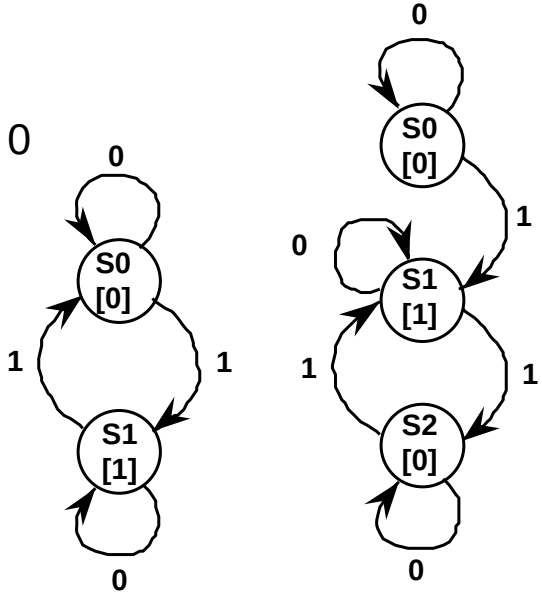
Ch 8. Working with Finite State Machines

State Minimization / Reduction

- Motivation
  - Odd Parity Checker example
    - Two alternative state diagrams
    - Identical output behavior on all input strings
    - FSMs are equivalent, but require different implementations
    - S0, S2 are equivalent states
      - Both output a 0
      - Both transition to S1 on a 1 and self-loop on a 0



State Minimization / Reduction (cont'd)

- Goal
  - Identify and combine equivalent states
    - Equivalent states:
      - same outputs (Mealy: for all input combinations)
      - for all input combinations, transition to same or equivalent states
  - Design state diagram without concern for # of states, reduce later
  - Implement FSM with fewest possible states
    - Reduce the number of gates and flip-flops needed for implementation

State Minimization / Reduction (cont'd)

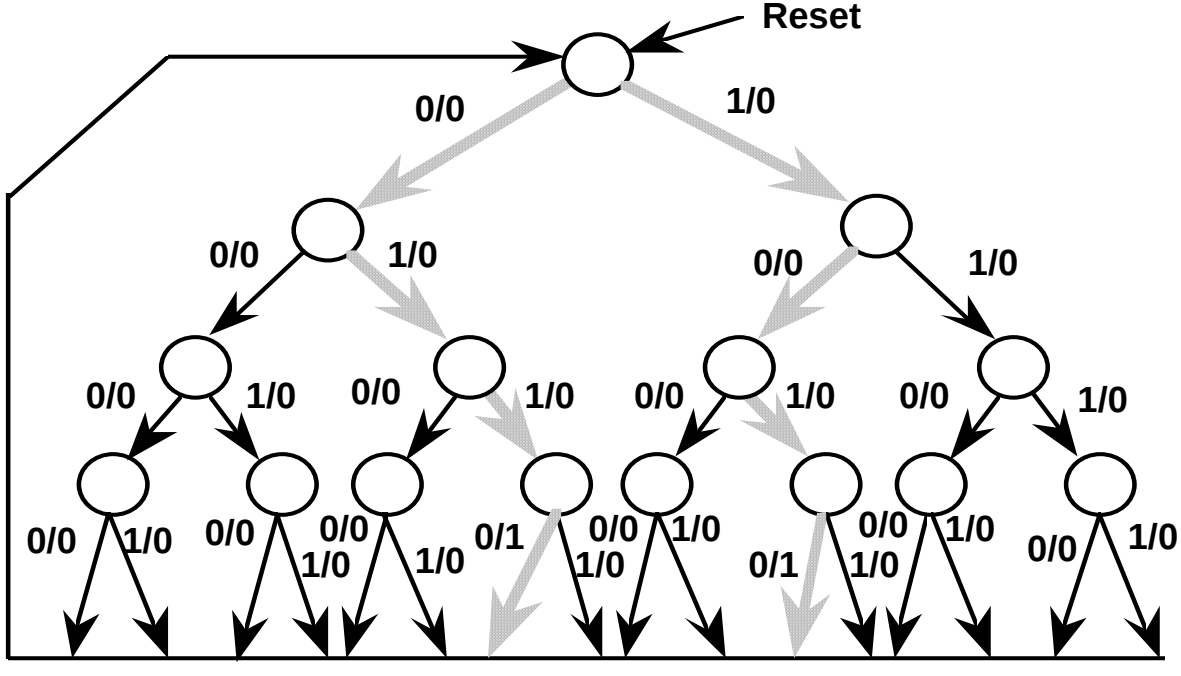
- Example specification
  - Name: four-bit sequence (0110 or 1010) detector
  - Input: X = {0, 1}
  - Output: Z = {0, 1}
  - Behavior:

$$Z = \begin{cases} 1 \text{ if each 4-bit input sequence (no overlap) is 0110 or 1010} \\ 0 \text{ otherwise} \end{cases}$$

- Sample behavior
  - X = 0010 0110 1100 1010 0011 …
  - Z = 0000 0001 0000 0001 0000 …

State Minimization / Reduction (cont'd)

- Initial State Diagram (of a Mealy implementation)
  - There are 16 unique paths through the state diagram, one for each possible 4-bit pattern.
  - 15 states, and 30 transitions.



State Minimization / Reduction (cont'd)

- Upper bound of #states and #transitions for n-length bit pattern
  - #states = $\sum_{i=0}^{n-1} 2^i = 2^n - 1$
  - #transition = 2 x #states = $2(2^n-1)$
  - Example: n = 3: 7 states, 14 transitions
    n = 4: 15 states, 30 transitions

State Minimization / Reduction (cont'd)

- Algorithm sketch for state reduction
  - 1. group together states that have the same outputs
    - These states are potentially equivalent.
  - 2. examine the transitions to see if they go to the same next state for every input combination
    - If they do, the states are equivalent.
    - combine them into a renamed new state.
    - change all transitions to the states into the newly combined states.
  - 3. repeat (1)~(2) until no additional states can be combined
  - polynomial time procedure

Row-Matching method

- Initial state transition table

Past Input Seq.	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_7	S_8	0	0
01	S_4	S_9	S_{10}	0	0
10	S_5	S_{11}	S_{12}	0	0
11	S_6	S_{13}	S_{14}	0	0
000	S_7	S_0	S_0	0	0
001	S_8	S_0	S_0	0	0
010	S_9	S_0	S_0	0	0
011	S_{10}	S_0	S_0	1	0
100	S_{11}	S_0	S_0	0	0
101	S_{12}	S_0	S_0	1	0
110	S_{13}	S_0	S_0	0	0
111	S_{14}	S_0	S_0	0	0

Row-Matching:
the same next-states and output values
S_{10} and S_{12} -> S_{10}'

Row-Matching method (cont'd)

- Revised state transition table after S10 and S12 are combined

Past Input Seq.	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_7	S_8	0	0
01	S_4	S_9	S'_{10}	0	0
10	S_5	S_{11}	S'_{10}	0	0
11	S_6	S_{13}	S_{14}	0	0
000	S_7	S_0	S_0	0	0
001	S_8	S_0	S_0	0	0
010	S_9	S_0	S_0	0	0
011 or 101	S'_{10}	S_0	S_0	1	0
100	S_{11}	S_0	S_0	0	0
110	S_{13}	S_0	S_0	0	0
111	S_{14}	S_0	S_0	0	0

Row-Matching method (cont'd)

- Row-matching iteration

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_7	S_8	0	0
01	S_4	S_9	S'_{10}	0	0
10	S_5	S_{11}	S'_{10}	0	0
11	S_6	S_{13}	S_{14}	0	0
000	S_7	S_0	S_0	0	0
001	S_8	S_0	S_0	0	0
010	S_9	S_0	S_0	0	0
011 or 101	S'_{10}	S_0	S_0	1	0
100	S_{11}	S_0	S_0	0	0
110	S_{13}	S_0	S_0	0	0
111	S_{14}	S_0	S_0	0	0

Row-Matching method (cont'd)

- Row-matching iteration (cont'd)

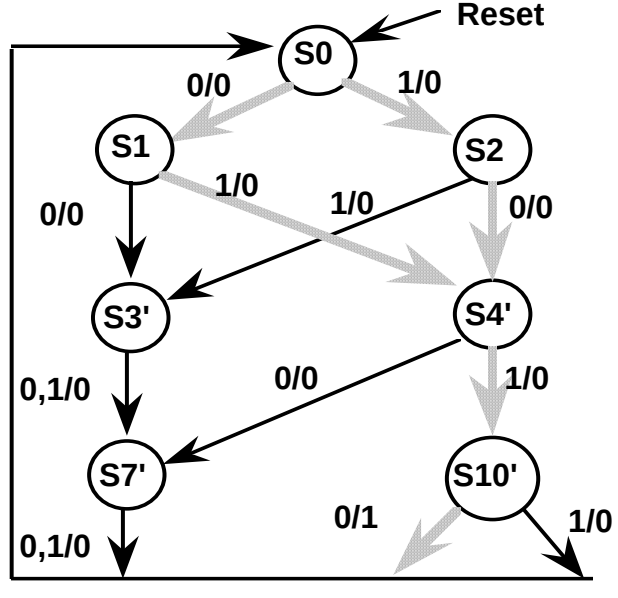
Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S'_7	S'_7	0	0
01	S_4	S'_7	S'_{10}	0	0
10	S_5	S'_7	S'_{10}	0	0
11	S_6	S'_7	S'_7	0	0
not (011 or 101)	S'_7	S_0	S_0	0	0
011 or 101	S'_{10}	S_0	S_0	1	0

Row-Matching method (cont'd)

- Final reduced state transition table

		Next State		Output	
Input Sequence	Present State	X=0	X=1	X=0	X=1
Reset	S0	S1	S2	0	0
0	S1	S3'	S4'	0	0
1	S2	S4'	S3'	0	0
00 or 11	S3'	S7'	S7'	0	0
01 or 10	S4'	S7'	S10'	0	0
not (011 or 101)	S7'	S0	S0	0	0
011 or 101	S10'	S0	S0	1	0

- Corresponding State Diagram

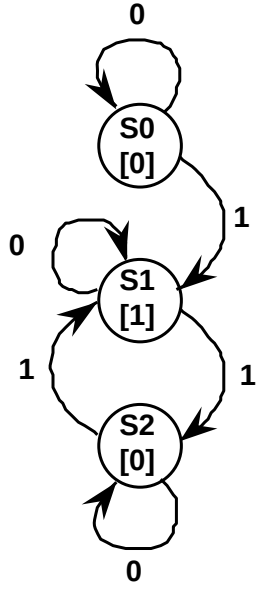


Row-Matching method (cont'd)

- Row-matching methods
  - Straightforward to understand and easy to implement
  - Problem: does not yield the most reduced state table
    - Example: 3 State Odd Parity Checker

Present State	Next State X=0	Next State X=1	Output
S_0	S_0	S_1	0
S_1	S_1	S_2	1
S_2	S_2	S_1	0

No way to combine states S0 and S2 based on Next State Criterion!



Implication Chart method

- Example specification
  - Name: three-bit sequence (010 or 110) detector
  - Input: X = {0, 1}
  - Output: Z = {0, 1}
  - Behavior:

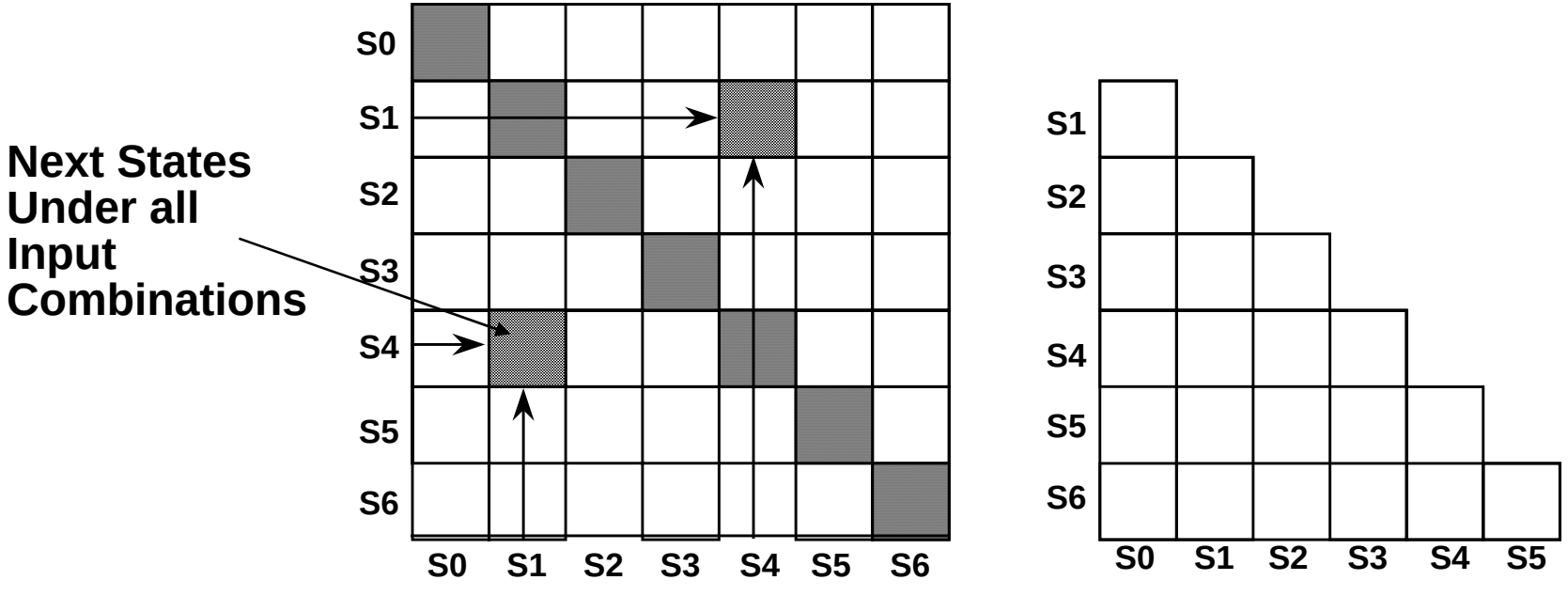
$$Z = \begin{cases} 1 \text{ if each 3-bit input sequence (no overlap) is 010 or 110} \\ 0 \text{ otherwise} \end{cases}$$

- Initial state transition table

Past Input Seq.	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_0	S_0	0	0
01	S_4	S_0	S_0	1	0
10	S_5	S_0	S_0	0	0
11	S_6	S_0	S_0	1	0

Implication Chart method (cont'd)

- Implication Chart
  - Enumerate all possible combinations of states taken two at a time

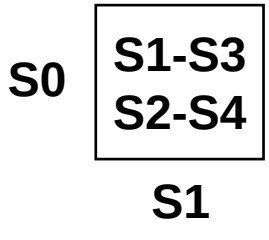


Naive Data Structure:
Xij will be the same as Xji
Also, can eliminate the diagonal

Implication Chart

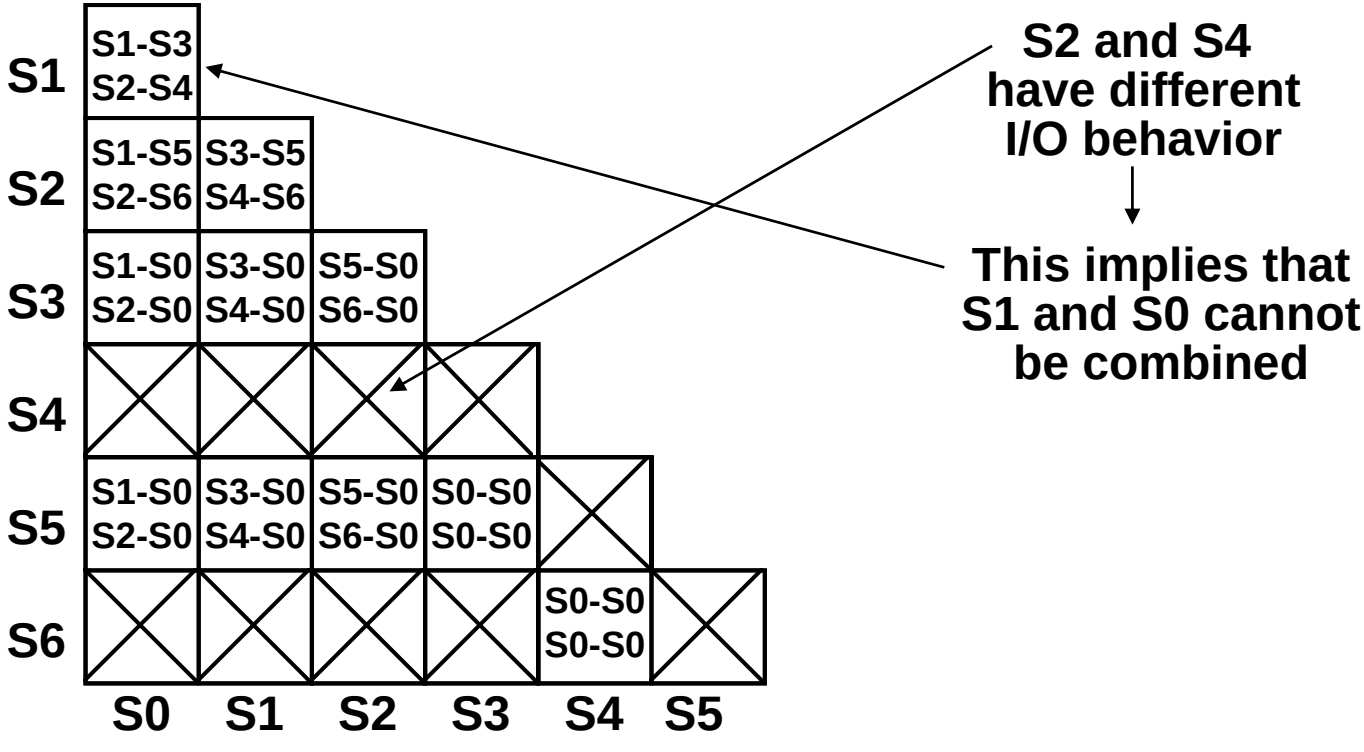
Implication Chart method (cont'd)

- Filling in the Implication Chart
  - Entry Xij: Row is Si, Column is Sj
  - Si is equivalent to Sj if outputs are the same and next states are equivalent
  - Xij contains the next states of Si, Sj which must be equivalent if Si and Sj are equivalent
  - If Si, Sj have different output behavior, then Xij is crossed out
- Example:
  - S0 transitions to S1 on 0, S2 on 1;
  - S1 transitions to S3 on 0, S4 on 1;
  - So square X<0,1> contains entries S1-S3 (transition on zero), S2-S4 (transition on one)



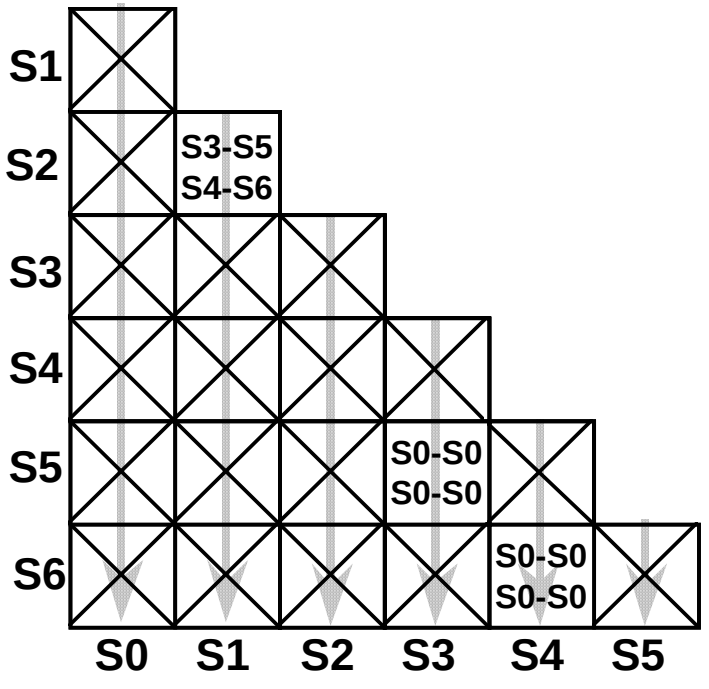
Implication Chart method (cont'd)

- Starting Implication Chart



Implication Chart method (cont'd)

- Results of First Marking Pass
- Second Pass Adds
  No New Information
  - S3 and S5 are equivalent
  - S4 and S6 are equivalent
  - This implies that S1 and S2 are too!

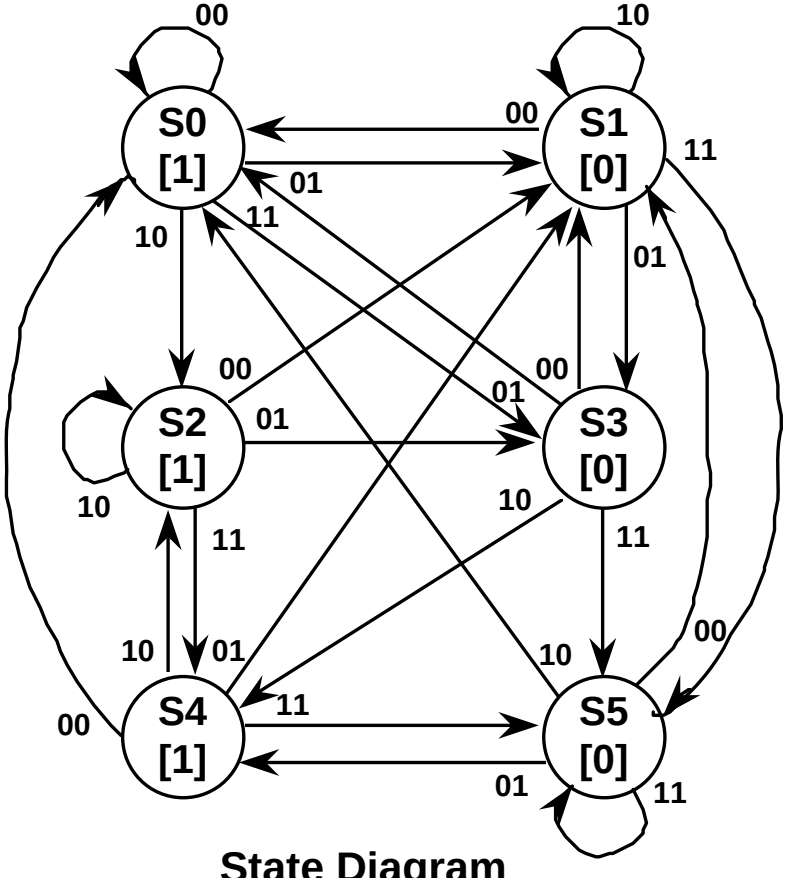


- Reduced State Transition Table

Input Sequence	Present State	Next State X =0	Next State X =1	Output X =0	Output X =1
Reset	S_0	S'_1	S'_1	0	0
0 or 1	S'_1	S'_3	S'_4	0	0
00 or 10	S'_3	S_0	S_0	0	0
01 or 11	S'_4	S_0	S_0	1	0

Implication Chart method (cont'd)

- Multiple Input State Diagram Example



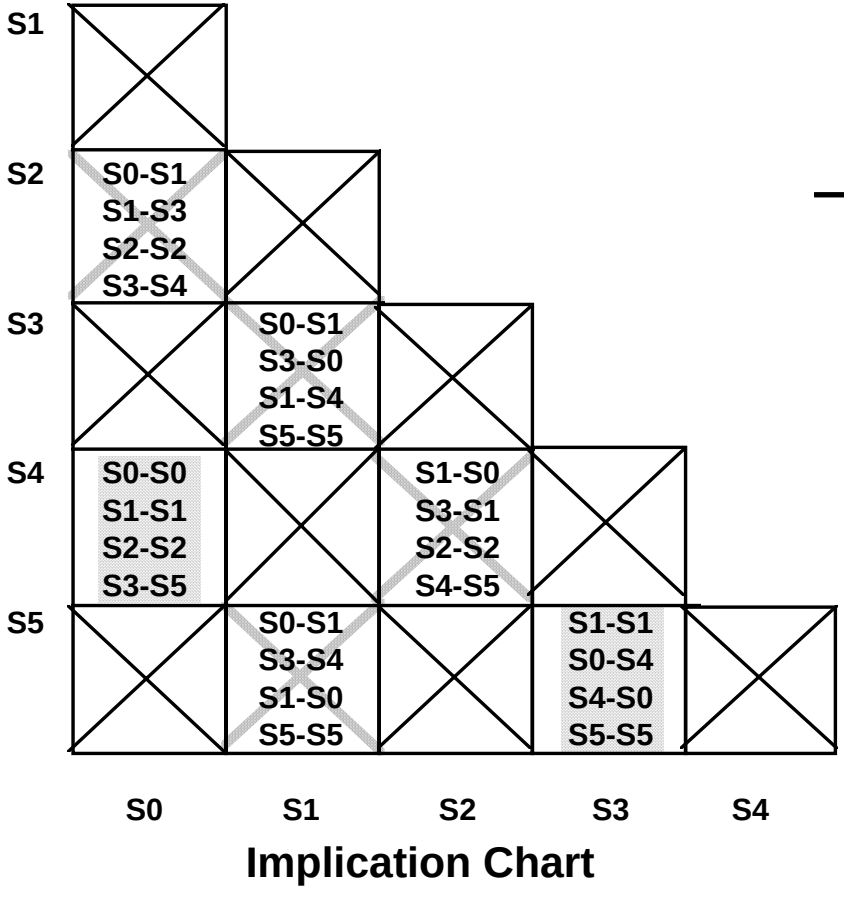
State Diagram

Present State	Next State 00	01	10	11	Output
S_0	S_0	S_1	S_2	S_3	1
S_1	S_0	S_3	S_1	S_5	0
S_2	S_1	S_3	S_2	S_4	1
S_3	S_1	S_0	S_4	S_5	0
S_4	S_0	S_1	S_2	S_5	1
S_5	S_1	S_4	S_0	S_5	0

Symbolic State Diagram

Implication Chart method (cont'd)

- Multiple Input Example



Implication Chart

Present State	Next State 00	01	10	11	Output
S_0'	S_0'	S_1	S_2	S_3'	1
S_1	S_0'	S_3'	S_1	S_3'	0
S_2	S_1	S_3'	S_2	S_0'	1
S_3'	S_1	S_0'	S_0'	S_3'	0

Minimized State Table

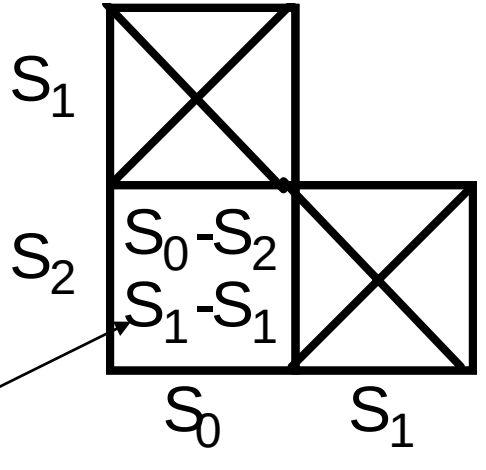
Implication Chart method (cont'd)

- The detailed algorithm:
  1. Construct implication chart, one square for each combination of states taken two at a time.
  2. For each square labeled Si, Sj,
     - if outputs differ, then cross out the square.
     - otherwise, write down next state pairs for all input combinations.
  3. Advancing through the chart top-to-bottom and left-to-right,
     - if square Si, Sj contains next state pair Sm-Sn and square Sm, Sn is already crossed out, then cross out squre Si, Sj.
  4. Continue executing Step 3 until no new squares are crossed out.
  5. For each remaining square Si, Sj, we conclude that Si and Sj are equivalent.

Implication Chart method (cont'd)

- Does the method solve the problem with the odd parity checker?

Present State	Next State X=0	Next State X=1	Output
S_0	S_0	S_1	0
S_1	S_1	S_2	1
S_2	S_2	S_1	0



S0 is equivalent to S2
since nothing contradicts this assertion!

Equivalent states in the presence of don't cares

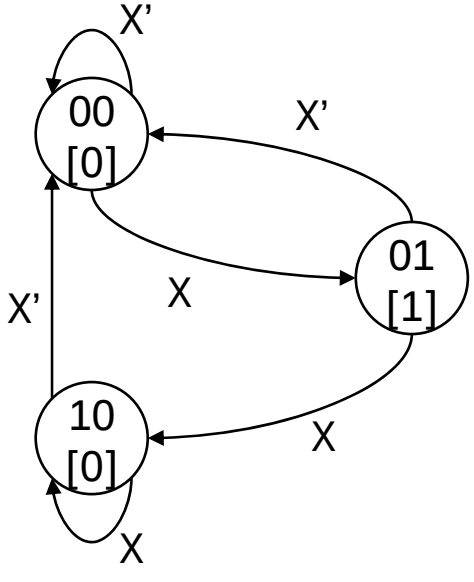
- Equivalence of states is transitive when machine is fully specified
- But its not transitive when don't cares are present
  - Example

e.g.,	state	output	
	S0	– 0	S1 is compatible with both S0 and S2
	S1	1 –	but S0 and S2 are incompatible
	S2	– 1	

- No polynomial time algorithm exists for determining best grouping of states into equivalent sets that will yield the smallest number of final states

When state minimization doesn't help

- Example: edge detector
  - outputs 1 when last two input changes from 0 to 1
- Implementation using minimized states



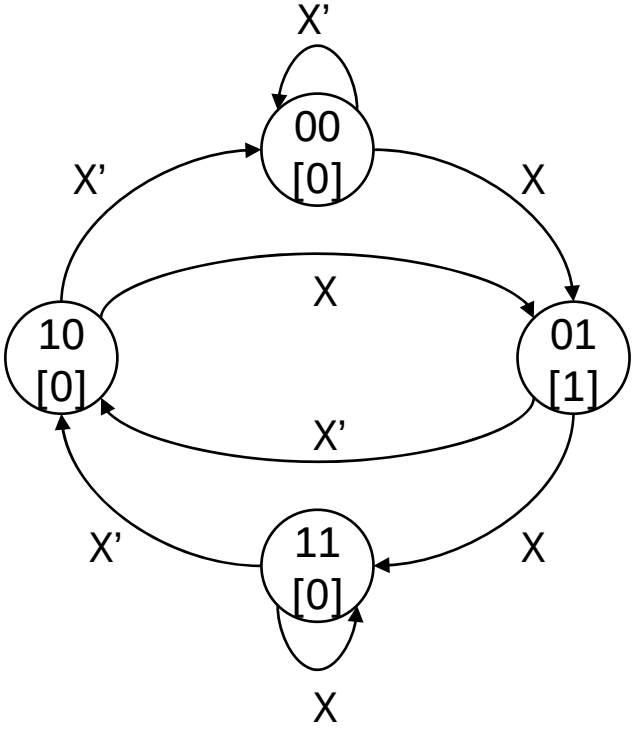
X	Q_1	Q_0	Q_1^+	Q_0^+
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
–	1	1	-	-

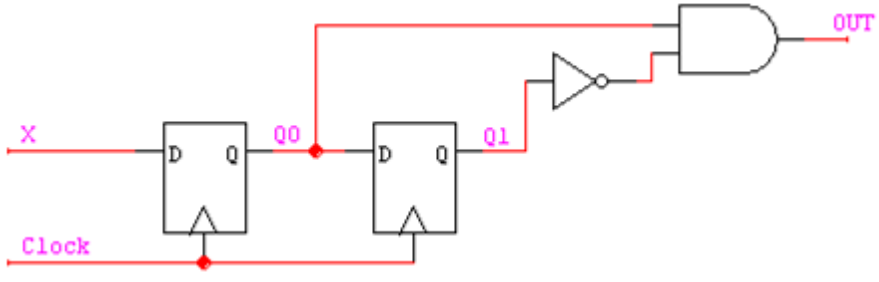
$Q_1^+ = X\ (Q_1 + Q_0)$

$Q_0^+ = X\ Q_1' Q_0'$

When state minimization doesn't help (cont'd)

- Another implementation of edge detector
  - "Ad hoc" solution - not minimal but cheap and fast





State assignment

- State assignment (encoding): choose bit vectors to assign to each “symbolic” state
  - with n state bits for m states ($n <= m <= 2^n$),
    - there are $2^n! / (2^n - m)!$ possible state assignments
    - huge number even for small values of n and m
      - intractable for state machines of any practical size
      - heuristics are necessary for practical solutions
  - state encoding with fewer bits has fewer equations to implement
    - however, each may be more complex
  - state encoding with more bits (e.g., one-hot) has simpler equations
    - complexity directly related to complexity of state diagram

State assignment (cont'd)

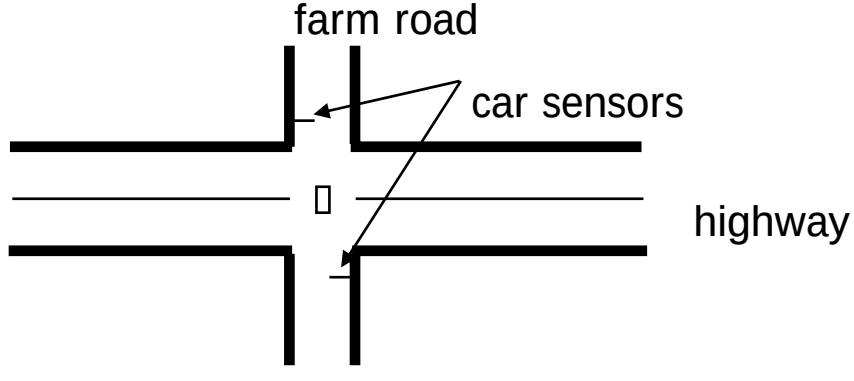
- Optimize some metric for the combinational logic
  - size (the amount of logic and number of FFs)
  - speed (depth of logic and fanout)
  - dependencies (decomposition)
- Possible strategies
  - sequential – just number states as they appear in the state table
  - random – pick random codes
  - one-hot – use as many state bits as there are states (bit=1 –> state)
  - output-oriented – use outputs to help encode states
  - heuristic – rules of thumb that seem to work in most cases
- No guarantee of optimality – another intractable problem

State assignment (cont'd)

- Example: traffic light controller
  - 4 states: 4 choices for first state, 3 for second, 2 for third, 1 for last
    -> 24 different encodings (4!)
  - Alternative state encodings of the traffic light controller

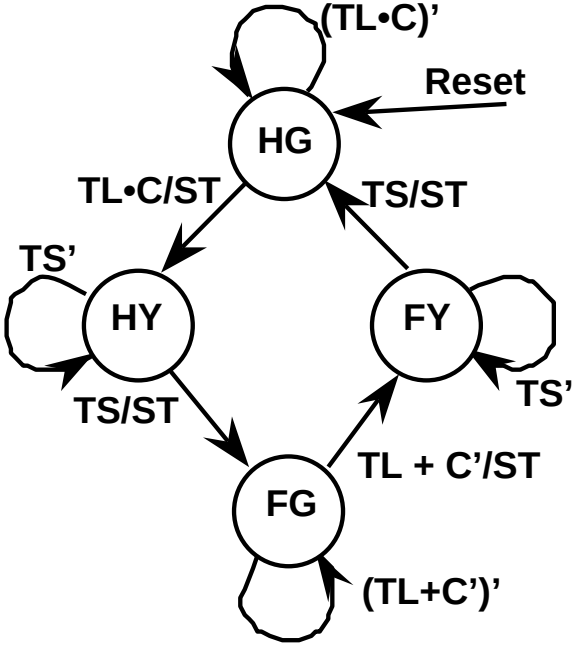
HG	HY	FG	FY
00	01	10	11
00	01	11	10
00	10	01	11
00	10	11	01
00	11	01	10
00	11	10	01
01	00	10	11
01	00	11	10
01	10	00	11
01	10	11	00
01	11	00	10
01	11	10	00

HG	HY	FG	FY
10	00	01	11
10	00	11	01
10	01	00	11
10	01	11	00
10	11	00	01
10	11	01	00
11	00	01	10
11	00	10	01
11	01	00	10
11	01	10	00
11	10	00	01
11	10	01	00



State assignment (cont'd)

- Example: traffic light controller (cont'd)
  - State diagram and Symbolic state transition table



Inputs			Present State	Next State	Outputs		
C	TL	TS	Q_1 Q_0	P_1 P_0	ST	H_1 H_0	F_1 F_0
0	X	X	HG	HG	0	00	10
X	0	X	HG	HG	0	00	10
1	1	X	HG	HY	1	00	10
X	X	0	HY	HY	0	01	10
X	X	1	HY	FG	1	01	10
1	0	X	FG	FG	0	10	00
0	X	X	FG	FY	1	10	00
X	1	X	FG	FY	1	10	00
X	X	0	FY	FY	0	10	01
X	X	1	FY	HG	1	10	01

Sequential encoding

- Sequential encoding
  - Simply replace the symbolic state names with a regular encoding sequence
- Examples:
  - Sequential encoding: HG=00, HY=01, FG=10, FY=11
  - Encoding with Gray-code: HG=00, HY=01, FG=11, FY=10

Inputs			Present State	Next State	Outputs		
C	TL	TS	Q_1 Q_0	P_1 P_0	ST	H_1 H_0	F_1 F_0
0	X	X	00	00	0	00	10
X	0	X	00	00	0	00	10
1	1	X	00	01	1	00	10
X	X	0	01	01	0	01	10
X	X	1	01	11	1	01	10
1	0	X	11	11	0	10	00
0	X	X	11	10	1	10	00
X	1	X	11	10	1	10	00
X	X	0	10	10	0	10	01
X	X	1	10	00	1	10	01

Sequential encoding (cont'd)

- Example (cont'd)
  - Two level equation

$$P_1 = TS \cdot Q_1^{'} \cdot Q_0 + \mathrm{C} \cdot TL^{'} \cdot Q_1 \cdot Q_0 + C^{'} \cdot Q_1 \cdot Q_0 + TL \cdot Q_1 \cdot \mathrm{Q}_0 + TS^{'} \cdot Q_1 \cdot Q_0^{'}$$

$$P_0 = C \cdot TL \cdot Q_1^{'} \cdot Q_0^{'} + TS^{'} \cdot Q_1^{'} \cdot Q_0 + TS \cdot Q_1^{'} \cdot Q_0 + C \cdot TL \cdot^{'} Q_1 \cdot Q_0$$

$$ST = C \cdot TL \cdot Q_1^{'} \cdot Q_0^{'} + TS \cdot Q_1^{'} \cdot Q_0 + C^{'} \cdot Q_1 \cdot Q_0 + TL \cdot Q_1 \cdot Q_0 + TS \cdot Q_1 \cdot Q_0^{'}$$

$$H_1 = C \cdot TL^{'} \cdot Q_1 \cdot Q_0 + C^{'} \cdot Q_1 \cdot Q_0 + TL \cdot Q_1 \cdot Q_0 + TS^{'} \cdot Q_1 \cdot Q_0^{'} + TS \cdot Q \cdot_1 Q_0^{'}$$

$$H_0 = TS^{'} \cdot Q_1^{'} \cdot Q_0 + TS \cdot Q_1^{'} \cdot Q_0$$

$$F_1 = C^{'} \cdot Q_1^{'} \cdot Q_0^{'} + TL^{'} \cdot Q_1^{'} \cdot Q_0^{'} + C \cdot TL \cdot Q_1^{'} \cdot Q_0^{'} + TS^{'} \cdot Q_1^{'} \cdot Q_0 + TS \cdot Q_1^{'} \cdot Q_0$$

$$F_0 = TS^{'} \cdot Q_1 \cdot Q_0^{'} + TS \cdot Q_1 \cdot Q_0^{'}$$

Sequential encoding (cont'd)

- Examples (cont'd):
  - Three-level implementation
    - More than two input gates: P1, P0, ST
    - Two input gates: the others

$$P_1 = TS \cdot HY + FG + TS' \cdot FY$$

$$P_0 = X \cdot HG + HY + Y \cdot FG$$

$$ST = X \cdot HG + TS \cdot HY + Y' \cdot FG + TS \cdot FY$$

$$H_1 = FG + FY$$

$$H_0 = HY$$

$$F_1 = HG + HY$$

$$F_0 = FY$$

$$HG = Q_1' \cdot Q_0'$$

$$HY = Q_1' \cdot Q_0$$

$$FG = Q_1 \cdot Q_0$$

$$FY = Q_1 \cdot Q_0'$$

$$X = C \cdot TL$$

$$Y = C \cdot TL'$$

Random encoding

- Random encoding
  - replace the symbolic state names with a random encoding sequence
- Example: HG=00, HY=10, FG=01, FY=11
  - Two level implementation
    - No gates of more than three inputs

Inputs			Present State	Next State	Outputs			
C	TL	TS	Q_1 Q_0	P_1 P_0	ST	H_1 H_0	F_1 F_0	
0	X	X	00	00	0	00	10	
X	0	X	00	00	0	00	10	
1	1	X	00	10	1	00	10	
X	X	0	10	10	0	01	10	
X	X	1	10	01	1	01	10	
1	0	X	01	01	0	10	00	
0	X	X	01	11	1	10	00	
X	1	X	01	11	1	10	00	
X	X	0	11	11	0	10	01	
X	X	1	11	00	1	10	01	

$$P_1 = C \cdot TL \cdot Q_1^{'} + TS' \cdot Q_1 + C' \cdot Q_1^{'} \cdot Q_0$$

$$P_0 = TS \cdot Q_1 \cdot Q_0^{'} + Q_1^{'} \cdot Q_0 + TS' \cdot Q_1 \cdot Q_0$$

$$ST = C \cdot TL \cdot Q_1^{'} + C' \cdot Q_1^{'} \cdot Q_0 + TS \cdot Q_1$$

$$H_1 = Q_0$$

$$H_0 = Q_1 \cdot Q_0^{'}$$

$$F_1 = Q_0^{'}$$

$$F_0 = Q_1 \cdot Q_0$$

One-Hot encoding

- One-Hot encoding
  - *n* states is encoded using *n* flip-flops
  - Only 1 bit is asserted in each of the states.
    - ex) 0001, 0010, 0100, 1000
- Properties
  - Simple: easy to encode, easy to debug
  - Small logic functions
    - each state function requires only predecessor state bits as input
    - a lot of don't-care opportunities
  - Good for programmable devices
    - lots of flip-flops readily available
    - simple functions with small support (signals it's dependent upon)
  - Impractical for large machines
    - too many states require too many flip-flops
    - decompose FSMs into smaller pieces that can be one-hot encoded
- Many slight variations to one-hot
  - one-hot + all-0

One-Hot encoding (cont'd)

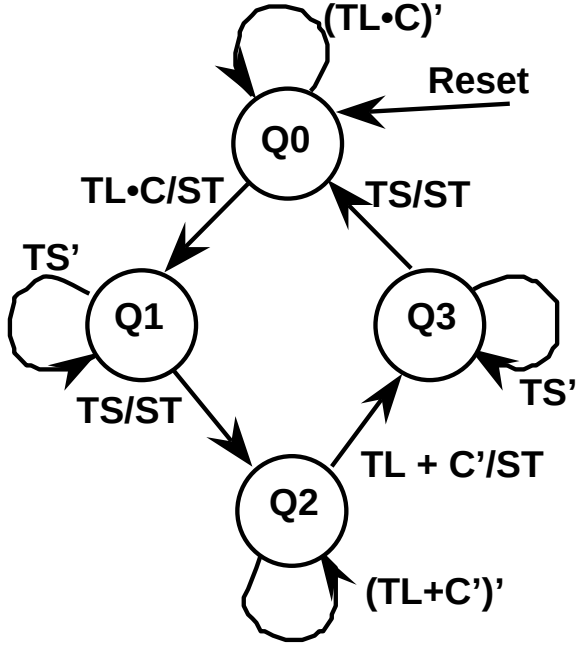
- Example:
  - HG=0001, HY=0010, FG=0100, FY=1000

Inputs			Present State	Next State	Outputs		
C	TL	TS	$Q_3\ Q_2\ Q_1\ Q_0$	$P_3\ P_2\ P_1\ P_0$	ST	$H_1\ H_0$	$F_1\ F_0$
0	X	X	0001	0001	0	00	10
X	0	X	0001	0001	0	00	10
1	1	X	0001	0010	1	00	10
X	X	0	0010	0010	0	01	10
X	X	1	0010	0100	1	01	10
1	0	X	0100	0100	0	10	00
0	X	X	0100	1000	1	10	00
X	1	X	0100	1000	1	10	00
X	X	0	1000	1000	0	10	01
X	X	1	1000	0001	1	10	01

One-Hot encoding (cont'd)

- Example (cont'd)
  - Implementation:

$$P_3 = (C'+TL)\cdot Q_2 + (TS')\cdot Q_3$$
$$P_2 = (TS)\cdot Q_1 + (C\cdot TL')\cdot Q_2$$
$$P_1 = (C\cdot TL)\cdot Q_0 + (TS')\cdot Q_1$$
$$P_0 = (C'+TL')\cdot Q_0 + (TS)\cdot Q_3$$
$$ST = (C\cdot TL)\cdot Q_0 + (TS)\cdot Q_1 + (C'+TL)\cdot Q_2 + (TS)\cdot Q_3$$
$$H_1 = Q_3 + Q_2$$
$$H_0 = Q_1$$
$$F_1 = Q_1 + Q_0$$
$$F_0 = Q_3$$



Output-Oriented encoding

- Output-oriented encoding:
  - Reuse outputs as state bits - use outputs to help distinguish states
    - why create new functions for state bits when output can serve as well
    - Synchronous Mealy outputs, since they are implemented directly as the output of a flip-flop, can also be used this way
- Example: the traffic-light controller

Inputs			Present State	Next State	Outputs		
C	TL	TS	Q_1 Q_0	P_1 P_0	ST	H_1 H_0	F_1 F_0
0	X	X	HG	HG	0	00	10
X	0	X	HG	HG	0	00	10
1	1	X	HG	HY	1	00	10
X	X	0	HY	HY	0	01	10
X	X	1	HY	FG	1	01	10
1	0	X	FG	FG	0	10	00
0	X	X	FG	FY	1	10	00
X	1	X	FG	FY	1	10	00
X	X	0	FY	FY	0	10	01
X	X	1	FY	HG	1	10	01

Output signals are unique for the transitions to each state

Output-Oriented encoding (cont'd)

- Example (cont'd)
  - Next state is represented by "present outputs" instead

State equations

$$HG = ST \cdot H_1 H_0^{'} F_1^{'} F_0 + ST' \cdot H_1^{'} H_0^{'} F_1 F_0^{'}$$

$$HY = ST \cdot H_1^{'} H_0^{'} F_1 F_0^{'} + ST' \cdot H_1^{'} H_0 F_1 F_0^{'}$$

$$FG = ST \cdot H_1^{'} H_0 F_1 F_0^{'} + ST' \cdot H_1 H_0^{'} F_1^{'} F_0^{'}$$

$$FY = ST \cdot H_1 H_0^{'} F_1^{'} F_0^{'} + ST' \cdot H_1 H_0^{'} F_1^{'} F_0$$

Output equations

$$ST = (C \cdot TL \cdot HG) + (TS \cdot HY) + (C' \cdot FG) + (TL \cdot FG) + (TS \cdot FY)$$

$$H_1 = FG + FY, \quad H_0 = HY$$

$$F_1 = HG + HY, \quad F_0 = FY$$

Inputs			Present State	Outputs			Next State
C	TL	TS	ST H_1 H_0 F_1 F_0	ST	H_1 H_0	F_1 F_0	ST H_1 H_0 F_1 F_0
0	X	X	HG: 00010 + 11001	0	00	10	HG: 00010 +11001
X	0	X	HG: 00010 + 11001	0	00	10	HG: 00010 +11001
1	1	X	HG: 00010 + 11001	1	00	10	HY: 10010 + 00110
X	X	0	HY: 10010 + 00110	0	01	10	HY: 10010 + 00110
X	X	1	HY: 10010 + 00110	1	01	10	FG: 10110 + 01000
1	0	X	FG: 10110 + 01000	0	10	00	FG: 10110 + 01000
0	X	X	FG: 10110 + 01000	1	10	00	FY: 11000 + 01001
X	1	X	FG: 10110 + 01000	1	10	00	FY: 11000 + 01001
X	X	0	FY: 11000 + 01001	0	10	01	FY: 11000 + 01001
X	X	1	FY: 11000 + 01001	1	10	01	HG: 00010 +11001

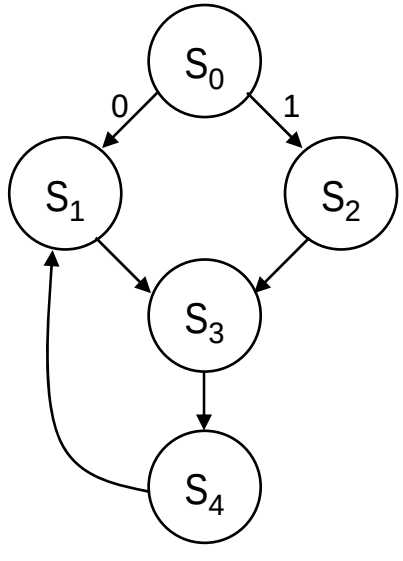
Heuristic methods for state assignment

- Heuristic methods
  - To make the state encoding problem more tractable
  - Try to reduce the distance in Boolean n-space between related states
  - All current methods are variants of this
    - 1) determine which states “attract” each other (weighted pairs)
    - 2) generate constraints on states (which should be in same cube)
    - 3) place states on Boolean cube so as to maximize constraints satisfied (weighted sum)
  - Can't consider all possible embeddings of state clusters in Boolean cube
    - heuristics for ordering embedding
    - to prune search for best embedding
    - expand cube (more state bits) to satisfy more constraints

Heuristic methods for state assignment (cont'd)

- State maps:
  - similar in concept to K-maps
  - If state X transitions to state Y, then assign "close" assignments to X and Y
  - provide a means of observing adjacencies in state assignments
- Example

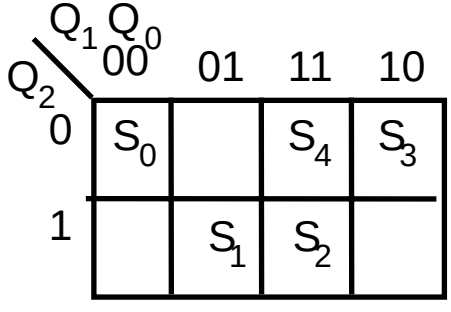
Present State	Next State 0	Next State 1
S_0	S_1	S_2
S_1	S_3	S_3
S_2	S_3	S_3
S_3	S_4	S_4
S_4	S_0	S_0

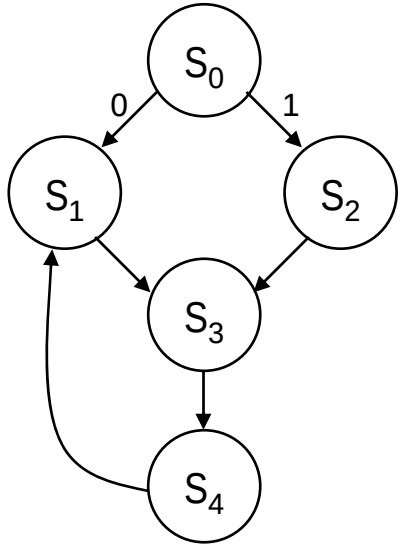


Heuristic methods for state assignment (cont'd)

- Example (cont'd)
  - First state assignment and its state map

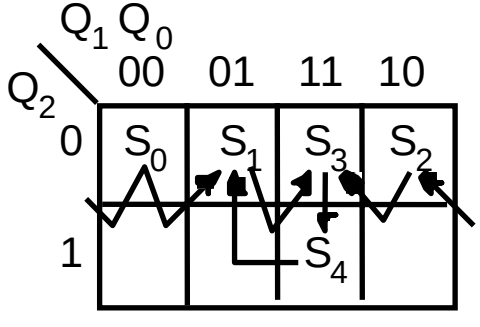
State Name	Assignment Q_2	Q_1	Q_0
S_0	0	0	0
S_1	1	0	1
S_2	1	1	1
S_3	0	1	0
S_4	0	1	1





  - Second state assignment and its state map

State Name	Assignment Q_2	Q_1	Q_0
S_0	0	0	0
S_1	0	0	1
S_2	0	1	0
S_3	0	1	1
S_4	1	1	1



Heuristic methods for state assignment (cont'd)

- Minimum Bit-Change Heuristic
  - Assigns states so that #(bit changes) for all transitions is minimized
- Example

Transition	First assignment bit changes	Second assignment bit changes
S0 to S1	2	1
S0 to S2	3	1
S1 to S3	3	1
S1 to S3	2	1
S3 to S4	1	1
S4 to S1	+) 2	+) 2
	13	7

  - cf. Traffic light controller: HG = 00, HY = 01, FG = 11, FY = 10
    - **yields minimum distance encoding but not best assignment!**

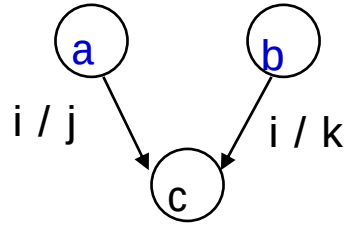
Heuristic methods for state assignment (cont'd)

- Guidelines based on Next state and I/O
- Adjacent codes to states that share a common next state
  - group 1's in next state map

Highest Priority

I	Q	Q^+	O
i	a	**c**	j
i	b	**c**	k

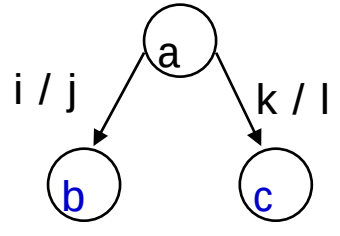
c = i * a + i * b



- Adjacent codes to states that share a common ancestor state
  - group 1's in next state map

I	Q	Q^+	O
i	**a**	b	j
k	**a**	c	l

b = i * a
c = k * a

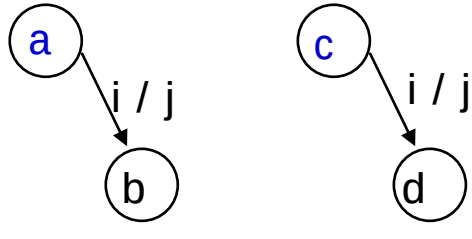


- Adjacent codes to states that have a common output behavior
  - group 1's in output map

Lowest Priority

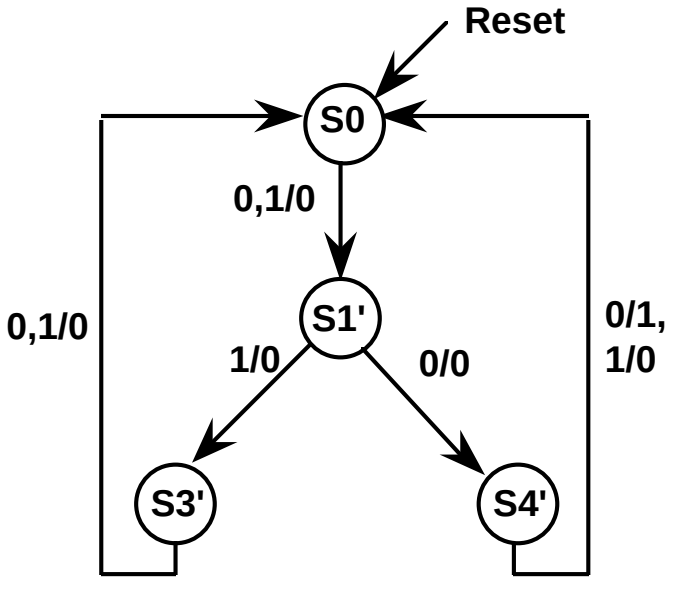
I	Q	Q^+	O
i	a	b	**j**
i	c	d	**j**

j = i * a + i * c
b = i * a
d = i * c



Heuristic methods for state assignment (cont'd)

- Example: 3-bit Sequence Detector



Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
S0	S1'	S1'	0	0
S1'	S3'	S4'	0	0
S3'	S0	S0	0	0
S4'	S0	S0	1	0

Highest Priority: (S3', S4')

Medium Priority: (S3', S4')

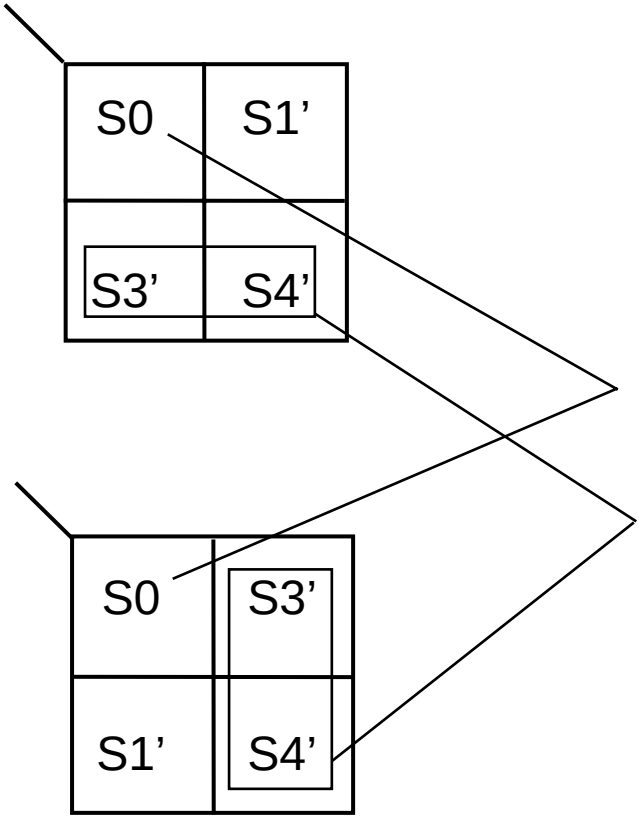
Lowest Priority:

0/0: (S0, S1', S3')

1/0: (S0, S1', S3', S4')

Heuristic methods for state assignment (cont'd)

- Example (cont'd)



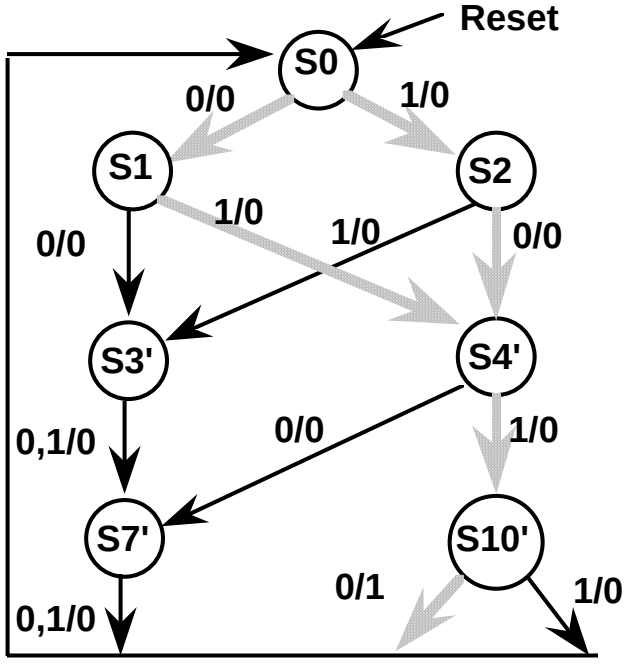
Reset State = 00

Highest Priority Adjacency (S3', S4')

Not much difference in these two assignments

Heuristic methods for state assignment (cont'd)

- Another Example: 4 bit String Recognizer



	Next State		Output	
Present State	X=0	X=1	X=0	X=1
S0	S1	S2	0	0
S1	S3'	S4'	0	0
S2	S4'	S3'	0	0
S3'	S7'	S7'	0	0
S4'	S7'	S10'	0	0
S7'	S0	S0	0	0
S10'	S0	S0	1	0

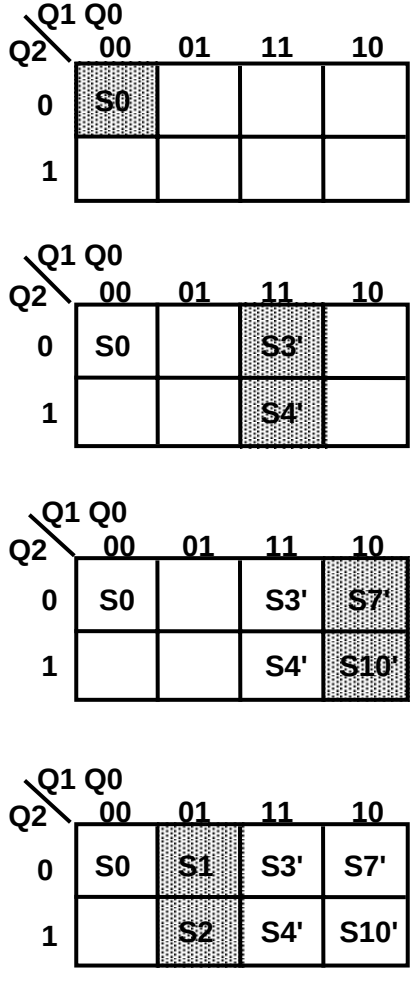
Highest Priority: (S3', S4'), (S7', S10')

Medium Priority:
(S1, S2), 2x(S3', S4'), (S7', S10')

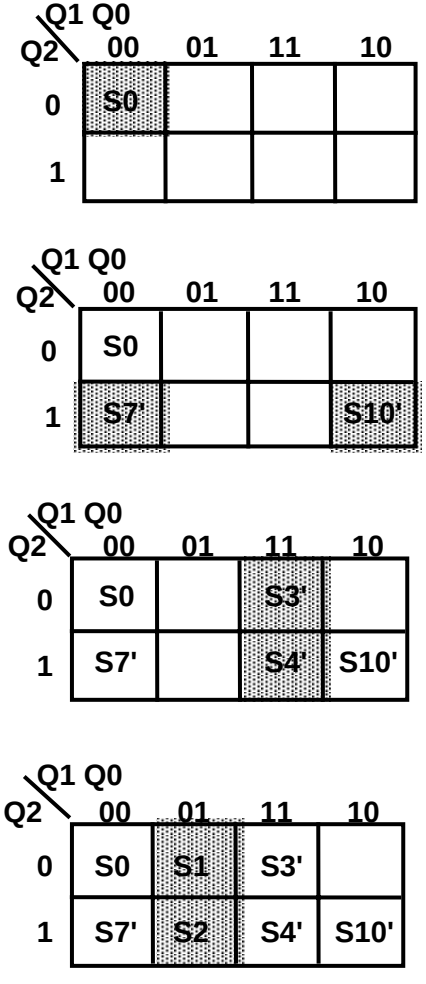
Lowest Priority:
0/0: (S0, S1, S2, S3', S4', S7')
1/0: (S0, S1, S2, S3', S4', S7')

Heuristic methods for state assignment (cont'd)

State Map



(a) First encoding



(b) Second encoding

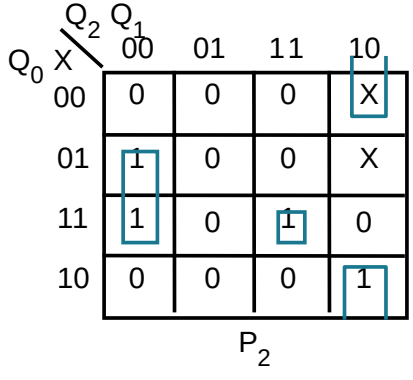
00 = Reset = S0

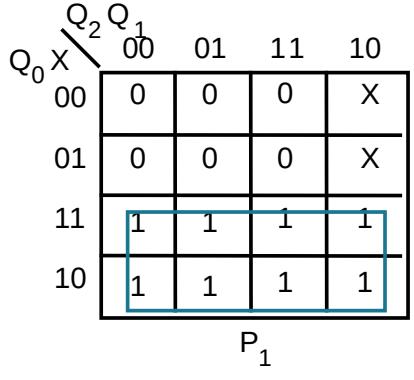
(S1, S2), (S3', S4'), (S7', S10') placed adjacently

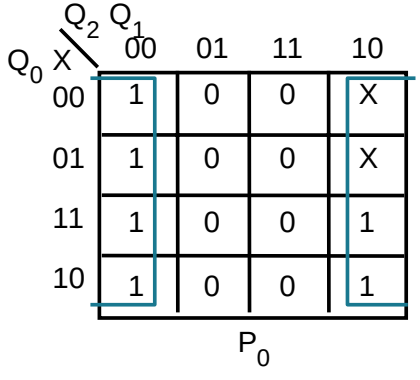
Heuristic methods for state assignment (cont'd)

- Effect of Adjacencies on Next State Map

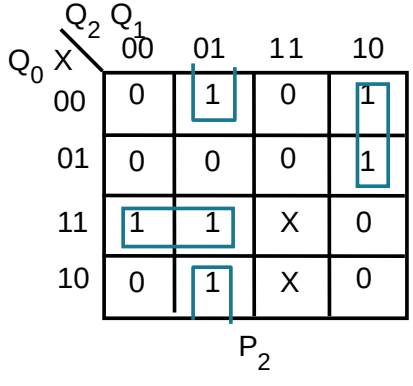
Current State	Next State X = 0	Next State X = 1
(S_0) 000	001	101
(S_1) 001	011	111
(S_2) 101	111	011
(S'_3) 011	010	010
(S'_4) 111	010	110
(S'_7) 010	000	000
(S'_{10}) 110	000	000

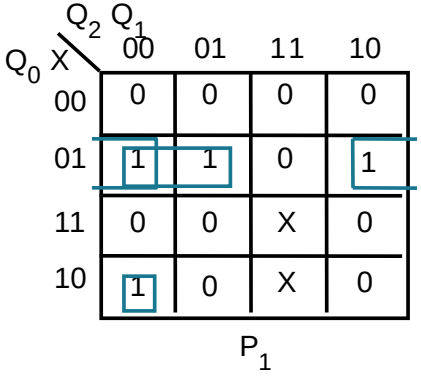


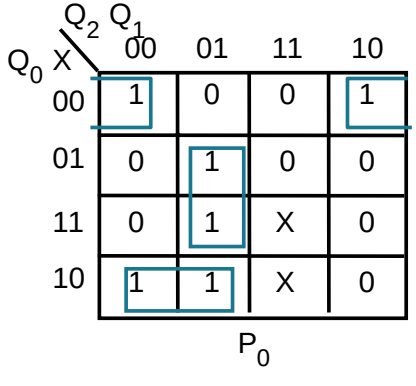




Current State	Next State X = 0	Next State X = 1
(S_0) 000	001	010
(S_1) 001	011	100
(S_2) 010	100	011
(S'_3) 011	101	101
(S'_4) 100	101	110
(S'_7) 101	000	000
(S'_{10}) 110	000	000







First encoding exhibits a better clustering of 1's in the next state map

Summary

- State minimization / reduction
  - introduction to the row-matching and implication chart methods
    - Identify and eliminate redundant states
    - Reduce the number of flip-flops needed to implement a particular FSM
  - straightforward in fully-specified machines
  - computationally intractable, in general (with don't cares)
- State assignment (encoding)
  - Various approaches to state assignment