Overview

- Last lecture
  - Introduction to Verilog

- Today
  - Introduction to sequential logic and systems
    - The basic concepts
    - A simple example

Sequential versus combinational

- A → C
  - Apply fixed inputs A, B
  - Wait for clock edge
  - Observe C
  - Wait for another clock edge
  - Observe C again

  Combinational: C will stay the same
  Sequential: C may be different

Sequential logic

- Two types
  - Synchronous = clocked
  - Asynchronous = self-timed

- Has state
  - State = memory

- Employs feedback

- Assumes steady-state signals
  - Signals are valid after they have settled
  - State elements hold their settled output values

Sequential versus combinational (again)

- Combinational systems are memoryless
  - Outputs depend only on the present inputs

- Sequential systems have memory
  - Outputs depend on the present and the previous inputs

Synchronous sequential systems

- Memory holds a system’s state
  - Changes in state occur at specific times
  - A periodic signal times or clocks the state changes
  - The clock period is the time between state changes

Steady-state abstraction

- Outputs retain their settled values
  - The clock period must be long enough for all voltages to settle to a steady state before the next state change
Example: A sequential system

- Door combination lock
  - Enter 3 numbers in sequence and the door opens
  - If there is an error the lock must be reset
  - After the door opens the lock must be reset
  - Inputs: Sequence of numbers, reset
  - Outputs: Door open/close
  - Memory: Must remember the combination

Understand the problem

- Consider I/O and unknowns
  - How many bits per input?
  - How many inputs in sequence?
  - How do we know a new input is entered?
  - How do we represent the system states?

Implement using sequential logic

- Behavior
  - Clock tells us when to look at inputs
    - After inputs have settled
    - Sequential: Enter sequence of numbers
    - Sequential: Remember if error occurred
  - Need a finite-state diagram
    - Assume synchronous inputs
    - State sequence
      - Enter 3 numbers serially
      - Remember if error occurred
    - All states have outputs
      - Lock open or closed

Finite-state diagram

- States: 5
  - Each state has outputs
  - Outputs: open/closed

We use state diagrams to represent sequential logic

System transitions between finite numbers of states

Separate data path and control

- Data path
  - Stores combination
  - Compares inputs with combination

- Control
  - Finite state-machine controller
  - Control for data path
  - State changes clocked

Refine diagram; generate state table

- Refine state diagram to include internal structure

- Generate state table

next state | new | equal | state
--- | --- | --- | ---
S1 | 1 | 1 | ERR
S2 | 1 | 1 | C2 closed
S3 | 1 | 1 | OPEN
Encode state table

- State can be: S1, S2, S3, OPEN, or ERR
  - Need at least 3 bits to encode: 000, 001, 010, 011, 100
  - Can use 5 bits: 00001, 00010, 00100, 01000, 10000
  - Choose 4 bits: 0001, 0010, 0100, 1000, 0000
- Output to mux can be: C1, C2, or C3
  - Need 2 or 3 bits to encode
  - Choose 3 bits: 001, 010, 100
- Output open/closed can be: Open or closed
  - Need 1 or 2 bits to encode
  - Choose 1 bit: 1, 0

Designing the datapath

- Four 3:1 multiplexers
  - 2-input ANDs and 3-input OR
- Four single-bit comparators
  - 2-input XNORs
  - 4-input AND

Where did we use memory?

- Memory: Stored combination, state (errors or successes in past inputs)

Where did we use feedback?

- Feedback: Comparator output ("equal" signal)
Where did we use clock?

- **Clock** synchronizes the inputs
  - Accept inputs when clock goes high
- **Controller is clocked**
  - Mux-control and open/closed signals change on the clock edge

![Diagram](image)

Then next 5 weeks...

- We learn the details
  - Latches, flip-flops, registers
  - Shift registers, counters
  - State machines
  - Timing and timing diagrams
  - Synchronous and asynchronous inputs
  - metastability
  - Clock skew
  - Moore and Mealy machines
  - One-hot encoding
  - More...