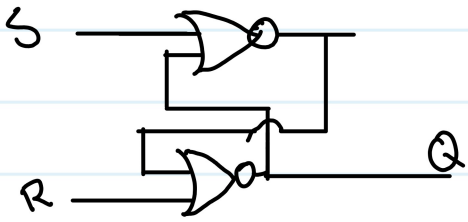
Latches & Flip Flops:

Suppose that, only using the tools we have available in combinational logic, we wanted to create a logic system that gave outputs based on the current state as well as the given input. This would be known as sequential logic, because to get a certain set of outputs you have to follow a certain "sequence" of steps. But how do we do this?

Well, we need some way to keep a bit in the system, and the way we can do that is through a "feedback loop" where we create some kind of loop from the output back into the system to keep track. The simplest system that does this is an SR (Set-Reset) latch. This device has 2 inputs, one to set the output high, and one to reset it to low. Once the output is set high, however, it will remain that way until reset is pushed. The schematic of it is below!

$Q=((S+Q)'+R)'$

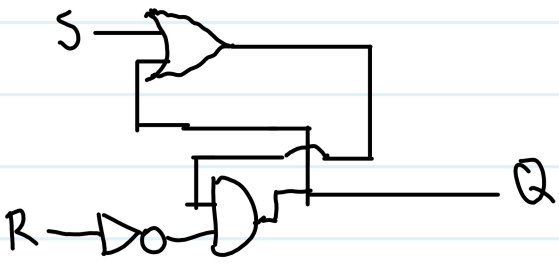


NOR:

00 = 1

1X = 0

If you take a second to look at this, and trace it, it kinda makes sense, and with this design we get the added benefit of having the output from the first gate being equal to Q! When I tried to do this the first time, however, I came up with a circuit that made more sense to me but had more gates!

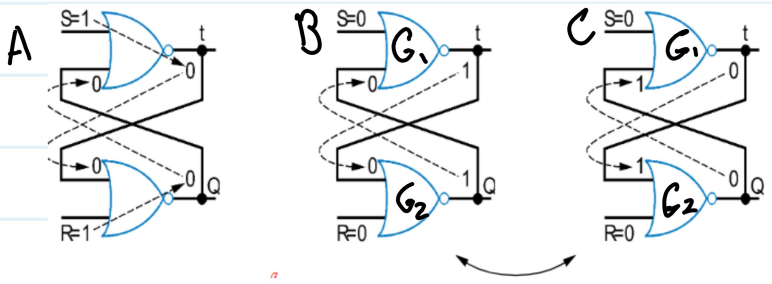


$Q=(S+Q)R'$

Because here we see that, if S or Q is high & R is low, then Q would be high, and that made sense, but with a little bit of boolean algebra we can show these to be the same:

$$Q = ((S+Q)' + R)'$$
$$= ((S+Q)')' R' \quad : \text{Demorgan's Law } (a+b)' = a'b'$$
$$= (S+Q) R' \quad : \text{Involution Law } (a')' = a$$

But the SR latch has a big big problem. What happens if we like multi-track drifting and press both at the same time? Well it comes down to gate delay's, something we can't ignore here. In our good design with the NOR gates, our signals have to propagate through 2 gates until Q is reliable, first it has to go through the Q' gate, and then the reset gate. If both are pressed and held high, Q=0, it's easy, reset overrides everything. But what happens when we release them to 00?



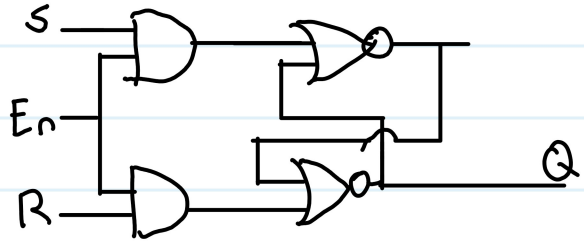
A. Start, Q'=1 b/c of S=1, Q=0 b/c of R=1

B. SR=00, Q was 0 so Q' updates to 1, but while G_1 is propagating, G_2 also propagates, but since at the moment Q' & R'=0, Q incorrectly assigns to 1

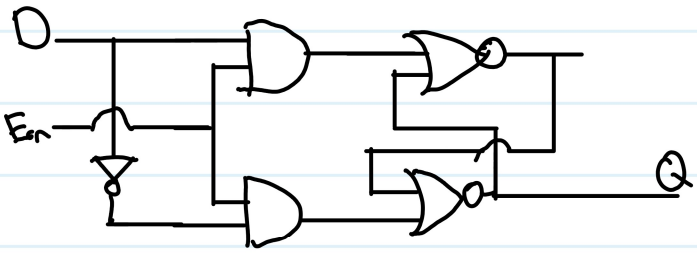
C. Now that Q' & Q had a chance to propagate, G_1 runs again, but now Q=1 so Q'=0, but again, at the same time, G_2 is seeing that Q' is high so it updates Q=0 and when they check again, we're back to step B

So now, our Q signal (in a perfect gate) will oscillate forever until a different input is passed. Oofy doofy. Realistically, however, it will oscillate for a bit and then settle on an undetermined value because gates aren't perfectly the same.

Okay, So let's try fixing this, what if we added a "Enable" pin to make things happen only periodically? Well, then we could avoid S=R=1 from glitches, and if we wanted, we could use an outside circuit to ensure R=1 only when S=0. This is called a level-Sensitive SR latch, because it's output is sensitive to the Enable level. It looks like this:



But we still have the problem of having to deal w/ensuring that SR=11 doesn't happen. If we don't need reset to be an offered feature, then we can put an inverter on the S pin to make reset happen automatically and form what we call a D-Latch

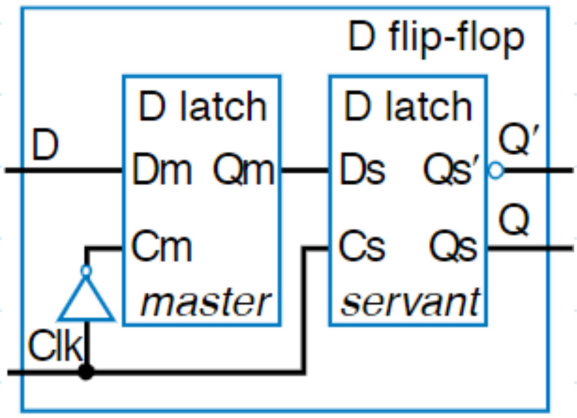


The big thing to note here is that Q is entirely dependant on D. if D is high then Q is high & the same for low. This is a very important difference from S-R where once S became high, Q stayed high regardless of S. Now, unlike in SR,

holding doesn't happen automatically. In SR, S could be 0 and Q could be 1 regardless of En being 1 or 0. Now, however, with a D-latch, the enable pin is very important because it enables updates. and Q only holds when En=0.

But if gate delays are a thing, and Q can only update when En=1, then what happens if En is only high for a very short amount of time? Well, then we're going to create a race condition like we saw when SR=11 because the signal doesn't have time to completely propagate before it changes again. But also, if En is high for a long, but slightly fluctuating time, and we're chaining latches together to store multiple bits, how do we consistently know how many get properly updated? We don't. We need to come up with a system that doesn't care how long En is high, only that it does go high.

What if we created a device that stores any bit while the clock is low, and then actually loaded/output it when the clock goes high, but once the clock is high, we stop looking at the input? Then, we could chain the devices together, knowing that a signal will only propagate one device per cycle. This would look something like this:



So now we have the building blocks, we need to talk about some important terms to highlight some differences.

Synchronous - Changes in the circuit can only happen at precise moments determined by the clock

Asynchronous - Changes in the circuit can happen at any given moment

Latches, as you might guess, are asynchronous when En is high, whereas flipflops are synchronous, only happening exactly when the clock goes high