Combinational Logic Design Process

- Create truth table from specification
- Generate K-maps & obtain logic equations
- Draw logic diagram (sharing common gates)
- Simulate circuit for design verification
  - Debug & fix problems when output is incorrect
    - Check truth table against K-map population
    - Check K-map groups against logic equation product terms
    - Check logic equations against schematic
- Circuit optimization for area and/or performance
  - Analyze verified circuit for optimization metric
    - G, G_{IO}, G_{del}, P_{del}
  - Use Boolean postulates & theorems
- Re-simulate & verify optimized design

K-mapping & Minimization Steps

Step 1: generate K-map

- Put a 1 in all specified minterms
- Put a 0 in all other boxes (optional)

Step 2: group all adjacent 1s without including any 0s

- All groups (aka *prime implicants*) must be rectangular and contain a "power-of-2" number of 1s
  - 1, 2, 4, 8, 16, 32, …
- An essential group (aka *essential prime implicant*) contains at least 1 minterm not included in any other groups
  - A given minterm may be included in multiple groups

Step 3: define product terms using variables common to all minterms in group

Step 4: sum all essential groups plus a minimal set of remaining groups to obtain a minimum SOP

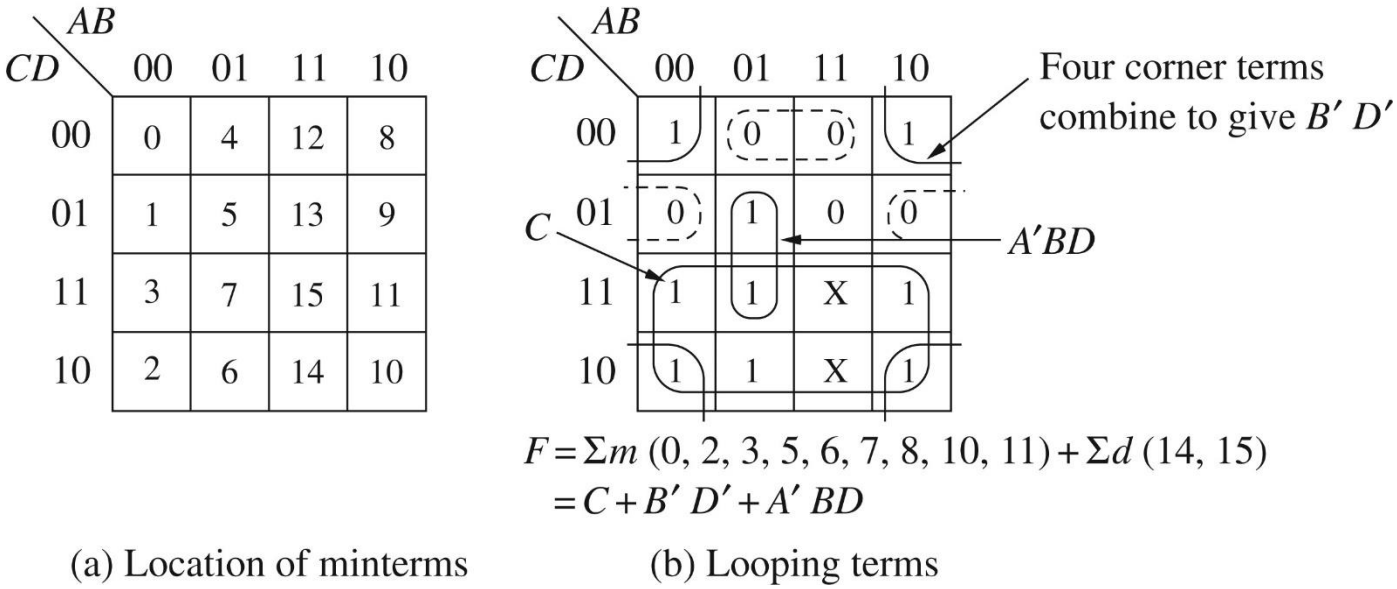
K-map Minimization Goals

- Larger groups:
  - Smaller product terms
    - Fewer variables in common
  - Smaller AND gates
    - In terms of number of inputs
- Fewer groups:
  - Fewer product terms
    - Fewer AND gates
    - Smaller OR gate
      - In terms of number of inputs

- Alternate method:
  - Group 0s
    - Could produce fewer and/or smaller product terms
  - Invert output
    - Use NOR instead of OR gate

K-map example (text)

FIGURE 1-3:
Four-Variable Karnaugh Maps

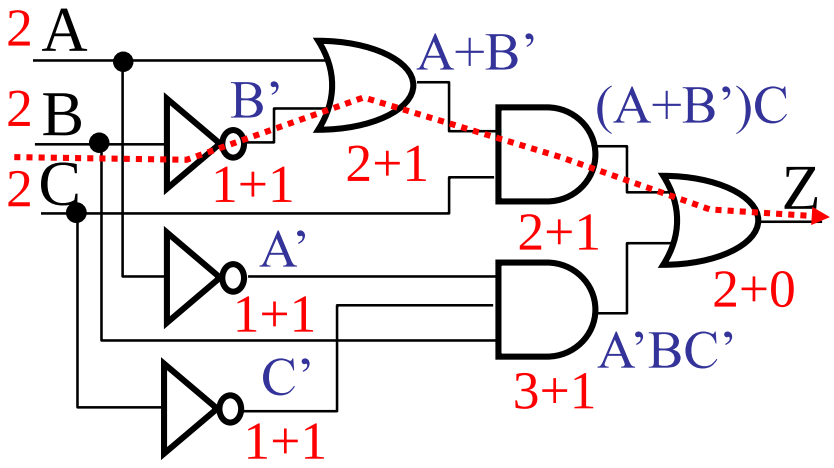


Circuit Analysis

- We can implement different circuits for same logic function that are *functionally equivalent* (produce the correct output response for all input values)
  - Which implementation is the best?
    - Depends on design goals and criteria
- Area analysis
  - Number of gates, G (most commonly used)
  - Number of gate inputs and outputs, G_{IO} (more accurate)
    - Bigger gates take up more area
- Performance analysis (*worst case path from inputs to outputs*)
  - Number of gates in worst case path from input to output, G_{del}
  - More accurate delay measurement per gate
    - Propagation delay = intrinsic (*internal*) delay + extrinsic (*external*) delay
    - Relative prop delay, P_{del} = # inputs to gate (*intrinsic*) + # loads (*extrinsic*)

Circuit Analysis Example

- From previous example:
  Z=(A+B')C+A'BC'
  - # gates: $G = 7$
  - # gate I/O: $G_{IO} = 19$
  - Gate delay: $G_{del} = 4$
    - worst case path: B→Z
  - Prop delay: $P_{del} = 12$
    - worst case path: B→Z



Design Verification Guidelines

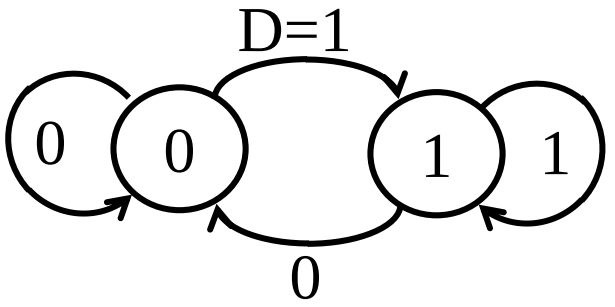
- Use all audits and analysis aids possible to help find potential design bugs
  - Investigate and correct all errors/warnings
- Simulate thoroughly but use stimuli that "eat their way into the design" testing one function at a time
  - more important for complex circuits
- When circuit doesn't work, see what works and what doesn't to narrow down the search space for the problem
  - Which outputs work
  - Which outputs fail and under what conditions
  - Monitor lots of internal nodes
  - Additional simulations (with different vectors) can be helpful
- "Debugging is just like solving a puzzle"
  - "If something doesn't look right, stop and check it out"
    - Don't overlook potential bugs
  - "When you've found the problem, everything starts makes sense"
- Always re-run audits and simulation after correcting any problem (or after making any changes)
  - Another bug could be lurking, or
  - The fix may have messed up something else

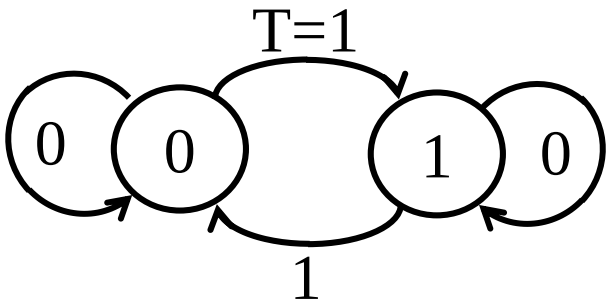
Sequential Logic Design Steps

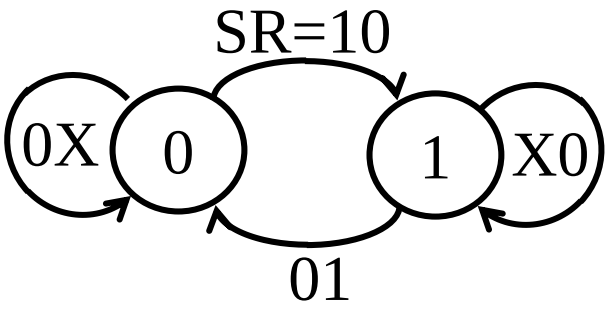
- Derive circuit state diagram from design specs
- Create state table
- Choose flip-flops (D, T, SR, JK)
- Create circuit excitation table
  - use flip-flop excitation tables
- Construct K-maps for:
  - flip-flop inputs
  - primary outputs
- Obtain minimized SOP equations
- Draw logic diagram
- Simulate to verify design & debug as needed
- Perform circuit analysis & logic optimization

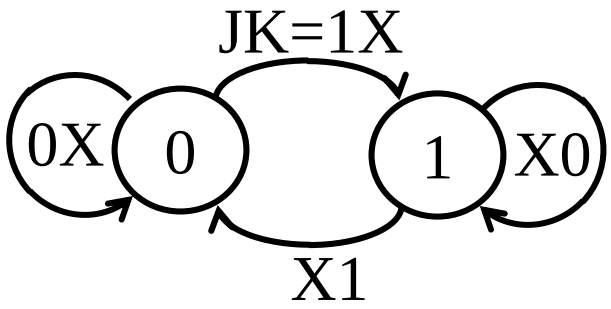
Flip-Flop Excitation Tables & State Diagrams

Q Q+	D	T	S R	J K
0 0	0	0	0 X	0 X
0 1	1	1	1 0	1 X
1 0	0	1	0 1	X 1
1 1	1	0	X 0	X 0





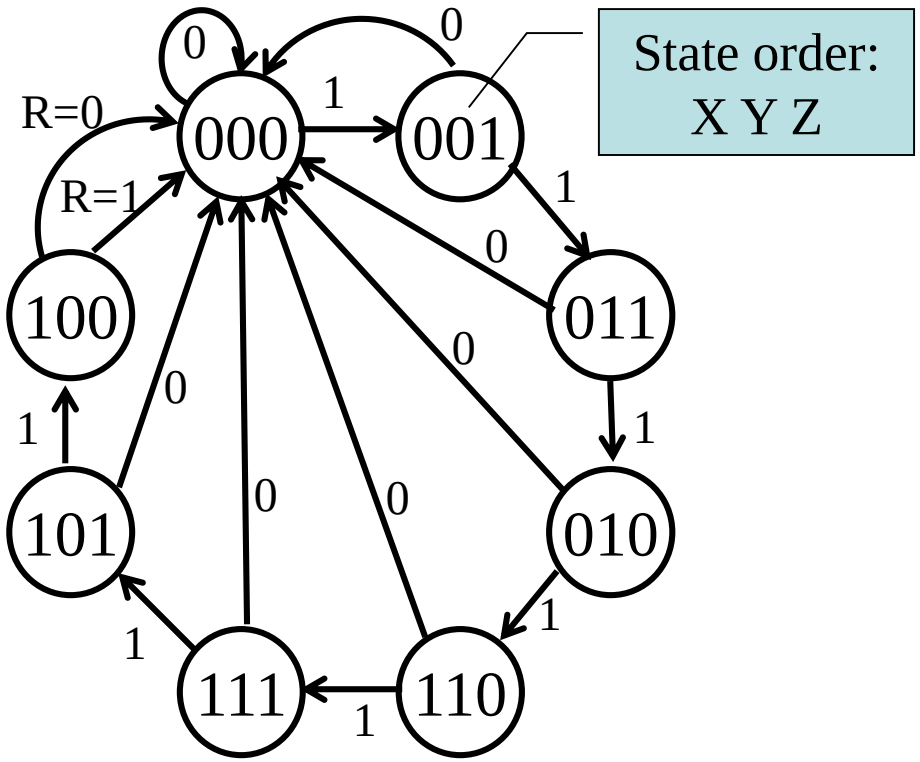




Sequential Design Example

Design a 3-bit gray code counter with
active low synchronous reset (R)

State Diagram



Inputs R	Current state (X Y Z)	Next state (X Y Z)
0	XXX	000
1	000	001
1	001	011
1	010	110
1	011	010
1	100	000
1	101	100
1	110	111
1	111	101

State Table

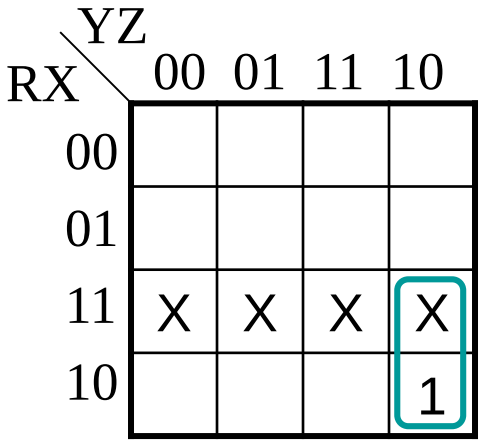
3-bit Gray Code Counter

- Choose flip-flops:
  - Let X be a JK
  - Let Y be a D
  - Let Z be a SR
- Create circuit excitation table

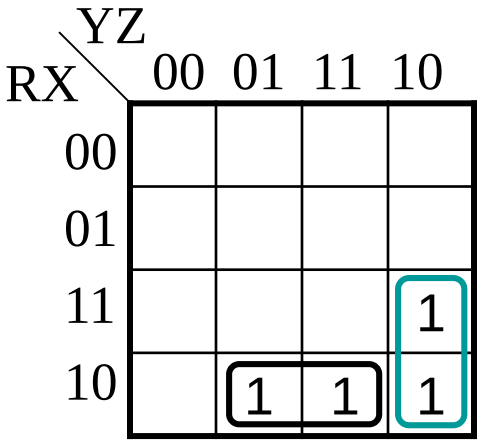
Inputs R	Current state (X Y Z)	Next state (X Y Z)	QX Jx Kx	QY Dy	QZ Sz Rz
0	X X X	0 0 0	0 1	0	0 1
1	0 0 0	0 0 1	0 X	0	1 0
1	0 0 1	0 1 1	0 X	1	X 0
1	0 1 0	1 1 0	1 X	1	0 X
1	0 1 1	0 1 0	0 X	1	0 1
1	1 0 0	0 0 0	X 1	0	0 X
1	1 0 1	1 0 0	X 0	0	0 1
1	1 1 0	1 1 1	X 0	1	1 0
1	1 1 1	1 0 1	X 0	0	X 0

3-bit Gray Code Counter (cont)

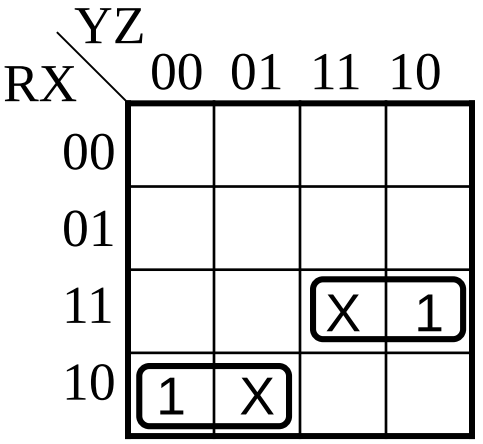
- Generate K-Maps & obtain minimized SOPs



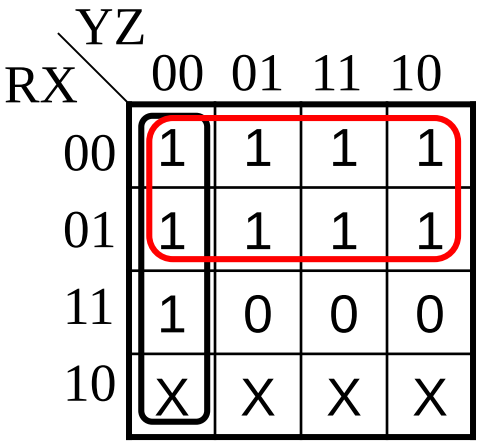
Jx = RYZ'



Dy = RYZ' + RX'Z



Sz = RXY + RX'Y'



Kx = R' + Y'Z'

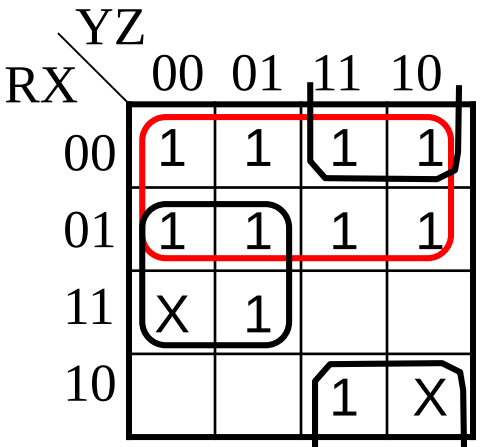
Further reductions:

Rz = R' + X⊕Y

Sz = R(X⊕Y)'

= (R' + X⊕Y)'

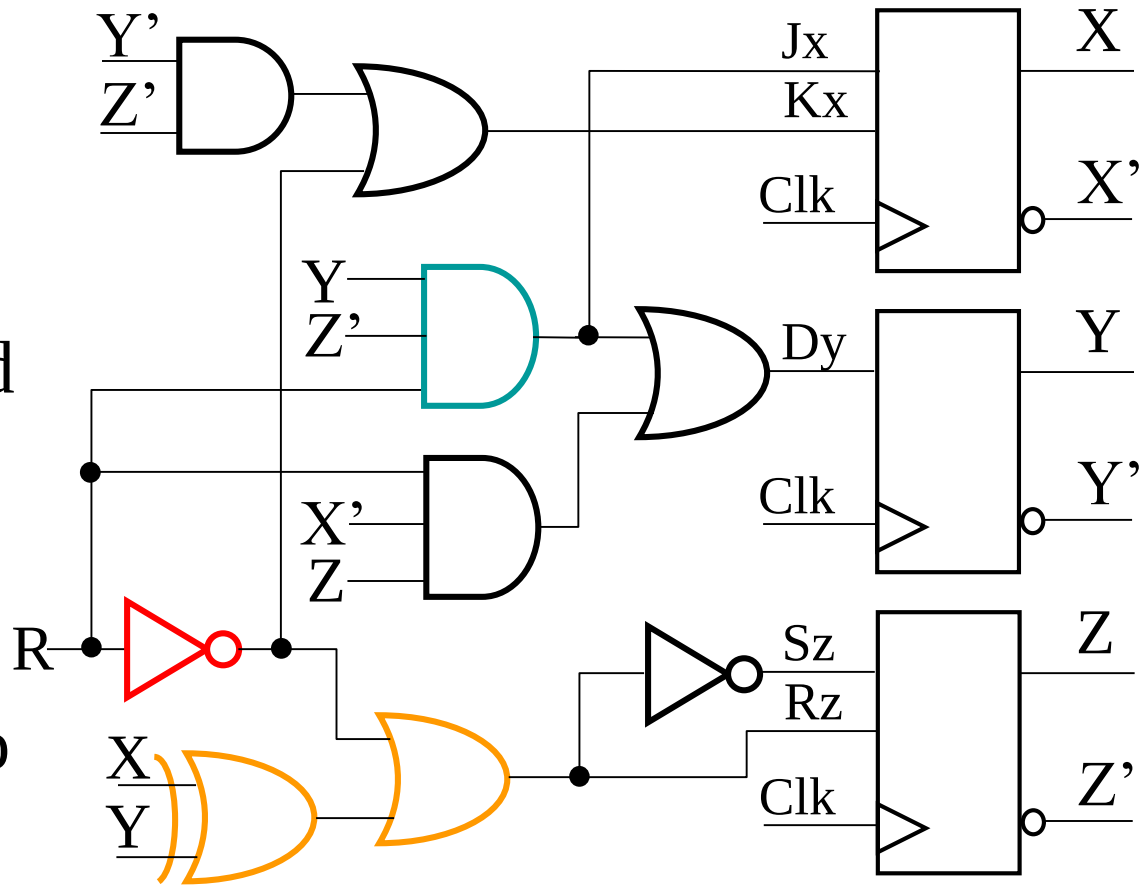
= Rz'



Rz = R' + XY' + X'Y

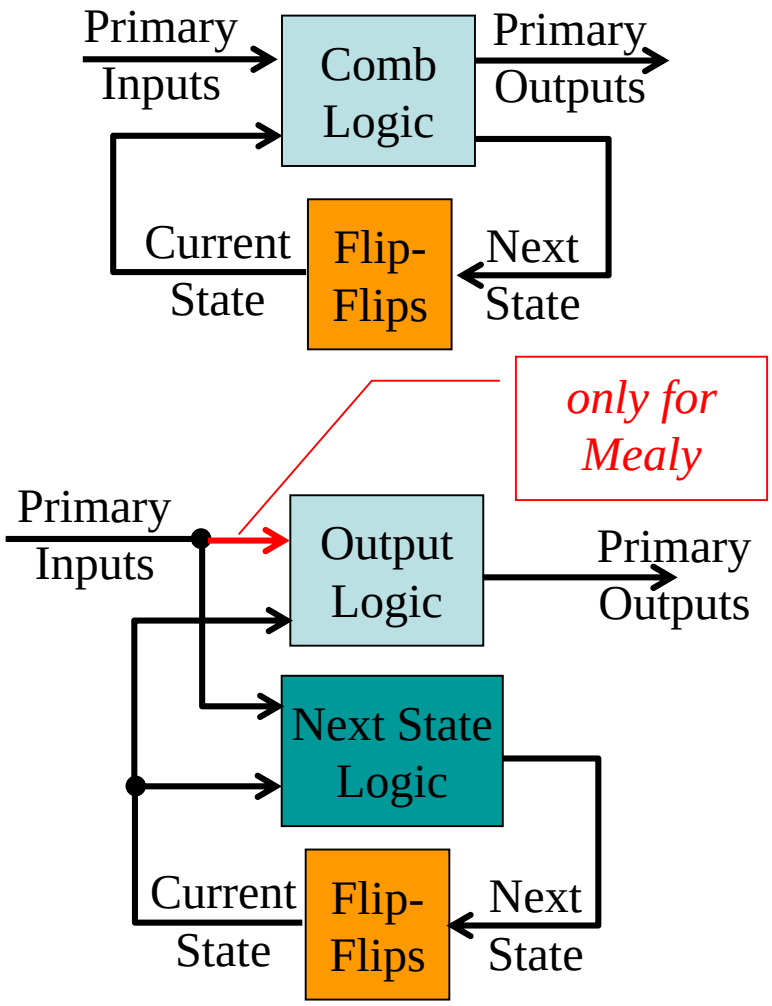
3-bit Gray Code Counter (cont)

- Logic diagram
- Then design verification via logic simulation
  - Debug as needed to obtain working circuit
  - Update logic diagram, logic equations, etc. to reflect fixes



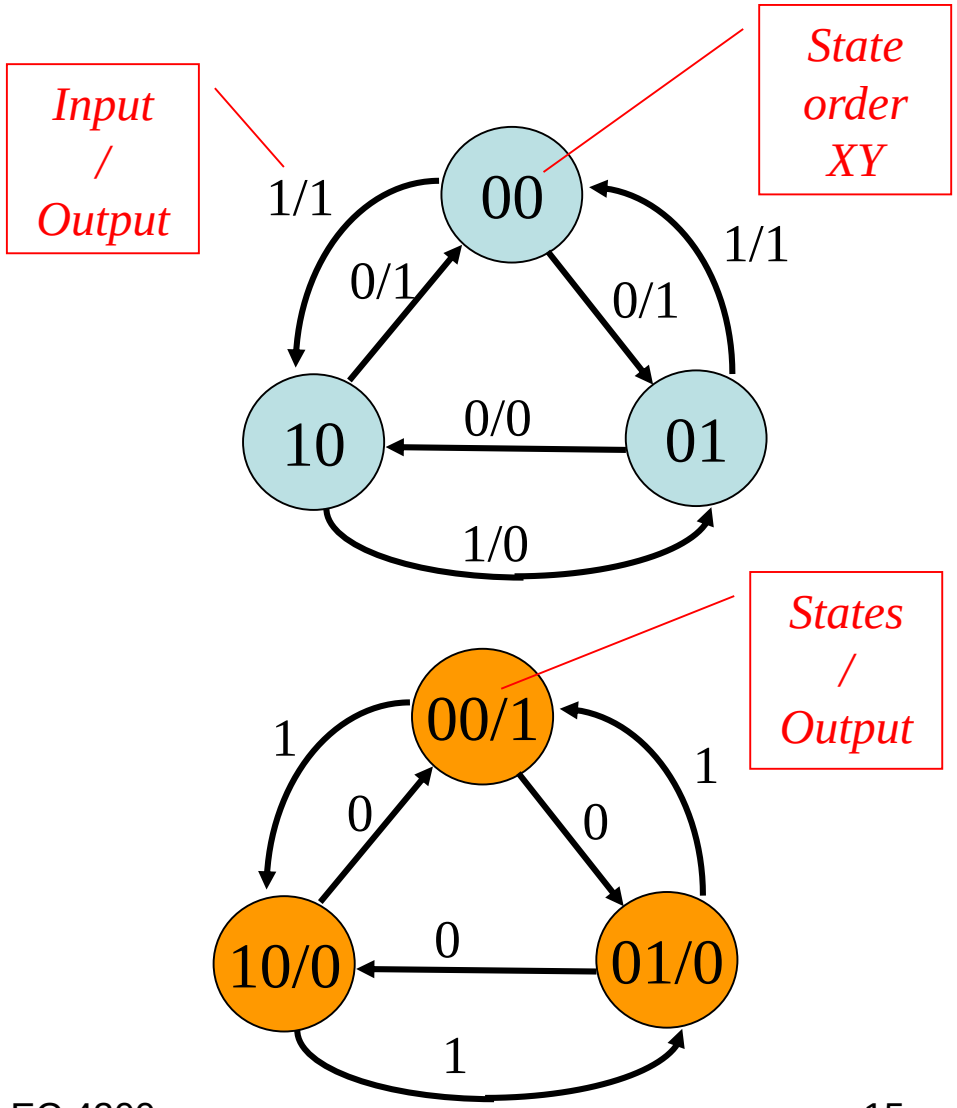
Sequential Logic Models

- Huffman model consists of two types:
  - Mealy model (aka Mealy machine)
    - Outputs are function inputs and current state
      - Outputs can change when inputs change or when current state changes
  - Moore model (aka Moore machine)
    - Outputs are function of current state only
      - Outputs can change only when current state changes

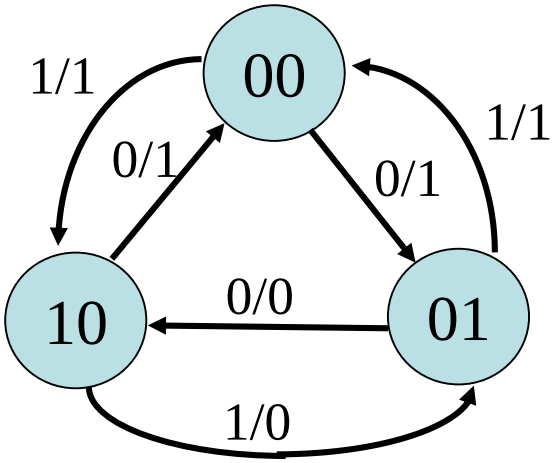


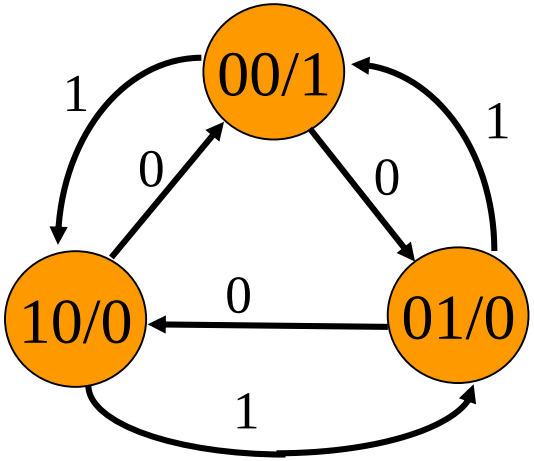
Mealy & Moore State Diagrams

- Mealy model
  - Outputs associated with state transition
  - Output values shown with inputs
- Moore model
  - Outputs associated with states only
  - Output values shown with states



Mealy & Moore State Tables





In	X	Y	X^+	Y^+	D_X	D_Y	O_{Mealy}	O_{Moore}
0	0	0	0	1	0	1	1	1
0	0	1	1	0	1	0	0	0
0	1	0	0	0	0	0	1	0
1	0	0	1	0	1	0	1	1
1	0	1	0	0	0	0	1	0
1	1	0	0	1	0	1	0	0
0	1	1	X	X	X	X	X	X

Note: next state (next state logic) is same for both Mealy & Moore – only output is different

Mealy & Moore Design Examples

In this example the Dx and Dy circuits are the same for both Mealy and Moore
But the outputs circuits are different with the Moore being a function of X and Y only

In \ XY	00	01	11	10
0	0	1	X	0
1	1	0	X	0

$D_X = In'Y + InX'Y'$

In \ XY	00	01	11	10
0	1	0	X	1
1	1	1	X	0

$O_{Mealy} = In'Y' + InX'$

In \ XY	00	01	11	10
0	1	0	X	0
1	0	0	X	1

$D_Y = InX + In'X'Y'$

In \ XY	00	01	11	10
0	1	0	X	0
1	1	0	X	0

$O_{Moore} = X'Y'$

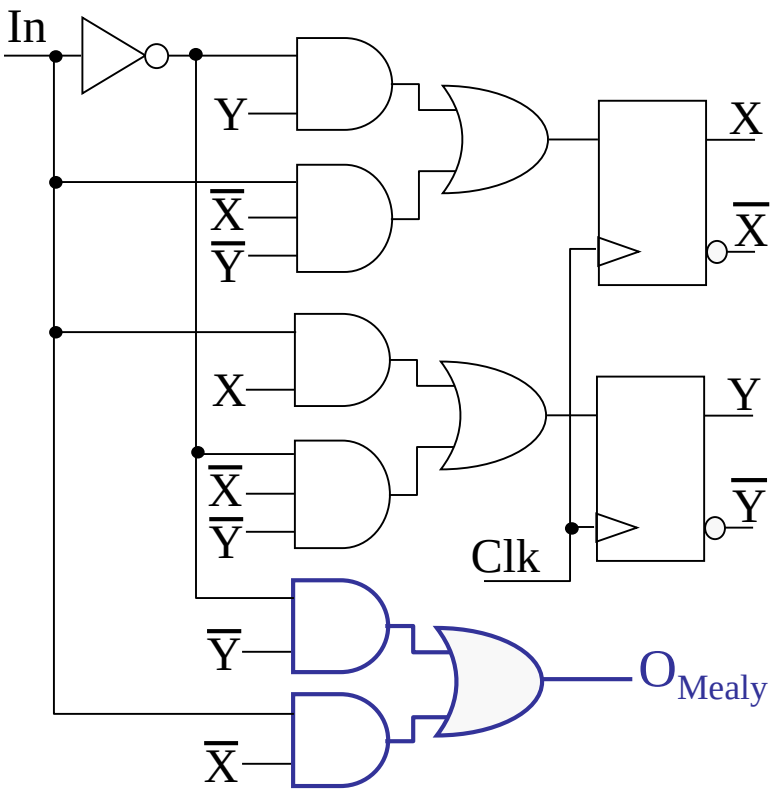
Mealy & Moore Design Examples

$O_{Mealy} = In'Y' + InX'$

$D_X = In'Y + InX'Y'$

$D_Y = InX + In'X'Y'$

$O_{Moore} = X'Y'$

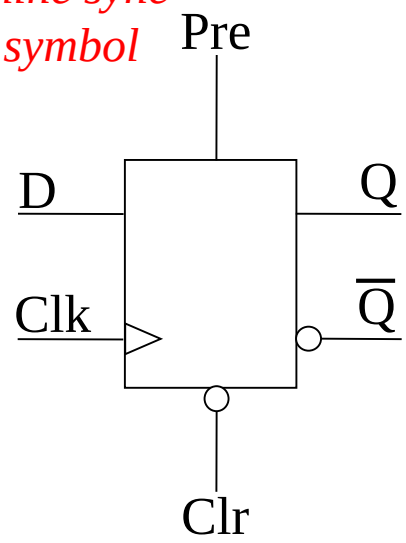


Note: O_{Mealy} ***is*** a function of In but O_{Moore} ***is not*** a function of In

Flip-Flop Initialization

- Preset (aka set) => $Q^+ = 1$
- Clear (aka reset) => $Q^+ = 0$
- Some flip-flops have:
  - Both preset and clear (set and reset)
  - A preset or a clear
  - Neither (JK & SR flops have set/reset functions)
- Preset and/or clear can be
  - Active high or active low
  - Synchronous => with respect to active edge of clock
  - Asynchronous => independent of clock edges
- Initialization important for:
  - logic simulation to remove undefined logic values
    - 2, 3, U, etc.
  - system operation to put system in a known state

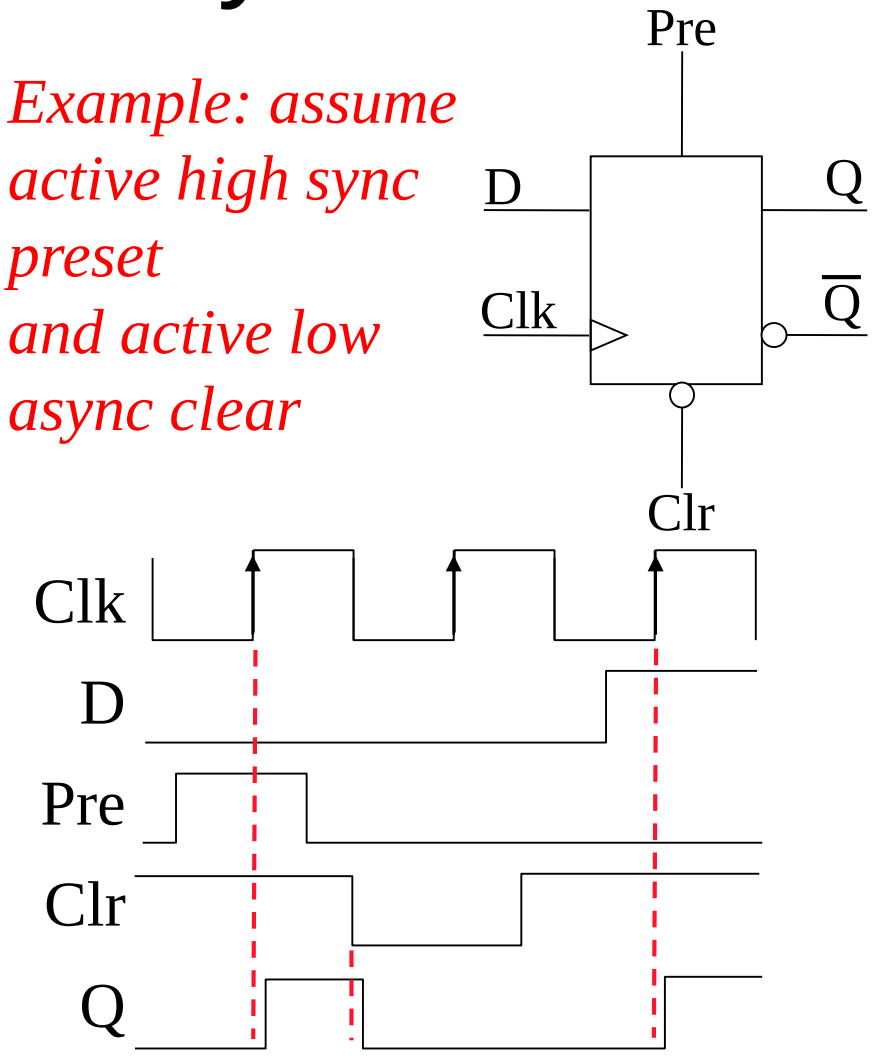
Typical logic symbol with active high preset and active low clear Cannot determine sync or async from symbol



Synchronous vs. Asynchronous

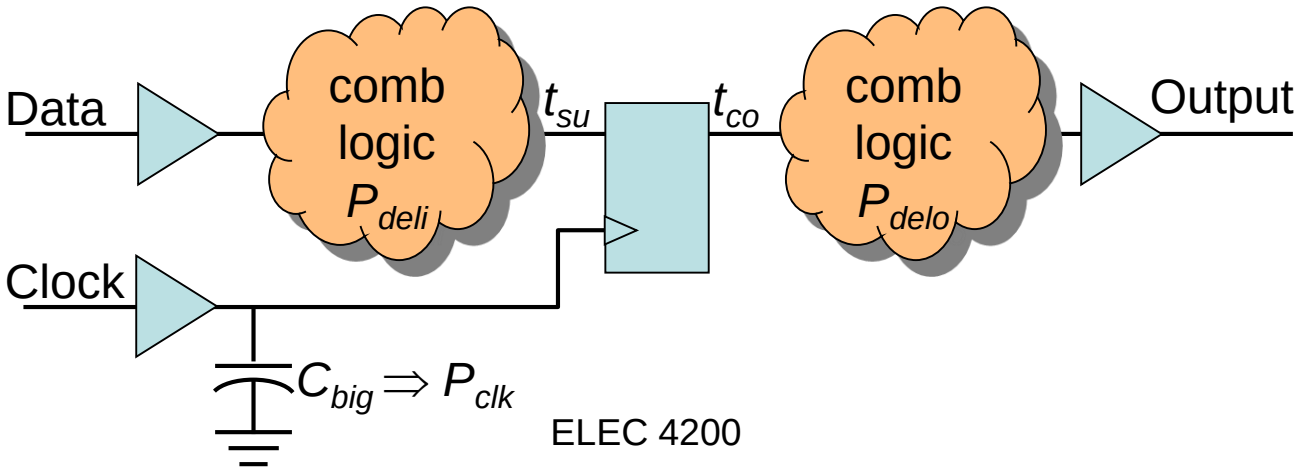
- Synchronous => states of memory elements change only with respect to active edge of clock
- Asynchronous => states of memory elements can change without an active edge of clock
  - Asynchronous designs often have timing problems

Example: assume active high sync preset and active low async clear



System-Level Timing

- System set-up time: $P_{deli} + P_{bufi} + t_{su} - P_{clk(min)}$
  - $P_{deli} + P_{bufi} + t_{su}$
- System hold time: $t_h + P_{clk} - P_{deli(min)} - P_{bufi(min)}$
  - $t_h + P_{clk}$
- System clock-to-output: $t_{co} + P_{delo} + P_{bufo} + P_{clk}$
- *Minimum times are difficult to guarantee*
  - *Typically assume 0*



System-Level Timing

- System set-up time: $P_{bufi} + t_{su(latch)} - P_{clk(input)min}$
- System hold time: $t_{h(latch)} + P_{clk(input)} - P_{bufi(min)}$
- System clock-to-output: $t_{co} + P_{bufo} + P_{clk(output)}$
- Improvement techniques:
  - Re-clock signals onto/off subcircuit, chip, PCB, or system
  - Fanout clock into input, main, and output clocks
  - 0-hold-time latches on input signals