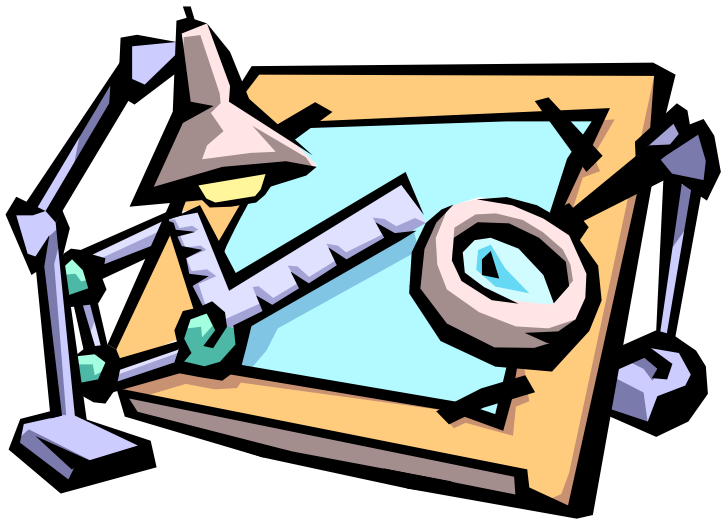
Sequential circuit design

- In sequential circuit design, we turn some description into a working circuit
  - We first make a state table or diagram to express the computation
  - Then we can turn that table or diagram into a sequential circuit



Sequence recognizers

- A sequence recognizer is a special kind of sequential circuit that looks for a special bit pattern in some input
- The recognizer circuit has only one input, X
  - One bit of input is supplied on every clock cycle
  - This is an easy way to permit arbitrarily long input sequences
- There is one output, Z, which is 1 when the desired pattern is found
- Our example will detect the bit pattern "1001":

  Inputs: 1 1 1 0 0 1 1 0 1 0 0 1 0 0 1 1 0 …
  Outputs: 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 …

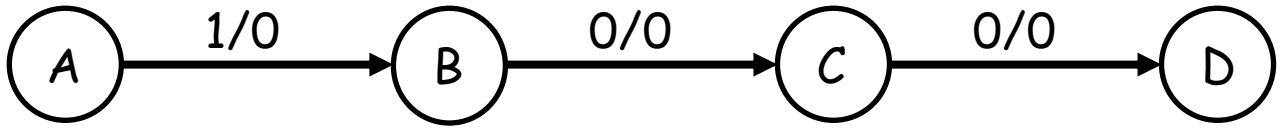
- A sequential circuit is required because the circuit has to "remember" the inputs from previous clock cycles, in order to determine whether or not a match was found

Step 1: Making a state table

- The first thing you have to figure out is precisely how the use of state will help you solve the given problem
  - Make a state table based on the problem statement. The table should show the present states, inputs, next states and outputs
  - Sometimes it is easier to first find a state diagram and then convert that to a table
- This is usually the most difficult step. Once you have the state table, the rest of the design procedure is the same for all sequential circuits

A basic state diagram

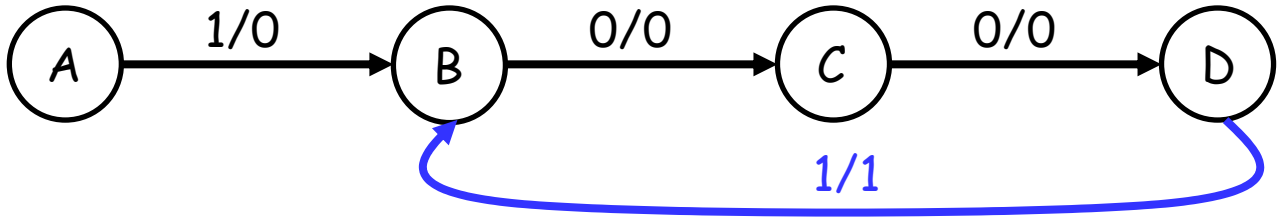
- What state do we need for the sequence recognizer?
  - We have to "remember" inputs from previous clock cycles
  - For example, if the previous three inputs were 100 and the current input is 1, then the output should be 1
  - In general, we will have to remember occurrences of parts of the desired pattern—in this case, 1, 10, and 100
- We'll start with a basic state diagram:



State	Meaning
A	None of the desired pattern (1001) has been input yet.
B	We've already seen the first bit (1) of the desired pattern.
C	We've already seen the first two bits (10) of the desired pattern.
D	We've already seen the first three bits (100) of the desired pattern.

Overlapping occurrences of the pattern

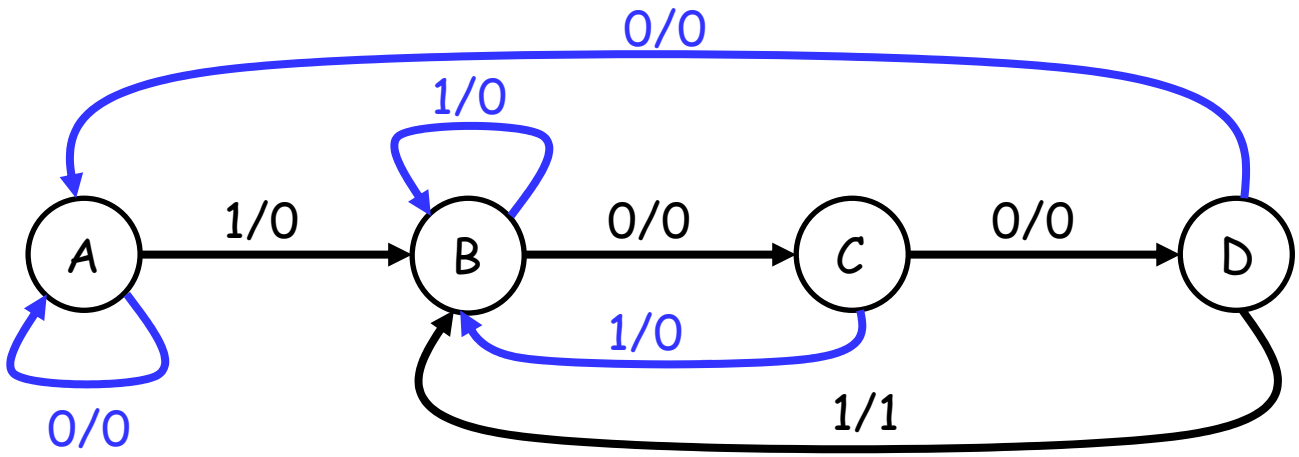
- What happens if we're in state D (the last three inputs were 100), and the current input is 1?
  - The output should be a 1, because we've found the desired pattern
  - But this last 1 could also be the start of another occurrence of the pattern! For example, 1001001 contains *two* occurrences of 1001
  - To detect overlapping occurrences of the pattern, the next state should be B.



State	Meaning
A	None of the desired pattern (1001) has been input yet.
B	We've already seen the first bit (1) of the desired pattern.
C	We've already seen the first two bits (10) of the desired pattern.
D	We've already seen the first three bits (100) of the desired pattern.

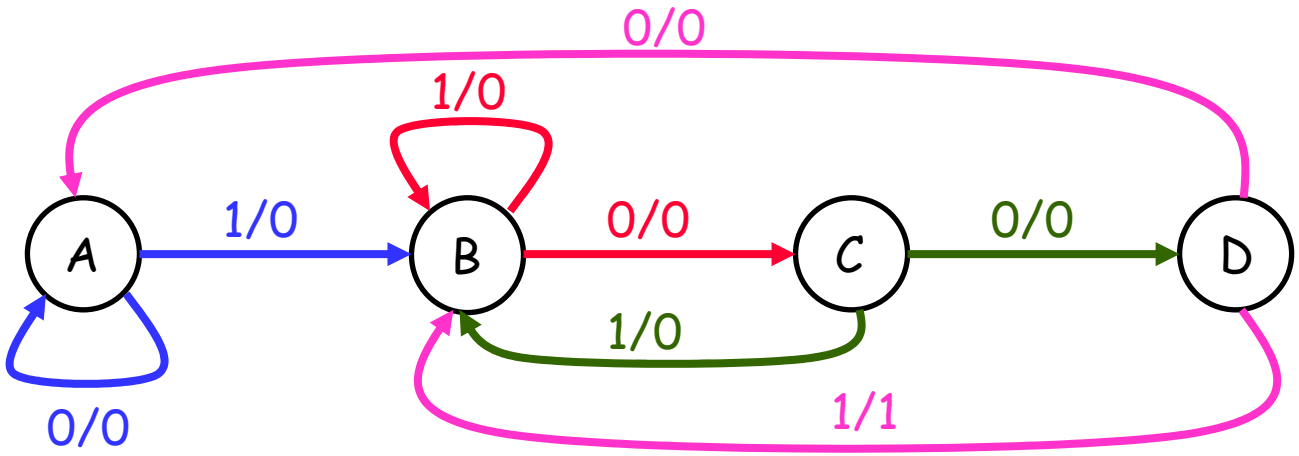
Filling in the other arrows

- *Two* outgoing arrows for each node, to account for the possibilities of X=0 and X=1
- The remaining arrows we need are shown in blue. They also allow for the correct detection of overlapping occurrences of 1001.

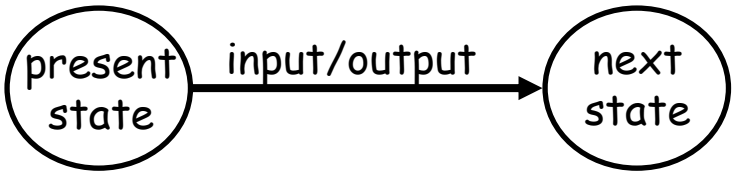


State	Meaning
A	None of the desired pattern (1001) has been input yet.
B	We've already seen the first bit (1) of the desired pattern.
C	We've already seen the first two bits (10) of the desired pattern.
D	We've already seen the first three bits (100) of the desired pattern.

Finally, making the state table



Remember how the state diagram arrows correspond to rows of the state table:



Present State	Input	Next State	Output
A	0	A	0
A	1	B	0
B	0	C	0
B	1	B	0
C	0	D	0
C	1	B	0
D	0	A	0
D	1	B	1

Sequential circuit design procedure

Step 1:

Make a state table based on the problem statement. The table should show the present states, inputs, next states and outputs. (It may be easier to find a state diagram first, and then convert that to a table)

Step 2:

Assign binary codes to the states in the state table, if you haven't already. If you have n states, your binary codes will have at least $\lceil \log_2 n \rceil$ digits, and your circuit will have at least $\lceil \log_2 n \rceil$ flip-flops

Step 3:

For each flip-flop and each row of your state table, find the flip-flop input values that are needed to generate the next state from the present state. You can use flip-flop excitation tables here.

Step 4:

Find simplified equations for the flip-flop inputs and the outputs.

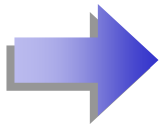
Step 5:

Build the circuit!

Step 2: Assigning binary codes to states

- We have four states ABCD, so we need at least two flip-flops Q_1Q_0
- The easiest thing to do is represent state A with Q_1Q_0 = 00, B with 01, C with 10, and D with 11
- The state assignment can have a big impact on circuit complexity, but we won't worry about that too much in this class

Present State	Input	Next State	Output
A	0	A	0
A	1	B	0
B	0	C	0
B	1	B	0
C	0	D	0
C	1	B	0
D	0	A	0
D	1	B	1



Present State Q_1	Present State Q_0	Input X	Next State Q_1	Next State Q_0	Output Z
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	0	1	1

Step 3: Finding flip-flop input values

- Next we have to figure out how to actually make the flip-flops change from their present state into the desired next state
- This depends on what kind of flip-flops you use!
- We'll use two JKs. For each flip-flip Q_i, look at its present and next states, and determine what the inputs J_i and K_i should be in order to make that state change.

Present State		Input	Next State		Flip flop inputs				Output
Q_1	Q_0	X	Q_1	Q_0	J_1	K_1	J_0	K_0	Z
0	0	0	0	0					0
0	0	1	0	1					0
0	1	0	1	0					0
0	1	1	0	1					0
1	0	0	1	1					0
1	0	1	0	1					0
1	1	0	0	0					0
1	1	1	0	1					1

Finding JK flip-flop input values

- For JK flip-flops, this is a little tricky. Recall the characteristic table:

J	K	Q(t+1)	Operation
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'(t)	Complement

- If the present state of a JK flip-flop is 0 and we want the next state to be 1, then we have *two* choices for the JK inputs:
  - We can use JK= 10, to explicitly set the flip-flop's next state to 1
  - We can also use JK=11, to complement the current state 0
- So to change from 0 to 1, we must set J=1, but K could be *either* 0 or 1
- Similarly, the other possible state transitions can all be done in two different ways as well

JK excitation table

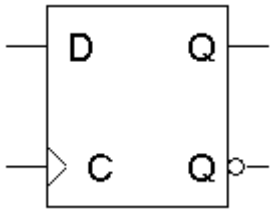
- An excitation table shows what flip-flop inputs are required in order to make a desired state change

Q(t)	Q(t+1)	J	K	Operation
0	0	0	x	No change/reset
0	1	1	x	Set/complement
1	0	x	1	Reset/complement
1	1	x	0	No change/set

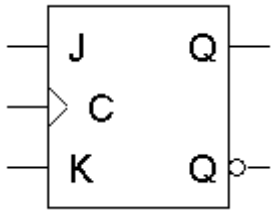
- This is the same information that's given in the characteristic table, but presented "backwards"

J	K	Q(t+1)	Operation
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'(t)	Complement

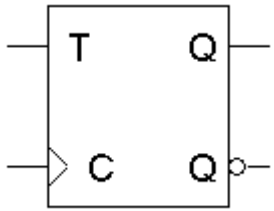
Excitation tables for all flip-flops



Q(t)	Q(t+1)	D	Operation
0	0	0	Reset
0	1	1	Set
1	0	0	Reset
1	1	1	Set



Q(t)	Q(t+1)	J	K	Operation
0	0	0	x	No change/reset
0	1	1	x	Set/complement
1	0	x	1	Reset/complement
1	1	x	0	No change/set



Q(t)	Q(t+1)	T	Operation
0	0	0	No change
0	1	1	Complement
1	0	1	Complement
1	1	0	No change

Back to the example

- Use the JK excitation table on the right to find the correct values for *each* flip-flop's inputs, based on its present and next states

Q(t)	Q(t+1)	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Present State		Input	Next State		Flip flop inputs				Output
Q_1	Q_0	X	Q_1	Q_0	J_1	K_1	J_0	K_0	Z
0	0	0	0	0	0	x	0	x	0
0	0	1	0	1	0	x	1	x	0
0	1	0	1	0	1	x	x	1	0
0	1	1	0	1	0	x	x	0	0
1	0	0	1	1	x	0	1	x	0
1	0	1	0	1	x	1	1	x	0
1	1	0	0	0	x	1	x	1	0
1	1	1	0	1	x	1	x	0	1

Step 4: Find equations for the FF inputs and output

- Now you can make K-maps and find equations for each of the four flip-flop inputs, as well as for the output Z
- These equations are in terms of the present state and the inputs
- The advantage of using JK flip-flops is that there are many don't care conditions, which can result in simpler MSP equations

Present State		Input	Next State		Flip flop inputs				Output
Q_1	Q_0	X	Q_1	Q_0	J_1	K_1	J_0	K_0	Z
0	0	0	0	0	0	x	0	x	0
0	0	1	0	1	0	x	1	x	0
0	1	0	1	0	1	x	x	1	0
0	1	1	0	1	0	x	x	0	0
1	0	0	1	1	x	0	1	x	0
1	0	1	0	1	x	1	1	x	0
1	1	0	0	0	x	1	x	1	0
1	1	1	0	1	x	1	x	0	1

$J_1 = X' Q_0$
$K_1 = X + Q_0$

$J_0 = X + Q_1$
$K_0 = X'$

$Z = Q_1 Q_0 X$

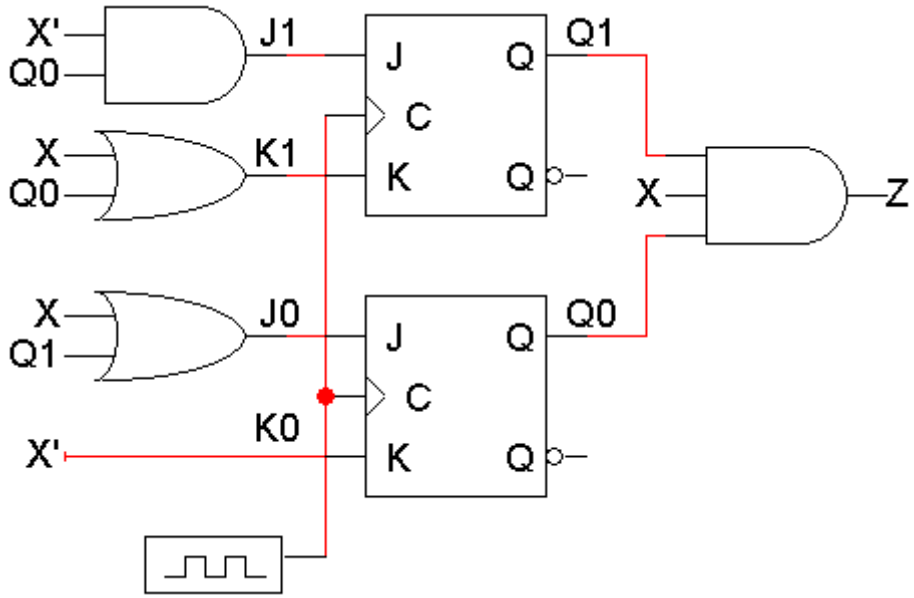
Step 5: Build the circuit

- Lastly, we use these simplified equations to build the completed circuit

$J_1 = X' Q_0$
$K_1 = X + Q_0$

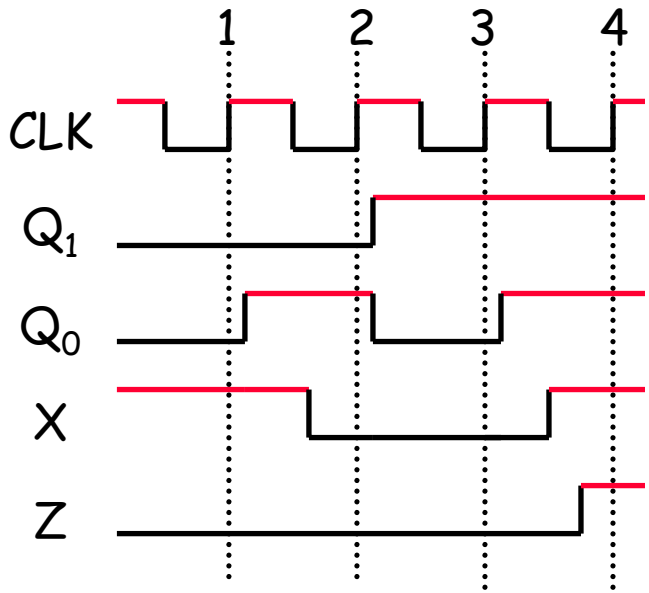
$J_0 = X + Q_1$
$K_0 = X'$

$Z = Q_1 Q_0 X$



Timing diagram

- Here is one example timing diagram for our sequence detector
  - The flip-flops Q_1Q_0 start in the initial state, 00
  - On the first three positive clock edges, X is 1, 0, and 0. These inputs cause Q_1Q_0 to change, so after the third edge $Q_1Q_0 = 11$
  - Then when X=1, Z becomes 1 also, meaning that 1001 was found
- The output Z does not have to change at positive clock edges. Instead, it may change whenever X changes, since $Z = Q_1Q_0X$



Building the same circuit with D flip-flops

- What if you want to build the circuit using D flip-flops instead?
- We already have the state table and state assignments, so we can just start from Step 3, finding the flip-flop input values
- D flip-flops have only one input, so our table only needs two columns for D_1 and D_0

Present State		Input	Next State		Flip-flop inputs		Output
Q_1	Q_0	X	Q_1	Q_0	D_1	D_0	Z
0	0	0	0	0			0
0	0	1	0	1			0
0	1	0	1	0			0
0	1	1	0	1			0
1	0	0	1	1			0
1	0	1	0	1			0
1	1	0	0	0			0
1	1	1	0	1			1

D flip-flop input values (Step 3)

- The D excitation table is pretty boring; set the D input to whatever the next state should be
- You don't even need to show separate columns for D_1 and D_0; you can just use the Next State columns

Q(t)	Q(t+1)	D	Operation
0	0	0	Reset
0	1	1	Set
1	0	0	Reset
1	1	1	Set

Present State		Input	Next State		Flip flop inputs		Output
Q_1	Q_0	X	Q_1	Q_0	D_1	D_0	Z
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	1	0	1	0
1	0	0	1	1	1	1	0
1	0	1	0	1	0	1	0
1	1	0	0	0	0	0	0
1	1	1	0	1	0	1	1

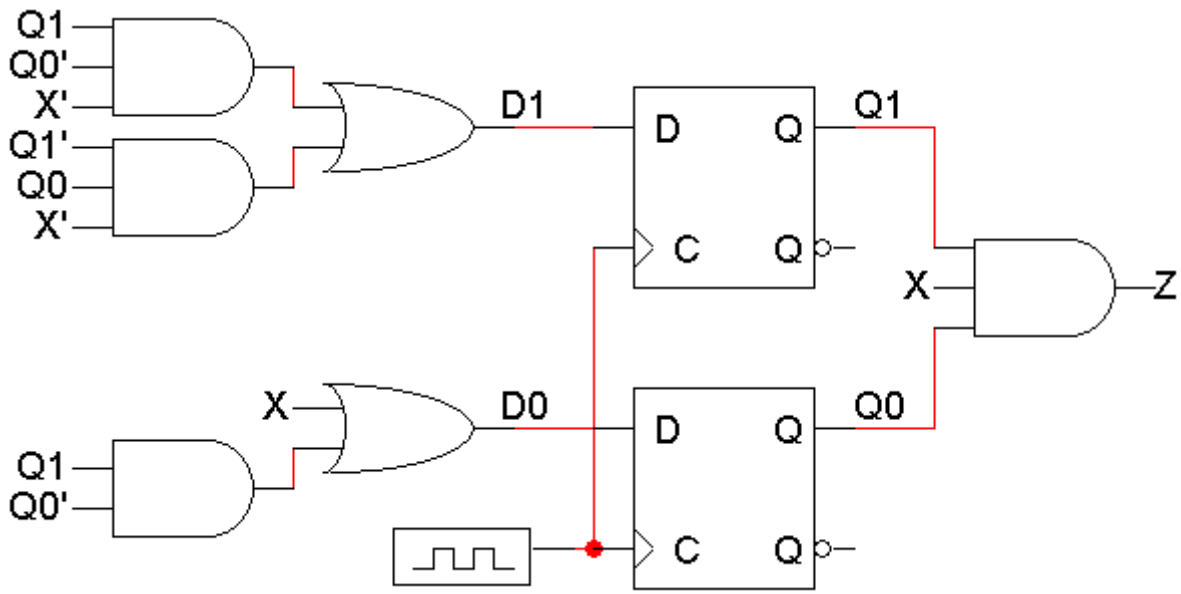
Finding equations (Step 4)

- You can do K-maps again, to find:

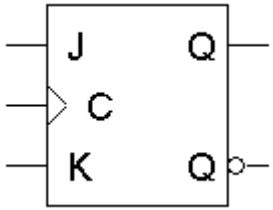
$$D_1 = Q_1\, Q_0'\, X' + Q_1'\, Q_0\, X'$$
$$D_0 = X + Q_1\, Q_0'$$
$$Z = Q_1\, Q_0\, X$$

Present State Q_1	Present State Q_0	Input X	Next State Q_1	Next State Q_0	Flip flop inputs D_1	Flip flop inputs D_0	Output Z
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	1	0	1	0
1	0	0	1	1	1	1	0
1	0	1	0	1	0	1	0
1	1	0	0	0	0	0	0
1	1	1	0	1	0	1	1

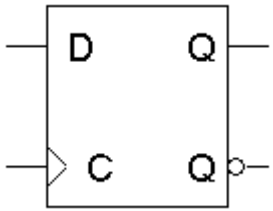
Building the circuit (Step 5)



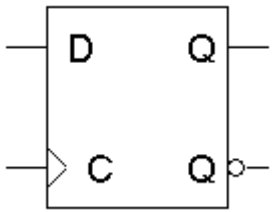
Flip-flop comparison



JK flip-flops are good because there are many don't care values in the flip-flop inputs, which can lead to a simpler circuit



D flip-flops have the advantage that you don't have to set up flip-flop inputs at all, since Q(t+1) = D. However, the D input equations are usually more complex than JK input equations



In practice, D flip-flops are used more often

- There is only one input for each flip-flop, not two
- There are no excitation tables to worry about
- D flip-flops can be implemented with slightly less hardware than JK flip-flops

Summary

- The basic sequential circuit design procedure:
  - Make a state table and, if desired, a state diagram. This step is usually the hardest
  - Assign binary codes to the states if you didn't already
  - Use the present states, next states, and flip-flop excitation tables to find the flip-flop input values
  - Write simplified equations for the flip-flop inputs and outputs and build the circuit