CENG-232
Logic Design

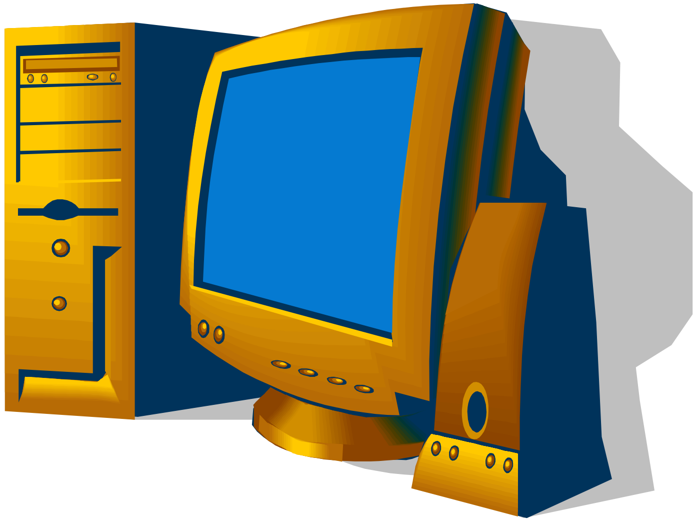
Lecture 5
Synchronous Sequential Logic

Spring 2015 - Uluç Saranlı
saranli@ceng.metu.edu.tr

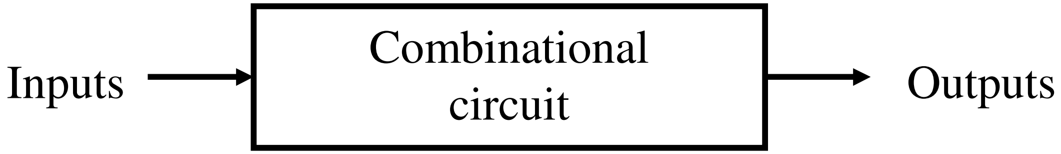
Acknowledgement: Most of the following slides are adapted from Prof. Kale's slides at UIUC, USA.

Latches

- The second part of CENG232 focuses on sequential circuits, where we add memory to the hardware that we’ve already seen.

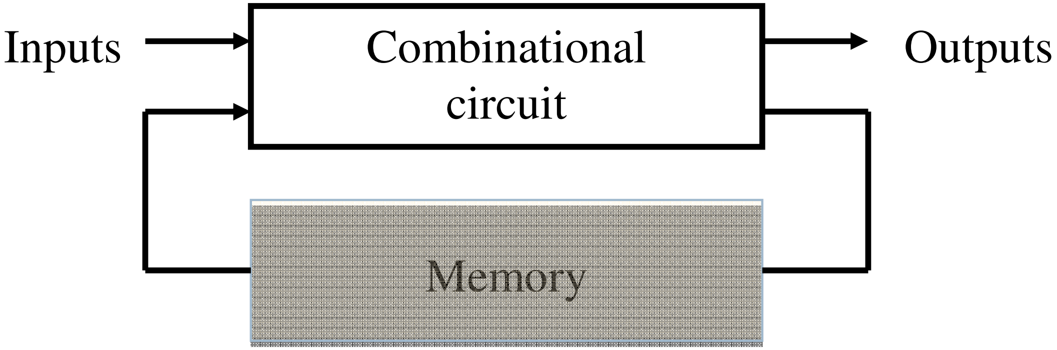


Combinational Circuits



- So far we’ve just worked with combinational circuits, where applying the same inputs always produces the same outputs.
- This corresponds to a mathematical function, where every input has a single, unique output.
- In programming terminology, combinational circuits are similar to “functional programs” that do not contain variables and assignments.

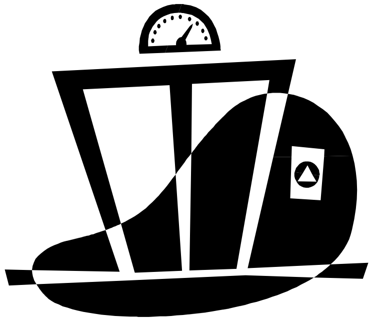
Sequential Circuits

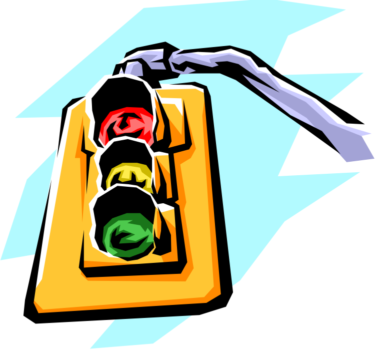


- In contrast, the outputs of a sequential circuit depend on not only the inputs, but also the state, or the current contents of some memory.
- This makes things more difficult to understand, since the same inputs can yield different outputs, depending on what's stored in memory.
- The memory contents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

Examples of Sequential Devices

- Many real-life devices are sequential in nature:
  - Combination locks open if you enter numbers in the right order.
  - Elevators move up or down and open or close depending on the buttons that are pressed on different floors and in the elevator itself.
  - Traffic lights may switch from red to green depending on whether or not a car is waiting at the intersection.
- Most importantly, computers are sequential! For example, key presses and mouse clicks mean different things depending on which program is running and the state of that program.





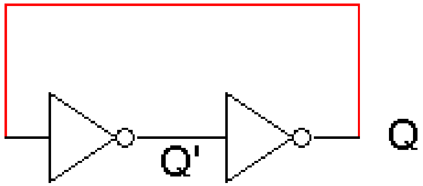
What exactly is memory?

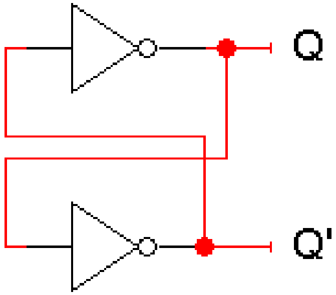
- A memory should have at least three properties.
  - It should be able to hold a value.
  - You should be able to read the value that was saved.
  - You should be able to change the value that’s saved.

- We’ll start with the simplest case, a one-bit memory.
  - It should be able to hold a single bit, 0 or 1.
  - You should be able to read the bit that was saved.
  - You should be able to change the value. Since there’s only a single bit, there are only two choices:
    - Set the bit to 1
    - Reset, or clear, the bit to 0.

The basic idea of "Storage"

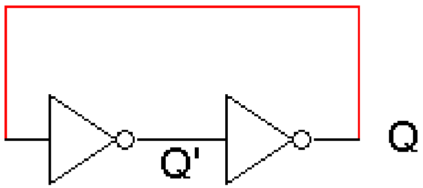
- How can a circuit "remember" anything?
  - … when it is just a bunch of gates that produce outputs according to the inputs?
- The basic idea is to make a loop (feedback), so the circuit outputs are also inputs.
- Here is one initial attempt, shown with two equivalent layouts:

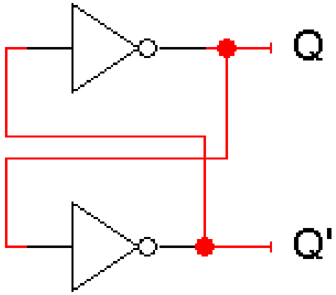




The basic idea of "Storage"

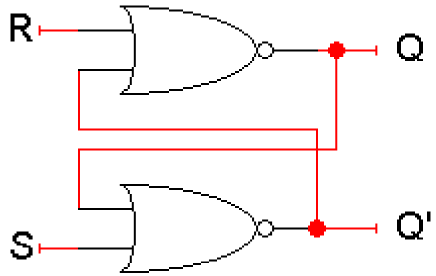
- Does this satisfy the properties of memory?
  - These circuits "remember" Q, because its value never changes. (Similarly, Q' never changes either.)
  - We can also "read" Q, by attaching a probe or another circuit.
  - But we can't change Q! There are no external inputs here, so we can't control whether Q=1 or Q=0.





A really “confusing” circuit

- Let’s use NOR (NAND) gates instead of inverters.
- The “SR latch” here has two inputs S and R, which will let us control the outputs Q and Q’.
  - Study the NAND case!



- Q and Q’ are fed back into the circuit. They’re not only outputs, but also inputs!
- To figure out how Q and Q’ change, we have to look at not only the inputs S and R, but also the current values of Q and Q’:

$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$

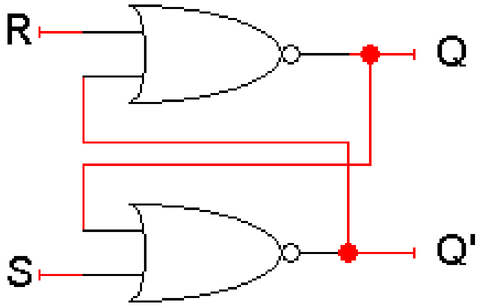
- Let’s see how different input values for S and R affect this thing ...

Storing a value: SR = 00

- What if S = 0 and R = 0?
- The circuit on the right reduces to:

  $Q_{next} = (0 + Q'_{current})' = Q_{current}$

  $Q'_{next} = (0 + Q_{current})' = Q'_{current}$

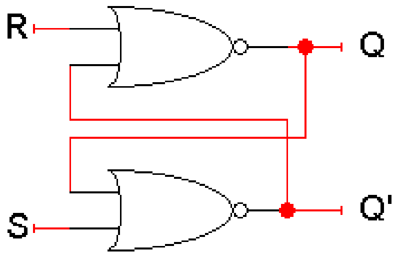


- So when SR = 00, then $Q_{next} = Q_{current}$
  - $Q'_{next} = Q'_{current}$
  - Whatever value Q has, it keeps.
- This is exactly what we need to store values in the latch.
  - “Remain in the same state”

$Q_{next} = (R + Q'_{current})'$

$Q'_{next} = (S + Q_{current})'$

Setting the Latch: SR = 10



- What if S = 1 and R = 0?
- Since S = 1, Q'_{next} is 0, regardless of $Q_{current}$:

  $$Q'_{next} = (1 + Q_{current})' = 0$$

- Then, this new value of Q' goes into the top NOR gate, along with R = 0.

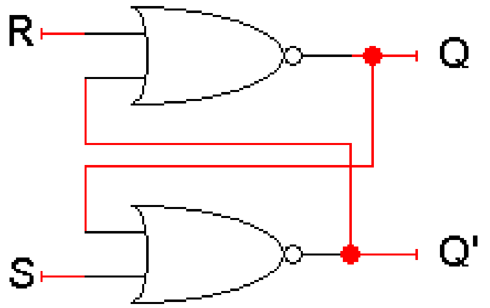
  $$Q_{next} = (0 + 0)' = 1$$

$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$

- So when SR = 10, then Q'_{next} = 0 and Q_{next} = 1.
- This is how you set the latch to 1 (The S input stands for "Set"; R: "Reset").
- Notice that it can take up to two steps (two gate delays) from the time S becomes 1 to the time Q_{next} becomes 1.
- But once Q_{next} becomes 1, the outputs will stop changing. This is a stable state.

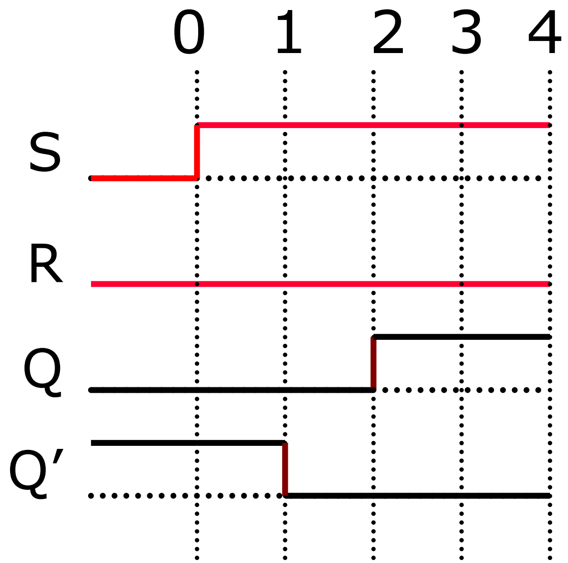
Latch Delays

- Timing diagrams are especially useful in understanding how sequential circuits work.
- Here is a timing diagram which shows an example of how our latch outputs change with inputs SR=10.
  - Suppose that initially, Q = 0 and Q' = 1.
  - Since S=1, Q' will change from 1 to 0 after one NOR-gate delay (marked by vertical lines in the diagram for clarity).
  - This change in Q', along with R=0, causes Q to become 1 after another gate delay.
  - The latch then stabilizes until S or R change again.

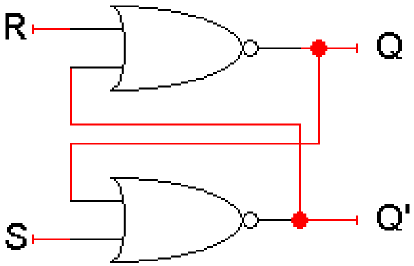


$$Q_{next} = (R + Q'_{current})'$$

$$Q'_{next} = (S + Q_{current})'$$



Resetting the Latch: SR = 01



- What if S = 0 and R = 1?
- Since R = 1, Q_{next} is 0, regardless of $Q_{current}$:

  $Q_{next} = (1 + Q'_{current})' = 0$
- Then, this new value of Q goes into the bottom NOR gate, where S = 0.

  $Q'_{next} = (0 + 0)' = 1$
- So when SR = 01, then …

  $Q_{next} = 0$ and $Q'_{next} = 1$.
- This is how you reset or clear the latch to 0. The R input stands for "reset."
- Again, it can take two gate delays before a change in R propagates to the output Q'_{next}.

$Q_{next} = (R + Q'_{current})'$

$Q'_{next} = (S + Q_{current})'$

SR latches are memories!

- This little table shows that our latch provides everything we need in a memory: we can set it, reset it, and remember the current value.
- The output Q represents the data stored in the latch. It is sometimes called the state of the latch.
- We can expand the table above into a state table, which explicitly shows that the next values of Q and Q’ depend on their current values, as well as on the inputs S and R.

S	R	Q
0	0	No change
0	1	0 (reset)
1	0	1 (set)

Inputs		Current		Next	
S	R	Q	Q’	Q	Q’
0	0	0	1	0	1
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0

SR latches are sequential!

- Notice that for inputs SR = 00, the next value of Q could be either 0 or 1, depending on the current value of Q.
- So the same inputs can yield different outputs, depending on whether the latch was previously set or reset.
- This is very different from the combinational circuits that we've seen so far, where the same inputs always yield the same outputs.

S	R	Q
0	0	No change
0	1	0 (reset)
1	0	1 (set)

Inputs		Current		Next	
S	R	Q	Q'	Q	Q'
0	0	0	1	0	1
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0

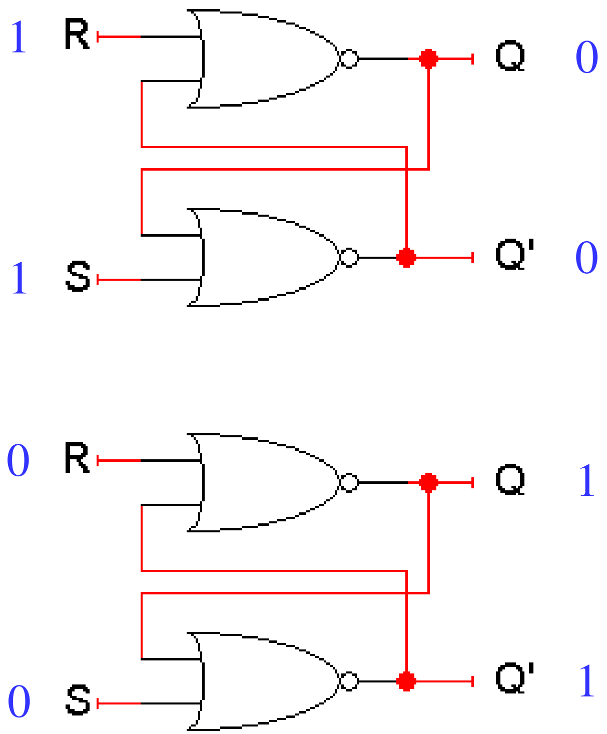
What about SR = 11?

- Both Q_{next} and Q'_{next} will become 0.
  - This contradicts the assumption that Q and Q' are always complements.

$$Q_{next} = (R + Q'_{current})'$$
$$Q'_{next} = (S + Q_{current})'$$

- Another problem is what happens if we then make S = 0 and R = 0 together.

$$Q_{next} = (0 + 0)' = 1$$
$$Q'_{next} = (0 + 0)' = 1$$



What about SR = 11?

- But these new values go back into the NOR gates, and in the next step we get:

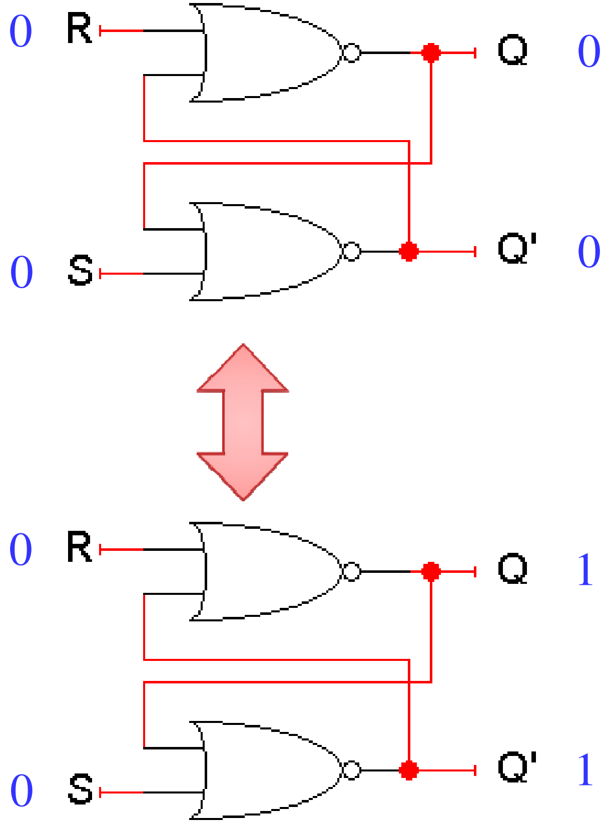
$$Q_{next} = (0 + 1)' = 0$$

$$Q'_{next} = (0 + 1)' = 0$$

- The circuit enters an infinite loop, where Q and Q' cycle between 0 and 1 forever.
- This is actually the worst case, but the moral is don't ever set SR=11!

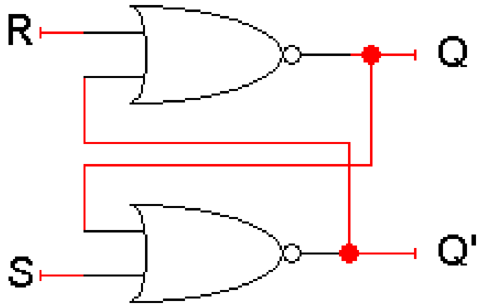
$$Q_{next} = (R + Q'_{current})'$$

$$Q'_{next} = (S + Q_{current})'$$



SR latch

- Completing the picture

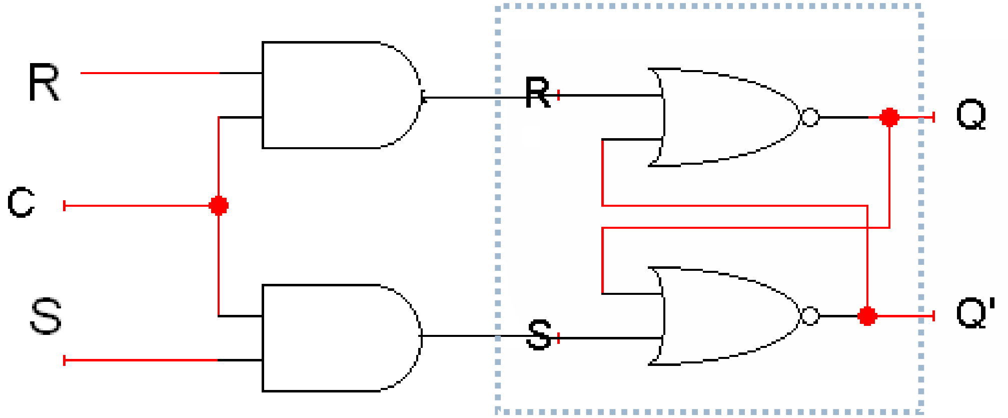


S	R	Q
0	0	No change
0	1	0 (reset)
1	0	1 (set)
1	1	*Avoid!*

An SR latch with a control input

- Here is an SR latch with a control input C.

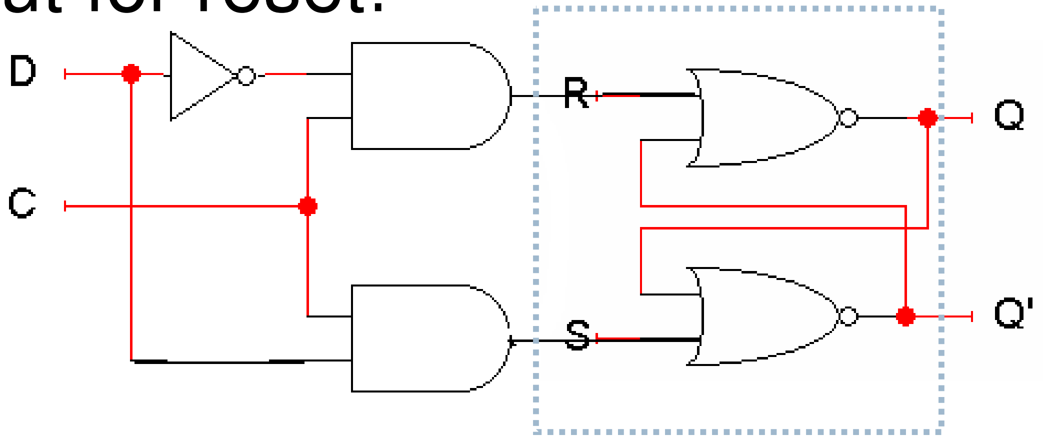
???



- Notice the hierarchical design!
  - The dotted box is the SR latch from the previous slide.
  - The additional AND gates are simply used to generate the correct inputs for the SR latch.
- The control input acts just like an enable.

D latch

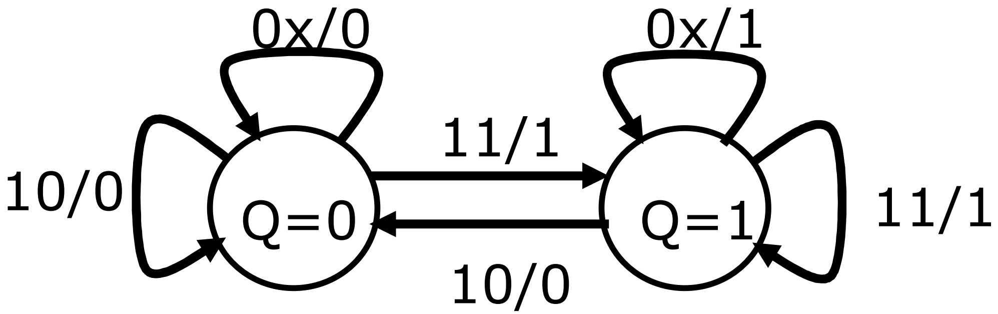
- Finally, a D latch is based on an SR latch. The additional gates generate the S and R signals, based on inputs D ("data") and C ("control").
  - When C = 0, S and R are both 0, so the state Q does not change.
  - When C = 1, the latch output Q will equal the input D.
- No more messing with one input for set and another input for reset!



C	D	Q
0	x	No change
1	0	0
1	1	1

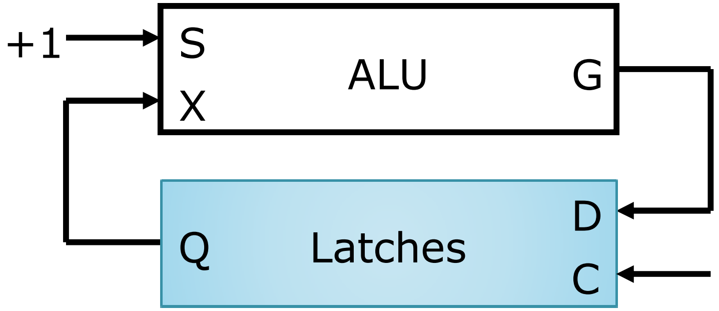
Sequential circuits and state diagrams

- To describe combinational circuits, we used Boolean expressions and truth tables. With sequential circuits, we can still use expression and tables, but we can also use another form called a state diagram.
- We draw one node for each state that the circuit can be in. Latches have only two states: Q=0 and Q=1.
- Arrows between nodes are labeled with “input/output” and indicate how the circuit changes states and what its outputs are. In this case the state and the output are the same.
- Here’s a state diagram for a D latch with inputs D and C.



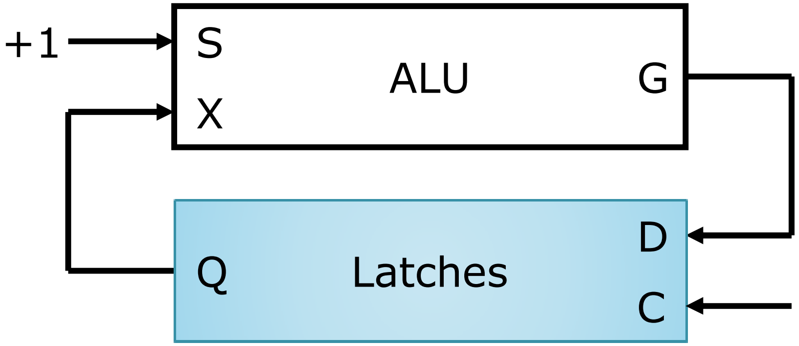
Using latches in real life

- We can connect some latches, acting as memory, to an ALU (Arithmetic Logic Unit).



- Let's say these latches contain some value that we want to increment.
  - The ALU should read the current latch value.
  - It applies the "G = X + 1" operation.
  - The incremented value is stored back into the latches.
- At this point, we have to stop the cycle, so the latch value doesn't get incremented again by accident.
- One convenient way to break the loop, is to disable the latches.

The problem with latches



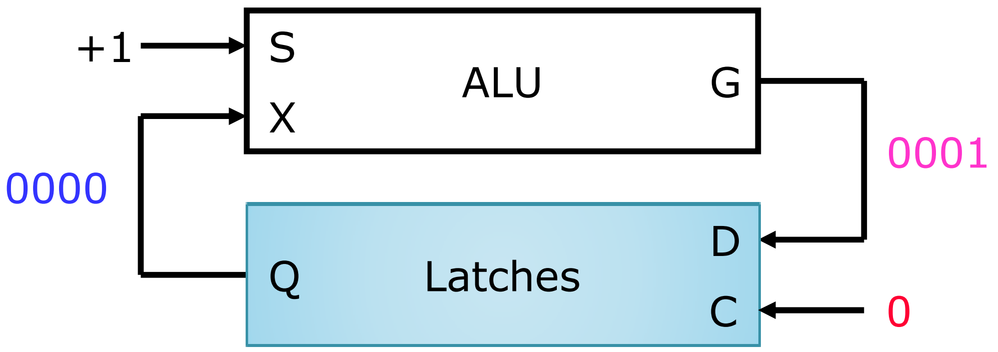
- The problem is exactly when to disable the latches. You have to wait long enough for the ALU to produce its output, but no longer.
  - Different ALU operations have different delays. For instance, arithmetic operations might go through an adder, whereas logical operations don't.
  - Changing the ALU implementation, such as using a carry-lookahead adder instead of a ripple-carry adder, also affects the delay.
- In general, it's very difficult to know how long operations take, and how long latches should be enabled for.

Flip-Flops (FFs)

- We'll address some of the shortcomings of latches by using them to build flip-flops.
  - Adding a "clock" signal helps circuits synchronize their actions.
  - We'll also need to disable memory units quickly.

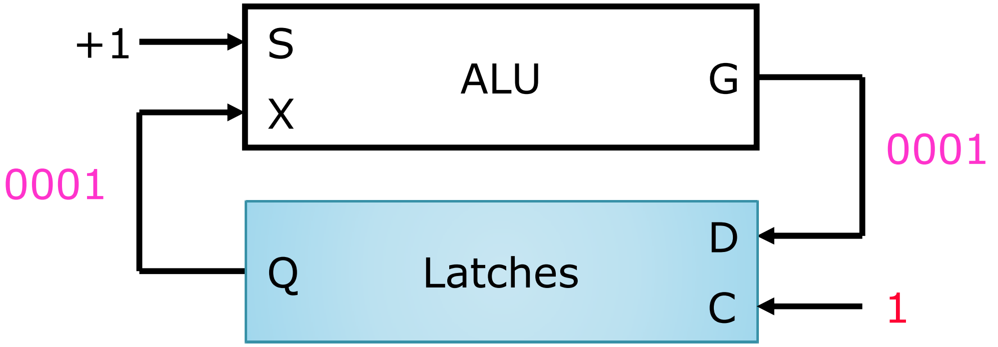
Making latches work right

- Our example used latches as memory for an ALU.
  - Let’s say there are four latches initially storing 0000.
  - We want to use an ALU to increment that value to 0001.
- Normally the latches should be disabled, to prevent unwanted data from being accidentally stored.
  - In our example, the ALU can read the current latch contents, 0000, and compute their increment, 0001.
  - But the new value cannot be stored back while the latch is disabled.

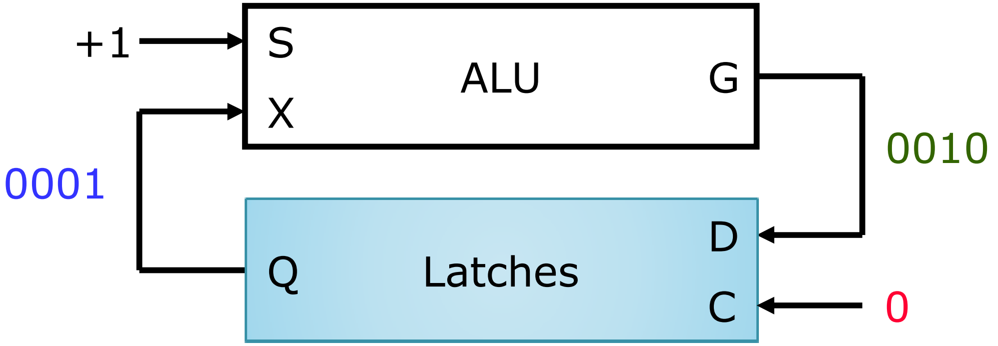


Writing to the latches

- After the ALU has finished its increment operation, the latch can be enabled, and the updated value is stored.



- The latch must be quickly disabled again, before the ALU has a chance to read the new value 0001 and produce a new result 0010.



Two main issues

- So to use latches correctly within a circuit, we have to:
  - Keep the latches disabled until new values are ready to be stored.
  - Enable the latches just long enough for the update to occur.
- There are two main issues we need to address:
  - How do we know exactly when the new values are ready?

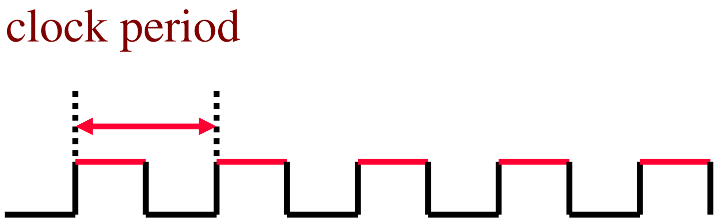
    We'll add another signal to our circuit. When this new signal becomes 1, the latches will know that the ALU computation has completed and data is ready to be stored.

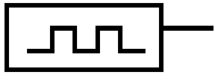
  - How can we enable and then quickly disable the latches?

    This can be done by combining latches together in a special way, to form what are called flip-flops.

Clocks and synchronization

- A clock is a special device that whose output continuously alternates between 0 and 1.

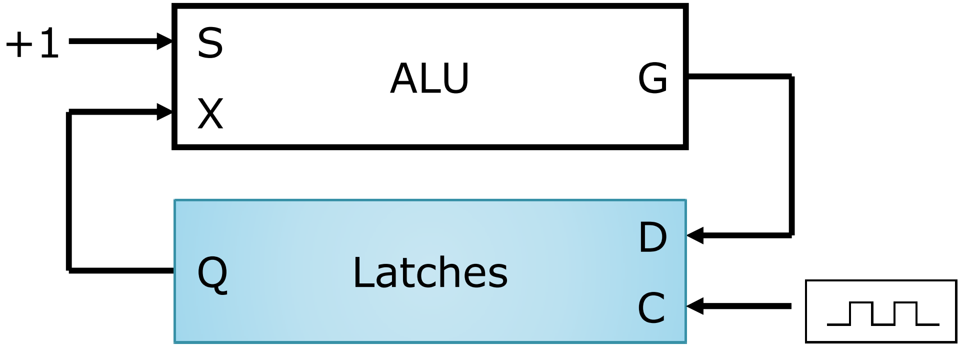




- The time it takes the clock to change from 1 to 0 and back to 1 is called the clock period, or clock cycle time (T).
- The clock frequency (f) is the inverse of the clock period (T = 1/f). The unit of measurement for frequency is the hertz.
- Clocks are often used to synchronize circuits.
  - They generate a repeating, predictable pattern of 0s and 1s that can trigger certain events in a circuit, such as writing to a latch.
  - If several circuits share a common clock signal, they can coordinate their actions with respect to one another.
- This is similar to how humans use real clocks for synchronization.

Synchronizing our example

- We can use a clock to synchronize our latches with the ALU.
  - The clock signal is connected to the latch control input C.
  - The clock controls the latches. When it becomes 1, the latches will be enabled for writing.



- The clock period must be set appropriately for the ALU.
  - It should not be too short. Otherwise, the latches will start writing before the ALU operation has finished.
  - It should not be too long either. Otherwise, the ALU might produce a new result that will accidentally get stored, as we saw before.
- The faster the ALU runs, the shorter the clock period can be.

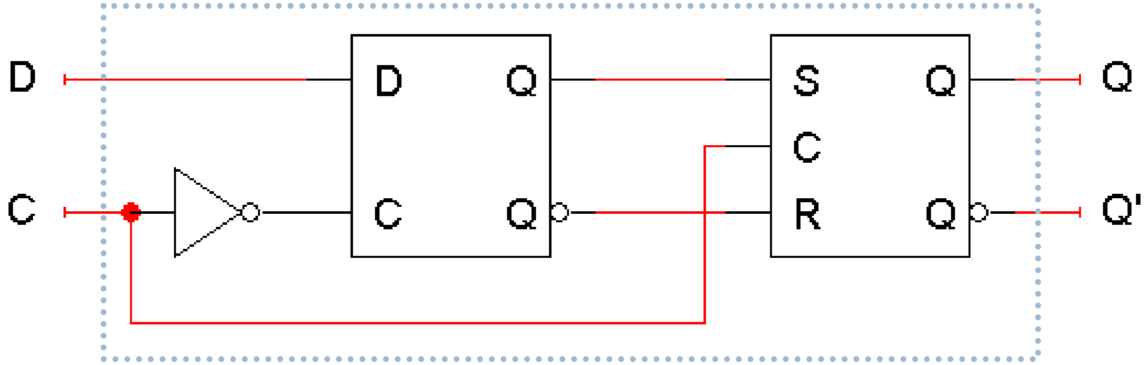
More about clocks

- Clocks are used extensively in computer architecture.
- All processors run with an internal clock.
  - Modern chips can run at frequencies around 2.8 GHz.
  - This works out to a cycle time as little as 0.36 ns!

- Memory modules are often rated by their clock speeds too. Examples include “PC133” and “PC800” memory.
- Be careful... higher frequencies do not always mean faster machines!
  - You also have to consider how much work is actually being done during each clock cycle.
  - How much stuff can really get done in just 0.36 ns?
  - More in CENG331.

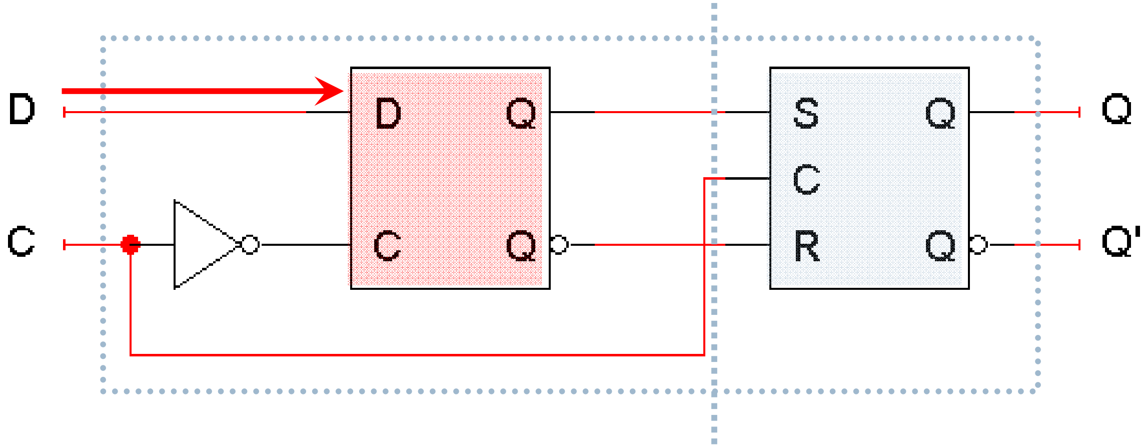
Flip-Flops

- The second issue was how to enable a latch for just an instant.
- Here is the internal structure of a D flip-flop.
  - The flip-flop inputs are C and D; and the outputs are Q and Q’.
  - The D latch on the left is the master, while the SR latch on the right is called the slave.



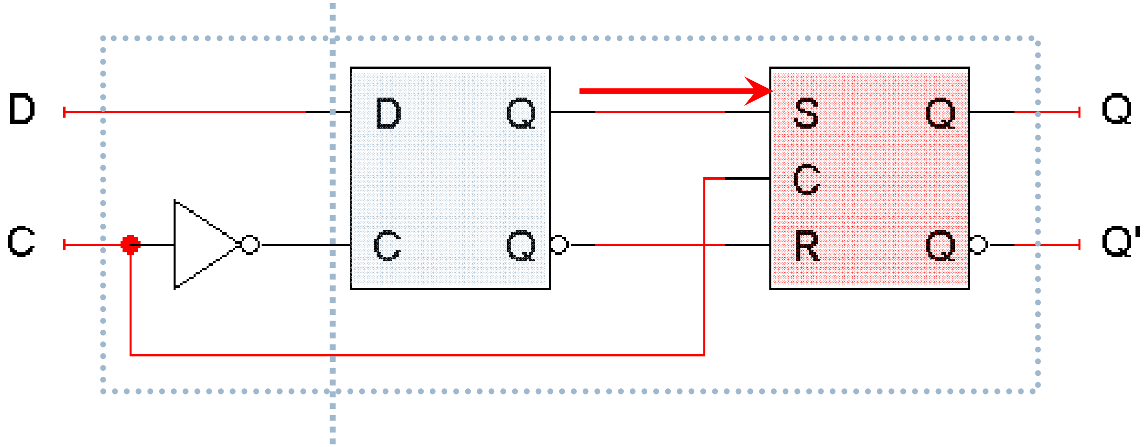
- Note the layout here.
  - The flip-flop input D is connected directly to the master latch.
  - The master latch output goes to the slave.
  - The flip-flop outputs come directly from the slave latch.

D flip-flops when C=0



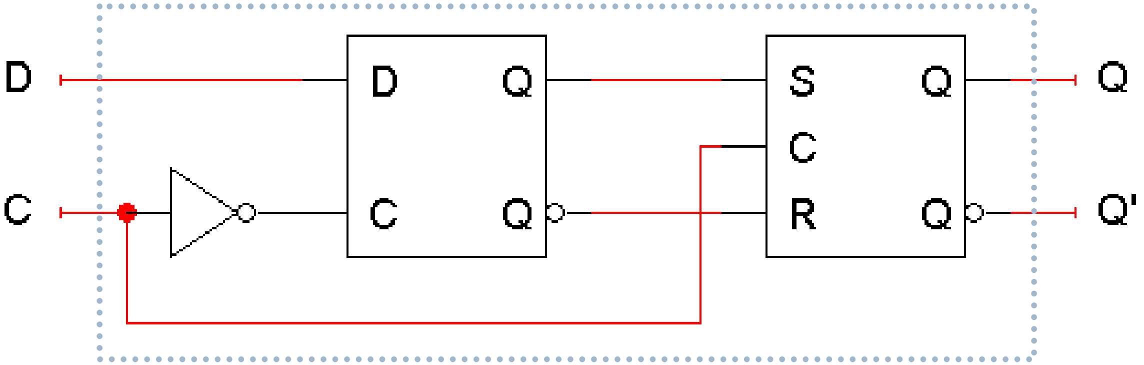
- The D flip-flop's control input C enables either the D latch or the SR latch, but not both.
- When C = 0:
  - The master latch is enabled, and it monitors the flip-flop input D. Whenever D changes, the master's output changes too.
  - The slave is disabled, so the D latch output has no effect on it. Thus, the slave just maintains the flip-flop's current state.

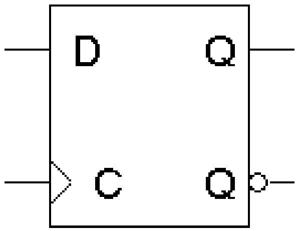
D flip-flops when C=1



- As soon as C becomes 1,
  - The master is disabled. Its output will be the last D input value seen just before C became 1.
  - Any subsequent changes to the D input while C = 1 have no effect on the master latch, which is now disabled.
  - The slave latch is enabled. Its state changes to reflect the master's output, which again is the D input value from right when C became 1.

Positive edge triggering



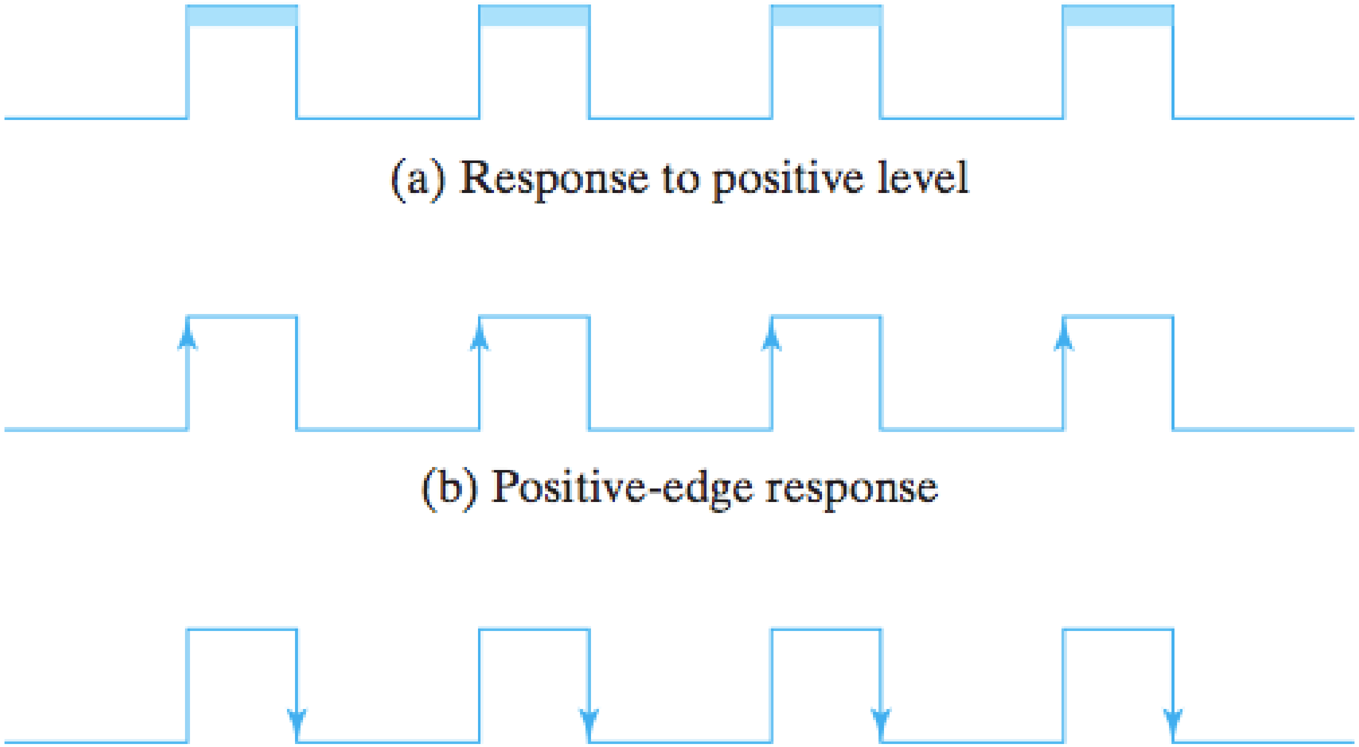


- This is called a positive edge-triggered flip-flop.
  - The flip-flop output Q changes only after the positive edge of C.
  - The change is based on the flip-flop input values that were present right at the positive edge of the clock signal.
- The D flip-flop's behavior is similar to that of a D latch except for the positive edge-triggered nature, which is not explicit in this table.

C	D	Q
0	x	No change
1	0	0 (reset)
1	1	1 (set)

Responding to a Clock

- Possible types of responding to clock

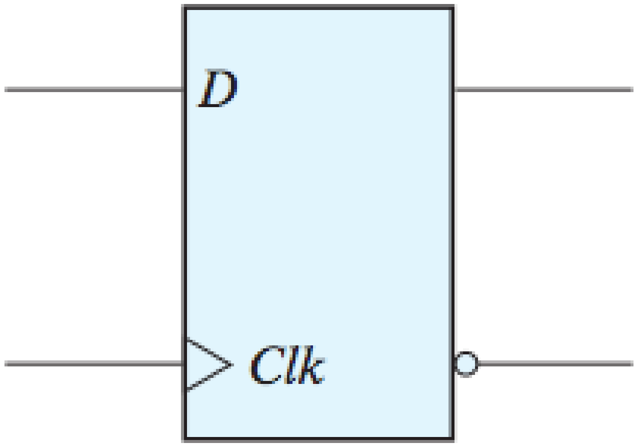


(a) Response to positive level

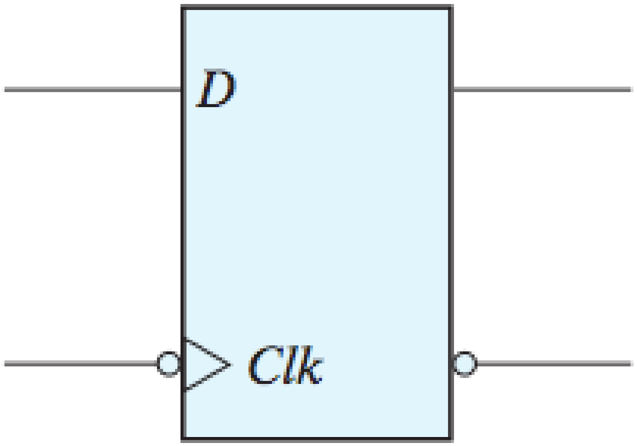
(b) Positive-edge response

(c) Negative-edge response

FF Symbols



(a) Positive-edge



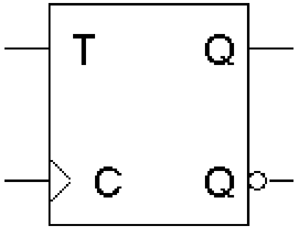
(a) Negative-edge

Direct inputs

- One last thing to worry about …
  - What is the starting value of Q?
- We could set the initial value synchronously, at the next positive clock edge, but this actually makes circuit design more difficult.
- Instead, most flip-flops provide direct, or asynchronous, inputs that let you immediately set or clear the state.
  - You would “reset” the circuit once, to initialize the flip-flops.
  - The circuit would then begin its regular, synchronous operation.

Flip-flop variations (T)

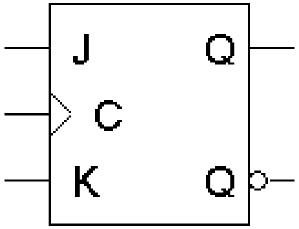
- A T flip-flop can only maintain or complement its current state.



C	T	Q_{next}
0	x	No change
1	0	No change
1	1	$Q'_{current}$

Flip-flop variations (JK)

- We can make different versions of flip-flops based on the D flip-flop, just like we made different latches based on the S’R’ latch.
- A JK flip-flop has inputs that act like S and R, but the inputs JK=11 are used to complement the flip-flop’s current state.



C	J	K	Q_{next}
0	x	x	No change
1	0	0	No change
1	0	1	0 (reset)
1	1	0	1 (set)
1	1	1	$Q'_{current}$

Characteristic Tables

- The tables that we’ve made so far are called characteristic tables.
  - They show the next state Q(t+1) in terms of the current state Q(t) and the inputs.
  - For simplicity, the control input C is not usually listed.
  - Again, these tables don’t indicate the positive edge-triggered behavior of the flip-flops that we’ll be using.

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

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

T	Q(t+1)	Operation
0	Q(t)	No change
1	Q’(t)	Complement

Characteristic Equations

- We can also write characteristic equations, where the next state Q(t+1) is defined in terms of the current state Q(t) and inputs.

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

$$Q(t+1) = D$$

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

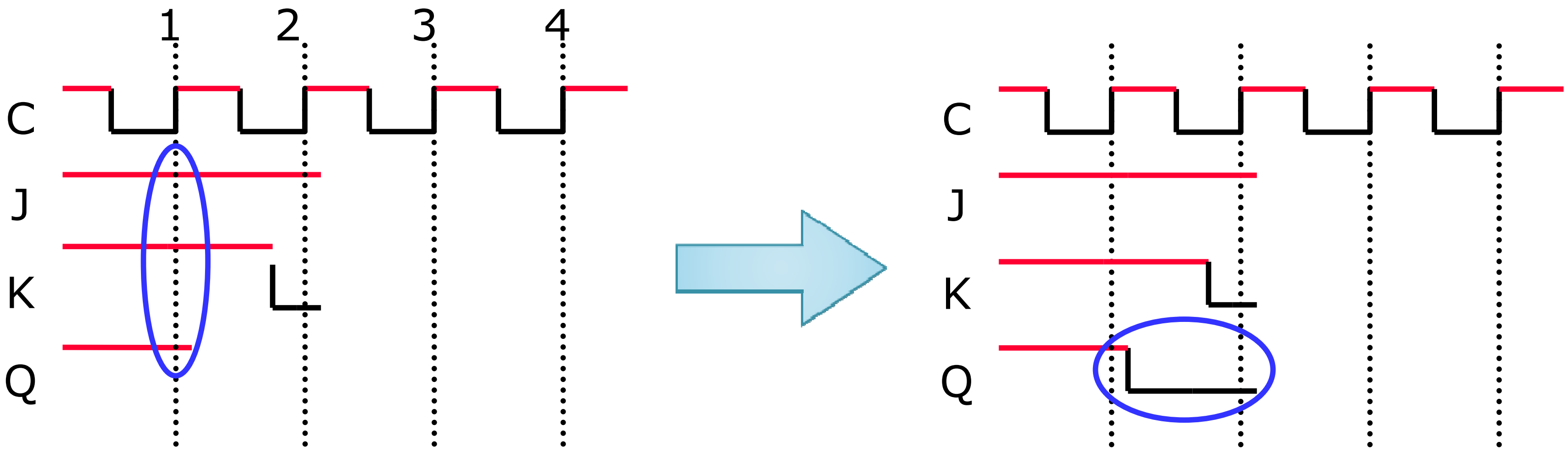
$$Q(t+1) = K'.Q(t) + J.Q'(t)$$

T	Q(t+1)	Operation
0	Q(t)	No change
1	Q'(t)	Complement

$$\begin{aligned} Q(t+1) &= T'.Q(t) + T.Q'(t) \\ &= T \oplus Q(t) \end{aligned}$$

Flip flop timing diagrams

- "Present state" and "next state" are relative terms.
- In the example JK flip-flop timing diagram on the left, you can see that at the first positive clock edge, J=1, K=1 and Q(1) = 1.
- We can use this information to find the "next" state,
  - Q(2) = Q(1)'.
- Q(2) appears right after the first positive clock edge, as shown on the right. It will not change again until after the second clock edge.

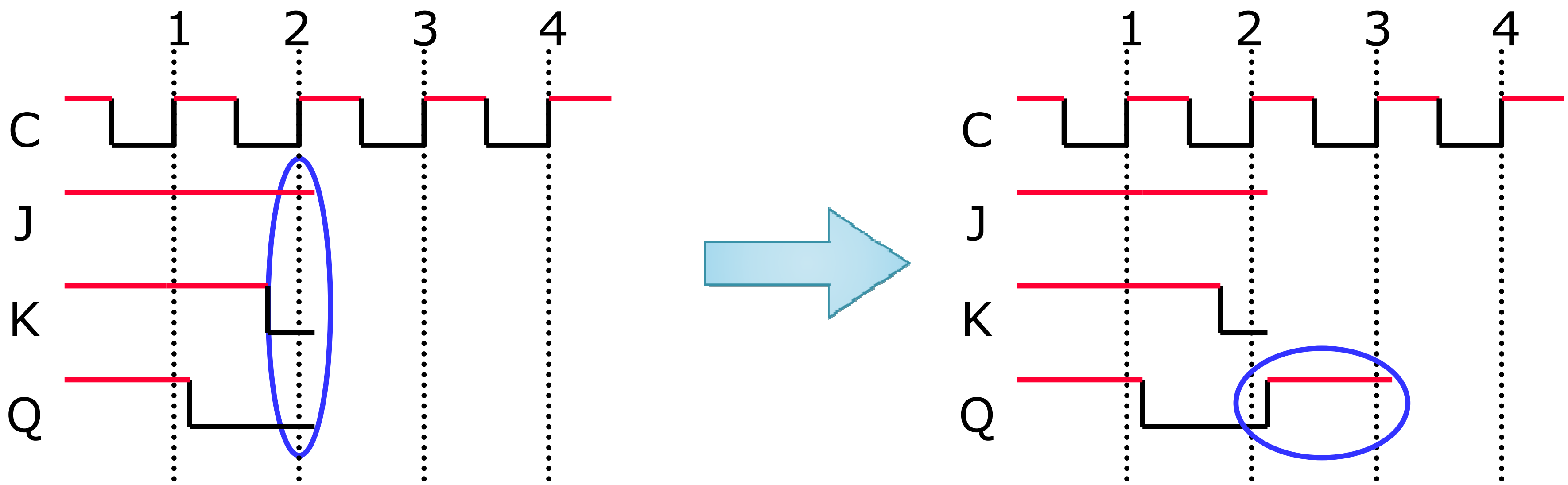


These values at clock cycle 1…

… determine the "next" Q

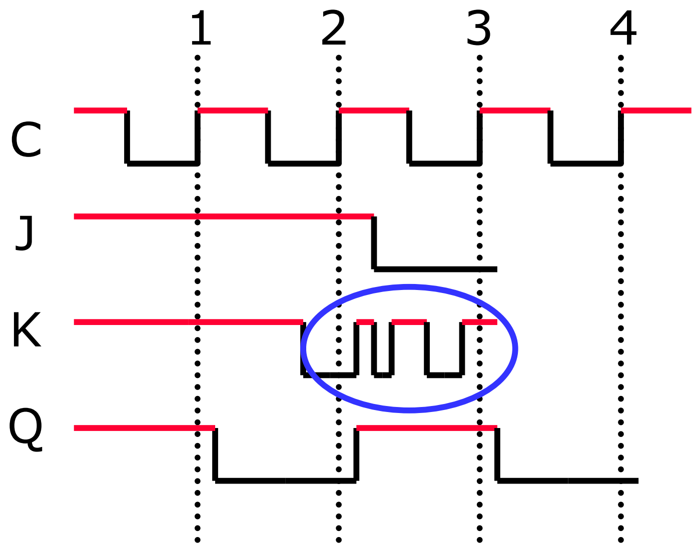
“Present” and “Next” are relative

- Similarly, the values of J, K and Q at the second positive clock edge can be used to find the value of Q during the third clock cycle.
- When we do this, Q(2) is now referred to as the “present” state, and Q(3) is now the “next” state.



Positive edge triggered

- One final point to repeat: the flip-flop outputs are affected only by the input values at the positive edge.
  - In the diagram below, K changes rapidly between the second (2) and third (3) positive edges.
  - But it's only the input values at the third clock edge (K=1, and J=0 and Q=1) that affect the next state, so here Q changes to 0.
- This is a fairly simple timing model. In real life there are "setup times" and "hold times" to worry about as well, to account for internal and external delays.

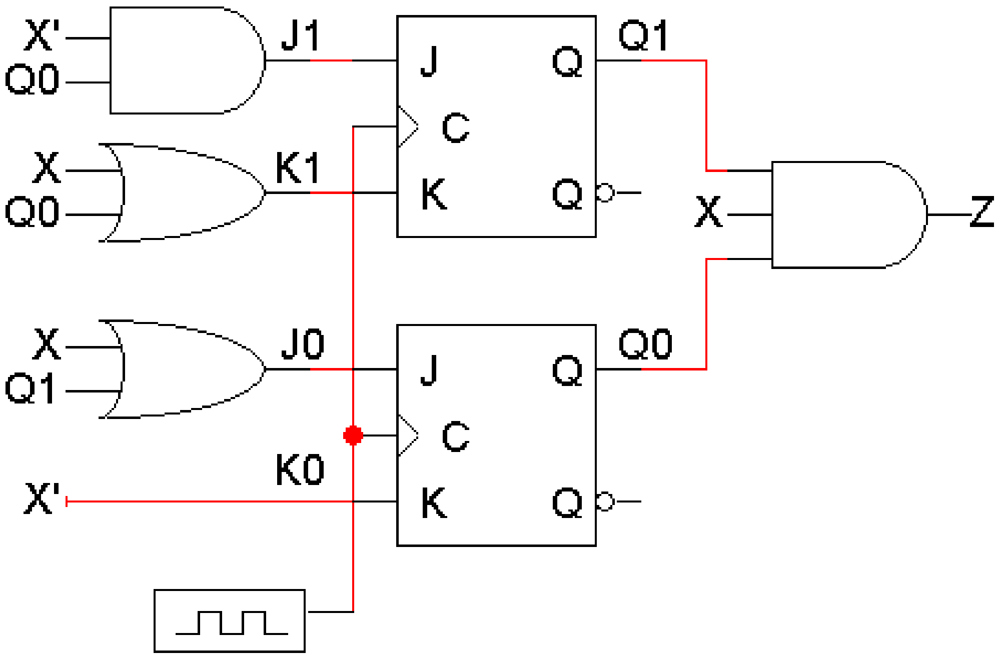


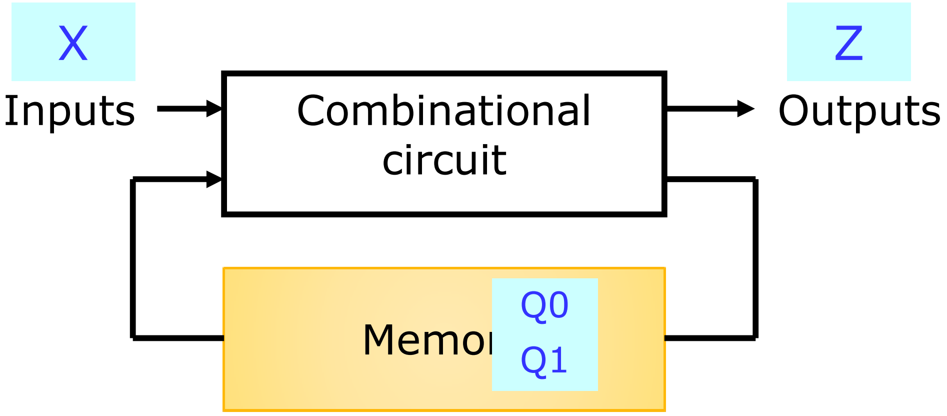
Summary

- To use memory in a larger circuit, we need to:
  - Keep the latches disabled until new values are ready to be stored.
  - Enable the latches just long enough for the update to occur.
- A clock signal is used to synchronize circuits. The cycle time reflects how long combinational operations take.
- Flip-flops further restrict the memory writing interval, to just the positive edge of the clock signal.
  - This ensures that memory is updated only once per clock cycle.
  - There are several different kinds of flip-flops, but they all serve the same basic purpose of storing bits.
- Next, we'll talk about how to analyze and design sequential circuits that use flip-flops as memory.

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 (Q1Q0) 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.




X

Inputs

Combinational circuit

Z

Outputs

Memor

Q0

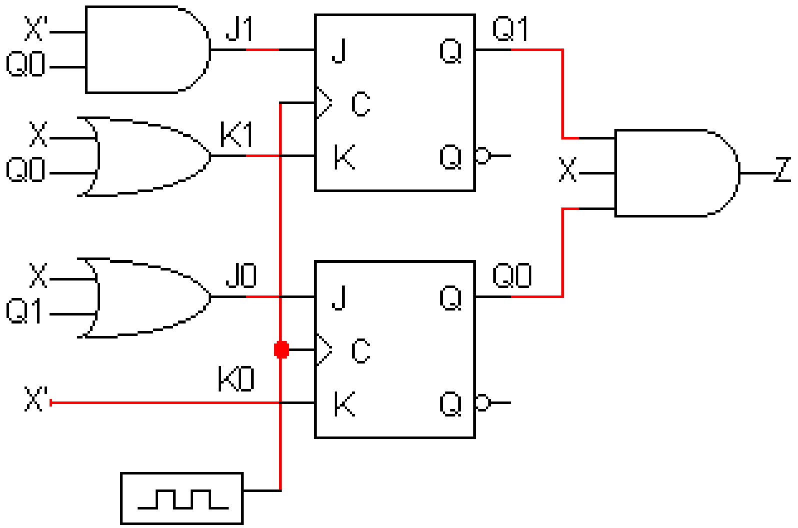
Q1

Analyzing a sequential circuit

- For a combinational circuit we could find a truth table, which shows how the outputs are related to the inputs.
- A state table is the sequential analog of a truth table. It shows inputs and current states on the left, and outputs and next states on the right.
  - For a sequential circuit, the outputs are dependent upon not only the inputs, but also the current state of the flip-flops.
  - In addition to finding outputs, we also need to find the state of the flip-flops on the next clock cycle.

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 Q1Q0.
- The present state Q1Q0 and the input will determine the next state and the output.



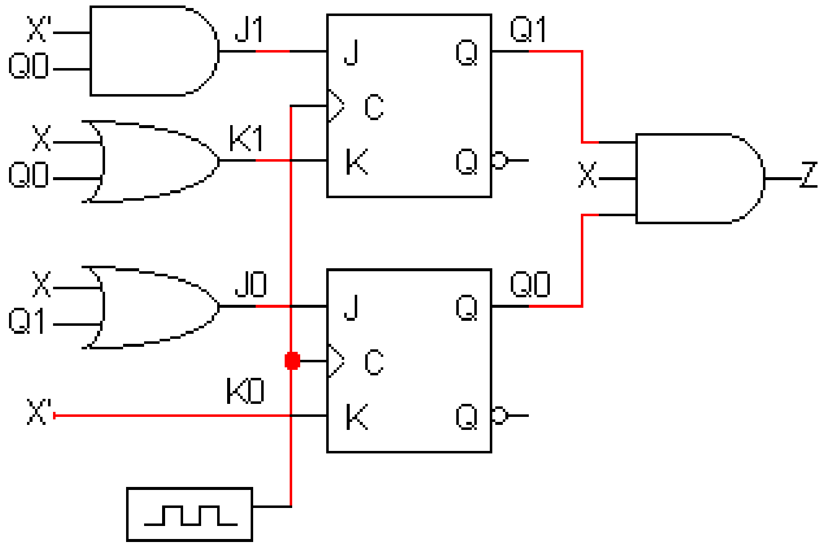
Present State		Inputs	Next State		Outputs
Q_1	Q_0	X	Q_1	Q_0	Z
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

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

  $Z = Q_1Q_0X$

  (Output at the current time)



Present State		Inputs	Next State		Outputs
Q_1	Q_0	X	Q_1	Q_0	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 flip-flops

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

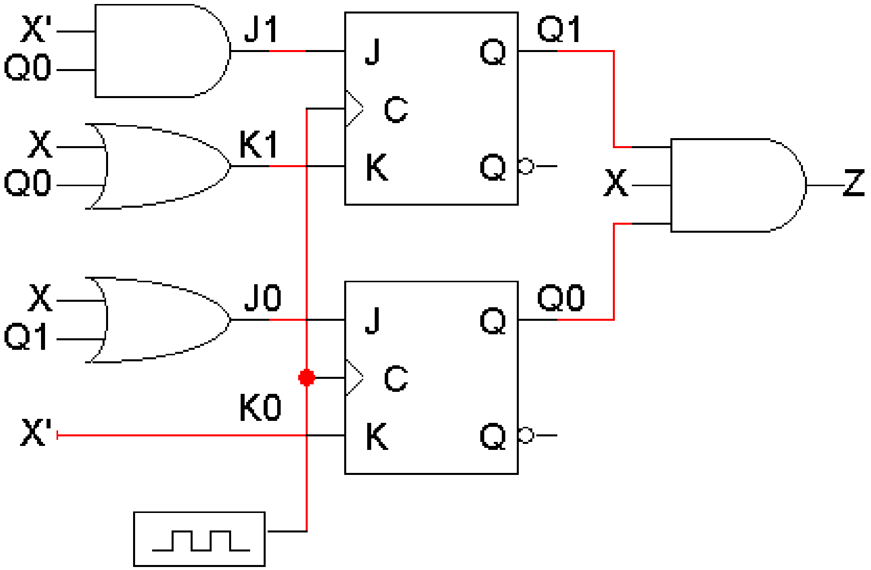
- For our example, the flip-flop input equations are:

  J1 = X' Q0

  K1 = X + Q0

  J0 = X + Q1

  K0 = 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 J1, K1, J0 and K0 for the different combinations of present state Q1Q0 and input X.

J1 = X' Q0 J0 = X + Q1

K1 = X + Q0 K0 = 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:

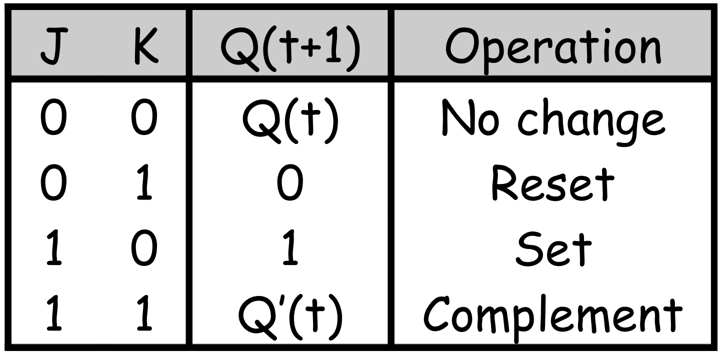
$$Q(t+1) = K'Q(t) + JQ'(t)$$

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

$$Q1(t+1) = K1'Q1(t) + J1Q1'(t)$$

$$Q0(t+1) = K0'Q0(t) + J0Q0'(t)$$

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

Step 3 Concluded

- Finally, here are the next states for Q1 and Q0, using these equations or the characteristic table:

$$Q1(t+1) = K1'Q1(t) + J1Q1'(t)$$

$$Q0(t+1) = K0'Q0(t) + J0Q0'(t)$$

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

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

Spring'12
232 - Logic Design
Page 54

Getting state table columns straight

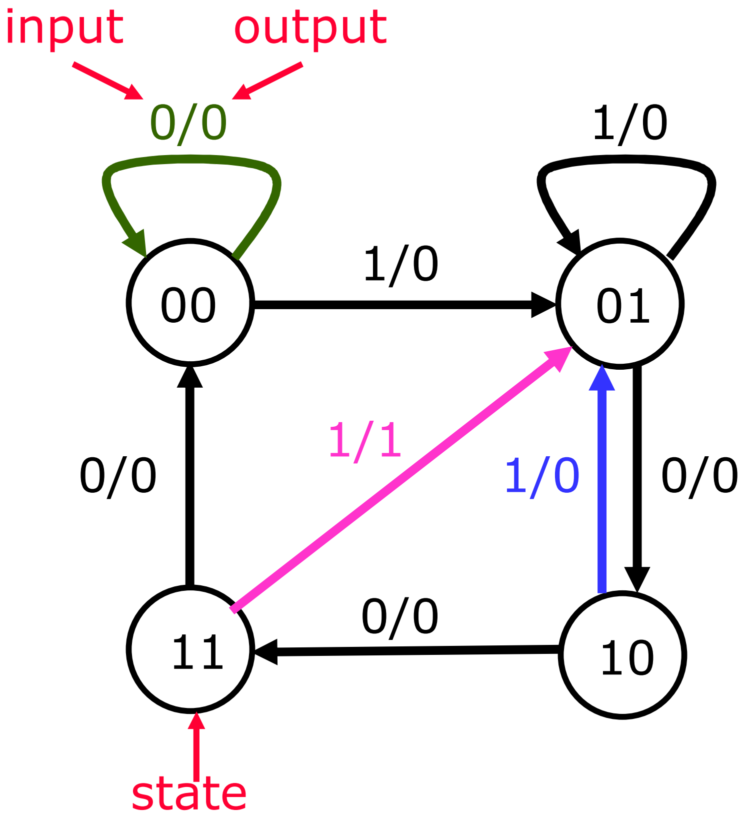
- The table starts with Present State and Inputs.
  - Present State and Inputs yield FF Inputs.
  - Present State and FF Inputs yield Next State, based on the flip-flop characteristic tables.
  - Present State and Inputs yield Output.
- We really only care about FF Inputs in order to find Next State.

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.
- State Encoding! (be careful)

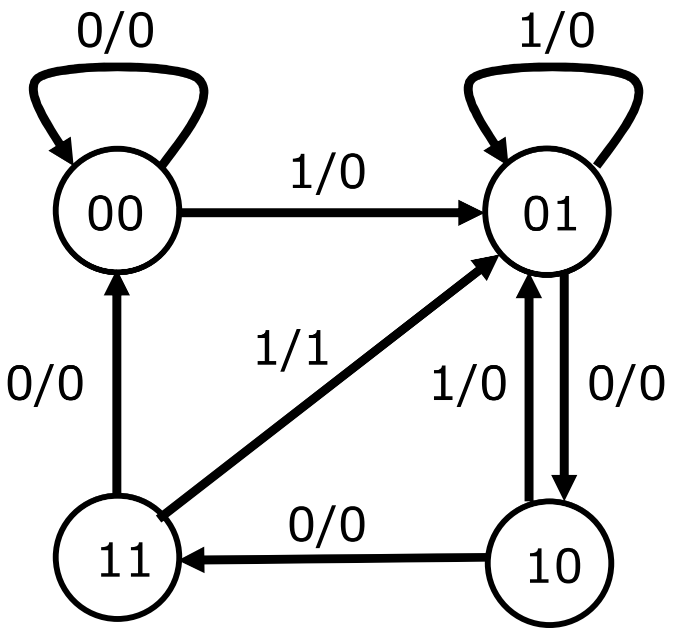
Present State		Inputs	Next State		Outputs
Q_1	Q_0	X	Q_1	Q_0	Z
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	0	1	1



Sizes of State Diagrams

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

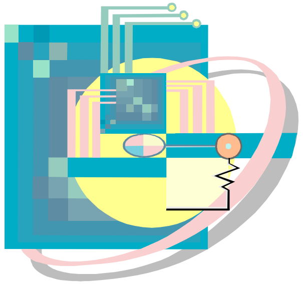
Present State		Inputs	Next State		Outputs
Q_1	Q_0	X	Q_1	Q_0	Z
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	0	1	1



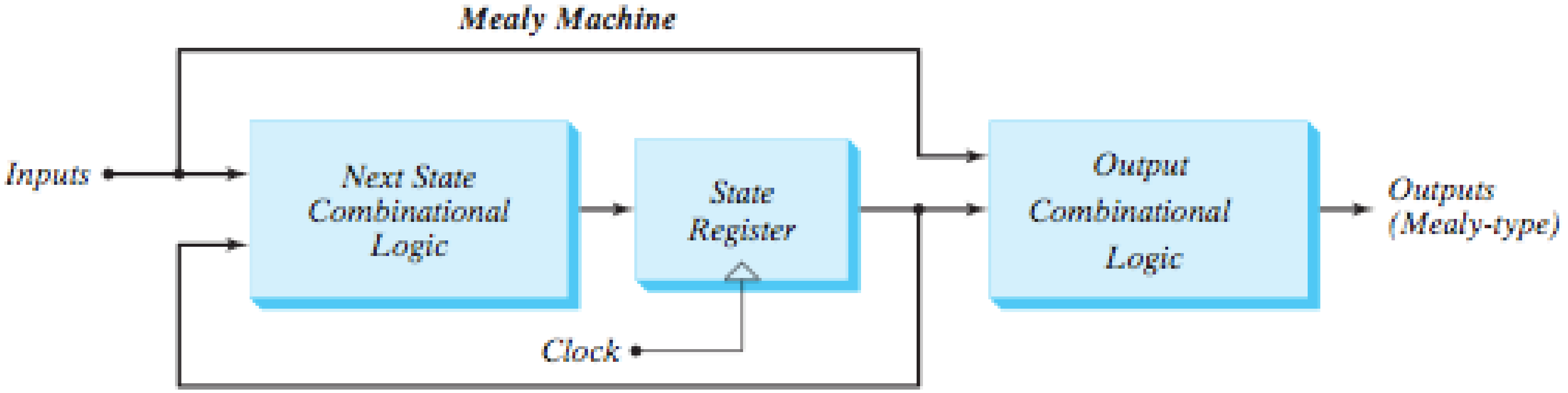
Sequential circuit analysis summary

- 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.

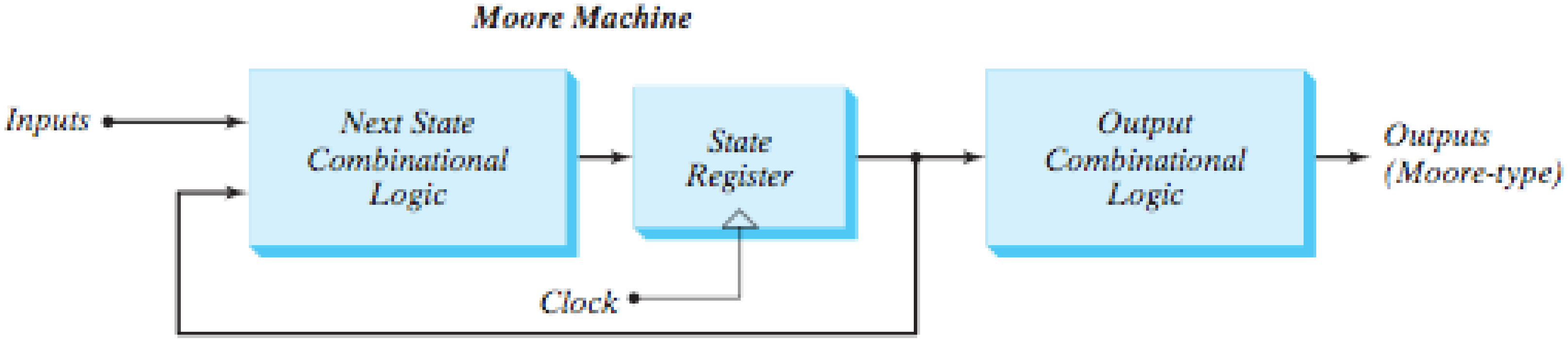
- The result of sequential circuit analysis is a state table or a state diagram describing the circuit.



FSM Models



(a)



(b)