We need to store bits in our circuits now. Essentially a good example of this is something like a lamp that only has one button.

When you press the button the lamp circuitry has to know what the current state is. If it is off, it will see that it is off and then turn it on. If it is on then it will need to know it is on and then turn it off.

We are going to cover 4 ways to store information:
More can be found here @ all about circuits: http://www.allaboutcircuits.com/vol_4/chpt_10/3.html

1. Basic SR Latch
2. Level sensitive SR Latch
3. level sensitive D latch
4. edge triggered D flip-flop //I feel like this may be the one that we are using mostly in lab #6.

The first attempt to save something was feed back using an or gate with the input of S and output of Q feeding back to the second input. This was kind of successful in some way, however, it then completely fails, as once you set S =1 you can never again unset Q.

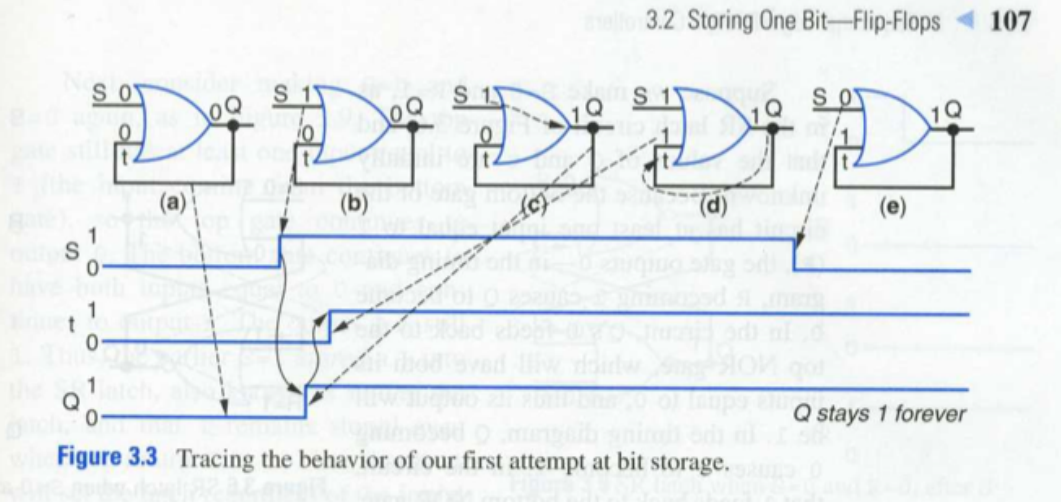


Figure 3.3 Tracing the behavior of our first attempt at bit storage.

1. Basic SR Latch
This is a pretty simple circuit that will allow me to save data. It is comprised of two NOR gates.

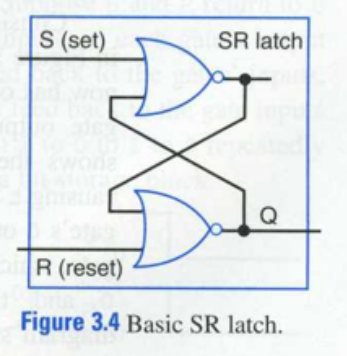


Figure 3.4 Basic SR latch.

There is one problem with this though.We can not make both set and reset to be equal to one. If this becomes true then we will have what is known as oscillation. The value of Q may go from 0 to 1 to 0 to 1 over and over and over...

Truth Table for S–R Flip-Flop

R	S	Q	Action
0	0	Last value	No change
0	1	1	Set
1	0	0	Reset
1	1	–	Invalid condition

2. Level Sensitive SR Latch

This type of latch is a possible solution to the SR=11 oscillation problem by adding a third input of Enable.

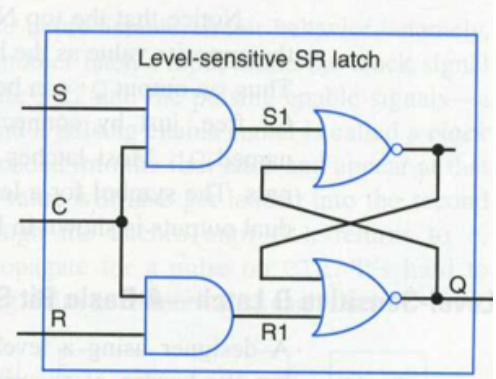
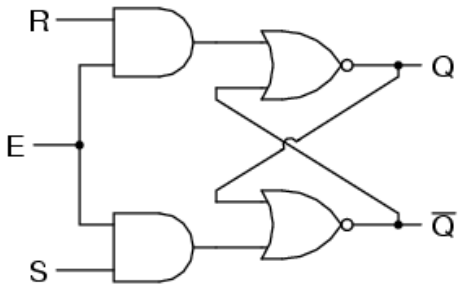


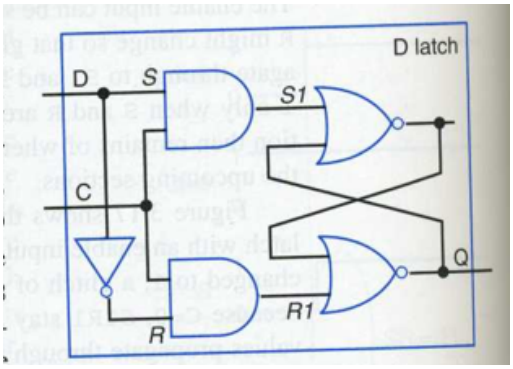
Figure 3.16 Level-sensitive SR latch—an SR latch with enable input C.

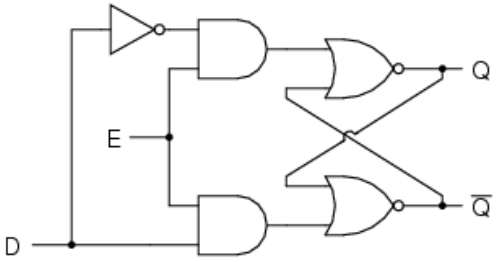
Here in this image the C is the enable input. This latch is made of two and gates and two nor gates.



E	S	R	Q	$\overline{Q}$
0	0	0	latch	latch
0	0	1	latch	latch
0	1	0	latch	latch
0	1	1	latch	latch
1	0	0	latch	latch
1	0	1	0	1
1	1	0	1	0
1	1	1	0	0

Level Sensitive D-Latch





E	D	Q	$\overline{Q}$
0	0	latch	latch
0	1	latch	latch
1	0	0	1
1	1	1	0

Edge Triggered Flip-Flop
This solves a problem with D-latch as seen below

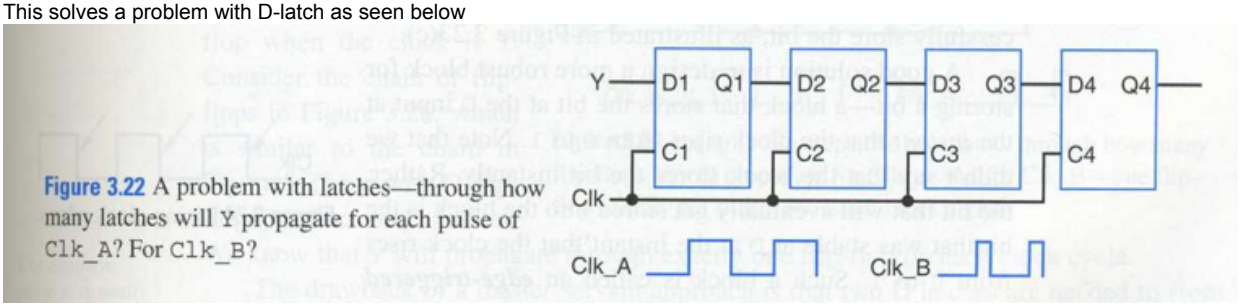


Figure 3.22 A problem with latches—through how many latches will `Y` propagate for each pulse of `Clk_A`? For `Clk_B`?

Basically the outputs from latches can propagate through to the input of others and cause the values to change when we do not want them to. I do not quite understand this, however, the edge triggered flip-flop is supposed to fix this problem.

Q: How the fuck does it do this?
A: It will store the value on every clock edge. We tend to stick with positive clock edges.

This seems to be done in verilog with block assigns…

```
always @(posedge clk)
begin
    nutsacks <= q;
end
```

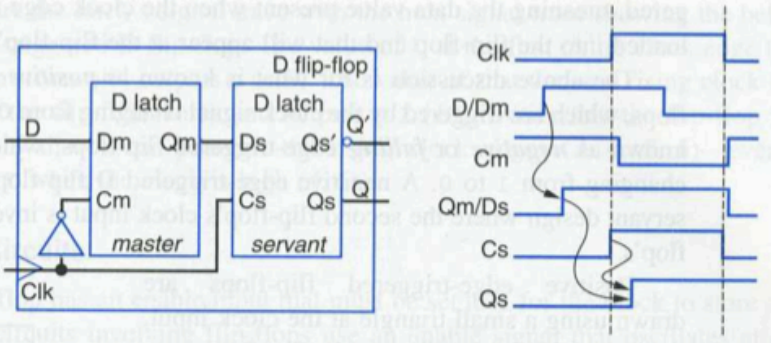


Figure 3.25 A D flip-flop implementing an edge-triggered bit storage block, internally using two D latches in a master-servant arrangement. The master D latch stores its `Dm` input while `Clk = 0`, but the new value appearing at `Qm`, and hence at `Ds`, does not get stored into the servant latch, because the servant latch is disabled when `Clk = 0`. When `Clk` becomes `1`, the servant D latch becomes enabled and thus gets loaded with whatever value was in the master latch at the instant that `Clk` changed from `0` to `1`.