

Finite state machine optimization

- State minimization
 - fewer states require fewer state bits
 - fewer bits require fewer logic equations
- Encodings: state, inputs, outputs
 - 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
 - input/output encoding may or may not be under designer control

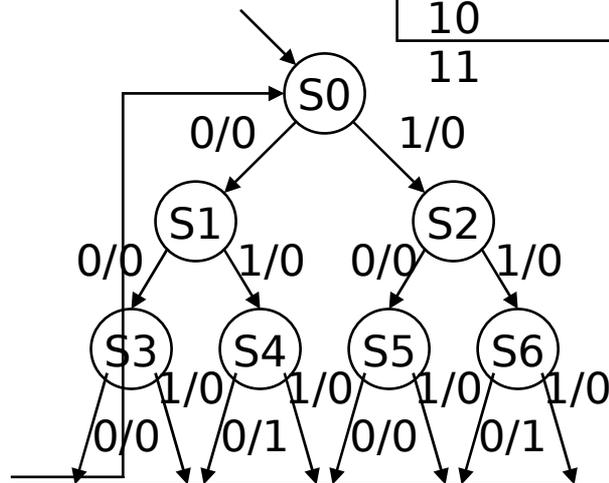
Algorithmic approach to state minimization

- Goal – identify and combine states that have equivalent behavior
- Equivalent states:
 - same output
 - for all input combinations, states transition to same or equivalent states
- Algorithm sketch
 - 1. place all states in one set
 - 2. initially partition set based on output behavior
 - 3. successively partition resulting subsets based on next state transitions
 - 4. repeat (3) until no further partitioning is required
 - states left in the same set are equivalent
 - polynomial time procedure

State minimization example

- Sequence detector for 010 or 110

Input Sequence	Present State	Next State		Output	
		X=0	X=1	X=0	X=1
Reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S0	S0	0	0
01	S4	S0	S0	1	0
10	S5	S0	S0	0	0
11	S6	S0	S0	1	0



Method of successive partitions

Input Sequence	Present State	Next State		Output	
		X=0	X=1	X=0	X=1
Reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S0	S0	0	0
01	S4	S0	S0	1	0
10	S5	S0	S0	0	0
11	S6	S0	S0	1	0

(S0 S1 S2 S3 S4 S5 S6)

S1 is equivalent to S2

(S0 S1 S2 S3 S5) (S4 S6)

S3 is equivalent to S5

(S0 S3 S5) (S1 S2) (S4 S6)

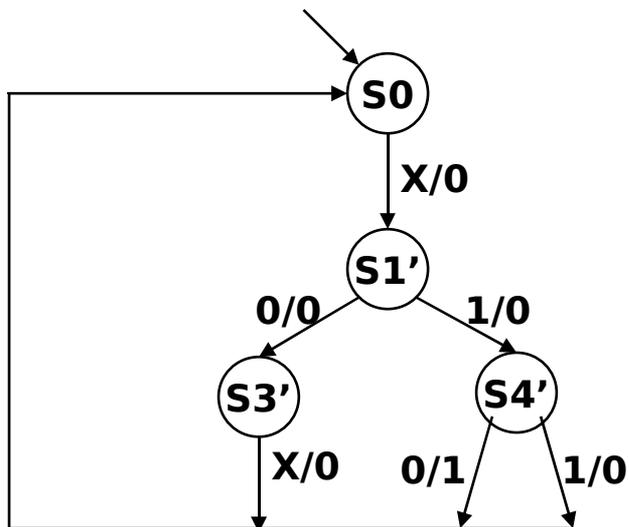
S4 is equivalent to S6

(S0) (S3 S5) (S1 S2) (S4 S6)

Minimized FSM

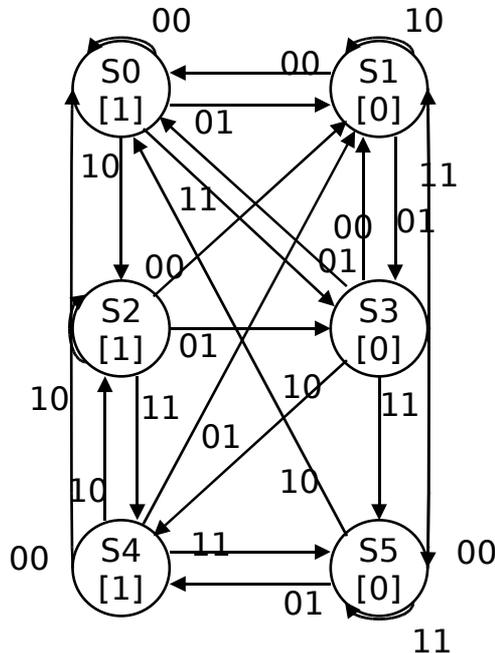
- State minimized sequence detector for 010 or 110

Input Sequence	Present State	Next State		Output	
		X=0	X=1	X=0	X=1
Reset	S0	S1'	S1'	0	0
0 + 1	S1'	S3'	S4'	0	0
X0	S3'	S0	S0	0	0
X1	S4'	S0	S0	1	0



More complex state minimization

- Multiple input example



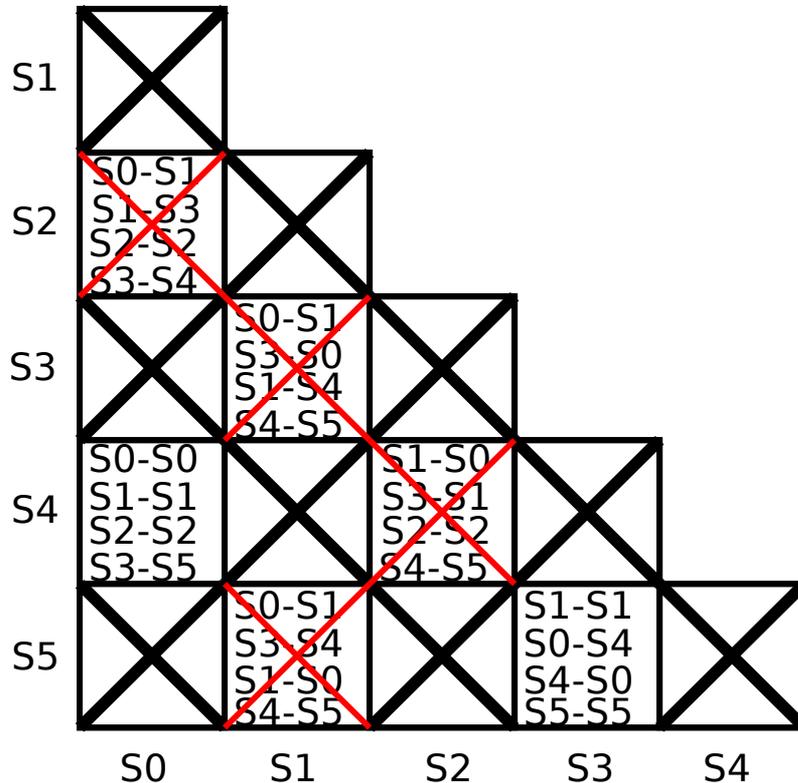
inputs here

present state	00	01	10	11	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S4	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

symbolic state transition table

Minimized FSM

- Implication chart method
 - cross out incompatible states based on outputs
 - then cross out more cells if indexed chart entries are already crossed out



present state	next state				output
	00	01	10	11	
S0'	S0'	S1	S2	S3'	1
S1	S0'	S3'	S1	S3'	0
S2	S1	S3'	S2	S0'	1
S3'	S1	S0'	S0'	S3'	0

minimized state table
(S0==S4) (S3==S5)

Minimizing incompletely specified FSMs

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

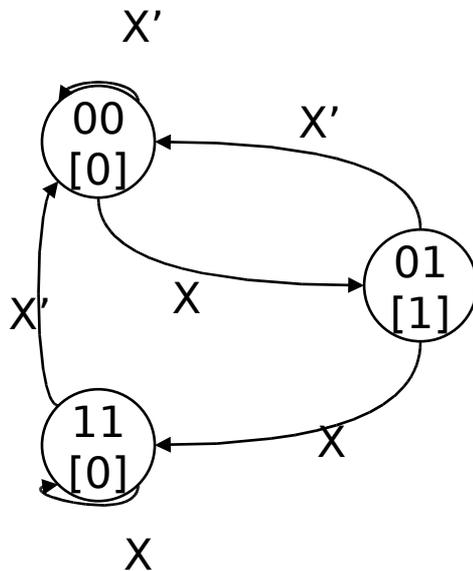
e.g.,

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

Minimizing states may not yield best circuit

- Example: edge detector - outputs 1 when last two input changes from 0 to 1



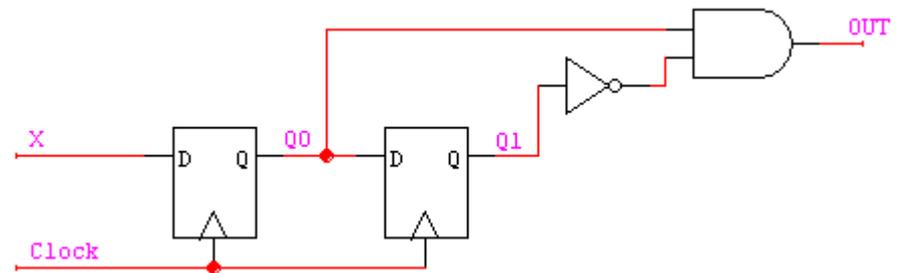
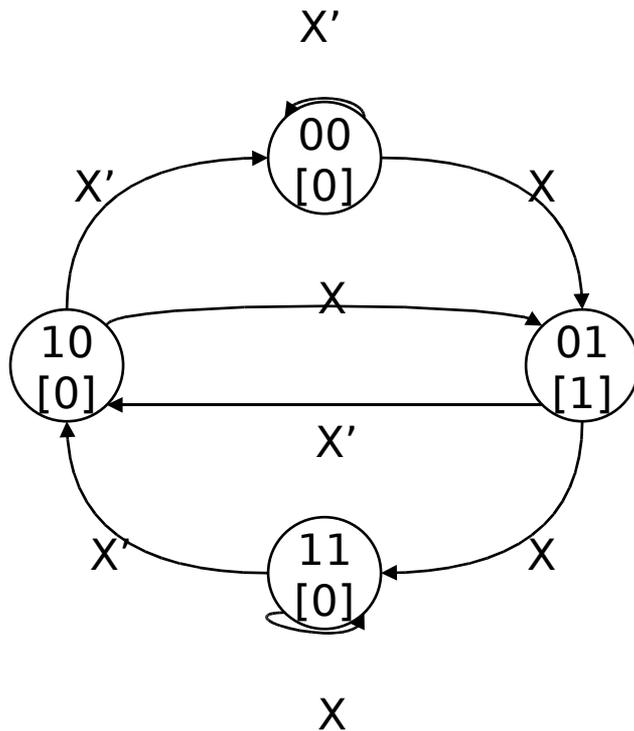
X	Q ₁	Q ₀	Q ₁ ⁺	Q ₀ ⁺
0	0	0	0	0
0	0	1	0	0
0	1	1	0	0
1	0	0	0	1
1	0	1	1	1
1	1	1	1	1
-	1	0	0	0

$$Q_1^+ = X (Q_1 \text{ xor } Q_0)$$

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

Another implementation of edge detector

- "Ad hoc" solution - not minimal but cheap and fast



State assignment

- Choose bit vectors to assign to each “symbolic” state
 - with n state bits for m states there are $2^n! / (2^n - m)!$
 $[\log n \leq m \leq 2^n]$
 - 2^n codes possible for 1st state, $2^n - 1$ for 2nd, $2^n - 2$ for 3rd, ...
 - huge number even for small values of n and m
 - intractable for state machines of any size
 - heuristics are necessary for practical solutions
 - optimize some metric for the combinational logic
 - size (amount of logic and number of FFs)
 - speed (depth of logic and fanout)
 - dependencies (decomposition)

State assignment strategies

- 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 – use outputs to help encode states
 - heuristic – rules of thumb that seem to work in most cases
- No guarantee of optimality – another intractable problem

One-hot state assignment

- Simple
 - easy to encode
 - easy to debug
- Small logic functions
 - each state function requires only predecessor state bits as input
- Good for programmable devices
 - lots of flip-flops readily available
 - simple functions with small support (signals its 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

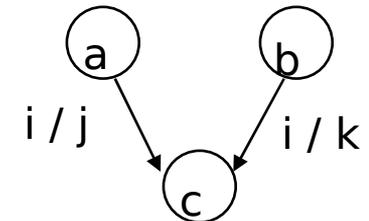
Heuristics for state assignment

- Adjacent codes to states that share a common next state

- group 1's in next state map

I	Q	Q ⁺	O
i	a	c	j
i	b	c	k

$$c = i * a + i * b$$



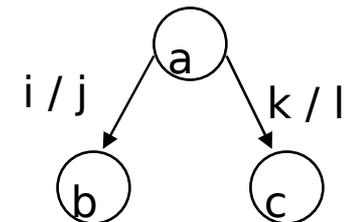
- Adjacent codes to states that share a common ancestor state

- 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

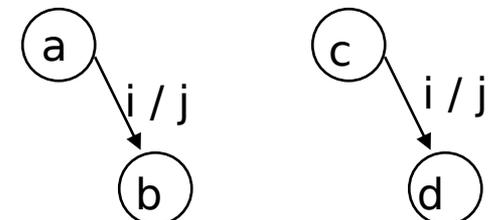
- group 1's in output map

I	Q	Q ⁺	O
i	a	b	j
i	c	d	j

$$j = i * a + i * c$$

$$b = i * a$$

$$d = i * c$$



General approach to heuristic state assignment

- All current methods are variants of this
 - 1) determine which states “attract” each other (weighted pairs)
 - 2) generate constraints on codes (which should be in same cube)
 - 3) place codes on Boolean cube so as to maximize constraints satisfied (weighted sum)
- Different weights make sense depending on whether we are optimizing for two-level or multi-level forms
- 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

Output-based 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
 - fits in nicely with synchronous Mealy implementations

Inputs			Present State	Next State	Outputs		
C	TL	TS			ST	H	F
0	-	-	HG	HG	0	00	10
-	0	-	HG	HG	0	00	10
1	1	-	HG	HY	1	00	10
-	-	0	HY	HY	0	01	10
-	-	1	HY	FG	1	01	10
1	0	-	FG	FG	0	10	00
0	-	-	FG	FY	1	10	00
-	1	-	FG	FY	1	10	00
-	-	0	FY	FY	0	10	01
-	-	1	FY	HG	1	10	01

$$HG = ST' H1' H0' F1 F0' + ST H1 H0' F1' F0$$

$$HY = ST H1' H0' F1 F0' + ST' H1' H0 F1 F0'$$

$$FG = ST H1' H0 F1 F0' + ST' H1 H0' F1' F0'$$

$$HY = ST H1 H0' F1' F0' + ST' H1 H0' F1' F0$$

Output patterns are unique to states, we do not need ANY state bits – implement 5 functions (one for each output) instead of 7 (outputs plus 2 state bits)

Current state assignment approaches

- For tight encodings using close to the minimum number of state bits
 - best of 10 random seems to be adequate (averages as well as heuristics)
 - heuristic approaches are not even close to optimality
 - used in custom chip design
- One-hot encoding
 - easy for small state machines
 - generates small equations with easy to estimate complexity
 - common in FPGAs and other programmable logic
- Output-based encoding
 - ad hoc - no tools
 - most common approach taken by human designers
 - yields very small circuits for most FSMs

Sequential logic optimization summary

- State minimization
 - straightforward in fully-specified machines
 - computationally intractable, in general (with don't cares)
- State assignment
 - many heuristics
 - best-of-10-random just as good or better for most machines
 - output encoding can be attractive (especially for PAL implementations)