Boolean Logic

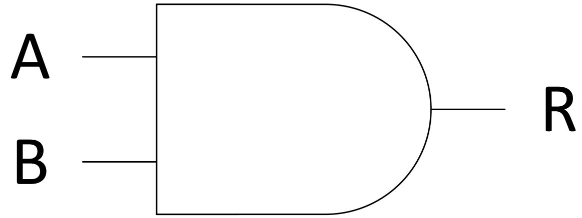
Prof. James L. Frankel
Harvard University

Version of 6:26 PM 10-Sep-2024
Copyright © 2024, 2023, 2021, 2020, 2019, 2017, 2016 James L. Frankel. All rights reserved.

Logic Levels

- Logic 0
    - Also called GND
    - Low
    - Off
    - False
- Logic 1
    - Also called V_{CC}
    - High
    - On
    - True

AND Gate



R = AB

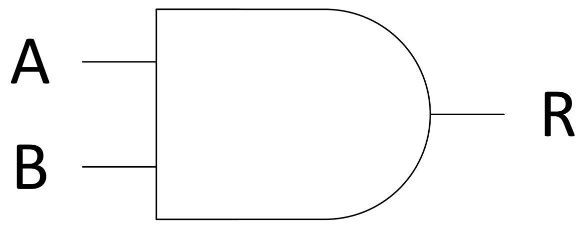
A	B	R
0	0	0
0	1	0
1	0	0
1	1	1

Inputs vs. Outputs

- A and B are inputs
  - Inputs are identified by letters at the beginning of the alphabet
  - H should not be used (confusable with H for High)
  - Try not to use I (looks like 1)
- R is the output
  - Outputs are identified by letters near the end of the alphabet
  - For a single output, Q is often used
  - X and Z should not be used (are used for other purposes)
  - L should not be used (confusable with L for Low)
  - Try not to use O (looks like 0)
- Both inputs and outputs may be named by longer identifiers (e.g. Clk, En)

- Inputs are separated from outputs by a solid vertical line
  - If multiple inputs are separated from each other by a solid vertical line and multiple outputs are separated from each other by a solid vertical line, then the inputs should be separated from the outputs by a ***double*** solid vertical line

The Schematic Symbol for an AND Gate



The Truth Table for an AND Gate

- Describes the behavior of outputs based on the state of inputs

A	B	R
0	0	0
0	1	0
1	0	0
1	1	1

Order in which Inputs are Listed

- In general, in a truth table the inputs are listed in numerical order as if:
  - The rightmost input is the ones place or column ($2^0 = 1$)
  - The next left-more input is the twos place ($2^1 = 2$)
  - The next left-more input is the fours place ($2^2 = 4$)
  - etc.
  - The leftmost input is the most significant bit

Number of Inputs

- The number of inputs to a gate can vary
- The number of inputs to a gate is referred to as **fan-in**
- Generally primitive gates (like our AND gate) will not have more than eight inputs
- Generally the number of inputs available in a non-custom primitive gate are:
    - 2
    - 3
    - 4
    - 8

Number of Outputs

- In general, primitive gates (like our AND gate) will have a single output

Boolean Formula for AND

- We can also write a formula for the AND gate we just designed

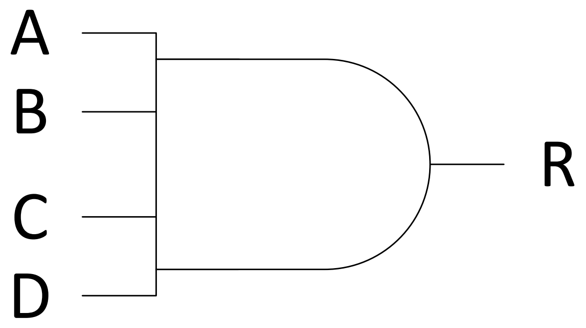
- The AND operator can be written as a middle dot (to signify multiplication): $\cdot$
  - $R = A \cdot B$
- Or, simply with no symbol – also as multiplication
  - $R = AB$
- Or, as a cap: $\wedge$
  - $R = A \wedge B$
- Or, as the uppercase word AND
  - R = A AND B
- Or, as an ampersand: &
  - R = A & B

AND Gate Observations

- If one input of the AND gate, say A, is under our control, but the other input, B, is not…
  - If we assert A Low, then the output is Low
  - If we assert A High, then the output is the same as the input, B

  - Thus, we can use the AND gate to enable or disable the propagation of a signal
  - This behavior is often referred to as **masking** or applying a **mask**

AND Gate with Four Inputs

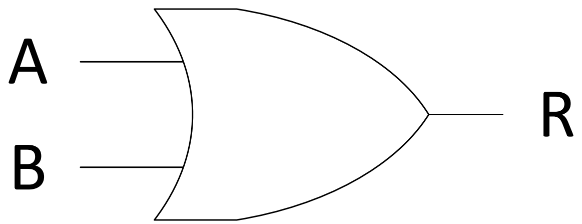


- All inputs must be 1 for the output, R, to be 1
- Otherwise, the output is 0

- This is equivalent to saying that if any input is a 0, the output, R, will be 0
- Otherwise, the output is 1
  - This is demonstrating the principle of **duality**

- R = ABCD

OR Gate



$R = A + B$

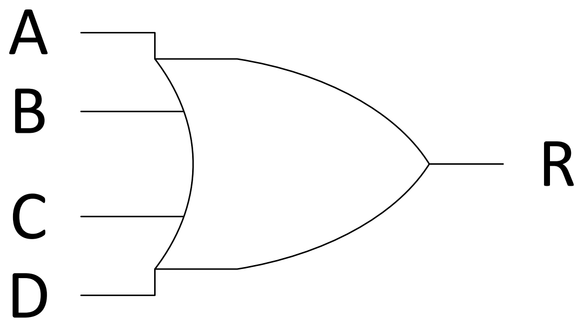
A	B	R
0	0	0
0	1	1
1	0	1
1	1	1

Boolean Formula for OR

- We can also write a formula for the OR gate we just designed

- The OR operator can be written as a plus symbol (to signify addition): +
  - R = A + B
- Or, as a cup: ∨
  - R = A ∨ B
- Or, as the uppercase word OR
  - R = A OR B
- Or, as a vertical bar: |
  - R = A | B

OR Gate with Four Inputs

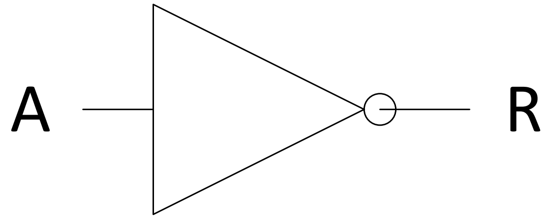


- If any input is a 1 for the output, R, will be 1
- Otherwise, the output is 0

- This is equivalent to saying that all inputs must be 0 for the output, R, to be 0
- Otherwise, the output is 1
    - This is demonstrating the principle of **duality**

- R = A + B + C + D

NOT Gate or Inverter



$R = \bar{A}$

A	R
0	1
1	0

Inverter Observations

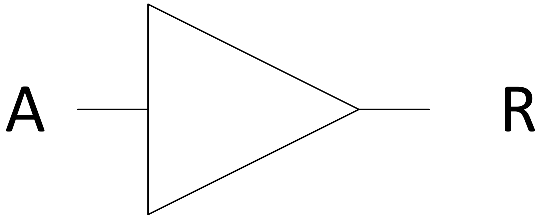
- The triangular shape is used to signify a digital (or Boolean) amplifier
  - A digital amplifier is a device that re-establishes the original noise margins and can drive a large number of inputs
- The bubble (or circle) on the output of the Inverter signifies Boolean inversion

Boolean Formula for NOT

- We can also write a formula for the Inverter gate we just designed

- The NOT operator can be written as a macron or overbar
  - R = Ā
- Or, as a prefix tilde
  - R = ~A
- Or, as a prefix hook or not-sign
  - R = ¬A
- Or, as a suffix prime symbol (or apostrophe or neutral (*i.e.,* straight) single quote)
  - R = A'
- Or, as the uppercase word NOT
  - R = NOT A

Buffer or Driver



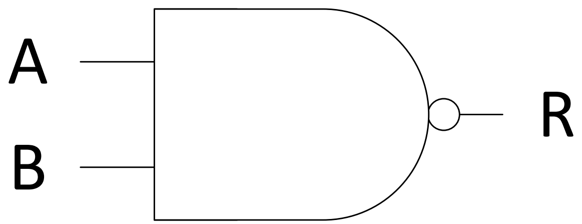
R = A

A	R
0	0
1	1

Buffer Observations

- The triangular shape is used to signify a digital (or Boolean) amplifier
  - A digital amplifier is a device that re-establishes the original noise margins and can drive a large number of inputs

NAND Gate



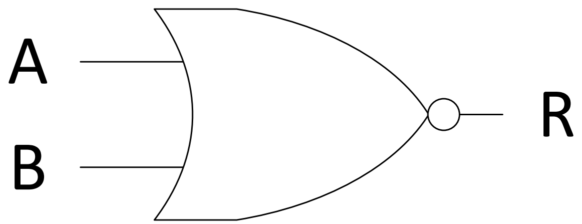
R = $\overline{AB}$

A	B	R
0	0	1
0	1	1
1	0	1
1	1	0

NAND Gate Observations

- The bubble (or circle) on the output of the AND gate signifies Boolean inversion
- Thus, the NAND gate is equivalent to an AND gate followed by a NOT gate

NOR Gate



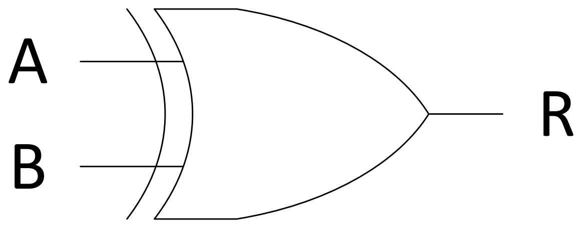
R = $\overline{A + B}$

A	B	R
0	0	1
0	1	0
1	0	0
1	1	0

NOR Gate Observations

- The bubble (or circle) on the output of the OR gate signifies Boolean inversion
- Thus, the NOR gate is equivalent to an OR gate followed by a NOT gate

Exclusive-OR or XOR Gate



$R = A \oplus B$

A	B	R
0	0	0
0	1	1
1	0	1
1	1	0

Boolean Formula for XOR

- We can also write a formula for the Exclusive-OR gate we just designed

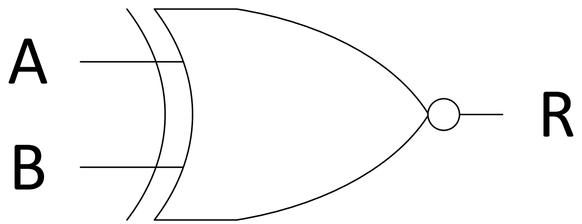
- The XOR operator can be written as a circled plus symbol: $\oplus$
  - R = A $\oplus$ B
- Or, as an underlined cup: $\underline{\vee}$
  - R = A $\underline{\vee}$ B
- Or, as the uppercase word XOR
  - R = A XOR B

XOR Gate Observations

- If one input of the XOR gate, say A, is under our control, but the other input, B, is not…
  - If we assert A Low, then the output is the same as the input, B
  - If we assert A High, then the output is the inverse of the input, B

  - Thus, we can use the XOR gate to selectively invert a signal

Equivalence (EQV) or XNOR Gate



R = $\overline{A \oplus B}$

A	B	R
0	0	1
0	1	0
1	0	0
1	1	1

XNOR Gate Observations

- The XNOR gate will output a High signal if its two inputs are the same; otherwise the output will be Low
- Hence, it is also referred to as an Equivalence gate

XNOR Gate Observations

- The bubble (or circle) on the output of the XOR gate signifies Boolean inversion

- Thus, the XNOR gate is equivalent to an XOR gate followed by a NOT gate

Boolean Formula Reflections

- Why is the plus (addition) symbol used for OR?
- Why is the middle dot (multiplication) symbol used for AND?

Derivation of All Gates from a Single Primitive Gate Type

- If we wanted to be able to derive all gates from a single primitive gate, what are the properties that that single gate would possess?

Derivation of All Gates from a Single Primitive Gate Type

- If we wanted to be able to derive all gates from a single primitive gate, what are the properties that that single gate would possess?
  - More than one input

Derivation of All Gates from a Single Primitive Gate Type

- If we wanted to be able to derive all gates from a single primitive gate, what are the properties that that single gate would possess?
  - More than one input
  - Inversion

Derivation of All Gates from a Single Primitive Gate Type

- If we wanted to be able to derive all gates from a single primitive gate, what are the properties that that single gate would possess?
  - More than one input
  - Inversion
  - Ability to set an input to High or Low

Gates that Include Inversion & Multiple Inputs

- NAND
- NOR
- XOR
- XNOR

Derivation of AND from NAND

- Can you derive the functionality of an AND gate from a circuit of just NAND gates?
  - Multiple NAND gates may be utilized

Derivation of NOT from NAND

- Can you derive the functionality of a NOT gate from a circuit of just NAND gates?
  - Multiple NAND gates may be utilized

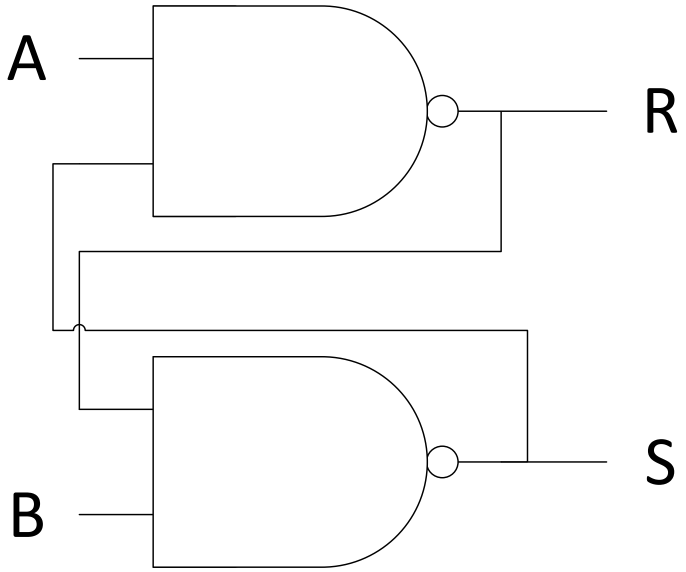
Derivation of AND from NOR

- Can you derive the functionality of the AND gate from a circuit of just NOR gates?
  - Multiple NOR gates may be utilized

Derivation of XOR and XNOR

- Can you derive the functionality of the XOR and XNOR gates from a circuit of AND, OR, NAND, NOR, and NOT gates?
    - More than one instance of each gate may be utilized

Flip Flop

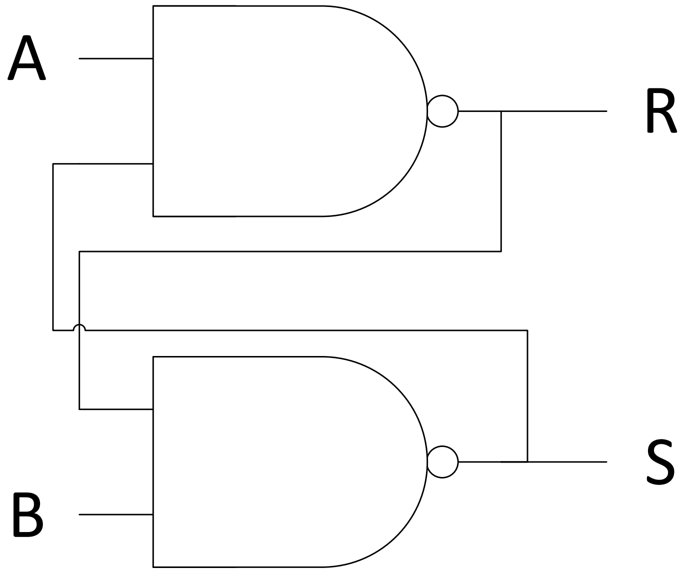


A	B	R	S
0	1	1	0
1	0	0	1
1	1	$\sim S_0$	$\sim R_0$
0	0	1	1

Flip Flop Observations

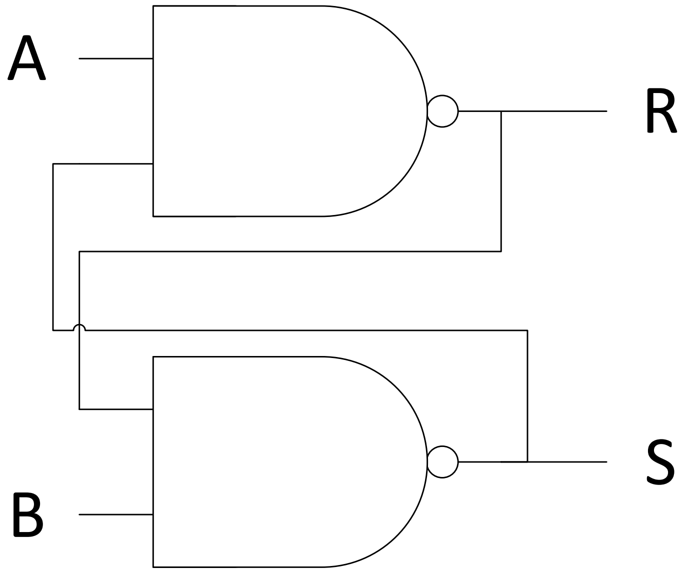
- This structure is called “cross-coupled NAND gates”
- The tilde (~) is used to indicate inversion
- The subscript of 0 signifies the previous state of the signal
- This circuit is unusual in that *it uses its **outputs as inputs***
- Note that the input A and B columns are not in binary numerical order in the truth table – we want to analyze the rows (almost) in this order

Flip Flop: A==0, B==1



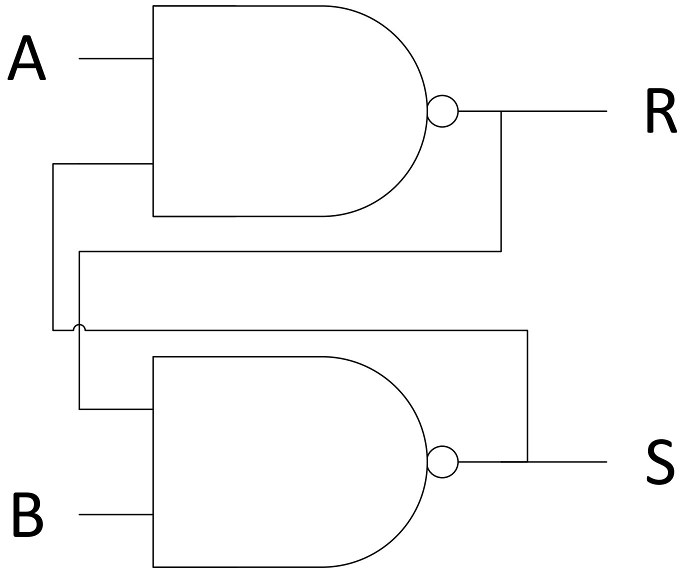
A	B	R	S
0	1	1	0
1	0	0	1
1	1	$\sim S_0$	$\sim R_0$
0	0	1	1

Flip Flop: A==1, B==0



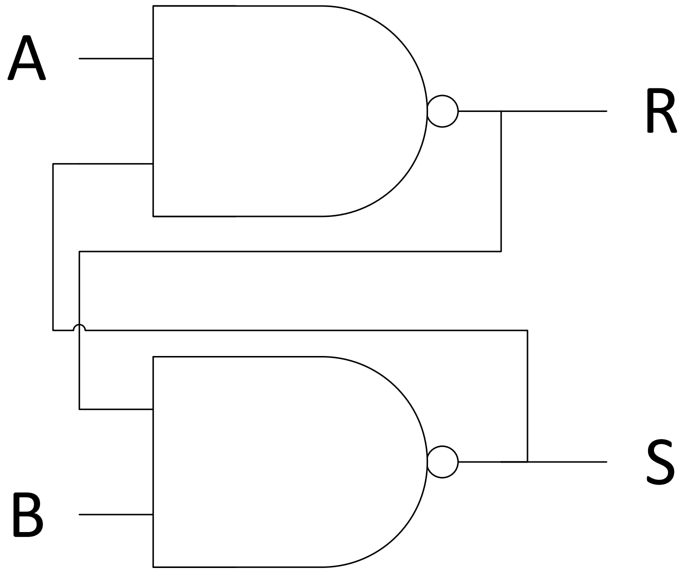
A	B	R	S
0	1	1	0
1	0	0	1
1	1	$\sim S_0$	$\sim R_0$
0	0	1	1

Flip Flop: A==0, B==0



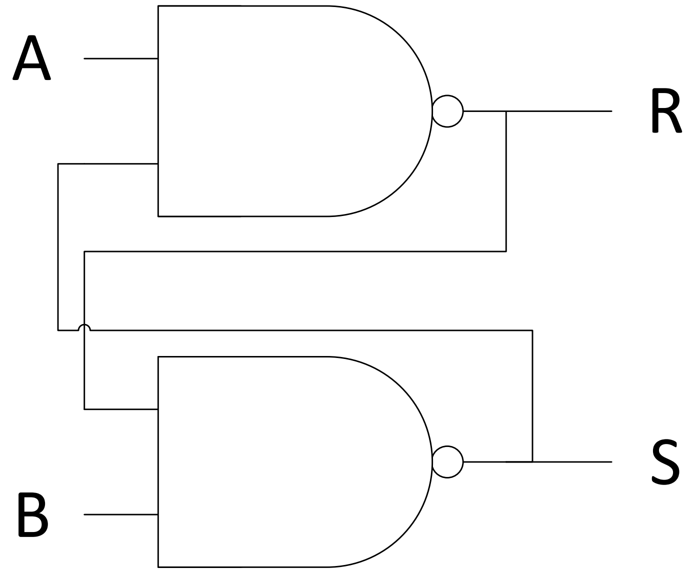
A	B	R	S
0	1	1	0
1	0	0	1
1	1	$\sim S_0$	$\sim R_0$
0	0	1	1

Flip Flop: A==1, B==1 is more complicated



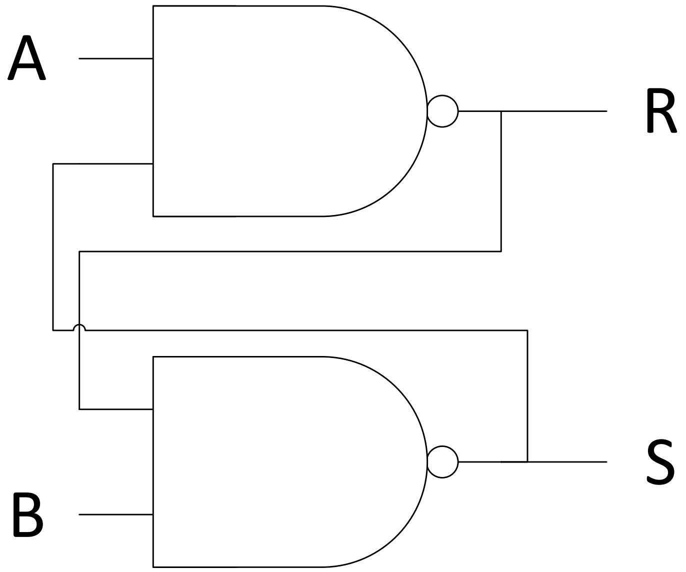
A	B	R	S
0	1	1	0
1	0	0	1
1	1	$\sim S_0$	$\sim R_0$
0	0	1	1

Flip Flop: A==1, B==1, with previous R==1, S==0



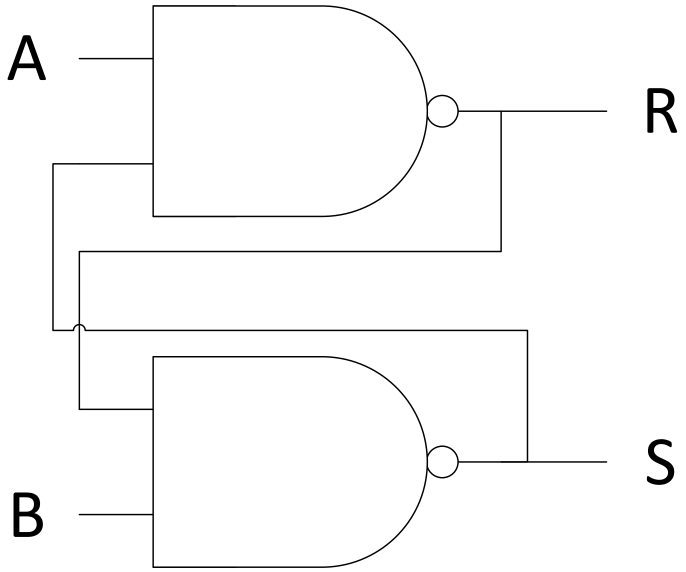
A	B	R	S
0	1	1	0
1	0	0	1
1	1	1	0
0	0	1	1

Flip Flop: A==1, B==1, with previous R==0, S==1



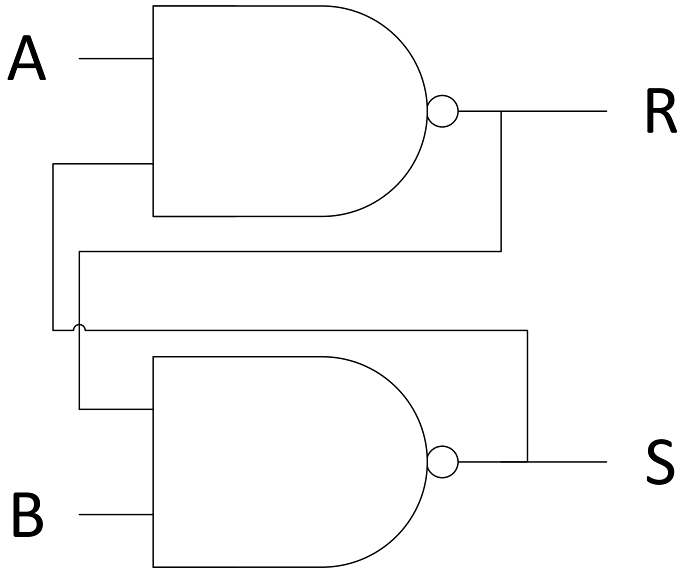
A	B	R	S
0	1	1	0
1	0	0	1
1	1	0	1
0	0	1	1

Flip Flop: A==1, B==1, with previous R==1, S==1



A	B	R	S
0	1	1	0
1	0	0	1
1	1	0	0
0	0	1	1

Flip Flop: A==1, B==1, with previous R==0, S==0

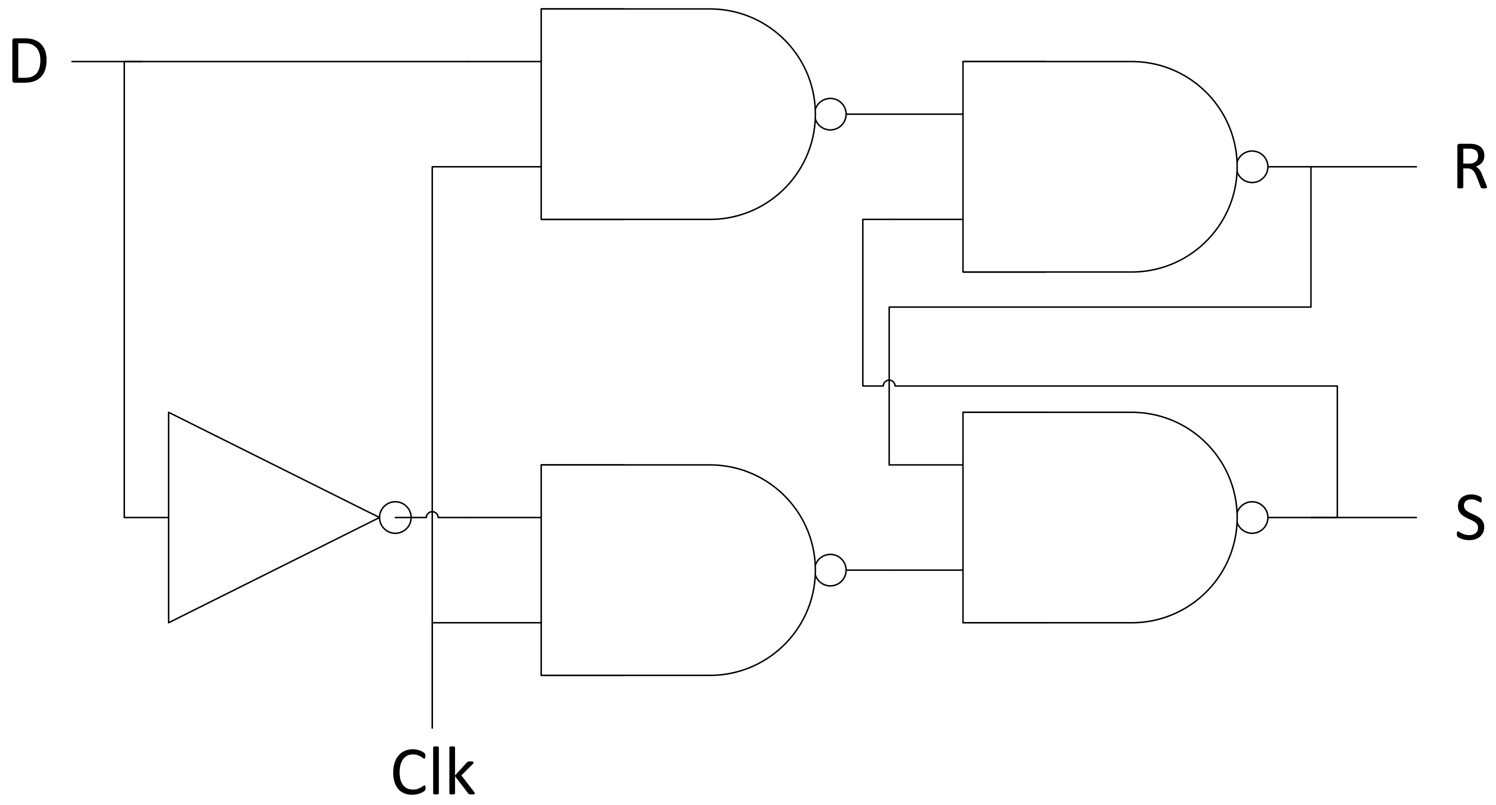


A	B	R	S
0	1	1	0
1	0	0	1
1	1	1	1
0	0	1	1

Flip Flop Memory Behavior

- Never set A to 0 and B to 0 at the same time
- If A != B, then outputs are stable and R = !A and S = !B
- If we are in a state where A != B and R != S and then we move to a state in which A = B = 1, then
  - Whatever the state of R and S were before we set A and B to both be 1, the flip flop will remember that state!
  - The flip flop will remember that previous state of the inputs in the outputs!

D Latch

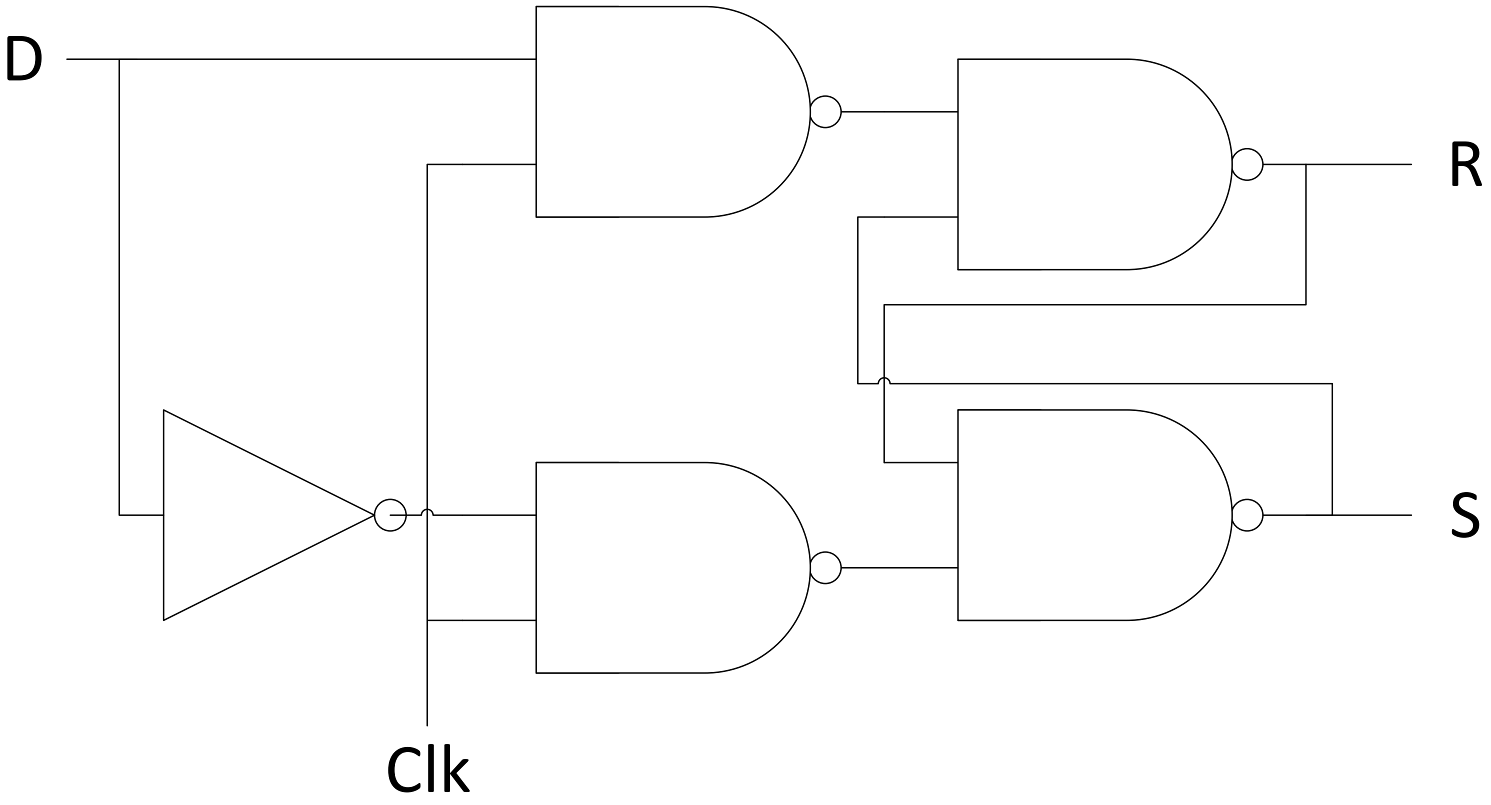


D	Clk	R	S
X	0	$\sim S_0 = R_0$	$\sim R_0 = S_0$
0	1	0	1
1	1	1	0

D Latch Observations

- The X input is a don’t-care symbol
- It means that the state of that input is irrelevant
  - It is a way to simplify the truth table
  - Otherwise, all states of that input would have to be listed

D Latch



D	Clk	R	S
X	0	$\sim S_0 = R_0$	$\sim R_0 = S_0$
D	1	D	~D

D Latch Observations

- This version of the truth table uses a symbol “D” as a variable
- This usage both simplifies the table and emphasizes that the state of that input is “stored”