Overview

- Last lecture
  - Sequential Logic Examples
- Today
  - State encoding
    - One-hot encoding
    - Output encoding
    - FSM partitioning

State encoding

- Assume n state bits and m states
  - $2^n! / (2^n - m)!$ possible encodings $[m \geq n \geq \log_2(m)]$
    - From binomial expansion
    - Example: 3 state bits, 4 states, 1680 possible state assignments
- Hard problem, with no known algorithmic solution
  - Can try heuristic approaches
  - Can try to optimize some metric
    - FSM size (amount of logic and number of FFs)
    - FSM speed (depth of logic and fanout)
    - FSM dependencies (decomposition)
- Need to consider startup
  - Self-starting FSM or explicit reset input

State-encoding strategies

- No guarantee of optimality
  - An intractable problem
- Most common strategies
  - Binary (sequential) – number states as in the state table
  - Random – computer tries random encodings
  - Heuristic – rules of thumb that seem to work well
    - e.g. Gray-code – try to give adjacent states (states with an arc between them) codes that differ in only one bit position
  - One-hot – use as many state bits as there are states
  - Output – use outputs to help encode states

One-hot encoding

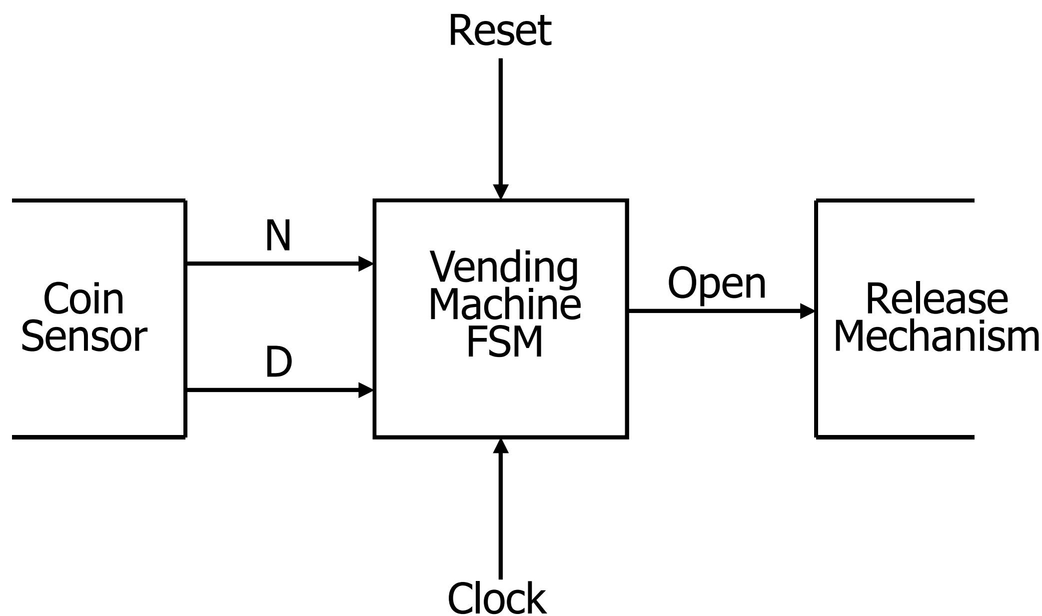
- One-hot: Encode n states using n flip-flops
  - Assign a single "1" for each state
    - Example: 0001, 0010, 0100, 1000
  - Propagate a single "1" from one flip-flop to the next
    - All other flip-flop outputs are "0"
- The inverse: One-cold encoding
  - Assign a single "0" for each state
    - Example: 1110, 1101, 1011, 0111
  - Propagate a single "0" from one flip-flop to the next
    - All other flip-flop outputs are "1"
- "almost one-hot" encoding
  - Use no-hot (000...0) for the initial (reset state)
  - Assumes you never revisit the reset state

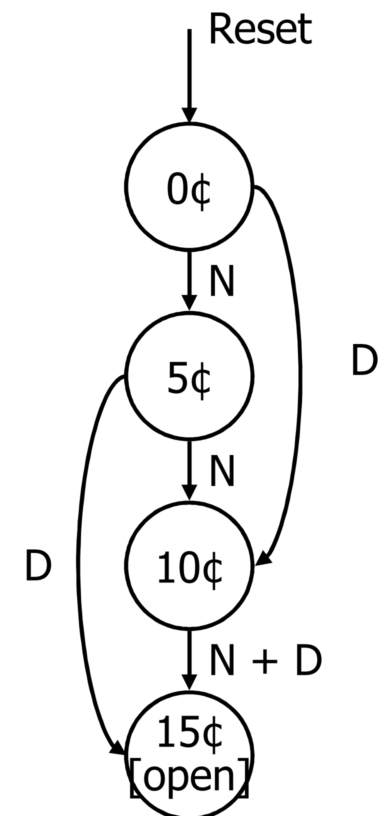
One-hot encoding (con't)

- Often the best approach for FPGAs
  - FPGAs have many flip-flops
  - One-hot machines use the least next-state logic
- Draw FSM directly from the state diagram
  - One product term per incoming arc
  - But complex state diagram $\Rightarrow$ complex design
- One-hot designs have many possible failure modes
  - All states that aren't one-hot
  - Can create logic to reset the FSM if it enters illegal state
- Large machines require many flip-flops
  - Decompose design into smaller one-hot encoded sub-designs
    - n+m states for two machines versus n*m states for one

Vending machine again...

- Release item after receiving 15 cents
  - Single coin slot for dimes and nickels
    - Sensor specifies coin type
  - Machine does not give change





One-hot encoded transition table

present state $Q_3Q_2Q_1Q_0$	inputs D	N	next state $D_3D_2D_1D_0$	output open
0 0 0 1	0	0	0 0 0 1	0
	0	1	0 0 1 0	0
	1	0	0 1 0 0	0
	1	1	– – – –	–
0 0 1 0	0	0	0 0 1 0	0
	0	1	0 1 0 0	0
	1	0	1 0 0 0	0
	1	1	– – – –	–
0 1 0 0	0	0	0 1 0 0	0
	0	1	1 0 0 0	0
	1	0	1 0 0 0	0
	1	1	– – – –	–
1 0 0 0	–	–	1 0 0 0	1

$D_0 = Q_0D'N'$

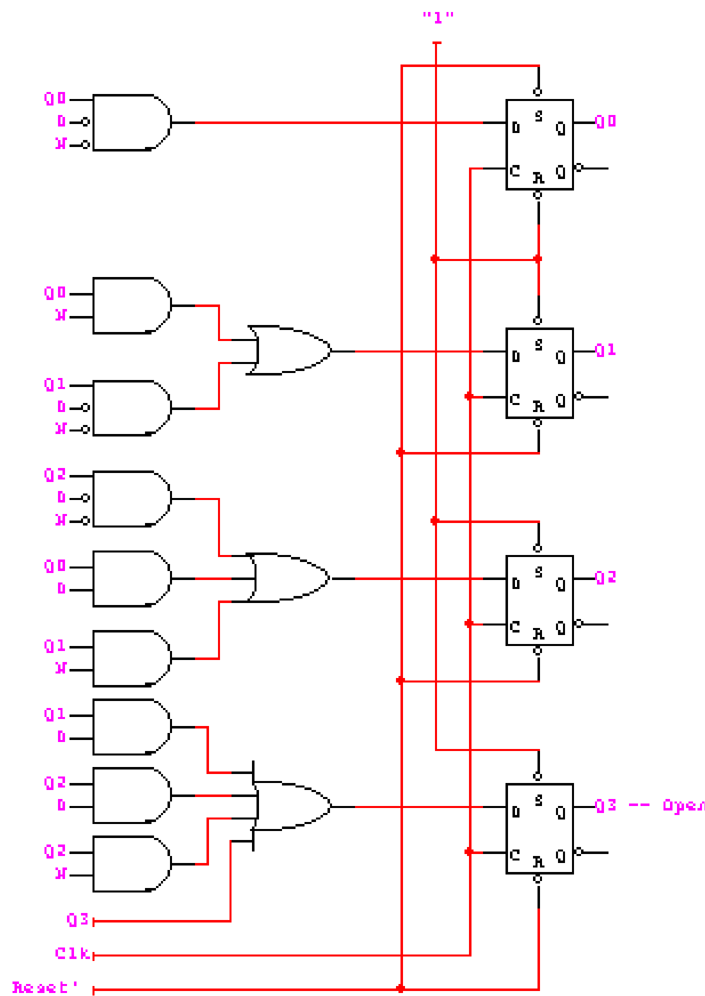
$D_1 = Q_0N + Q_1D'N'$

$D_2 = Q_0D + Q_1N + Q_2D'N'$

$D_3 = Q_1D + Q_2D + Q_2N + Q_3$

$OPEN = Q_3$

One-hot encoded vending machine



$D_0 = Q_0D'N'$

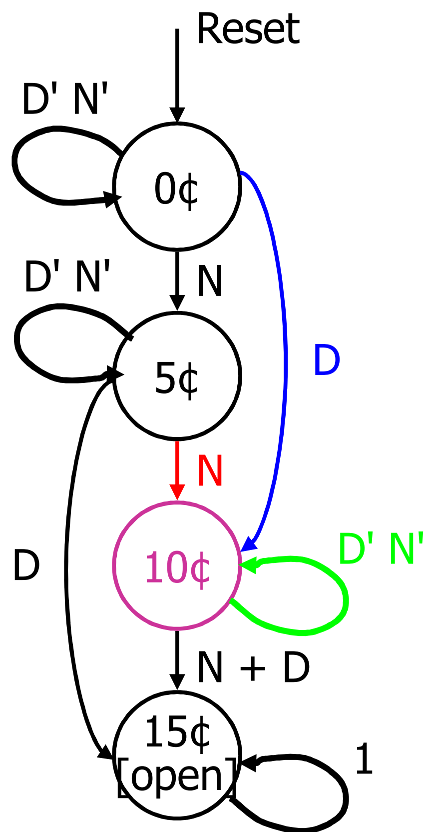
$D_1 = Q_0N + Q_1D'N'$

$D_2 = Q_0D + Q_1N + Q_2D'N'$

$D_3 = Q_1D + Q_2D + Q_2N + Q_3$

$OPEN = Q_3$

Designing from the state diagram



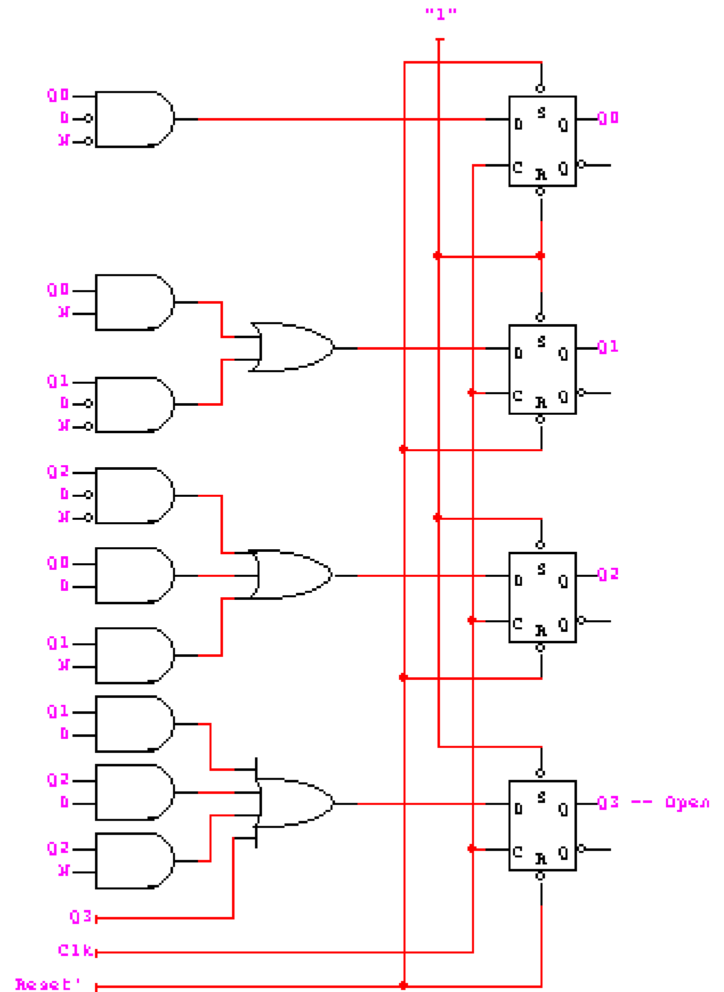
$D_0 = Q_0D'N'$

$D_1 = Q_0N + Q_1D'N'$

$D_2 = Q_0D + Q_1N + Q_2D'N'$

$D_3 = Q_1D + Q_2D + Q_2N + Q_3$

$OPEN = Q_3$



Output encoding

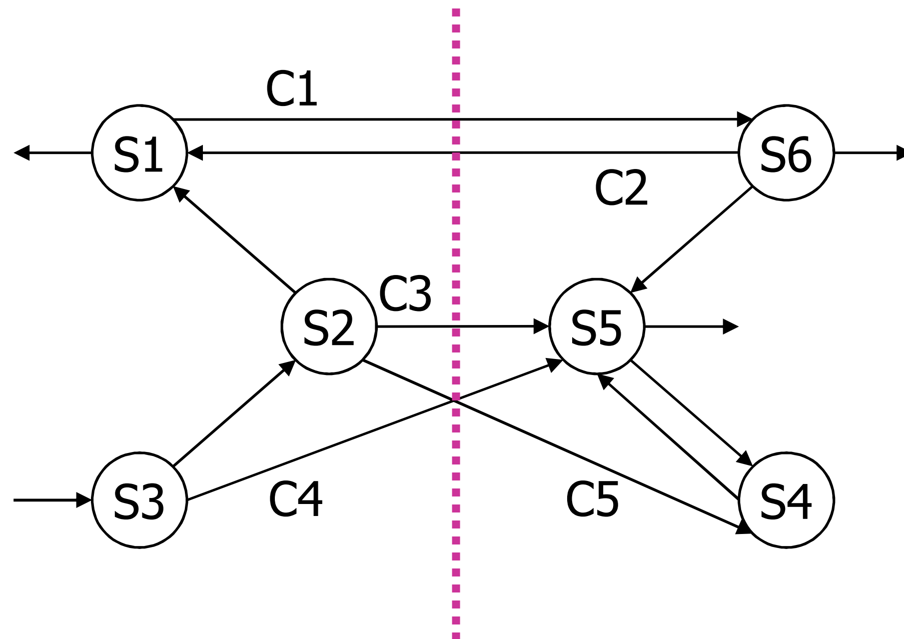
- Reuse outputs as state bits
  - Why create new functions when you can use outputs?
  - Bits from state assignments are the outputs for that state
    - Take outputs directly from the flip-flops
- ad hoc - no tools
  - Yields small circuits for most FSMs
  - Fits nicely with synchronous Mealy machines

FSM partitioning

- Break a large FSM into two or more smaller FSMs
- Rationale
  - Less states in each partition
    - Simpler minimization and state assignment
    - Smaller combinational logic
    - Shorter critical path
  - But more logic overall
- Goal
  - Minimize communication between partitions
    - Minimize wires & I/O
- Partitions are synchronous
  - Same clock!!!

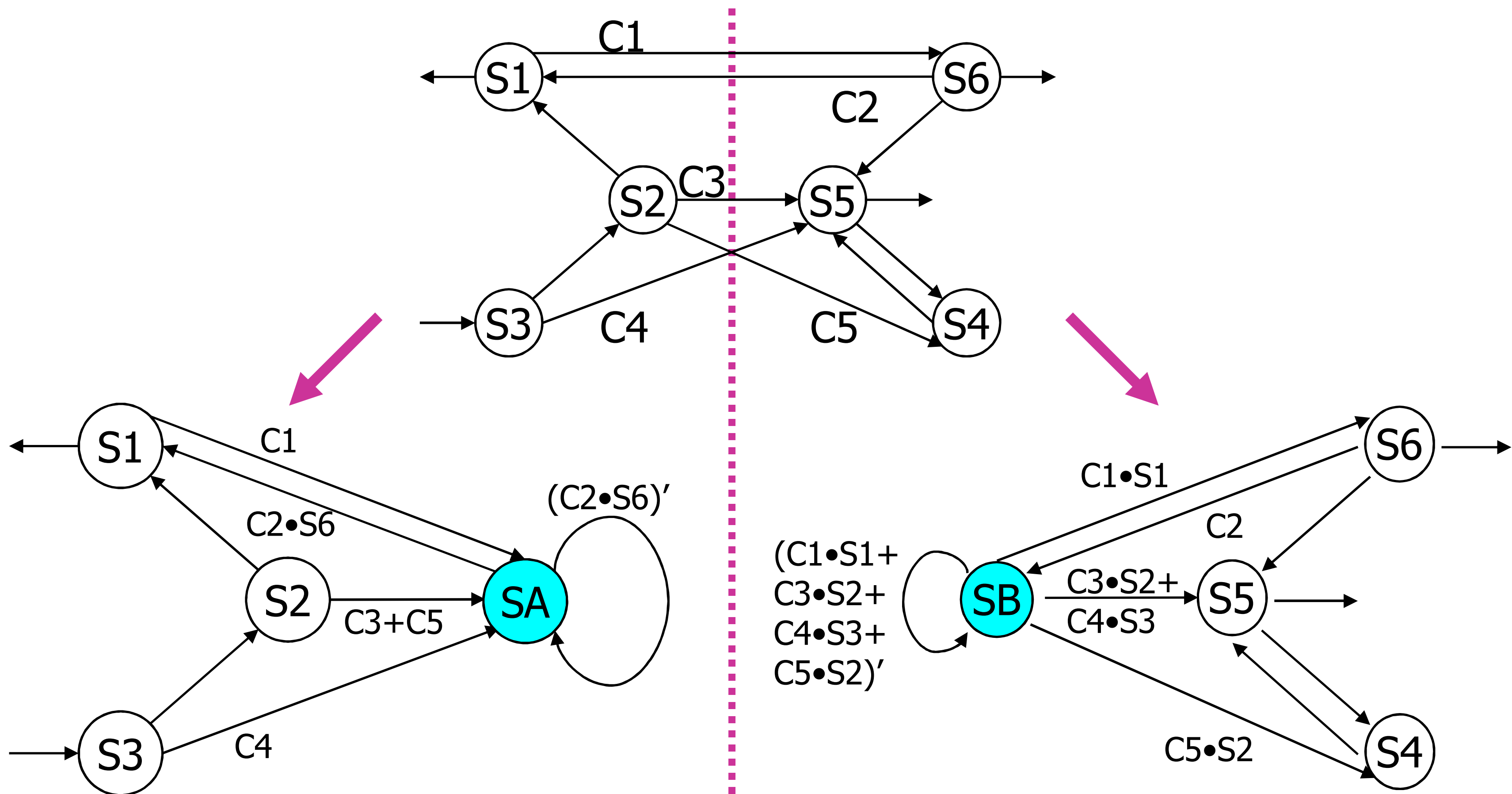
Example: Partition the machine

- Partition into two halves



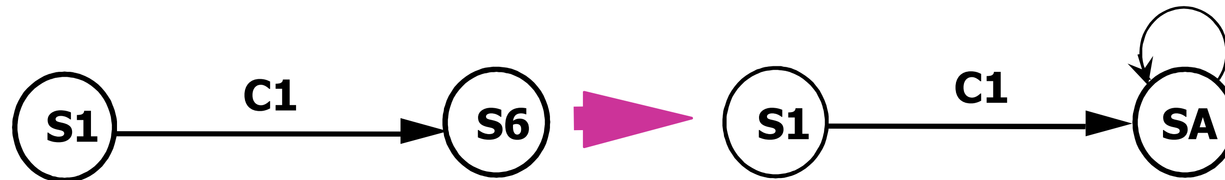
Introduce idle states

- SA and SB handoff control between machines

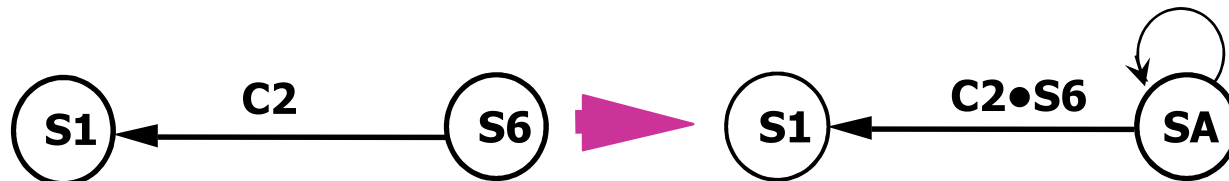


Partitioning rules

Rule #1: Source-state transformation
Replace by transition to idle state (SA)



Rule #2: Destination state transformation
Replace with exit transition from idle state

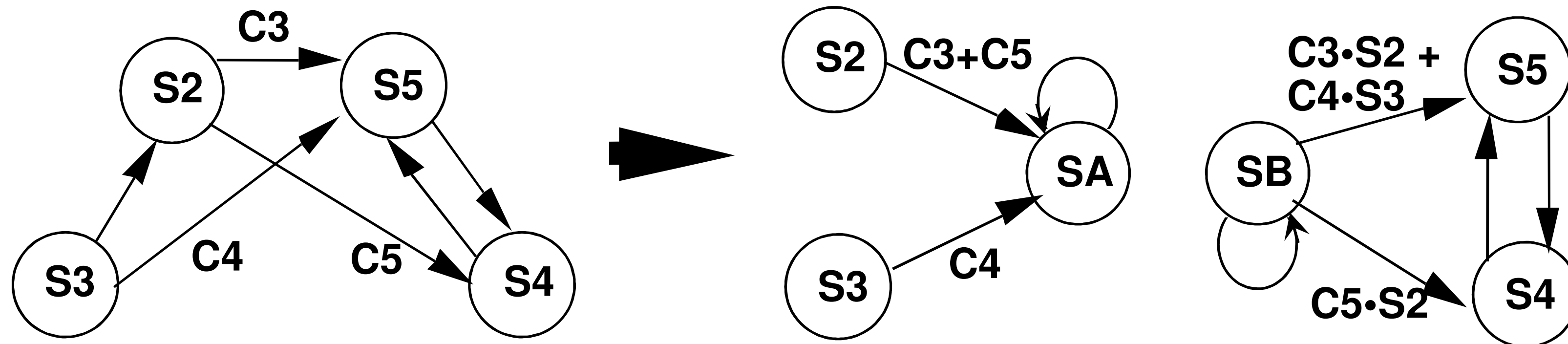


Partitioning rules (con't)

Rule #3: Multiple transitions with same source or destination

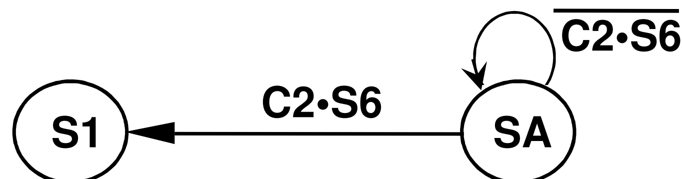
Source ⇒ Replace by transitions to idle state (SA)

Destination ⇒ Replace with exit transitions from idle state



Rule #4: Hold condition for idle state

OR exit conditions and invert

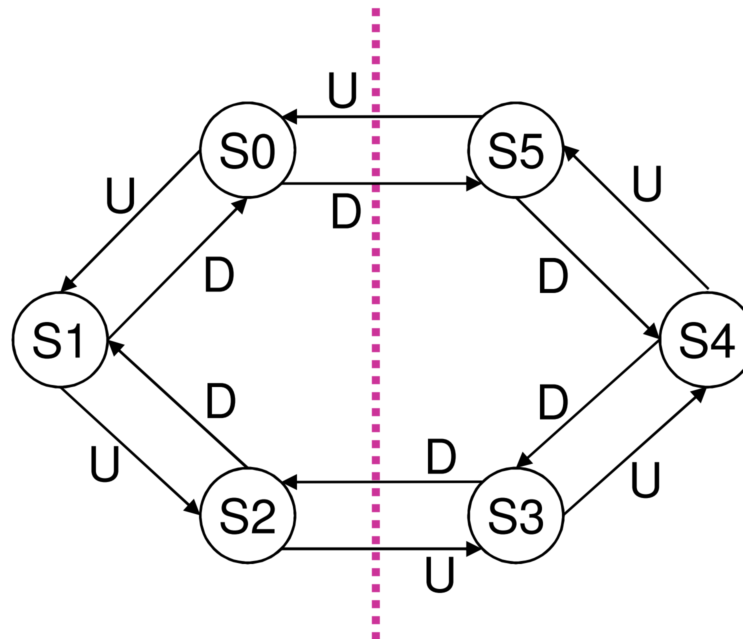


Mealy versus Moore partitions

- Mealy machines undesirable
  - Inputs can affect outputs immediately
    - "output" can be a handoff to another machine!!!
  - Inputs can ripple through several machines in one clock cycle
- Moore or synchronized Mealy desirable
  - Input-to-output path always broken by a flip-flop
  - But…may take several clocks for input to propagate to output
    - Output may derive from other side of a partition

Example: Six-state up/down counter

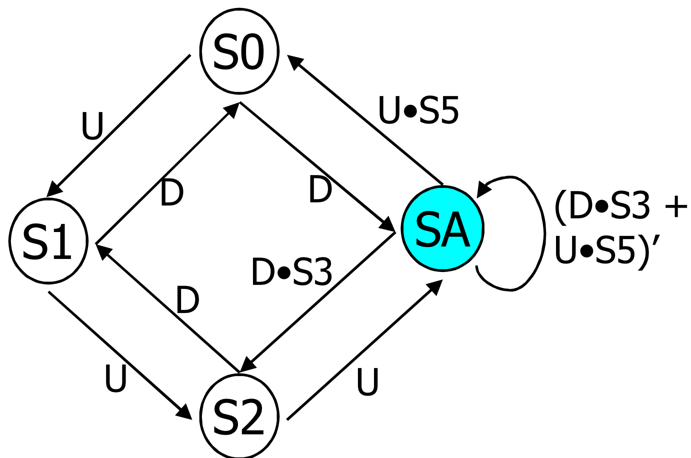
- Break into 2 parts

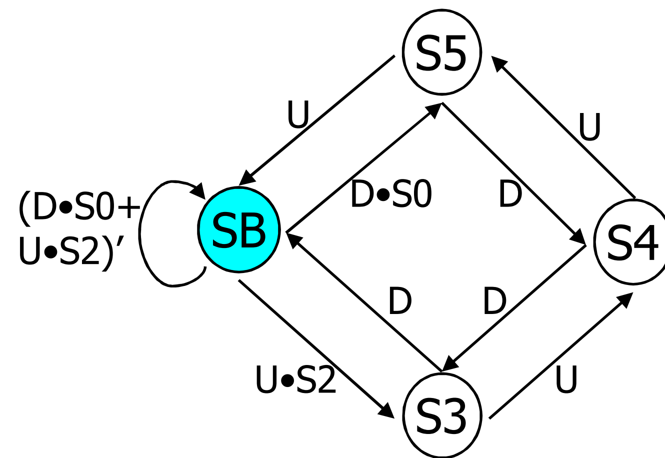


U ≡ count up
D ≡ count down

Example: 6 state up/down counter (con't)

- Count sequence S_0, S_1, S_2, S_3, S_4, S_5
  - S_2 goes to S_A and holds, leaves after S_5
  - S_5 goes to S_B and holds, leaves after S_2
  - Down sequence is similar





Minimize communication between partitions

- Ideal world: Two machines handoff control
  - Separate I/O, states, etc.
- Real world: Minimize handoffs and common I/O
  - Minimize number of state bits that cross boundary
  - Merge common outputs
- Look for:
  - Disjoint inputs used in different regions of state diagram
  - Outputs active in only one region of state diagram
  - Isomorphic portions of state diagram
    - Add states, if necessary, to make them so
  - Regions of diagram with a single entry and single exit point

Sequential logic: What you should know

- Sequential logic building blocks
  - Latches (R-S and D)
  - Flip-flops (master/slave D, edge-triggered D & T)
  - Latch and flip-flop timing (setup/hold time, prop delay)
  - Timing diagrams
  - Flip-flop clocking
  - Asynchronous inputs and metastability
  - Registers

Sequential logic: What you should know

- Counters
  - Timing diagrams
  - Shift registers
  - Ripple counters
  - State diagrams and state-transition tables
  - Counter design procedure
    1. Draw a state diagram
    2. Draw a state-transition table
    3. Encode the next-state functions
    4. Implement the design
  - Self-starting counters

Sequential logic: What you should know

- Finite state machines
  - Timing diagrams (synchronous FSMs)
  - Moore versus Mealy versus registered Mealy
  - FSM design procedure
    1. Understand the problem (state diagram & state-transition table)
    2. Determine the machine's states (minimize the state diagram)
    3. Encode the machine's states (state assignment)
    4. Design the next-state logic (minimize the combinational logic)
    5. Implement the FSM
  - FSM design guidelines
    - Separate datapath and control
  - One-hot encoding
  - FSM partitioning procedure