# Karnaugh Maps

- Yet another way of deriving the simplest Boolean expressions from behaviour.
  - Easier than using algebra (which can be hard if you don't know where you're going).
- Example

| A | B | C | X |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

each 1 here gives a minterm e.g. A'BC

# Karnaugh Maps

- A Karnaugh map sets out the minterms pictorially.
- Example for 3 variables

| A \ BC | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | | | 1 | |
| 1 | 1 | | 1 | 1 |

- Like a truth table
  - each 1 represents the presence of that minterm in the CSOP form

- Not like a truth table
  - set out differently (each column/row differs in 1 variable only from its neighbours).

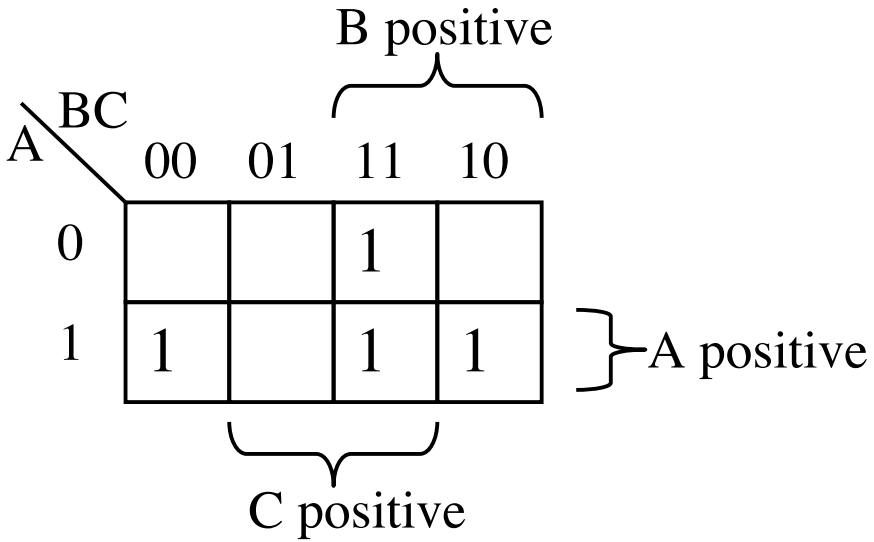
| A \ BC | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | *1* | *2* | *4* | *3* |
| 1 | *5* | *6* | *8* | *7* |

  - numbers correspond to rows in a truth table

# Mechanics and Semantics

- Different algebraic expressions are generated by recognising patterns and grouping **adjacent** cells.
- Loop adjacent cells in **$2^x$ sized** groups (i.e. 2,4,8, ….)
- Try to form **as few groups as possible** (i.e. groups are as large as possible)
- Adjacency can wrap round the edges, so e.g. the four corners are all adjacent.
- Algebraically equivalent to eliminating terms of the form A + A'.
  - (recall a common simplification is to use DISTR to take out a common factor, and OR to make B + B' = 1, e.g. AB + AB' = A )

# Example 1




- (confirm this by Boolean algebra)

- before grouping

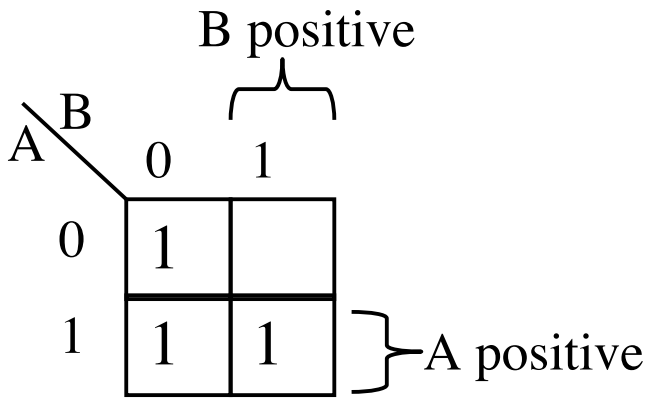
  …

- after grouping

  …

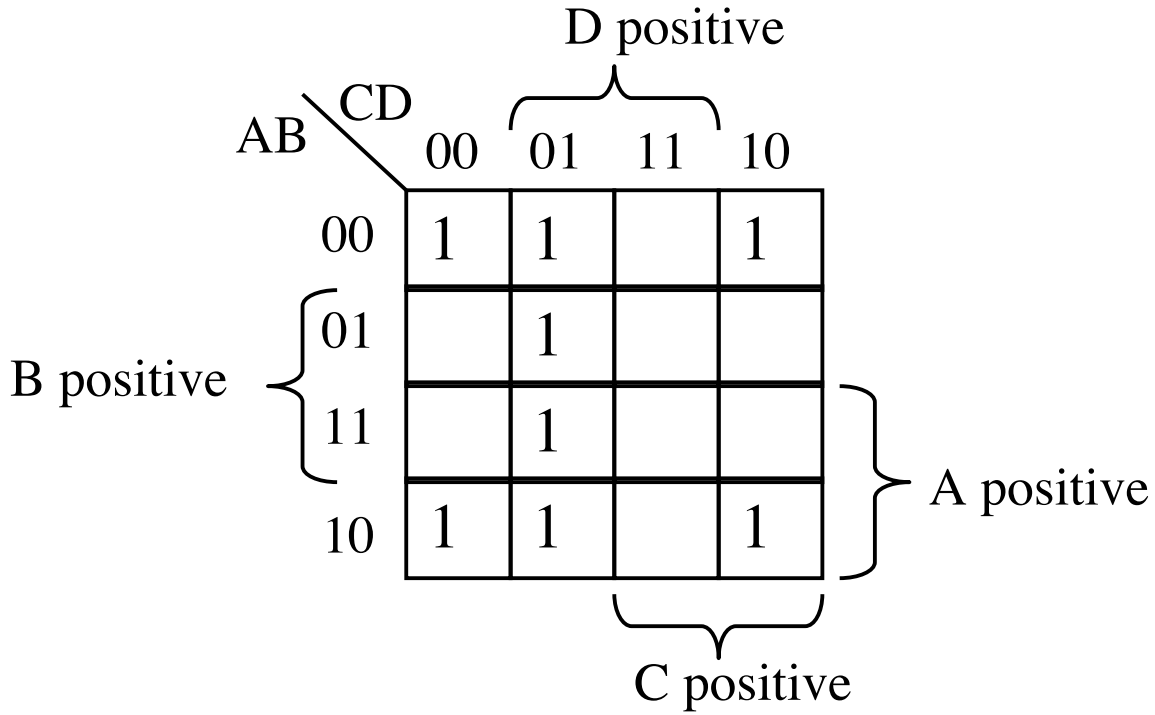
# Example 2 (Karnaugh map in 2 vars)

- A'B' + AB' + AB



# Example 3 (Karnaugh map in 4 vars)

- A'B'C'D' + A'B'C'D + A'B'CD' + A'BC'D + ABC'D + AB'C'D' + AB'C'D + AB'CD'



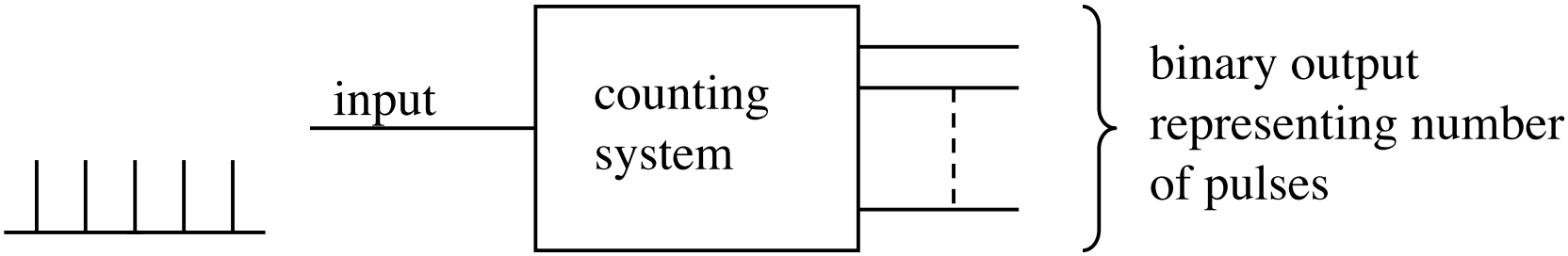
# Sequential Logic Systems

- The defining characteristic of a sequential logic system (SLS) is that the outputs of the system depend not only on the present inputs to the system, but also on **the past history** of the inputs and outputs of the system

- i.e. SLSs have a form of **memory.**

# Example

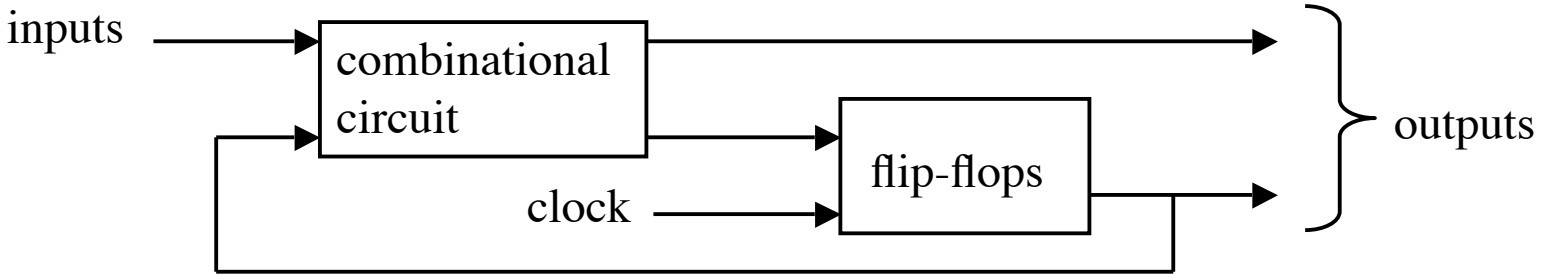
- Consider a counting system which receives input pulses and provides an output representing the total number of pulses received so far.




- When a new pulse arrives, the output must have '1' added to it. This requires knowledge of the preceding output, i.e. the system is sequential - it depends on input **and** previous outputs.

# SLS

- The basic building block of the sequential logic system is the **flip-flop**. A typical scheme of an SLS is



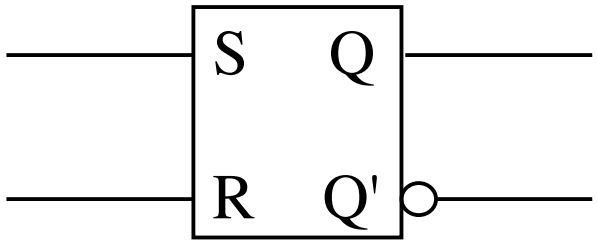

- The clock is included to control the flip flops - it ensures that the outputs change at certain instants of time, which may be required to synchronise the SLS with other parts of the whole system.

# Flip-Flops and Latches

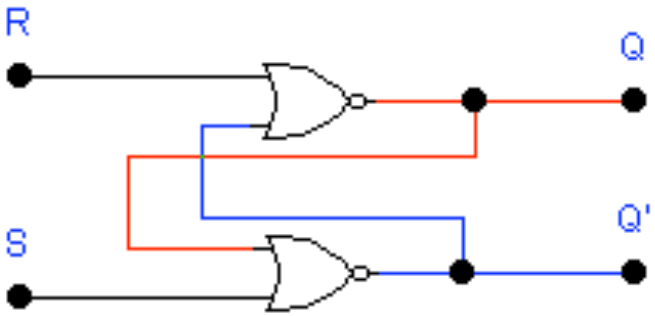
- Latches and flip-flops are both bistable devices (two stable states).
  - Latch - level triggered
  - Flip flop - edge triggered
- I.e. to do with how state changes are triggered.
  - But the terminology vague and mixed!

- There are a number of different kinds of flip flops and latches

# Set/Reset (S - R) latch




- Broadly, S = 1 causes Q = 1, R = 1 causes Q = 0.



# Latch Operation

- The latch operation is described in terms of its characteristic table (like a truth table but with state)

| S | R | Q | |
|---|---|---|---|
| 0 | 0 | Q | no change |
| 0 | 1 | 0 | clear/reset |
| 1 | 0 | 1 | set to 1 |
| 1 | 1 | ? | undefined |

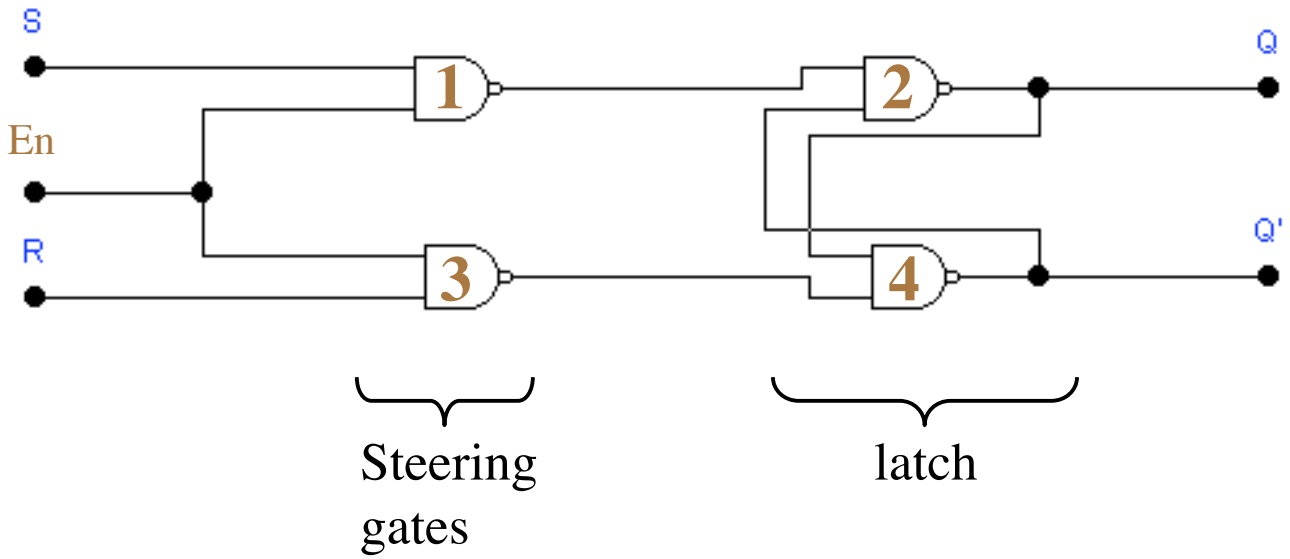
- For active high latch

# Latch application

- SR latch can also be used to combat *switch debouncing*.
  - When a mechanical switch is opened or closed the contacts may bounce and a dirty transition results (instead of a nice clean one).
- results in clean transition - small bounces on S make no difference to Q after initial 1.

# S-R Gated Latch

- Controlled by an enable.




- If LOW enable signal then Q does not change, regardless of values of S and R. Only when enable changes to HIGH can the values of S and R be used to affect the output Q.

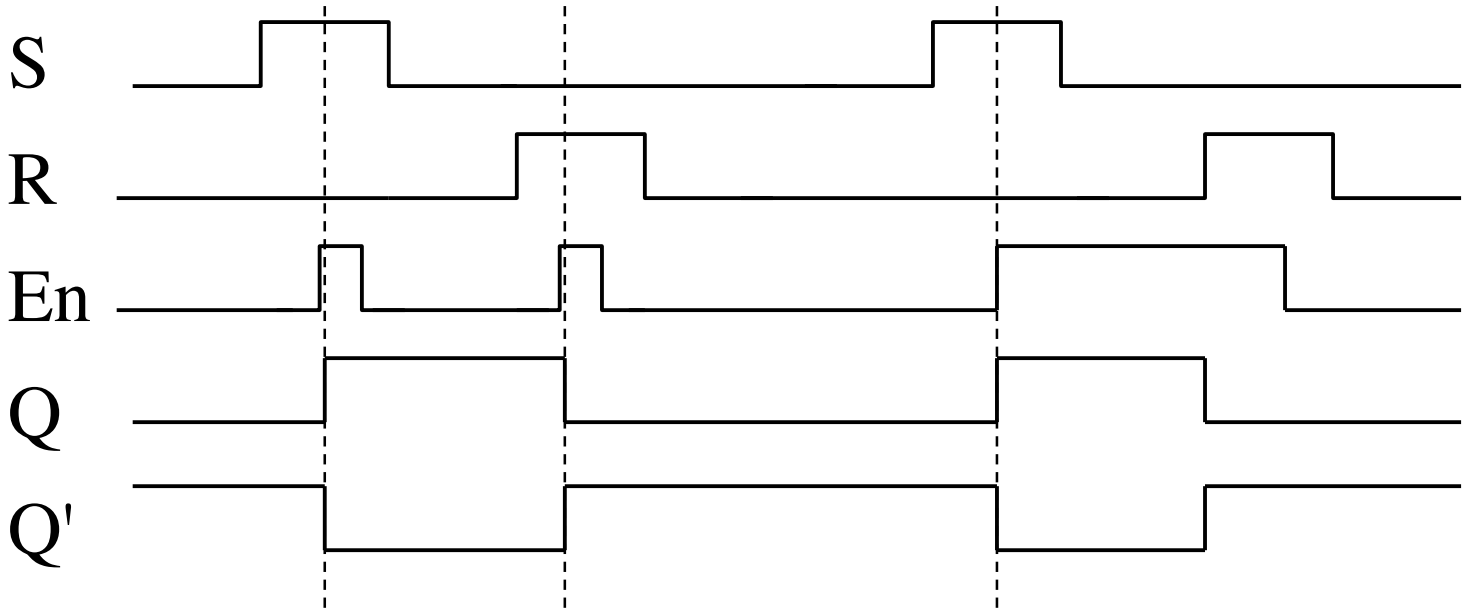
# S-R Gated Latch

- The latch operation is described in terms of its truth table
- When En = 1

| S | R | Q | |
|---|---|---|---|
| 0 | 0 | Q | no change |
| 0 | 1 | 0 | clear/reset |
| 1 | 0 | 1 | set to 1 |
| 1 | 1 | ? | undefined |

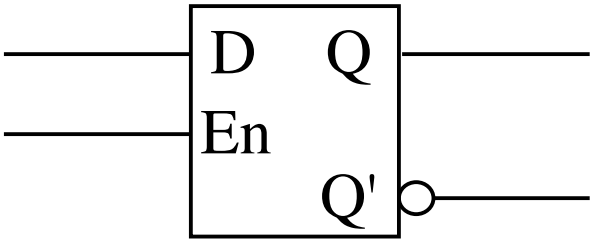
# S-R Gated Latch

- The operation can also be seen by looking at the timing (waveform) diagram.



# D type latch

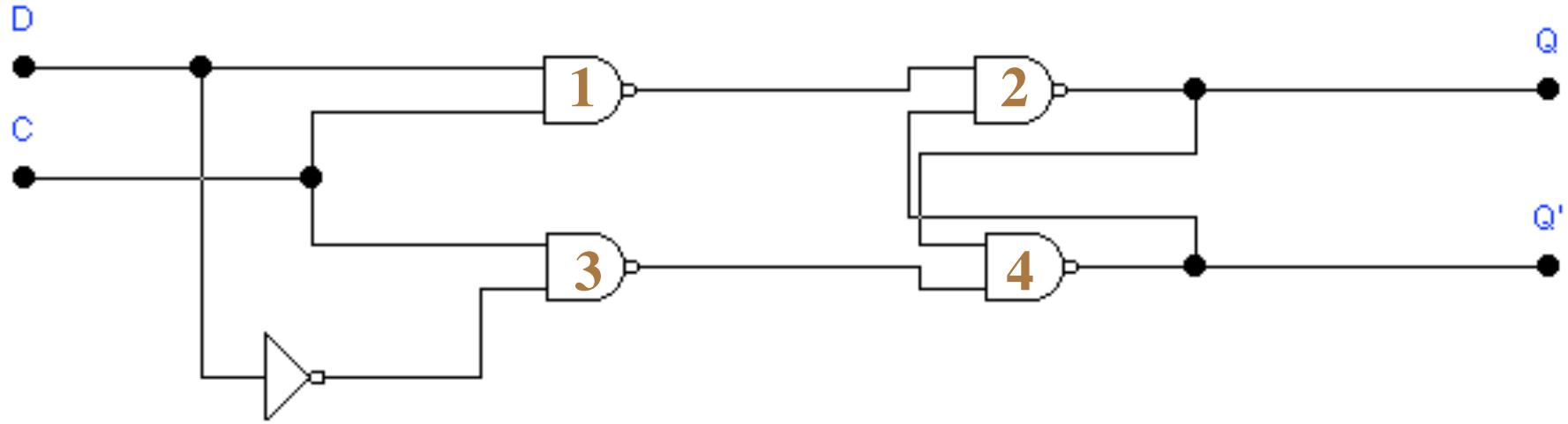
- Removes problem of undefined output. One input D and an enable, changes in D are reflected at the output when *enable* = 1. (i.e. latch is transparent when enable is high)




| En | $D_{(t)}$ | $Q_{(t+1)}$ | $Q'_{(t+1)}$ |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |
| 0 | X | no change | |

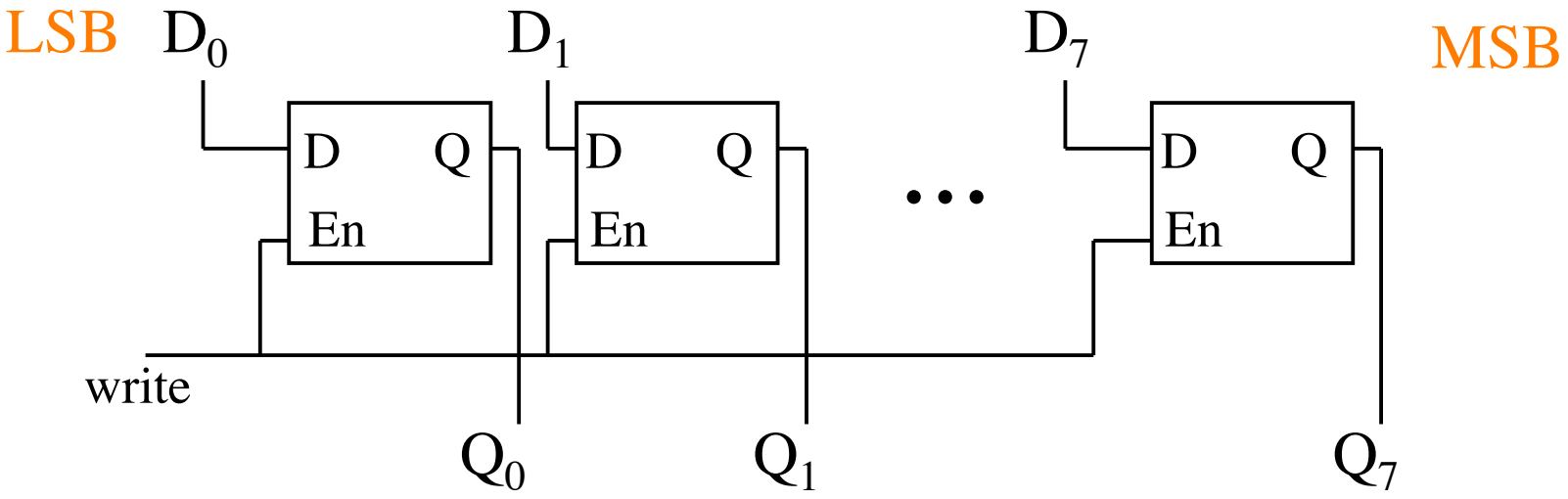
- In the characteristic table (t) is time now, and (t+1) is the next time step.
- D stands for *Delay* (output is delayed while enable = 0)

# D type latch




D = S and D' = R therefore never have S = R

# Example: a register




- Data $D_0$ $D_0$ …. $D_7$ is loaded in parallel into the D latches on a *write* signal.
- The register may also include a special reset to clear all simultaneously.

# Power Up

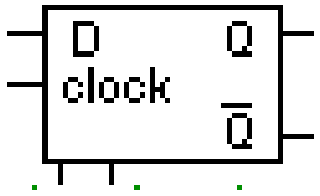
- A system may contain a number of flip flops, counters, shift registers, etc.
- What is their initial state immediately after power is applied?
- We need a known (and reproducible) state at power up, e.g. flip-flops reset, counters zeroed, shift registers clears and so on.
- Use a simple switch (interactive input) to reset all elements simultaneously

# Edge Detection

- Problem: when enable = 1 the output varies depending on the input. We might prefer that on the transition 0-1 the input is sampled once and the output fixed, so changes of input are not reflected until the next 0-1 transition.
- Difference between a gated latch and a flip-flop is the use of edge triggering.
  - Flip-flop changes on 0-1 transition (or 1-0 transition)
  - Latch changes on HIGH level (or LOW level)

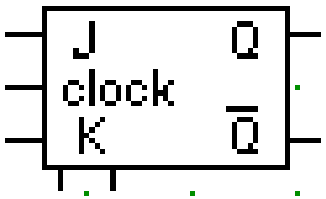
# Edge triggered flip flops

- D flip flop




- Rising edge triggered ie 0-1 transition. Also clear and preset inputs.
- The clock is dynamic input

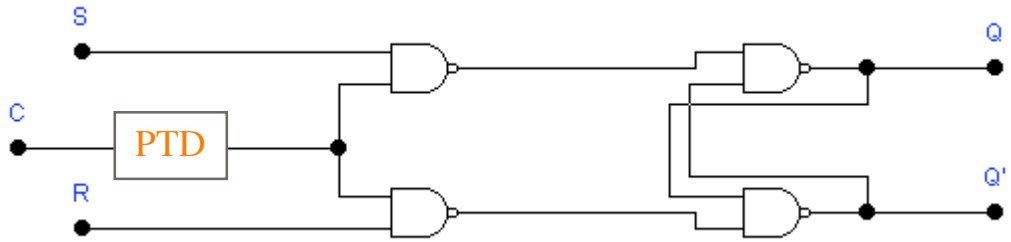
- JK flip flop




- Falling edge triggered (1-0 transition). Also clear and preset inputs.

# How does edge triggering work?

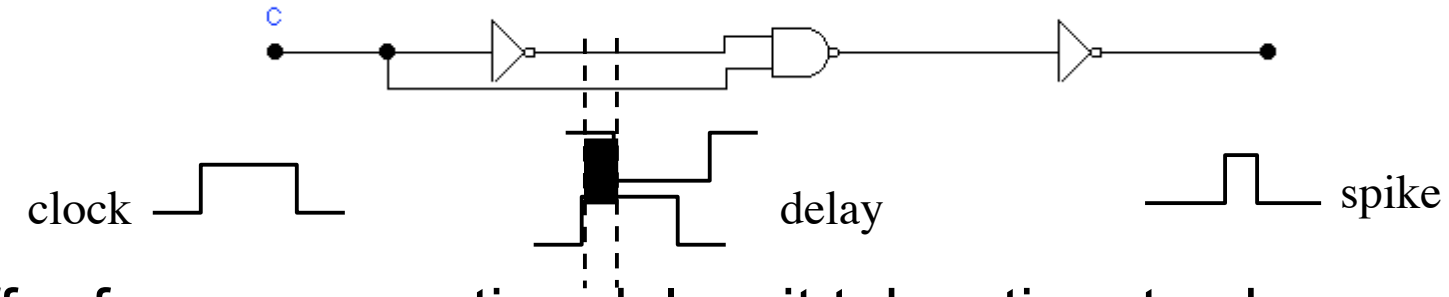
- On the SR flip flop, a pulse transition detector (PTD) is added to the clock line.




- The PTD produces a short spike (rather than a square pulse) from the clock.
- This means that only the data on S and R during the spike will be sampled.

# PTD

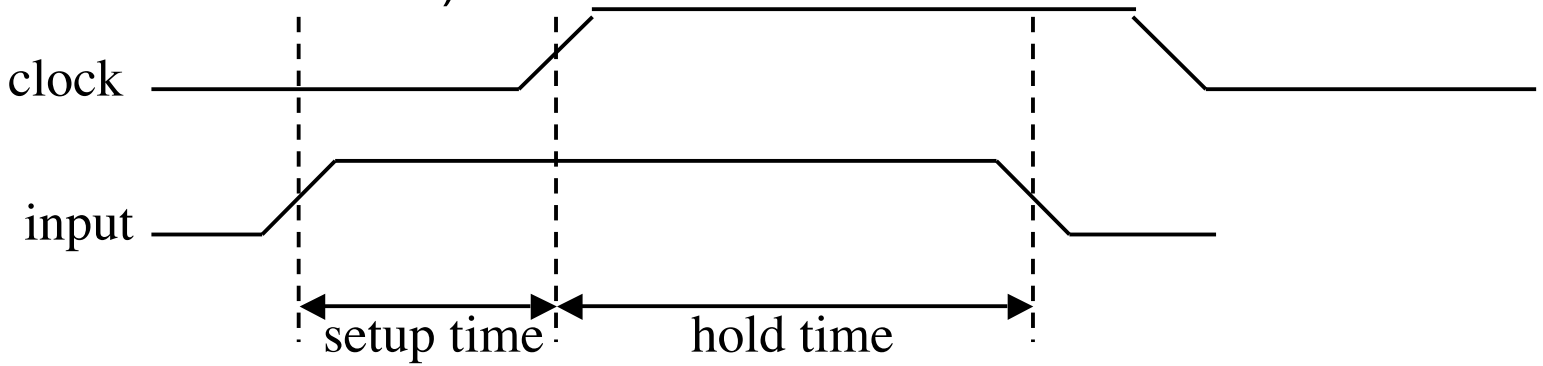
- The Pulse Transition Detector is implemented by the following gates.




- Gates suffer from propagation delay; it takes time to change the output to reflect new inputs.
- Therefore, both C and C' will be high for a few nanoseconds.
- Sometimes propagation delay can be useful (as above) but sometimes it just leads to (momentary) wrong answers.
- In clocked synchronous circuits all components are guaranteed to change at the same instant.

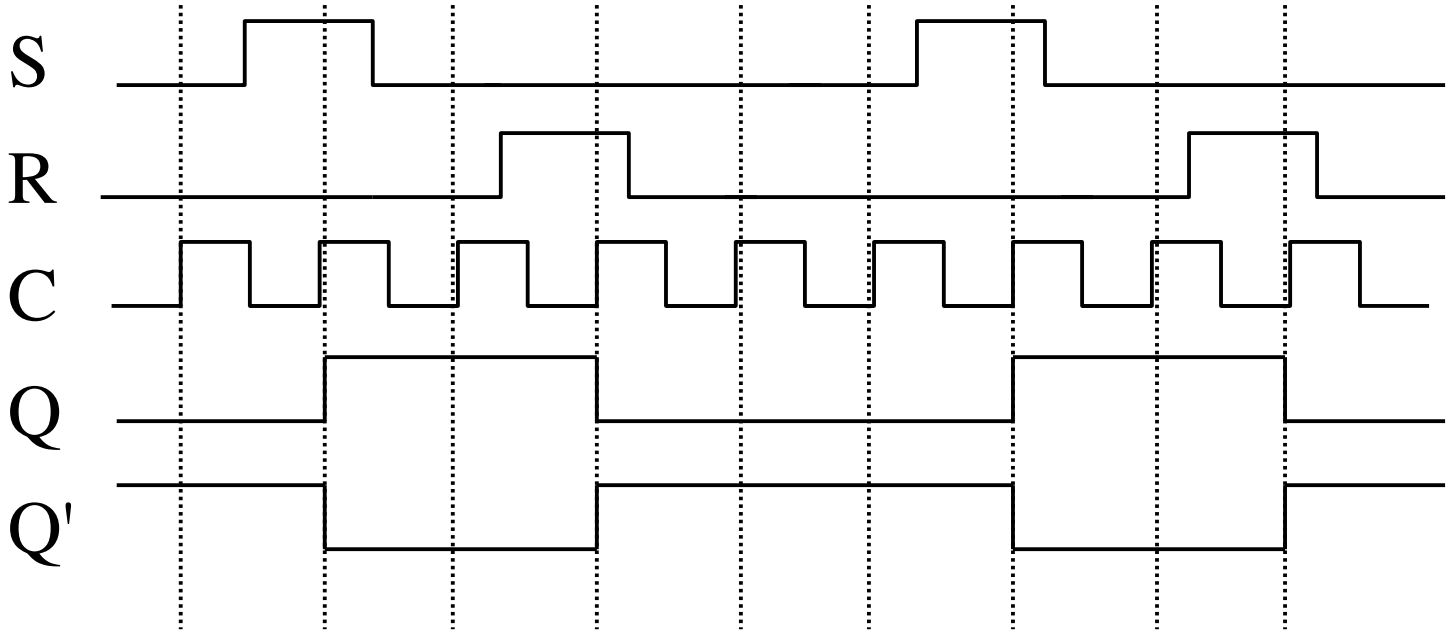
# Presentation of Data

- Time constraints on presentation of data (for it to be reliably clocked into the flip flop).
- The **set-up time** is the minimum time between the leading edge of an input data pulse and the triggering clock pulse.
- The **hold time** is the time between the clock transition changing the output and the end of the input pulse. After the hold time the inputs may change with no effect on the output.
- The smaller these times the better. (e.g. setup times vary from 2 to 20 ns, hold times from 0.5 to 3 ns)



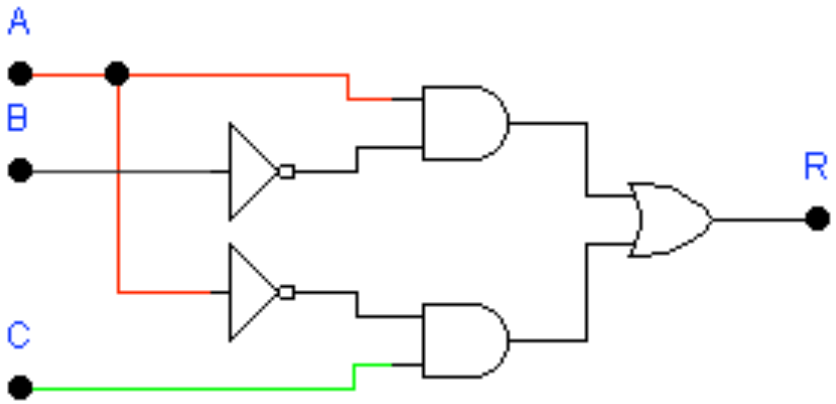
# S-R Flip Flop Timing Diagram

- Note that the output Q only changes at the 0 - 1 clock transition, not when S or R first changes.



# Propagation Delays

- Consider the following circuit (R = A.B' + A'.C)




- Let A = 1, B = 0, C = 1. What happens when A changes to 0?
- Logically the value of R should not change, but physically it does.

# Static Hazards

- Let propagation delay of all gate be $\delta$, and A changed at time $t_0$

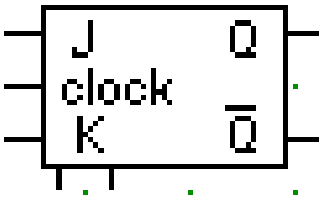
| time | A | B | C | B' | A' | A.B' | A'.C | R |
|---|---|---|---|---|---|---|---|---|
| $< t_0$ | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| $t_0$ | 0 | 0 | 1 | 1 | 0* | 1* | 0* | 1* |
| $t_0 + \delta$ | 0 | 0 | 1 | 1 | 1 | 0 | 0* | 1* |
| $t_0 + 2\delta$ | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0* |
| $t_0 + 3\delta$ | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |

* means old values used - change has not yet propagated

- These are called **static hazards** (when we have A and A')
- A hazard is likely if 2 adjacent cells in a Karnaugh map are not looped together (may give larger terms, but eliminates hazard)

# J - K Flip Flops

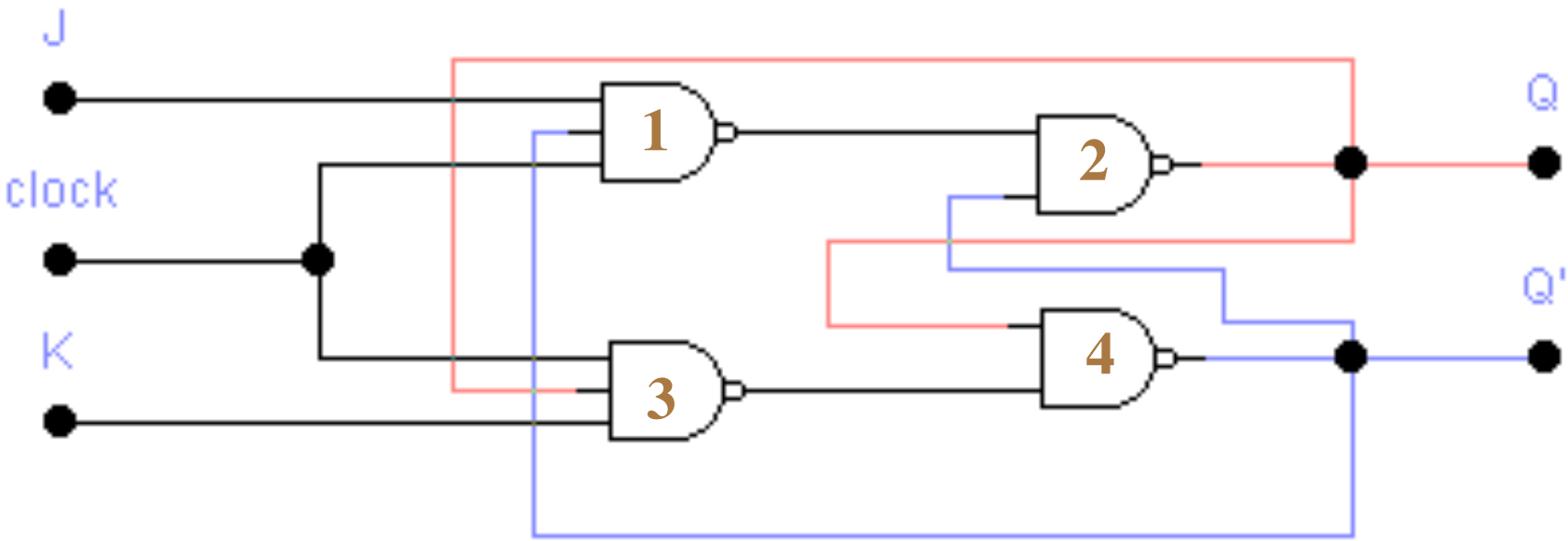
- Another attempt to solve the problem of undefined output on the SR latch (when S = R = 1)



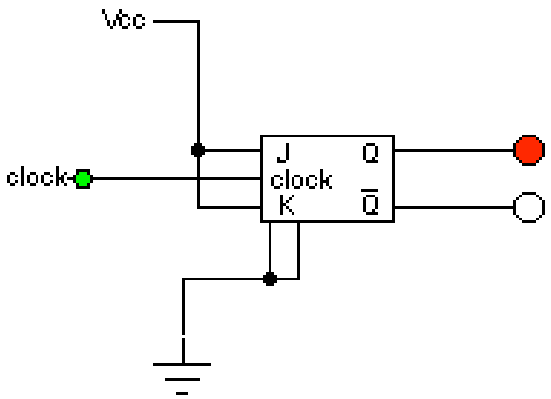

| J | K | $Q_{(t+1)}$ | |
|---|---|---|---|
| 0 | 0 | $Q_{(t)}$ | no change |
| 0 | 1 | 0 | clear |
| 1 | 0 | 1 | set |
| 1 | 1 | $Q_{(t)}'$ | toggle |

- like SR, but Q fed back to K gate, Q' back to J gate
  - If J = K = 0 enabling does nothing
  - If J ≠ K enabling causes $Q_{(t+1)} = J$
  - If J = K = 1 enabling causes **toggle**

# J-K Flip Flops (inside)



# Example: Divide by 2

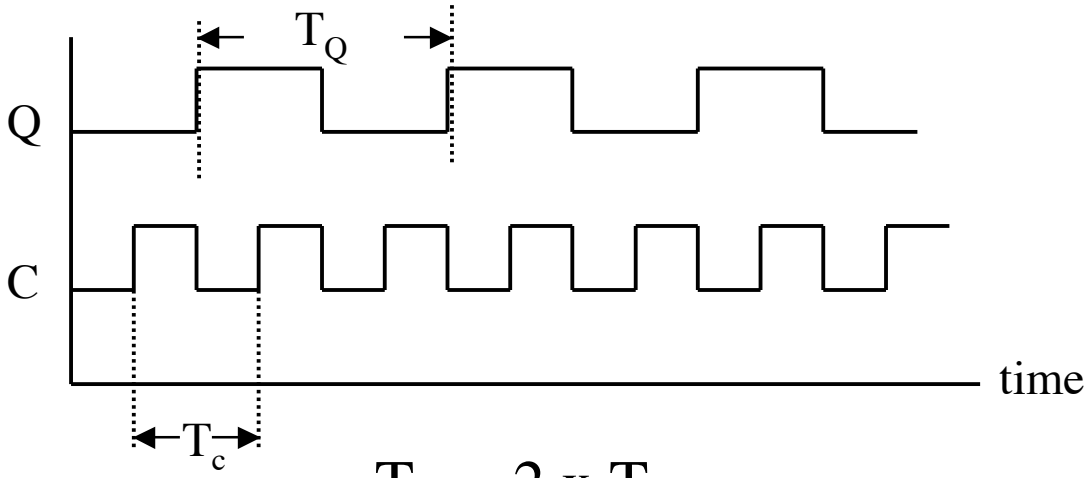



J = K = 1 therefore the circuit toggles on the falling clock input.

The frequency of the output waveform is 1/2 that of the input (clock) waveform.

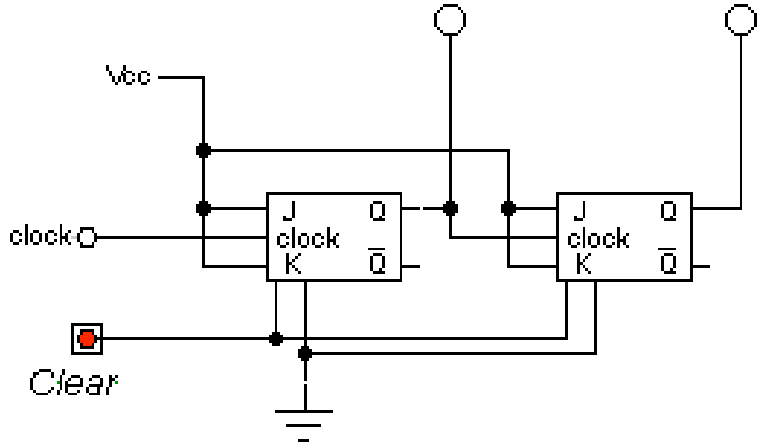
| Q | C |
|---|---|
| 0 | 0 |
| 0 | 1 |
| 1 | 0 |
| 1 | 1 |

repeat

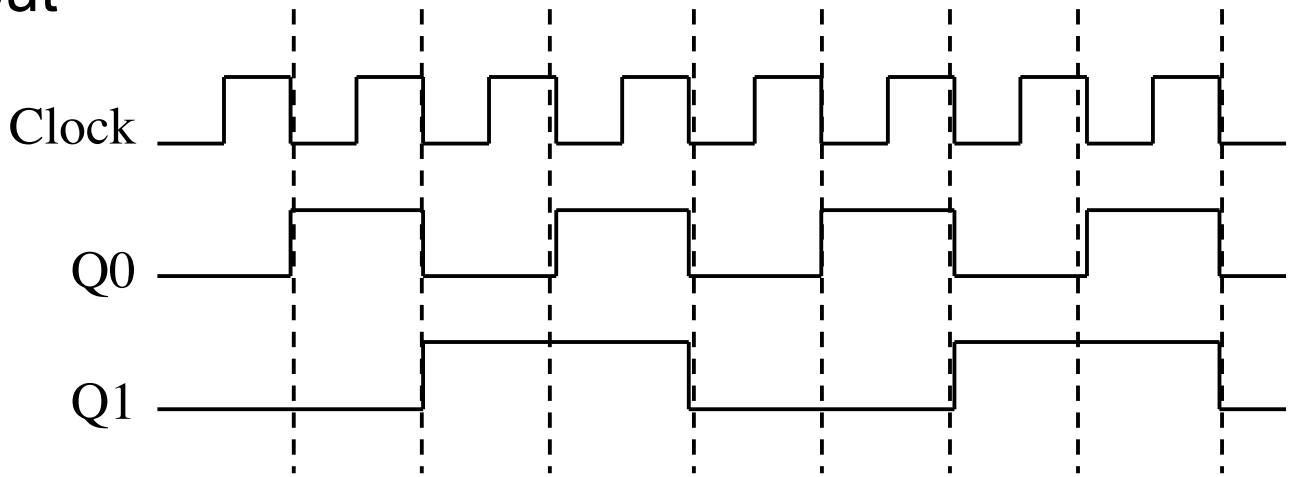



$$T_Q = 2 \times T_c$$

# Binary Counters (Ripple counters)




- Output



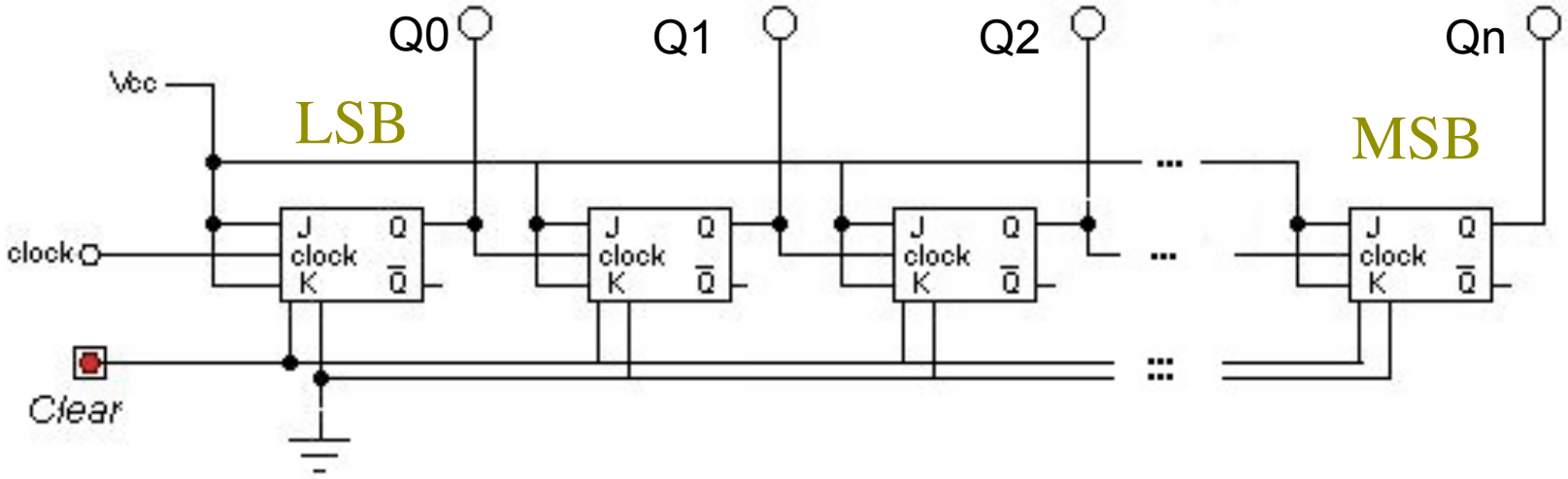
# Binary Counters (Ripple counters)

- Represent the output of the circuit as a table:

| Clock pulse number | Q1 | Q0 | |
|---|---|---|---|
| 0 | 0 | 0 | counts 0, 1, 2, 3 |
| 1 | 0 | 1 | |
| 2 | 1 | 0 | |
| 3 | 1 | 1 | |
| 4 | 0 | 0 | repeats |
| 5 | 0 | 1 | |
| 6 | 1 | 0 | |
| 7 | 1 | 1 | |

- The system counts 4 clock pulses and then repeats.
- Counters for any number ($2^n$) can be created by adding more J-K flip flops. The Q output is connected to the clock input for the next stage.

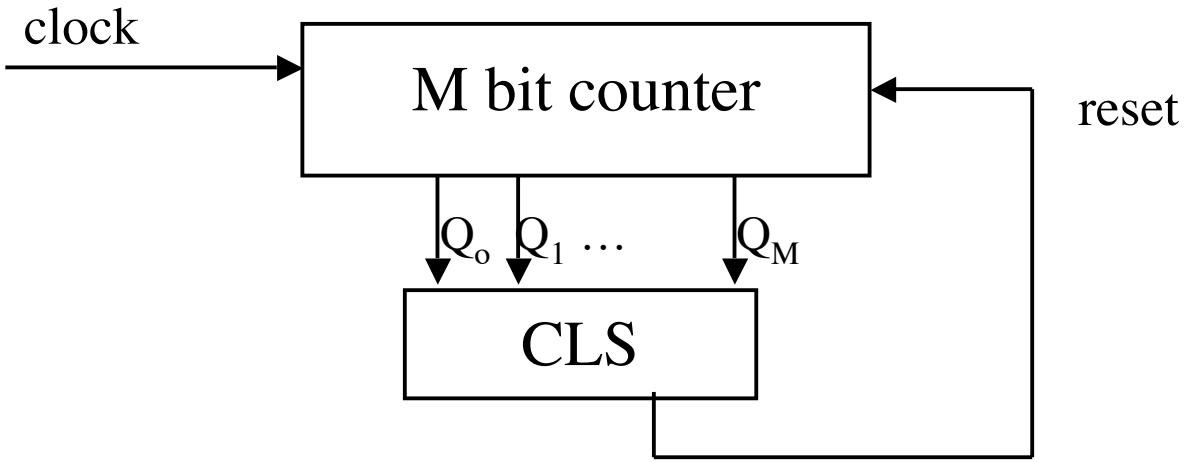
# N-bit Binary counter




- The output sequence repeats every $2^n$ clock pulses
- The effect of the clock “ripples” through the flip-flops
- Propagation delay means each ff changes slightly later.
- -ve edge triggered flip flops are used
- +ve edge triggered flip flops cause the count to proceed in reverse (ie 3, 2, 1, 0…..)

# Divide by N Counter

- The binary counter above only counts up to $2^m$ where M is the number of flip flops. How can we make a counter for N, where N is not a power of 2?
- The Divide by N counter **detects** when the *upper limit* has been reached and **resets** the counter to zero.
- A simple combinational logic system is used.



# Example: A decade counter

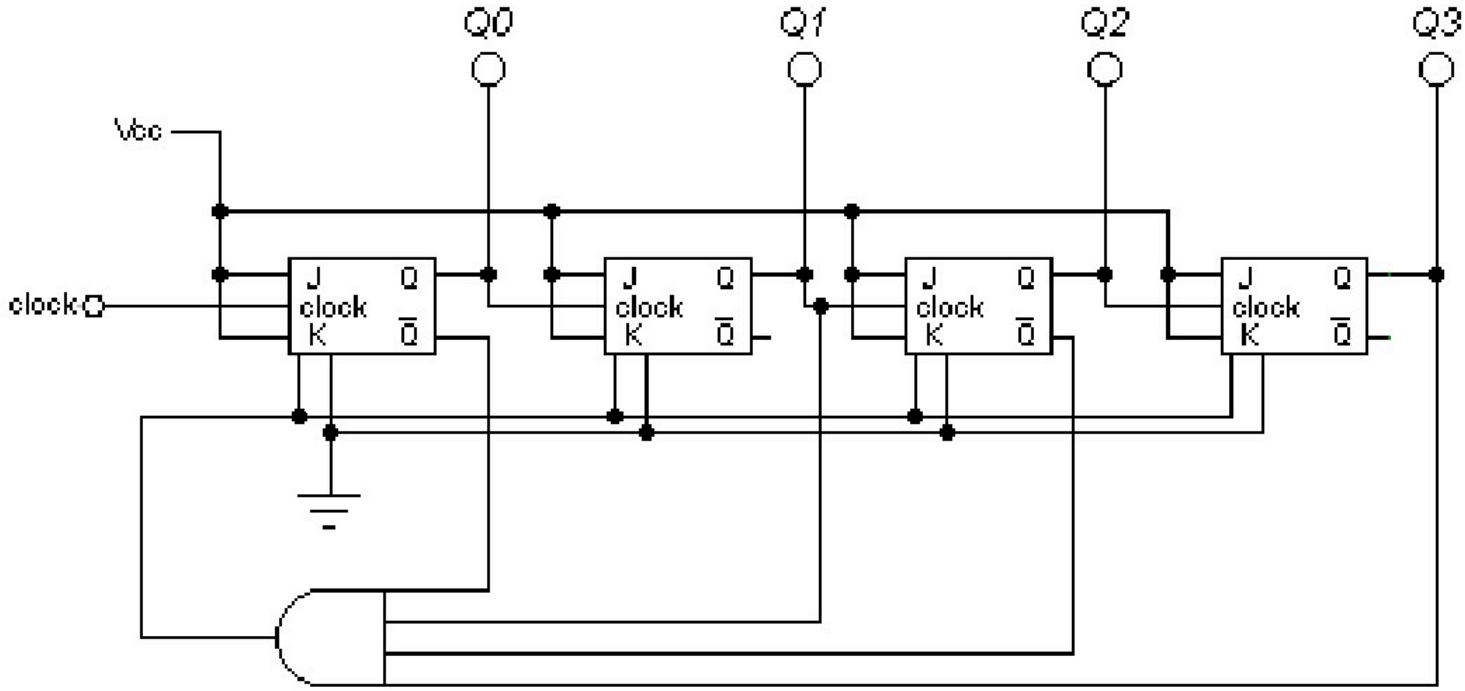
- Decade counter counts from 0 to 9 and repeats. Count sequence is

| pulse number | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |
| 10 | 1 | 0 | 1 | 0 |

reset

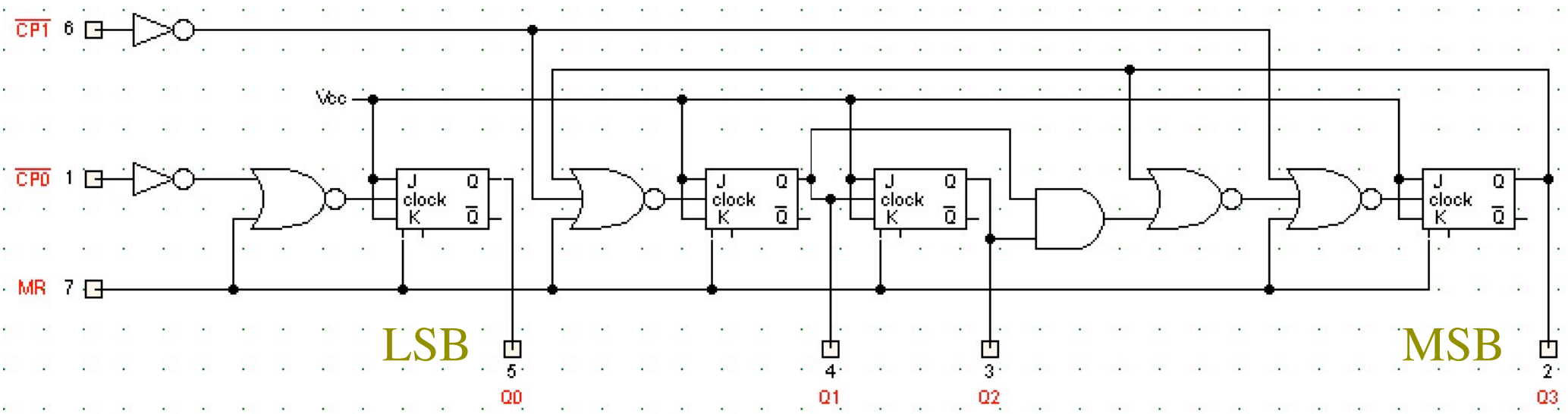
- How many flip flops?
  - $2^3 = 8$, $2^4 = 16$, therefore 4 flip flops required.
- Reset occurs when
  - $Q_0 = 0$, $Q_1 = 1$, $Q_2 = 0$, $Q_3 = 1$
- Assume clr is **active high** (ie 1 on clr means reset, 0 means do nothing) and -ve edge triggered (1-0 transition)
- Clr = $(Q_0' \cdot Q_1 \cdot Q_2' \cdot Q_3)$

# Decade Counter Implementation




- Problems:
  - Redundancy - can count to 15!
  - Not practical - state 10 exists - there's a momentary glitch
  - Ripple counters slow(er)

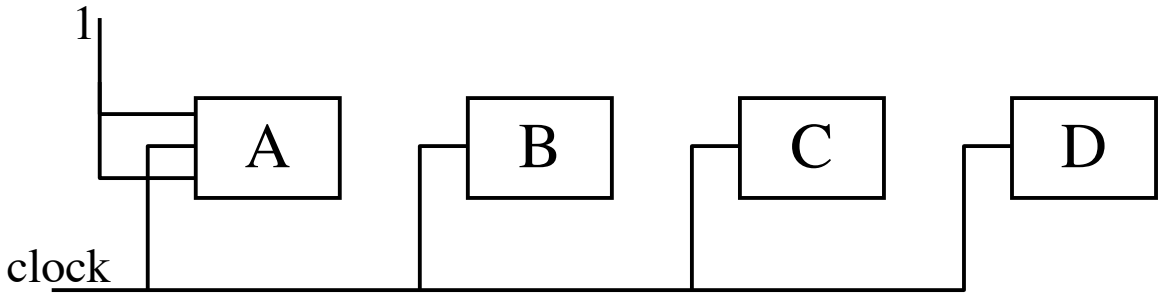
# Decade Counter Implementation




- Problems:
  - Redundancy - can count to 15!
  - Not practical - state 10 exists - there's a momentary glitch
  - Ripple counters slow(er)

# Synchronous Counter

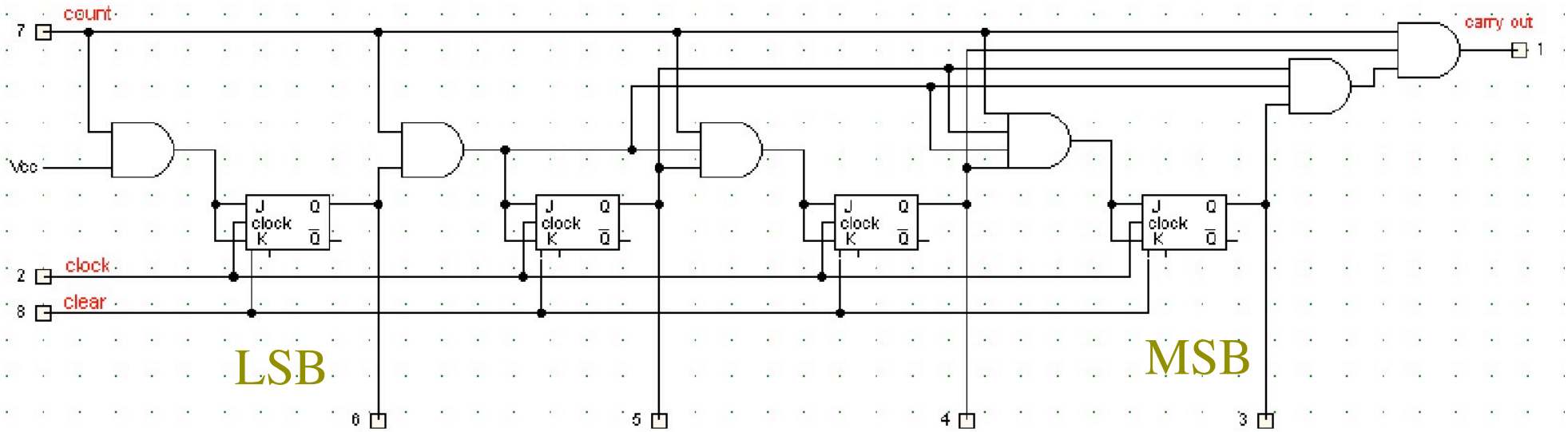
- Consider clocking all flip flops at the same time.




- How do we make the flip-flops toggle at the right time?
  - Flip flop A toggles on every clock pulse
  - Flip flop B toggles only when A is HIGH and there's a clock pulse
  - Flip flop C toggles only when A and B are HIGH and there's a clock pulse
  - Flip flop D toggles only when A and B and C are HIGH and there's a clock pulse

# Synchronous Counter: Implementation

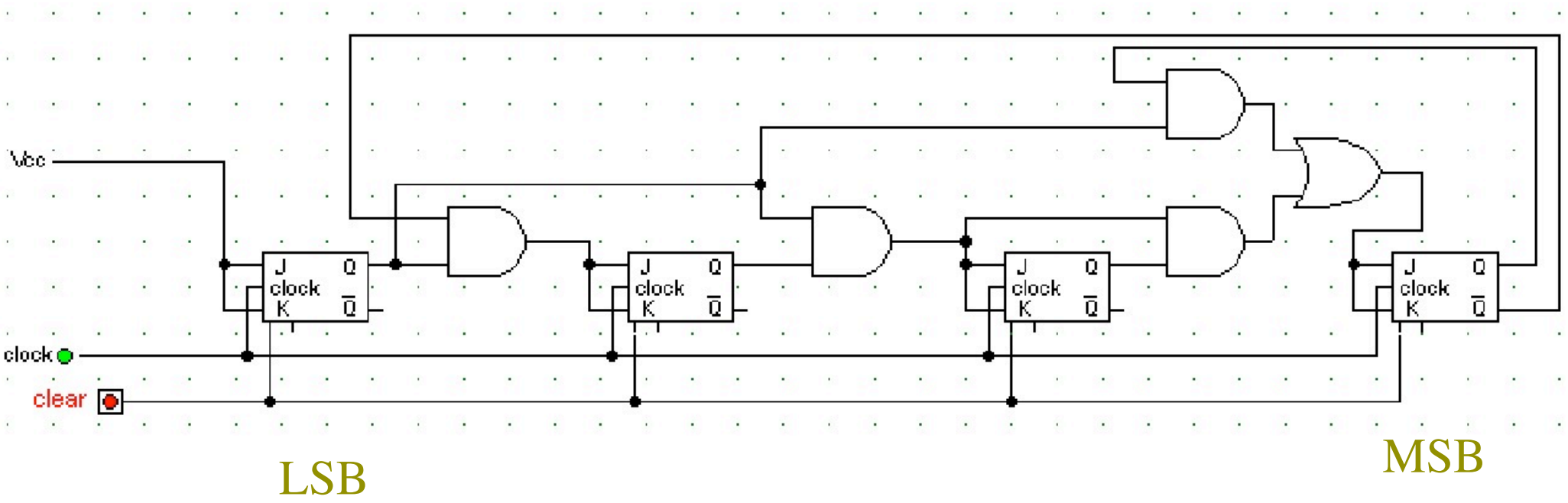
- The JK inputs come from previous Q inputs “anded” together.




- How can this be altered to make a decade counter?

# Synchronous Decade Counter

- More complicated arrangement to trigger each bit



# Synchronous Decade Counter

- Recall the table

| pulse number | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |
| 10 | 1 | 0 | 1 | 0 |

reset

$Q_1$ triggers only on 1, 3, 5, 7 (but not 9)
i.e. when Q1 is 1 but Q3 is 0

$Q_2$ triggers only on 3 and 7
i.e. when both Q1 and Q0 are 1

$Q_3$ triggers only on 7 and 9