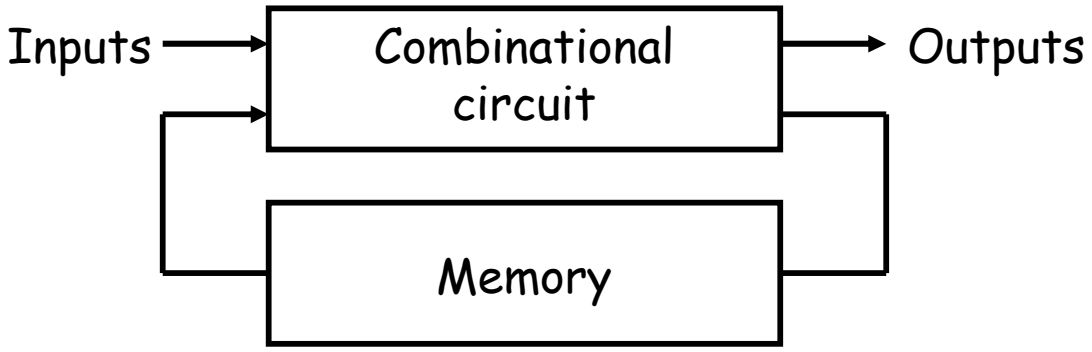
Lecture 9

Sequential circuits: Design and Analysis

Learning outcomes

- Produce a state diagram/table to describe the behavior of a sequential circuit.
- Produce a circuit from a state diagram/table.
- Describe a Mealy machine.
- Describe a Moore machine.
- Turn a sequential circuit schematic into a state diagram/table describing the circuit.

Sequential circuits



- The outputs of a **sequential circuit** depend on not only the inputs, but also the **state**, or the current contents of some memory.
- This makes things more difficult to understand, since the same inputs can yield different outputs, depending on what's stored in memory.
- The memory contents can also change as the circuit runs.
- We'll some need new techniques for analyzing and designing sequential circuits.

Sequential Circuit Design and Analysis

- 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.
- In **sequential circuit analysis**, we turn some sequential circuit into a state table/state diagram describing the circuit.
- In this lecture, we start reviewing this material through the design of a sequence recognizer.

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. For example, it would take 20 cycles to scan a 20-bit input.
  - 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 …

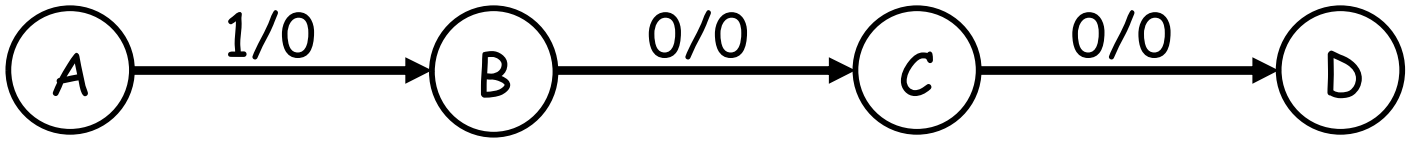
Outputs: 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 …

Here, one input and one output bit appear every clock cycle.

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

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 (edges are labeled input/output):



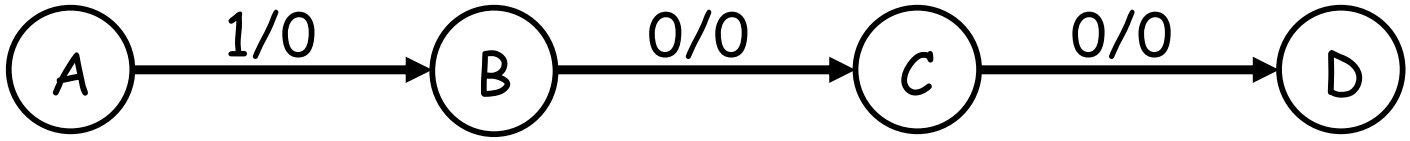
State	Meaning
A	None of the desired pattern (1001) has been input yet.
B	We've already seen the first bit (1) of the desired pattern.
C	We've already seen the first two bits (10) of the desired pattern.
D	We've already seen the first three bits (100) of the desired pattern.

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.
- Sequence recognizers are especially hard! They're the hardest example we'll see in this class, so if you understand this you're in good shape.

A basic state diagram

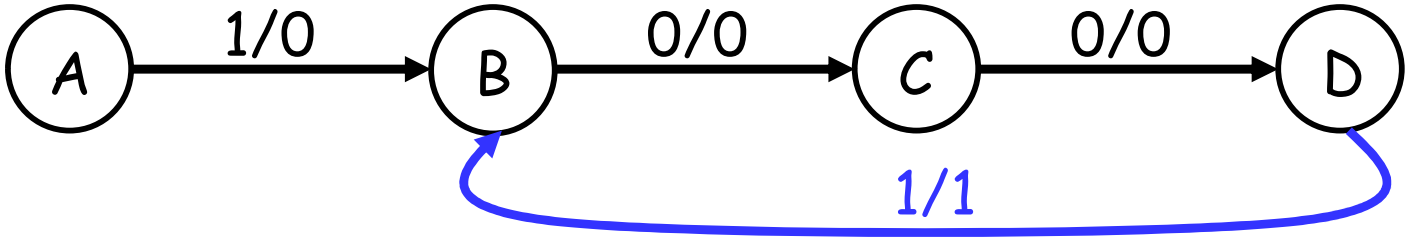
- What state do we need for the sequence recognizer?
  - We have to "remember" inputs from previous clock cycles.
  - For example, if the previous three inputs were 100 and the current input is 1, then the output should be 1.
  - In general, we will have to remember occurrences of parts of the desired 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

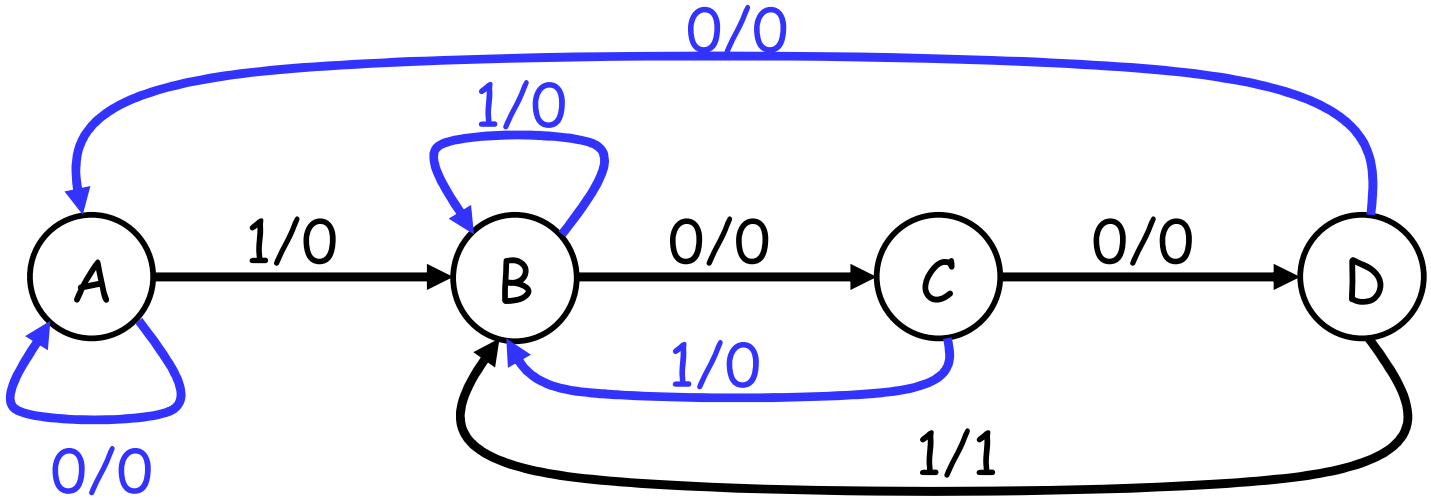
- What happens if we're in state D (the last three inputs were 100), and the current input is 1?
  - The output should be a 1, because we've found the desired pattern.
  - But this last 1 could also be the start of another occurrence of the pattern! For example, 1001001 contains two occurrences of 1001.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

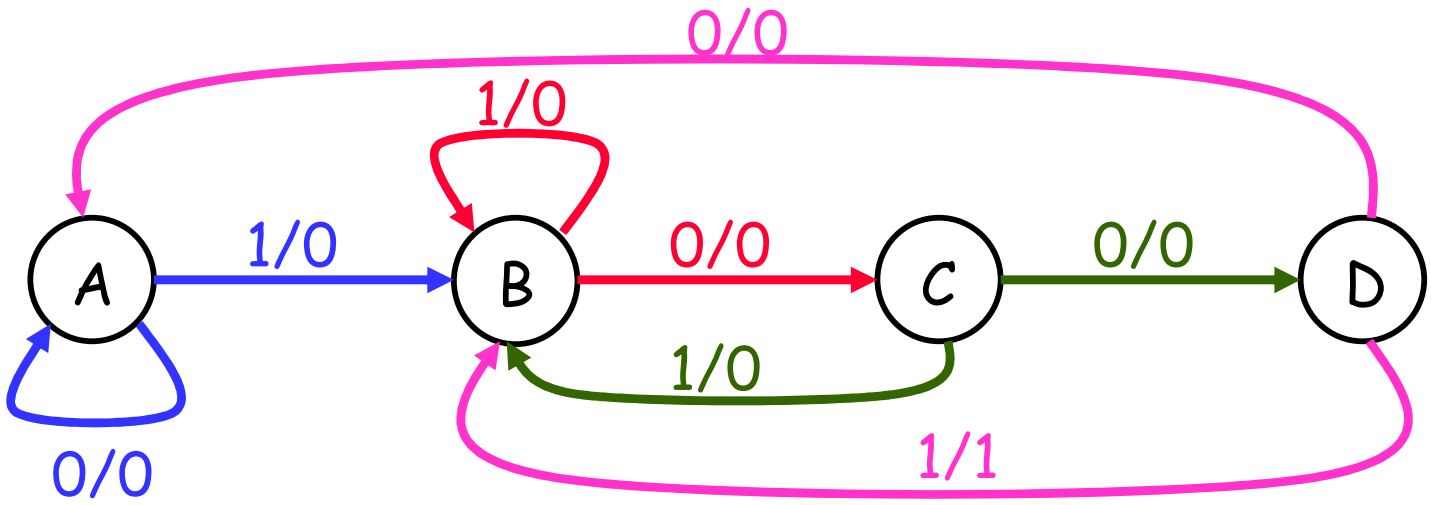
Filling in the other arrows

- Remember that we need *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.

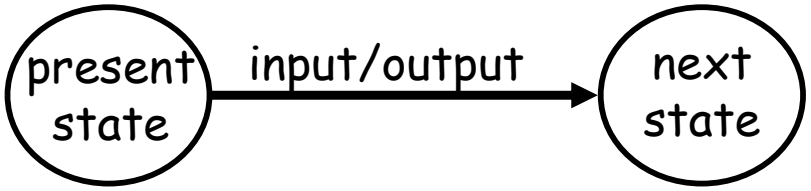


State	Meaning
A	None of the desired pattern (1001) has been input yet.
B	We've already seen the first bit (1) of the desired pattern.
C	We've already seen the first two bits (10) of the desired pattern.
D	We've already seen the first three bits (100) of the desired pattern.

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

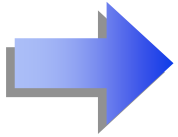
Sequential circuit design procedure

- **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 $\log_2$ (n) digits, and your circuit will have at least $\log_2$ (n) 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. More on this later.

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

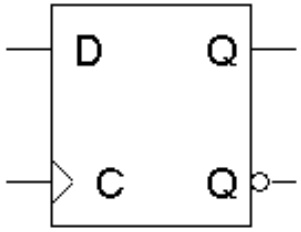
- An **excitation table** shows what flip-flop inputs are required in order to make a desired state change.

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

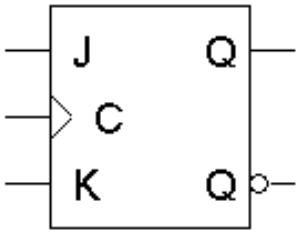
- This is the same information that's given in the characteristic table, but presented "backwards."

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

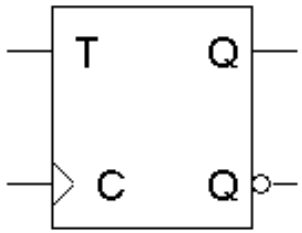
Excitation tables for all flip-flops



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

- We can now 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 Q_1	Present State Q_0	Input X	Next State Q_1	Next State Q_0	Flip flop inputs J_1	Flip flop inputs K_1	Flip flop inputs J_0	Flip flop inputs K_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

Back to the example

- We can now 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 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

$J_1 = X' Q_0$

$K_1 = X + Q_0$

$J_0 = X + Q_1$

$K_0 = X'$

$Z = Q_1 Q_0 X$

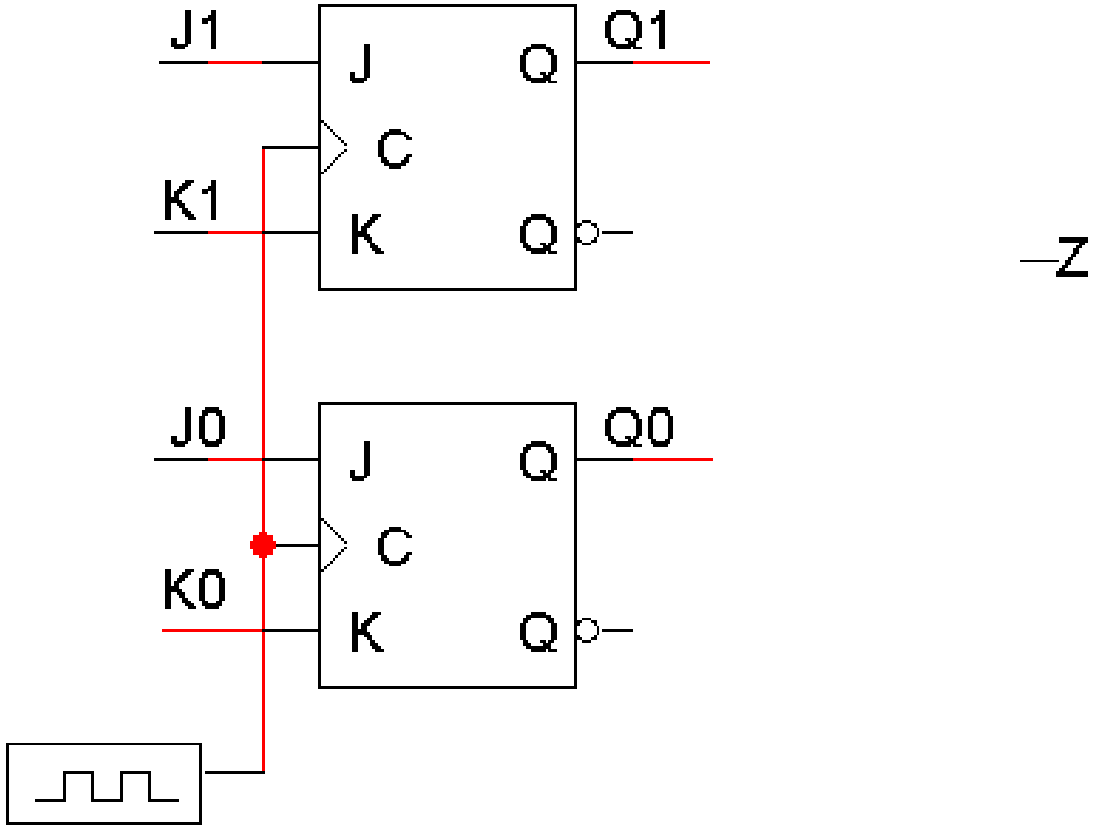
Step 5: Build the circuit

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

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

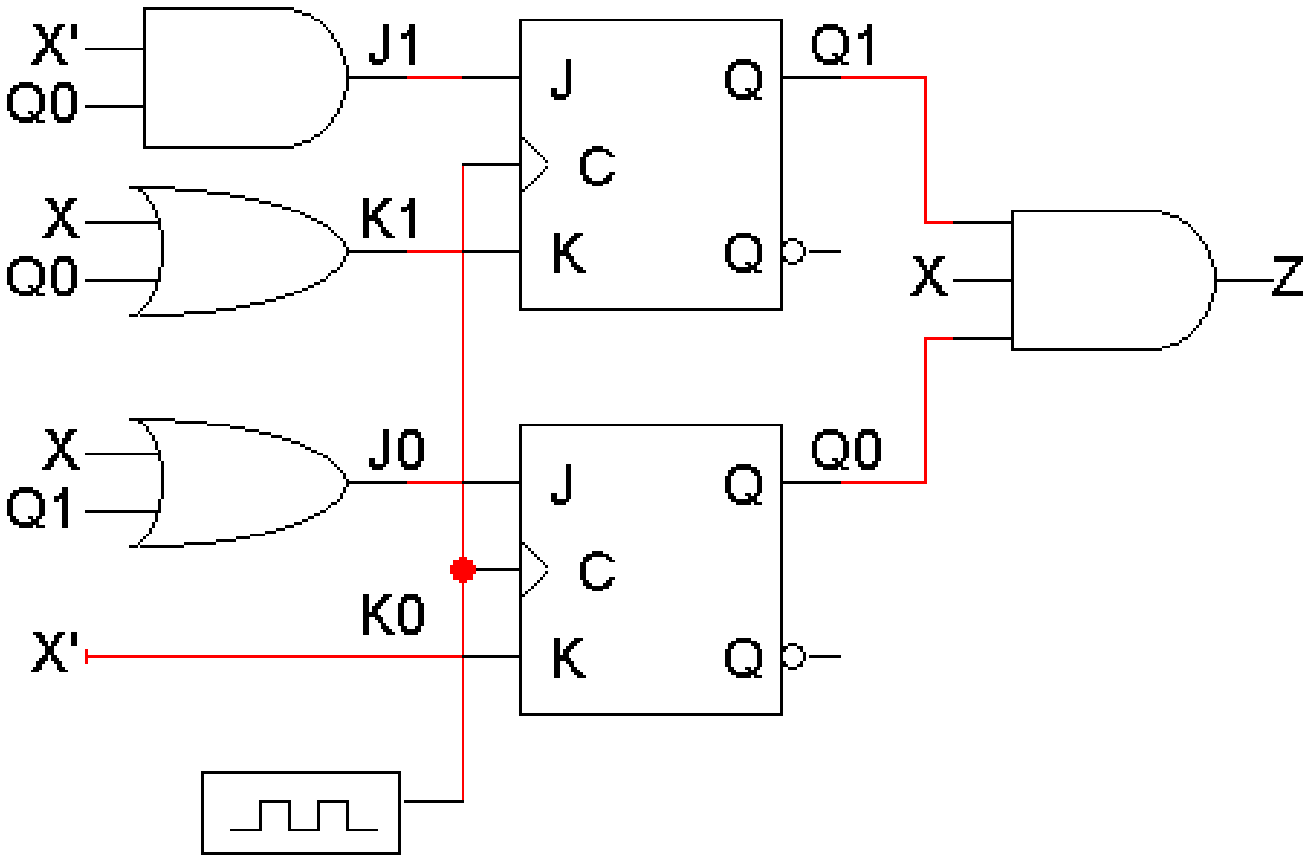
$J_0 = X + Q_1$
$K_0 = X'$

$Z = Q_1 Q_0 X$



Step 5: Build the circuit

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



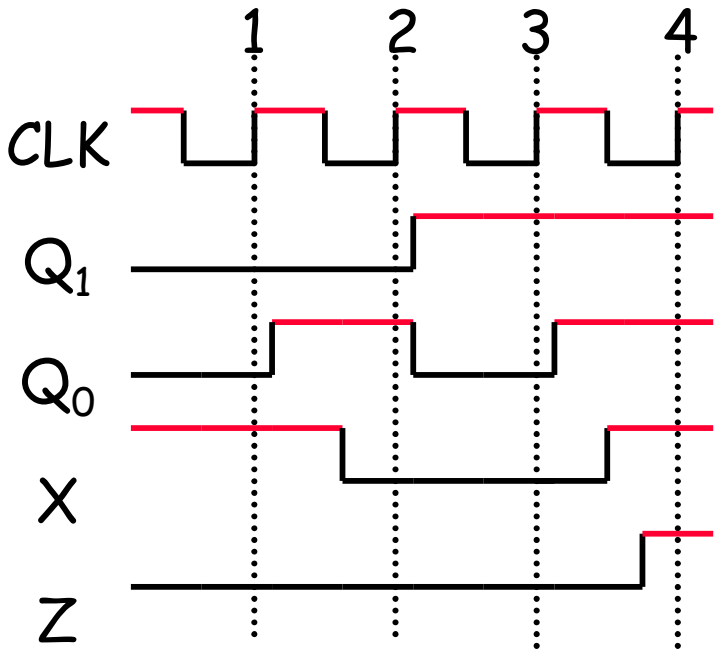
$J_1 = X' Q_0$
$K_1 = X + Q_0$

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

$Z = Q_1 Q_0 X$

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:

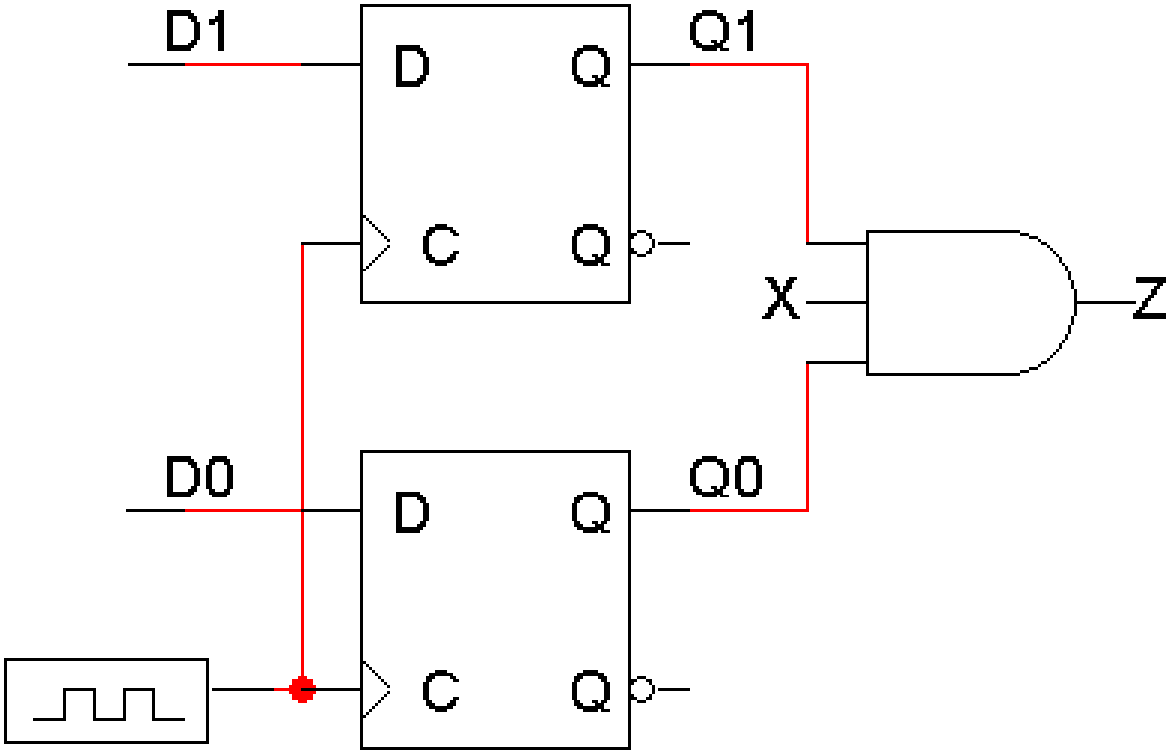
$$D_1 = Q_1\, Q_0'\, X' + Q_1'\, Q_0\, X'$$

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

$$Z = Q_1\, Q_0\, X$$

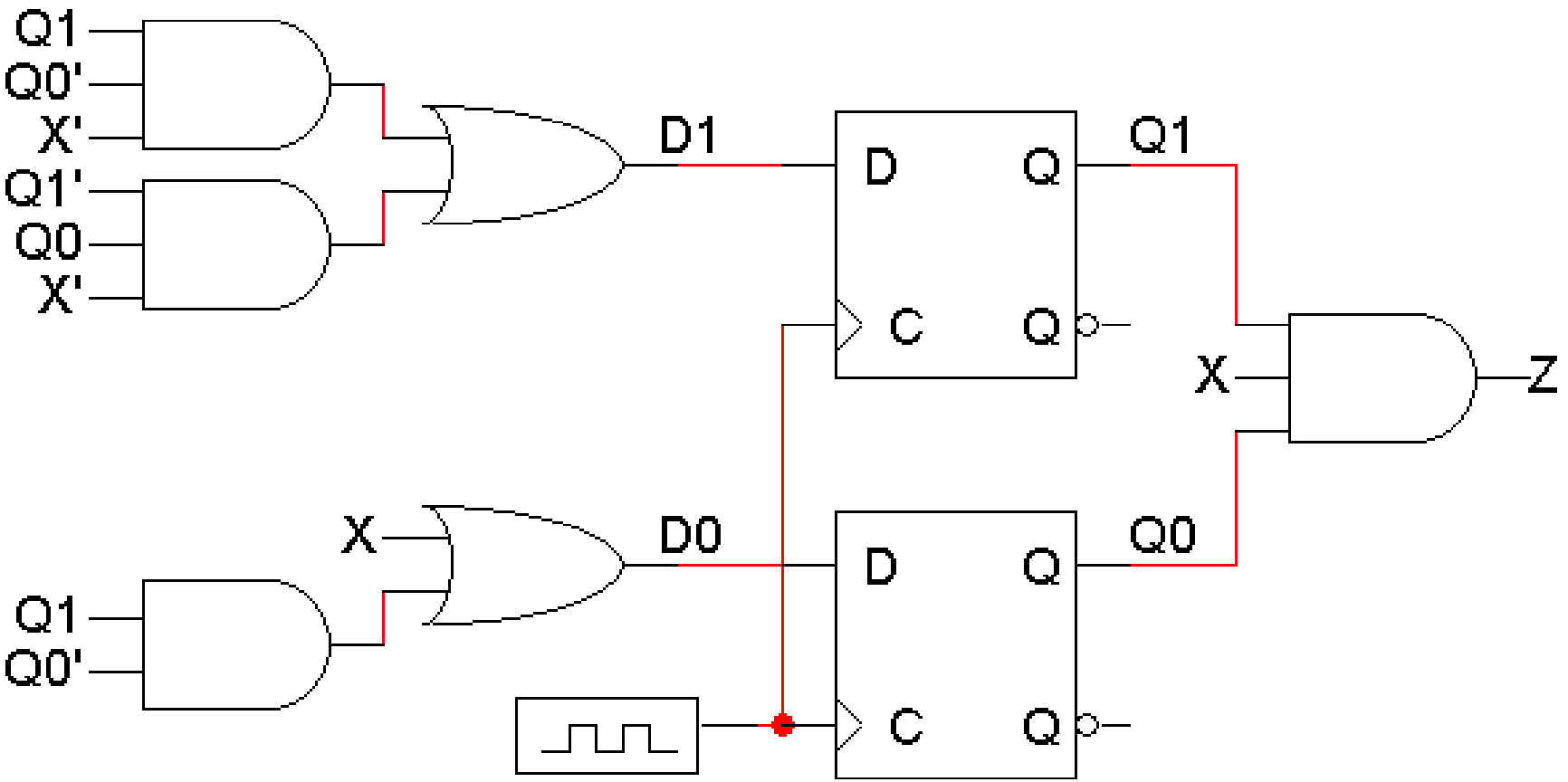
Present State		Input	Next State		Flip flop inputs		Output
Q_1	Q_0	X	Q_1	Q_0	D_1	D_0	Z
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	1	0	1	0
1	0	0	1	1	1	1	0
1	0	1	0	1	0	1	0
1	1	0	0	0	0	0	0
1	1	1	0	1	0	1	1

Building the circuit (Step 5)

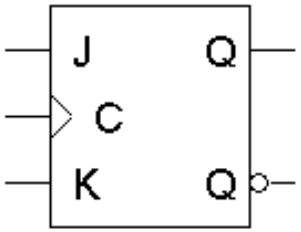


D1 = Q1 Q0′ X′ + Q1′ Q0 X′
D0 = X + Q1 Q0′

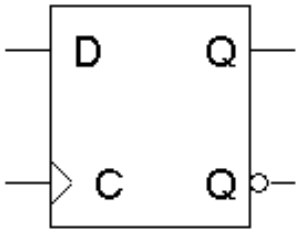
Building the circuit (Step 5)



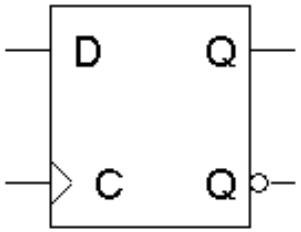
Flip-flop comparison



- 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

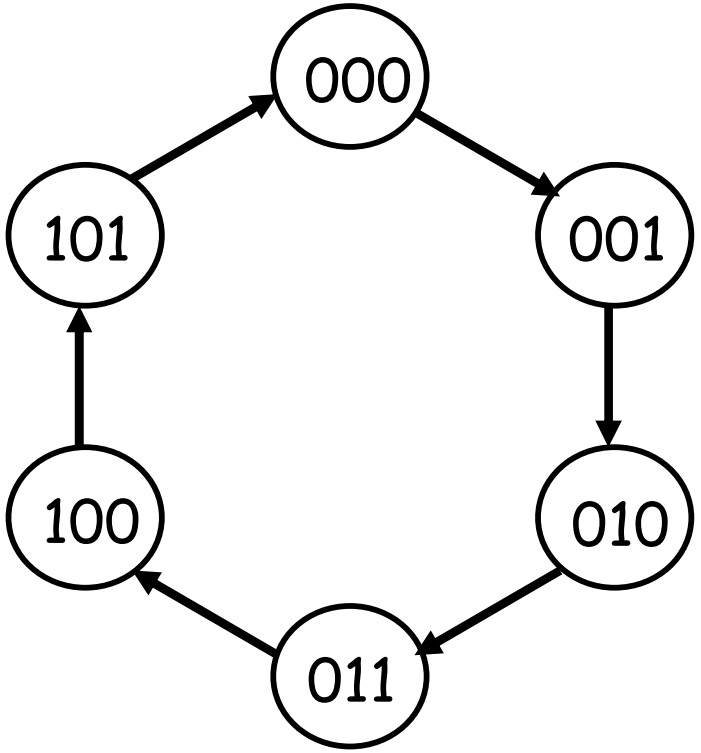


- In practice, D flip-flops are used more often.
  - There is only one input for each flip-flop, not two.
  - There are no excitation tables to worry about.
  - D flip-flops can be implemented with slightly less hardware than JK flip-flops.

Unused states

- The examples shown so far have all had 2^n states, and used n flip-flops. But sometimes you may have unused, leftover states.
- For example, here is a state table and diagram for a counter that repeatedly counts from 0 (000) to 5 (101).
- What should we put in the table for the two unused states?

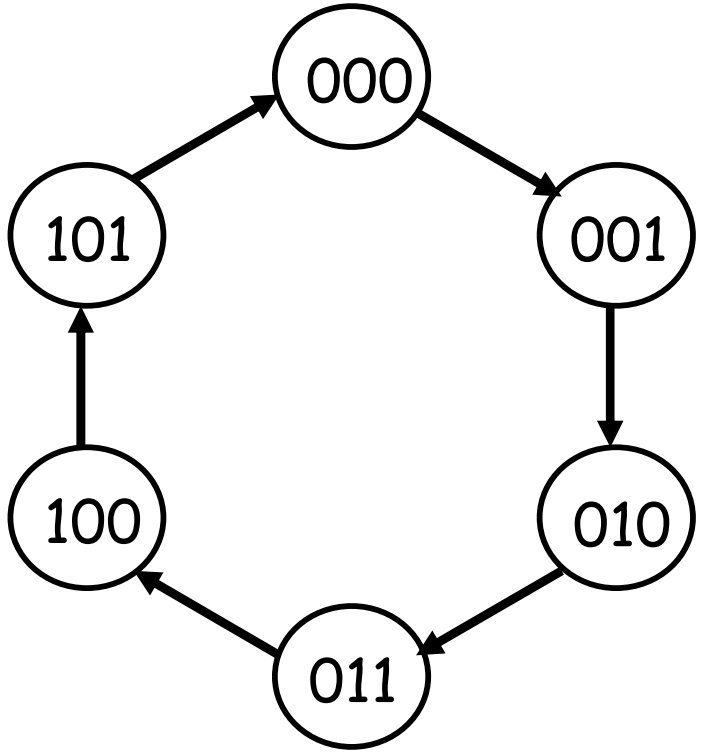
Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	0	0	0
1	1	0	?	?	?
1	1	1	?	?	?



Unused states can be don't cares…

- To get the *simplest* possible circuit, you can fill in don't cares for the next states. This will also result in don't cares for the flip-flop inputs, which can simplify the hardware.
- If the circuit somehow ends up in one of the unused states (110 or 111), its behavior will depend on exactly what the don't cares were filled in with.

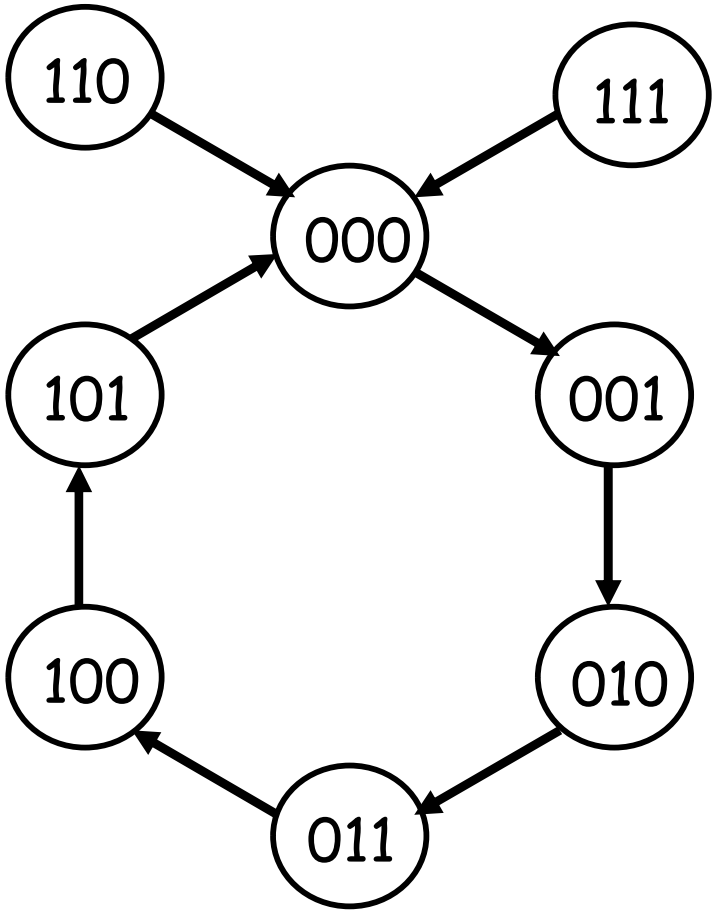
Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	0	0	0
1	1	0	x	x	x
1	1	1	x	x	x



…or maybe you *do* care

- To get the *safest* possible circuit, you can explicitly fill in next states for the unused states 110 and 111.
- This guarantees that even if the circuit somehow enters an unused state, it will eventually end up in a valid state.
- This is called a **self-starting counter**.

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

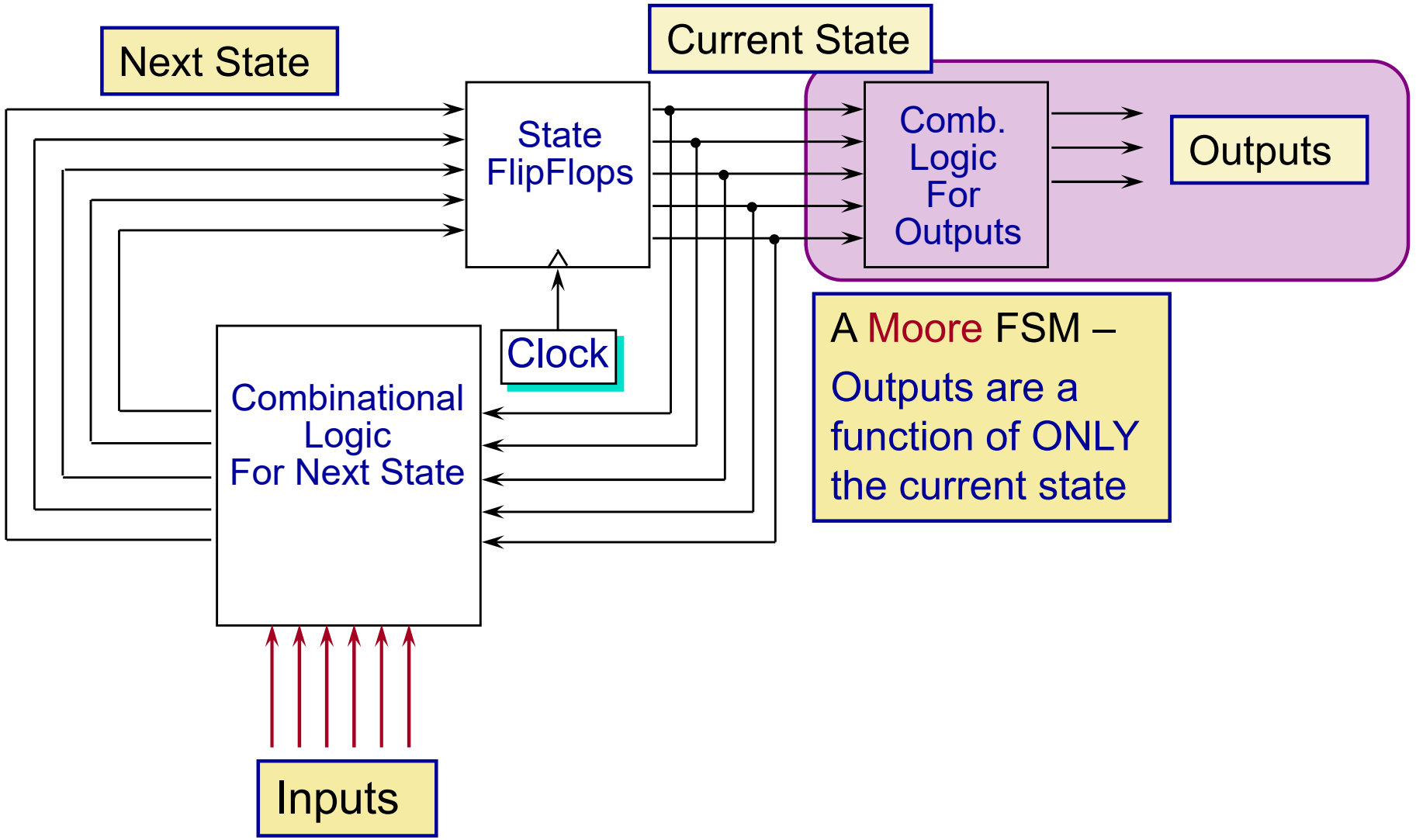


Mealy and Moore Machines

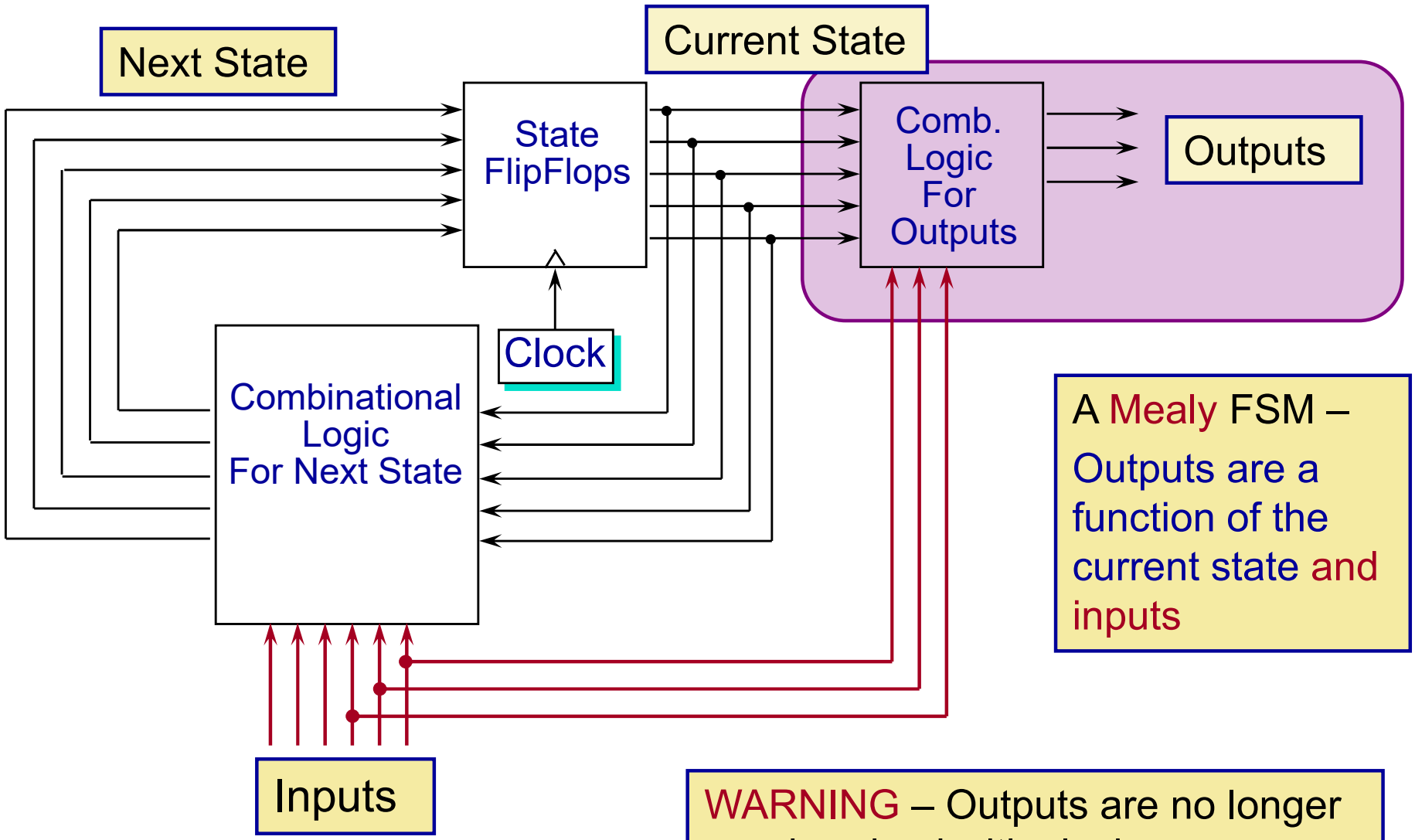
Mealy and Moore Machines

- Values stored in registers represent the state of the circuit.
- States determined by possible values in sequential storage elements
- Transitions: change of state
- Clock controls when state can change by controlling storage elements
- Combinational logic computes:
  - next state
    - function of current state and inputs
  - outputs
    - function of current state and inputs (Mealy machine)
    - function of current state only (Moore machine)

Moore Machines



Mealy Machines



A Mealy FSM –
Outputs are a function of the current state and inputs

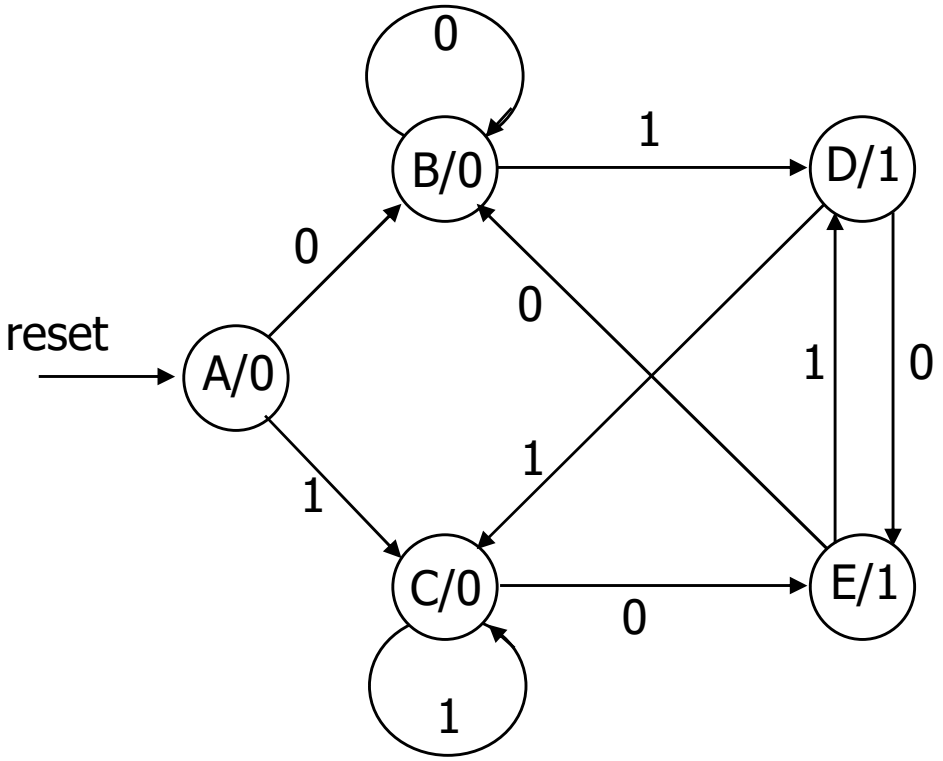
WARNING – Outputs are no longer synchronized with clock

Comparison of Mealy and Moore machines

- Mealy machines tend to have less states
  - different outputs on arcs rather than states
- Moore machines are safer to use
  - outputs change at clock edge
  - in Mealy machines, input change can cause output change as soon as logic is done – a big problem when two machines are interconnected –Mealy machines react faster to inputs
  - react in same cycle – don't need to wait for clock
  - in Moore machines, more logic may be necessary to decode state into outputs – more gate delays after clock edge

Specifying outputs for a Moore machine

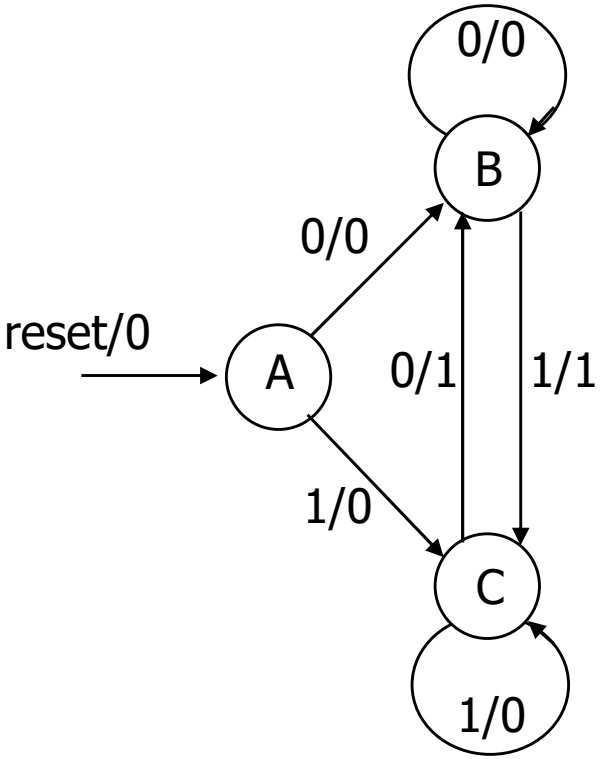
- Output is only function of state
  - specify in state bubble in state diagram
  - Output associated with state
  - example: sequence detector for 01 or 10



reset	input	current state	next state	output
1	–	–	A	
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	D	0
0	0	C	E	0
0	1	C	C	0
0	0	D	E	1
0	1	D	C	1
0	0	E	B	1
0	1	E	D	1

Specifying outputs for a Mealy machine

- Output is function of state and inputs
  - specify output on transition arc between states
  - Output associated with transition
  - example: sequence detector for 01 or 10



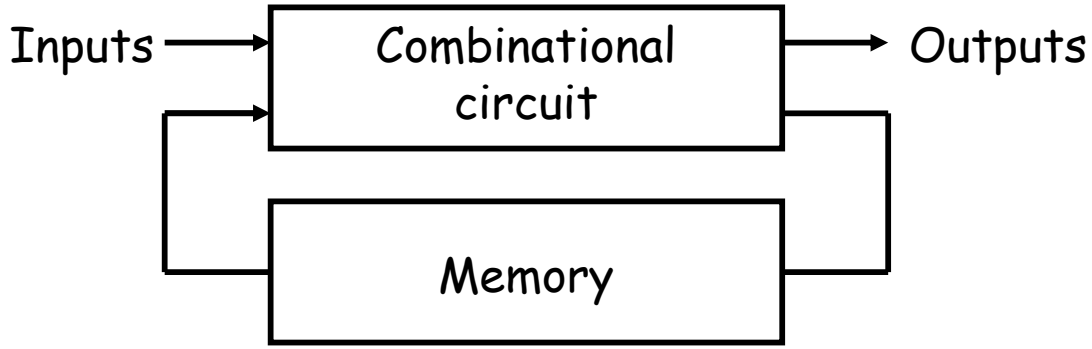
reset	input	current state	next state	output
1	–	–	A	0
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	C	1
0	0	C	B	1
0	1	C	C	0

Sequential Circuit Analysis

Sequential Circuit Analysis

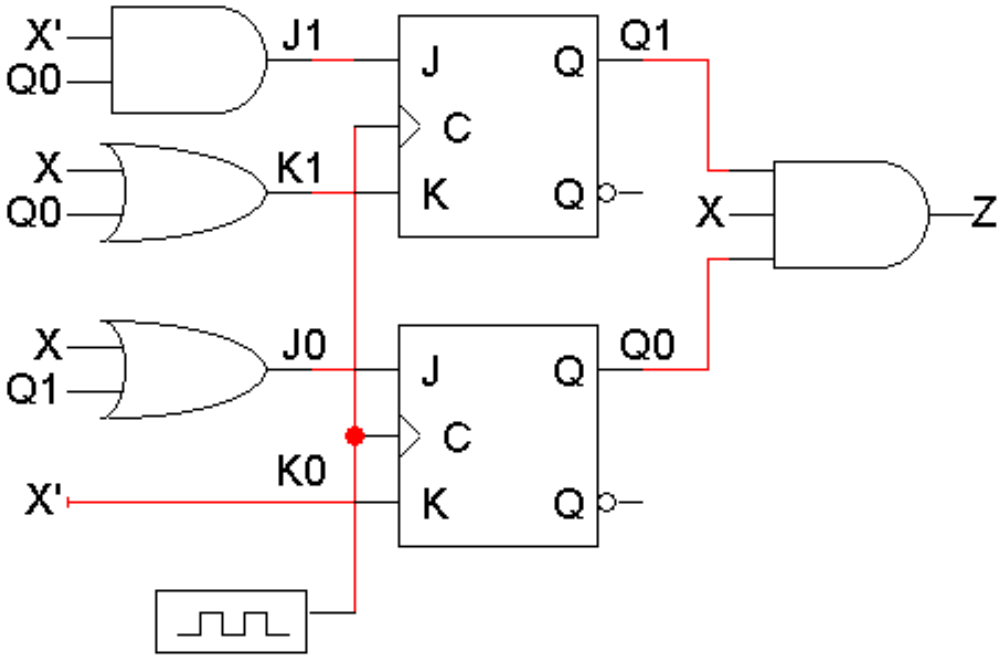
- In sequential circuit design, we turn some description into a working circuit.
  - We first make a state table or diagram to express the computation.
  - Then we can turn that table or diagram into a sequential circuit.

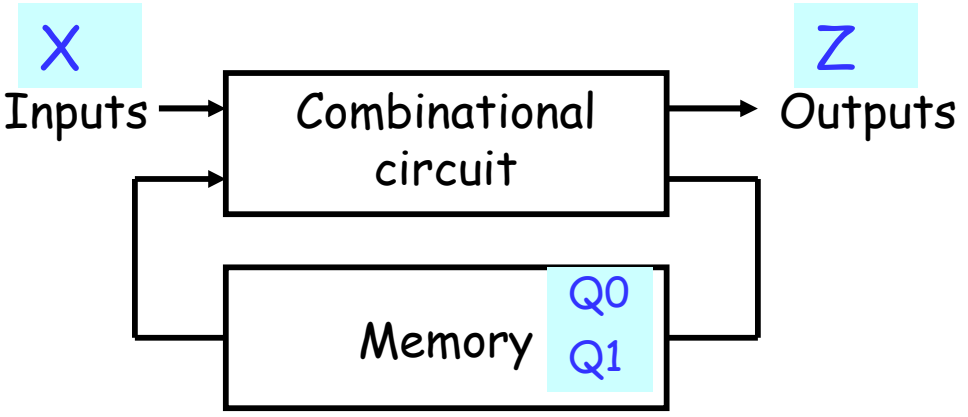
- In **sequential circuit analysis**, we turn some sequential circuit into a state table/state diagram describing the circuit.



An example sequential circuit

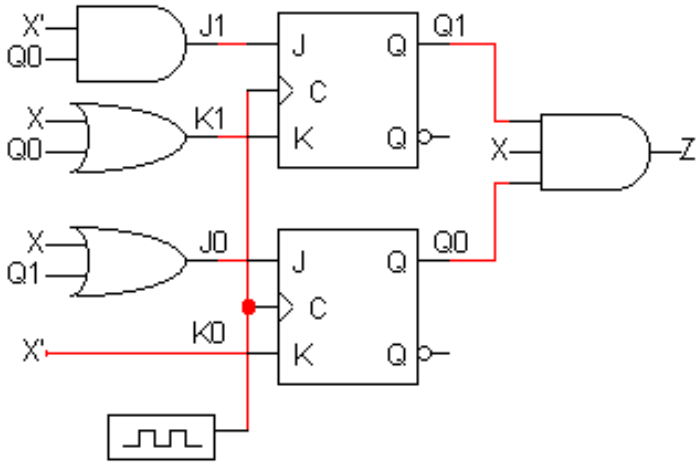
- Here is a sequential circuit with two JK flip-flops. There is one input, X, and one output, Z.
- The values of the flip-flops (Q_1Q_0) form the state, or the memory, of the circuit.
- The flip-flop outputs also go back into the primitive gates on the left. This fits the general sequential circuit diagram at the bottom.





Analyzing our example circuit

- A basic state table for our example circuit is shown below.
- Remember that there is one input X, one output Z, and two flip-flops Q_1Q_0.
- The present state Q_1Q_0 and the input will determine the next state and the output.

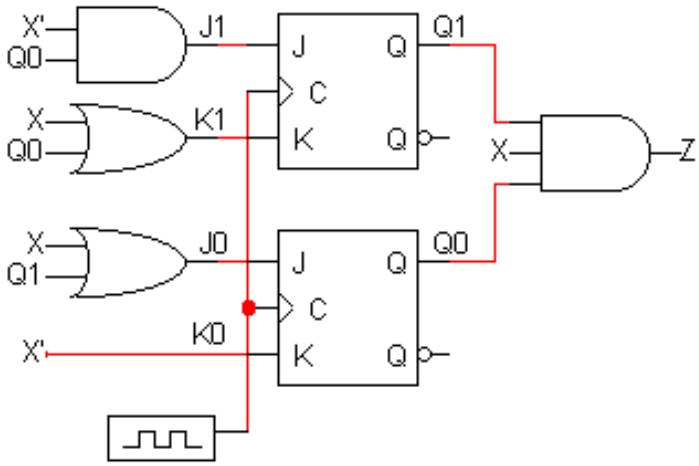


Present State Q_1	Present State Q_0	Inputs X	Next State Q_1	Next State Q_0	Outputs Z
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

The outputs are easy

- The output depends on the current state – Q_0 and Q_1 – as well as the inputs.
- From the diagram, you can see that

$$Z = Q_1 Q_0 X$$



Output at the current time

Present State Q_1	Present State Q_0	Inputs X	Next State Q_1	Next State Q_0	Outputs Z
0	0	0			0
0	0	1			0
0	1	0			0
0	1	1			0
1	0	0			0
1	0	1			0
1	1	0			0
1	1	1			1

Flip-flop input equations

- Finding the next states is harder. To do this, we have to figure out how the flip-flops are changing.
  - **Step 1:** Find Boolean expressions for the flip-flop inputs, i.e. How do the inputs (say, J & K) to the flipflops depend on the current state and input.
  - **Step 2:** Use these expressions to find the actual flip-flop input values for each possible combination of present states and inputs, i.e. fill in the state table (with new intermediate columns)
  - **Step 3:** Use flip-flop characteristic tables or equations to find the next states, based on the flip-flop input values and the present states.

Step 1: Flip-flop input equations

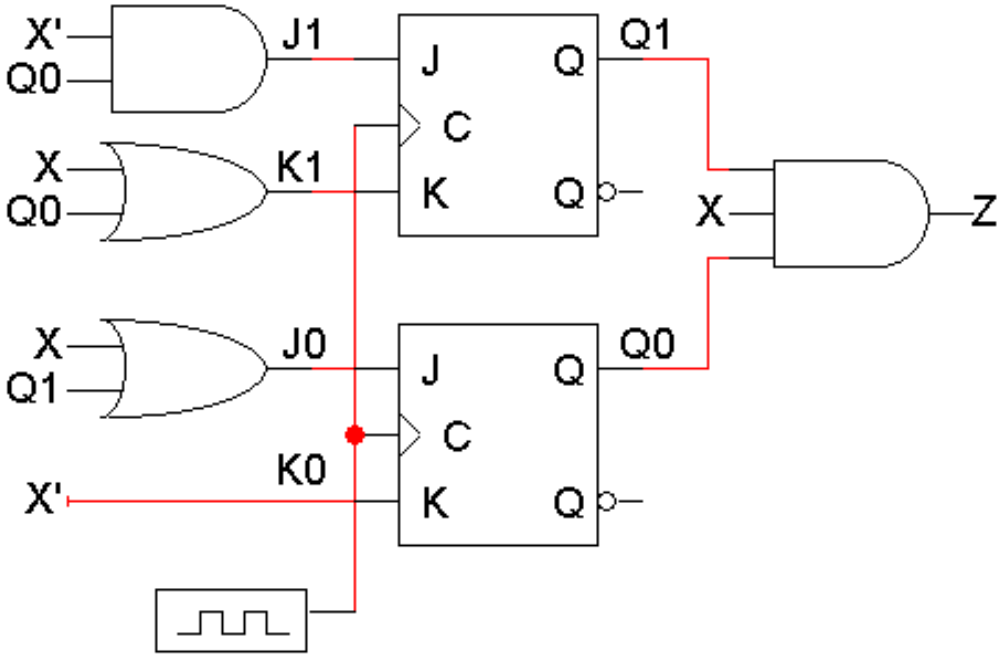
- For our example, the **flip-flop input equations** are:

$$J_1 = X' Q_0$$

$$K_1 = X + Q_0$$

$$J_0 = X + Q_1$$

$$K_0 = X'$$



- JK flip-flops each have *two* inputs, J and K. (D and T flip-flops have one input each.)

Step 2: Flip-flop input values

- With these equations, we can make a table showing J_1, K_1, J_0 and K_0 for the different combinations of present state Q_1Q_0 and input X.

$$J_1 = X' Q_0 \qquad J_0 = X + Q_1$$

$$K_1 = X + Q_0 \qquad K_0 = X'$$

Present State		Inputs	Flip-flop Inputs			
Q_1	Q_0	X	J_1	K_1	J_0	K_0
0	0	0	0	0	0	1
0	0	1	0	1	1	0
0	1	0	1	1	0	1
0	1	1	0	1	1	0
1	0	0	0	0	1	1
1	0	1	0	1	1	0
1	1	0	1	1	1	1
1	1	1	0	1	1	0

Step 3: Find the next states

- Finally, use the JK flip-flop characteristic tables or equations to find the next state of *each* flip-flop, based on its present state and inputs.
- The general JK flip-flop characteristic equation is:

$$Q(t+1) = K'Q(t) + JQ'(t)$$

- In our example circuit, we have two JK flip-flops, so we have to apply this equation to *each* of them:

$$Q_1(t+1) = K_1'Q_1(t) + J_1Q_1'(t)$$

$$Q_0(t+1) = K_0'Q_0(t) + J_0Q_0'(t)$$

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

- We can also determine the next state for each input/current state combination directly from the characteristic table.

Step 3 concluded

- Finally, here are the next states for Q_1 and Q_0, using these equations:

$$Q_1(t+1) = K_1'Q_1(t) + J_1Q_1'(t)$$

$$Q_0(t+1) = K_0'Q_0(t) + J_0Q_0'(t)$$

Present State		Inputs	FF Inputs				Next State	
Q_1	Q_0	X	J_1	K_1	J_0	K_0	Q_1	Q_0
0	0	0	0	0	0	1	0	0
0	0	1	0	1	1	0	0	1
0	1	0	1	1	0	1	1	0
0	1	1	0	1	1	0	0	1
1	0	0	0	0	1	1	1	1
1	0	1	0	1	1	0	0	1
1	1	0	1	1	1	1	0	0
1	1	1	0	1	1	0	0	1

Getting the state table columns straight

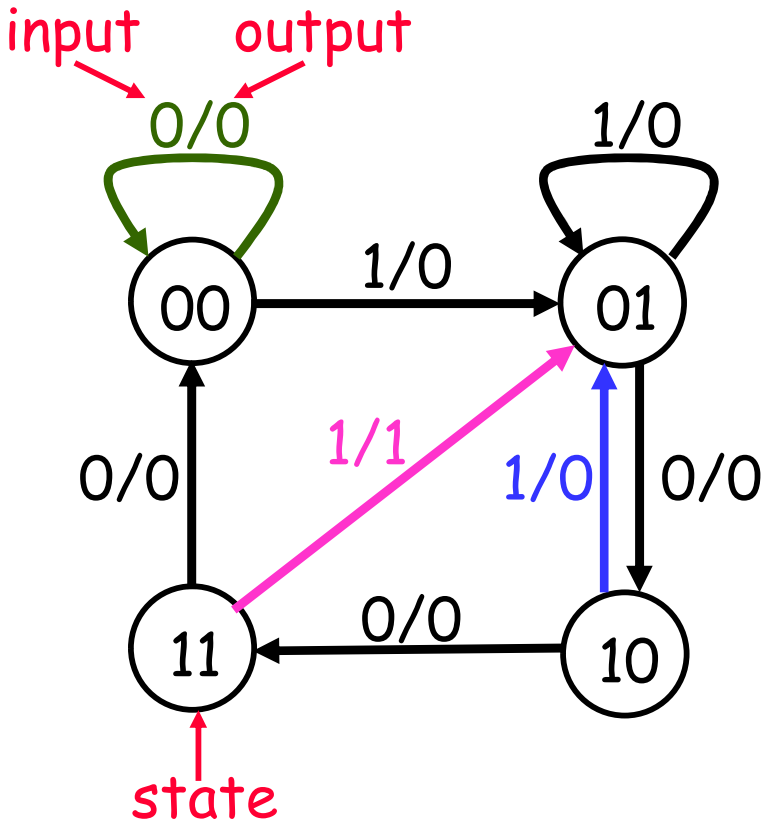
- The table starts with Present State and Inputs.
  - Present State and Inputs yield FF Inputs.
  - Present State and FF Inputs yield Next State, based on the flip-flop characteristic tables.
  - Present State and Inputs yield Output.
- We really only care about FF Inputs in order to find Next State.
- Note: the outputs occur *this cycle* and the next state in *the next cycle*

Present State		Inputs	FF Inputs				Next State		Outputs
Q_1	Q_0	X	J_1	K_1	J_0	K_0	Q_1	Q_0	Z
0	0	0	0	0	0	1	0	0	0
0	0	1	0	1	1	0	0	1	0
0	1	0	1	1	0	1	1	0	0
0	1	1	0	1	1	0	0	1	0
1	0	0	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0	1	0
1	1	0	1	1	1	1	0	0	0
1	1	1	0	1	1	0	0	1	1

State diagrams

- We can also represent the state table graphically with a state diagram.
- A diagram corresponding to our example state table is shown below.

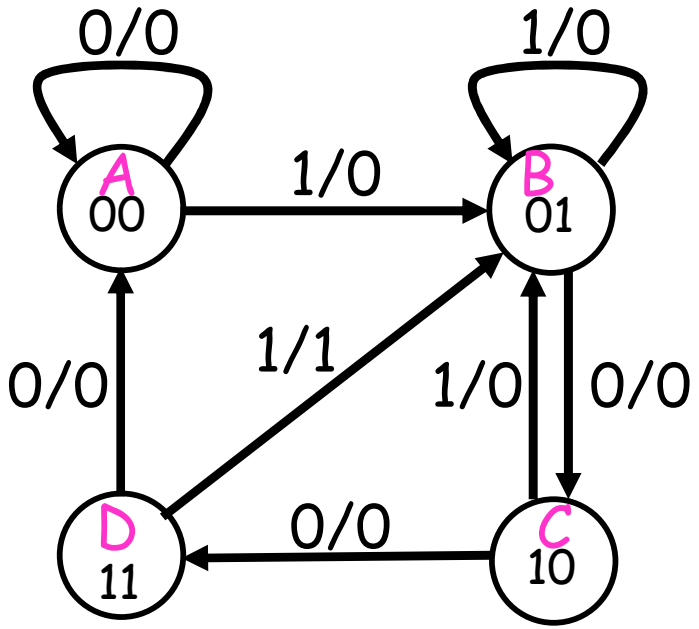
Present State Q_1	Present State Q_0	Inputs X	Next State Q_1	Next State Q_0	Outputs Z
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	0	1	1



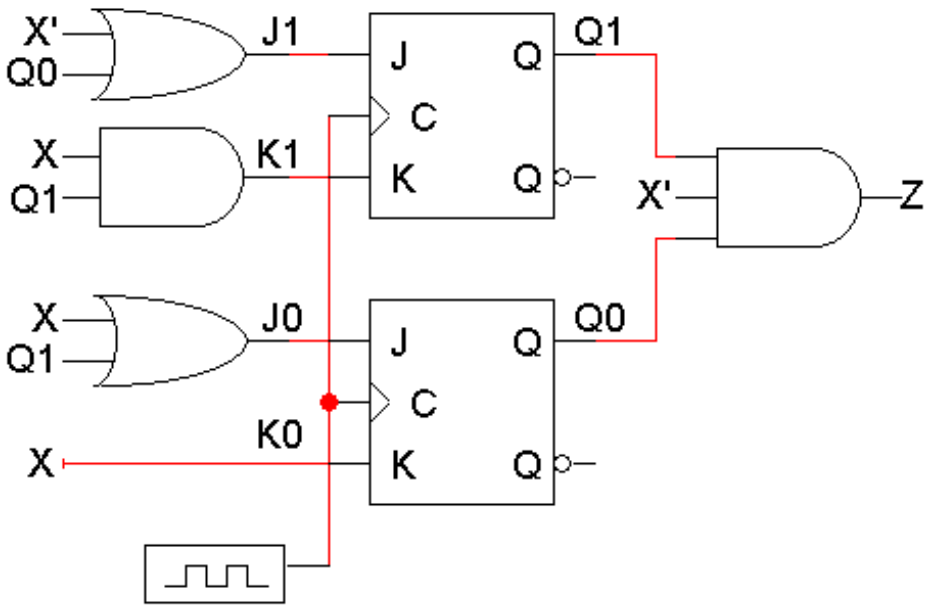
Sizes of state diagrams

- Always check the size of your state diagrams.
  - If there are *n* flip-flops, there should be 2^n nodes in the diagram.
  - If there are *m* inputs, then each node will have 2^m outgoing arrows.
    - From each state
- In our example,
  - We have two flip-flops, and thus four states or nodes.
  - There is one input, so each node has two outgoing arrows.

Present State Q_1	Present State Q_0	Inputs X	Next State Q_1	Next State Q_0	Outputs 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

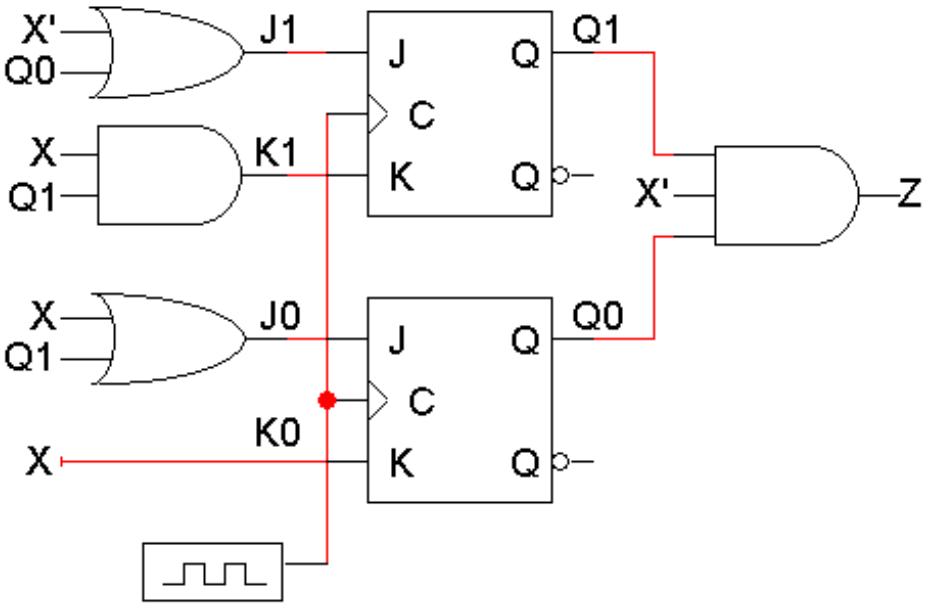


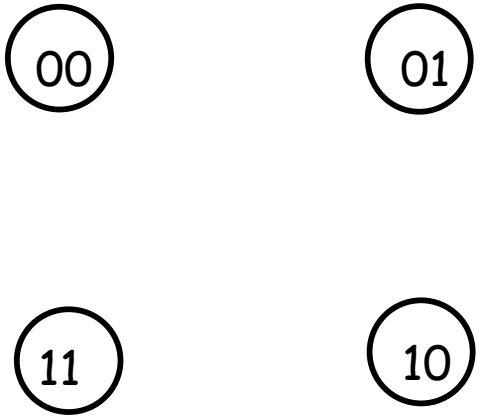
Example



Present State		Inputs	FF Inputs				Next State		Outputs
Q_1	Q_0	X	J_1	K_1	J_0	K_0	Q_1	Q_0	Z
0	0	0							
0	0	1							
0	1	0							
0	1	1							
1	0	0							
1	0	1							
1	1	0							
1	1	1							

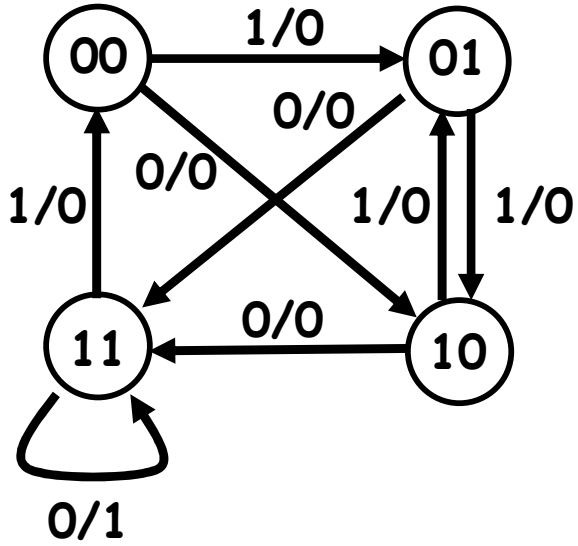
Example





Present State		Inputs	FF Inputs				Next State		Outputs
Q_1	Q_0	X	J_1	K_1	J_0	K_0	Q_1	Q_0	Z
0	0	0	1	0	0	0	1	0	0
0	0	1	0	0	1	1	0	1	0
0	1	0	1	0	0	0	1	1	0
0	1	1	1	0	1	1	1	0	0
1	0	0	1	0	1	0	1	1	0
1	0	1	0	1	1	1	0	1	0
1	1	0	1	0	1	0	1	1	1
1	1	1	1	1	1	1	0	0	0

Example



Present State		Inputs	FF Inputs				Next State		Outputs
Q_1	Q_0	X	J_1	K_1	J_0	K_0	Q_1	Q_0	Z
0	0	0	1	0	0	0	1	0	0
0	0	1	0	0	1	1	0	1	0
0	1	0	1	0	0	0	1	1	0
0	1	1	1	0	1	1	1	0	0
1	0	0	1	0	1	0	1	1	0
1	0	1	0	1	1	1	0	1	0
1	1	0	1	0	1	0	1	1	1
1	1	1	1	1	1	1	0	0	0

Sequential circuit analysis

- To analyze sequential circuits, you have to:
  - Find Boolean expressions for the outputs of the circuit and the flip-flop inputs.
  - Use these expressions to fill in the output and flip-flop input columns in the state table.
  - Finally, use the characteristic equation or characteristic table of the flip-flop to fill in the next state columns.

- The result of sequential circuit analysis is a state table or a state diagram describing the circuit.

Acknowledgments

- Credit is acknowledged where credit is due. Please refer to the full list of references.