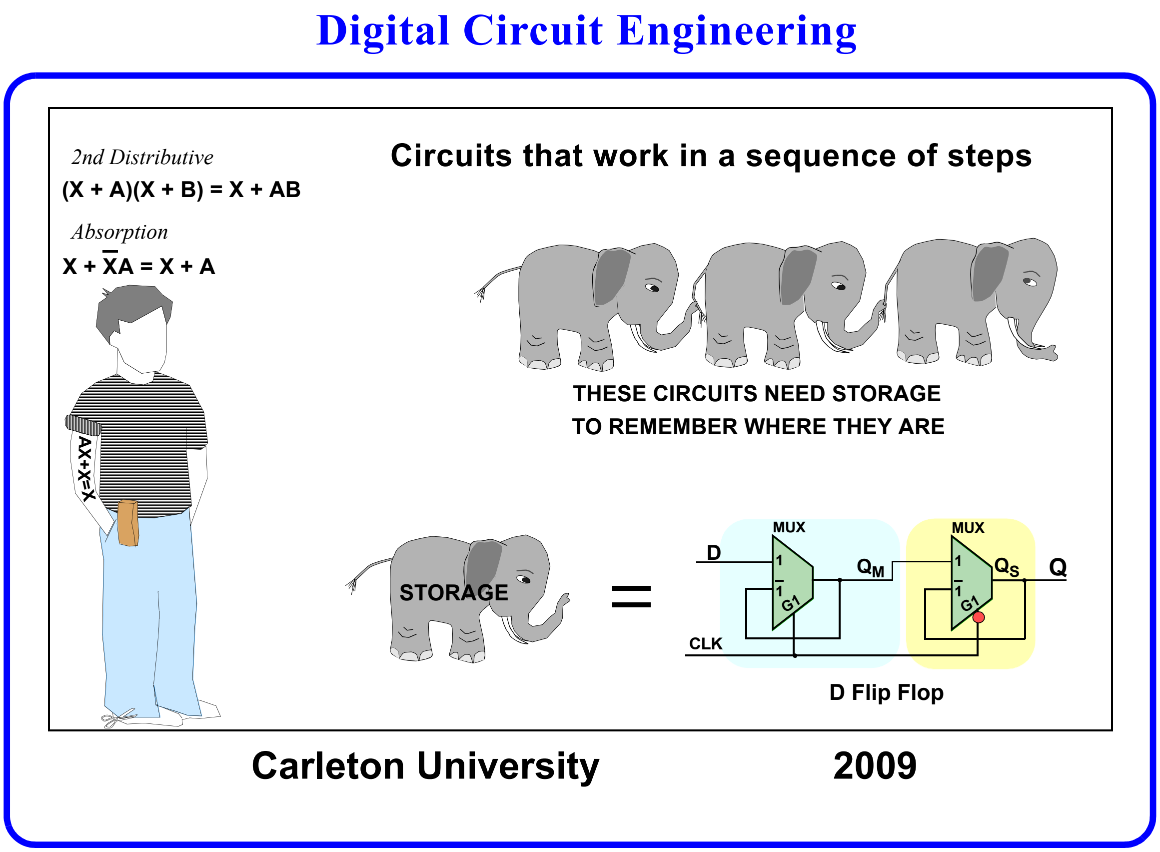


Digital Circuit Engineering

2nd Distributive

$(X + A)(X + B) = X + AB$

Absorption

$X + \overline{X}A = X + A$

Circuits that work in a sequence of steps

THESE CIRCUITS NEED STORAGE
TO REMEMBER WHERE THEY ARE

D Flip Flop

Carleton University **2009**

▪

Finite State Machines

- Definition of a State

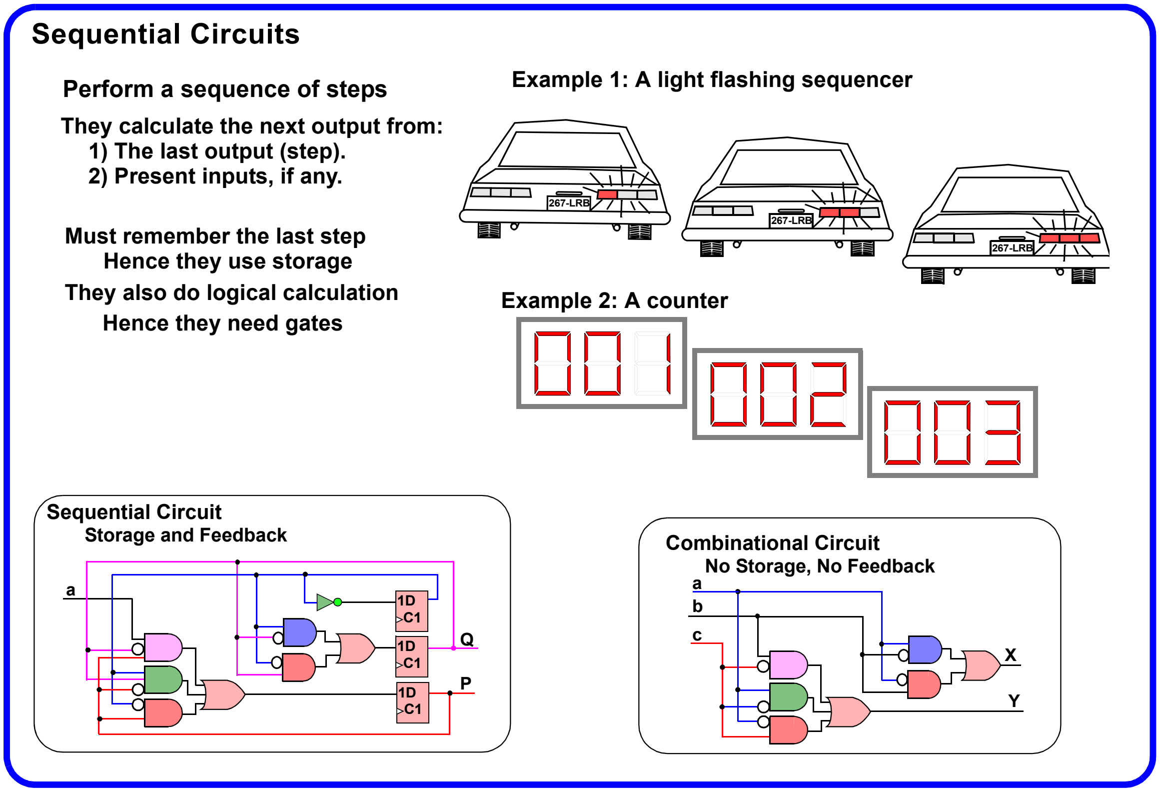
Sequencing of States

- State Graphs
- State Tables
- State Table to Karnaugh Map to Circuit
- Mealy and Moore Outputs
- Constructing The Output Circuits From The State Table
- Properties of Mealy and Moore Outputs

Overview of Designing Finite State Machines

- Construction of State Graphs
- Examples
  - Toggle Flip Flop
  - Detecting an Input Sequence

Sequential Circuits



Sequential Circuits

Perform a sequence of steps

They calculate the next output from:
1) The last output (step).
2) Present inputs, if any.

Must remember the last step
Hence they use storage

They also do logical calculation
Hence they need gates

Example 1: A light flashing sequencer

Example 2: A counter

Sequential Circuit
Storage and Feedback

Combinational Circuit
No Storage, No Feedback

Sequential Circuits

These circuits contain storage

- They remember the result of previous calculation.
- The total information that they remember is called the *state* of the circuit.
- The *state* and any inputs are used to calculate what the *next state* will be.
- After a delay, here the time to the next active clock edge, this *next state* becomes the *state*.

Since they go through a sequence of states, they are called *sequential circuits*.

Since the amount of storage limits them to a finite number of states,
they are called *Finite-State Machines.*

What is storage

In theory, any storage counts. This could be flip flops, latches and RAM. Read-only storage, or storage that cannot be changed by the machine, cannot be part of an FSM.

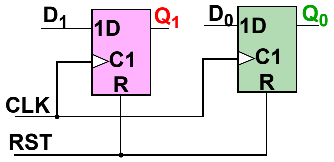
Software *Finite-State Machines*, usually consider RAM the storage.

Hardware *Finite-State Machines* use flip flops for storage, and change state just after an active clock edge. This is a matter of convenience. It is much much easier to design machines with clocked flip flops. However at the end of the course, we will consider a type of circuit called *asynchronous finite-state machines* (a*synchronous circuits* for short), which uses latches, and do not use a clock

States; Finite-State Machines

The State

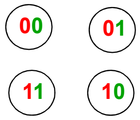
- The ***state*** is the output of all storage elements.
  Collected in some prearranged order.
- Machines based on states are called ***State Machines***.
  In digital they are ***Finite-State Machines*** (FSMs).
  We can't have an infinite number of digital states.
- In this course, and most designs, the storage is flip flops.



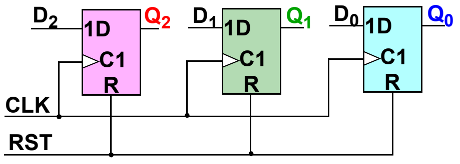
two flip flops

Four
Possible States

Q_1	Q_0
0	0
0	1
1	1
1	0



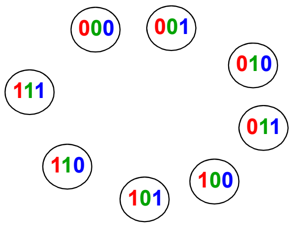
We often show the states inside circles



three flip flops

Eight
Possible States

Q_2	Q_1	Q_0
0	0	0
0	0	1
0	1	1
0	1	0
1	0	0
1	0	1
1	1	1
1	1	0



Finite-State Machines

The state is the output of all storage elements collectives in some prearranged order

Take a finite-state machine with 4 flip flops, and let the order in the state be alphabetical, that is Q_A,Q_B,Q_C,Q_D. If Q_A,Q_B,Q_C,Q_D contain 1,1,0,1 at this moment, then the state is 1,1,0,1.

The order is arbitrary; you could use say Q_A,Q_D,Q_C,Q_B, only once you define it for one state, you must use the same order for all states.

8-1.•PROBLEM

How many states are possible with 8 flip flops?

8-2.•PROBLEM

How many flip flops would be needed to make a digital counter that would count to 20 in binary?

Finite-State Machines

Basis Finite-State Machine

States

What is stored in all memory in the machine

Inputs

External signals entering the machine.

Clock and reset are special and usually don't count as inputs

Outputs

Output signals like Y and Z

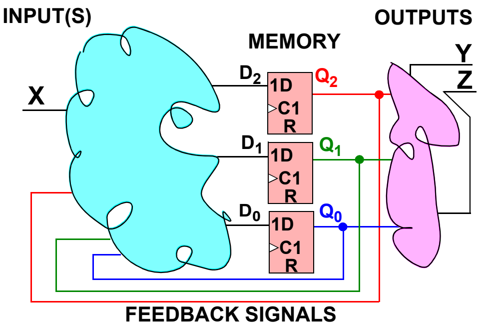
Feedback Signals

Here Q_0 Q_1 and Q_2

Order for going between states

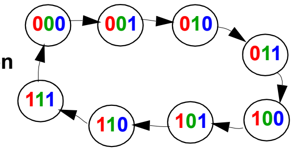
We have to specify an order based on:
1) the present state.
2) inputs.

Which define the next state.



CLK and RESET inputs assumed.
See next slide.

An order for going between states



Questions

a. What are the inputs?
b. What are the outputs?
c. If all the memory is in the 3 flip-flops, what is in the colourful blobs?
d. If the machine above has $Q_0Q_1Q_2$=010, what state is it in?
e. How many different states is it possible for this machine to have?
f. All the machine knows about its previous history is contained in?

Sequencing of States

State

All memory in the machine, means all in the part under consideration. A large machine will almost certainly be partitioned into smaller FSMs, in which case one considers only the memory in the partition.

Inputs

Most, but not all, machines have an external input. If you partition a big FSM into two smaller ones, the outputs of one FSM will normally provide inputs to the other.

Feedback signals

State and Next State

The machine has its flip flops in some state, say 101. What is important is the next state. The way the machine sequences *state* --> *next state* is the main difference between two machines.
All the machine knows about its previous history is contained in the state.
The feedback signals send the *state* back into logic which calculates the *next state*. Feedback are output signals which are fed back as (internal) inputs.

8-3.•PROBLEM

Design the state graph for a counter that counts down 0->7->6->5->4->3->2->1->0->7... in binary.

8-4.•PROBLEM

Design the state graph for a counter that counts down 0->7->6->5->4->3->2->1->7->6... in binary. Note that the zero states comes up on initial reset using the R pin on the flip flops, but cannot be entered again. The reset signal is not cosidered in this graph.

Answers a) There is one, X. b) Y and Z. c) logic gates with no feedback. d) 010. e) 8. f) the state (the flip flops)

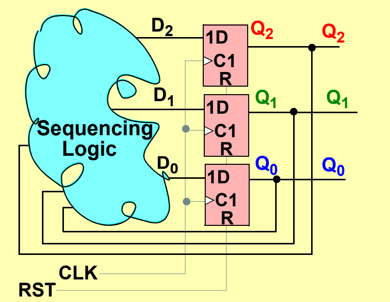
FSM: Sequencing; Counters, State Graph

Counter

This circuit counts in binary.

Increments by one every rising clock edge.

- Count from 0 to 7 (000 to 111) and repeat.



Counter Finite-State Machine

No inputs

Clk is an input we don't count
We often don't show the wires

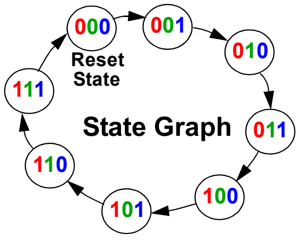
RST is an input we don't count
We often don't show the wires
Here wire is behind the flip flops

Outputs are the state
In this example

Eight states
Arranged to
count

	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

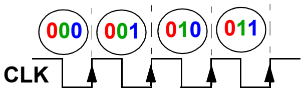
time



The sequencing between
states is all important.

The state graph
shows sequencing between states.

The state changes after
each rising clock edge.



Use RST to make the count start at zero. 000

Counters

This counter is an FSM which has no inputs except the clock and a reset. With no other inputs, such a circuit cannot do anything except count in some sequence. The sequence here is binary, but another counter might count in binary-coded decimal, two's complement, or whatever one wants.

The sequence is cyclic, so once the counter reaches 7 (111) it automatically goes back to 0 (000) and starts counting again. We say such a counter counts *modulo* 8, or it is a *mod 8 counter*.

RSR

The reset (RST) signal is essential to a well behaved counter since, it can be used to start the count at zero. Without it, the flip flops would come in a random state when the power was applied to them, and one would not know where the counter would start.

State Change

The state changes immediately after the active clock edge.
This is the only time the state can change.

Connections to CLK and RST

Just like one does not show the power and ground connections on theoretical diagrams, one often does not show the clock and reset signals. In the Slide above they are shown, although one has to infer that the RST wire goes underneath the flip flops. On Slide 4 only the terminals on the flip flop are shown, and the reader must know how they are connected.

When counting inputs, we say the counter above has no external inputs, and don't count CLK and RST.

FSM: Sequencing of States

Sequencing of States

The next state depends on:

a. The present state

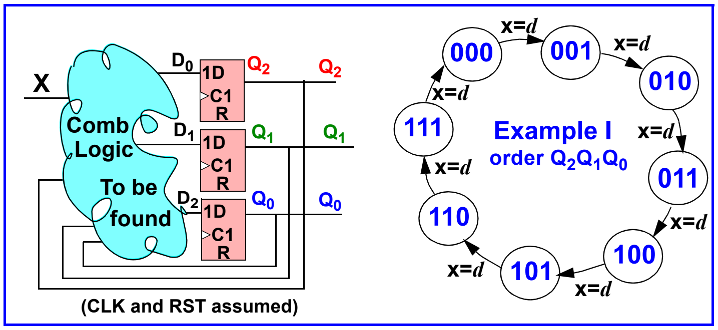
b. The inputs

State Graph

Shows sequencing

State is inside circle

Input is beside arrow



(CLK and RST assumed)

Example I

Counter, as on last slide.

Input not used

Write x=*d*, or put nothing beside the arrow.

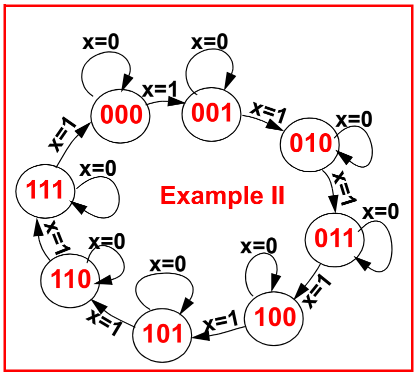
Change state each rising clock edge.

Example II

Controlled Counter

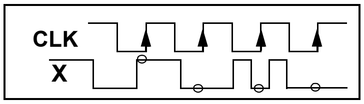
Count (go to next state) when x=1

Hold old count (stay in old state) when x=0



Question

When, in the clock cycle, is X checked to see if the state changes or not?



Sequence of State

Three Counters (two above, one on Slide 8)

1. Example I has an input, but at might as well not have, because it doesn’t use it
2. Example II has an input which stops the count when X=0.
3. Example III (Slide 8) makes the count become 4 (100) whenever X=0.

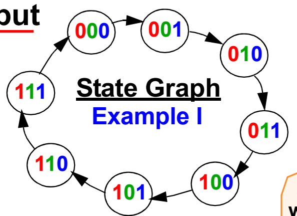
The difference between these three counters is easily seen on the state graph

FSM: Sequencing of States; State Tables

State Tables With and Without Input

The state graph and the state table.
Both show sequencing between states.
The graph is easier for initial design.
The table is used for final design.

No Input



State Table Example I

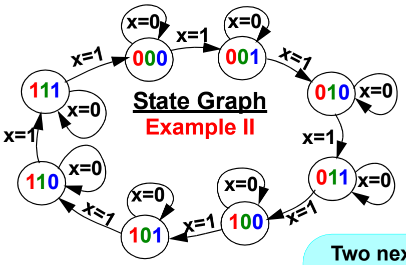
Present State Q_2	Q_1	Q_0	Next State Q_2^+	Q_1^+	Q_0^+
0	0	0	0	0	1
0	0	1	0	1	0
0	1	1	1	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	0	1	1	1	0
1	1	1	0	0	0
1	1	0	1	1	1

K-map order

Next State written as Q^+

Next State comes after next rising clock edge

Input



State Table Example II

Two next state columns

Present State Q_2	Q_1	Q_0	x=0 Next State Q_2^+	Q_1^+	Q_0^+	x=1 Next State Q_2^+	Q_1^+	Q_0^+
0	0	0	0	0	0	0	0	1
0	0	1	0	0	1	0	1	0
0	1	1	0	1	1	1	0	0
0	1	0	0	1	0	0	1	1
1	0	0	1	0	0	1	0	1
1	0	1	1	0	1	1	1	0
1	1	1	1	1	1	0	0	0
1	1	0	1	1	0	1	1	1

K-map order

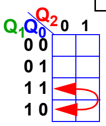


Table is arranged like a K-map
because later we transfer it to K-maps

State Tables

State tables contain the same information as the state graph. Once you have the graph, you can easily construct the table.

The states can be arranged in any order; just make sure the state and next state entries match. For counters, it is common to arrange them in the same sequence as going around the state graph. However it will turn out that the present state is best arranged in Karnaugh Map order. This makes it easy to convert the table into maps.

If the present state is written as Q..., we commonly write the next state as Q^+...

How Many Next State Columns

When there is no input, or the input does nothing like in example I. then there is only one next state column.

When there is an input, there are two next states. This means the state table needs two next state columns.

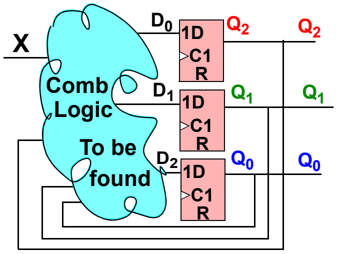
If there were two inputs, there would be 4 next states, and there will be 4 next-state columns in the state tables.

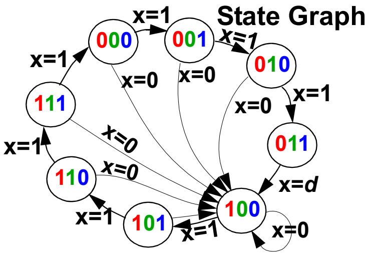
FSM: State Table & State Graph; Inputs

Example III: Binary Counter With “Go To 4” Input

- If x=1 counts in binary: 0→1→2→3→4→5→6→7→0→1→2...
- If x=0, go to count of 4 (100) and stays until x=1

(CLK and RST assumed)





Any time x=0,
the next state is 4 (100).

State Table

K-map order

Present State			Next State x=1			Next State x=0		
Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	Q_2^+	Q_1^+	Q_0^+
0	0	0	0	0	1	1	0	0
0	0	1	0	1	0	1	0	0
0	1	1	1	0	0	1	0	0
0	1	0	0	1	1	1	0	0
1	0	0	1	0	1	1	0	0
1	0	1	1	1	0	1	0	0
1	1	1	0	0	0	1	0	0
1	1	0	1	1	1	1	0	0

The state graph and the state table both show sequencing between states.

The sequencing is different between when x=1 and x=0

Example III State Graph and State Table

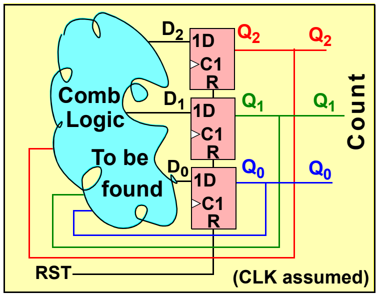
A counter that goes to state 4 (100) when input X=0

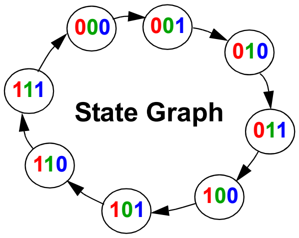
Note the table is arranged, not following order of the arrows in the state graph, but so that Q_1Q_0 follow the order of a K-map.

FSM: Counter Design; Completing Details

Completing the 0 to 7 Counter Design

Design so far: It counts 0→1→2→3→4→5→6→7→0→1→2...
It increments by one on every rising clock edge
Reset takes the count (state) to 000





State Table

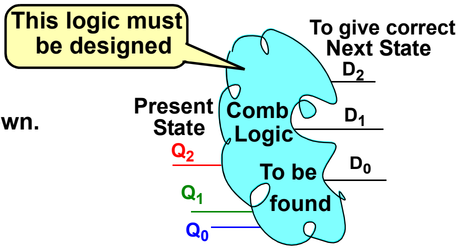
Present State				Next State		
Q_2	Q_1	Q_0		Q_2^+	Q_1^+	Q_0^+
0	0	0		0	0	1
0	0	1		0	1	0
0	1	1		1	0	0
0	1	0		0	1	1
1	0	0	K-map	1	0	1
1	0	1	order	1	1	0
1	1	1		0	0	0
1	1	0		1	1	1

Circuit Design

3 flip-flops can give the 2^3 = 8 states
The *Count* output is taken from the flip flops
RST clears the flip-flops to state 000.

Only the next-state (combinational) logic is unknown.

It has inputs Q_2, Q_1, and Q_0
It has outputs D_1, D_2, and D_0
These Ds must give the right next state after the clock edge.



From State Graph to Circuit

Start with the State graph

This was done before

Construct the State Table

This was done before

Consider What to do Next

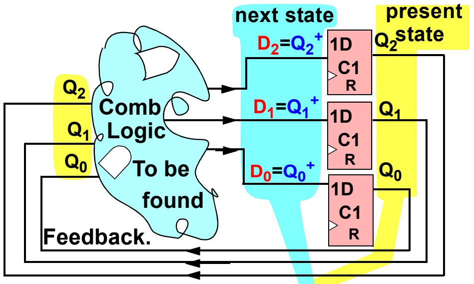
We know the circuit has 3 flip flops.
We want the the D inputs to the flip flops to be the correct next state.

We know this will be a combinational (gates only) circuit which has the state Q_2, Q_1,Q_0 as its input and gives out D_2, D_1, D_0 as the output.

This circuit will be designed from Karnaugh maps.

FSMs: Design of a Binary Counter

Counter Design



The present state is the value of Q_2 Q_1 Q_0
the present Q outputs

The next state is the value of Q_2^+ Q_1^+ Q_0^+
the present D inputs.

Design the combinational (next state) logic
so that inputs Q_2 Q_1 Q_0
give outputs D_2 D_1 D_0 = Q_2^+ Q_1^+ Q_0^+

Counter state table

Count	State Q_2 Q_1 Q_0	Next State $Q_2^+Q_1^+Q_0^+$	D inputs D_2 D_1 D_0
0	0 0 0	0 0 1	0 0 1
1	0 0 1	0 1 0	0 1 0
3	0 1 1	1 0 0	1 0 0
2	0 1 0	0 1 1	0 1 1
4	1 0 0	1 0 1	1 0 1
5	1 0 1	1 1 0	1 1 0
7	1 1 1	0 0 0	0 0 0
6	1 1 0	1 1 1	1 1 1

K-map order

Method:

From the state table
Make K-maps

Make 3 maps; each with inputs Q_2 Q_1 Q_0
map for D_2
map for D_1
map for D_0

The row order in the state table does not matter <u>in theory</u>.

K-Map Order

Constructing the rows in K-map order makes drawing maps easier.

Counter Design

Next state and the D inputs

The present state is the present value of Q_2,Q_1,Q_0
The next state is the present value of Q_2^+,Q_1^+,Q_0^+ which is equal [1]to D_2,D_1,D_0

Clock and Reset

The clock and reset lines to the flip flops are not shown in theoretical diagrams, but you cannot eliminate them in a real circuit!

8-5.•PROBLEM

Do Prob 8-8-4.• first, which is: “Design the state graph for a counter that counts down 0→7→6→5→4→3→2→1→7→6... in binary. Note that the zero states comes up on reset, and is not entered again.”

Make the state table for a counter that counts down 0→7→6→5→4→3→2→1→7... in binary. Include columns for the state, the next state, and the D input that will give that next state.
Arrange the rows so the states are in Karnaugh map order, i.e. 0,1,3,2; 4,5,7,6.

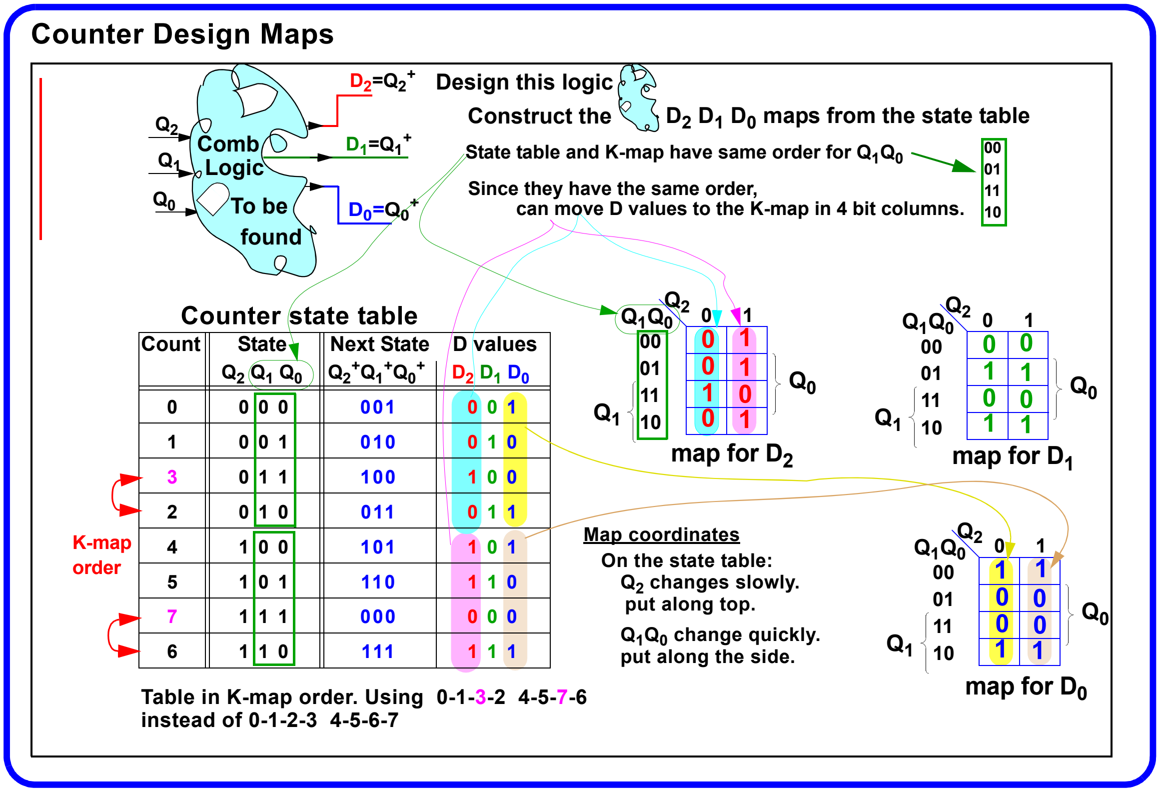
8-6.•PROBLEM (CHANGED FROM IEEE PRINTING)

Design a counter that counts in reflected Gray code where only one bit changes as one goes between states. This code counts
000→001→011→010→110→111→101→100→000
Make:
- a state graph
- a state table in K-map order.
- three K-maps as is done on Slide 11.

---

[1] This is true for FSMs using D flip-flops. Older books design circuits with J-K flip-flops where the flip-flop inputs are not equal to the next state. Be thankful you no longer have to learn to use them! It is also not true for the T flip-flops you will use in the MIDI lab.

FSMs: Design of a Binary Counter



Counter Design Maps

Design this logic

Construct the D2 D1 D0 maps from the state table

State table and K-map have same order for Q_1Q_0 → 00 01 11 10

Since they have the same order, can move D values to the K-map in 4 bit columns.

Counter state table

Count	State Q_2 Q_1 Q_0	Next State $Q_2^+Q_1^+Q_0^+$	D values D_2 D_1 D_0
0	0 0 0	001	0 0 1
1	0 0 1	010	0 1 0
3	0 1 1	100	1 0 0
2	0 1 0	011	0 1 1
4	1 0 0	101	1 0 1
5	1 0 1	110	1 1 0
7	1 1 1	000	0 0 0
6	1 1 0	111	1 1 1

Table in K-map order. Using 0-1-3-2 4-5-7-6 instead of 0-1-2-3 4-5-6-7

Q_1Q_0 Q_2	0	1
00	0	1
01	0	1
11	1	0
10	0	1

map for D_2

Q_1Q_0 Q_2	0	1
00	0	0
01	1	1
11	0	0
10	1	1

map for D_1

Map coordinates

On the state table:
Q_2 changes slowly.
put along top.
Q_1Q_0 change quickly.
put along the side.

Q_1Q_0 Q_2	0	1
00	1	1
01	0	0
11	0	0
10	1	1

map for D_0

Counter Design Maps

Order the table rows to match the map

To make the data easy to transfer from the state table to the map:

- Order the table rows so that Q1 and Q0 change in the table the same way they would along the y-axis of the map.

Choose the map axis to match the table

- Put rapidly changing state variables Q_1,Q_0 on the vertical map axis.
- Put slowly changing map variable(s) Q2 on the horizontal map axis.

Transfer the D values from the state table to the K-map

- This can be done in blocks of four values.

Looping the maps

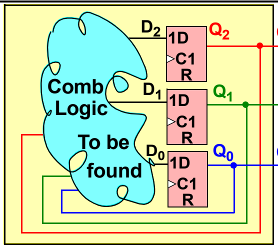
We now have a three-output gate (combinational) logic problem. We want to loop the three maps to minimize the logic. Don't forget that this is a multiple output problem, and one may be able to share gates.

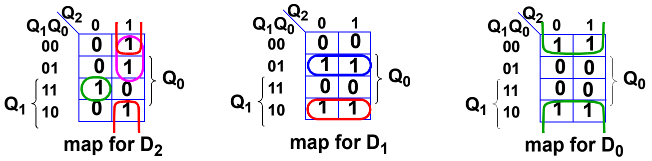
FSMs: Counter Circuit; Obtaining Circuit

Get Equations and Circuit

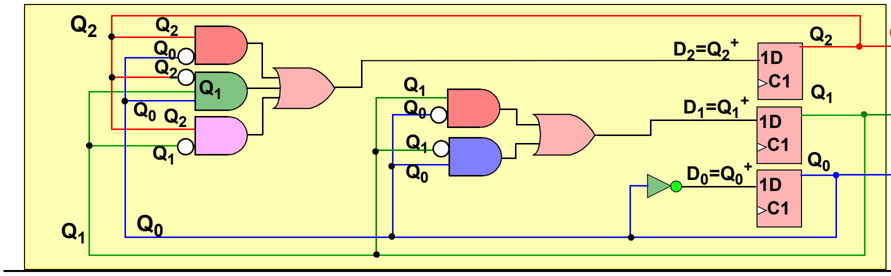
K-Maps from last slide
Loop them
Write the equations from the loops
Draw the circuit

Original High-level View of Circuit





$D_2 = Q_2\overline{Q}_1 + \overline{Q}_2 Q_1 Q_0 + Q_2\overline{Q}_0$ $D_1 = \overline{Q}_1 Q_0 + Q_1\overline{Q}_0$ $D_0 = \overline{Q}_0$



Final Detailed View of Circuit

Counter Design

Review of steps used (so far) in Designing Finite-State Machines

- Understand the problem. This is the hardest part for more complex machines.
- Draw a state graph
- Construct a state table showing the next state. Here the states were named 0, 1, 2, 3, 4 . . .
- Fill in bits for the states. Here we translated 2→ 010, 3→ 011 ...
- Put the state table in K-map order
- Draw the K-maps from the state table.
- Loop the K-maps to get the best equations.
- Draw the logic from the equations.

8-7.•PROBLEM

Do problems 8-4.•, 8-5.•,. Then:
a) write out the Karnaugh maps for the three flip-flop inputs
b) loop the Karnaugh maps[1],
c) obtain the next state logic equations,
d) and draw the circuit including the flip flops,
for a counter that counts down:
0→7→6→5→4→3→2→1→7→6→ ...

8-8.•PROBLEM (Uses XK-maps)
Using XORs, show that D2 has a very simple XSofP expression. (3 letters)

---

[1] A+ students know that this is a multiple-output circuit and loop accordingly.

FSMs: State Graphs With Less Than 2^N States

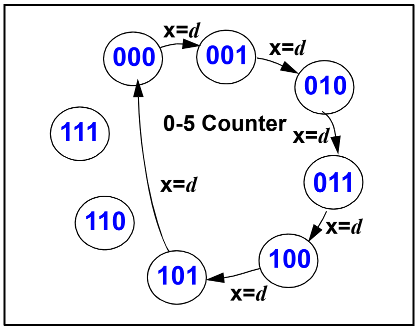
All possible states may not be used

Example Counter

Counts 0-to-5

States 111 and 110 not included in sequence

Those states still exist.
What happens too them?



What to do with unused states

Lock Up Counter

Normally counter counts properly, but

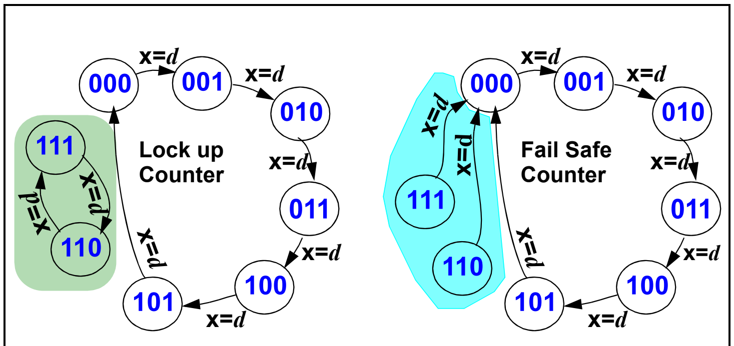
if one gets into state 110 or 111,
one cannot get out.

Fail Safe Counter

If one gets into state 111 or 110,
one goes to state 000 on next clock edge.

Question

Which is the better design?



What to do with unused states

Machines With Less Than 2^N states

What causes going into a bad state?

- Not resetting flip-flops properly on start-up.
- Fast power supply surges which flips a flip-flop that should not have been flipped.
- Large pulses on nearby wires which capacitively couple to a flip-flop input (*Cross-talk*).

The chances of getting into a bad state are very small, but it does happen.

Answer

The lock-up circuit requires one to reset the circuit to get out of the lock-up states

The fail-safe circuit lets the counter extract itself.

Which is better depends on the application. Most people think the fail-safe circuit is better. However the lock-up circuit will let you know if the count went bad. The fail-safe circuit will continue with the wrong count.

Comments on using the extra states

If the sequencing for the extra states is not given at all, then these states can have don’t care next states. These can be used to reduce the amount of logic in the blob.

Symbolic State Names

Often Convenient to Give the States Names

Example: 0-6 Counter, Which Counts if X=1

Suggestive names

Starting state is often RST (reset)

Counting states are B1(1st bottle), B2, ... B5(5th bottle)

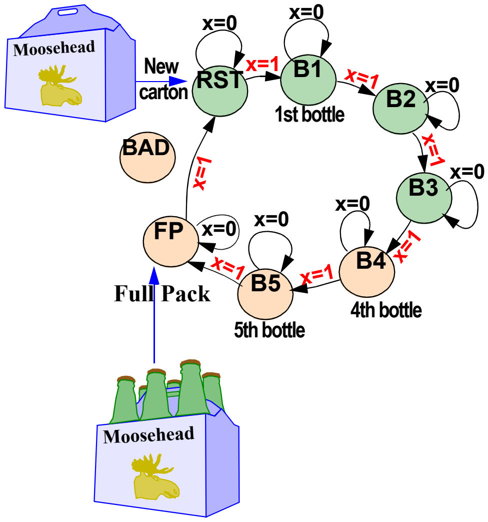
State where the 6-pack is full is FP

Make bad state BAD.

State Table with Symbolic States

State	Next State $Q_1^+Q_2^+Q_3^+$	
	x=0	x=1
RST	RST	B1
B1	B1	B2
B3	B3	B4
B2	B2	B3
B4	B4	B5
B5	B5	FP
BAD	*d d d*	*d d d*
FP	FP	S0

Bottle Counter



Symbolic State Names

8-9.•PROBLEM

Design a the state graph for a circuit which counts 0 to 6 bottles using four bits. The most significant bit is a even- parity bit which is set to always give an even number of 1s in the four bits. Thus the count

B1→B2→B3→B4→B5→FP→,

will, after state assignment read,

0000→1001→1010→0011→1100→0101→0110.

If the counter enters a bad state, have it recover by going to the highest count which differs by 1 bit from the bad count. This assumes the chances of more than a one-bit error are small, and that it is better to underfill a 6-pack than smash bottles.

8-10.•PROBLEM

Design the state graph of a counter which counts 0-2-4-6. If some circuit noise places it in states 1,3,5 or 7, it will go to state 7 on the next clock cycle and stay there. This indicates that something is wrong.

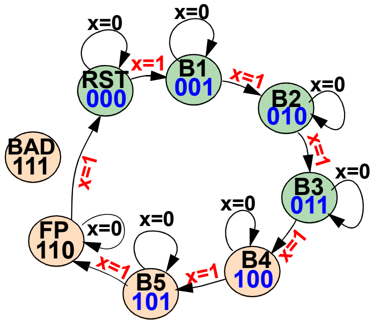
Give Bit Patterns to Symbolic Names

Assigning Bit Patterns

Called *State Assignment*

Example: 0-6 Counter, Which Counts if X=1

Assign RST=000; B1=001; B2=010; B3=011
B4=100; B5=101; FP=110
BAD=111 (that's all that's left)



Make a State Table from state graph infprmation

The State Table Contains

- One present *State* column
- Two *Next State* columns, one for x=0, one for x=1.
- Will use both symbolic names; RST, B1 . . . and bit patterns
- Arrange table in Karnaugh map order
- Make unused state BAD have don't care next states This gives minimum logic.

State Table,

State	Next State $Q_1^+Q_2^+Q_3^+$	
$Q_2Q_1Q_0$	x=0	x=1
RST=000	RST=000	B1=001
B1=001	B1=001	B2=010
B3=011	B3=011	B4=100
B2=010	B2=010	B3=011
B4=100	B4=100	B5=101
B5=101	B5=101	FP=110
BAD=111	*d d d*	*d d d*
FP=110	FP=110	RST=000

Questions

Does it matter what order we use for the present states in the table?

What do we mean by Karnaugh map order?

Could one put a symbolic state table (last slide) in Karnaugh map order?

Is taking the next states for BAD as *ddd* a good strategy?

Assigning Bit Patterns

Answers

1. Since one places the present state on the same table line as the next state, a state table is still valid if these lines are placed in any order. If the circuit is a counter, one naturally wants to place these lines in order i.e. 0, 1, 2, 3. . .

2. Karnaugh map order is for the right-most two bits in the present state. Then if we use those two bits as the y-axis when we make a K-map from the state table, the K-map axis will be in correct order. The first bit will be used for the x-axis. If one is going to make Karnaugh maps from the table, it is very convenient to have the states arranged in the table in the order in which they will be placed on the map.

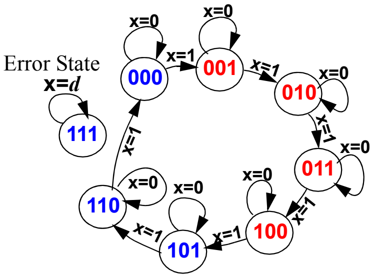
3. A symbolic state table has no bit patterns to place in K-map order.

On this and the next page, the state tables are in K-map order so it is very easy to transfer data from the state table to the K-maps a column at a time.

4. Making unused states don't cares will minimize the logic.

An alternatives is to make it "fail safe". In a previous slide, this was taken as sending the count to zero and continuing.

Another strategy would be to make 111 an "error" state. Whenever one got into 111 one would stay there.

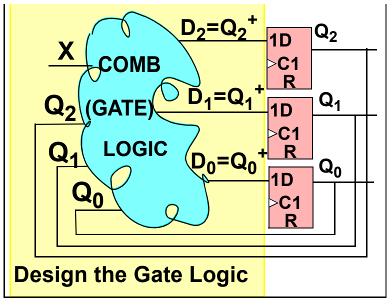


Logic for Counter With Unused State

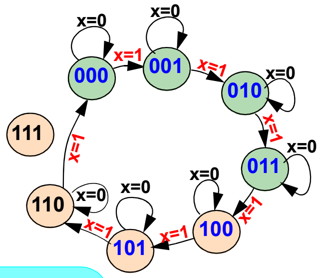
Obtain Logic From State Table

Example: 0-6 Counter Which Counts if X=1

- Gate Logic has inputs Q_2,Q_1,Q_0, X
- Gate Logic has outputs D_2,D_1,D_0
- The next state $Q_2^+,Q_1^+,Q_0^+ = D_2,D_1,D_0$
- Get K-Maps for Gate Logic from state table (next slide)
- Get Gate Logic from K-Maps (next slide)



Design the Gate Logic



State Table with D inputs added,

State	Next State $Q_2^+Q_1^+Q_0^+$		D inputs $D_2D_1D_0$	
$Q_2Q_1Q_0$	x=0	x=1	x=0	x=1
RST=000	RST=000	B1=001	000	001
B1=001	B1=001	B2=010	001	010
B3=011	B3=011	B4=100	011	100
B2=010	B2=010	B3=011	010	011
B4=100	B4=100	B5=101	100	101
B5=101	B5=101	FP=110	101	110
BAD=111	*d d d*	*d d d*	*d d d*	*d d d*
FP=110	FP=110	RST=000	110	000

Since $Q_2^+,Q_1^+,Q_0^+ = D_2,D_1,D_0$ we won't show them as seperate columns for very long.

Obtaining Logic From The State Table

K-Maps for Counter with Unused State

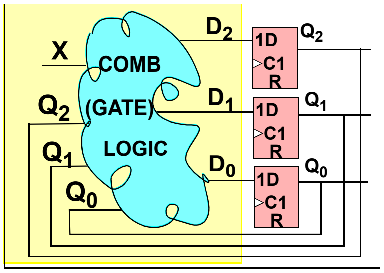
Obtain Logic From State Table; Making K-Maps

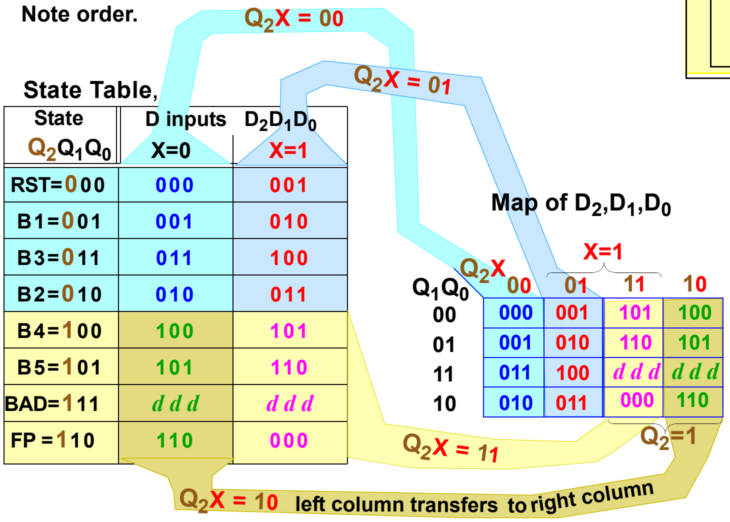
Example: 0-6 Counter Which Counts if X=1

Design the Gate Logic
Take state table in K-map order

Make Q2 and x the horizontal axis coordinates
must arrange Q2 and x in K-map order.

Transfer Next State patterns to K-map in blocks
Note order.





State Table,

State	D inputs	$D_2D_1D_0$
$Q_2Q_1Q_0$	X=0	X=1
RST=000	000	001
B1=001	001	010
B3=011	011	100
B2=010	010	011
B4=100	100	101
B5=101	101	110
BAD=111	*d d d*	*d d d*
FP =110	110	000

Map of D_2,D_1,D_0

		X=1		
Q_1Q_0 \ Q_2X	00	01	11	10
00	000	001	101	100
01	001	010	110	101
11	011	100	*d d d*	*d d d*
10	010	011	000	110

Obtain K-Maps From State Table

Write the state table so the two most rapidly changing bits are in Karnaugh map order (Q_1Q_0).

One needs a map relating the four inputs $Q_2XQ_1Q_0$ to the three outputs $D_2D_1D_0$.

Making the map with Q_1Q_0 as vertical coordinates and Q_2X as horizontal coordinates allows one to easily build such a map. The order is very close to the order in the state table so one can split the state the table in half, and put the bottom beside the top. However K-map order is not quite the same as the state table. To fix it so the X=1 columns are together one has to flip the left and right column.

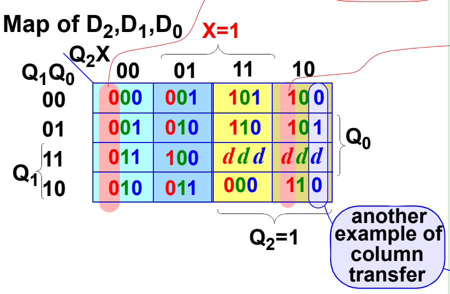
This map has three outputs in each square instead of one which is not good for loopling. This will be split into three maps on the next page.

Changes Maps to Have 1-Bit-Per Square

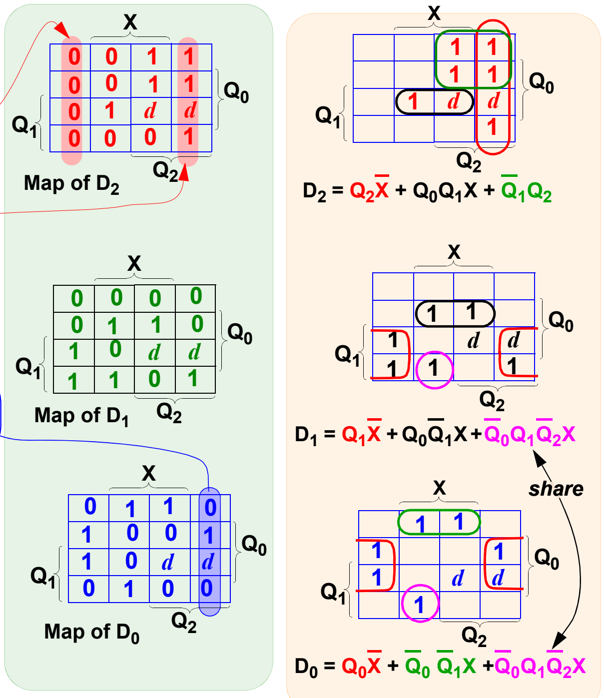
Change 3-Output/Square Map into Three 1-Output/Square Maps

0-6 Counter, Counts if X=1

- Can't loop 3-output map below
- Break into three maps by
  - Transfering columns



Q_1Q_0 \ Q_2X	00	01	11	10
00	000	001	101	100
01	001	010	110	101
11	011	100	ddd	ddd
10	010	011	000	110



Map of D_2

0	0	1	1
0	0	1	1
0	1	d	d
0	0	0	1

Map of D_1

0	0	0	0
0	1	1	0
1	0	d	d
1	1	0	1

Map of D_0

0	1	1	0
1	0	0	1
1	0	d	d
0	1	0	0

$$D_2 = Q_2\overline{X} + Q_0Q_1X + \overline{Q_1}Q_2$$

$$D_1 = Q_1\overline{X} + Q_0\overline{Q_1}X + \overline{Q_0}Q_1\overline{Q_2}X$$

$$D_0 = Q_0\overline{X} + \overline{Q_0}\ \overline{Q_1}X + \overline{Q_0}Q_1\overline{Q_2}X$$

- Loop the maps to get logic equations

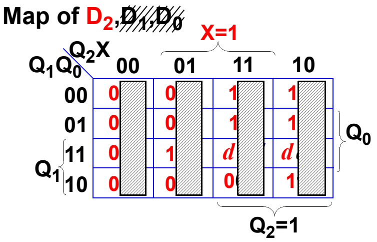
Questions

Should we have made the unused states don't cares?

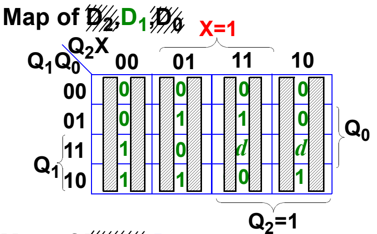
What are the alternatives?

Change 3-Output/Square Map into Three 1-Output/Square Maps

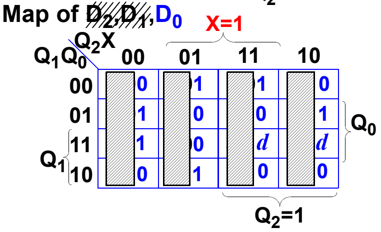
Above, the first bit in each square is D_2, so to get the K-maps for D_2, blank out the 2nd and 3rd bit in each square.



Above, the first bit in each square is D_1, so to get the K-maps for D_1, blank out the 1st and 3rd bit in each square.



Above, the first bit in each square is D_0, so to get the K-maps for D_0, blank out the 1st and 2nd bit in each square.



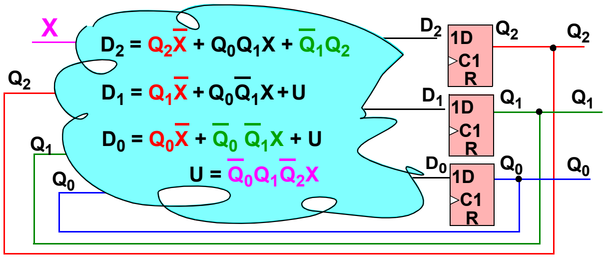
Transfering Equations Into Next-State Logic

0-6 Counter, Counts if X=1

Equations from last slide

$D_2 = Q_2\overline{X} + Q_0Q_1X + \overline{Q_1}Q_2$ $D_1 = Q_1\overline{X} + Q_0\overline{Q_1}X + \overline{Q_0}Q_1\overline{Q_2}X$ $D_0 = Q_0\overline{X} + \overline{Q_0}\,\overline{Q_1}X + \overline{Q_0}Q_1\overline{Q_2}X$

Where the logic equations fit in the state machine



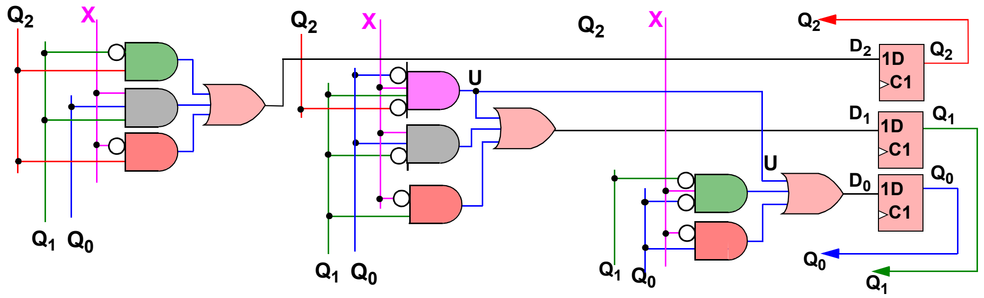
$$D_2 = Q_2\overline{X} + Q_0Q_1X + \overline{Q_1}Q_2$$

$$D_1 = Q_1\overline{X} + Q_0\overline{Q_1}X + U$$

$$D_0 = Q_0\overline{X} + \overline{Q_0}\,\overline{Q_1}X + U$$

$$U = \overline{Q_0}Q_1\overline{Q_2}X$$

Complete circuit



Maps The Final Design

Where the logic fits

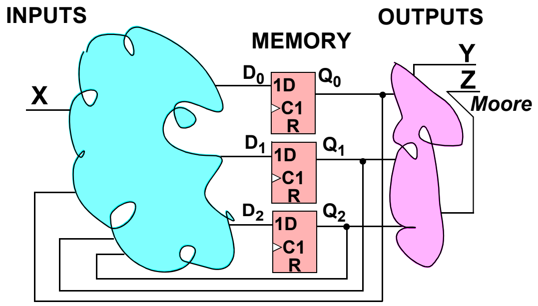
The logic transforms the state and the inputs into the D inputs that will give us the correct next state.

8-11.•PROBLEM

Design a counter which counts 0-3-5-7-2-0 when x=1 and 0-1-3-5-7-0 when x=0.
Obtain the logic equations, but you do not need to draw the circuit.

FSMs: Outputs; Moore and Mealy

Outputs



<u>Direct Outputs</u>

Like previous circuits

Outputs were directly from flip-flops

<u>Calculated Outputs</u>

There is logic between states and outputs.

Moore Outputs

Outputs calculated from state alone

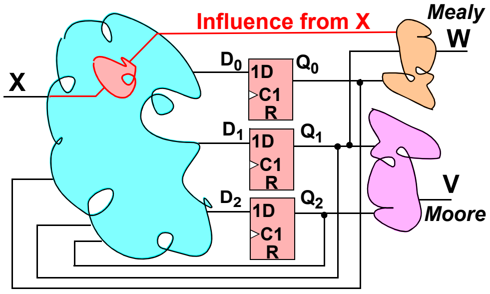
Input X can affect outputs only after passing through flip-flops.

Y, Z and V are Moore outputs

Mealy Outputs

Outputs can be directly influenced by inputs.

W is a Mealy output.



<u>Questions</u>

What is in the colourful blobs?

What is not in them?

Outputs

<u>Answers</u>

There are logic gates in each one. They may be as simple as a single wire, or as complex as you want.

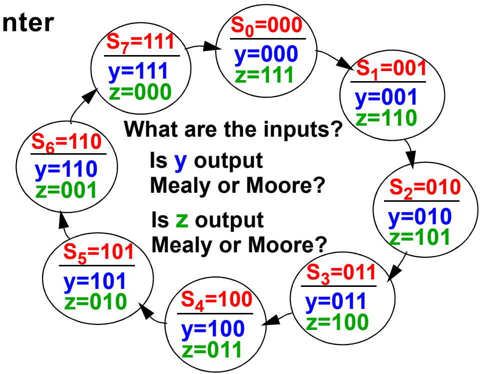
There is no storage in the blobs. There are no latches or flip-flops.

Outputs in State Graphs and Tables

Example: 0-7 Forwards and Backwards Counter

Moore Outputs

y output counts up
z output counts down



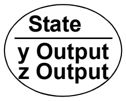
What are the inputs?

Is y output Mealy or Moore?

Is z output Mealy or Moore?

State Graph

Outputs written underneath state

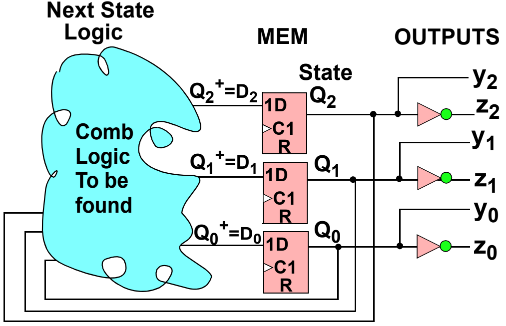


The State Table

Add columns for outputs

State Table,

State $Q_2Q_1Q_0$	Next State $Q_2^+Q_1^+Q_0^+$	Up Output $y_2y_1y_0$	Dn Output $z_2z_1z_0$
S0=000	001	000	111
S1=001	010	001	110
S3=011	100	011	100
S2=010	011	010	101
S4=100	101	100	011
S5=101	110	101	010
S7=111	000	111	000
S6=110	111	110	001



Outputs in State Graphs and Tables

Moore Outputs

Moore outputs in state tables

The outputs are placed in extra columns in the table

Combinational logic to calculate outputs

The outputs need more combinational logic.

The inputs are the states.
Normally one would need another Karnaugh map for each output bit.
Fortunately here the outputs are so simple one can do the logic in ones head.

8-12.• PROBLEM (COUNTER WITH TWO OUTPUTS)

a) draw the state graph of a counter that counts in binary:
0 ⇒ 1 ⇒ 2 ⇒ 3 ⇒ 4 ⇒ 5 ⇒ 6 ⇒ 7 ⇒ 0
It has three outputs bits XYZ. These are outputs, not the state bits.

When the x input = 1, the **outputs** are the same as the state, i.e. XYZ=000 for state 0, 001 for 1, 010 for 2, etc.
When x=0, the **outputs** are the state less one, i.e. state 2 gives out 1, state 1 gives out 0, state 0 gives out 7, etc.

b) Make a state table that includes a column for both outputs.

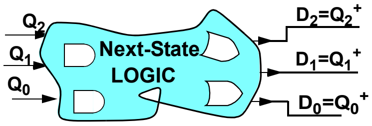
c) Get the equations for the next-state logic. You can find them in the notes on p. ____ (You tell me).

d) Derive the equations for the output logic.
(Check: F= $\overline{A}\cdot\overline{x}$ + Ax)

Answers: Y and Z are both Moore outputs.

Next State Logic Maps (Moore Outputs Circuit)

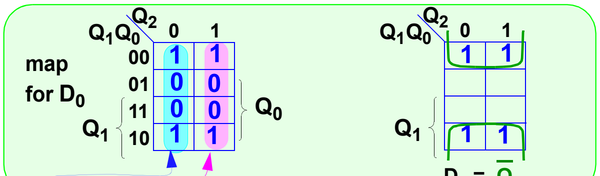
Design this logic



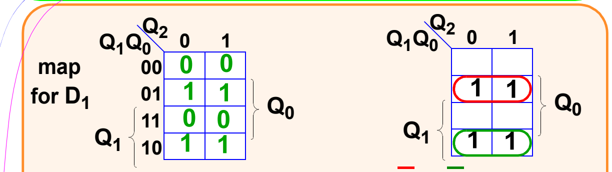
Ignore output logic for now

Counter state table

State $Q_2\ Q_1\ Q_0$	Next State $Q_2^+Q_1^+Q_0^+$	D values $D_2\ D_1\ D_0$
S0=0 0 0	001	0 0 1
S1=0 0 1	010	0 1 0
S3=0 1 1	100	1 0 0
S2=0 1 0	011	0 1 1
S4=1 0 0	101	1 0 1
S5=1 0 1	110	1 1 0
S7=1 1 1	000	0 0 0
S6=1 1 0	111	1 1 1

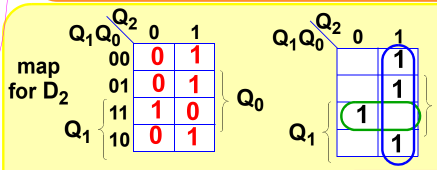


$D_0 = \overline{Q_0}$



$D_1 = Q_0\overline{Q_1} + \overline{Q_0}Q_1 = Q_0 \oplus Q_1$

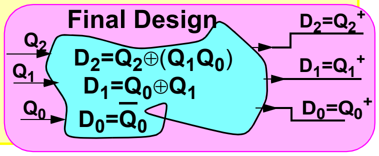
$x\bar{z}+z\bar{x}=x\oplus z$



$D_2 = Q_2 \oplus (Q_1Q_0)$

Extended Disjunctive Theorem

Remember one can loop "0"s if:
- They are contained in an even number of loops.
- One uses ⊕ instead of +



Derivation of Counter Next-State Logic Equations

- y output is the same as the state. We don't need any more logic.
- z output is the inverse of y. We don't need any more logic, just inverters.
- We only calculate the state logic,
  which are the same as the D outputs from the logic.

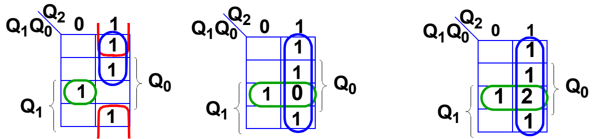
Summary of logic equation derivation

We calculate a 3-output 3-input block of combinational output logic.

We make a Karnaugh map for each combinational output.

We loop the maps to minimize the logic.

Reduction of D2 Using XK-Maps



map for D_2

Extend loops over 0

Parity (number of loops)
Odd for 1s
Even for 0s

$D_2 = Q_2 \oplus (Q_1Q_0)$

Extended Disjunctive Theorem

Reduction of D2 Using Algebra

$$
\begin{aligned}
D_2 &= Q_2\overline{Q_1} + Q_2\overline{Q_0} + Q_1Q_0\overline{Q_2} \\
&= Q_2(\overline{Q_1}+\overline{Q_0}) + (Q_1Q_0)\overline{Q_2} \\
&= Q_2(\overline{Q_1Q_0}) + (Q_1Q_0)\overline{Q_2} \\
&= Q_2 \oplus (Q_1Q_0)
\end{aligned}
$$

Distrb law

deMorgan

$x\bar{z}+z\bar{x}=x\oplus z$

FSMs: Mealy Outputs in State Graphs and Tables

Example: 0-7 Forwards and Backwards Counter

Mealy Output

If b=0, z output counts up
If b=1, z output counts down

State Graph

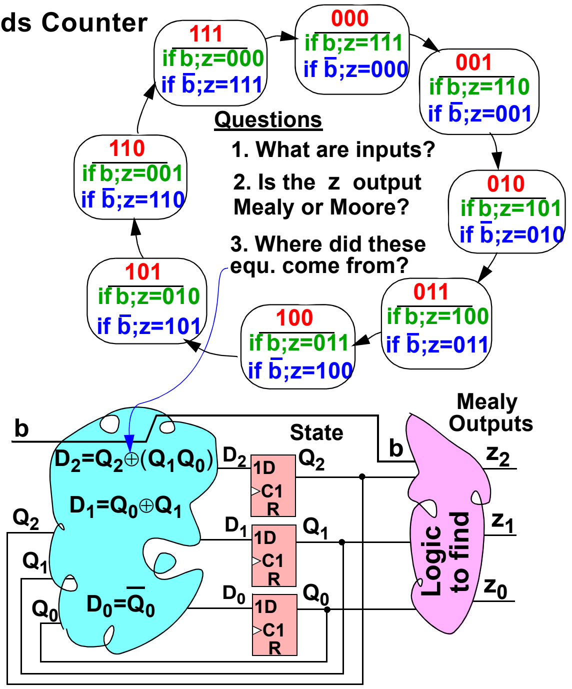
Outputs written underneath state
With associated input

The State Table

Add columns for outputs
A column for each input

State Table,

State $Q_2Q_1Q_0$	Next State $Q_2^+Q_1^+Q_0^+$	Output $z_2z_1z_0$ b=0	 b=1
S0=000	001	000	111
S1=001	010	001	110
S3=011	100	011	100
S2=010	010	010	101
S4=100	101	100	011
S5=101	110	101	010
S7=111	000	111	000
S6=110	111	110	001



Questions

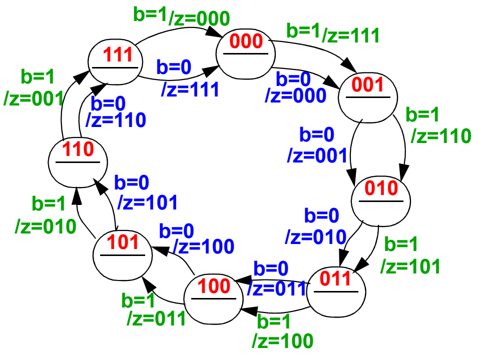
1. What are inputs?
2. Is the z output Mealy or Moore?
3. Where did these equ. come from?

Mealy Outputs

Representing Mealy outputs on state graphs

Many, in fact most, books use Mealy outputs attached to the state transition arrows.

You may use this notation if you find it easier.



Answers

1. There is only one input b. The CLK and RESET are inputs but we do not need to do any special logic design for them so they are not shown.

2. The Z output is Mealy since it can be directly controlled by an input.

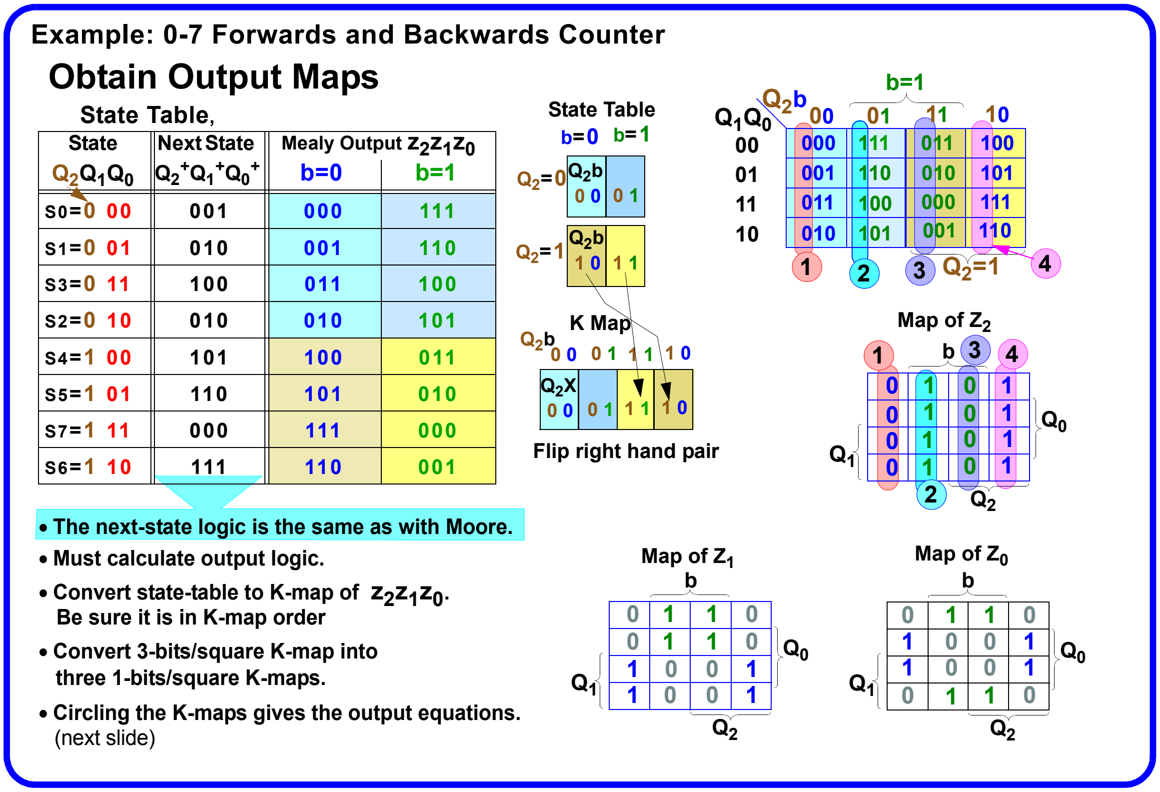
3. The state graph has the same states and state transitions (arrows) as for the circuit with Mealy outputs, only the outputs differ. Hence the next state equations are the same as for the circuit with Moore outputs.

Up-Down Counters vs Forwards and Backwards counters

This forwards and backwards counter change its count from X → 2^3-X-1 when one changes from counting up to down, i.e. 0→2^3-0-1=7, 1→2^3-1-1=6, 2→2^3-2-1=5 . . .

Most applications reverse the present count so that that 7→6, 6→5, . . . 0→7 when **b** is changed from 1 to 0. We will call these up-down counters. Their circuit is much more complex.

Mealy Outputs K-Maps and Circuits



Example: 0-7 Forwards and Backwards Counter

Obtain Output Maps

State Table,

State $Q_2Q_1Q_0$	Next State $Q_2^+Q_1^+Q_0^+$	Mealy Output $z_2z_1z_0$ b=0	b=1
s0=0 00	001	000	111
s1=0 01	010	001	110
s3=0 11	100	011	100
s2=0 10	010	010	101
s4=1 00	101	100	011
s5=1 01	110	101	010
s7=1 11	000	111	000
s6=1 10	111	110	001

- The next-state logic is the same as with Moore.
- Must calculate output logic.
- Convert state-table to K-map of $z_2z_1z_0$.
  Be sure it is in K-map order
- Convert 3-bits/square K-map into
  three 1-bits/square K-maps.
- Circling the K-maps gives the output equations.
  (next slide)

State Table

Flip right hand pair

Q_1Q_0 \ Q_2b	00	01	11	10
00	000	111	011	100
01	001	110	010	101
11	011	100	000	111
10	010	101	001	110

Map of Z_2

0	1	0	1
0	1	0	1
0	1	0	1
0	1	0	1

Map of Z_1

0	1	1	0
0	1	1	0
1	0	0	1
1	0	0	1

Map of Z_0

0	1	1	0
1	0	0	1
1	0	0	1
0	1	1	0

Mealy Outputs

Constructing the 4-input map

We have 3 state variables, and one input which, because it is Mealy, is an input variable for the output maps. Thus we must construct a 4-variable map from the output state table.

The easy way to do this:
-Move the four half-columns of 3-bit numbers from the state table to a Karnaugh map containing all the outputs.

- When moving the bottom to state table columns to the right hand side of the K-map, be sure to interchange the position of the columns for Q_2X=10 and Q_2X=11. Note their right-left position flips between the state table and the K-map.

After the map containing all three outputs is constructed, it should be split into three individual maps.

8-13.• PROBLEM (HARD)

Take a state graph in which there are 4-state bits and an input. Show, how the state table columns should be broken into blocks of 4 states, and exactly how these should be moved into a 5-input K-map.

Circuit For Mealy Forwards and Backwards Counter

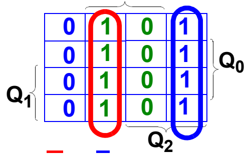
Example: 0-7 Forwards and Backwards Counter

Next State

State Table,

State $Q_2Q_1Q_0$	Next State $Q_2^+Q_1^+Q_0^+$	Mealy Output $Z_2Z_1Z_0$ b=0	b=1
S0=000	001	000	111
S1=001	010	001	110
S3=011	100	011	100
S2=010	100	010	101
S4=100	101	100	011
S5=101	110	101	010
S7=111	000	111	000
S6=110	111	110	001

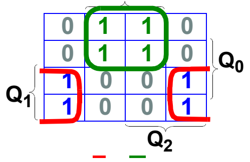
Map of Z_2



0	1	0	1
0	1	0	1
0	1	0	1
0	1	0	1

$Z_2 = b\overline{Q}_2 + \overline{b}Q_2 = b \oplus Q_2$

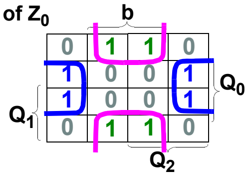
Map of Z_1



0	1	1	0
0	1	1	0
1	0	0	1
1	0	0	1

$Z_1 = Q_1\overline{b} + \overline{Q}_1 b = Q_1 \oplus b$

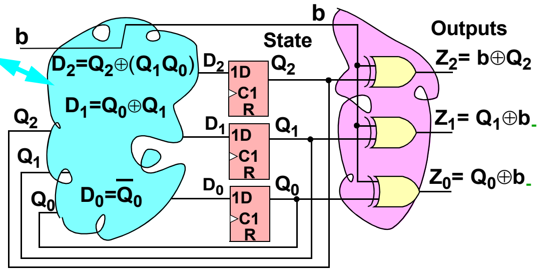
Map of Z_0



0	1	1	0
1	0	0	1
1	0	0	1
0	1	1	0

$Z_0 = Q_0\overline{b} + \overline{Q}_0 b = Q_0 \oplus b$

- The next-state logic is the same done for Moore.
- Must calculate output logic.
- Circling the K-maps gives the output equations.
- These give the output logic.



Mealy Output Circuit

Student's often ask about using XOR gates in schematics.

If the object is to get the smallest size CMOS circuit then XOR gates will not allow you to get a good measure of the circuit cost.

If the object is to get a compact set of equations to show other people, then they are quite useful. The object here was to fit the circuit on one slide without too much clutter.

If you are going to implement them on a complex-programmable-logic device (CPLD) as is done in the lab, it does not matter because the software that translates them into a program for the PLAs will take them out anyway.

For some logic families, like MOS-current-mode-logic (MCML), which are used at very high speeds, an XOR gate can be implemented easily, and may be the best choice of gate.

Systematic Design of Finite State Machines

Specification

Step 1

Decide on inputs and outputs

Draw a block diagram

Draw typical output waveforms

Step 2

Moore outputs or Mealy?

How many states?

Construct the state graph

Thinking Part

Step 3

Do state assignment

Step 4

Construct state table

Put table in K-Map order

Step 5

Make maps from table

- Next-state logic maps
- Output logic maps

Step 6

Derive quations

- Check for gate sharing

Step 7

Draw circuit

Turn the Crank Part

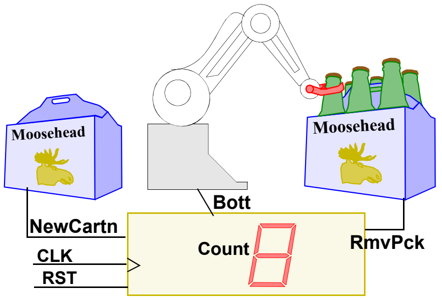
Systematic Design

Up to the State Graph, The Hard Part!

Basic Steps (Guidelines)

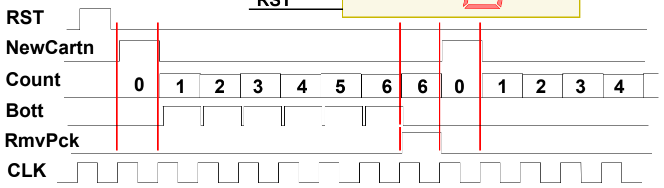
Specification

Design a machine to count bottles placed in a 6-pack.
1) It sends a signal to position the empty pack.
2-7) It sends a signal to place each bottle one at a time.
- It counts the bottles.
8) It sends a signal to removes the full pack.
Each step takes 0.5 sec.



Step 1

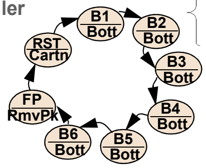
Decide on Inputs and Outputs
User interface
Draw a block diagram
Draw typical output waveforms



Step 2

Moore Outputs or Mealy?
These critera decribed on Slide 38
Mealy: Fast response?
Mealy: Fewer states => smaller
Moore: Glitch free outputs?

An extra clock cycle delay is OK.
Smaller size lowers cost.
Glitches are complicated; say it's OK for now.



How many states?

Construct the State Graph
using suggestive symbols

State graph construction is hard.
This chapter and the next describe it!

Steps In Finite-State Machine Design

Specification

This may be a very complete and detailed document. However it is often, short, incomplete and may have contradictions. You may have to revise it several times, and go back to the customer to get them to approve a revised specification.

Step 1

Input and Outputs

One needs their logical function, their duration, their timing with respect to the clock edge, their voltage levels. For the relatively simple circuits in this section, we will show the signals on a block diagram and on the waveforms of a timing diagram. For the moment we will require inputs to overlap at least one active clock edge so the flip flops can capture their values.

This overlaps with Step 2, because to draw proper outputs on the timing diagram, one needs to know if the are Moore or Mealy.

Step 2

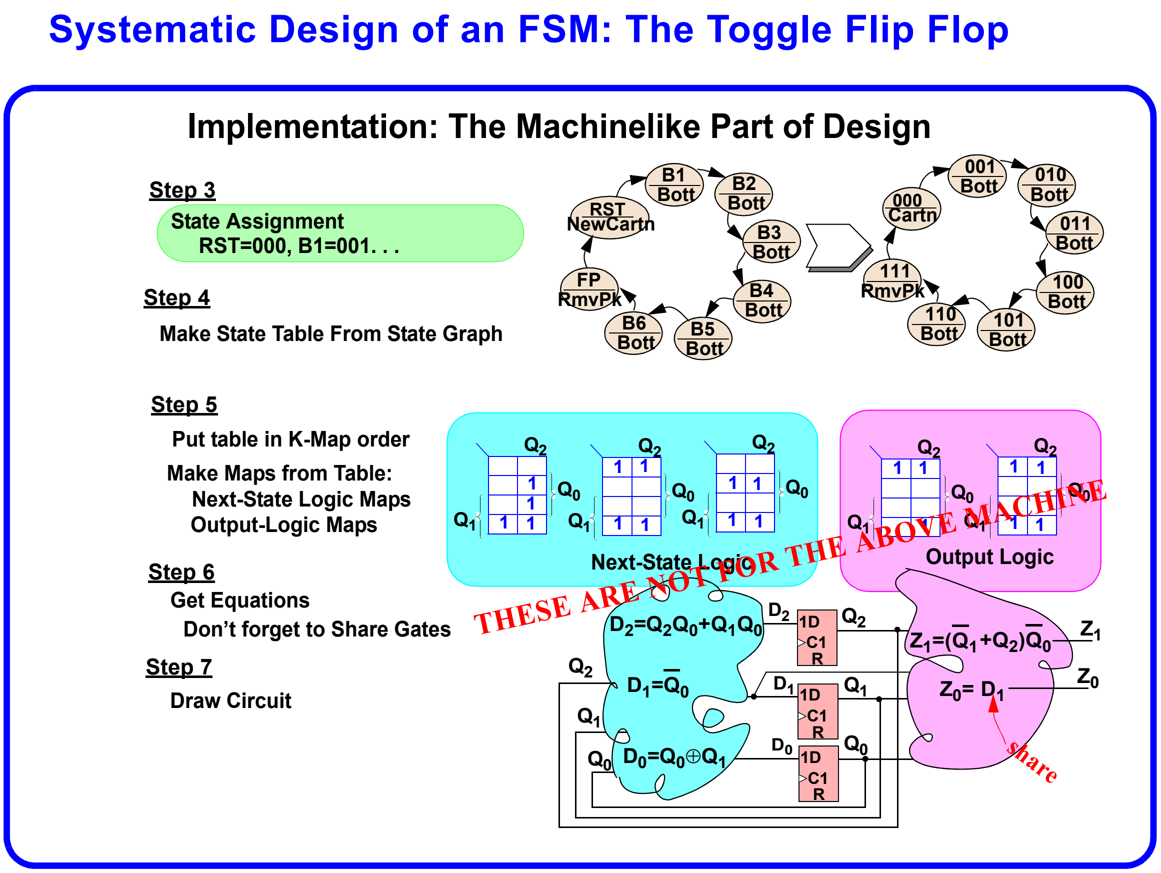
Mealy or Moore

More about this choice is given on Slide 38. The Mealy outputs are faster because a signal can cause an output to change immediately, rather than waiting for the clock. Also Mealy outputs tend to give a simpler machine.

State Graph

The flow between states is the part of the design which requires the most thinking, and is the main subject of this Section. The states will eventually have binary values, like 1001, 1101, ... However it is usually easier at this stage to give them suggestive names, like B1 (1st bottle), RST and FP(full pack). Later they will be converted to binary strings.

The number of states becomes apparent as the state graph design proceeds.



Systematic Design of an FSM: The Toggle Flip Flop

Implementation: The Machinelike Part of Design

Step 3

State Assignment
RST=000, B1=001. . .

Step 4

Make State Table From State Graph

Step 5

Put table in K-Map order

Make Maps from Table:
Next-State Logic Maps
Output-Logic Maps

Step 6

Get Equations
Don’t forget to Share Gates

Step 7

Draw Circuit

THESE ARE NOT FOR THE ABOVE MACHINE

Steps (Guidelines) In Finite-State Machine Design (cont)

The more mechanical steps

These are very algorithmic. The designer must know what is happening, but the routine boring work can be done by a computer program.

Step3.

Previously binary strings were used for states. If symbolic names are used like “Clear”, one must translate these into bit strings like “000”. This is called *state assignment,* further a good state assignment can make the logic noticeably simpler.

Step 4

Making a state table from the state graph,

Step 5

Arranging the state table in Karnaugh map order,
Extracting the maps from the table,

Step 6

Loop the maps, usually for minimum logic
Obtain the equations from looping the maps.
Note that there are one set of maps for the next-state equations and another set for the Mealy or Moore output equations.

Step 7

Draw the circuit

Systematic Design of an FSM: The Toggle Flip Flop

The "Toggle Flip-Flop"

Made from a D flip-flop.

Specification

- **Shall have two states S0 and S1**
- **Input signal T**
- **On every active clock edge:**
  If T=1 change output.
  If T=0 hold old output.

Step 1

- **Draw block diagram.**
- **Draw typical waveforms.**

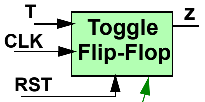
Step 2

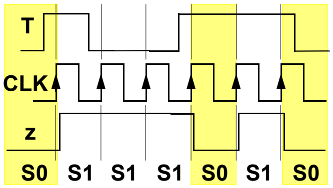
How many states?
Mealy or Moore?
Draw state graph

Choose Moore

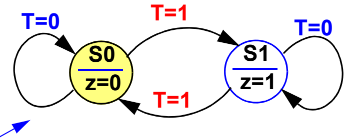
Step3

Make state table.





Don't make inputs change on clock edge



State Table (with symbolic states),

State	Next State Q^+		D input		output
Q	T=0	T=1	T=0	T=1	z
S0	S0	S1			0
S1	S1	S0			1

Design of a Toggle Flip-Flop

Step 1

Drawing the block diagram establishes in your mind what the inputs and outputs are.
An extra input, RST, not given in the specifications was added. This was because practically all flip-flops have them. However we will assume it was built into the underlying D flip-flop and does not have to be specifically designed here.

The waveforms help you to be sure you understand the specifications. Try to include all the scenarios such as two T=0 inputs when z starts at 1, two T-0 inputs when x starts at 0 (not included here).

Here the states were written at the bottom of the waveforms. Often, one can tell how many states one needs from the waveforms. Sometimes it is useful to put the states under the waveform as one draw the state graph.

Step 2

How many States?

This may be clear, or one may have to wait till the state graph has been done.

Choice of Mealy or Moore

There is no designer choice here. The specification says change after the clock edge, based on the input present just before and during the clock edge. This is a Moore output.

A Mealy output may change when an input changes without waiting for the clock.

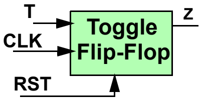
Step 3

The state table can be drawn from the state graph. However it can only be filled in as letters, S0 and S1. The table cannot be completed until state assignment is done on the next page.

FSM Design: Detect 101 in a Sequence of Bits

Example (Cont):

The "Toggle Flip-Flop"



State Assignment
S0=0
S1=1

Step 4

State assignment
Trivial here.
Later will be harder

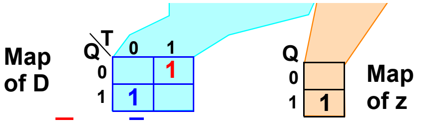
Complete state table
fill in assignment

State Table,

State	Next State Q^+		D input		output
Q	T=0	T=1	T=0	T=1	z
S0=0	S0=0	S1=1	0	1	0
S1=1	S1=1	S0=0	1	0	1

Step 5

Make K-maps from table



$D = \overline{Q}T + Q\overline{T} = Q \oplus T$

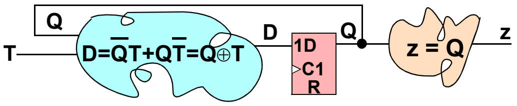
$z = Q$

Step 6

Get next state equations

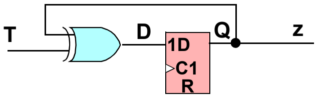
Watch for multiple output simplification.
(None here with only one map)

Get output logic equations.



Step7

Draw circuit



Design of a Toggle Flip-Flop (Cont)

Step 4

State assignment is the process of assigning a sequence of ones and zeros to each state.

Here one can only make S0=0 or S0=1. Making S0=0 makes the state match the output. This is convenient.

Since S0=0, this forces S1=1.

Step 5 and 6.

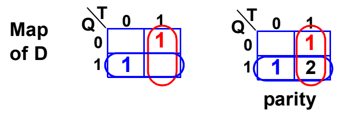
The K-maps are difficult only for over-simple K-maps like D = $\overline{Q}$.

However these are multiple output maps. Remember to look for shared gates

Step 7

Easy.

Finding XSofP



D = Q ⊕T

8-14.• PROBLEM

Draw the timing diagram (waveform) showing **T**, **clk** and **z** for a toggle flip-flop which has input T =101 000 111 010. These numbers are the value of the input just before the clock edge in each cycle.
Assume the flip-flop was reset to Q=0 at the start and that the left most bit of the sequence above is at time = 0.

FSM Design: A Moore 101 Sequence Detector

Complete Design of a Finite-State Machine

A “101” Sequence Detector

<u>Specification</u>

x=serial bit coming in at clock rate

z = 1 after sequence 101 is detected

No overlap

don’t count, overlap

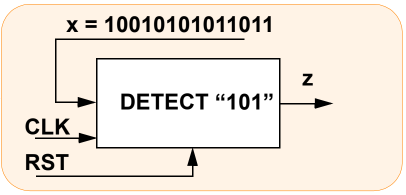
1010101

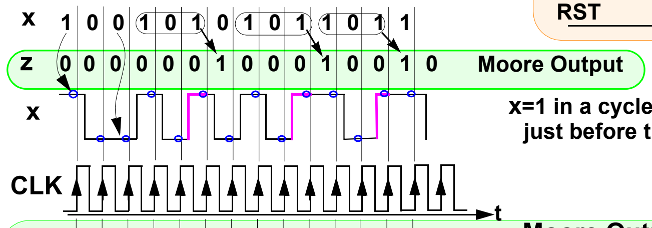
ok ok

<u>Step 1</u>

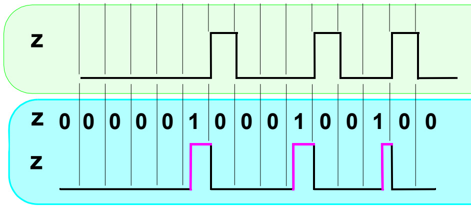
Draw block diagram.

Draw waveforms.





x=1 in a cycle, means x=1 just before the end-of-cycle clock edge



Moore Output (No overlap)

Output waits till the clock edge after 101 is complete.

Mealy Output (No overlap)

Output rises with the 2nd “1” in 101.

Goes down next clock edge.

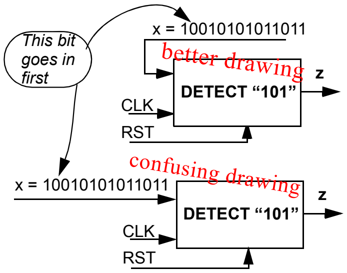
Design of a Moore “101” Machine

<u>No overlap</u>

In the specifications, no overlap means that one does not recognize sequences which overlap another sequence. Thus 10101 only has one nonoverlapping 101 sequence, but it has two if one counts overlaps.

<u>Block diagrams</u>

If input sequences are shown on a block diagram, and shown on a timing diagram, there is confusion about which bit goes in first. In the timing diagram the left most bit goes in first (at t=0). In the block diagram, this appears strange unless one bends the input lead



<u>Where does one change x</u>

The flip-flops sample the inputs just before the active clock edge. It does not matter where x changes as long as the value is correct just before that edge.

Note x must not change while the clock is changing since one would not know which value the flip-flop would eventually capture.

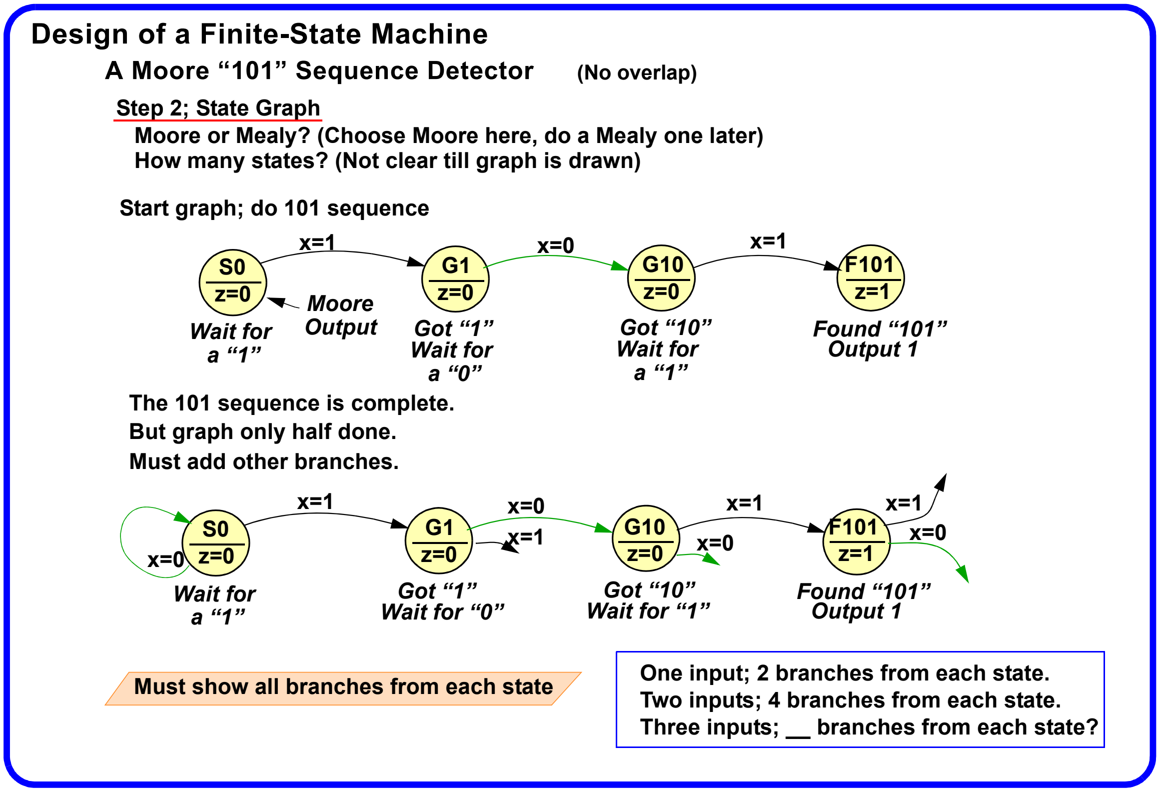
The circles on the x waveform above, indicate the value of x that is captured. That must agree with the numeric values shown above it, or on the block diagram.

<u>Delay in Moore Outputs</u>

Notice the <u>Mealy outputs end at the time the Moore outputs start.</u> The Moore outputs are delayed into the next cycle from the Mealy outputs.

If x changes late in the clock cycle, the Mealy output will be a short pulse (look at the 3rd Mealy pulse). The Moore output is always a full cycle.

FSM Design: The Moore 101 Sequence Detector



Design of a Finite-State Machine

A Moore “101” Sequence Detector (No overlap)

Step 2; State Graph

Moore or Mealy? (Choose Moore here, do a Mealy one later)
How many states? (Not clear till graph is drawn)

Start graph; do 101 sequence

The 101 sequence is complete.
But graph only half done.
Must add other branches.

Must show all branches from each state

One input; 2 branches from each state.
Two inputs; 4 branches from each state.
Three inputs; __ branches from each state?

Moore “101” Sequence Detector

Drawing the State Graph

Starting the graph

First set up a state sequence that will recognize the “101” sequence.

It is good to write a comment under each state, explaining what it means to be in that state.

Finishing the graph

Then go back and fill in the other branches on each state.

In this case, with one input, you must show two branches from each state.

Much of the work in developing the state graph is deciding where these other outputs go.

How many branches from each state?

For one input x, one has 2 next state choices, one for x=0, one for x=1. One must always decide where both branches go; be careful.

For two inputs, x and y, one has 4 next states choices, one for each of xy=00, xy=01, xy=11, xy =10.

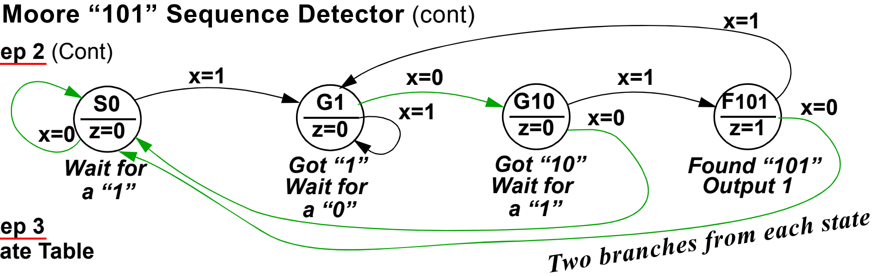
For three inputs, x y and z, one has 2^3 =8 next states choices. A 3-input state graph can becomes very messy.

FSM Design: The Moore 101 Sequence Detector

Design of a Finite-State Machine

A Moore “101” Sequence Detector (cont)

Step 2 (Cont)



Step 3
State Table

Symbolic State Table

State Q	Next State Q^+ x=0	x=1	output z
S0	S0	G1	0
G1	G10	G1	0
G10	S0	F101	0
F101	S0	G1	1

Step 4
State Assignment
Substitute bits for state names.
For now choose arbitrary bits
S0=00, G1=01, G10=10, F101=11

State Table; After State Assignment

State Q_1Q_0	Next State $Q_1^+Q_0^+$ x=0	x=1	D inputs D_1D_0 x=0	x=1	output z
S0=00	S0=00	G1=01	00	01	0
G1=01	G10=10	G1=01	10	01	0
G2=10	S0=00	F101=11	00	11	0
F101=11	S0=00	G1=01	00	01	1

A Moore “101” Machine (cont)

Completing the State Graph

Consider state G1

When in G1 (Got 1), one has had an first 1 of “101”. Suppose one get a second 1. Then the first 1 is discarded as not part of a “101” sequence, but the second 1 may be the start of another “101” sequence, hence stay in G1.

Consider state G10

When in G10, one knows one received “10”. However suppose one then gets a 2nd 0, for a sequence so far of “100”
This is not part of anything so one must go back to S0 and wait for a 1.

Building State Tables

First build a table with symbolic names like S0, G1, . . .(step 3)

Then do a state assignment (step 4).
This state assignment may be done:
- to minimize logic,
- to minimize the number of flip-flops that change between states.
  This tends to save logic and especially power.
- to minimize designer effort. This often means an arbitrary or random assignment.

FSM Design: The Moore 101 Sequence Detector

Design of a Finite-State Machine

A Moore "101" Sequence Detector (cont^2)

State Table in Karnaugh Map Order

State	Next State $Q_1^+Q_0^+$		D inputs D_1D_0		output
Q_1Q_0	x=0	x=1	x=0	x=1	z
S0=00	S0=00	G1=01	00	01	0
G1=01	G10=10	G1=01	10	01	0
F101=11	S0=00	G1=01	00	01	1
G10=10	S0=00	F101=11	00	11	0

Q_1Q_0=11

map for D_1

	Q_1Q_0 \ X	0	1	
	00	0	0	
	01	1	0	Q_0
Q_1	11	0	0	Q_0
Q_1	10	0	1	

$D_1 = \overline{X}\cdot\overline{Q_1}Q_0 + XQ_1\overline{Q_0}$

map for D_0

	Q_1Q_0 \ X	0	1	
	00	0	1	
	01	0	1	Q_0
Q_1	11	0	1	Q_0
Q_1	10	0	1	

$D_0 = X$

Step 5
Make K-Map from table

Step 6
Get logic equations

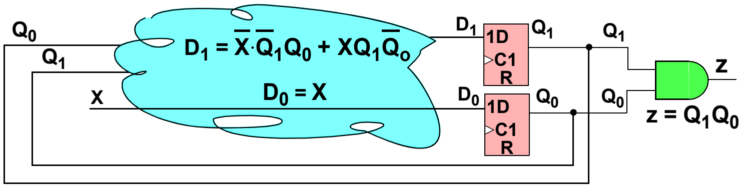
Step 7
Draw circuit

Q_1Q_0 \ X	0	1
00	00	01
01	10	01
11	00	01
10	00	11

map for D_1D_0

Q_1 \ Q_0	0	1
0	0	0
1	0	1

map for z
$z = Q_1Q_0$

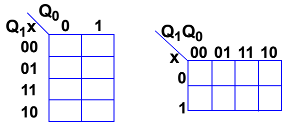


A Moore "101" Machine (cont^2)

Don't forget to arrange your state table in K-map order

If you don't it is much harder to transfer your results to a Karnaugh map.

Also arrange your K-maps so the variables axes follow the state tables. For example, *Poor Map 1* would be very hard to use, and even *Poor Map 2* would be a little confusing since the data would be rotated 90°.



Poor Map 1 Poor Map 2
Inputs and states
mixed on vertical axis

Step 7: Draw the circuit

By this time you should be an expert at drawing circuits from logic equations.
From now on, we will tend to draw circuits with blobs with equations inside unless, in certain case like the output above, which only has one gates. Then it is clearer to draw the gate.

8-15.•PROBLEM

Draw the state graph, and make the state table for with a Moore output which is 1 when the input sequence is 1101 or 1001.
Do not recognize overlap.
This machine needs 5 states.
Do not make design the circuit.

8-16.•PROBLEM

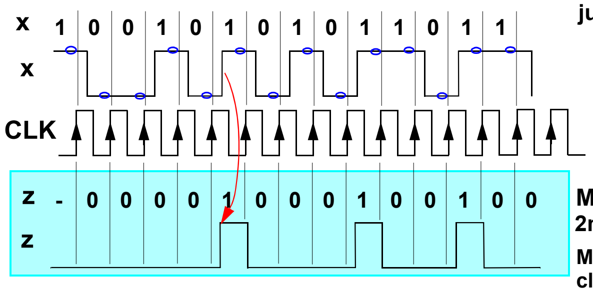
Draw the state graph and make the state table for a two Moore output machine the first output is one for the sequence 1101, and the second for 1001. Do not recognize overlap.

FSM Design: The Mealy 101 Sequence Detector

Design of a Mealy Finite-State Machine

A Mealy “101” Sequence Detector

Step 1: See Slide 27



x=1 in a cycle means x must be 1 just before the clock edge.

(No overlap)

Mealy Output rises with the 2nd “1” in 101, as soon as x=1.

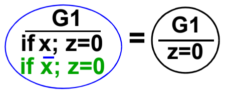
Mealy Output goes down at the next clock edge. It was not specified to keep it high for the next cycle

Step 2: State Graph

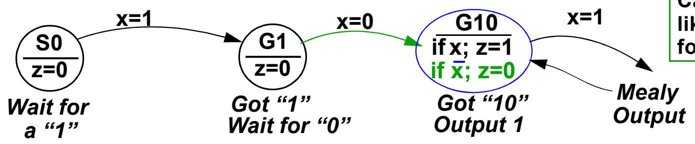
Moore or Mealy? (Choose Mealy)

Start graph; do 101 sequence

Now the output is z=1 as soon as the 2nd x=1 appears



Can write Mealy output like a Moore output for states like G1 and S0



A Mealy “101” Sequence Detector

Mealy outputs change before Moore outputs

Mealy outputs can be directly controlled by inputs and hence do not have to wait for the clock.

Mealy State Graph

Notice there were four states for the graph with Moore outputs but only three here. The direct input-to-output connection often saves having to remember some information, and thus saves states.

Mealy Outputs on State Graphs

Moore outputs are traditionally written in the bottom of the circles

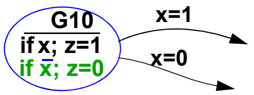
Here we show three ways of writing Mealy outputs,

The 2nd one is simple, when Z is a simple function of X.

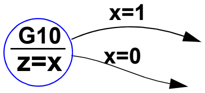
The 3rd one the one most often used in books.
Different inputs usually take the machine to different states. Also, for Mealy outputs, different inputs give different outputs.
Thus one write the output(z=1) for x=1, over the arrow that shows the state change for x=1.

Unfortunately, one must remember one is still in state G10 when considering the output. This is clearer in 1) and 2).

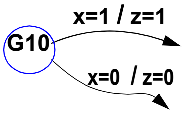
1)



2)



3)



FSM Design: The Mealy 101 Sequence Detector

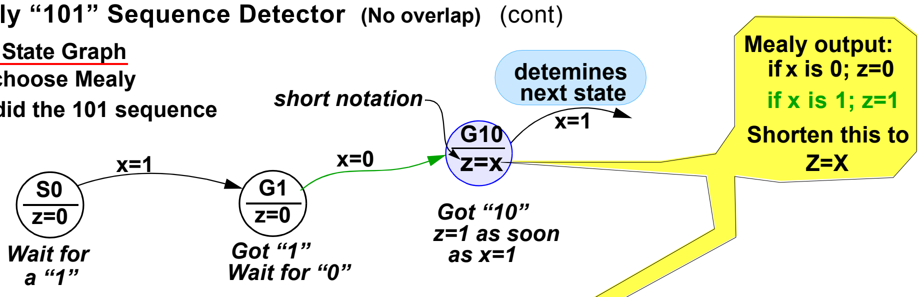
Design of a Mealy Finite-State Machine

A Mealy “101” Sequence Detector (No overlap) (cont)

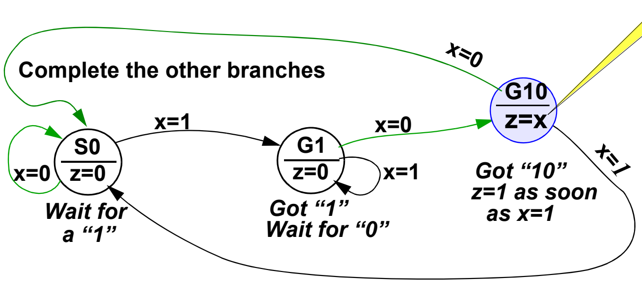
Step 2; State Graph

We choose Mealy

We did the 101 sequence



Complete the other branches



Step 3; State Table

Symbolic State Table

State Q	Next State Q^+ x=0	Next State Q^+ x=1	output x=0	output x=1
S0	S0	G1	0	0
G1	G10	G1	0	0
G10	S0	S0	0	1
Su				

Su is an unused state

A Mealy “101” Sequence Detector (cont)

Completing the Sate Graph

After completing the path followed by the desired “101” input, decide what to do with the other possible inputs.
Each state has two cases: x=0 and x=1.
Be sure there is an arrow for each case.
Any “1” input may be the start of another “101” sequence. However, without overlap, the final “1” in “101” does not start a new sequence.

The State Table

The system has only 3 states
Thus there are don’t cares in the state table.

8-17.•PROBLEM

Design a the state graph for a Mealy machine which puts out a 1 when it sees the final 1 at the end of the input sequence 1011.
The left most bit comes in first.
Do not recognize overlap.

FSM Design: The Mealy 101 Sequence Detector

Design of a Mealy Finite-State Machine

A Mealy “101” Sequence Detector (cont)

Step 4

State Assignment

- Substitute bits for state names.
  For now choose:
  S0=00, G1=01, G10=10, Su=11
- Make table with assignment.
  Put table in K-map order.

State Table; After State Assignment

State	Next State $Q_1^+Q_0^+$		D inputs D_1D_0		output	
Q_1Q_0	x=0	x=1	x=0	x=1	x=0	x=1
S0=00	S0=00	G1=01	00	01	0	0
G1=01	G10=10	G1=01	10	01	0	0
Su=11			*dd*	*dd*	*d*	*d*
G10=10	S0=00	S0=00	00	00	0	1

Step 5

Make K-Maps.

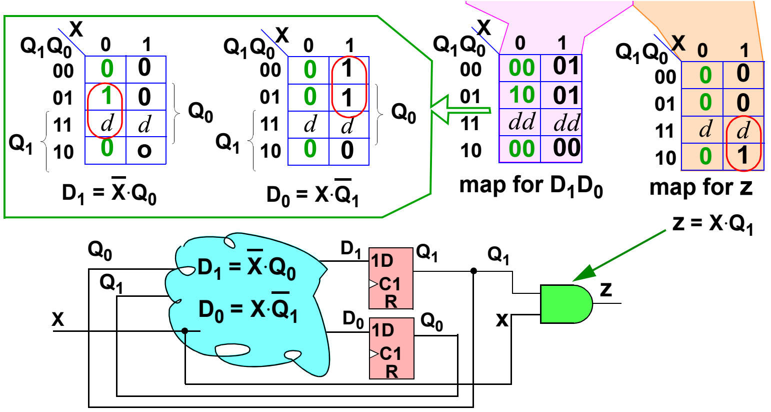
- Make output map
  z map
- Make D_1D_0 map
  Separate D_1D_0 map
  into D_1
  and D_0 maps

Step 6

Get logic equations
from maps

Step 7

Draw circuit



Q_1Q_0 \ X	0	1
00	0	0
01	1	0
11	*d*	*d*
10	0	o

$D_1 = \overline{X}\cdot Q_0$

Q_1Q_0 \ X	0	1
00	0	1
01	0	1
11	*d*	*d*
10	0	0

$D_0 = X\cdot\overline{Q_1}$

Q_1Q_0 \ X	0	1
00	00	01
01	10	01
11	*dd*	*dd*
10	00	00

map for D_1D_0

Q_1Q_0 \ X	0	1
00	0	0
01	0	0
11	*d*	*d*
10	0	1

map for z

$z = X\cdot Q_1$

$D_1 = \overline{X}\cdot Q_0$

$D_0 = X\cdot\overline{Q_1}$

A Mealy “101” Sequence Detector (cont^2)

Don’t care states

Once the machine is designed and the logic equations known, state Su no longer has unknown next states. They can calculated from the equations or read off the Karnaugh maps.

8-18.•PROBLEM

Complete the next state and output columns for state Su for the completed circuit, using the values the gates coosen will give.

FOR PROBLEM **8-18.•**

State	Next State $Q_1^+Q_0^+$		output	
Q_1Q_0	x=0	x=1	x=0	x=1
S0=00	S0=00	G1=01	0	0
G1=01	G10=10	G1=01	0	0
Su=11				
G10=10	S0=00	S0=00	0	1

State assignment and circuit size

The choice of state assignment can make a difference in the complexity of the next-state logic.

8-19.•PROBLEM

Complete the nonoverlapping Mealy “101” machine with state assignment:

S0=00, G1=01, G10=11, Su=10. Su is a state that is not used.

When you loop the maps, remember to share loops between the three maps (D_1, D_0, and z), whenever it will reduce the total logic.

8-20.•PROBLEM

Design a the state graph for a Mealy machine which puts out a 1 when it sees the final 1 at the end of the input sequence 111[1]01. Here [1] means a sequence of 1s of any length including zero.
The left most bit comes in first.
Do not recognize overlap. The answer is given at the end.

8-21.•PROBLEM

Do Problem 8-20.• recognizing overlap.

Mealy and Moore Revisited

Mealy vs. Moore

Output Response Speed

Mealy can respond directly to inputs without waiting for CLK

Moore response is delayed until CLK edge

Input Glitches

Mealy passes input glitches to output
Moore filters input glitches.

Often glitches are not a problem.

Short Output Pulses

Mealy outputs can be very short.
Moore are always a clock cycle.

Complexity

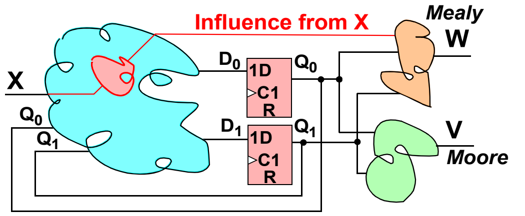
Mealy usually has fewer states.

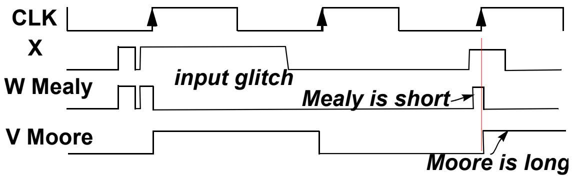
Fewer states means simpler logic, usually.

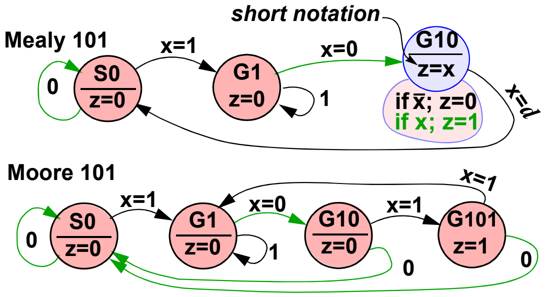
Question

Why does Moore not respond to glitches?

What happens to the logic size if one goes from 9 states to 8 states?







Mealy vs Moore

Output Response Speed

Moore outputs have to go through flip-flops. One must wait for the active clock edge before the flip-flop can change. Therefore Moore outputs must wait for the clock

Glitch Transfer

Moore outputs have no direct connection to the inputs, they must go through a flip-flop. The only time a glitch can affect a flip-flop is if it happens right on the clock edge.

A glitch on the clock edge will be captured either as a “1” or a “0”, but it will not come through as a glitch.

Moreover, standard design methods keep glitches away from the clock edge

Short Output Pulses

Moore outputs always last a clock cycle. Mealy outputs depend on the input pulse position.

Complexity

In general a Mealy design will be smaller than a Moore design. One reason is if one can go from 2^N+1 states down to 2^N states, one can save a flip-flop. Since one can double the number of states by adding just one flip-flop, going Mealy instead of Moore virtually never saves more than one flip-flop.

However, for each state saved, one reduces the amount of next-state calculation logic needed. This can be considerable, since it is proportional to the number of states saved.

Summary

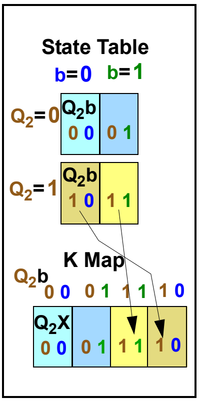
In synchronous circuits with flip-flop storage, glitches are usually not a problem.

The reduced size and the increased speed are good enough reasons to do most designs as Mealy machines.

However in very large (million gate) circuits, better control of delays between machines can be had with Moore outputs, and they are often used for outputs between modules.

“Must Remembers” in FSM Design

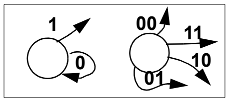
a. Rearrange the state table into Karnaugh map order.
Be careful with 4 column maps



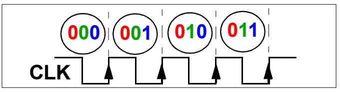
b. Choose Moore or Mealy outputs from the description.
Mealy output problems have words like:
“The output becomes true as soon as the final bit in the sequence starts”.
“The output does not have to wait for the clock.”
“Choose an output type most likely to reduce the size of the circuit.”

Moore output problems have words like:
“The output becomes true during the state after the final input bit is received.”
“The output must not transfer any glitches that are on the input.”

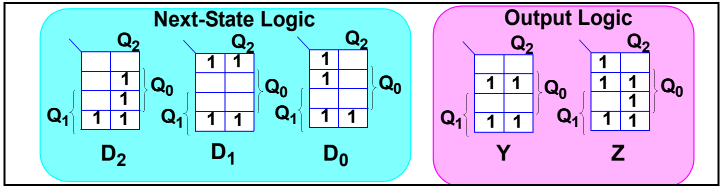
c. Each state has 2 outgoing arrows for 1 input,
or 4 outgoing arrows for 2 inputs.



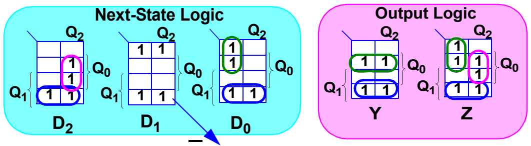
d. States only change at active clock edge.



e. FSMs are multiple output machines.
Watch for shared gates when you loop the maps.



By sharing gates, the logic can be minimized.



$D_2=W+Q_2Q_0$

$W=Q_1\overline{Q}_0$

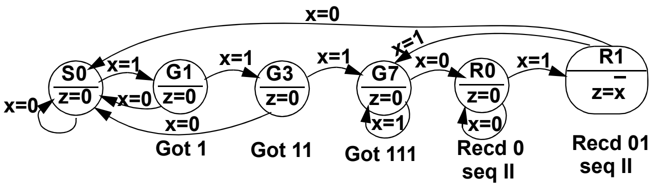
$D_1=\overline{Q}_0$ half map

$D_0=W+\overline{Q}_1\overline{Q}_2$

$Y=D_0$

$Z=D_2+D_0$

8-22.•SOLUTION TO PROBLEM 8-20.• PAGE 73 as a Mealy machine.



Design a the state graph for a Mealy machine which puts out a 1 when it sees the final 1 at the end of the input sequence 111[1]01. Here [1] means a sequence of 1s of any length including zero. The left most bit comes in first. Do not recognize overlap