14:332:231
DIGITAL LOGIC DESIGN

Ivan Marsic, Rutgers University
Electrical & Computer Engineering
Fall 2013

Lecture #18: State Machine Design and Synthesis

State Machine Design and Synthesis

The creative part ("art"), like writing a program

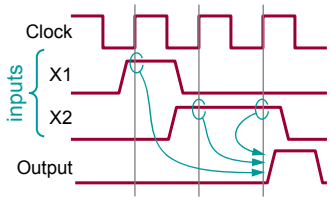
"Turning the crank", like a compiler does

- The flowchart is in *inverse sequence* (compared to analysis). The state/output comes the first and the drawing of the logic diagram comes the last:

state/output table → transition table
→ transition equation → characteristic equation
→ excitation equation → logic diagram

Clocked Synchronous FSM Structure

- Example: Design a combination lock with two inputs, X1 and X2.
  Open for the sequence X1, X2, X2 (one input per clock)
- Success scenario:



- But there are many potential failure scenarios that need to be considered …

Clocked Synchronous FSM Structure

- Example: Design a combination lock with two inputs, X1 and X2.
  Open for the sequence X1, X2, X2 (one input per clock)

inputs

State		***X1 X2***				***Output***
Meaning	Name	00	01	10	11	UNLOCK
Start	A	A	A	B	A	0
Got X1	B	A	C	A	A	0
Got X1, X2	C	A	D	A	A	0
Got X1, X2, X2	D	A	A	B	A	1

next state

- Specification ambiguities are resolved in the state table

State Assignment

- Can minimize the number of states but hardly anyone bothers anymore
- Need to assign binary-variable combinations to states
  - Minimum number of variables for n states is $\lceil \log_2 n \rceil$
  - Using *more than minimum* number may be advantageous in some situations, e.g., one variable per state
    ("one-hot": one-out-of-n pattern)
  - Example: 4 states → 2 state variables (Q1, Q2)

A = 00
B = 01
C = 10
D = 11

Up to this point is the "art" part of FSM design; the rest is just "turning the crank" part

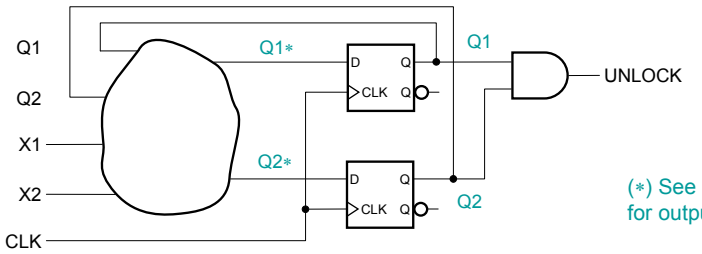
Transition Table

- Substitute state-variable combinations for symbolic state names in the state table

State		***X1 X2***				***Output***
Meaning	Q1 Q2	0 0	0 1	1 0	1 1	UNLOCK
Start	0 0	0 0	0 0	0 1	0 0	0
Got X1	0 1	0 0	1 0	0 0	0 0	0
Got X1, X2	1 0	0 0	1 1	0 0	0 0	0
Got X1, X2, X3	1 1	0 0	0 0	0 1	0 0	1
		Q1* Q2*				

Transition Equations; Circuit

- Transition table specifies each state variable (Q1∗, Q2∗) as a combinational logic function of Q1, Q2, X1, X2
  - Find a realization of each function by your favorite means—ad hoc, minimal sum-of-products, etc.
- Build the circuit

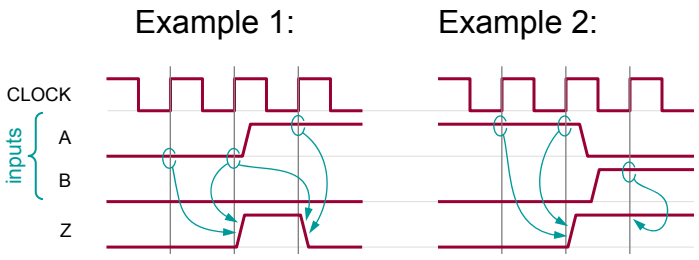


(∗) See state table for output logic

A Complete Design Example

- PROBLEM: Design a machine with inputs A and B and output Z that is "1" if any is true:
  - A had the same value at the *two previous* ticks

OR: – B has been "1" since *the last time* the above was true



A Complete Design Example

- PROBLEM: Design a machine with inputs A and B and output Z that is "1" if any is true:
  - A had the same value at the *two previous* ticks

OR: – B has been "1" since *the last time* the above was true

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	?	?	?	?	0
		S*				

State with symbolic name "A0" means:

Got A=0 on the previous tick, A≠0 on the tick before that, and B≠1 at some time since the previous pair of equal A inputs

A Complete Design Example

- PROBLEM: Design a machine with inputs A and B and output Z that is "1" if any is true:
  - A had the same value at the *two previous* ticks

OR: – B has been "1" since *the last time* the above was true

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	?	?	?	?	0
		S*				

➡

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1					0
Two equal A inputs	OK	?	?	?	?	1
		S*				

State "OK" means:

Got a pair of equal A inputs (0,0 or 1,1) on the previous two ticks.

Remains in "OK" state as long as A remains constant or B=1.

But, how to know if A "remained constant" ? ➔ need to split "OK" state

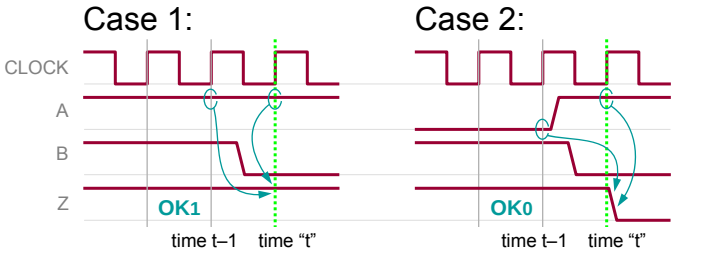
A Complete Design Example

- PROBLEM: Design a machine with inputs A and B and output Z that is "1" if any is true:
  - A had the same value at the *two previous* ticks

OR: – B has been "1" since *the last time* the above was true

Why we need to split the state "OK" into OK_0 and OK_1:

- If we don't know what was the value of A before time "t", then when B≠1, we cannot know if A "remained constant" (Case 1) or not (Case 2)



OK_0: Machine arrived to "OK" via 2×(A=0)

OK_1: Machine arrived to "OK" via 2×(A=1)

A Complete Design Example

- PROBLEM: Design a machine with inputs A and B and output Z that is "1" if any is true:
  - A had the same value at the *two previous* ticks

OR: – B has been "1" since *the last time* the above was true

Machine arrived to "OK" via 2×(A=0)

Machine arrived to "OK" via 2×(A=1)

Meaning	S	A B 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK_0	OK_0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK_1	OK_1	0
Two equal, A=0 last	OK_0	OK_0	OK_0	OK_1	A1	1
Two equal, A=1 last	OK_1	A0	OK_0	OK_1	OK_1	1
		S*				

- We achieved "closure" of the state table, which now describes a *finite*-state machine

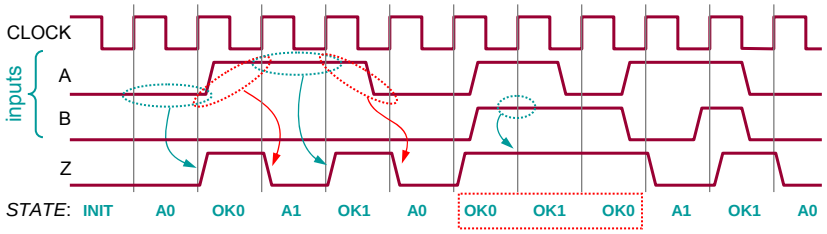
Timing Diagram for Example FSM

- Output Z is "1" if any is true:
  - A had the same value at the *two previous* ticks

OR:

  - B has been "1" since *the last time* the above was true

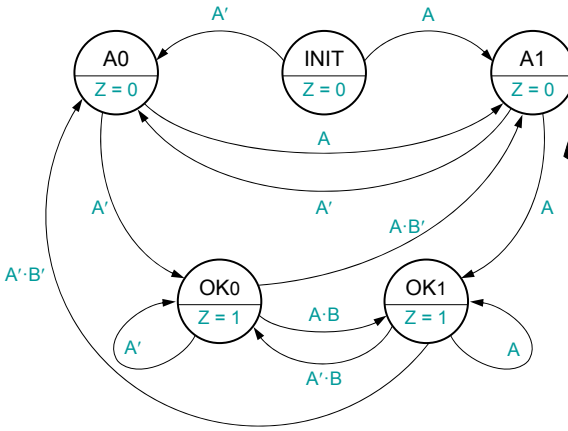
S	A B 00	01	11	10	Z
INIT	A0	A0	A1	A1	0
A0	OK0	OK0	A1	A1	0
A1	A0	A0	OK1	OK1	0
OK0	OK0	OK0	OK1	A1	1
OK1	A0	OK0	OK1	OK1	1
S*					



illustrates why OK0 vs. OK1 needed

State Diagram

- State Diagram is drawn from the state/output table:
  - First draw ovals for all states
  - Second, for each state ("current state") draw outgoing arcs for different inputs
    - The endpoint of an arc ("next state") is determined by the state table:



inputs

S	A B 00	01	11	10	Z
INIT	A0	A0	A1	A1	0
A0	OK0	OK0	A1	A1	0
A1	A0	A0	OK1	OK1	0
OK0	OK0	OK0	OK1	A1	1
OK1	A0	OK0	OK1	OK1	1
S*					

current state

next state

State Assignment

- Determine how many binary variables to represent the states in the state table
  - For **s** states we need $\lceil \log_2 s \rceil$ binary variables
- Coded state = binary combination assigned to a particular state
- In our example:
  - five states → $\lceil \log_2 5 \rceil = 3$
  - $2^3 = 8$ → three unused binary combinations (a.k.a. unused coded states)

15 of 25

State Assignment

- There are $\binom{8}{5}$ = 6,720 different state assignments of 5 states to 8 possible states (3 binary variables)
  - And there are many more using 4 or more binary variables
- Simplest is *counting order*, but may not lead to simplest excitation & output equations, nor the simplest logic circuit

	Coded State Assignment
State Name	***Simplest Q1–Q3***
INIT	000
A0	001
A1	010
OK0	011
OK1	100

16 of 25

Heuristics for "Best" State Assignment

- Choose an initial coded state into which the machine can be easily forced at reset
- Minimize the number of state variables that change on each transition
- Maximize the number of variables that don't change in a group of "related" states
- Exploit symmetries in problem-spec / state-table
- Decompose the set of variables into individual bits, where each bit has a well-defined meaning w.r.t. input effects or output behavior of the machine
- Etc. → see Wakerly, 4th edition, page 561

State Assignment Examples

- Here are a few "obvious" or "interesting" assignments
  - *Decomposed*:
    - Initial state is "000", which is easy to force to, e.g., applying RESET signal to flop-flops' CLR inputs
    - For remaining four states, Q1 used to indicate if the machine is in INIT
    - When Q1=1, Q2 and Q3 used to distinguish among the four non-INIT states
  - *One-hot* uses one bit per state (one-out-of-5 pattern: 5 bits instead of minimum 3)
    - Advantage: leads to simple excitation equations || Disadvantage: requires more flip-flops
  - *Almost One-hot* — uses "no-hot" combination "0000" for the initial state

	Coded State Assignment			
State Name	***Simplest Q1–Q3***	***Decomposed Q1–Q3***	***One-hot Q1–Q5***	***Almost One-hot Q1–Q5***
INIT	000	000	00001	0000
A0	001	100	00010	0001
A1	010	101	00100	0010
OK0	011	110	01000	0100
OK1	100	111	10000	1000

Dealing with Unused States

- Minimum risk:
  Assuming that the machine may somehow get into one of the unused (or "illegal") states, all unused states automatically go to the "initial" state
  (we will *first* use this design for our example)
- Minimal cost:
  Assuming that the machine will never enter an unused state, all unused states are labeled with "d" (don't-care) and are used if the minimization requires it
  (will see this design later)

Transition/Output Table

- For *transition table*, simple textual substitution
- Assuming "decomposed" state assignment:

	A B				
S	00	01	11	10	Z
INIT	A0	A0	A1	A1	0
A0	OK0	OK0	A1	A1	0
A1	A0	A0	OK1	OK1	0
OK0	OK0	OK0	OK1	A1	1
OK1	A0	OK0	OK1	OK1	1
	S*				

state/output table

	A B				
Q1 Q2 Q3	00	01	11	10	Z
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	Q1* Q2* Q3*				

transition/output table

Excitation Table ("decomposed" state assignment)

- Assuming D flip-flops (characteristic equation Q∗ = D), *excitation table* is identical to transition table, D = Q∗

	A B				
Q1 Q2 Q3	00	01	11	10	Z
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	Q1∗ Q2∗ Q3∗				

	A B				
Q1 Q2 Q3	00	01	11	10	Z
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	D1 D2 D3				

transition/output table

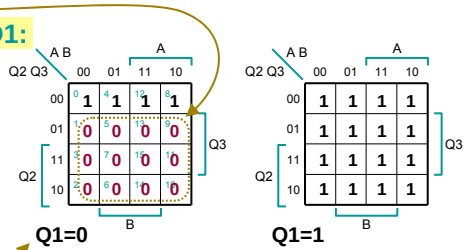
excitation/output table

Excitation Table as Truth Table

- Excitation table as *truth table* for three combinational logic functions (D1, D2, D3) and five variables (A, B, Q1, Q2, Q3)
- Developing excitation equations using a *5-variable Karnaugh map* [recall Lecture #7]
- But, excitation table is not quite a truth table—doesn't specify functional values for *all* input combinations (i.e., unused states)
  - *Minimal risk* approach to handling unused states: next-state = INIT = 000

Three states {001, 010, 011} are not used!

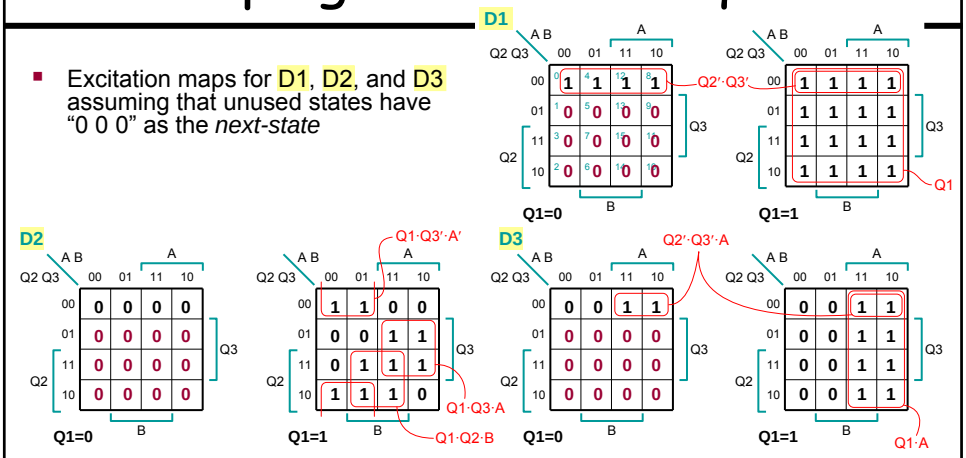
	A B				
Q1 Q2 Q3	00	01	11	10	Z
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	D1 D2 D3				



excitation/output table

Developing Excitation Equations

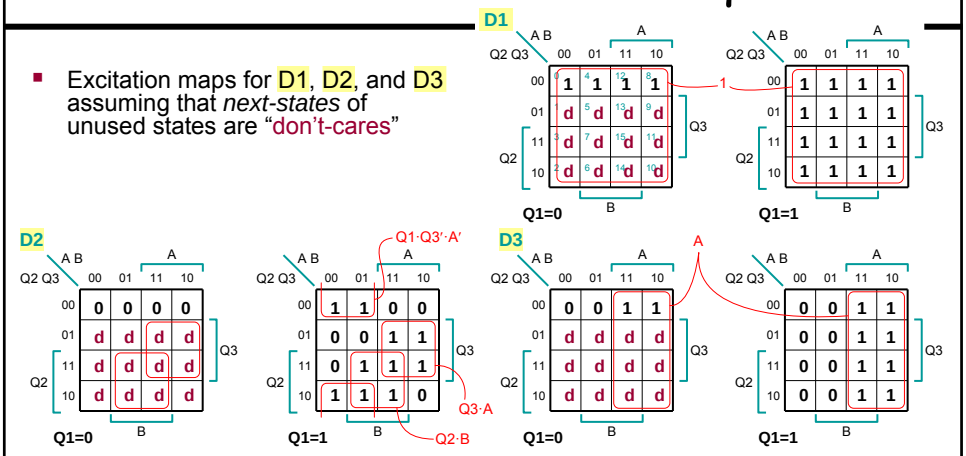
- Excitation maps for D1, D2, and D3 assuming that unused states have "0 0 0" as the *next-state*



- Flip-flop excitation inputs:
  D1 = Q1 + Q2′·Q3′
  D2 = Q1·Q3′·A′ + Q1·Q3·A + Q1·Q2·B
  D3 = Q1·A + Q2′·Q3′·A
  Z = Q1·Q2·Q3′ + Q1·Q2·Q3 = Q1·Q2

Minimal Cost Excitation Equations

- Excitation maps for D1, D2, and D3 assuming that *next-states* of unused states are "don't-cares"



- Flip-flop excitation inputs are now simpler ("minimal cost"):
  D1 = 1
  D2 = Q1·Q3′·A′ + Q3·A + Q2·B
  D3 = A
  Z = Q2

draw the circuit …

Minimal Cost Circuit

- Logic diagram for the excitation maps using "don't-cares" as next states of unused states