Lecture 23

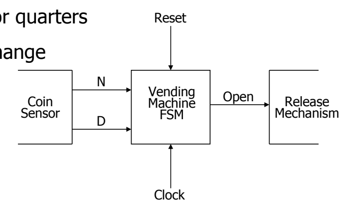
- Logistics
  - HW8 due Wednesday, March 11
  - Ant extra credit due Friday, March 13
  - Final exam, Wednesday March 18, 2:30-4:20 pm here
  - Review session Monday, March 16, 4:30 pm, Place TBA
- Last lecture
  - General FSM Minimization
- Today
  - State encoding
    - One-hot encoding
    - Output encoding
  - State partitioning

FSM design

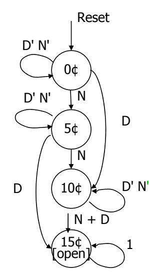
- FSM-design procedure
  1. State diagram
  2. state-transition table
  3. State minimization
  4. State encoding
  5. Next-state logic minimization
  6. Implement the design

Usual example: A vending machine

- 15 cents for a cup of coffee
- Doesn't take pennies or quarters
- Doesn't provide any change



A vending machine: After state minimization



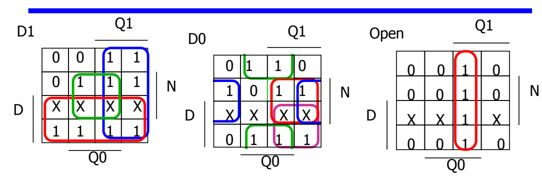
present state	inputs D	inputs N	next state	output open
0¢	0	0	0¢	0
	0	1	5¢	0
	1	0	10¢	0
	1	1	–	–
5¢	0	0	5¢	0
	0	1	10¢	0
	1	0	15¢	0
	1	1	–	–
10¢	0	0	10¢	0
	0	1	15¢	0
	1	0	15¢	0
	1	1	–	–
15¢	–	–	15¢	1

symbolic state table

A vending machine: State encoding

present state Q1	present state Q0	inputs D	inputs N	next state D1	next state D0	output open
0	0	0	0	0	0	0
		0	1	0	1	0
		1	0	1	0	0
		1	1	–	–	–
0	1	0	0	0	1	0
		0	1	1	0	0
		1	0	1	1	0
		1	1	–	–	–
1	0	0	0	1	0	0
		0	1	1	1	0
		1	0	1	1	0
		1	1	–	–	–
1	1	–	–	1	1	1

A vending machine: Logic minimization

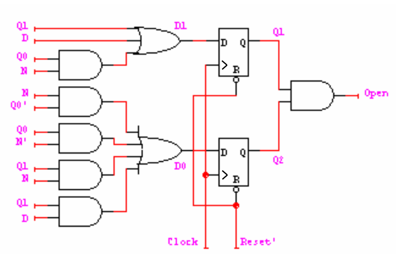


D1 = Q1 + D + Q0 N

D0 = Q0' N + Q0 N' + Q1 N + Q1 D

OPEN = Q1 Q0

A vending machine: Implementation



State encoding

- Assume n state bits and m states
  - $2^n! / (2^n - m)!$ possible encodings
    - Example: 3 state bits, 4 states, 1680 possible state assignments
- Want to pick state encoding strategy that results in optimizing your criteria
  - FSM size (amount of logic and number of FFs)
  - FSM speed (depth of logic and fan-in/fan-out)
  - FSM ease of design or debugging

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
  - Hybrid – mix of a few different ones (e.g. One-hot + heuristic)

One-hot encoding

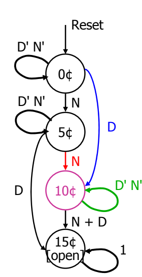
- One-hot: Encode n states using n flip-flops
  - Assign a single "1" for each state
    - Example: 0001, 0010, 0100, 1000
  - Propagate a single "1" from one flip-flop to the next
    - All other flip-flop outputs are "0"
- The inverse: One-cold encoding
  - Assign a single "0" for each state
    -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

One-hot encoding (con't)

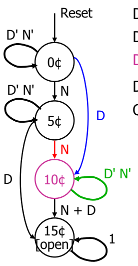
- Often the best/convenient approach for FPGAs
  - FPGAs have many flip-flops
- Draw FSM directly from the state diagram
  - + One product term per incoming arc
  - - Complex state diagram ⇒ complex design
  - - Many states ⇒ many flip flops

Vending Machine: 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



Advantage of one-hot encoding: Designing from the state diagram



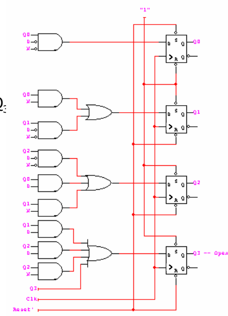
$D_0 = Q_0D'N'$

$D_1 = Q_0N + Q_1D'N'$

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

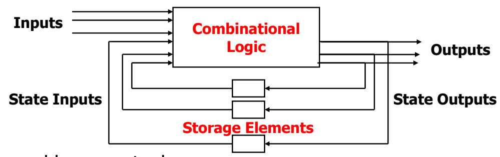
$D_3 = Q_1D + Q_2D + Q_2N + Q_3$

$OPEN = Q_3$



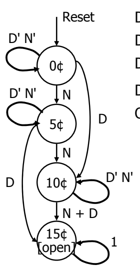
Output encoding

- 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

Vending machine --- already in output encoding form



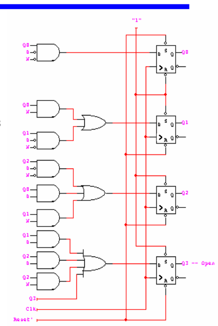
$D_0 = Q_0D'N'$

$D_1 = Q_0N + Q_1D'N'$

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

$D_3 = Q_1D + Q_2D + Q_2N + Q_3$

$OPEN = Q_3$

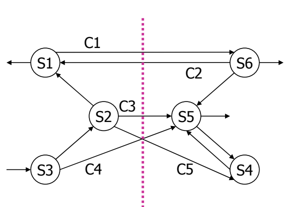


FSM partitioning

- 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
- Partitions are synchronous
  - Same clock!!!

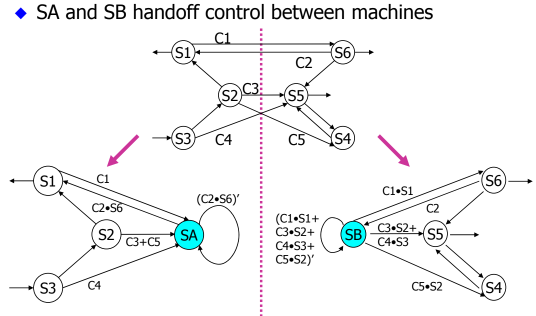
Example: Partition the machine

- Partition into two halves



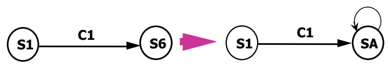
Introduce idle states for each partition

- SA and SB handoff control between machines

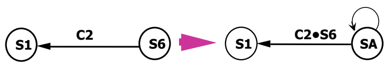


Partitioning rules

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

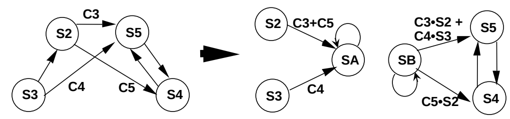


Rule #2: Destination state transformation
Replace with exit transition from idle state

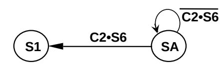


Partitioning rules (con't)

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



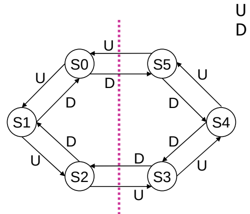
Rule #4: Hold condition for idle state
"OR exit conditions and invert"



Example: Six-state up/down counter

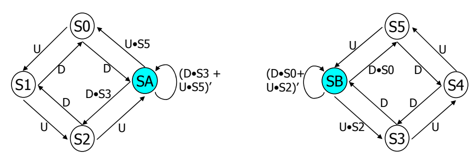
- Break into 2 parts

U ≡ count up
D ≡ count down



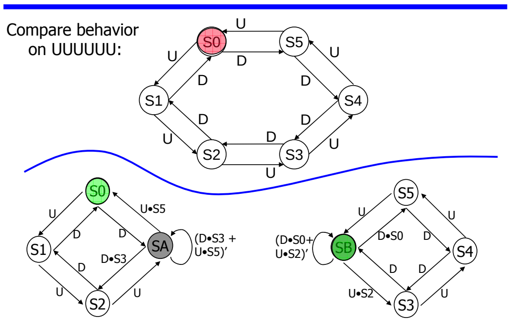
Example: 6 state up/down counter

- 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



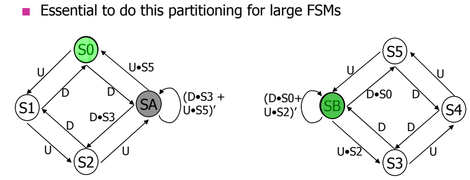
Example: 6 state up/down counter

Compare behavior
on UUUUUU:



Example: 6 state up/down counter

- 4-state machines need 2 state bits each – total 4 state bits
  - Enough to represent 16 states, though the combination of the two FSMs has only 6 different configurations
- Why do this?
  - Each FSM may be much simpler to think about (and design logic for) than the original FSM (not here, though)
  - Essential to do this partitioning for large FSMs



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

Mealy versus Moore partitions

- Mealy machine partitioning is undesirable
  - Inputs can affect outputs immediately
    - "output" can be a handoff to another machine!!!
- Moore machine partitioning is desirable
  - Input-to-output path always broken by a flip-flop
  - But…may take several clock cycles for input to propagate to output