K-maps and Sequential Circuits

250H

K-Maps

- Previously in recitation, we looked at simplifying boolean expressions using algebraic manipulation

K-Maps

- Previously in recitation, we looked at simplifying boolean expressions using algebraic manipulation
- Karnaugh Maps also simplify expressions in order to reduce the number of gates and inputs

K-Maps

- Previously in recitation, we looked at simplifying boolean expressions using algebraic manipulation
- Karnaugh Maps also simplify expressions in order to reduce the number of gates and inputs
- Maurice Karnaugh developed the Karnaugh Map at Bell Labs in 1953

K-Maps

- Previously in recitation, we looked at simplifying boolean expressions using algebraic manipulation
- Karnaugh Maps also simplify expressions in order to reduce the number of gates and inputs
- Maurice Karnaugh developed the Karnaugh Map at Bell Labs in 1953
- K-maps come from venn diagrams

K-Maps

- Previously in recitation, we looked at simplifying boolean expressions using algebraic manipulation
- Karnaugh Maps also simplify expressions in order to reduce the number of gates and inputs
- Maurice Karnaugh developed the Karnaugh Map at Bell Labs in 1953
- K-maps come from venn diagrams
- Most people will use K-maps instead of boolean algebra when simplifying

K-Maps

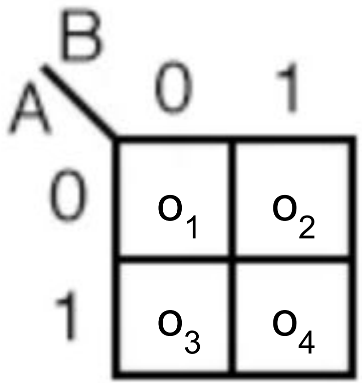
- Previously in recitation, we looked at simplifying boolean expressions using algebraic manipulation
- Karnaugh Maps also simplify expressions in order to reduce the number of gates and inputs
- Maurice Karnaugh developed the Karnaugh Map at Bell Labs in 1953
- K-maps come from venn diagrams
- Most people will use K-maps instead of boolean algebra when simplifying
- K-maps do NOT always give the smallest circuit but they often do

K-Maps

- Previously in recitation, we looked at simplifying boolean expressions using algebraic manipulation
- Karnaugh Maps also simplify expressions in order to reduce the number of gates and inputs
- Maurice Karnaugh developed the Karnaugh Map at Bell Labs in 1953
- K-maps come from venn diagrams
- Most people will use K-maps instead of boolean algebra when simplifying
- K-maps do NOT always give the smallest circuit but they often do
- The problem of getting the BEST circuit is thought to be hard

K-maps for 2 Variables

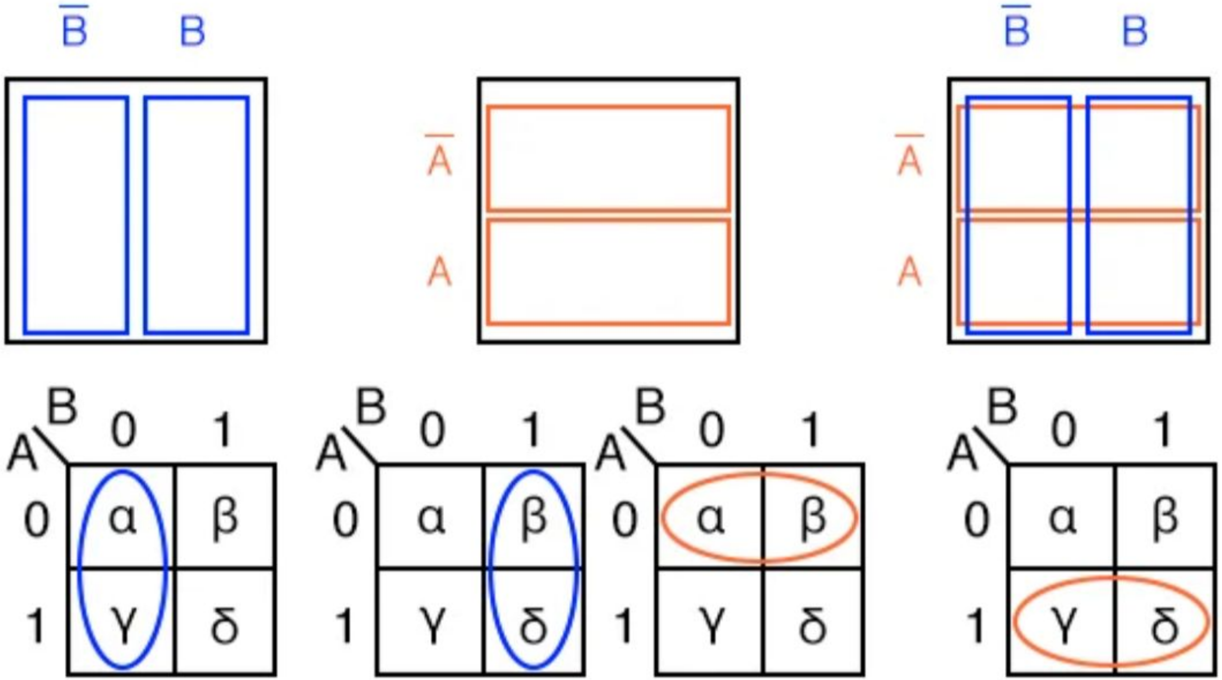
- The outputs of a truth table correspond with a Karnaugh map entries



A	B	Output
0	0	o_1
0	1	o_2
1	0	o_3
1	1	o_4

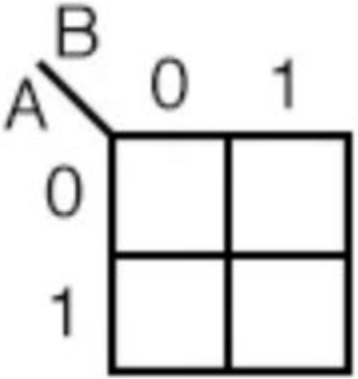
K-maps for 2 Variables

- We can simplify the expression by using the regions shown here



K-maps for 2 Variables Example

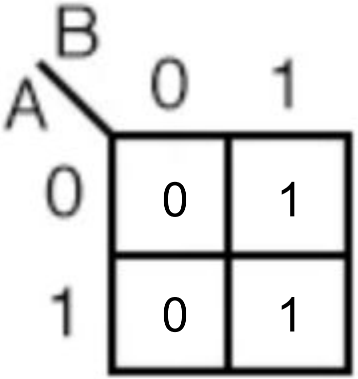
- Without simplifying we can see that the output would be:
  - $\overline{A}B + AB$



A	B	Output
0	0	0
0	1	1
1	0	0
1	1	1

K-maps for 2 Variables Example

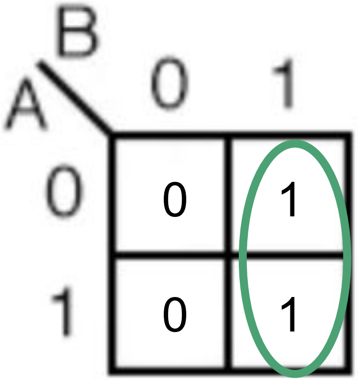
- Without simplifying we can see that the output would be:
    - $\overline{A}B + AB$
- We first translate the table to our k-map



A	B	Output
0	0	0
0	1	1
1	0	0
1	1	1

K-maps for 2 Variables Example

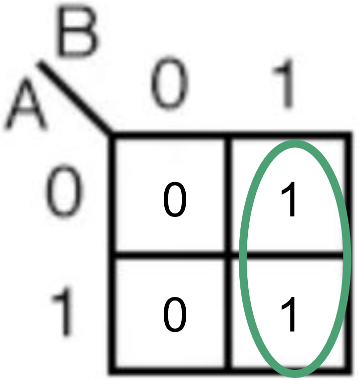
- Without simplifying we can see that the output would be:
  - $\overline{A}B + AB$
- We first translate the table to our k-map
- We now want to look at the relationships of the 1's



A	B	Output
0	0	0
0	1	1
1	0	0
1	1	1

K-maps for 2 Variables Example

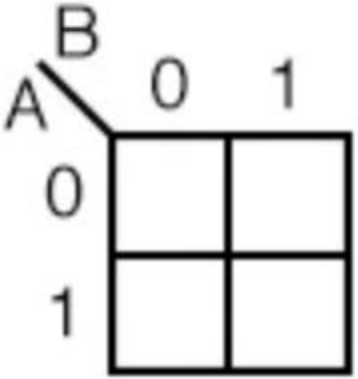
- Without simplifying we can see that the output would be:
  - $\overline{A}B + AB$
- We first translate the table to our k-map
- We now want to look at the relationships of the 1’s
- Since we see that the 1’s have a B in common
  - $\overline{A}B + AB \equiv B$



A	B	Output
0	0	0
0	1	1
1	0	0
1	1	1

K-maps for 2 Variables Example

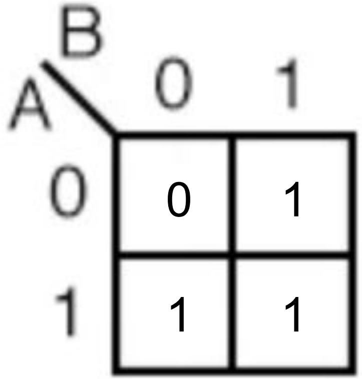
- Without simplifying we can see that the output would be:
  - $\overline{A}B + A\overline{B} + AB$



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

K-maps for 2 Variables Example

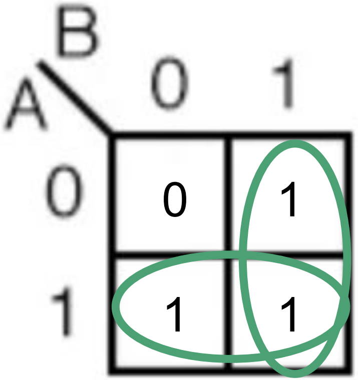
- Without simplifying we can see that the output would be:
    - $\overline{A}B + A\overline{B} + AB$
- We first translate the table to our k-map



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

K-maps for 2 Variables Example

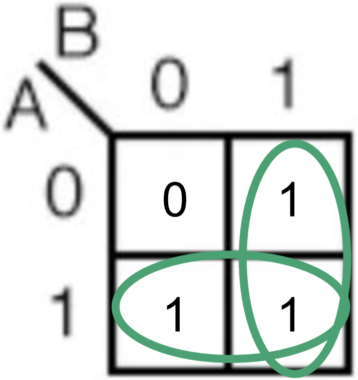
- Without simplifying we can see that the output would be:
  - $\overline{A}B + A\overline{B} + AB$
- We first translate the table to our k-map
- We now want to look at the relationships of the 1’s



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

K-maps for 2 Variables Example

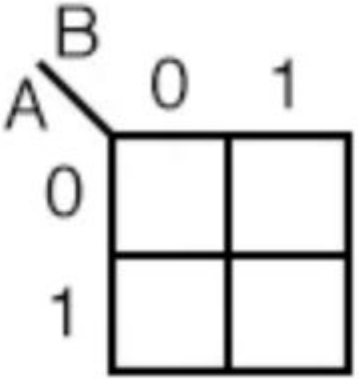
- Without simplifying we can see that the output would be:
  - $\overline{A}B + A\overline{B} + AB$
- We first translate the table to our k-map
- We now want to look at the relationships of the 1's
- Since we see that the 1's have a B and A in common
  - $\overline{A}B + A\overline{B} + AB \equiv A + B$



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

K-maps for 2 Variables Example

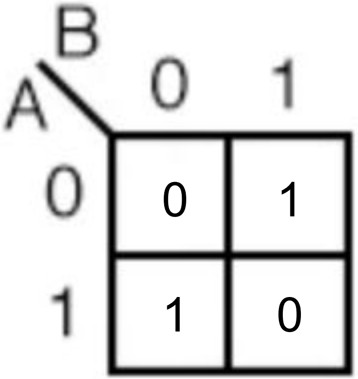
- Without simplifying we can see that the output would be:
  - $\overline{A}B + A\overline{B}$



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

K-maps for 2 Variables Example

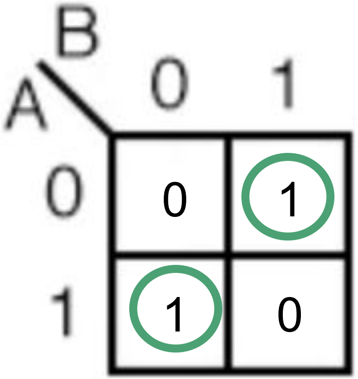
- Without simplifying we can see that the output would be:
    - $\overline{A}B + A\overline{B}$
- We first translate the table to our k-map



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

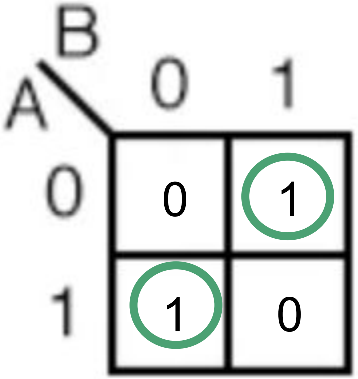
K-maps for 2 Variables Example

- Without simplifying we can see that the output would be:
    - $\overline{A}B + A\overline{B}$
- We first translate the table to our k-map
- We now want to look at the relationships of the 1’s



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

K-maps for 2 Variables Example



- Without simplifying we can see that the output would be:
  - $\overline{A}B + A\overline{B}$
- We first translate the table to our k-map
- We now want to look at the relationships of the 1's
- Since we see that we have nothing in common the simplest we can make this statement is
  - $\overline{A}B + A\overline{B}$

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

K-maps for 3 and 4 Variables

- Similarly we can extend this concept to more variables

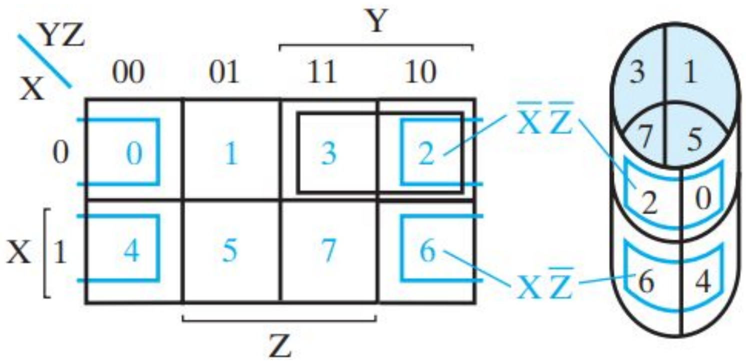
K-maps for 3 and 4 Variables

- Similarly we can extend this concept to more variables
- When we have more variables we have to “fold” the map in order to see the relationships

K-maps for 3 and 4 Variables

- Similarly we can extend this concept to more variables
- When we have more variables we have to “fold” the map in order to see the relationships

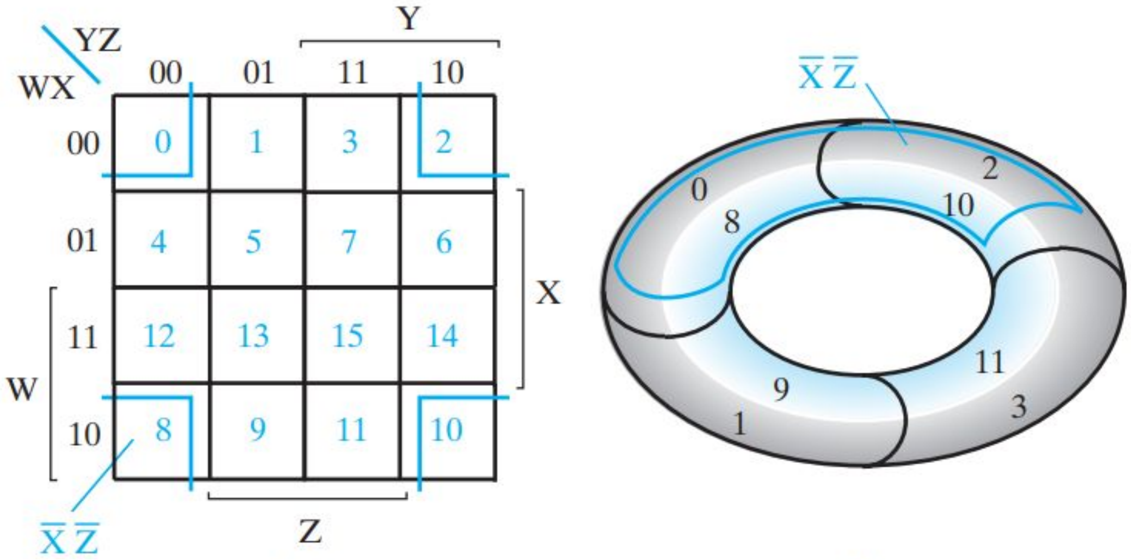
Three Variables:



K-maps for 3 and 4 Variables

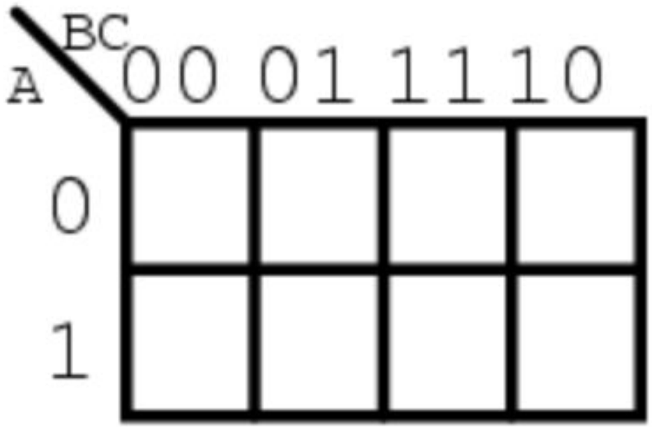
- Similarly we can extend this concept to more variables
- When we have more variables we have to “fold” the map in order to see the relationships

Four Variables:



K-maps for 3 Variables Example

A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



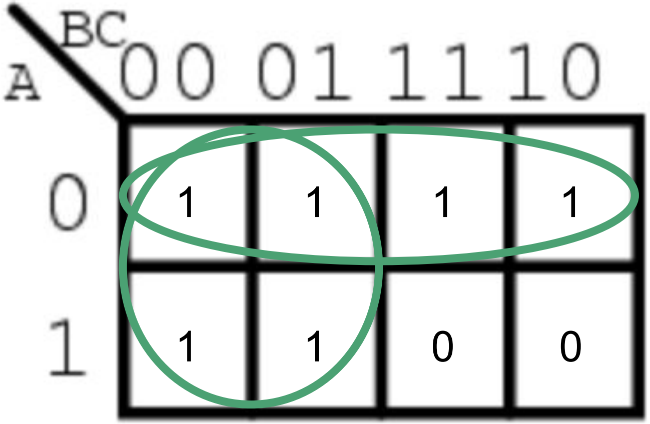
K-maps for 3 Variables Example

A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

A \ BC	00	01	11	10
0	1	1	1	1
1	1	1	0	0

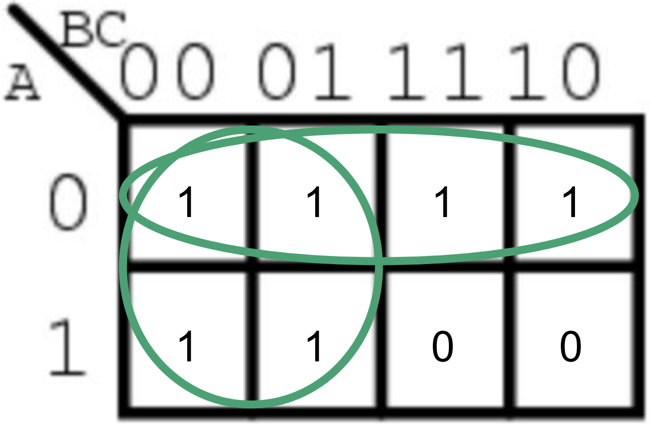
K-maps for 3 Variables Example

A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



K-maps for 3 Variables Example

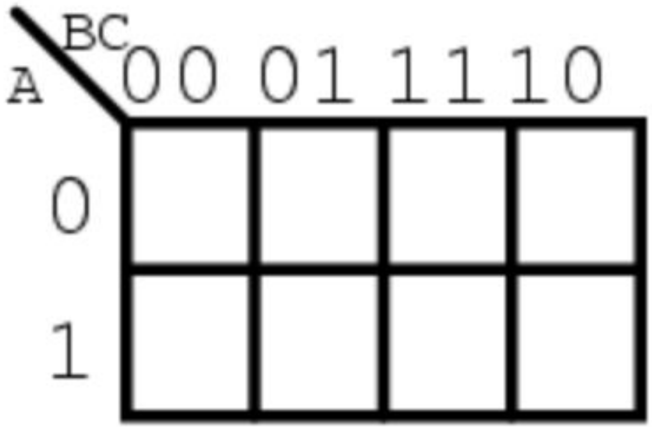
A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



Output: $\overline{A} + \overline{B}$

K-maps for 3 Variables Example

A	B	C	Output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



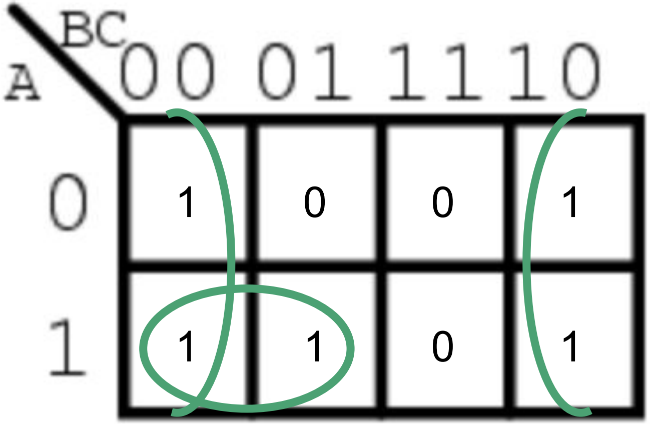
K-maps for 3 Variables Example

A	B	C	Output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

A \ BC	00	01	11	10
0	1	0	0	1
1	1	1	0	1

K-maps for 3 Variables Example

A	B	C	Output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



K-maps for 3 Variables Example

A	B	C	Output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

A \ BC	00	01	11	10
0	1	0	0	1
1	1	1	0	1

Output: $A\overline{B} + \overline{C}$

Sequential Circuits

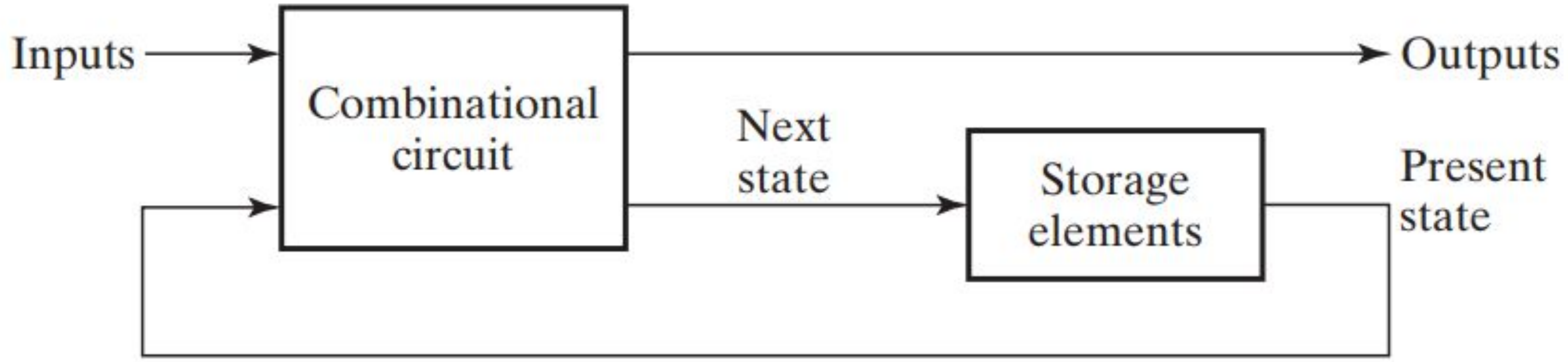
- The digital circuits we looked at previously have been combinational

Sequential Circuits

- The digital circuits we looked at previously have been combinational
- Digital systems include a combinational circuit and storage elements

Sequential Circuits

- The digital circuits we looked at previously have been combinational
- Digital systems include a combinational circuit and storage elements
- These storage elements are described as sequential circuits



SR Latch

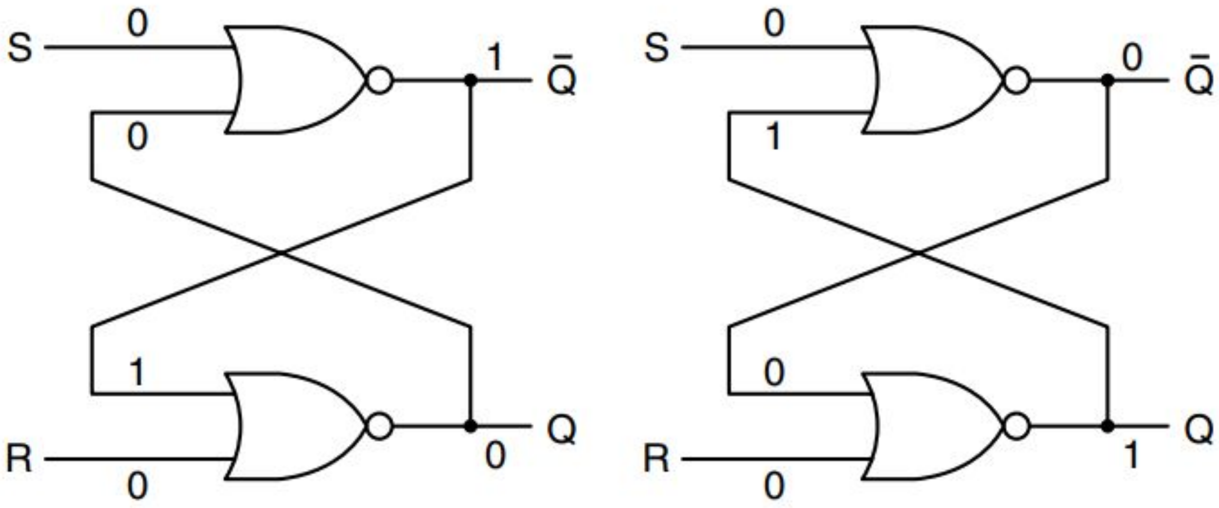
- To create a 1-bit memory, we need a circuit to remember a previous input value

SR Latch

- To create a 1-bit memory, we need a circuit to remember a previous input value
- We can construct this with two NOR gates

SR Latch

- To create a 1-bit memory, we need a circuit to remember a previous input value
- We can construct this with two NOR gates



SR Latch

- Two inputs
  - S: Setting the Latch
  - R: Resetting the latch

SR Latch

- Two inputs
  - S: Setting the Latch
  - R: Resetting the latch
- The outputs of the latch are not uniquely determined by the current inputs

SR Latch

- Two inputs
  - S: Setting the Latch
  - R: Resetting the latch
- The outputs of the latch are not uniquely determined by the current inputs
- When S is set to 1 momentarily, the latch ends up in state Q = 1, regardless of what state it was previously in

SR Latch

- Two inputs
  - S: Setting the Latch
  - R: Resetting the latch
- The outputs of the latch are not uniquely determined by the current inputs
- When S is set to 1 momentarily, the latch ends up in state Q = 1, regardless of what state it was previously in
- Setting R to 1 momentarily forces the latch to state Q = 0

SR Latch

- Two inputs
    - S: Setting the Latch
    - R: Resetting the latch
- The outputs of the latch are not uniquely determined by the current inputs
- When S is set to 1 momentarily, the latch ends up in state Q = 1, regardless of what state it was previously in
- Setting R to 1 momentarily forces the latch to state Q = 0
- The circuit “remembers” whether S or R was last on

SR Latch

- Two inputs
  - S: Setting the Latch
  - R: Resetting the latch
- The outputs of the latch are not uniquely determined by the current inputs
- When S is set to 1 momentarily, the latch ends up in state Q = 1, regardless of what state it was previously in
- Setting R to 1 momentarily forces the latch to state Q = 0
- The circuit “remembers” whether S or R was last on
- Using this property, we can build computer memories

Clocked SR Latches

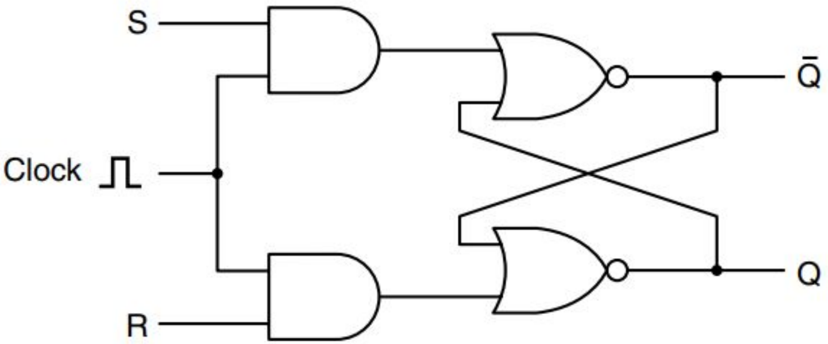
- We might need to prevent the latch from changing state except at certain specified times

Clocked SR Latches

- We might need to prevent the latch from changing state except at certain specified times
- We can modify the SR Latch slightly to get a clocked SR latch

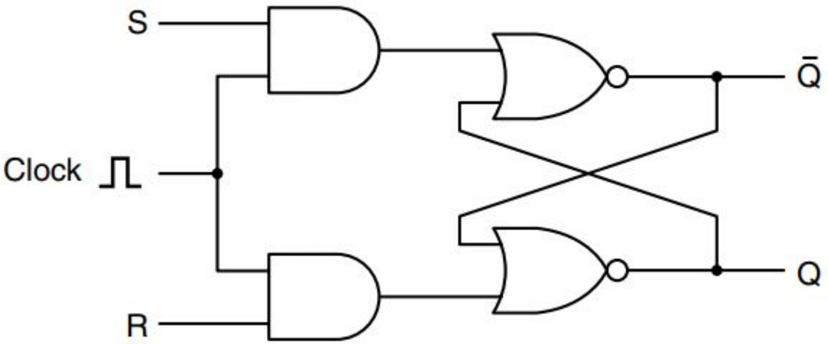
Clocked SR Latches

- We might need to prevent the latch from changing state except at certain specified times
- We can modify the SR Latch slightly to get a clocked SR latch



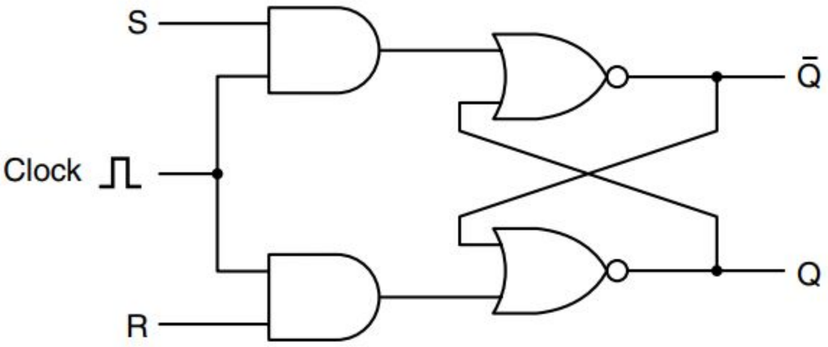
Clocked SR Latches

- We might need to prevent the latch from changing state except at certain specified times
- We can modify the SR Latch slightly to get a clocked SR latch
- The circuit has an additional input for the clock and is normally set to 0



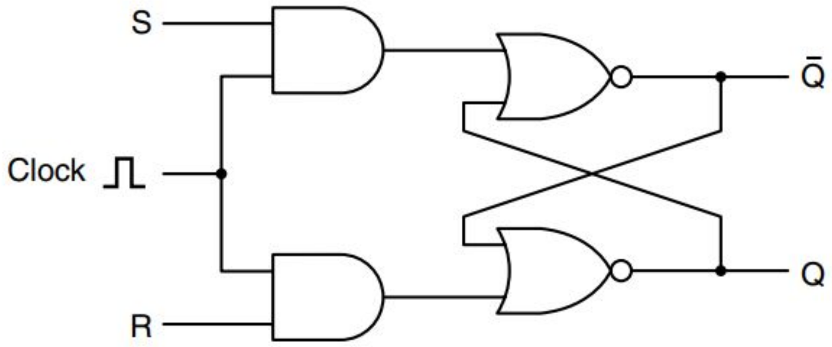
Clocked SR Latches

- We might need to prevent the latch from changing state except at certain specified times
- We can modify the SR Latch slightly to get a clocked SR latch
- The circuit has an additional input for the clock and is normally set to 0
- With the clock 0, both AND gates output 0, independent of S and R, and the latch does not change state



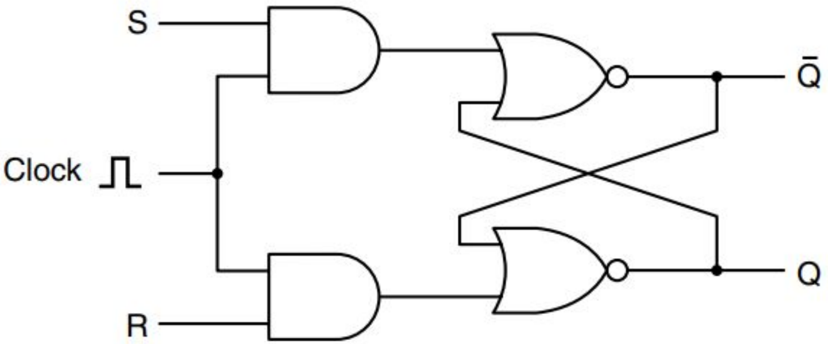
Clocked SR Latches

- With the clock 0, both AND gates output 0, independent of S and R, and the latch does not change state
- When the clock is 1, the effect of the AND gates vanishes and the latch relies on S and R



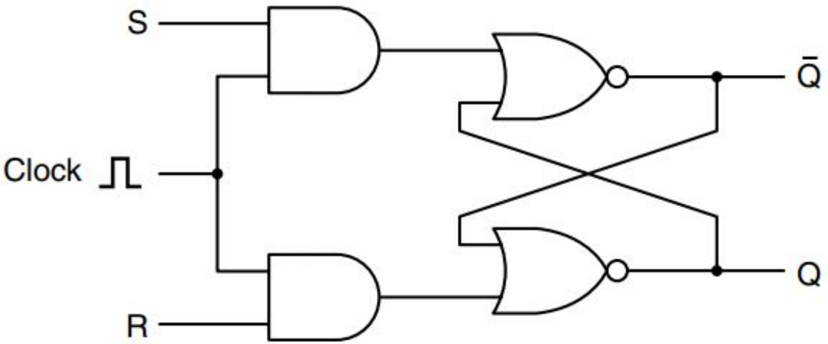
Clocked D Latch

- What happens when S and R both equal 1?



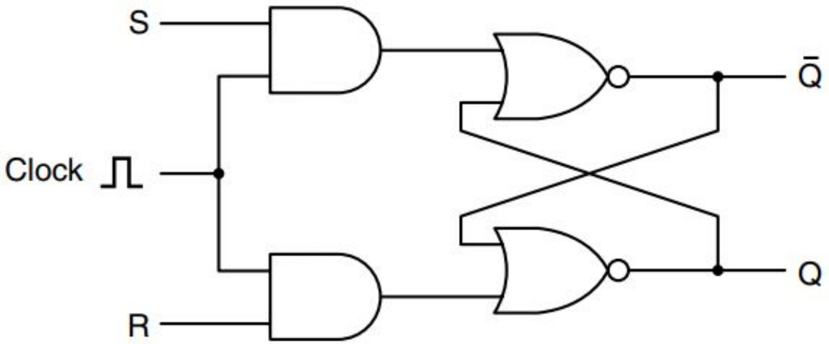
Clocked D Latch

- What happens when S and R both equal 1?
- The circuit becomes nondeterministic when both R and S finally return to 0



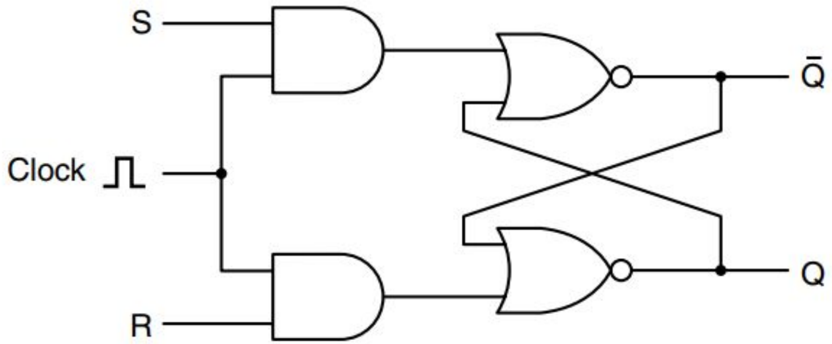
Clocked D Latch

- What happens when S and R both equal 1?
- The circuit becomes nondeterministic when both R and S finally return to 0
- The only consistent state for S = R = 1 is $Q = \overline{Q} = 0$, but as soon as both inputs return to 0, the latch must jump to one of its two stable states



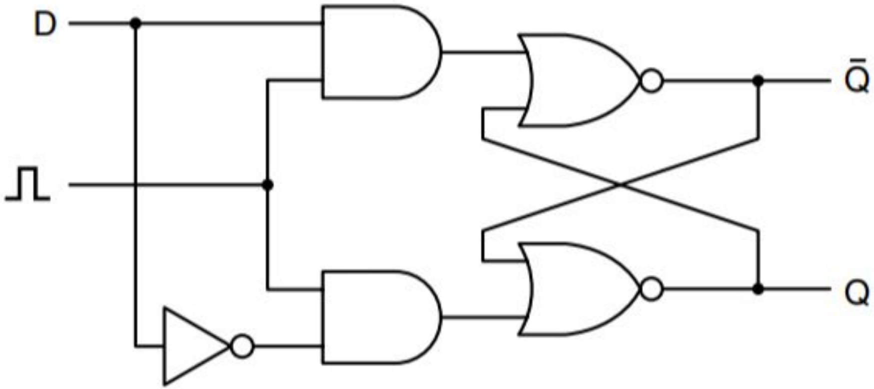
Clocked D Latch

- What happens when S and R both equal 1?
- The circuit becomes nondeterministic when both R and S finally return to 0
- The only consistent state for S = R = 1 is $Q = \overline{Q} = 0$, but as soon as both inputs return to 0, the latch must jump to one of its two stable states
- The latch will jump to one of its stable states at random



Clocked D Latch

- We resolve this issue by preventing it from ever happening
- We create a circuit that only has one input: D
- Because the input to the lower AND gate is always the complement of the input to the upper one, the problem of both inputs being 1 never arises.



Flip-Flops

- Sometimes we need to sample the value on a certain line at a particular instant and store it

Flip-Flops

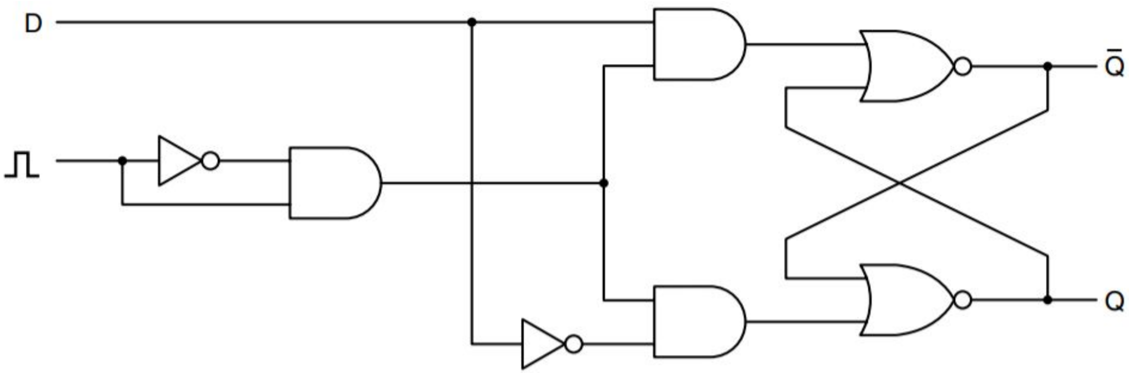
- Sometimes we need to sample the value on a certain line at a particular instant and store it
- The state transition during the clock transition from 0 to 1 or from 1 to 0 instead of when the clock is 1

Flip-Flops

- Sometimes we need to sample the value on a certain line at a particular instant and store it
- The state transition during the clock transition from 0 to 1 or from 1 to 0 instead of when the clock is 1
- This is called a Flip-Flop

Flip-Flops

- Sometimes we need to sample the value on a certain line at a particular instant and store it
- The state transition during the clock transition from 0 to 1 or from 1 to 0 instead of when the clock is 1
- This is called a Flip-Flop



Flip-Flops

- Flip-flops can be combined in groups to create registers, which hold data types larger than 1 bit in length

Flip-Flops

- Flip-flops can be combined in groups to create registers, which hold data types larger than 1 bit in length
- Eight flip-flops can be put together to form an 8-bit storage register

Flip-Flops

- Flip-flops can be combined in groups to create registers, which hold data types larger than 1 bit in length
- Eight flip-flops can be put together to form an 8-bit storage register
- The register accepts an 8-bit input value when the clock transitions

Flip-Flops

- Flip-flops can be combined in groups to create registers, which hold data types larger than 1 bit in length
- Eight flip-flops can be put together to form an 8-bit storage register
- The register accepts an 8-bit input value when the clock transitions
- To implement a register, all the clock lines are connected to the same input signal

Flip-Flops

- Flip-flops can be combined in groups to create registers, which hold data types larger than 1 bit in length
- Eight flip-flops can be put together to form an 8-bit storage register
- The register accepts an 8-bit input value when the clock transitions
- To implement a register, all the clock lines are connected to the same input signal
- Each register will accept the new 8-bit data value on the input bus