Sequential logic and the R-S latch
COMSM1302 Overview of Computer Architecture

John Lapinskas, University of Bristol

What is sequential logic?

In all the circuits you've seen so far, the output only depends on the combination of the inputs. Independently of everything else, $1 \wedge 1 = 1$.

This is called **combinational** or **combinatorial** logic. Everything we've done so far works in this paradigm, and it's still useful — the ALU from week 2 is the same one we'll use in creating the CPU.

What is sequential logic?

In all the circuits you've seen so far, the output only depends on the combination of the inputs. Independently of everything else, $1 \wedge 1 = 1$.

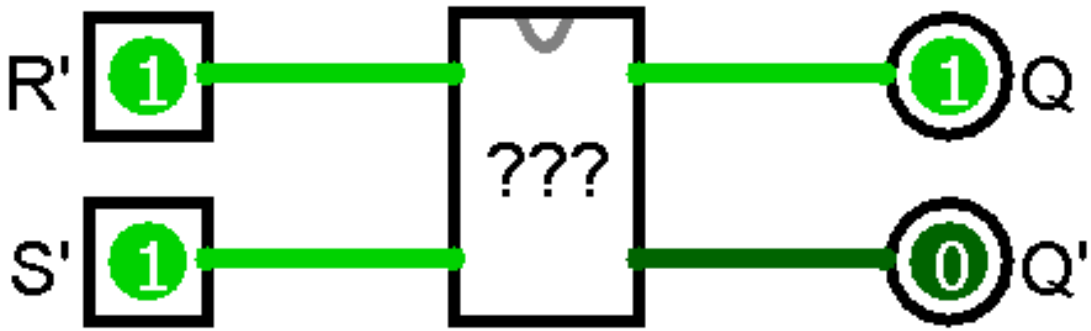
This is called **combinational** or **combinatorial** logic. Everything we've done so far works in this paradigm, and it's still useful — the ALU from week 2 is the same one we'll use in creating the CPU.

But computers don't work that way! They maintain internal state. To build a computer, we have to go further.

In **sequential** logic outputs depend on *past* inputs, not just present inputs.

Demonstration

[Demonstration of R-S latch behaviour in Logisim — see video.]



R-S latches

This circuit is an **R-S latch**, and it has no (formal) truth table:

R'	S'	Q	Q'
0	0	1	1
0	1	0	1
1	0	1	0
1	1	"Hold"	"Hold"

The first three lines are fine, but when $R' = S' = 1$, the output of the circuit *stays the same* as it previously was.

We won't actually care about the behaviour when $R' = S' = 0$. This isn't part of the intended use of the circuit, and if it happens something has gone wrong. The standard way of writing this in a truth table is with an **X**.

R-S latches

This circuit is an **R-S latch**, and it has no (formal) truth table:

R'	S'	Q	Q'
0	0	X	X
0	1	0	1
1	0	1	0
1	1	"Hold"	"Hold"

The first three lines are fine, but when $R' = S' = 1$, the output of the circuit *stays the same* as it previously was.

We won't actually care about the behaviour when $R' = S' = 0$. This isn't part of the intended use of the circuit, and if it happens something has gone wrong. The standard way of writing this in a truth table is with an **X**.

R-S latches

This circuit is an **R-S latch**, and it has no (formal) truth table:

R'	S'	Q	Q'
0	0	X	X
0	1	0	1
1	0	1	0
1	1	"Hold"	"Hold"

Q' will always be $\neg Q$ (barring X inputs). We think of Q as the main output.

We think of R' as a **reset** input and S' as a **set** input.
Both are activated by going from 1 to 0, not from 0 to 1!

If R' is activated, Q is set to 0, and stays 0 until S' is next activated.
If S' is activated, Q is set to 1, and stays 1 until R' is next activated.

This behaviour is incredibly important — we'll use it to build RAM.

R-S latches

This circuit is an **R-S latch**, and it has no (formal) truth table:

R'	S'	Q	Q'
0	0	X	X
0	1	0	1
1	0	1	0
1	1	"Hold"	"Hold"

What do the primes mean? Formally, nothing! They're *conventions*, often used in datasheets to help users quickly understand unfamiliar circuits.

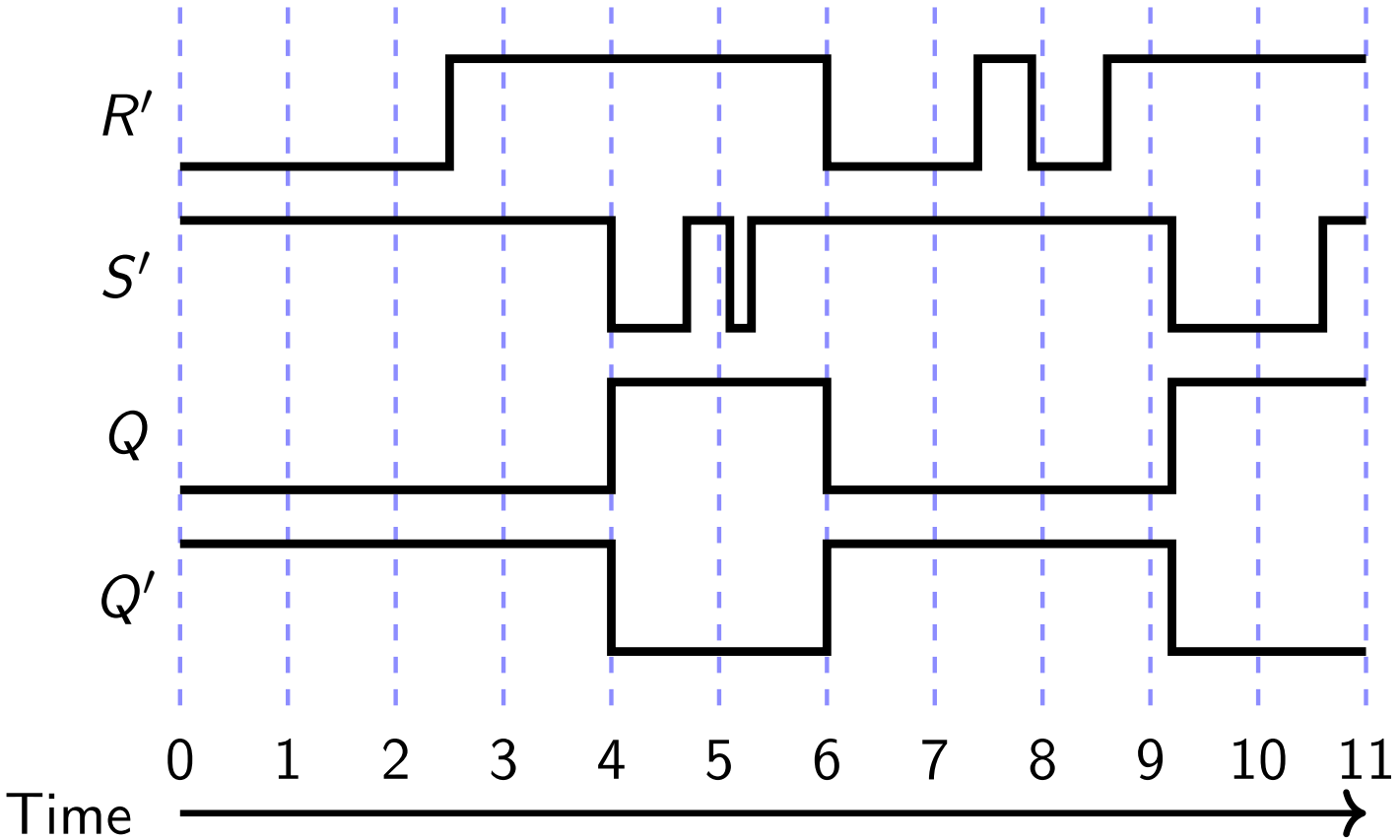
A $'$ or $\bar{}$ on an output (e.g. Q' or $\bar{Q}$) means that it is inverted — that $Q' = \neg Q$, where Q is the "real" output.

A $'$ or $\bar{}$ on an input (e.g. R' or $\bar{R}$) means that it is "activated" by going from 1 to 0. We call inputs like these **active low**.

Inputs which are "activated" by going from 0 to 1 are called **active high**.

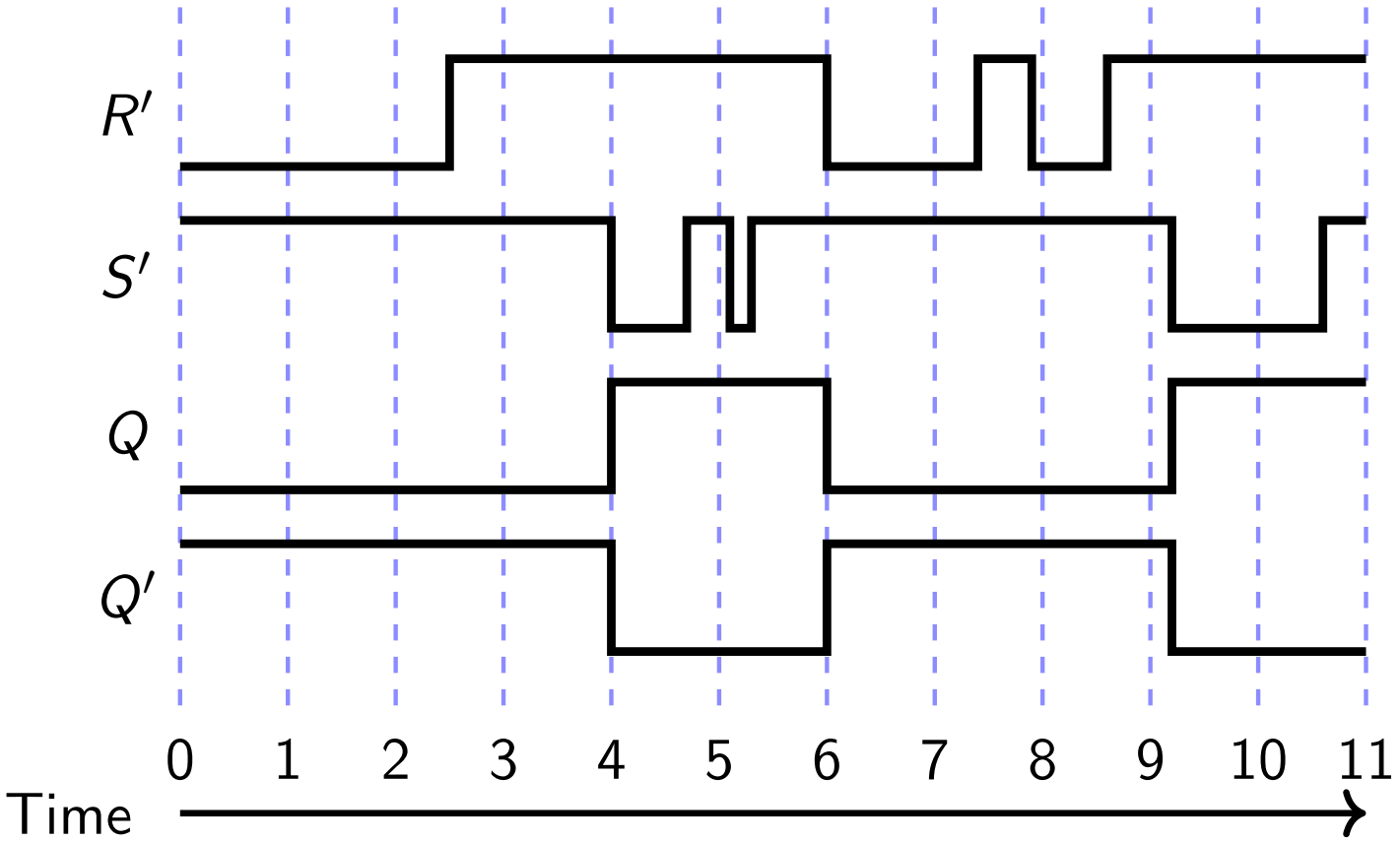
Timing diagrams

In sequential logic, we use not just truth tables but **timing diagrams**.



Timing diagrams

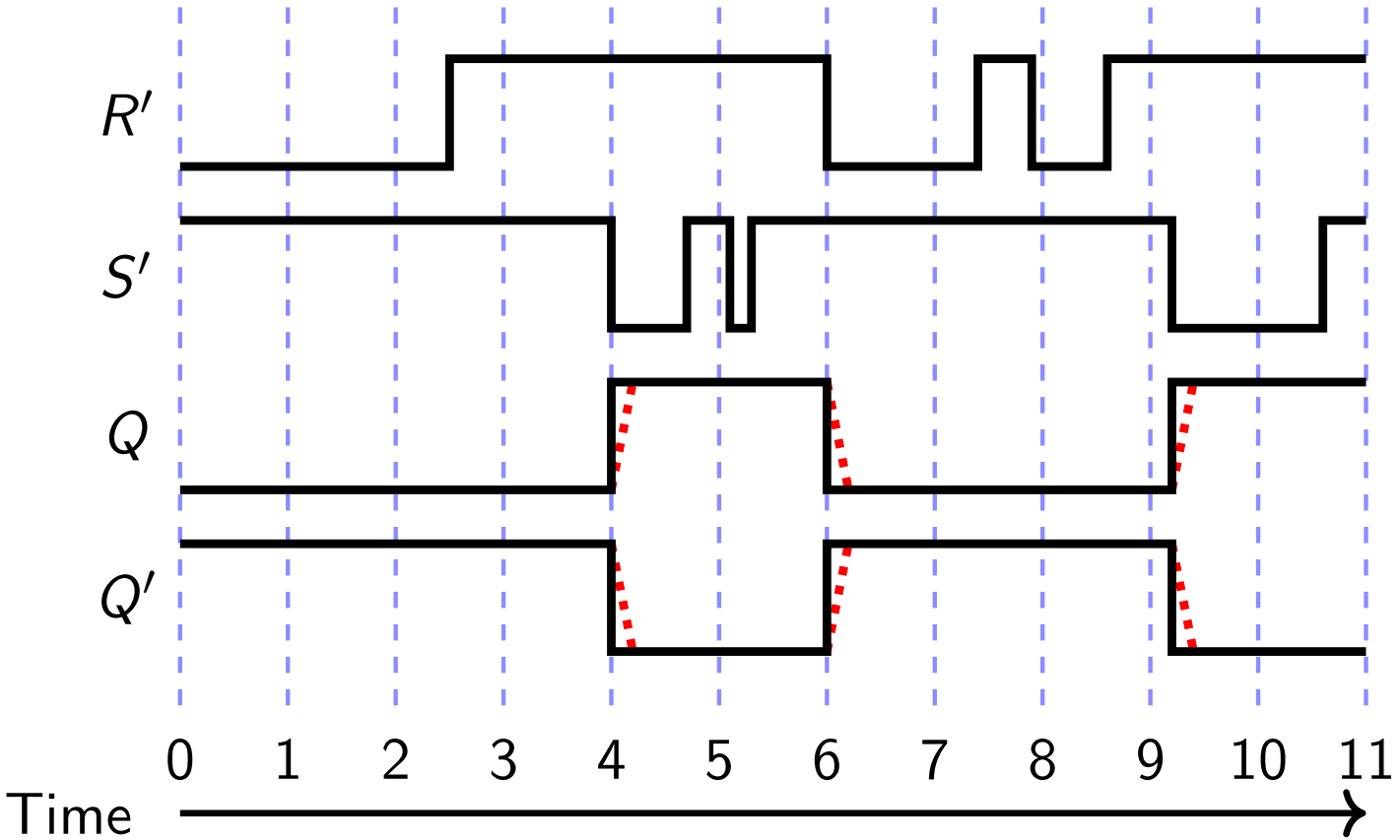
In sequential logic, we use not just truth tables but **timing diagrams**.



This diagram contains a subtle lie — signals don't transition between 1 and 0 instantly! And even if R' and S' did, Q and Q' wouldn't — it takes time (the **propagation delay**) for electricity to pass through the latch.

Timing diagrams

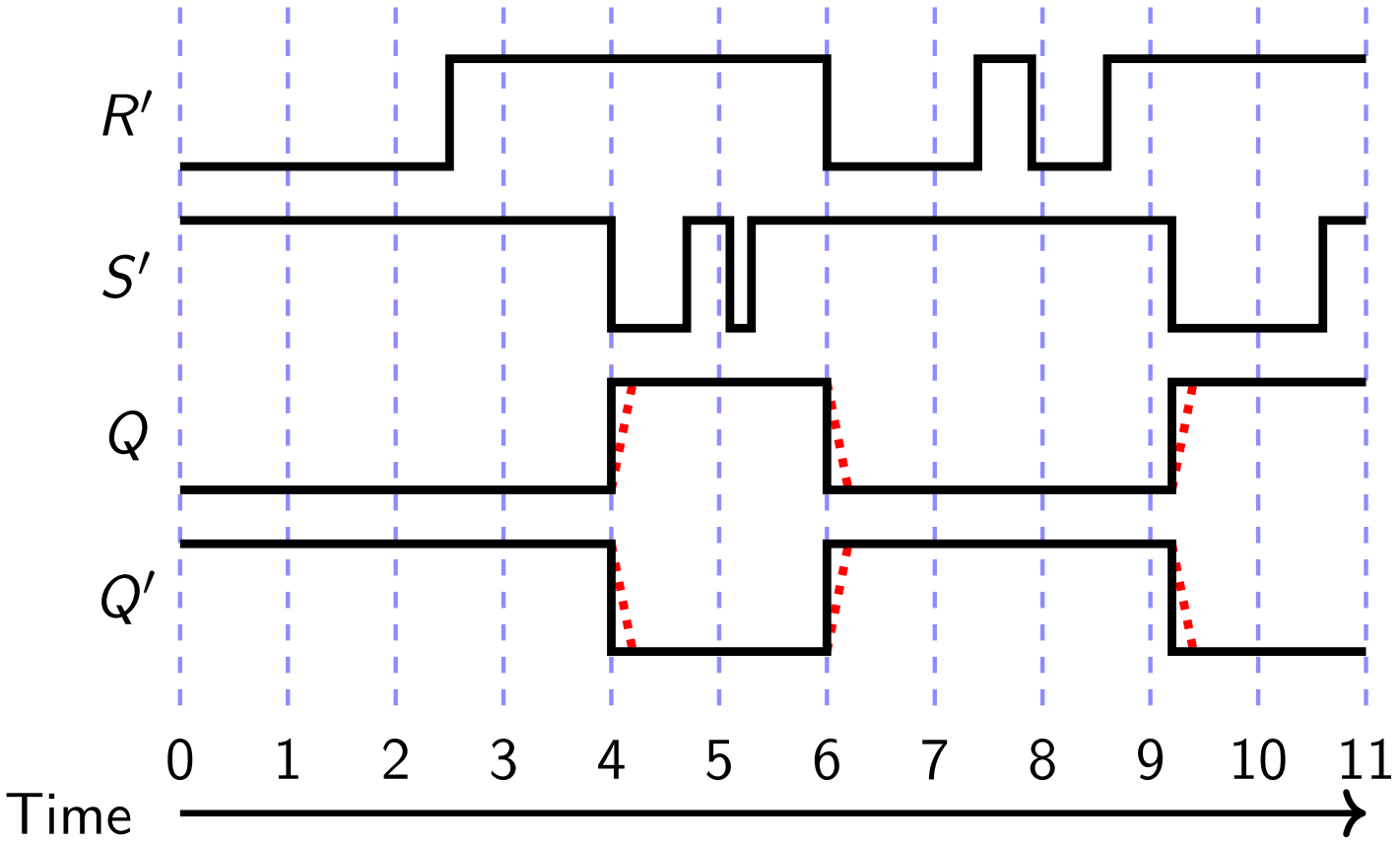
In sequential logic, we use not just truth tables but **timing diagrams**.



This diagram contains a subtle lie — signals don't transition between 1 and 0 instantly! And even if R' and S' did, Q and Q' wouldn't — it takes time (the **propagation delay**) for electricity to pass through the latch.

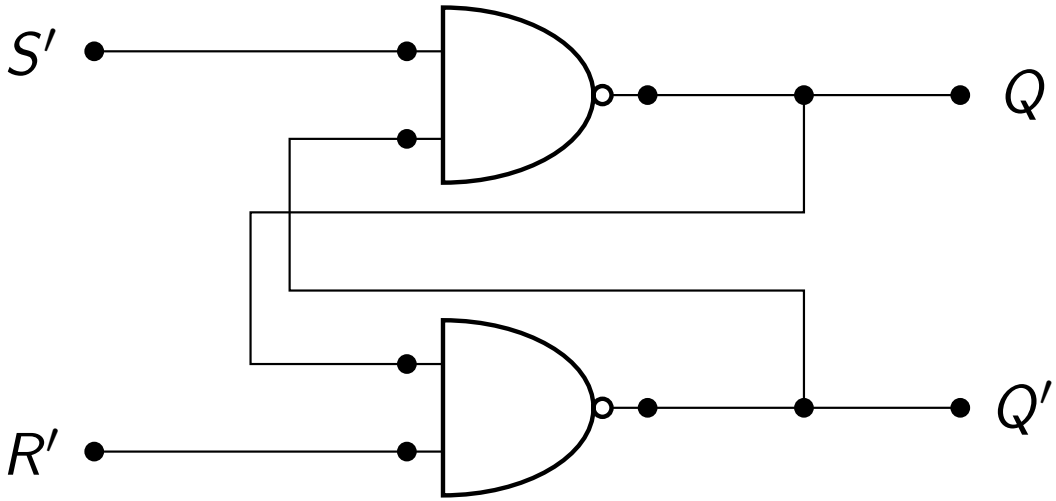
Timing diagrams

In sequential logic, we use not just truth tables but **timing diagrams**.



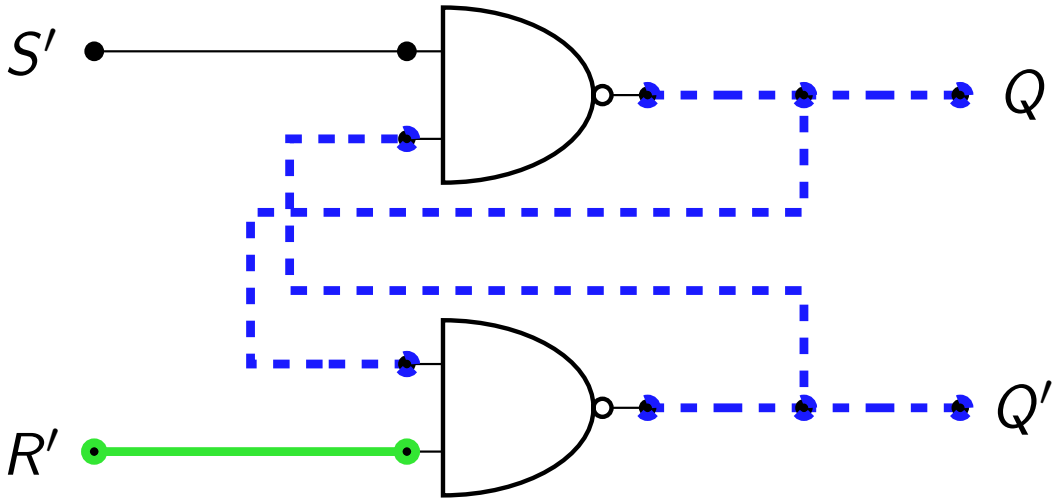
Propagation delay is normally measured in nanoseconds (10^{-9} seconds) or picoseconds (10^{-12} seconds). In this unit we will mostly ignore it, but sometimes it's important — e.g. as a constraint on CPU clock speeds.

Unpacking the mystery box



This is an R-S latch — just two NAND gates! Why does this work?

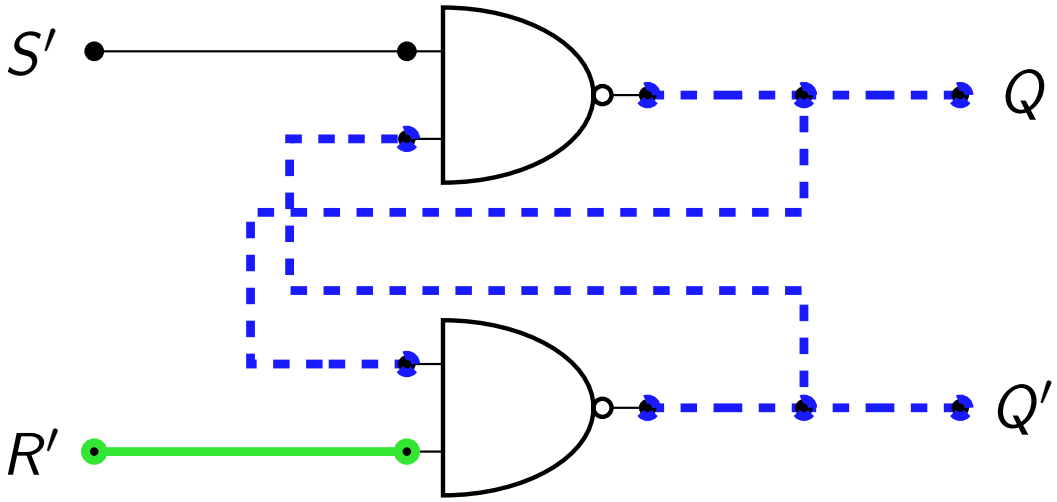
Unpacking the mystery box



This is an R-S latch — just two NAND gates! Why does this work?

Let's say set is active (so S' is low and R' is high). How does this propagate?

Unpacking the mystery box

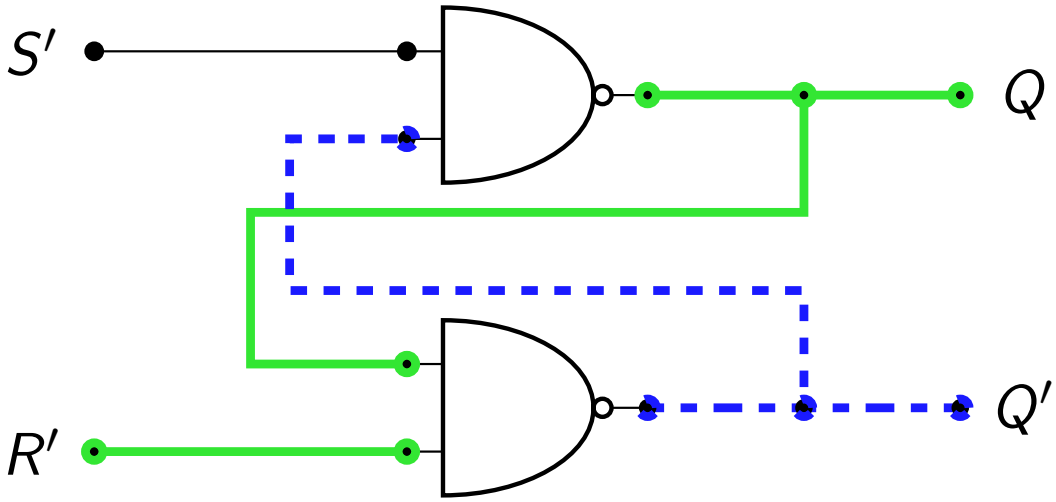


This is an R-S latch — just two NAND gates! Why does this work?

Let's say set is active (so S' is low and R' is high). How does this propagate?

Since S' is low, the top NAND gate must be high regardless of Q'.

Unpacking the mystery box

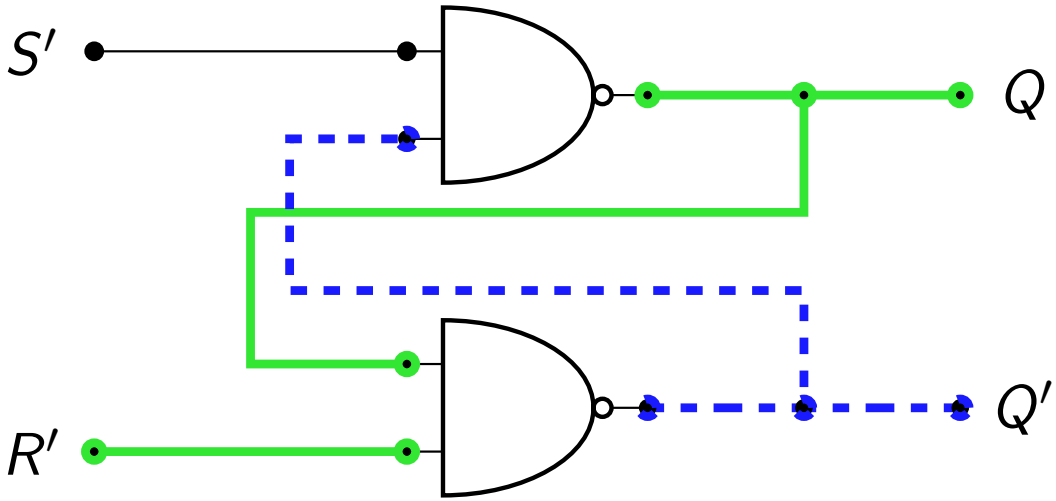


This is an R-S latch — just two NAND gates! Why does this work?

Let's say set is active (so S' is low and R' is high). How does this propagate?

Since S' is low, the top NAND gate must be high regardless of Q'.

Unpacking the mystery box



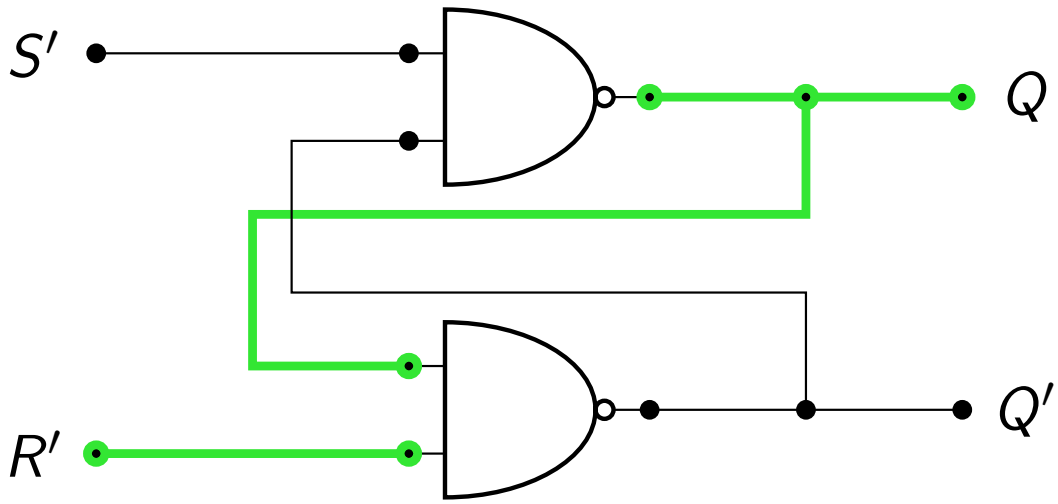
This is an R-S latch — just two NAND gates! Why does this work?

Let's say set is active (so S' is low and R' is high). How does this propagate?

Since S' is low, the top NAND gate must be high regardless of Q'.

Since R' is high too, this forces the bottom NAND gate low. This is stable.

Unpacking the mystery box



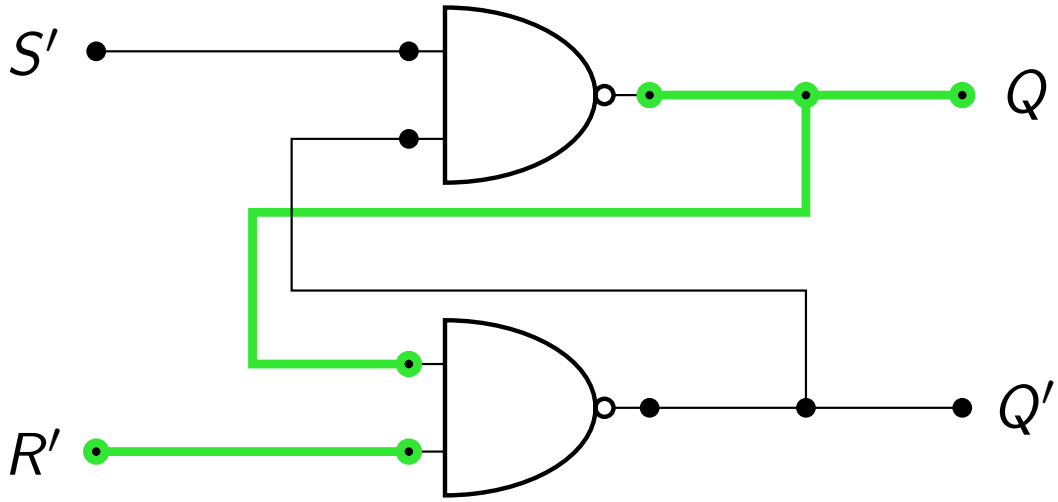
This is an R-S latch — just two NAND gates! Why does this work?

Let's say set is active (so S' is low and R' is high). How does this propagate?

Since S' is low, the top NAND gate must be high regardless of Q'.

Since R' is high too, this forces the bottom NAND gate low. This is stable.

Unpacking the mystery box



This is an R-S latch — just two NAND gates! Why does this work?

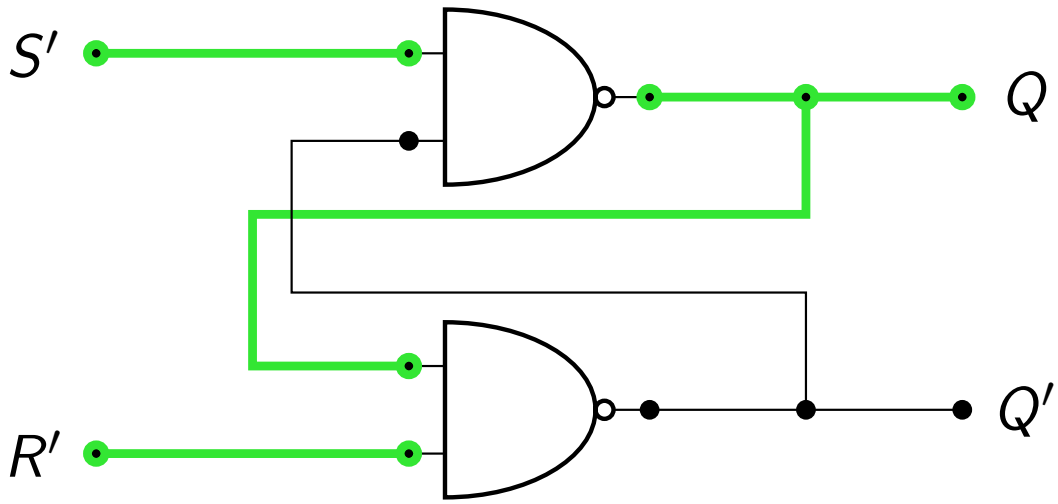
Let's say set is active (so S' is low and R' is high). How does this propagate?

Since S' is low, the top NAND gate must be high regardless of Q'.

Since R' is high too, this forces the bottom NAND gate low. This is stable.

Even after S' goes high again, the outputs stay the same.

Unpacking the mystery box



This is an R-S latch — just two NAND gates! Why does this work?

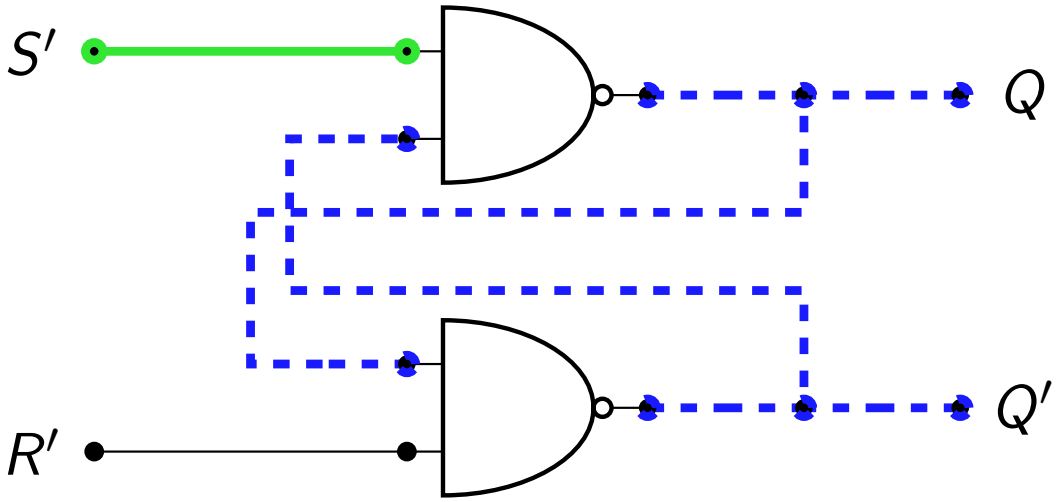
Let's say set is active (so S' is low and R' is high). How does this propagate?

Since S' is low, the top NAND gate must be high regardless of Q'.

Since R' is high too, this forces the bottom NAND gate low. This is stable.

Even after S' goes high again, the outputs stay the same.

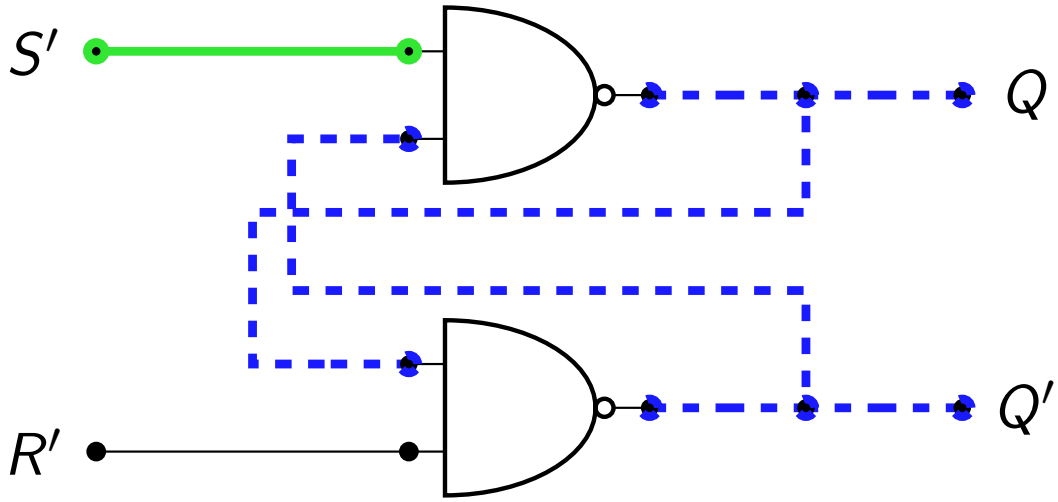
Unpacking the mystery box



This is an R-S latch — just two NAND gates! Why does this work?

Let's now say reset is active (so R' is low and S' is high).

Unpacking the mystery box

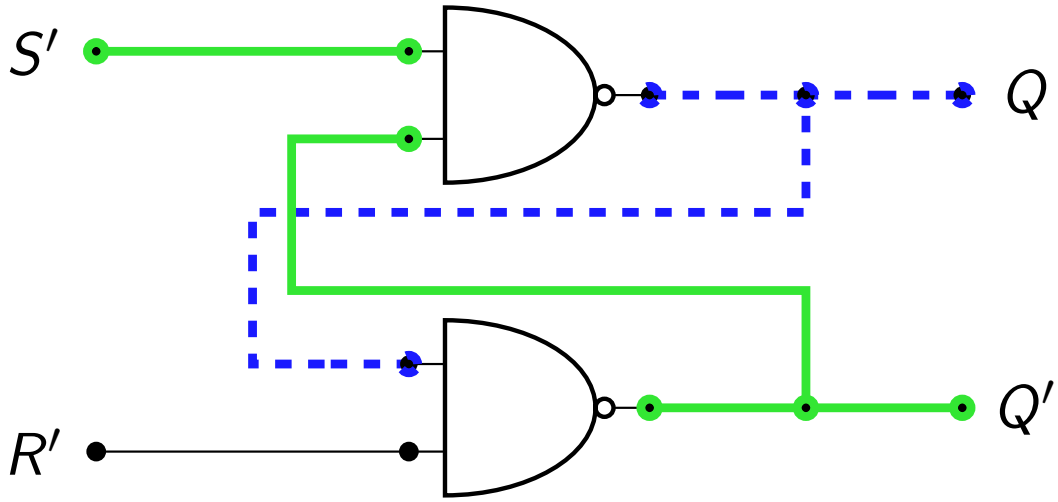


This is an R-S latch — just two NAND gates! Why does this work?

Let's now say reset is active (so R' is low and S' is high).

Since R' is low, the bottom NAND gate must be high regardless of Q.

Unpacking the mystery box

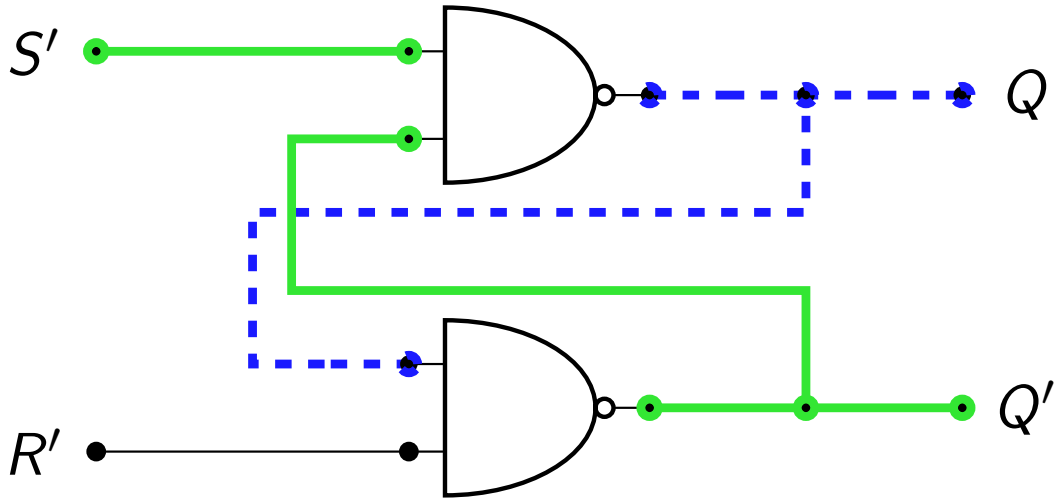


This is an R-S latch — just two NAND gates! Why does this work?

Let's now say reset is active (so R' is low and S' is high).

Since R' is low, the bottom NAND gate must be high regardless of Q.

Unpacking the mystery box



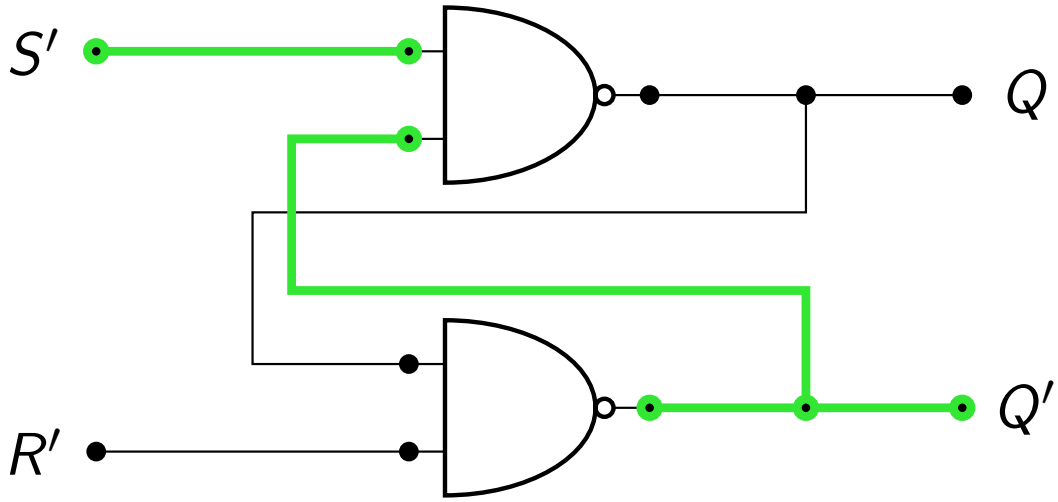
This is an R-S latch — just two NAND gates! Why does this work?

Let's now say reset is active (so R' is low and S' is high).

Since R' is low, the bottom NAND gate must be high regardless of Q.

Since S' is high too, this forces the top NAND gate low. This is stable.

Unpacking the mystery box



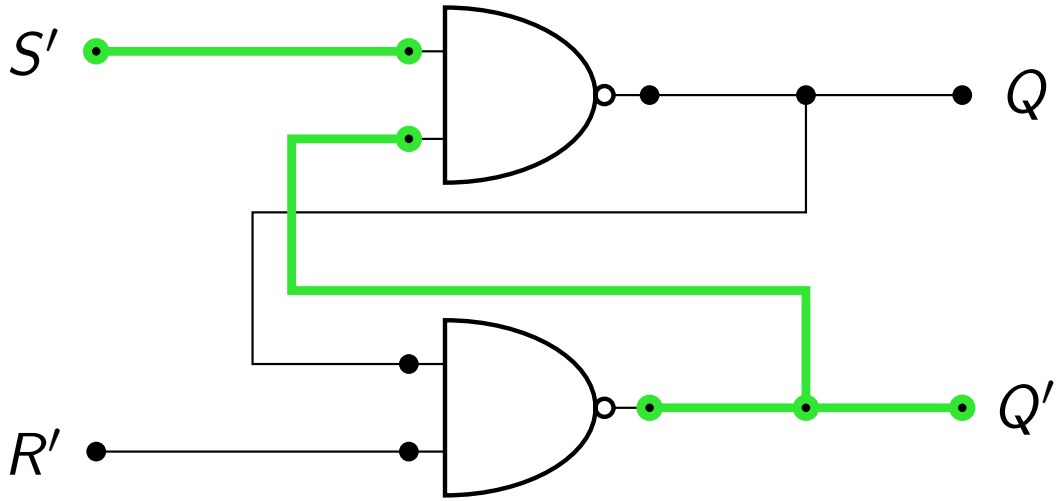
This is an R-S latch — just two NAND gates! Why does this work?

Let's now say reset is active (so R' is low and S' is high).

Since R' is low, the bottom NAND gate must be high regardless of Q.

Since S' is high too, this forces the top NAND gate low. This is stable.

Unpacking the mystery box



This is an R-S latch — just two NAND gates! Why does this work?

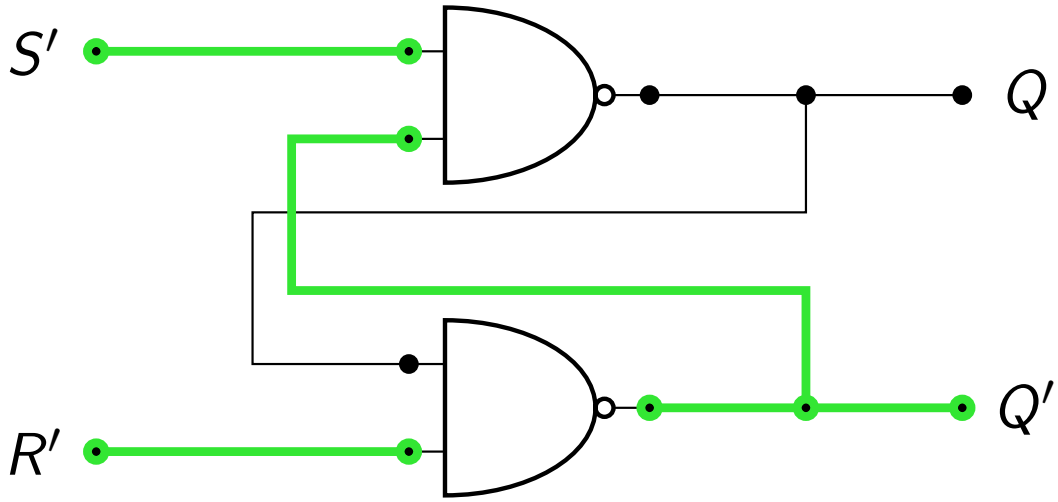
Let's now say reset is active (so R' is low and S' is high).

Since R' is low, the bottom NAND gate must be high regardless of Q.

Since S' is high too, this forces the top NAND gate low. This is stable.

Even after R' goes high again, the outputs stay the same.

Unpacking the mystery box



This is an R-S latch — just two NAND gates! Why does this work?

Let's now say reset is active (so R' is low and S' is high).

Since R' is low, the bottom NAND gate must be high regardless of Q.

Since S' is high too, this forces the top NAND gate low. This is stable.

Even after R' goes high again, the outputs stay the same.