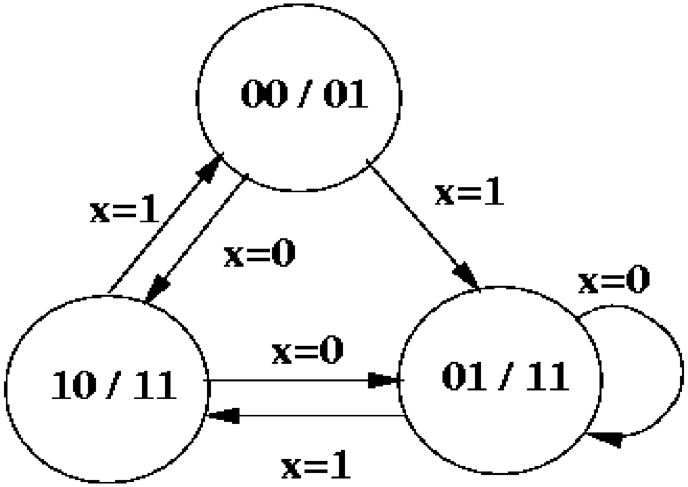
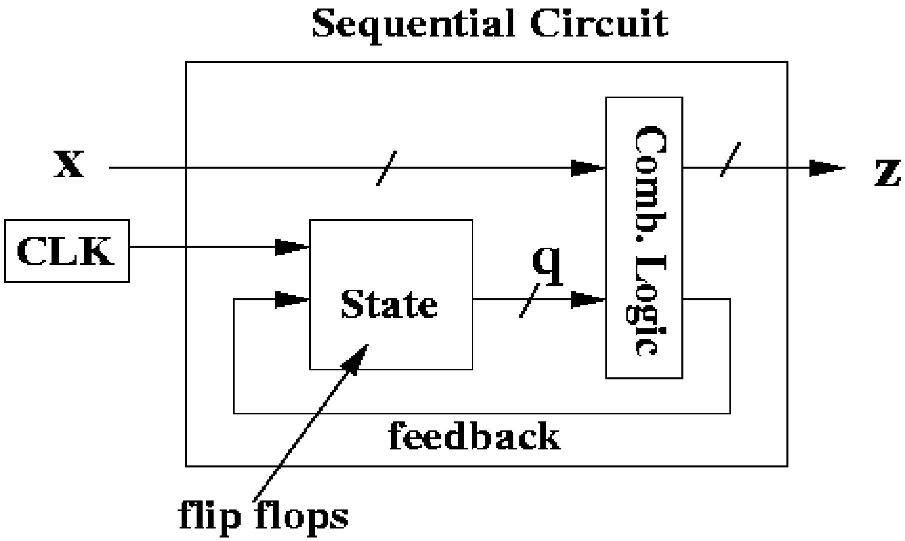
Finite state machines: implementing

Want to implement FSM in circuit:





Steps to do this:

1. Create a state transition table
2. Decide how many flip flops you need, and what kind
3. Use the flip flop excitation table to fill out the rest of the chart
4. Implement the circuit

Steps are easy

Hard part: understanding what is going on

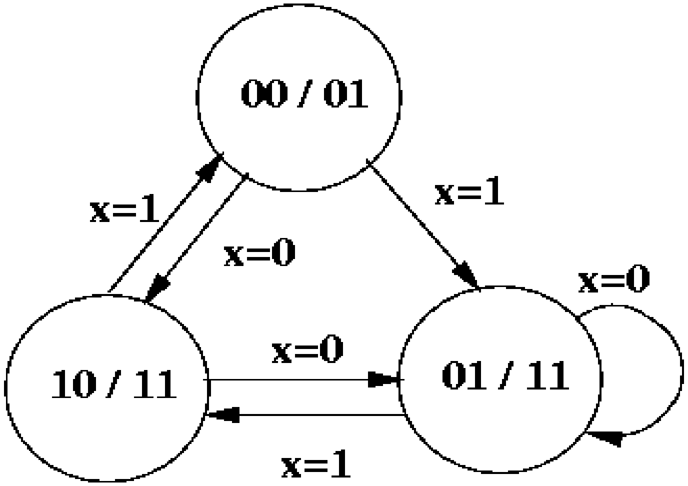
Finite state machines: implementing

State: 2 bits (q_1q_0)

2 flip-flops to store

Input: 1 bit (x)

Output: 2 bits (z_1z_0)



Step 1: State transition table

Start with state and input

Each row represents a state and a possible input value

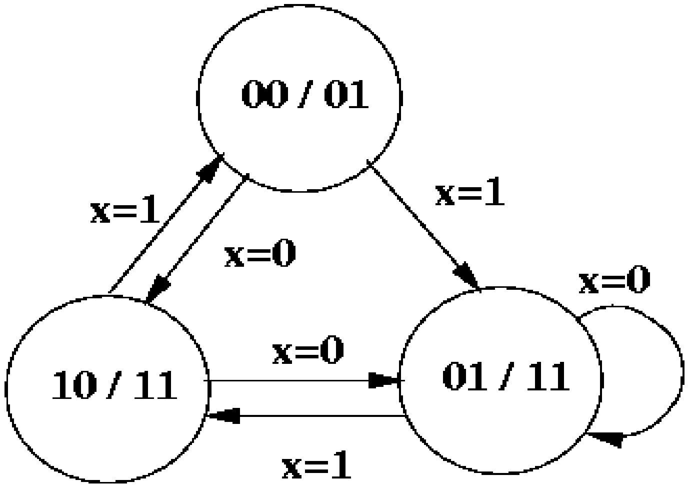
States before inputs:

Look at every possible input for each state

q_1	q_0	x
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Finite state machines: implementing

q_1	q_0	x	q_1^+	q_0^+
0	0	0	1	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	d	d
1	1	1	d	d



Step 1: State transition table

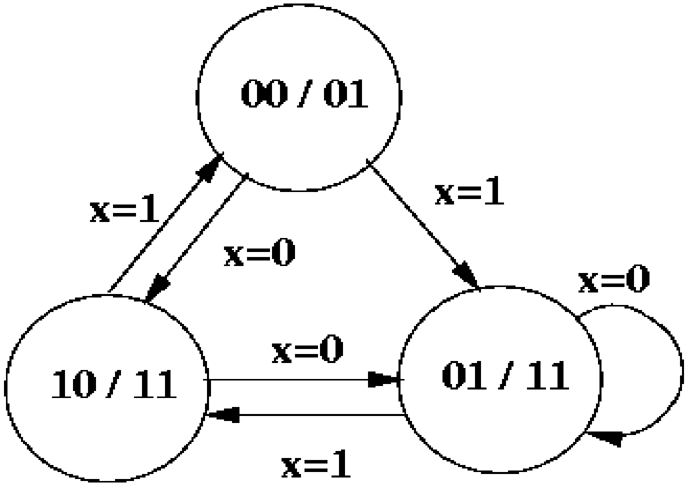
(a) Write next state

There is no state 11

Put d for "don't care" as next state

Finite state machines: implementing

q_1	q_0	x	q_1^+	q_0^+	z_1	z_0
0	0	0	1	0	0	1
0	0	1	0	1	0	1
0	1	0	0	1	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	1
1	0	1	0	0	1	1
1	1	0	d	d	d	d
1	1	1	d	d	d	d



Step 1: State transition table

(b) Write outputs

These are just a function of the state

State 11 doesn't exist: "don't care"

Finite state machines: implementing

Step 2: Pick flip-flops

What kind?

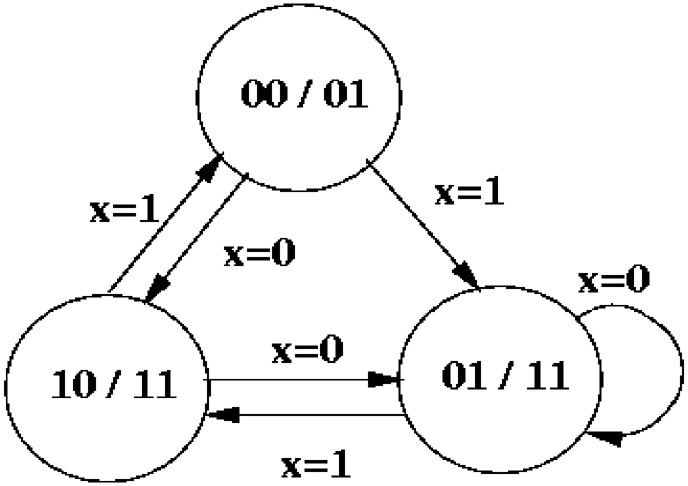
Some combination of D and/or T

Use D_1 to store q_1 and T_0 to store q_0

Step 3: Use the flip-flop excitation tables to determine D and T

What input for D generates an output of 1?

The entire column for D is a copy of q_1^+



q_1	q_0	x	q_1^+	q_0^+	z_1	z_0	D_1	T_0
0	0	0	1	0	0	1	1	
0	0	1	0	1	0	1	0	
0	1	0	0	1	1	1	0	
0	1	1	1	0	1	1	1	
1	0	0	0	1	1	1	0	
1	0	1	0	0	1	1	0	
1	1	0	d	d	d	d	d	
1	1	1	d	d	d	d	d	

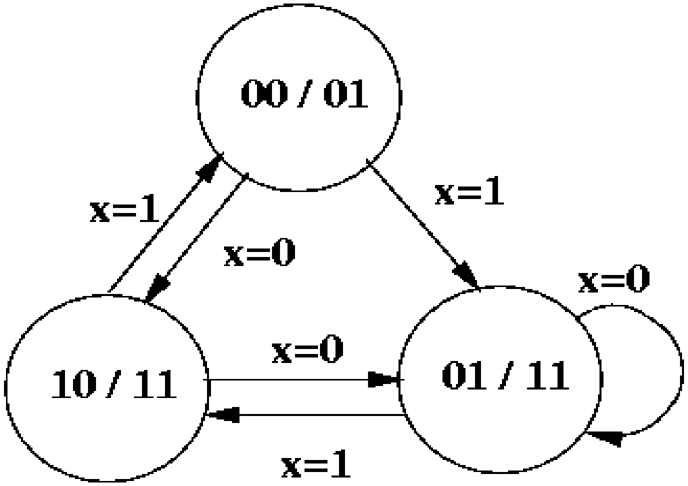
Finite state machines: implementing

Step 3: Use the flip-flop excitation tables to determine D and T

What value for T do we need to get state 0 from state 0?

What value for T do we need to get state 1 from state 0?

Fill in rest of column with hold ($q_0^+ = q_0$) or toggle ($q_0^+ \, != \, q_0$)



q_1	q_0	x	q_1^+	q_0^+	z_1	z_0	D_1	T_0
0	0	0	1	0	0	1	1	0
0	0	1	0	1	0	1	0	1
0	1	0	0	1	1	1	0	0
0	1	1	1	0	1	1	1	1
1	0	0	0	1	1	1	0	1
1	0	1	0	0	1	1	0	0
1	1	0	d	d	d	d	d	d
1	1	1	d	d	d	d	d	d

Finite state machines: implementing

Step 4: Implement a circuit to generate the control signals

One way to do this: use a ROM (read-only memory)

Input to ROM: k-bit address

Output from ROM: contents of location at given address

Number of bits needed for address: inputs to truth table

2 bits for state + 1 bit for input in this case

Number of bits at each location: number of output bits + number of flip-flops

4 bits in this case

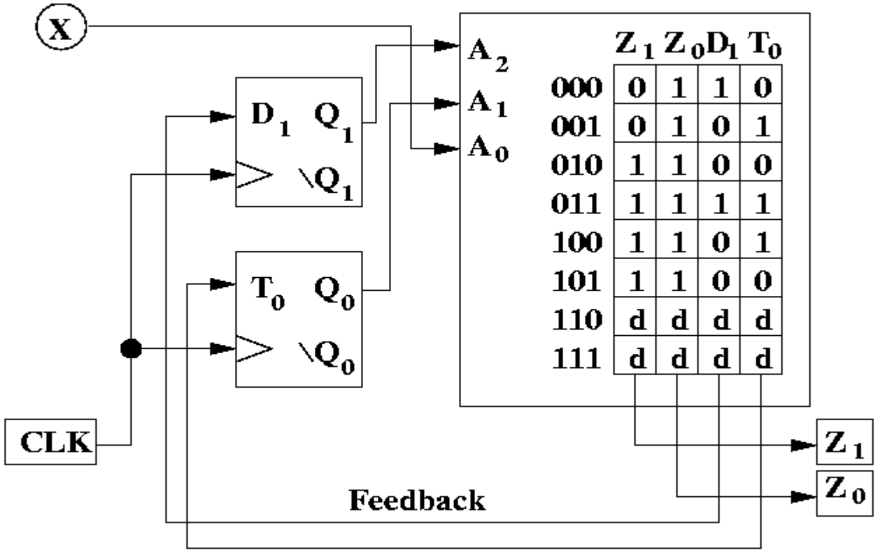
q_1	q_0	x	q_1^+	q_0^+	z_1	z_0	D_1	T_0
0	0	0	1	0	0	1	1	0
0	0	1	0	1	0	1	0	1
0	1	0	0	1	1	1	0	0
0	1	1	1	0	1	1	1	1
1	0	0	0	1	1	1	0	1
1	0	1	0	0	1	1	0	0
1	1	0	d	d	d	d	d	d
1	1	1	d	d	d	d	d	d

Finite state machines: implementing

Step 4: Implement a circuit to generate the control signals

Copy last 4 columns to ROM

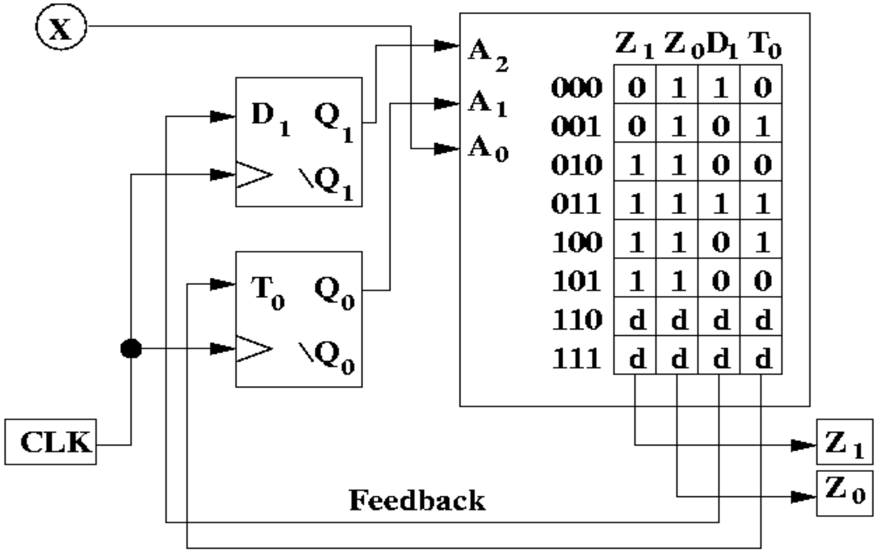
Connect inputs and outputs



	Z_1	Z_0	D_1	T_0
000	0	1	1	0
001	0	1	0	1
010	1	1	0	0
011	1	1	1	1
100	1	1	0	1
101	1	1	0	0
110	d	d	d	d
111	d	d	d	d

q_1	q_0	x	q_1^+	q_0^+	z_1	z_0	D_1	T_0
0	0	0	1	0	0	1	1	0
0	0	1	0	1	0	1	0	1
0	1	0	0	1	1	1	0	0
0	1	1	1	0	1	1	1	1
1	0	0	0	1	1	1	0	1
1	0	1	0	0	1	1	0	0
1	1	0	d	d	d	d	d	d
1	1	1	d	d	d	d	d	d

Finite state machines: implementing



	Z$_1$	Z$_0$	D$_1$	T$_0$
000	0	1	1	0
001	0	1	0	1
010	1	1	0	0
011	1	1	1	1
100	1	1	0	1
101	1	1	0	0
110	d	d	d	d
111	d	d	d	d

Notes:

Label the address bits: $A_2A_1A_0$

Connect inputs to address bits in order: q_1q_0x

Label ROM addresses: 000 to 111

Label ROM data columns: $Z_1Z_0D_1T_0$

Don't include next state in ROM. Where does it go?

D_1 and T_0 feed back to flip-flop inputs

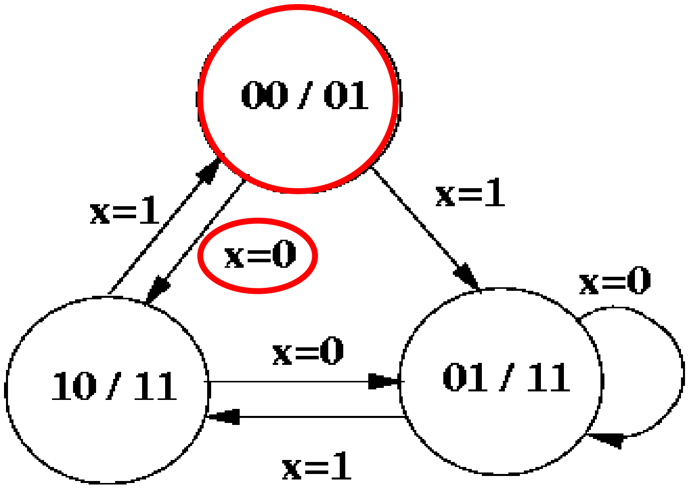
Leave the "don't cares" as they are

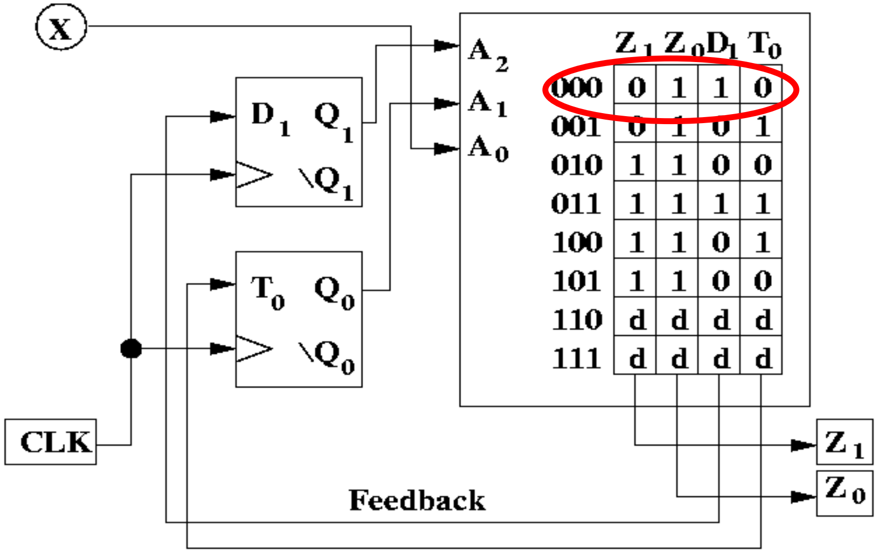
Circle input x to show it comes from an external source

Square the outputs z to show they are external

"Don't forget the stinkin' clock"

Finite state machines: implementing





	Z_1	Z_0	D_1	T_0
000	0	1	1	0
001	0	1	0	1
010	1	1	0	0
011	1	1	1	1
100	1	1	0	1
101	1	1	0	0
110	d	d	d	d
111	d	d	d	d

How it works:

Assume state 00, input 0

Select ROM address 000, data 0110

Output is 01

Next state is determined by D_1 = 1, T_0 = 0

Q_1 becomes 1 (D input)

Q_0 becomes 0 (hold)

Next state is 10

Try other states and inputs

Finite state machines: implementing

How else could we implement the circuit?

Inputs			Next		Outputs				Minterms
q_1	q_0	x	q_1^+	q_0^+	z_1	z_0	D_1	T_0	z_1
0	0	0	1	0	0	1	1	0	
0	0	1	0	1	0	1	0	1	
0	1	0	0	1	1	1	0	0	$\backslash q_1q_0\backslash x$
0	1	1	1	0	1	1	1	1	$\backslash q_1q_0x$
1	0	0	0	1	1	1	0	1	$q_1\backslash q_0\backslash x$
1	0	1	0	0	1	1	0	0	$q_1\backslash q_0x$
1	1	0	d	d	d	d	d	d	
1	1	1	d	d	d	d	d	d	

$$z_1 = \backslash q_1q_0\backslash x + \backslash q_1q_0x + q_1\backslash q_0\backslash x + q_1\backslash q_0x$$

etc.

Simplified:

$$z_1 = q_1 + q_0$$

$$z_0 = 1$$

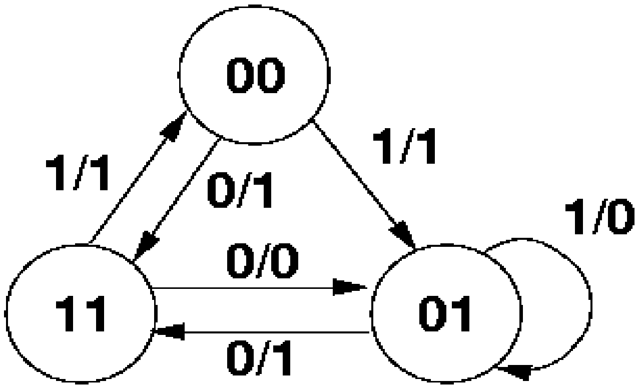
$$D_1 = \backslash q_1 (\backslash q_0\backslash x + q_0x)$$

$$T_0 = \backslash q_1x + q_1\backslash x$$

Implement using AND and OR gates or PLA

Finite state machines: implementing

q_1	q_0	x	q_1^+	q_0^+	z
0	0	0	1	1	1
0	0	1	0	1	1
0	1	0	1	1	1
0	1	1	0	1	0
1	0	0	d	d	d
1	0	1	d	d	d
1	1	0	0	1	0
1	1	1	0	0	1



In state 01, input of 0 gives state 11 and output 1
In state 01, input of 1 gives state 01 and output 0
In state 11, input of 0 gives state 01 and output 0
In state 11, input of 1 gives state 00 and output 1

Rest of steps:

- pick flip-flops
- use excitation tables
- draw circuit (ROM, gates, or PLA)