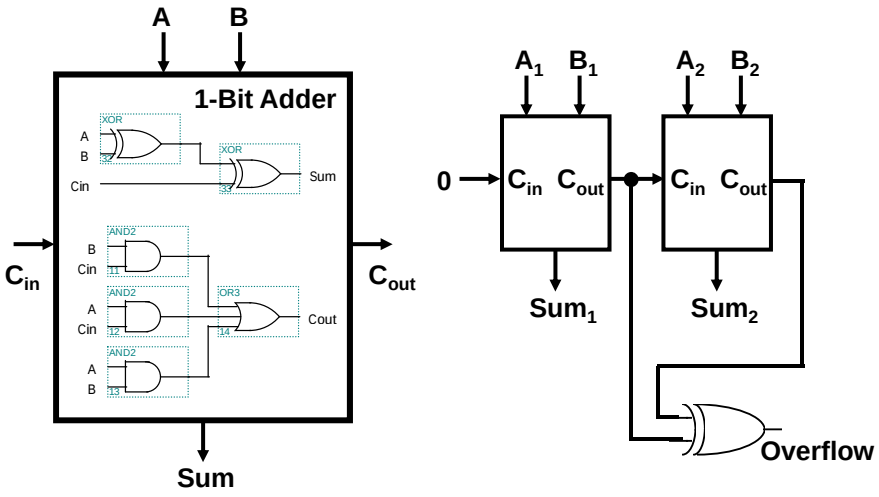
Lecture 12

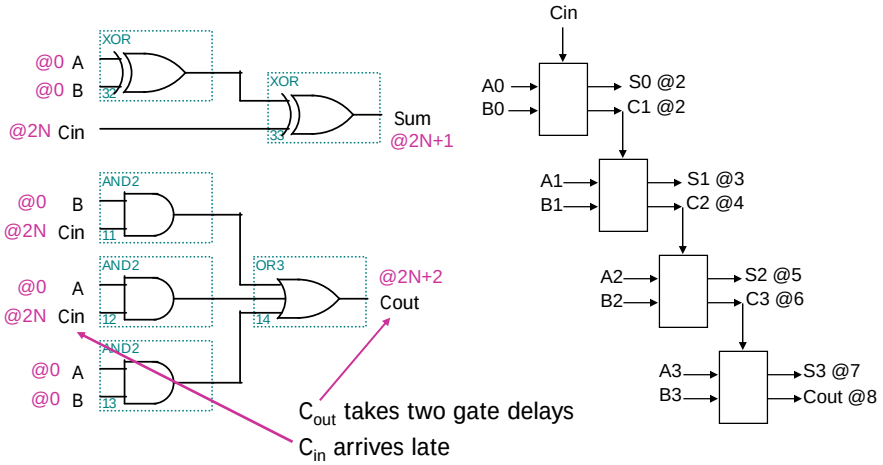
- Logistics
  - HW4 was due yesterday
  - HW5 was out yesterday (due next Wednesday)
  - Feedback: thank you!
  - Things to work on: Big picture, Book chapters, Exam comments
- Last lecture
  - Adders
- Today
  - Clarification of Adders
  - Summary of Combinational Logic
  - Introduction to Sequential Logic
    - The basic concepts
    - A simple example

Adders allow computers to add numbers
2-bit ripple-carry adder



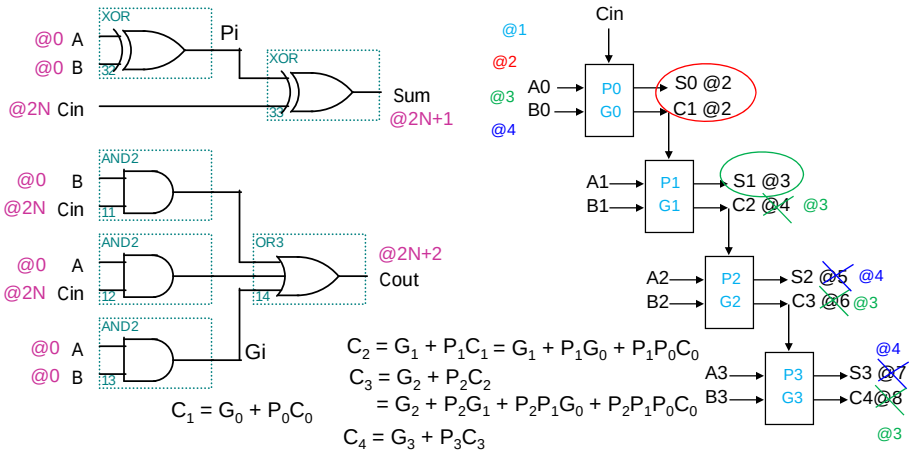
Problem: Ripple-carry delay

- Carry propagation limits adder speed (we want to add fast)



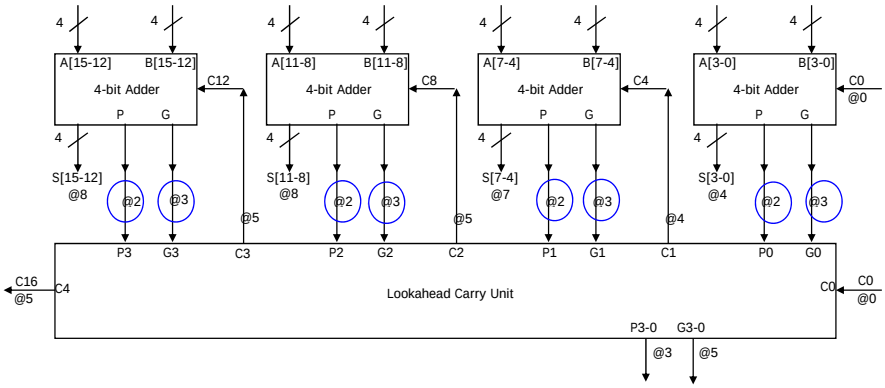
One Solution: Carry lookahead logic

- Get Pi (propagate) and Gi (generate)



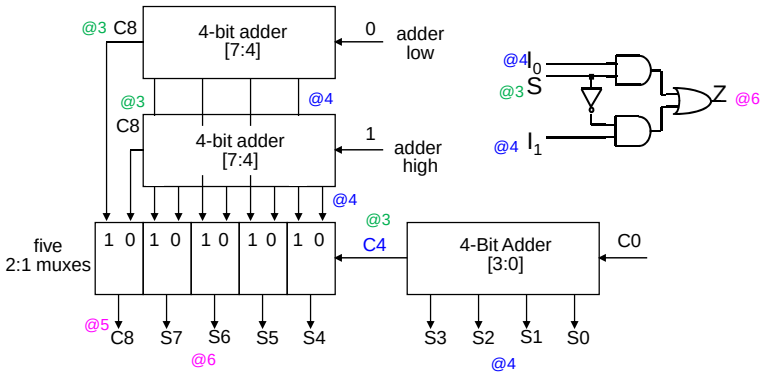
Cascaded carry-lookahead adder (in your HW5)

- 4 four-bit adders with internal carry lookahead
  - Second level lookahead extends adder to 16 bits



Another solution: Carry-select adder

- Redundant hardware speeds carry calculation
  - Compute two high-order sums while waiting for carry-in (C4)
  - Select correct high-order sum after receiving C4



We've finished combinational logic...

- What you should know
  - Twos complement arithmetic
  - Truth tables
  - Basic logic gates
  - Schematic diagrams
  - Timing diagrams
  - Minterm and maxterm expansions (canonical, minimized)
  - de Morgan's theorem
  - AND/OR to NAND/NOR logic conversion
  - K-maps, logic minimization, don't cares
  - Multiplexers/demultiplexers
  - PLAs/PALs
  - ROMs
  - Adders

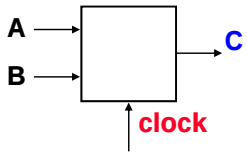
We had no way to store memory:
When the input changed, the output changed

Next: Sequential logic can store memory…

Sequential Logic (next 5 weeks!)

- We learn the details
  - Latches, flip-flops, registers (storage)
  - Shift registers, counters (we can count now!)
  - State machines
  - Timing and timing diagrams
    - timing more important than combinational logic
  - Synchronous and asynchronous inputs
    - Metastability (problem!)
  - Moore and Mealy machines (types of state machines)
  - More...

Sequential versus combinational

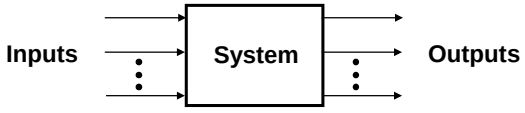


Apply fixed inputs A, B
Wait for clock edge
Observe C
Wait for another clock edge
Observe C again

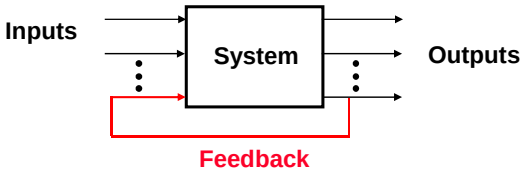
Combinational: C will stay the same
Sequential: C may be different

Sequential versus combinational (again)

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

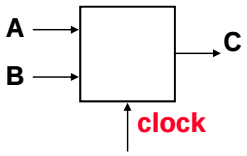


- Sequential systems have memory
  - Outputs depend on the present and the previous inputs

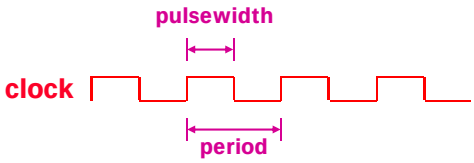


Synchronous sequential systems

- 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



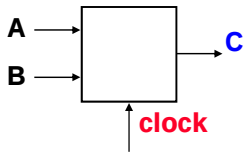
State changes occur at rising edge of clock



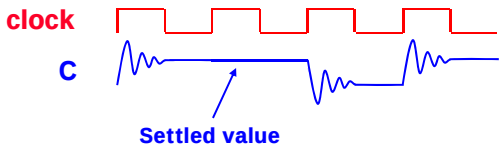
duty cycle = pulsewidth / period (here it is 50%)

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



Clock hides transient behavior



What did I just say about sequential logic?

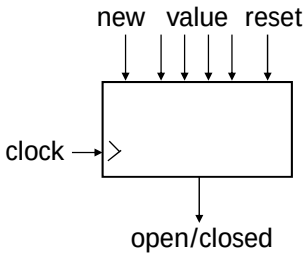
- Has clock
  - Synchronous = clocked
  - Exception: Asynchronous
- Has state
  - State = memory
- Employs feedback
- Assumes steady-state signals
  - Signals are valid after they have settled
  - State elements hold their settled output values

Example: 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

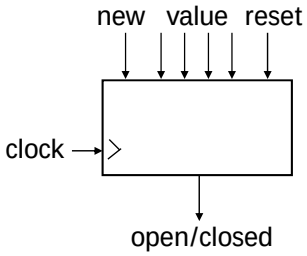
Understand the problem

- 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

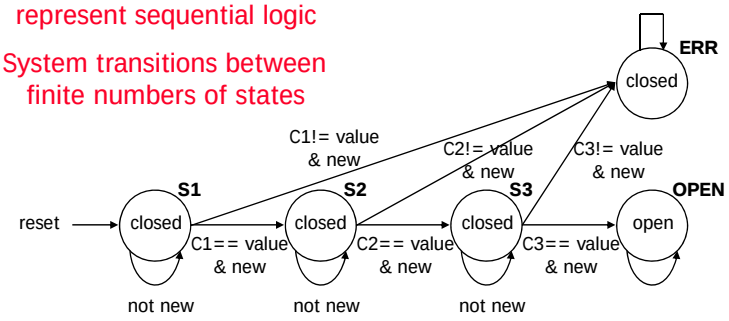


Finite-state diagram

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

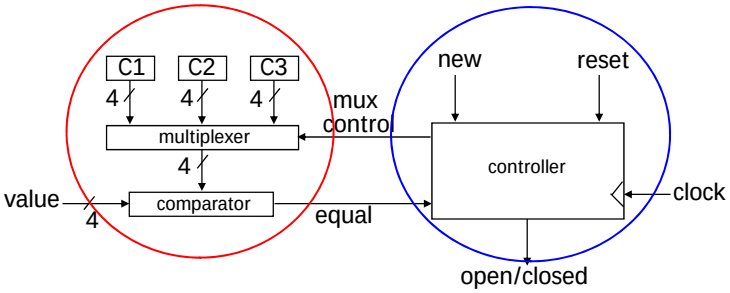
We use state diagrams to represent sequential logic

System transitions between finite numbers of states



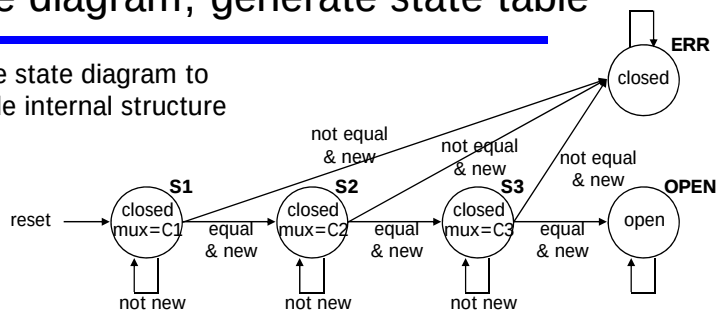
Separate data path and control

- 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

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
…						