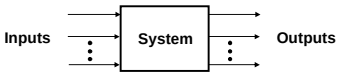
# Lecture 13

- Overview of sequential logic
  - Basic concepts
  - An example

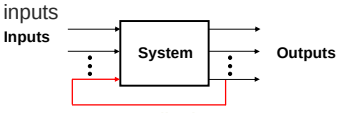


# Sequential vs. combinational

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

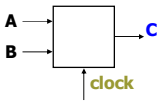



- Sequential systems have memory
  - Outputs depend on the present and the previous inputs






# Sequential vs. combinational




Apply fixed inputs A, B
When the clock ticks, the output becomes available
Observe C
Wait for another clock tick
Observe C again

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

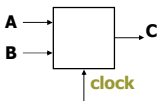


# Synchronous sequential systems

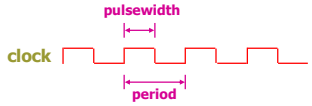
- Memory holds a system's *state*
  - Changes in state occur at specific times
  - A periodic signal times or clocks the state changes



# Clock



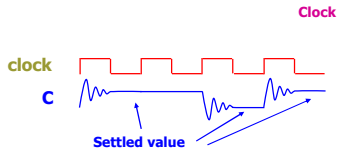

State changes occur at rising edge of clock






# Steady-state abstraction

- The clock period must be long enough for all voltages to settle to a steady state before the next state change



## Recap: Sequential logic

- Mostly has clock (for us, always)
  - Synchronous = clocked
  - Exception: Asynchronous circuits
- Has state
  - State = memory
- Employs feedback
- Assumes steady-state signals
  - Signals are valid after they have settled
  - State elements hold their settled output values



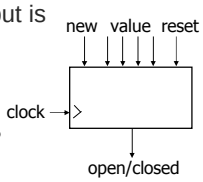
## Example: Sequential system

- Door combination lock
  - Enter three numbers in sequence and the door opens
  - As each new number is entered, press 'new' (like 'enter)
  - If there is an error the lock must be reset
  - After the door opens the lock must be reset
- Inputs?
  - Sequence of numbers, reset, new
- Outputs?
  - Door open/close
- Memory?
  - Must remember the combination and what was entered



## Understand the problem

- 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?
  - What are the system states?






## Implementation

- A diagram may be helpful
  - Assume synchronous inputs
  - State sequence
    - Enter 3 numbers serially
    - Remember if error occurred
  - All states have outputs
    - Lock open or closed



## Finite-state diagram

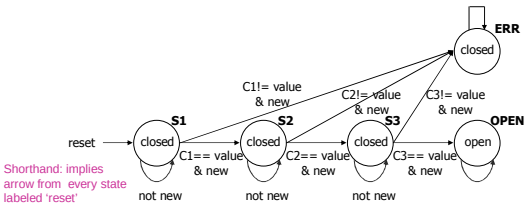
- States: 5
  - Each state has outputs
- Outputs: open/closed
- Inputs: reset, new, results of comparisons
  - Assume synchronous inputs

We use state diagrams to represent sequential logic

System transitions between finite numbers of states

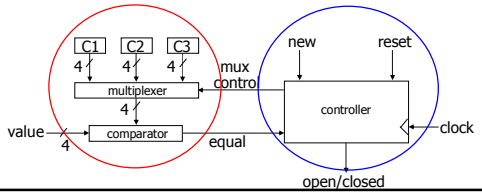


## Finite-state diagram



## Separate data path and control

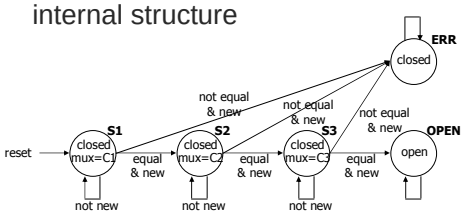
- Data path
  - Stores combination
  - Compares inputs with combination
- Control
  - State-machine controller
  - State changes clocked






## Refine state diagram

- Refine state diagram to include internal structure






## Generate state table

| reset | new | equal | state | next state | mux | open/closed |
|---|---|---|---|---|---|---|
| 1 | – | – | – | S1 | C1 | closed |
| 0 | 0 | – | S1 | S1 | C1 | closed |
| 0 | 1 | 0 | S1 | ERR | – | closed |
| 0 | 1 | 1 | S1 | S2 | C2 | closed |
| … | | | | | | |
| 0 | 1 | 1 | S3 | OPEN | – | 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



## Encode state table

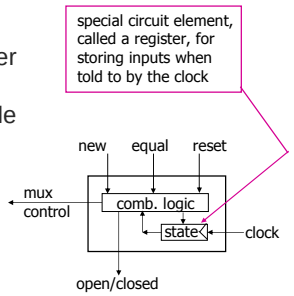
- Good encoding choice!
  - Mux control is identical to last 3 state bits
  - Open/closed is identical to first state bit
  - Output encoding ⇒ the outputs and state bits are the same

| reset | new | equal | state | next state | mux | open/closed |
|---|---|---|---|---|---|---|
| 1 | – | – | – | 0001 | 001 | 0 |
| 0 | 0 | – | 0001 | 0001 | 001 | 0 |
| 0 | 1 | 0 | 0001 | 0000 | – | 0 |
| 0 | 1 | 1 | 0001 | 0010 | 010 | 0 |
| … | | | | | | |
| 0 | 1 | 1 | 0100 | 1000 | – | 1 |
| … | | | | | | |

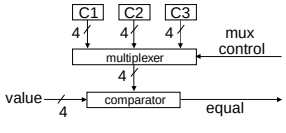


## Implementing the controller

- Will learn how to design the controller given the encoded state-transition table



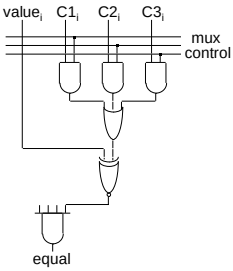
## Designing the datapath






## Designing the datapath

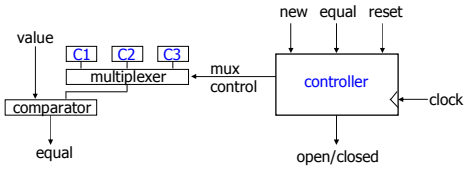
- Four multiplexers
  - 2-input ANDs and 3-input OR
- Four single-bit comparators
  - 2-input XNORs
- 4-input AND






## Where did we use memory?

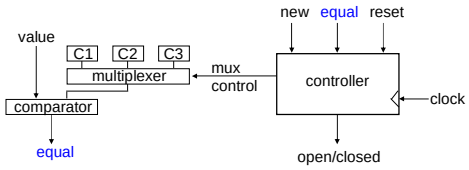
- Memory: Stored combination, state (errors or successes in past inputs)






## 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