cdot m_2 \cdot m_1' \cdot \text{leap} \)
  - \( d_{31} = 1 \) when \( \text{month}=0001 \) or \( \text{month}=0011 \) or ... \( \text{month}=1100 \)
  - \( d_{31} = (m_8' \cdot m_4' \cdot m_2 \cdot m_1') + (m_8' \cdot m_4 \cdot m_2' \cdot m_1') + ... \)
  - \( d_{31} = \text{can we simplify more?} \)

<table>
<thead>
<tr>
<th>month</th>
<th>leap</th>
<th>d_{28}</th>
<th>d_{29}</th>
<th>d_{30}</th>
<th>d_{31}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0010</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0011</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0100</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1100</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1101</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1111</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0000</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Combinational example (cont’d)

- \( d_{28} = m_8' \cdot m_4' \cdot m_2 \cdot m_1' \cdot \text{leap} \)
- \( d_{29} = m_8 \cdot m_4 \cdot m_2 \cdot m_1 \cdot \text{leap} \)
- \( d_{30} = (m_8' \cdot m_4' \cdot m_2 \cdot m_1') + (m_8' \cdot m_4 \cdot m_2' \cdot m_1') + (m_8 \cdot m_4' \cdot m_2' \cdot m_1') + (m_8 \cdot m_4' \cdot m_2 \cdot m_1') \)
- \( d_{31} = (m_8' \cdot m_4 \cdot m_2' \cdot m_1') + (m_8' \cdot m_4' \cdot m_2 \cdot m_1') + (m_8 \cdot m_4' \cdot m_2 \cdot m_1') + (m_8 \cdot m_4 \cdot m_2' \cdot m_1') \)
Activity

- How much can we simplify d31?

- What if we started the months with 0 instead of 1? 
  (i.e., January is 0000 and December is 1011)

Combinational example (cont’d)

- \( d_{28} = m^8 \cdot m^4 \cdot m^2 \cdot m^1 \cdot \text{leap}' \)
- \( d_{29} = m^8 \cdot m^4 \cdot m^2 \cdot m^1 \cdot \text{leap} \)
- \( d_{30} = (m^8 \cdot m^4 \cdot m^2 \cdot m^1') + (m^8 \cdot m^4 \cdot m^2 \cdot m^1') + (m^8 \cdot m^4 \cdot m^2 \cdot m^1) + (m^8 \cdot m^4 \cdot m^2 \cdot m^1) \)
- \( d_{31} = (m^8 \cdot m^4 \cdot m^2 \cdot m^1) + (m^8 \cdot m^4 \cdot m^2 \cdot m^1) + (m^8 \cdot m^4 \cdot m^2 \cdot m^1) + (m^8 \cdot m^4 \cdot m^2 \cdot m^1) + (m^8 \cdot m^4 \cdot m^2 \cdot m^4') + (m^8 \cdot m^4 \cdot m^2 \cdot m^1') + (m^8 \cdot m^4 \cdot m^2 \cdot m^1') \)
Another example

- Door combination lock:
  - punch in 3 values in sequence and the door opens; if there is an error the lock must be reset; once the door opens the lock must be reset
  - inputs: sequence of input values, reset
  - outputs: door open/close
  - memory: must remember combination or always have it available as an input

```c
integer combination_lock() {
    integer v1, v2, v3;
    integer error = 0;
    static integer c[3] = 3, 4, 2;

    while (!new_value());
    v1 = read_value();
    if (v1 != c[1]) then error = 1;

    while (!new_value());
    v2 = read_value();
    if (v2 != c[2]) then error = 1;

    while (!new_value());
    v3 = read_value();
    if (v3 != c[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 comparisons
  - 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

```
<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
<th>C3</th>
</tr>
</thead>
<tbody>
<tr>
<td>mux</td>
<td>controller</td>
<td></td>
</tr>
<tr>
<td></td>
<td>clock</td>
<td></td>
</tr>
</tbody>
</table>
```

value → equal

new equal reset

Sequential example (cont’d): finite-state machine

- Finite-state machine
  - refine state diagram to include internal structure

```
S1
open/closed
reset
not new
equal & new
mux=C

S2
not equal & new
equal & new
mux=C

S3
not equal & new
equal & new
mux=C

OPEN
```

ERR

CLOSED

Sequential example (cont’d):
finite-state machine

- Finite-state machine
  - generate state table (much like a truth-table)

<table>
<thead>
<tr>
<th>reset</th>
<th>new</th>
<th>equal</th>
<th>state</th>
<th>next state</th>
<th>mux</th>
<th>open/closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>S1</td>
<td>C1</td>
<td>closed</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S1</td>
<td>S1</td>
<td>C1</td>
<td>closed</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>S1</td>
<td>ERR</td>
<td>–</td>
<td>closed</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>S1</td>
<td>S2</td>
<td>C2</td>
<td>closed</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>–</td>
<td>S2</td>
<td>S2</td>
<td>C2</td>
<td>closed</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>S2</td>
<td>ERR</td>
<td>–</td>
<td>closed</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>S2</td>
<td>S3</td>
<td>C3</td>
<td>closed</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>–</td>
<td>S3</td>
<td>S3</td>
<td>C3</td>
<td>closed</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>S3</td>
<td>ERR</td>
<td>–</td>
<td>closed</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>S3</td>
<td>OPEN</td>
<td>–</td>
<td>open</td>
</tr>
<tr>
<td>0</td>
<td>–</td>
<td>–</td>
<td>OPEN</td>
<td>OPEN</td>
<td>–</td>
<td>open</td>
</tr>
<tr>
<td>0</td>
<td>–</td>
<td>–</td>
<td>ERR</td>
<td>ERR</td>
<td>–</td>
<td>closed</td>
</tr>
</tbody>
</table>

Sequential example (cont’d):
encoding

- 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, 10000
    - choose 4 bits: 0001, 0010, 0100, 1000, 0000
  - 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): encoding

- 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

<table>
<thead>
<tr>
<th>reset</th>
<th>new</th>
<th>equal</th>
<th>state</th>
<th>next state</th>
<th>mux</th>
<th>open/closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0001</td>
<td>001</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0001</td>
<td>0001</td>
<td>001</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0001</td>
<td>0000</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0001</td>
<td>0010</td>
<td>010</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0010</td>
<td>0010</td>
<td>010</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0010</td>
<td>0000</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0010</td>
<td>0100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0100</td>
<td>0100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0100</td>
<td>0000</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0100</td>
<td>1000</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0000</td>
<td>0000</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

Have lock always wait for 3 key presses exactly before making a decision
Sequential example (cont’d): controller implementation

- Implementation of the controller

  ![Diagram showing the controller and its components]

  - Special circuit element, called a register, for remembering inputs when told to by clock

Design hierarchy

![Diagram showing the design hierarchy]

- System
  - Data-path
    - Code registers
    - Multiplexer
    - Comparator
  - Control
    - State registers
    - Combinational logic
- Logic
- Register
- Switching networks
Summary

- That was what the entire course is about (mostly)
  - converting solutions to problems into combinational and sequential networks effectively organizing the design hierarchically
  - doing so with a modern set of design tools that lets us handle large designs effectively
  - taking advantage of optimization opportunities
  - Exploring pre-packaged IP (cores)

- Now lets do it again
  - this time we’ll take nine weeks instead of one