Asynchronous (ripple) counters

Basic idea of counters

Asynchronous (ripple) counter using *JK* flip-flops

counting through binary power (e.g. 2^4 = 16 states, 0–15)

counting through an arbitrary range (e.g. 0–9, decade counter)

Effect of propagation delay

causes clock to ‘ripple’ through counter

causes the system to pass through wrong states (briefly)

hence ripple counters are used only for undemanding applications where the timing is not critical

Other types of asynchronous counters can also be made

down counters, for example

Counting in binary

Counters are one of the simplest digital systems. The binary counting sequence is shown on the right.

Each flip-flop changes periodically from 0 to 1 and back again — in other words, they **toggle**.

Each column changes at half the speed of the column on its right, so counting is closely related to dividing by 2.

C	B	A
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1
0	0	0

J	K	Q_n	Q_{n+1}	description
0	0	0	0	hold
		1	1	
0	1	X	0	clear
1	0	X	1	set
1	**1**	**0**	**1**	**toggle**
		1	**0**	

Building block of a counter

Look in more detail. **When does each column change?**

B toggles whenever **A** goes from 1 to 0

C toggles whenever **B** goes from 1 to 0

General rule: each bit toggles when the less significant bit changes from a 1 to a 0.

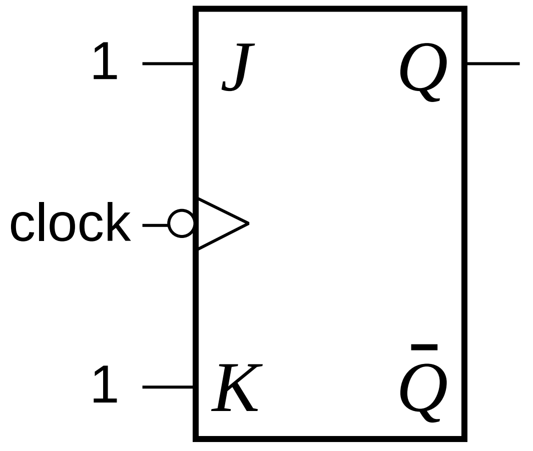
Each flip-flop is triggered by the less significant one:
B is triggered by A, C is triggered by B, and so on.

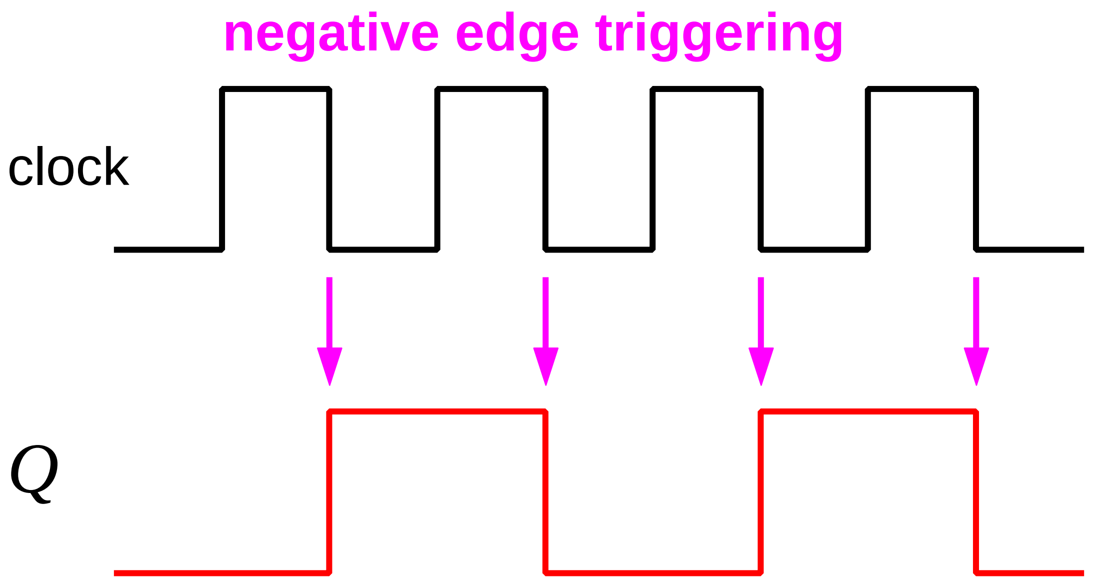
Triggering is when the less significant bit goes from 1 to 0:
negative edge triggering.

The toggling action needs a *T* (toggle) flip-flop with *T* = 1,
or a *JK* with *J* = *K* = 1.

C		B		A
0		**0**		**0**
0		0	↓	**1**
0		**1**		**0**
0	↓	1	↓	**1**
1		**0**		**0**
1		0	↓	**1**
1		**1**		**0**
1	↓	1	↓	**1**
0		**0**		**0**

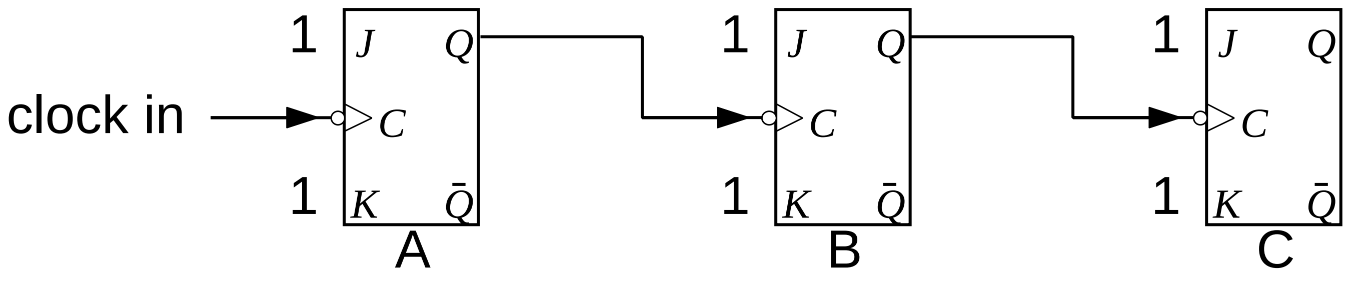
Action of a clocked *JK* or toggle flip-flop





- Flip-flop has $J = K = 1$ so that it **toggles** on each clock transition
- Flip-flop triggers on **negative edge** (clock goes from 1 to 0)
- This **halves the frequency** as required: $f_Q = f_{\text{clock}}/2$
- For more than one bit, use Q output as clock input for next stage

3 bit counter with *JK* flip-flops



Each stage divides the frequency by 2

Called a **ripple counter** because the clock 'ripples' from stage to stage

- We shall later look at **synchronous** counters, where the same clock is applied directly to all flip-flops

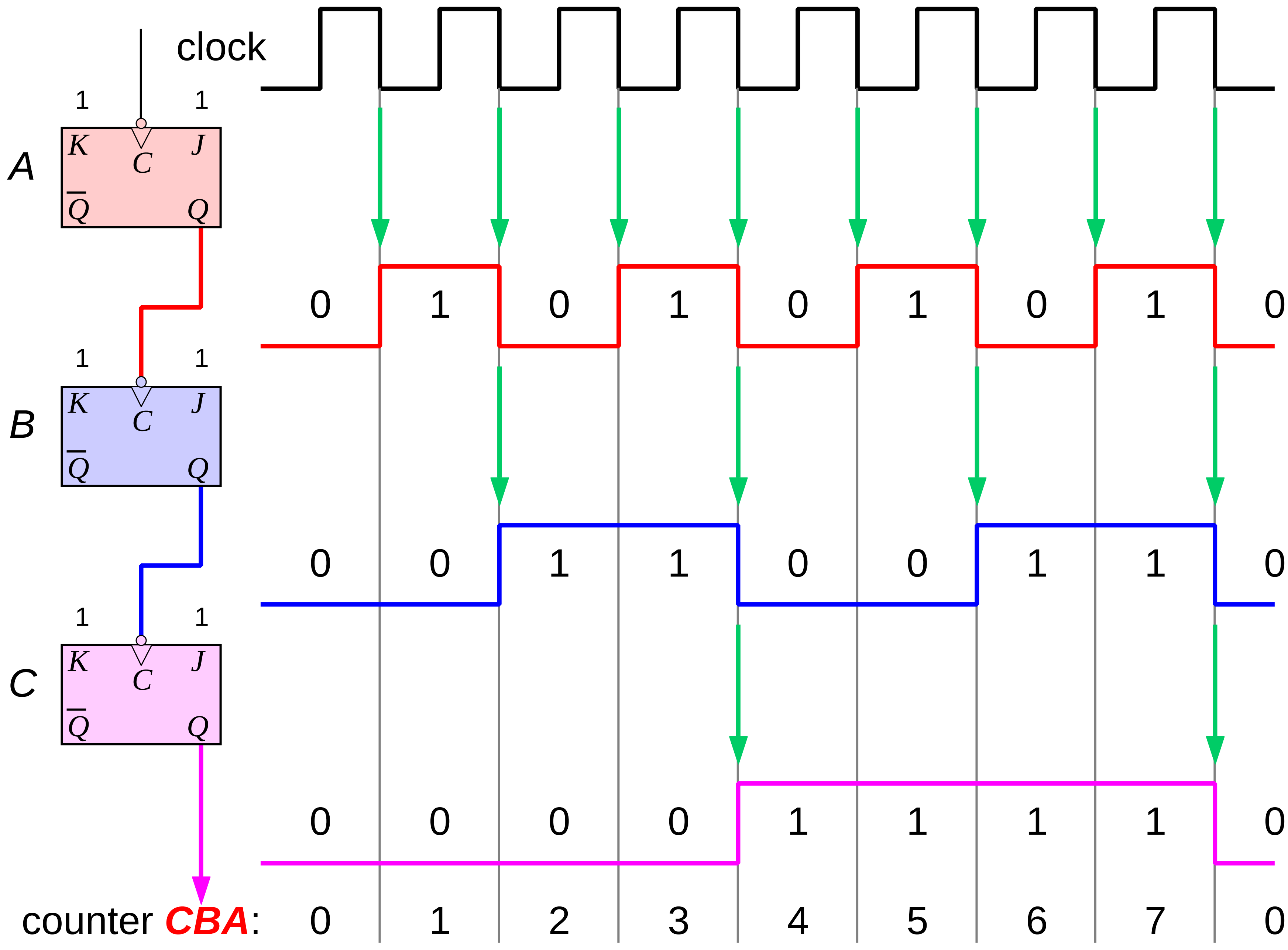
Waveforms sketched on next page

- Propagation delay is not shown but will be important!

Note that flip-flops are lettered A, B, C from left to right

- A is least significant bit (units), B is twos, C is fours
- but numbers are written in the sequence *CBA*!

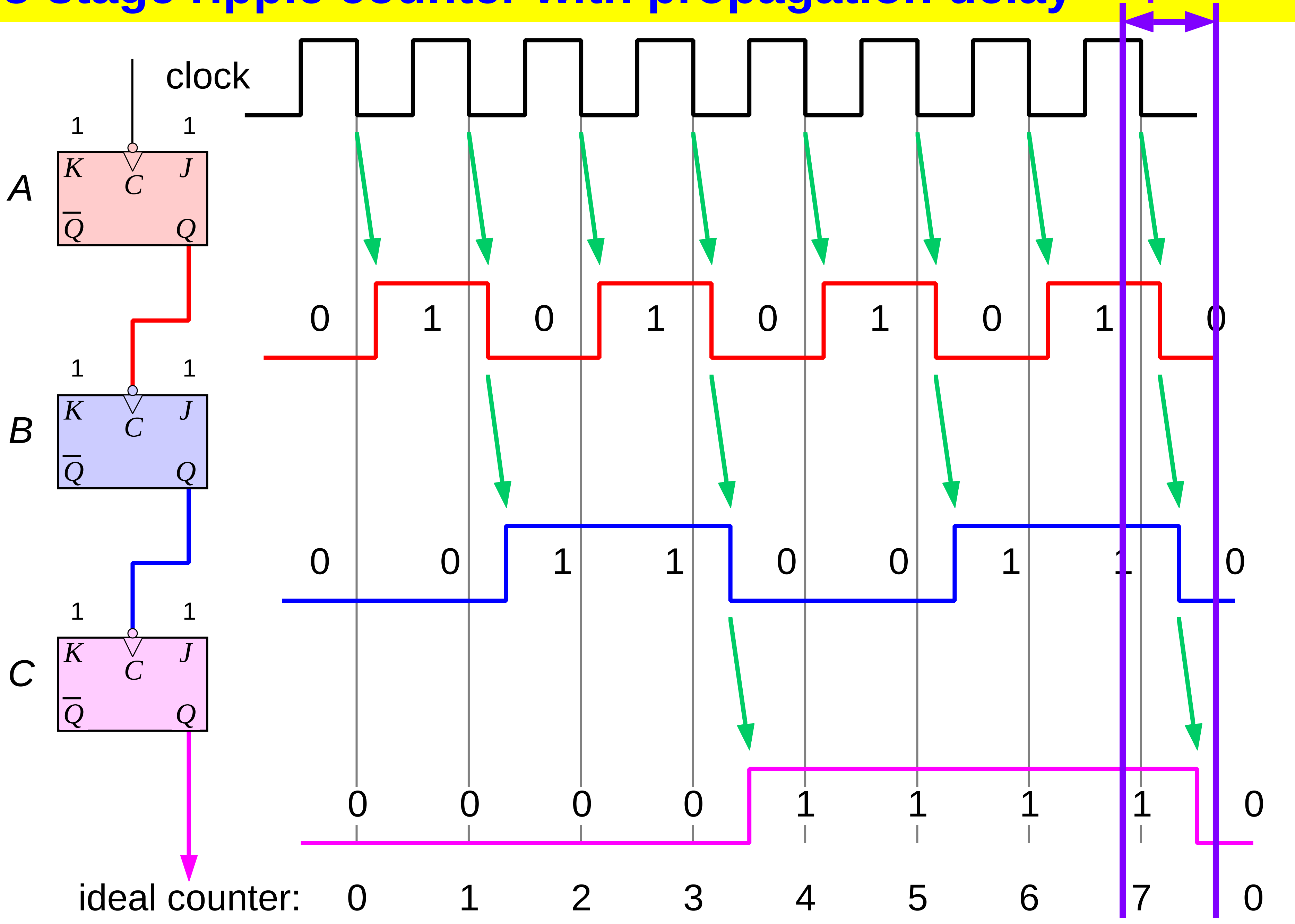
3-stage ripple counter (idealized)



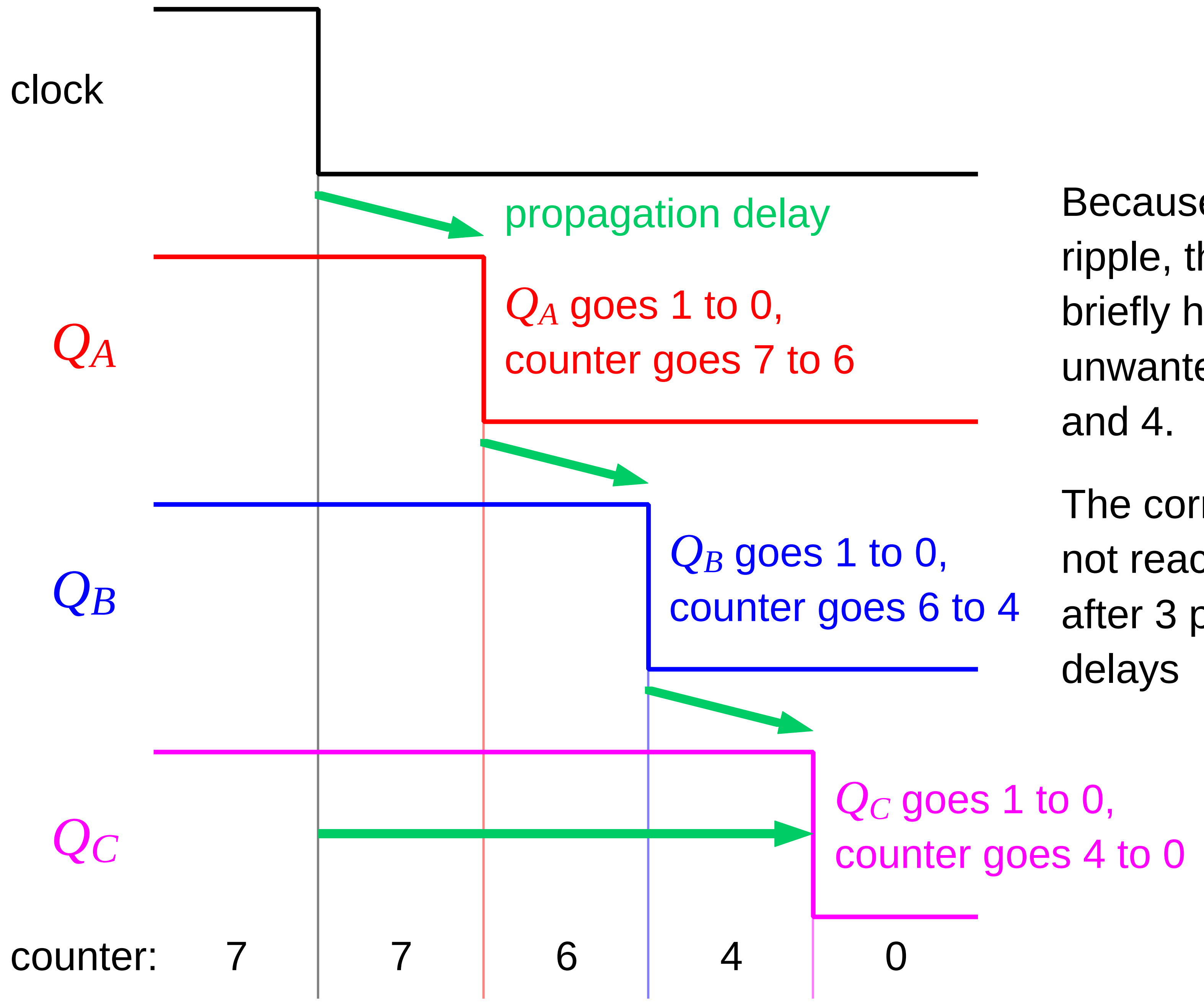
Ripple counters

- This is a basic ripple counter; more complicated systems can be built
- It is important that the flip-flops are negative edge triggered! See the tutorial sheet
- The circuit is usually drawn with the least significant bit (LSB, flip-flop *A*) on the left, but numbers are written in the opposite order (…*CBA*)
- It **rolls over** from 7 back to 0 when it exhausts its range of $2^3 = 8$ states
- It is called a **ripple counter** because the clock ‘ripples’ through the system, from flip-flop to flip-flop
- The clock gets later by the **propagation delay** in each flip-flop, so the flip-flops for more significant bits change later than those for less significant bits — see next diagram
- The counter contains incorrect values (glitches) while the clock is rippling
    - This effect is biggest when the most significant bit (MSB) changes
    - **Enlarge this part of the timing diagram**

3-stage ripple counter with propagation delay



Detail of ripple effect in transition from 7 to 0



Because of the ripple, the counter briefly holds the unwanted values 6 and 4.

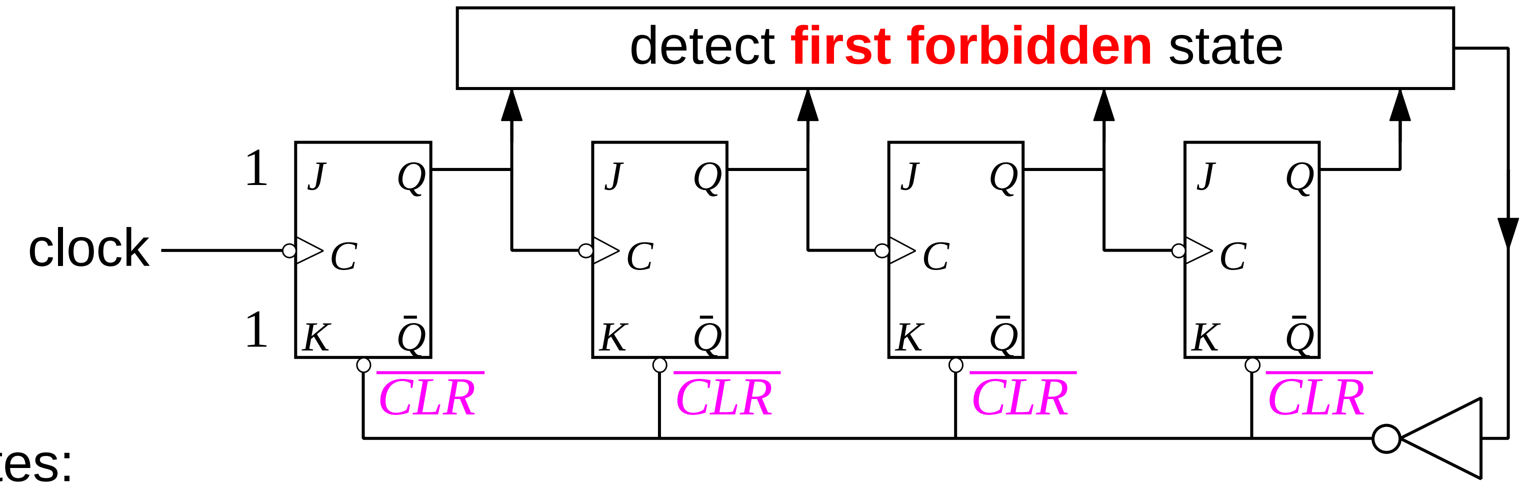
The correct value is not reached until after 3 propagation delays

Decade counter

How do we make a counter whose range is not a power of 2, such as a **decade counter** (10 states, 0–9)?

Solution: Use a binary counter (4 flip-flops, 16 states, 0–15 naturally) but clear it (reset to 0) as soon as it enters the **first forbidden state** (10).

Use *CLR* (clear) control input of flip-flops.



Notes:

- *CLR* must act **instantly**, not wait for next clock transition — needs **asynchronous/direct/jam clear**
- check whether *CLR* is **active low** (as above) or high (less common)

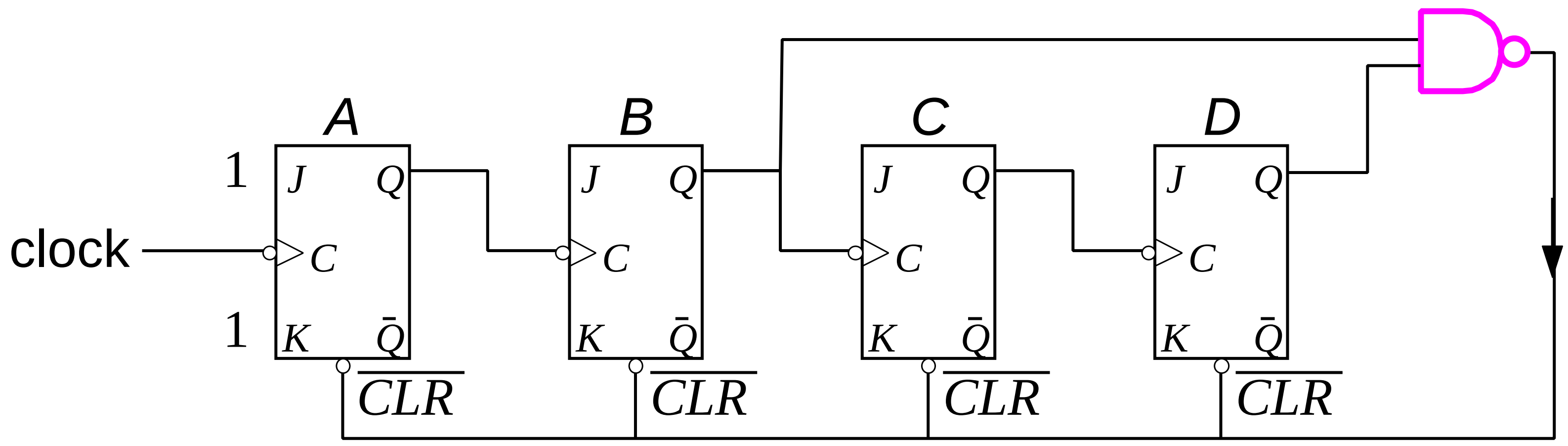
Circuit for decade counter

The reset circuit must detect $DCBA$ = decimal 10 = binary 1010. Therefore

clear = $D\,\bar{C}\,B\,\bar{A}$ (needs 4-input AND)

In fact this is overkill and clear = $D \cdot B$ is sufficient — why?

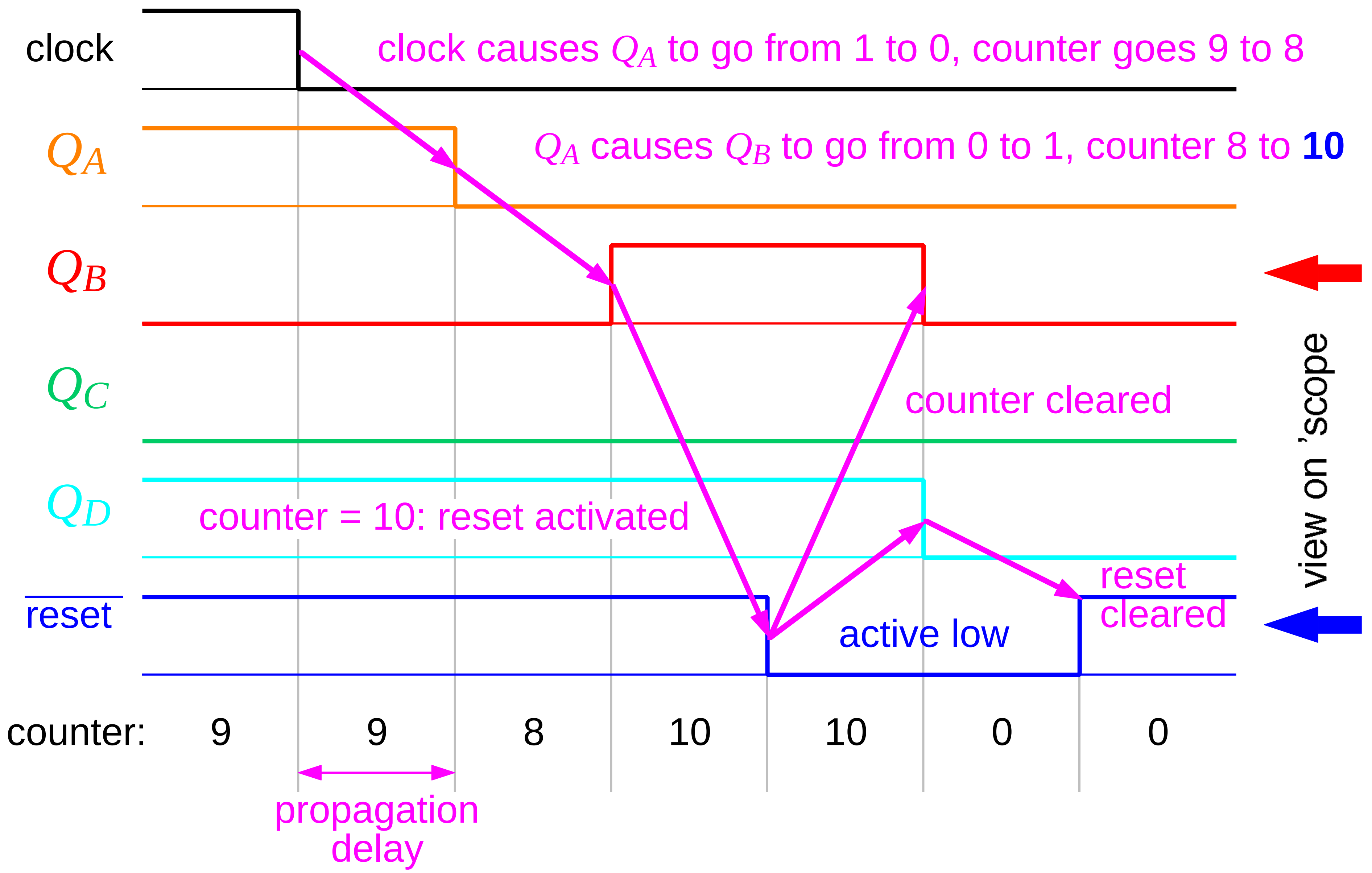
The circuit below has **NAND** rather than AND because the clear input is **active low**: It should be kept high during normal counting and low for clear.



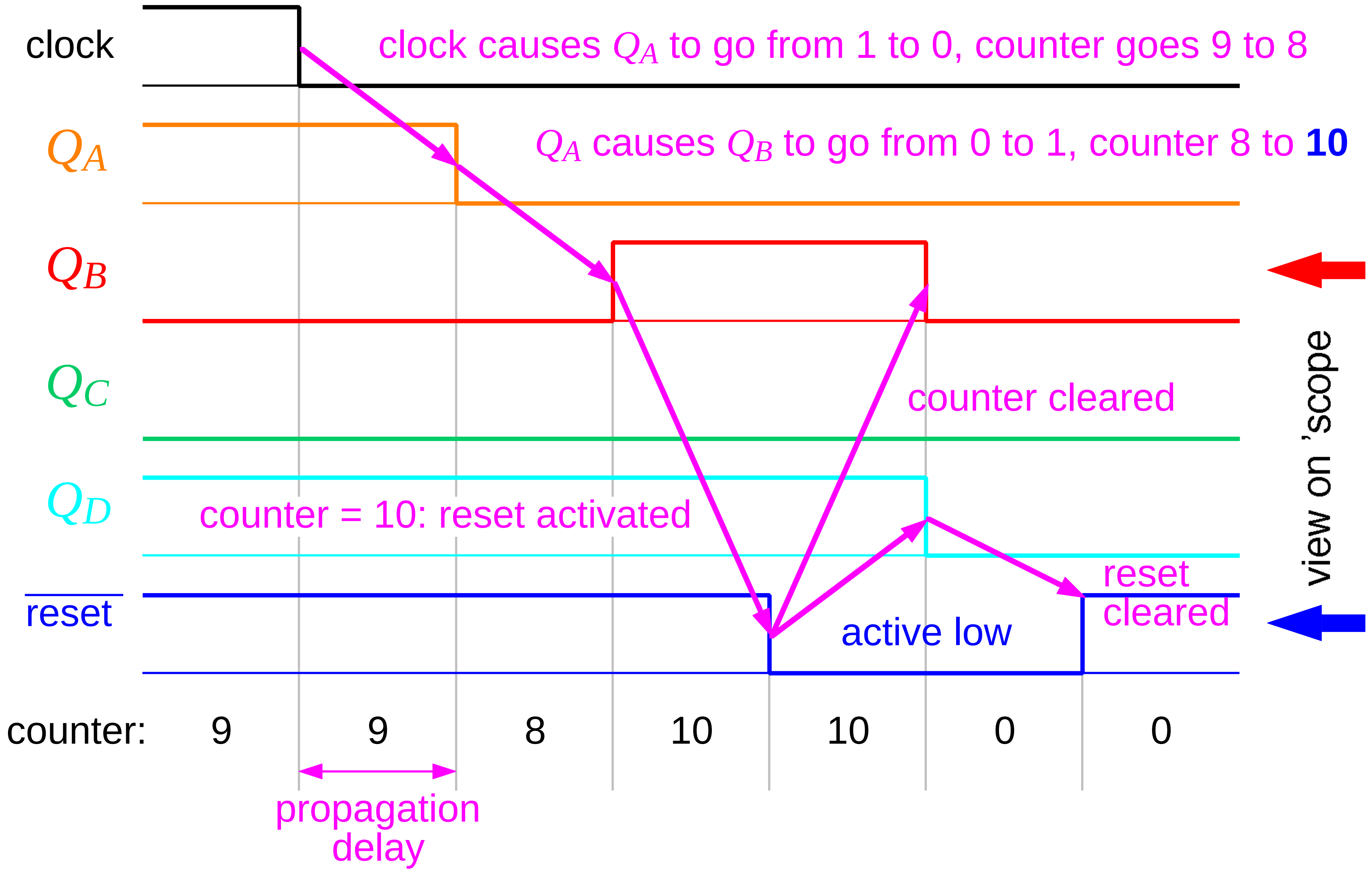
This will now roll over from 9 to 0 as required

- **but** it must enter the first unwanted state (10) briefly so that the reset (clear) pulse can be generated
- **look at the reset process in more detail**

Reset sequence in an asynchronous decade counter

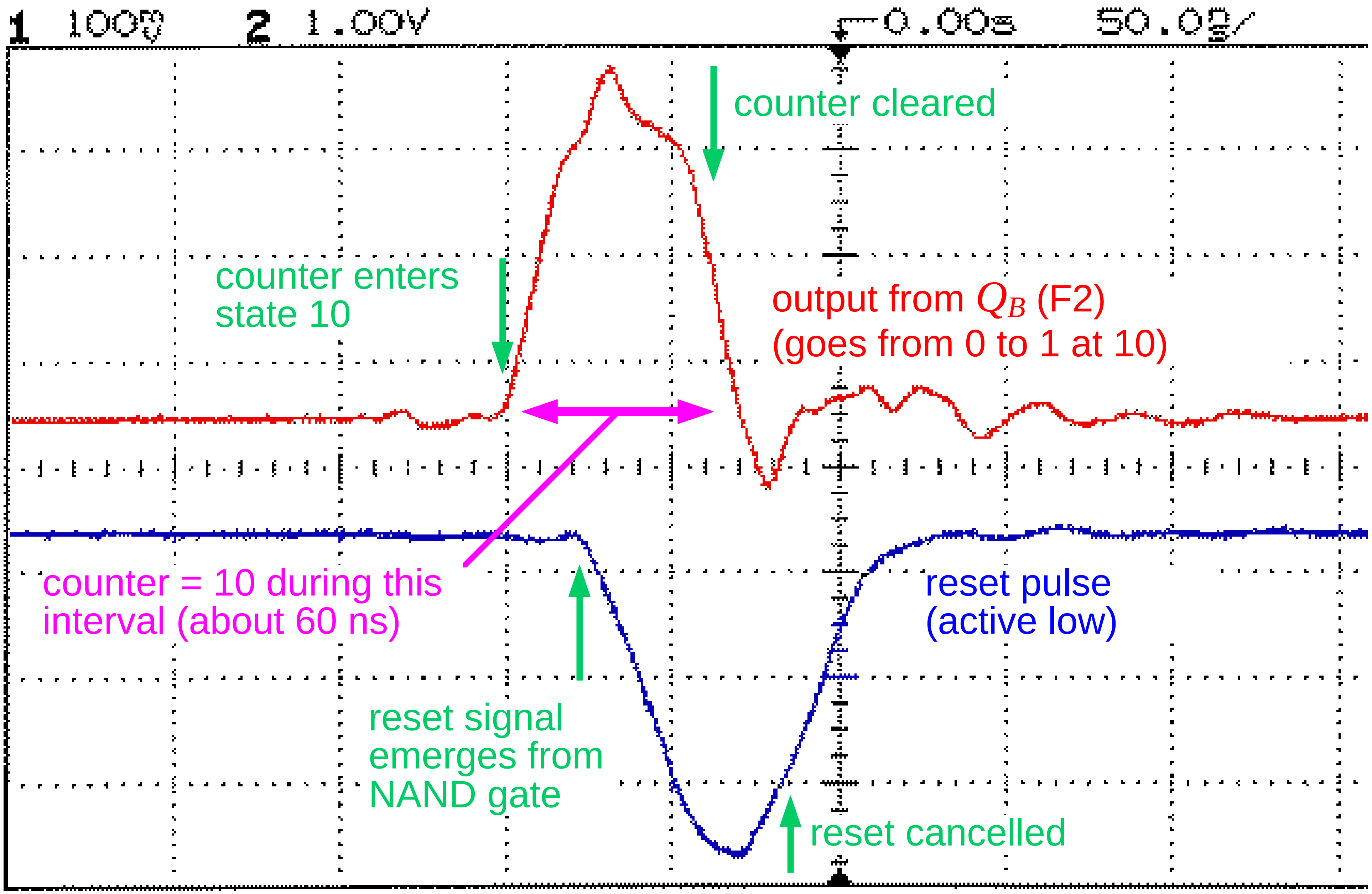

clock
Q_A
Q_B
Q_C
Q_D
$\overline{\text{reset}}$
clock causes Q_A to go from 1 to 0, counter goes 9 to 8
Q_A causes Q_B to go from 0 to 1, counter 8 to **10**
counter cleared
counter = 10: reset activated
reset cleared
active low
view on 'scope
counter: 9 9 8 10 10 0 0
propagation delay

Reset sequence in an asynchronous decade counter



clock
Q_A
Q_B
Q_C
Q_D
$\overline{\text{reset}}$
clock causes Q_A to go from 1 to 0, counter goes 9 to 8
Q_A causes Q_B to go from 0 to 1, counter 8 to **10**
counter cleared
counter = 10: reset activated
reset cleared
active low
view on 'scope
counter: 9 9 8 10 10 0 0
propagation delay

Unwanted state (10) and reset pulse in asynchronous decade counter



Example — design a counter to cycle through 1–6

This might be used to build an **electronic die** — singular of ‘dice’ — for example. (You will build a die later in the course with a microcontroller, but this project used flip-flops and gates in the past.)

This can be designed in the same way as the decade counter, but needs to be reset to 1 rather than 0. This can be done by using both Preset and Clear inputs.

Down counters can be built in a very similar way to up counters. See examples on the tutorial sheet.

Review exercises

What is the connection between **counting in binary** and **division by 2**?

Why is it important that the flip-flops in a simple, ripple, up counter be **negative edge triggered**?

A circuit diagram shows the flip-flops in a simple binary ripple counter. The values in the flip-flops, read from left to right, are 1010. What is the numerical value held in the counter as a whole?

An **oscilloscope** displays the clock and *Q* outputs from a binary counter. What feature shows that it is a ripple counter?

Why do ripple counters hold **incorrect values** for some of the time?

What is the general rule for designing a **counter whose range is not a power of 2**, such as a decade counter?
Hint: which state should be detected by the reset circuit?

How (in outline) would you design a counter to cycle around 1–10?

How do you make a binary **down** counter? — see tutorial sheet.