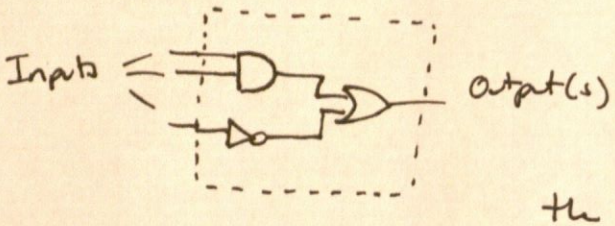
Lecture 4

So far we have only learned about <u>combinational logic</u>,



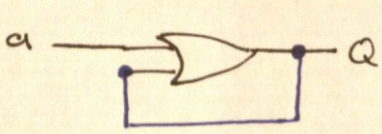
where the outputs are simple combinations based on inputs alone.

Combinational logic always responds the same way to inputs, regardless of past use (history). (e.g. light switch).

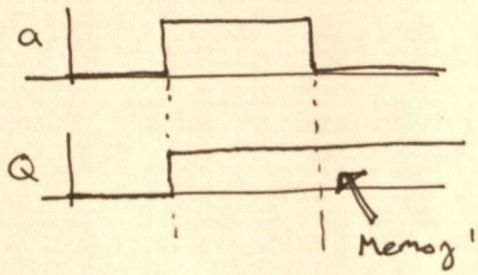
But suppose you want to do something <u>sequentially</u>, like cycle through a traffic light in a specified order. In this case, the response of the system depends both on inputs (e.g. pressing a button) <u>and</u> the current state (if the light is green, change to yellow; if yellow, change to red; if red, change to green).

The system must know its current <u>state</u>: it needs <u>memory</u>.

Consider the effect of <u>feedback</u>

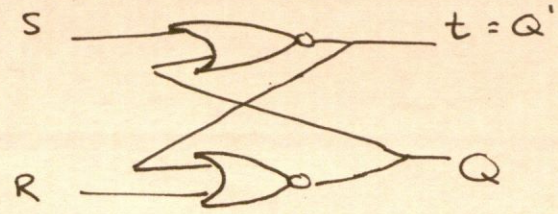


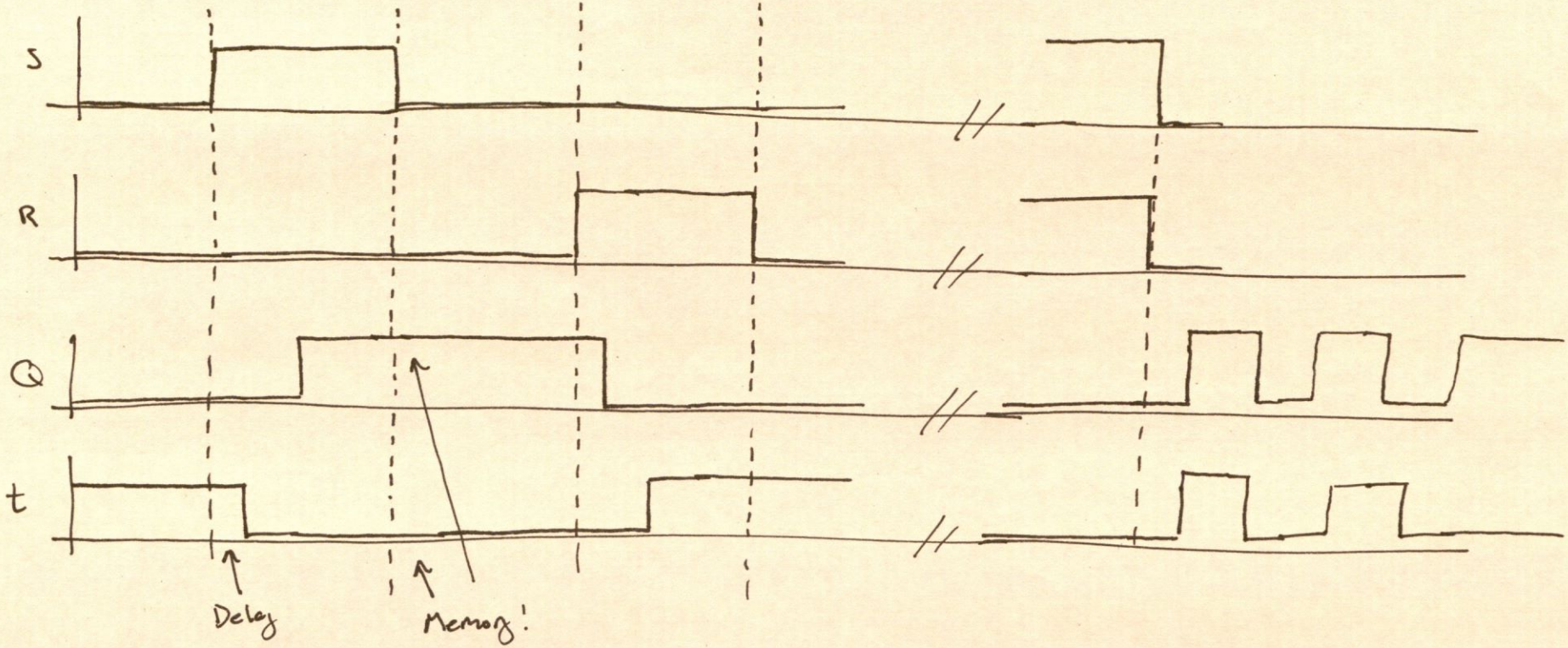
(Bad) Example



But can't switch it back.

SR Latch





S "sets" the latch

R "resets" the latch.

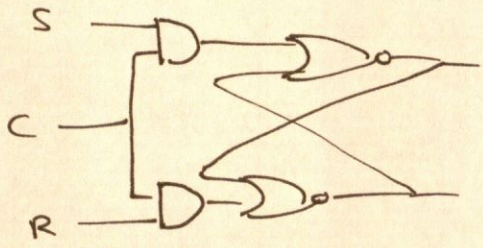
Note that $t = Q'$ during normal operation

Problem with SR Latch

"Race Condition" commits the cardinal sin of digital electronics: you don't know what the output will be!

Can we prevent SR = "11" from occurring? – No.

Can we stop it from causing problems?



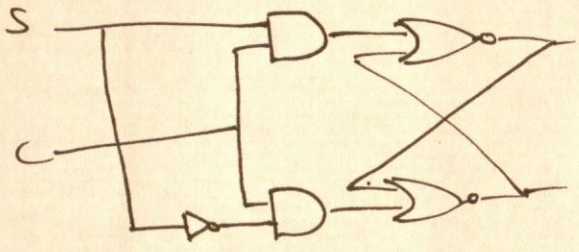
Enabled SR Latch

C = Enable signal.

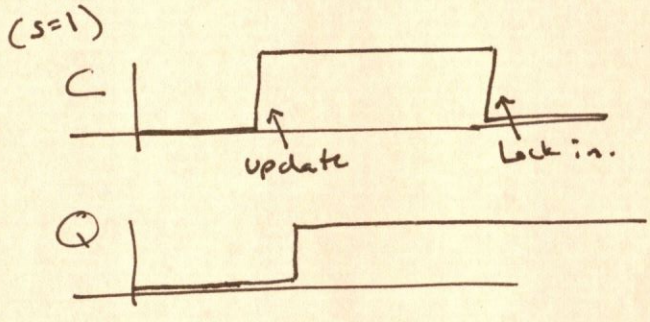
But: same problem if SRC = "111"

which we still can't prevent.

D-Latch

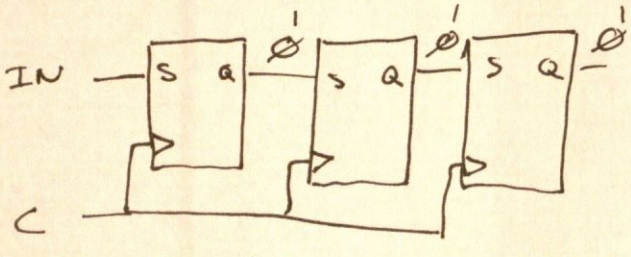


"R" is now automatically S' – no problem!



While C is on, the latch is transparent: as S changes, Q will change. When C turns off, Q gets "locked in".

Final Problem

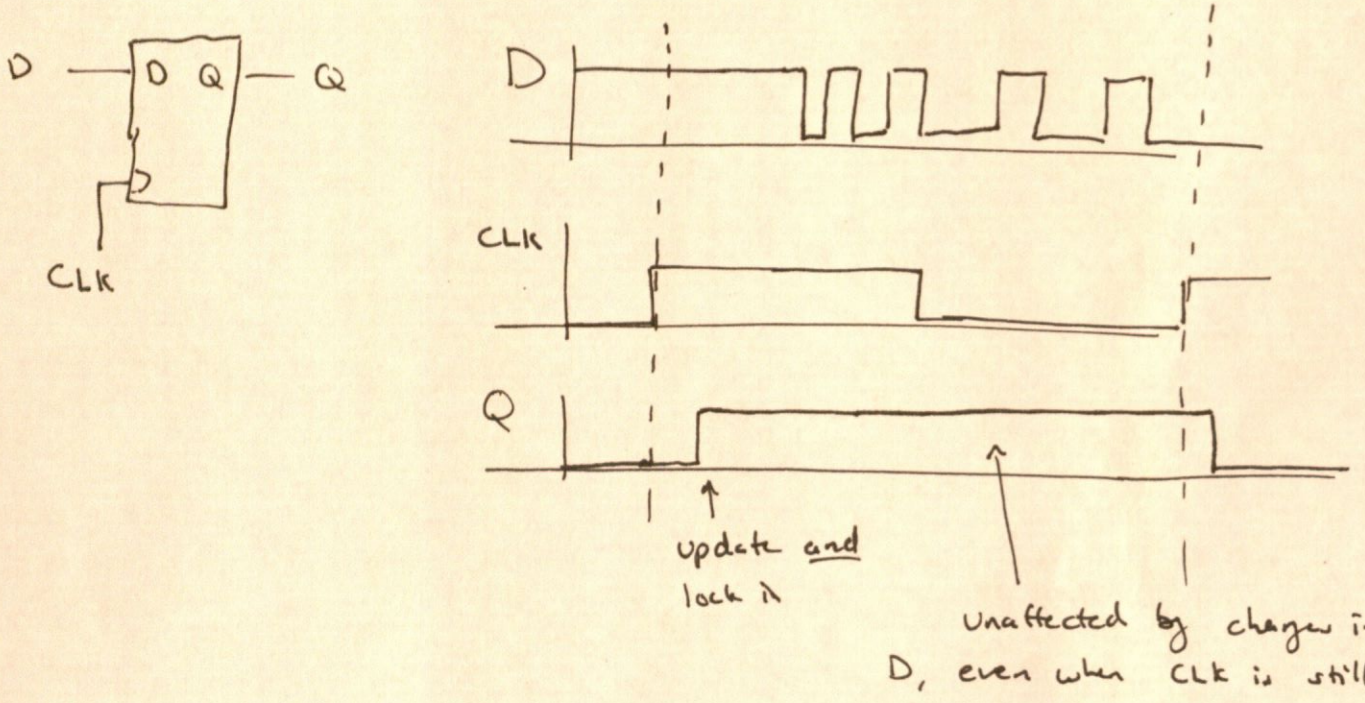


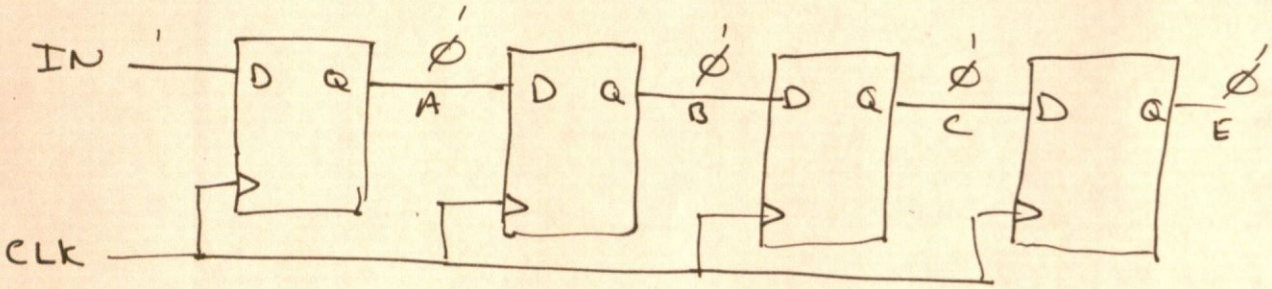
While C is on, the IN signal will race through as many latches as it can until C turns off.

1) Not very elegant
2) Impossible to predict (!)

We need a solution that updates and locks in at the same time.

D Flip Flop "Edge Triggered"





IN = 1, others start at 0