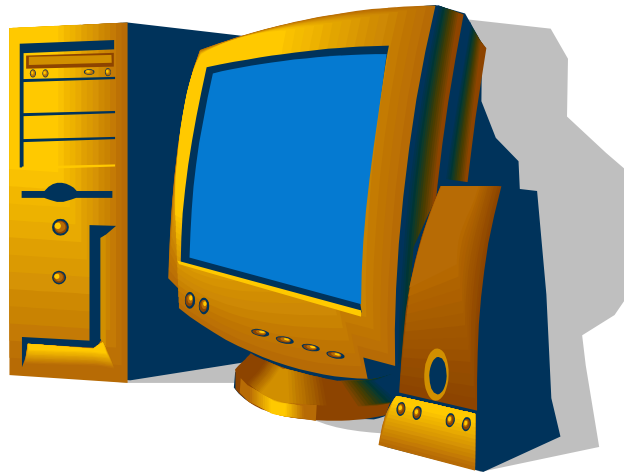


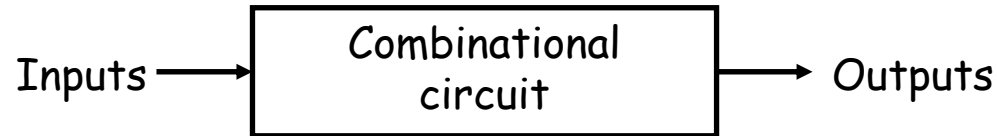
-
- Chapter 5: Sequential Circuits (LATCHES)

Latches

- We focus on **sequential circuits**, where we add memory to the hardware that we've already seen
- Our schedule will be very similar to before:
 - We first show how primitive memory units are built
 - Then we talk about analysis and design of sequential circuits
 - We also see common sequential devices like registers and counters

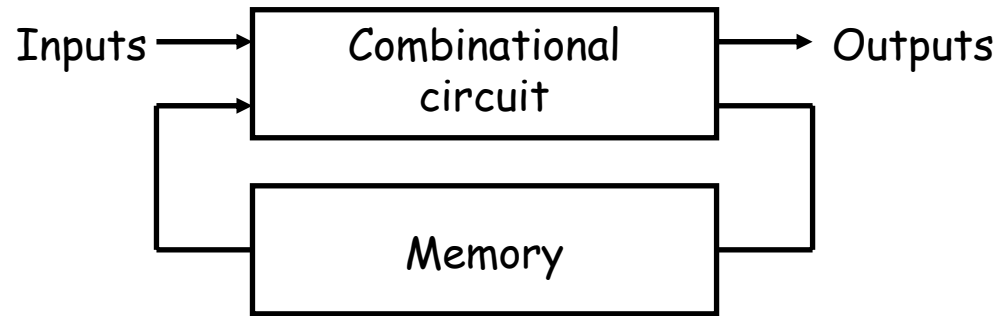


Combinational circuits



- 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

Sequential circuits



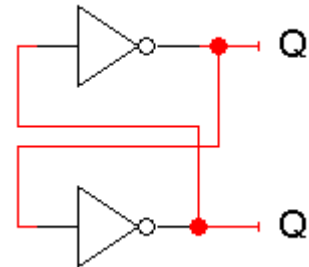
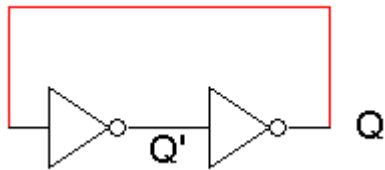
- In contrast, the outputs of a **sequential circuit** depend on not only the inputs, but also the **state**, or the current contents of some memory
- This makes things more difficult to understand, since the same inputs can yield *different* outputs, depending on what's stored in memory
- The memory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

What exactly is memory?

- A memory should have at least three properties
 1. It should be able to hold a value.
 2. You should be able to *read* the value that was saved
 3. You should be able to *change* the value that's saved
- We'll start with the simplest case, a one-bit memory
 1. It should be able to hold a single bit, 0 or 1
 2. You should be able to read the bit that was saved
 3. 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

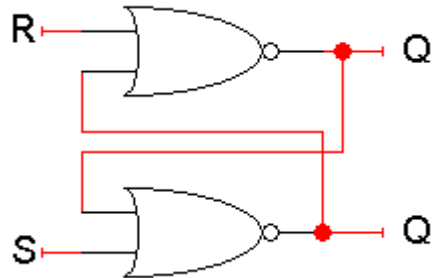
- How can a circuit “remember” anything, when it’s just a bunch of gates that produce outputs according to the inputs?
- The basic idea is to make a loop, so the circuit outputs are also inputs
- Here is one initial attempt, shown with two equivalent layouts:



- 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 gates instead of inverters. The **SR latch** below has two inputs S and R, which will let us control the outputs Q and Q'



- Here Q and Q' feed back into the circuit. They're not only outputs, they're 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':

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

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

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

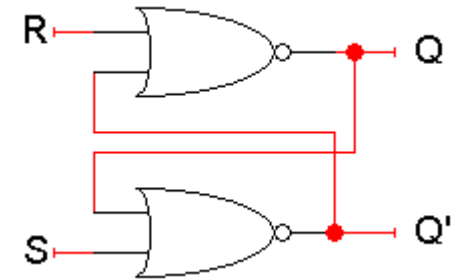
Storing a value: $SR = 00$

- What if $S = 0$ and $R = 0$?
- The equations on the right reduce to:

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

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

- So when $SR = 00$, then $Q_{\text{next}} = Q_{\text{current}}$.
Whatever value Q has, it keeps
- This is exactly what we need to **store** values in the latch.



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

Setting the latch: $SR = 10$

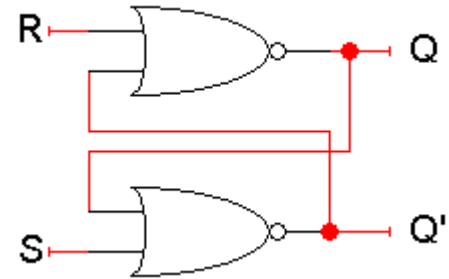
- What if $S = 1$ and $R = 0$?
- Since $S = 1$, Q'_{next} is 0, *regardless* of Q_{current} :

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

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

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

- So when $SR = 10$, then $Q'_{\text{next}} = 0$ and $Q_{\text{next}} = 1$
- This is how you **set** the latch to 1. The S input stands for "set"
- 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**

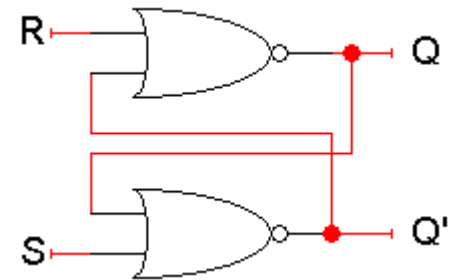


$$Q_{\text{next}} = (R + Q'_{\text{current}})'$$
$$Q'_{\text{next}} = (S + Q_{\text{current}})'$$

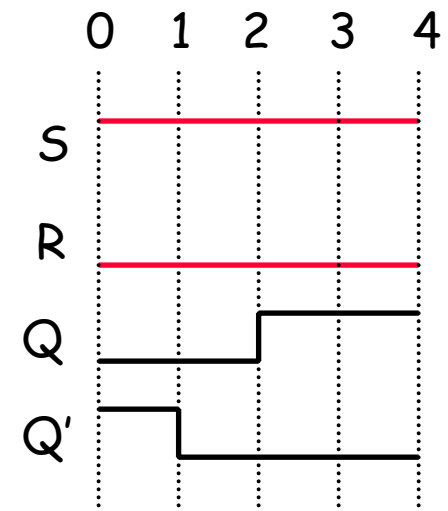
Latch delays

- Timing diagrams are especially useful in understanding how sequential circuits work
- Here is a diagram which shows an example of how our latch outputs change with inputs SR=10

0. Suppose that initially, $Q = 0$ and $Q' = 1$
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)
2. This change in Q' , along with $R=0$, causes Q to become 1 after another gate delay
3. The latch then stabilizes until S or R change again



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



Resetting the latch: $SR = 01$

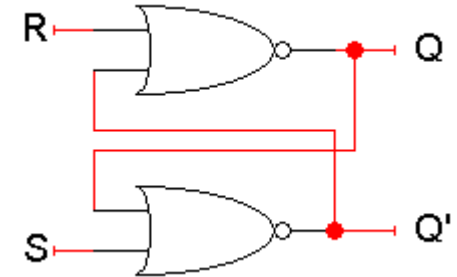
- What if $S = 0$ and $R = 1$?
- Since $R = 1$, Q_{next} is 0, *regardless* of Q_{current} :

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

- Then, this new value of Q goes into the bottom NOR gate, where $S = 0$

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

- So when $SR = 01$, then $Q_{\text{next}} = 0$ and $Q'_{\text{next}} = 1$
- 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}



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

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

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

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

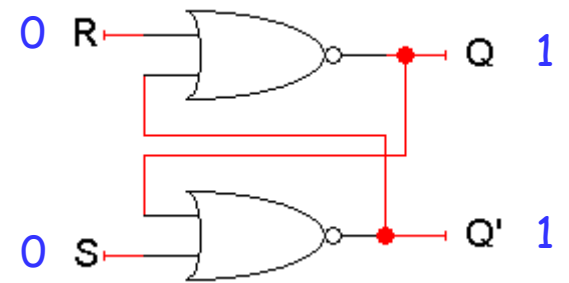
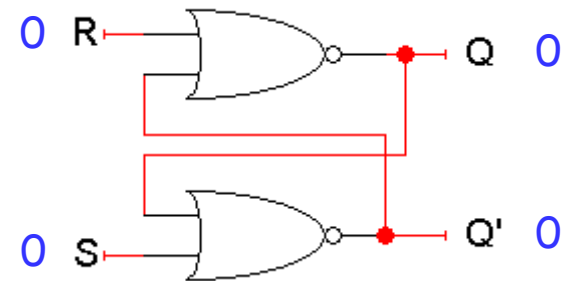
- But these new values go back into the NOR gates, and in the next step we get:

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

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

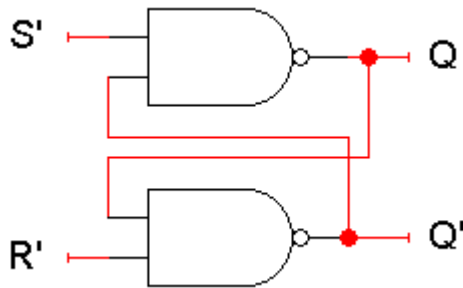
- The circuit enters an infinite loop, where Q and Q' cycle between 0 and 1 forever
- This is actually the worst case, but the moral is don't ever set $SR=11$!

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



S'R' latch

- There are several varieties of latches
- You can use NAND instead of NOR gates to get a **S'R' latch**

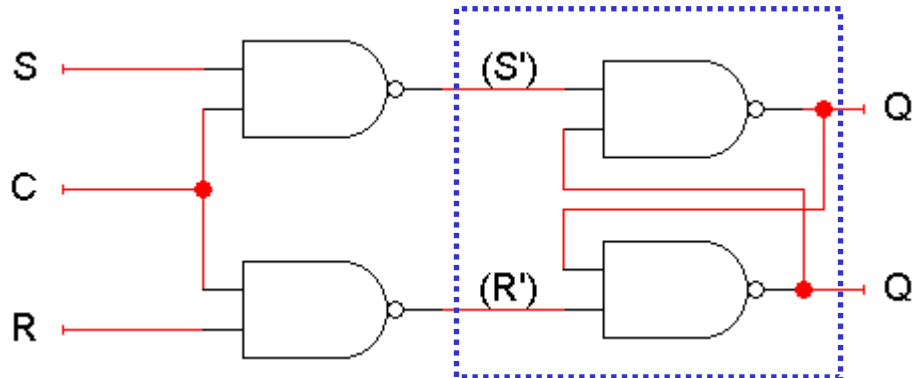


S'	R'	Q
1	1	No change
1	0	0 (reset)
0	1	1 (set)
0	0	Avoid!

- This is just like an SR latch, but with inverted inputs, as you can see from the table
- You can derive this table by writing equations for the outputs in terms of the inputs and the current state, just as we did for the SR latch

An SR latch with a control input

- Here is an SR latch with a control input C

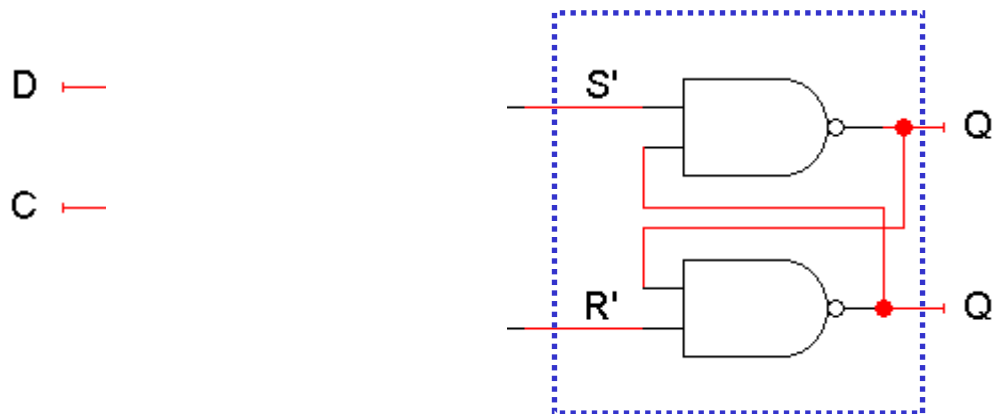


C	S	R	S'	R'	Q
0	x	x	1	1	No change
1	0	0	1	1	No change
1	0	1	1	0	0 (reset)
1	1	0	0	1	1 (set)
1	1	1	0	0	Avoid!

- Notice the hierarchical design!
 - The dotted blue box is the $S'R'$ latch from the previous slide
 - The additional NAND gates are simply used to generate the correct inputs for the $S'R'$ latch
- The control input acts just like an enable

D latch

- Finally, a D latch is based on an S'R' 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 1, 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!

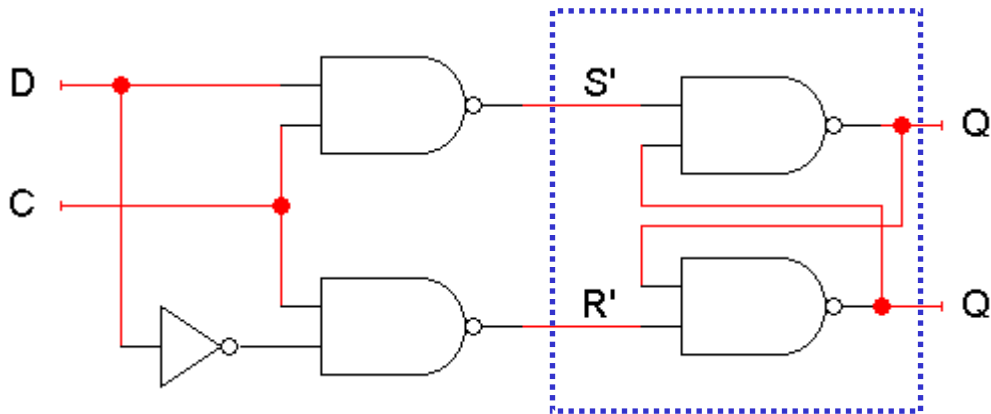


C	D	Q
0	x	No change
1	0	0
1	1	1

- Also, this latch has no "bad" input combinations to avoid. Any of the four possible assignments to C and D are valid.

D latch

- Finally, a D latch is based on an S'R' 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 1, 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!

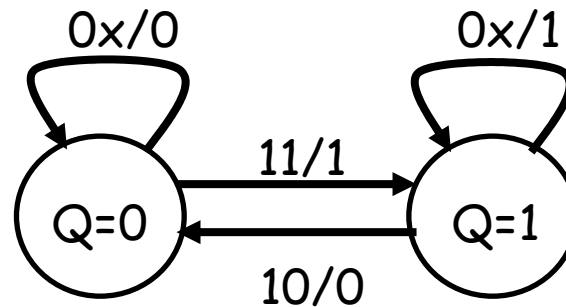


C	D	Q
0	x	No change
1	0	0
1	1	1

- Also, this latch has no "bad" input combinations to avoid. Any of the four possible assignments to C and D are valid.

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



- 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
 - But 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

Summary

- A sequential circuit has memory. It may respond differently to the same inputs, depending on its current state
- Memories can be created by making circuits with feedback
 - Latches are the simplest memory units, storing individual bits
 - It's difficult to control the timing of latches in a larger circuit
- Next, we'll improve upon latches with **flip-flops**, which change state only at well-defined times. We will then use flip-flops to build all of our sequential circuits

