## Analysis of Sequential Logic

- For each flip-flop, derive excitation equations, then next-state equations
- Derive next-state table, where columns represent sets of input values
  - Here "clk" is the only input
  - Each entry = value after the next rising clock edge
- Alternate views:
  - State diagram
    - Arcs are labeled with input value that causes the transition
  - Timing diagram
    - Here we've made some assumptions as to starting state and input values (cnt)
    - To fully characterize the circuit, we'd have to compute timing diagrams for all other combinations



## Analysis of Several Modulo-4 Counters

- First version — no outputs
- Second version
  - Output (Y) depends solely on present state — this is a called a *state-based* or *Moore-type* finite state machine (FSM)
  - Add a column to the next-state table for each output, also add to state diagram
- Third version
  - Output (Y) depends on input values as well as present state — this is a called a *input-based* or *Mealy-type* finite state machine (FSM)
  - Every entry in the next-state table now shows its corresponding output value
  - In state diagram, output is now associated with each arc rather than each state



## Synthesis of Sequential Logic

- First step: generate a state diagram and / or next-state and output tables
- Example: modulo-3 up / down counter
  - Input C = count enable
    - Counts when C = 1, stops when C = 0
  - Input D = count direction
    - Counts up when D = 0, down when D = 1
  - Output Y = 1 when counter reaches 2 while counting up, or 0 when counting down
- Has to remember 0, 1, 2 => 2 flip-flops
  - Up-sequence of three states
  - Down-sequence of three states
  - Might need to change direction when counting; continue from peer state in other sequence



## Synthesis of Sequential Logic (cont.)

- Second step: state minimization
  - Reduce the number of states so as to reduce the number of flip-flops
    - If reduce below present power-of-2 level
    - From 6 to 4 is good; 6 to 5 doesn't help
  - The behavior of 2 FSMs is said to be equivalent if they produce the same sequence of output symbols for every sequence of input symbols
    - May be equivalent even if they have different numbers of states, or different state transitions for every input sequence
  - State minimization = merge equivalent states into a single state
  - 2 states are equivalent iff:
    - Both states produce the same output symbol for every input symbol
    - Both states have equivalent next states for every input symbol

# Synthesis of Sequential Logic (cont.)

- Third step: state encoding
  - How will the states be encoded?
    - 4! = 4•3•2•1 = 24 possible encodings
  - Could try all, but that would be tedious
  - Heuristics for state minimization:
    - Minimize total number of bit changes
    - Others…
  - One-hot state encoding — one flip-flop for each state (at any given time, only a single flip-flop is set to 1)
  - Three possible encodings for the modulo-3 up / down counter: (Q1, Q0)
    - S0 = 0 0, S1 = 0 1, S2 = 1 0
    - S0 = 0 1, S1 = 0 0, S2 = 1 0
    - S0 = 0 0 1, S1 = 0 1 0, S2 = 1 0 0
  - Determine cost and delay for each, and select the best one