P2 Digital Electronics

Lecture 6: Memory, Latches and Clock Mode Circuits

Mark Cannon

mark.cannon@eng.ox.ac.uk

Trinity Term 2026

Overview of lectures

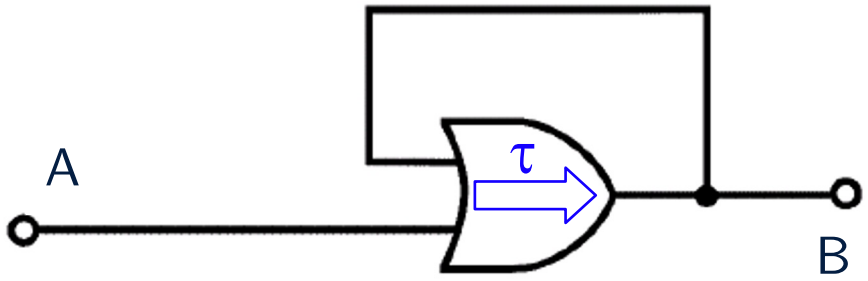
1. Logical functions and logic gates
2. Low level logic design
3. Binary number representation
4. Binary arithmetic
5. Integration of digital logic components
6. **Memory and sequential circuits**
7. Design of sequential logic
8. Data converters: analogue to digital / digital to analogue

Please send feedback, comments and corrections to mark.cannon@eng.ox.ac.uk

Outline of lecture 6

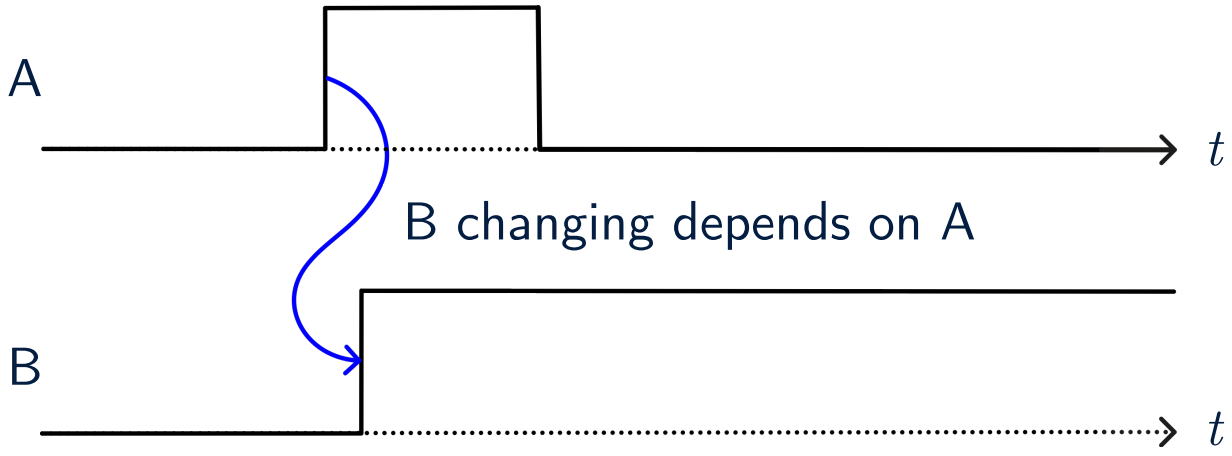
- Memory – the SR latch
- Transparency
- Clocked master-slave flip-flop
- D-type latch
- Edge triggered D-type flip-flop
- Shift register
- Asynchronous counter
- D-type flip-flop with reset

Creating a memory: a simple latch



OR gate with feedback

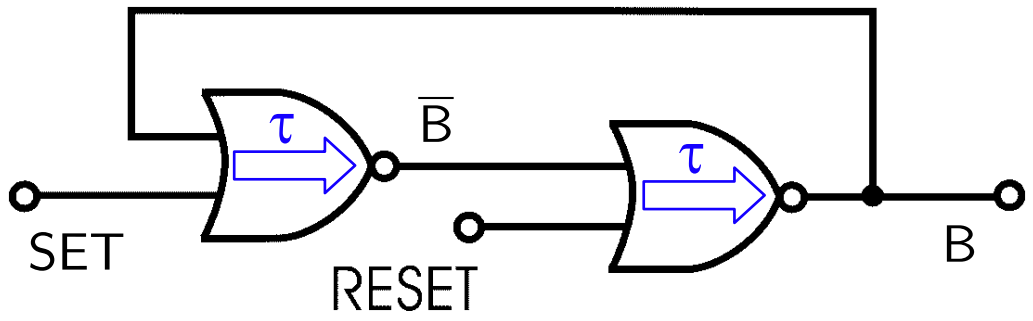
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1



Timing diagram showing edge dependence

The feedback allows us to record an event (A changing from 0 to 1) ...
but we can't set B back to 0!

S-R latch



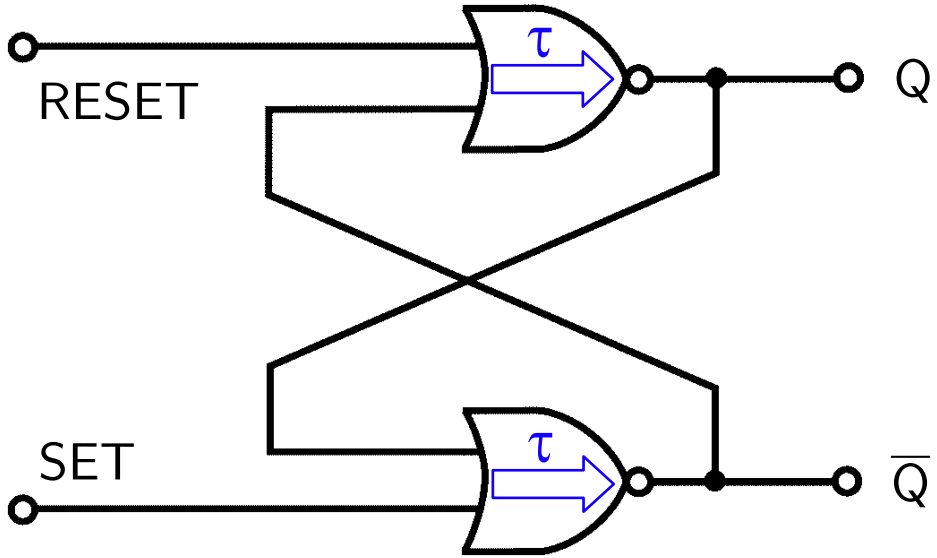
$$\overline{B}_{n+1} = \overline{\overline{\overline{B}_n + \text{RESET} + \text{SET}}}$$

S	R	Q_{n+1}	
0	0	Q_n	Stay same
0	1	0	Record a 0
1	0	1	Record a 1
1	1	X	Not allowed!

- 2 NOR gates
- inputs: Set and Reset
- outputs: Q and NOT Q

$$Q_{n+1} = Q_n.\overline{\text{RESET}} + \text{SET}$$

$$\overline{Q}_{n+1} = \overline{\overline{(\overline{Q_n} + \text{RESET})} + \text{SET}}$$

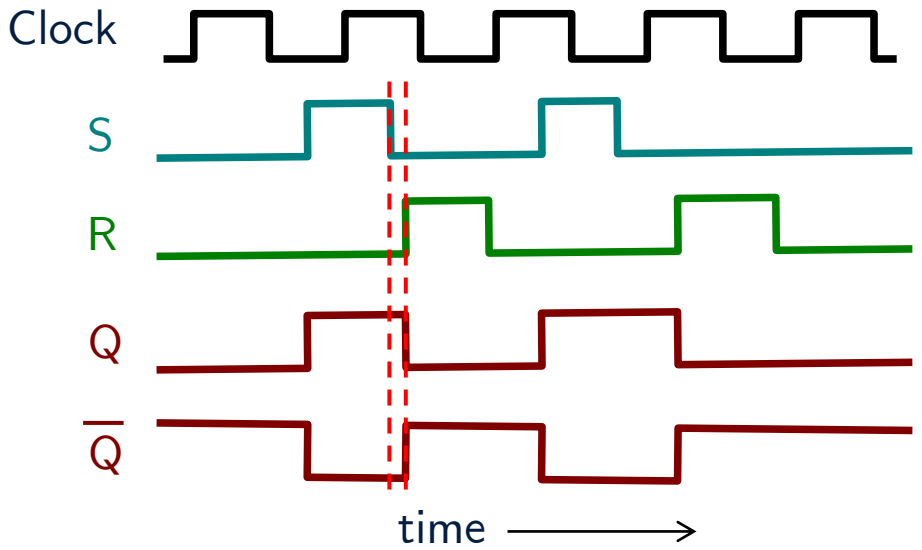


SR latch is TRANSPARENT - outputs change (almost) immediately

Clocked latches

Transparent latches are useful, but have a drawback:

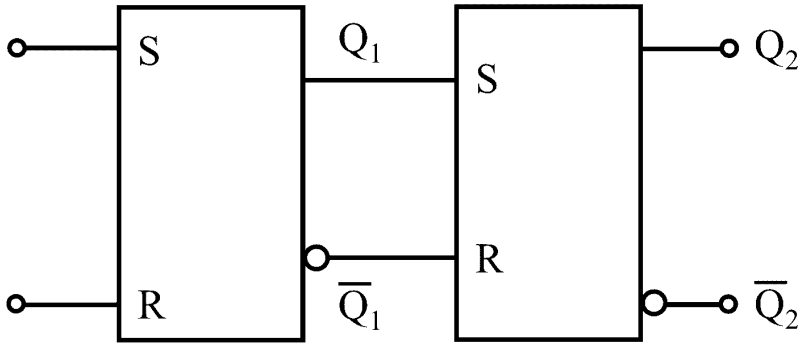
- the outputs change as soon as the inputs vary
- if the two inputs are coming from different parts of a system, they may not both change simultaneously
- need a method to synchronize transitions to a clock



S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	X

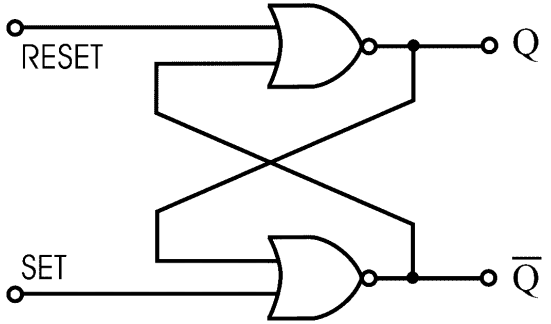
Not allowed!

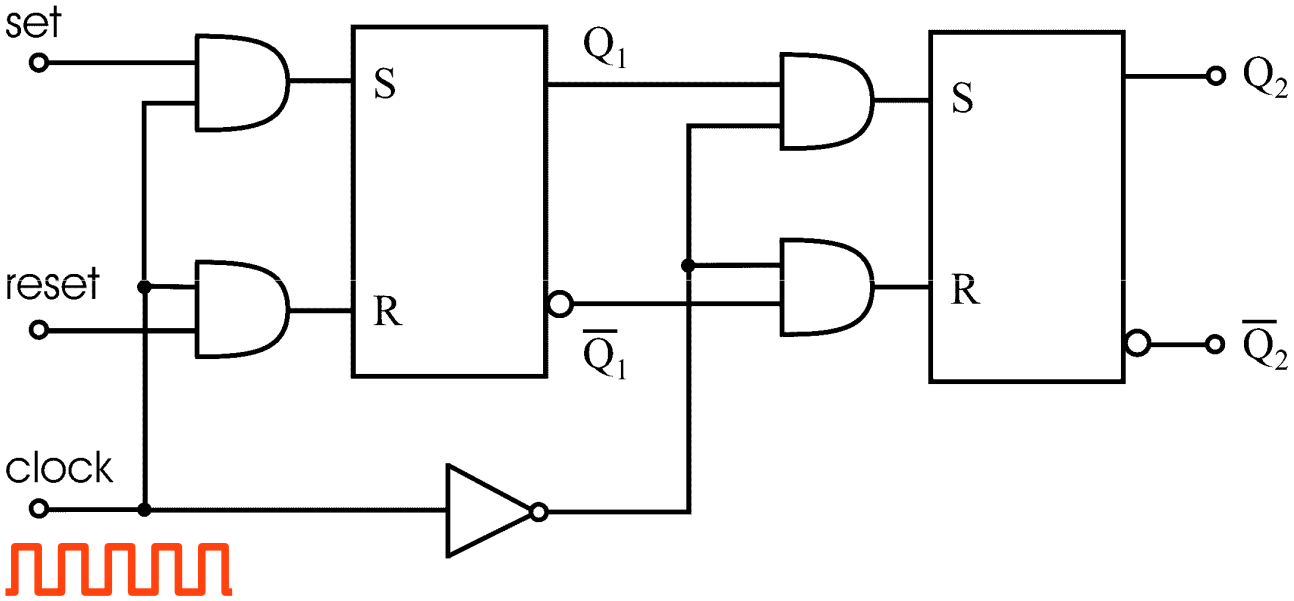
Clocked master-slave design



To decouple the input and output changes we could try linking two **latches:** one to store the bit, one to control changes

But we need synchronisation



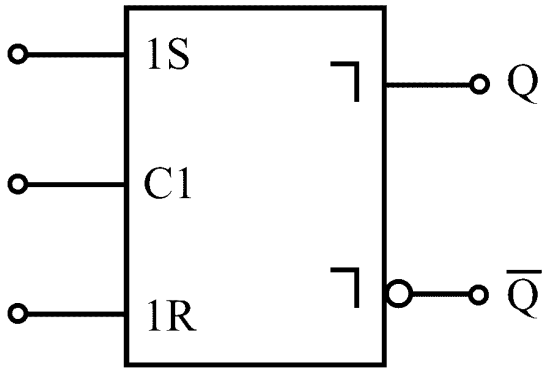


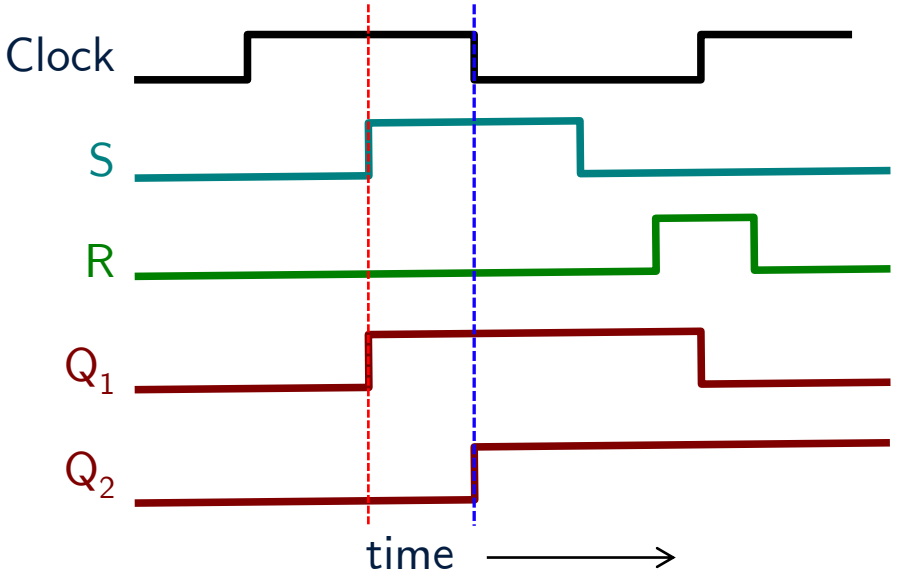
Gates added, with a 'clock' signal

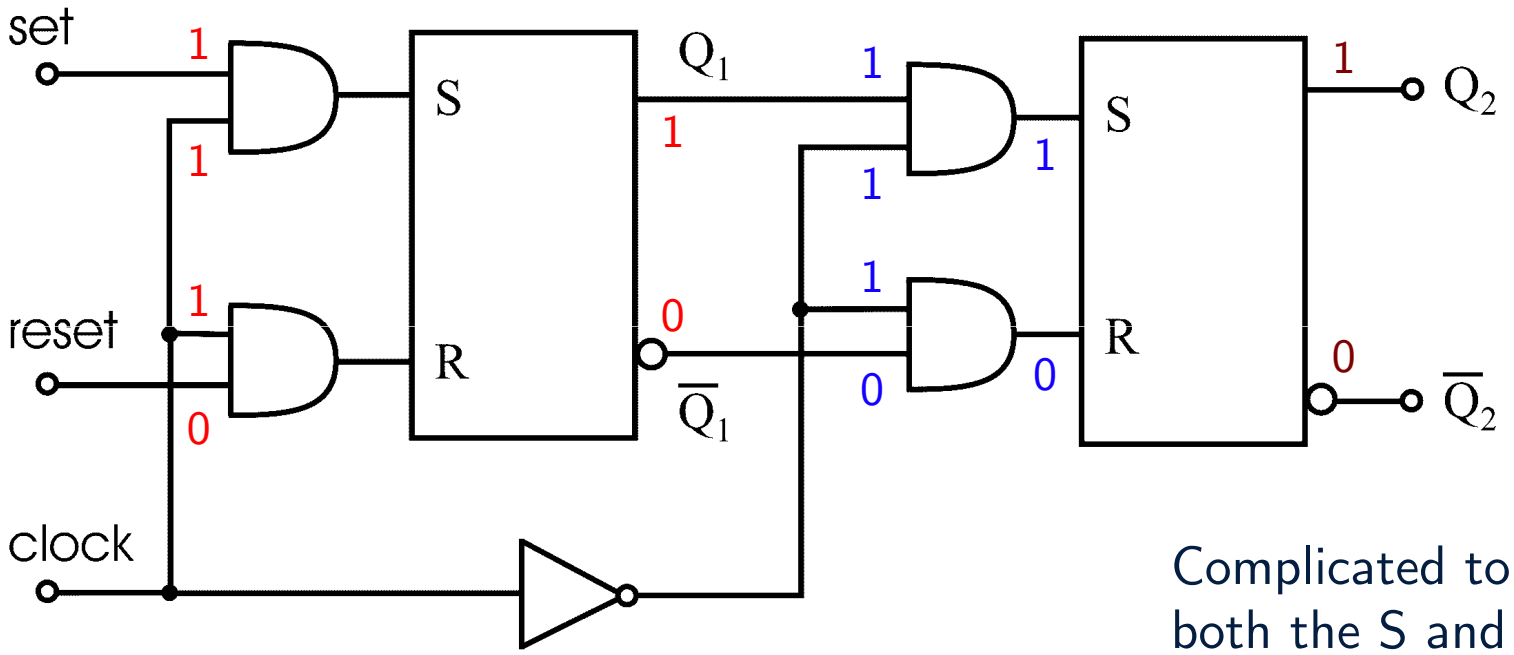
Now inputs of both latches only change when a clock pulse arrives

Output Q_2 only changes on a clock edge

How it works....





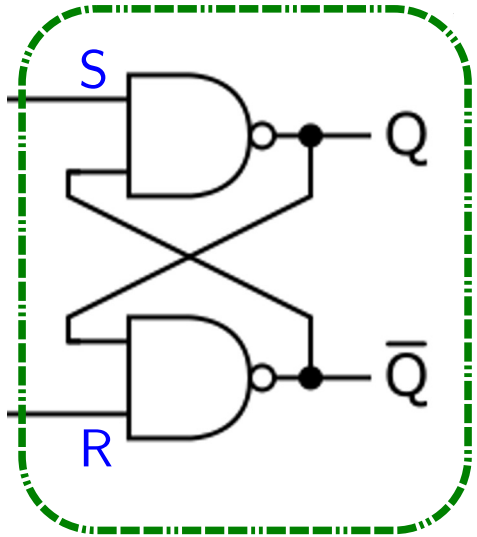


Transitions are now synchronised

Complicated to implement as we need both the S and R signals

NAND SR latch

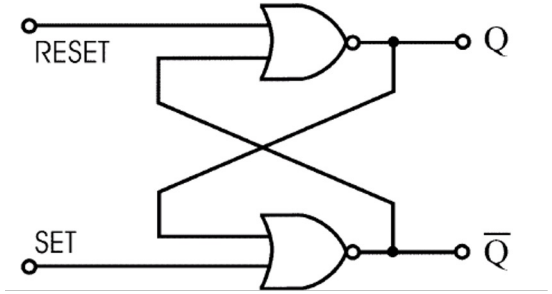
NAND SR latch



Same topology as NOR SR latch, but NAND SR latch has different truth table:

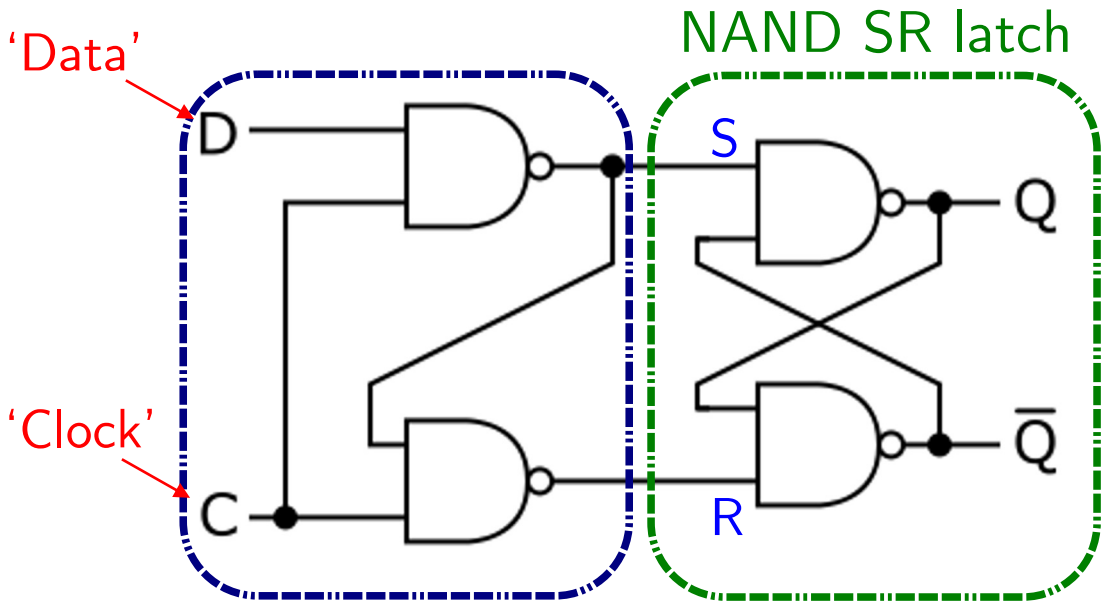
S	R	Q_{n+1}
0	0	X
0	1	1
1	0	0
1	1	Q_n

Compare NOR SR latch:



S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	X

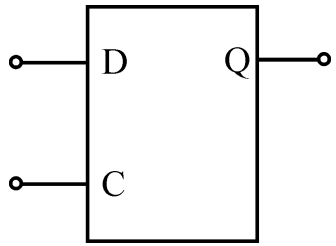
D-type latch

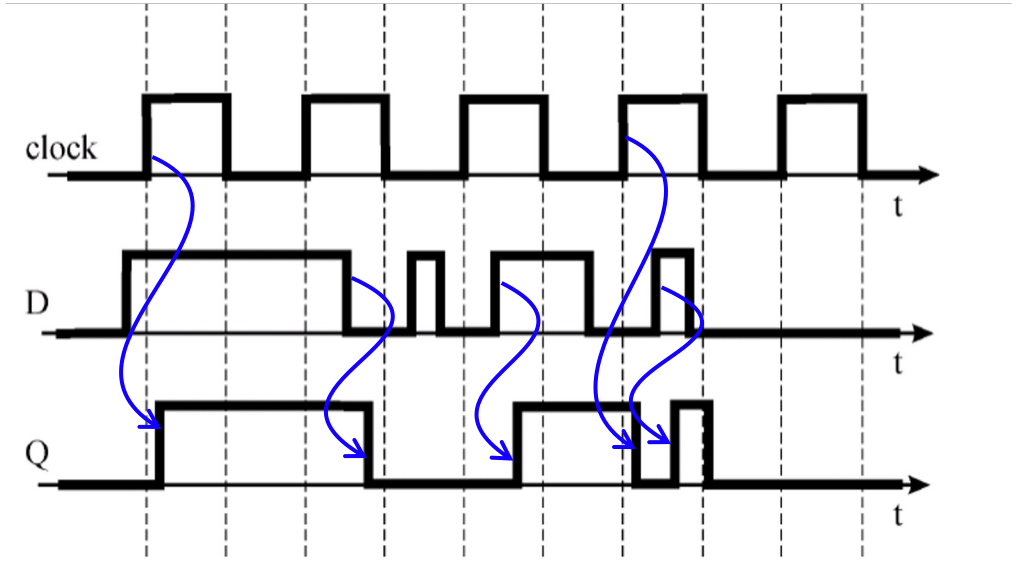


D	C	S	R
0	0	1	1
0	1	1	0
1	0	1	1
1	1	0	1

S	R	Q_{n+1}
0	0	X
0	1	1
1	0	0
1	1	Q_n

D	CLK	Q_{n+1}
X	0	Q_n
0	1	0
1	1	1

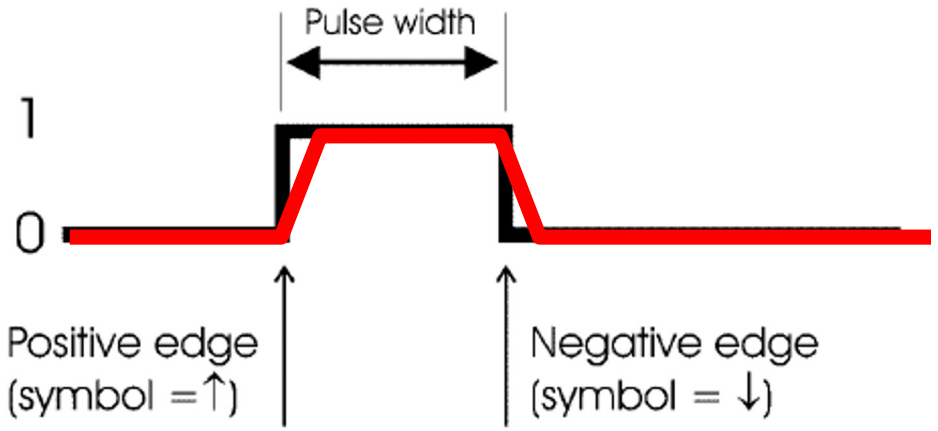




The D type latch is transparent when the clock is high,

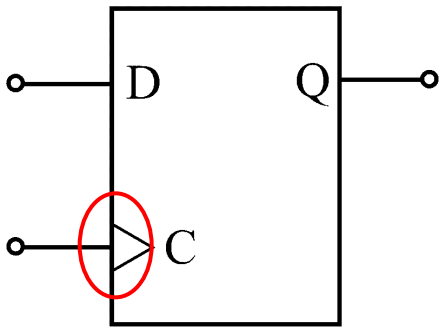
holds previous value if clock is low

The clock pulse



- We can build circuits with both positive and negative clock edge triggering
- The output changes shortly after the appropriate clock transition
- Edges are not infinitely steep hence slight delays always present

Edge triggered D-type flip-flop



D	C	Q_{n+1}
X	0	Q_n
X	1	Q_n
0	↑	0
1	↑	1

We only get a transition in the output when a positive clock edge arrives

D latch is a clocked device

- ⋆ transparent when clock is high

D-type flip-flop is also clocked

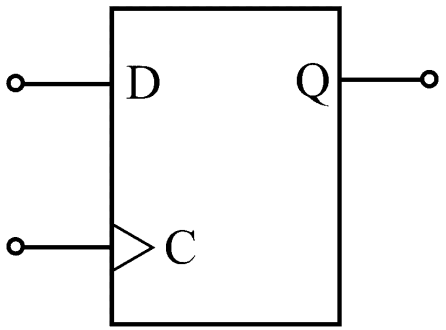
- ⋆ only changes on clock edge
- ⋆ both can store 1 bit of data

Note the difference. . .

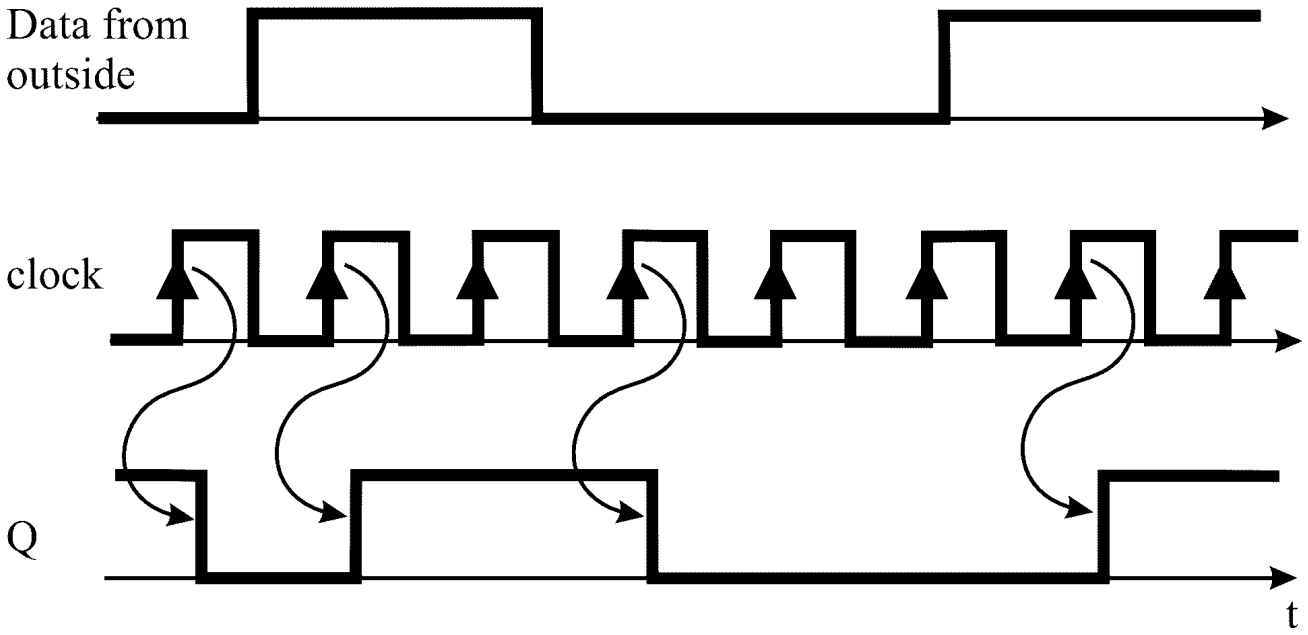
Latch – output changes when any input changes (after propagation delay)

Flip-flop – output only changes when the clock changes

D-type flip-flop timing



D	C	Q_{n+1}
X	0	Q_n
X	1	Q_n
0	↑	0
1	↑	1



Inside a D-type flip-flop

Actually 3 SR latches - NAND version

Input stage processes clock and D signals to ensure correct signals at SR stage

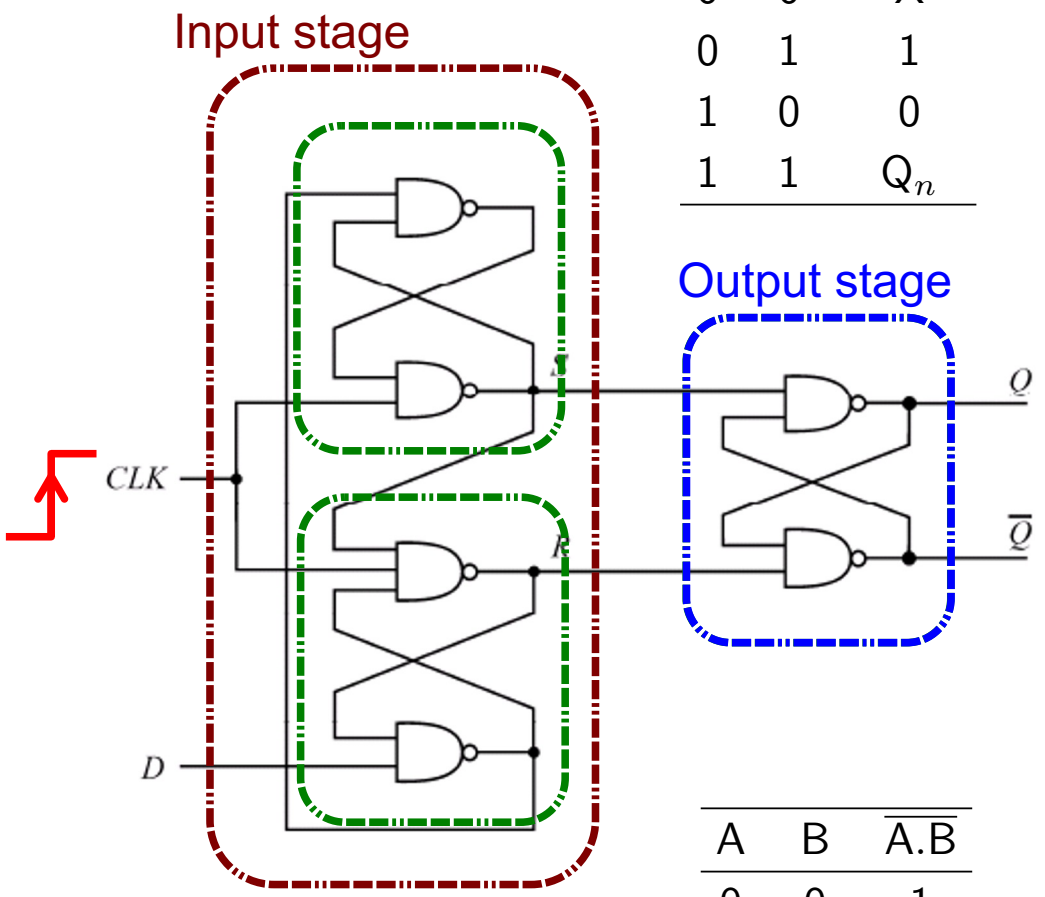
If clock low, both input stage outputs are high: output maintains its last value

On clock positive transition only one of the Input stage outputs change (depending on the value of D)

D $= 0$: output is set

D $= 1$: output is reset

Output latch is only there to store state between clock transitions



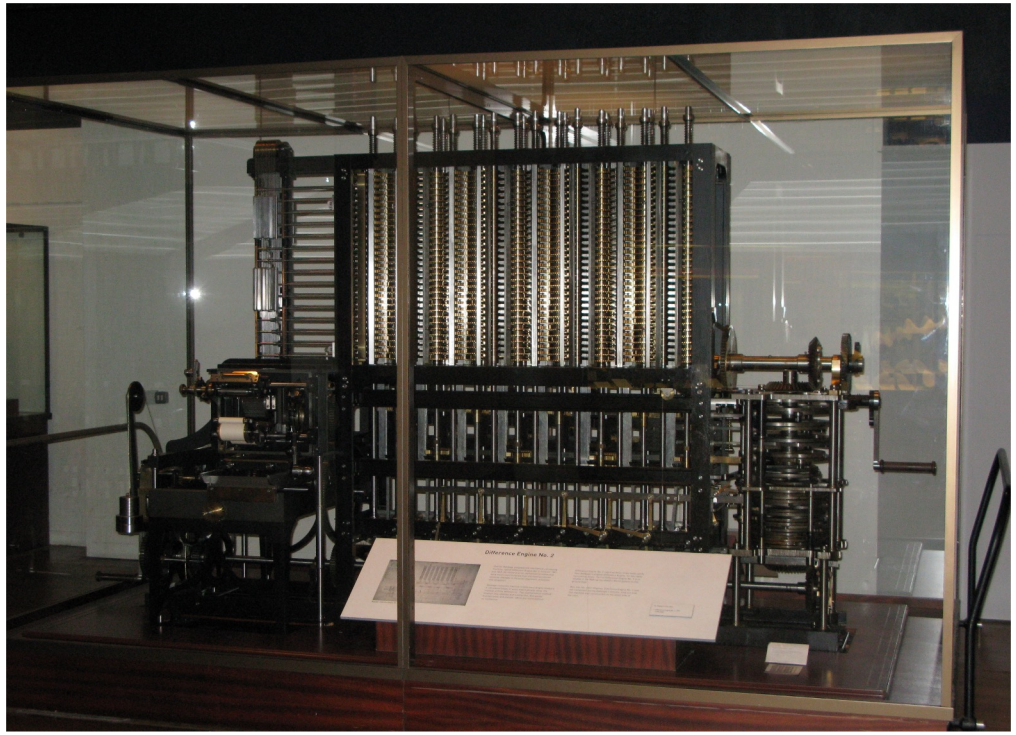
S	R	Q_{n+1}
0	0	X
0	1	1
1	0	0
1	1	Q_n

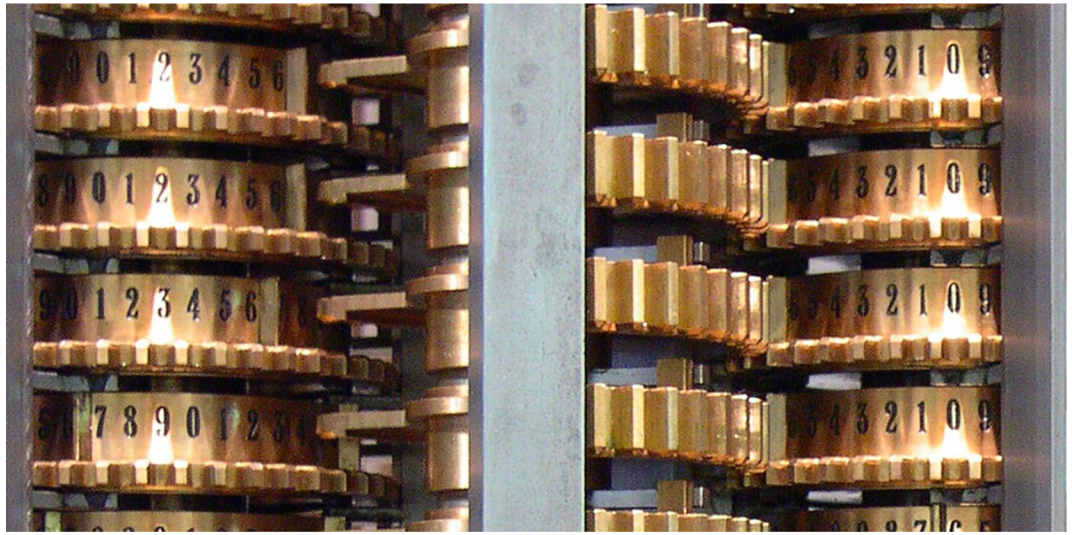
A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

The origins of programming

Babbage's analytical engine:

- ⋆ mechanical general purpose computer
- ⋆ prototype for many of the features of modern computing
- ⋆ ALU, controller, conditional branching, loops



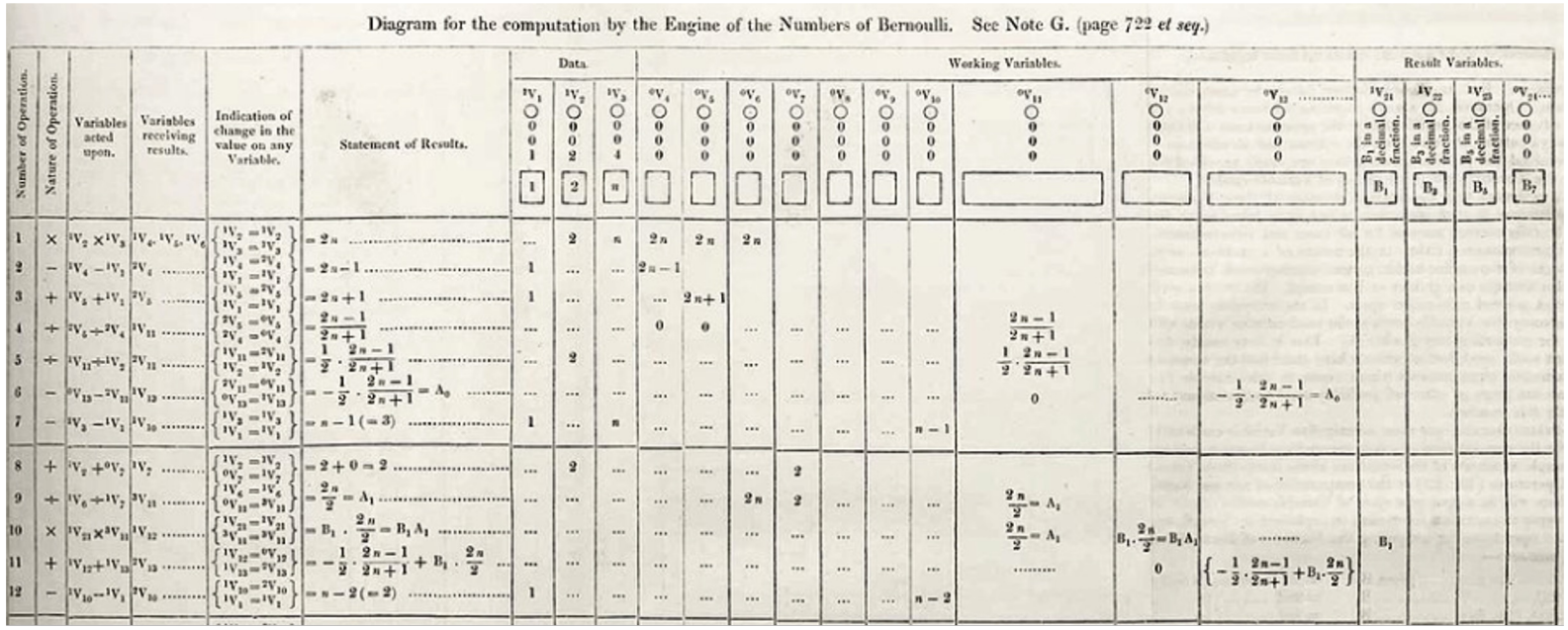


The origins of programming

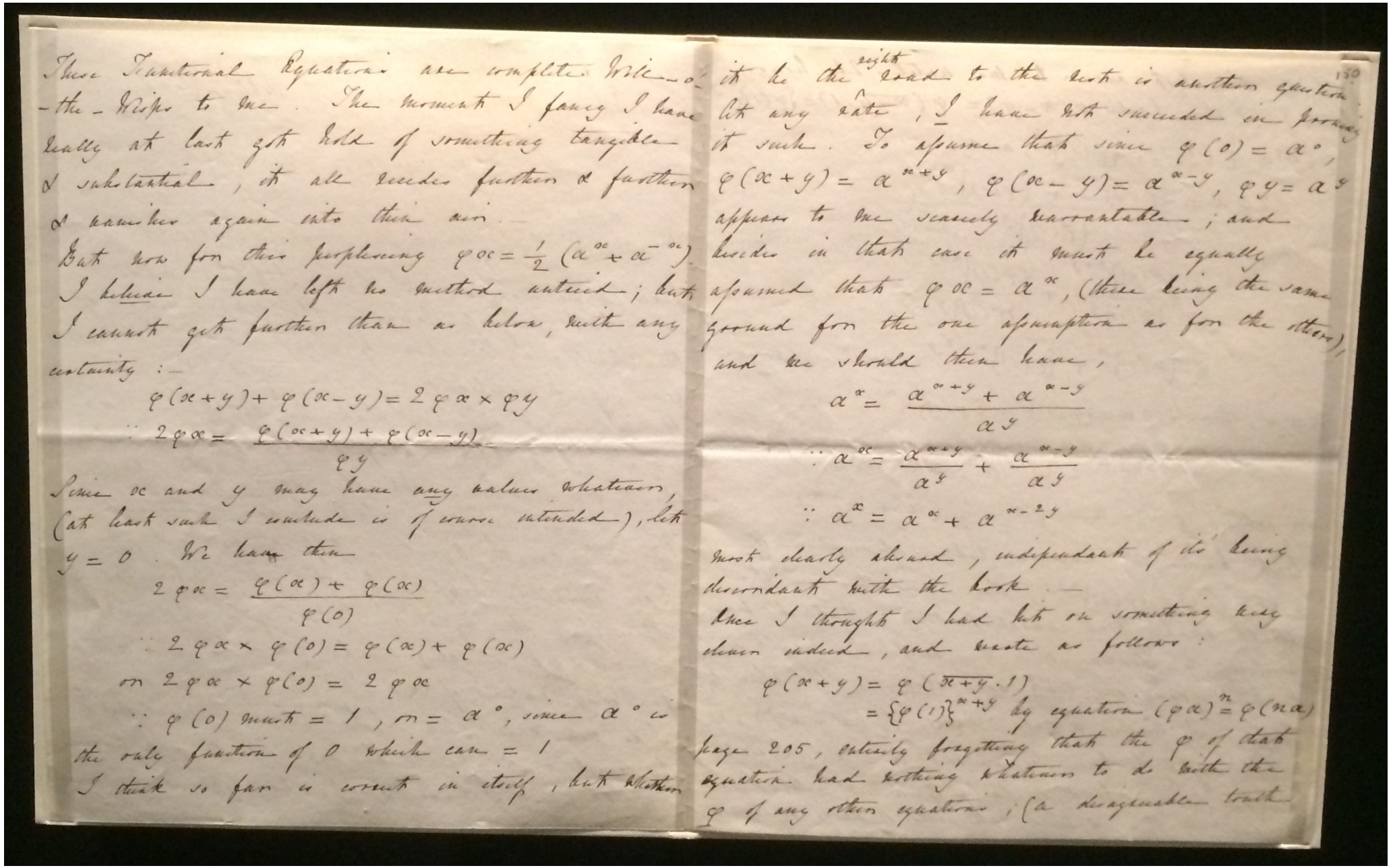
The first computer programs

- ⋆ Ada Lovelace (1815-1852) published an article In 1843 including an algorithm for calculating Bernoulli numbers
- ⋆ First realisation that computers could go beyond simple arithmetic

https://commons.wikimedia.org/wiki/File:Ada_Lovelace_portrait.jpg



https://commons.wikimedia.org/wiki/File:Diagram_for_the_computation_of_Bernoulli_numbers.jpg



These Functional Equations are complete Nile-o'-the-Wisps to me. The moment I fancy I have really at last got hold of something tangible & substantial, it all recedes further & further & vanishes again into thin air.—

But now for this perplexing $\varphi x = \frac{1}{2}(a^x + a^{-x})$. I believe I have left no method untried; but I cannot get further than as below, with any certainty:—

$$\varphi(x+y) + \varphi(x-y) = 2\varphi x \times \varphi y$$

$$\therefore 2\varphi x = \frac{\varphi(x+y) + \varphi(x-y)}{\varphi y}$$

Since x and y may have any values whatever, (at least such I conclude is of course intended), let $y = 0$. We have then

$$2\varphi x = \frac{\varphi(x) + \varphi(x)}{\varphi(0)}$$

$$\therefore 2\varphi x \times \varphi(0) = \varphi(x) + \varphi(x)$$

$$\text{or } 2\varphi x \times \varphi(0) = 2\varphi x$$

$\therefore \varphi(0)$ must $= 1$, or $= a^0$, since a^0 is the only function of 0 which can $= 1$

I think so far is correct in itself, but whether it be the right road to the end is another question. At any rate, I have not succeeded in pursuing it such. To assume that since $\varphi(0) = a^0$, $\varphi(x+y) = a^{x+y}$, $\varphi(x-y) = a^{x-y}$, $\varphi y = a^y$ appears to me scarcely warrantable; and besides in that case it must be equally assumed that $\varphi x = a^x$, (there being the same ground for the one assumption as for the other), and we should then have,

$$a^x = \frac{a^{x+y} + a^{x-y}}{a^y}$$

$$\therefore a^x = \frac{a^{x+y}}{a^y} + \frac{a^{x-y}}{a^y}$$

$$\therefore a^x = a^x + a^{x-2y}$$

most clearly absurd, independant of its being discordant with the book.—

Once I thought I had hit on something very clever indeed, and wrote as follows:

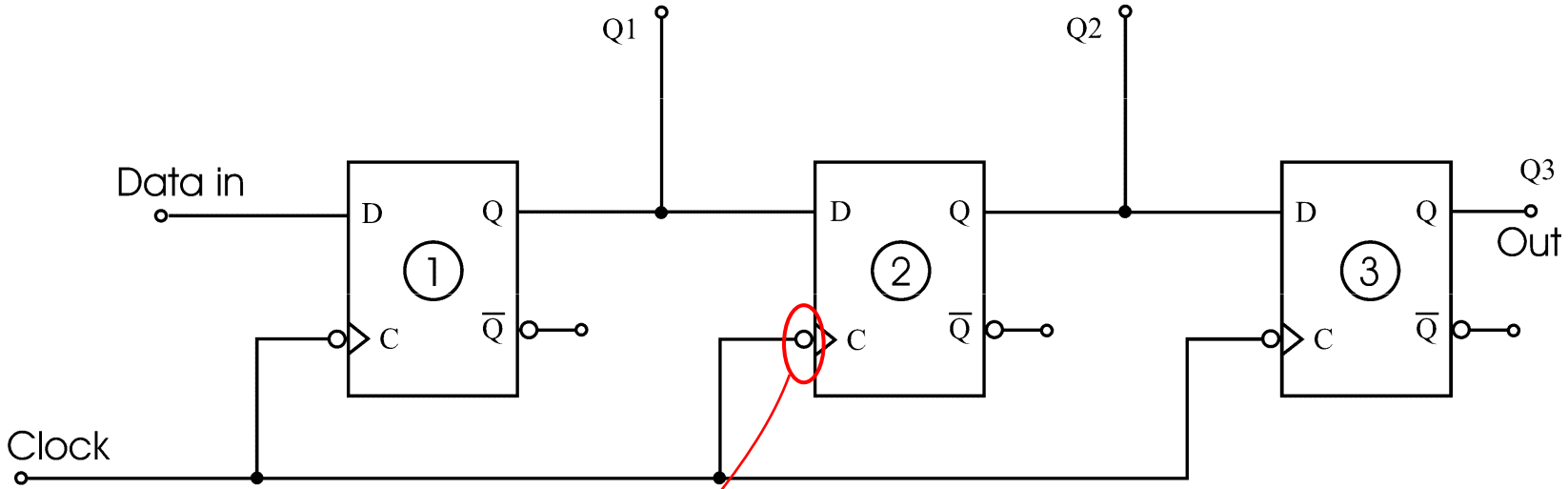
$$\varphi(x+y) = \varphi(\overline{x+y} \cdot 1)$$

$$= \{\varphi(1)\}^{x+y} \text{ by equation } (\varphi\alpha)^n = \varphi(n\alpha)$$

page 205, entirely forgetting that the φ of that equation had nothing whatever to do with the φ of any other equation; (a distinguishable touch

Correspondence between Ada Lovelace and Augustus De Morgan (Bodleian Library)

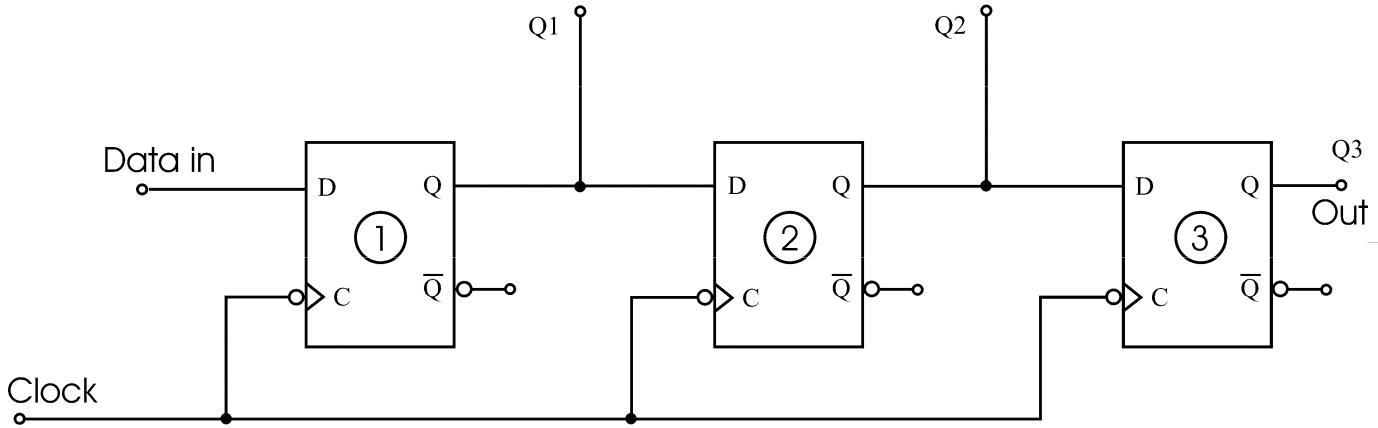
D-type flip-flop in action: Shift register



A 3-bit shift register with D-type flip-flops

D	C	Q_{n+1}
0	0	Q_n
1	0	Q_n
0	1	Q_n
1	1	Q_n
X	↓	D_n

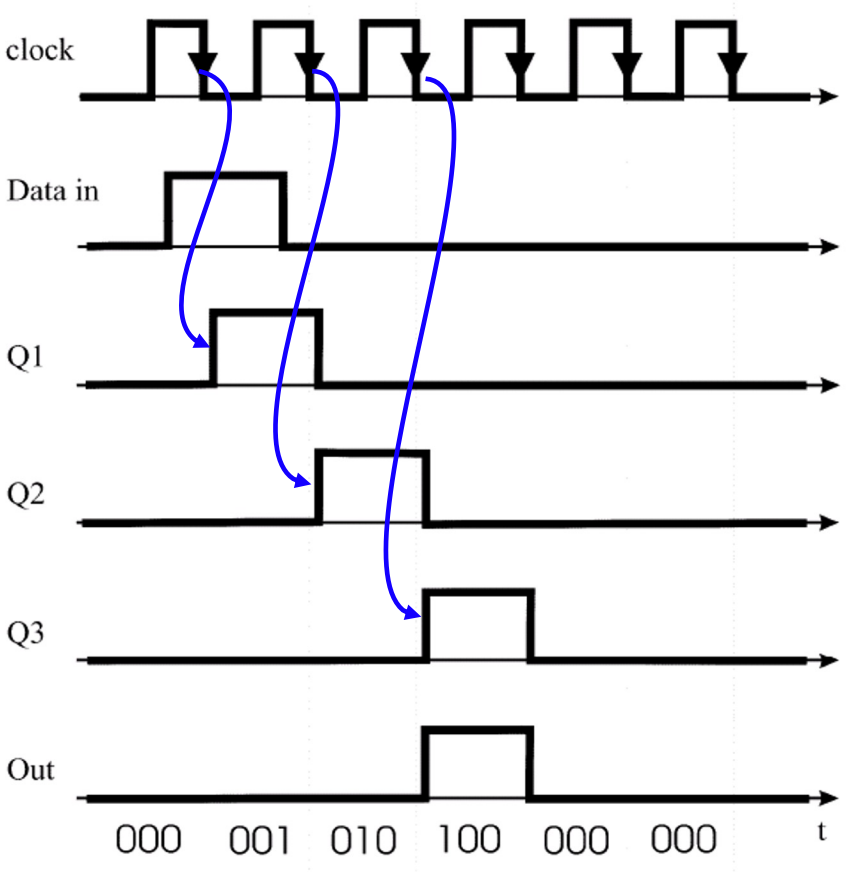
D-type flip-flop in action: Shift register



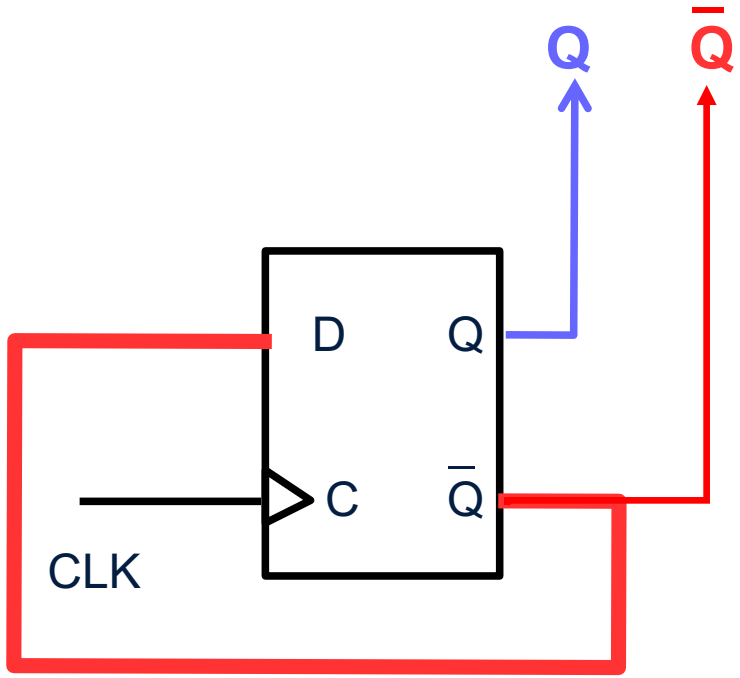
A 3-bit shift register with D-type flip-flops

D	C	Q_{n+1}
0	0	Q_n
1	0	Q_n
0	1	Q_n
1	1	Q_n
X	↓	D_n

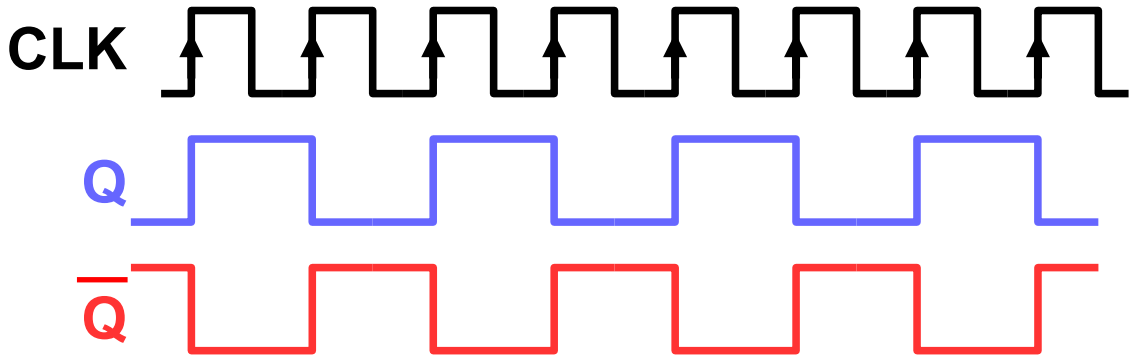
Negative edge triggered



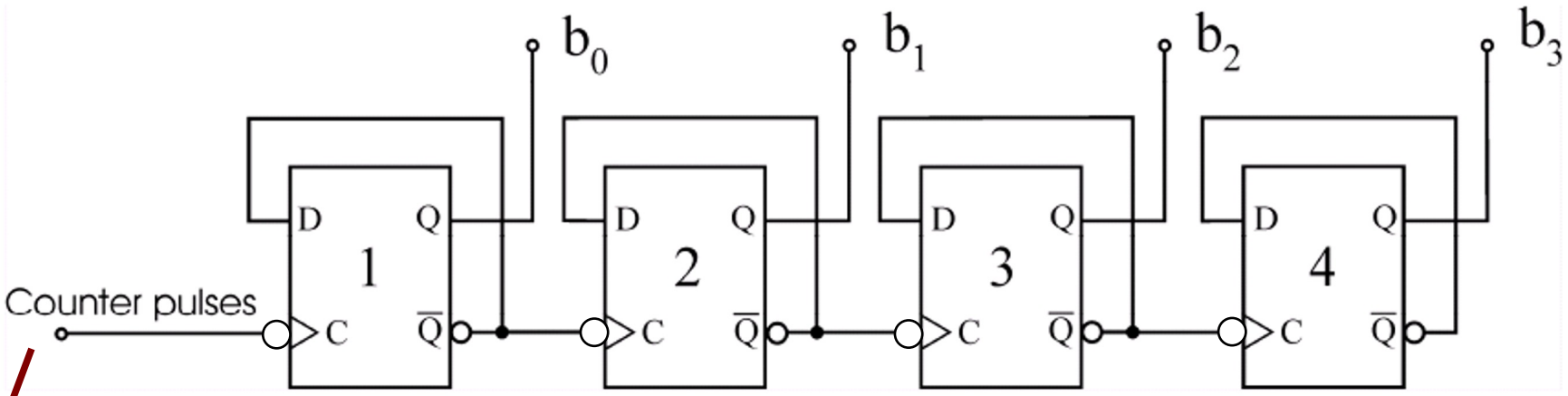
Counting with flip-flops

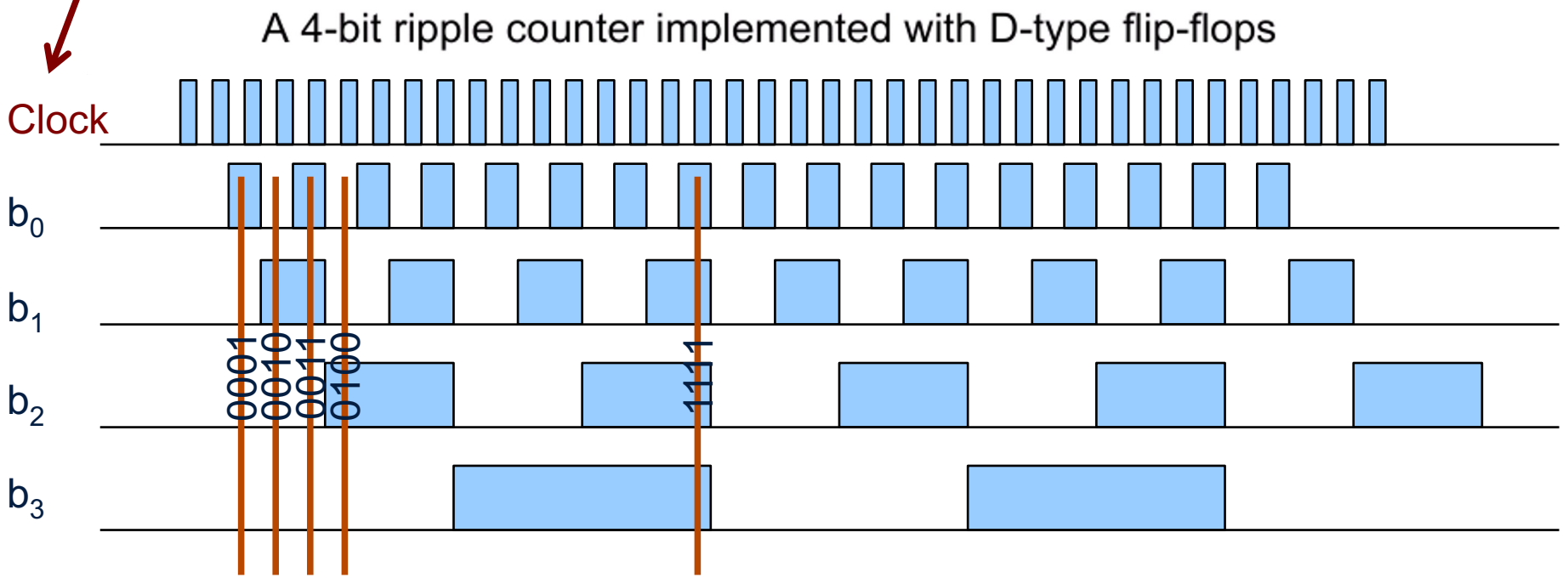


D	C	Q_{n+1}
0	0	Q_n
1	0	Q_n
0	1	Q_n
1	1	Q_n
X	↑	D_n

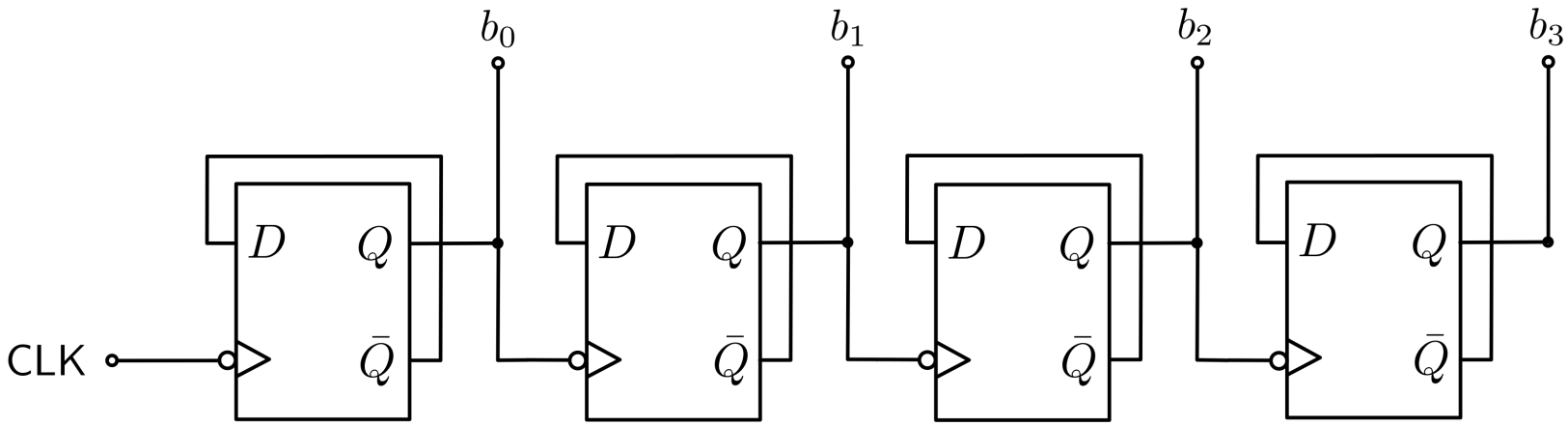


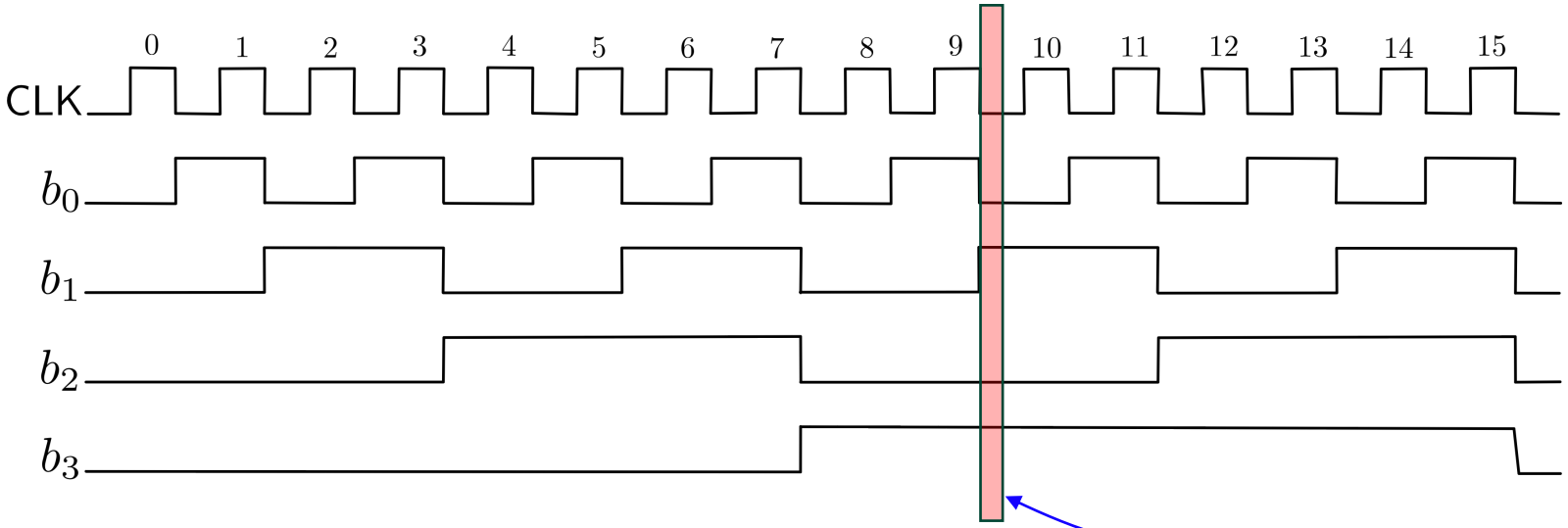
Asynchronous (ripple) counter





Asynchronous counter





What if we don't want to count up to 15?

We need a way to reset the system when a lower number is reached

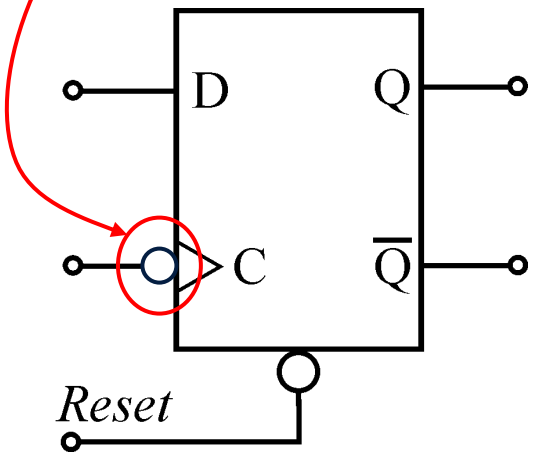
To count only from 0 to 9 we must activate reset

when $b_3\, b_2\, b_1 b_0 = 1010 = 10_{10}$

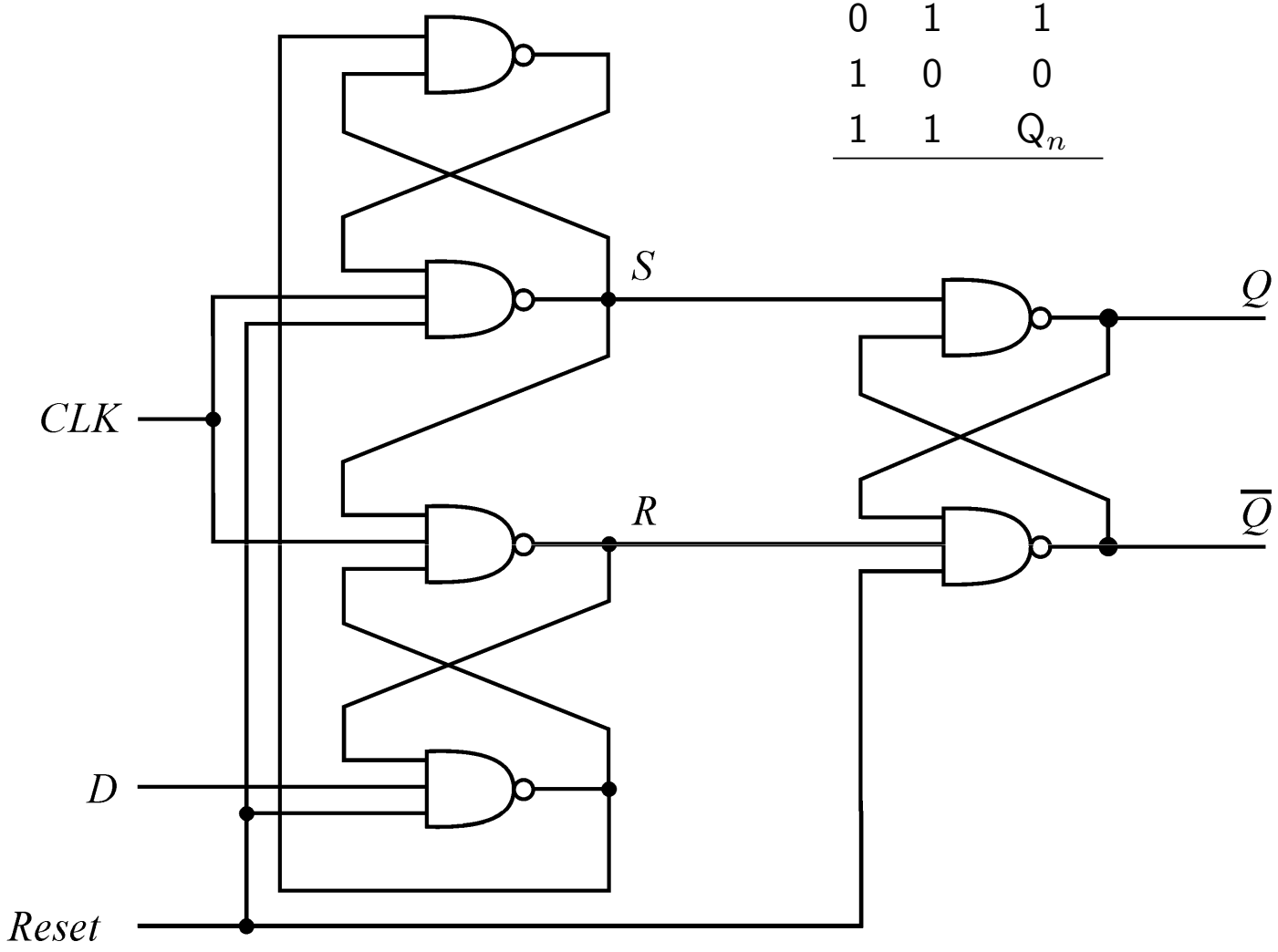
i.e. when $b_3.\bar{b}_2.b_1.\bar{b}_0 = 1$

D-type with (synchronous) reset

D	C	Reset	Q_{n+1}
X	0	1	Q_n
X	1	1	Q_n
0	↓	1	0
1	↓	1	1
X	X	0	0



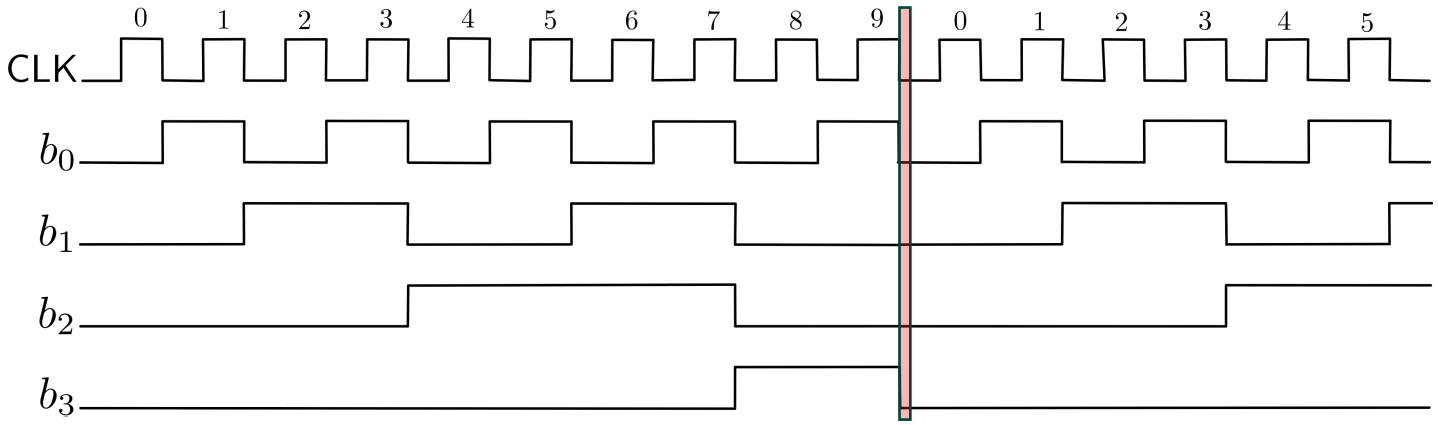
S	R	Q_{n+1}
0	0	X
0	1	1
1	0	0
1	1	Q_n

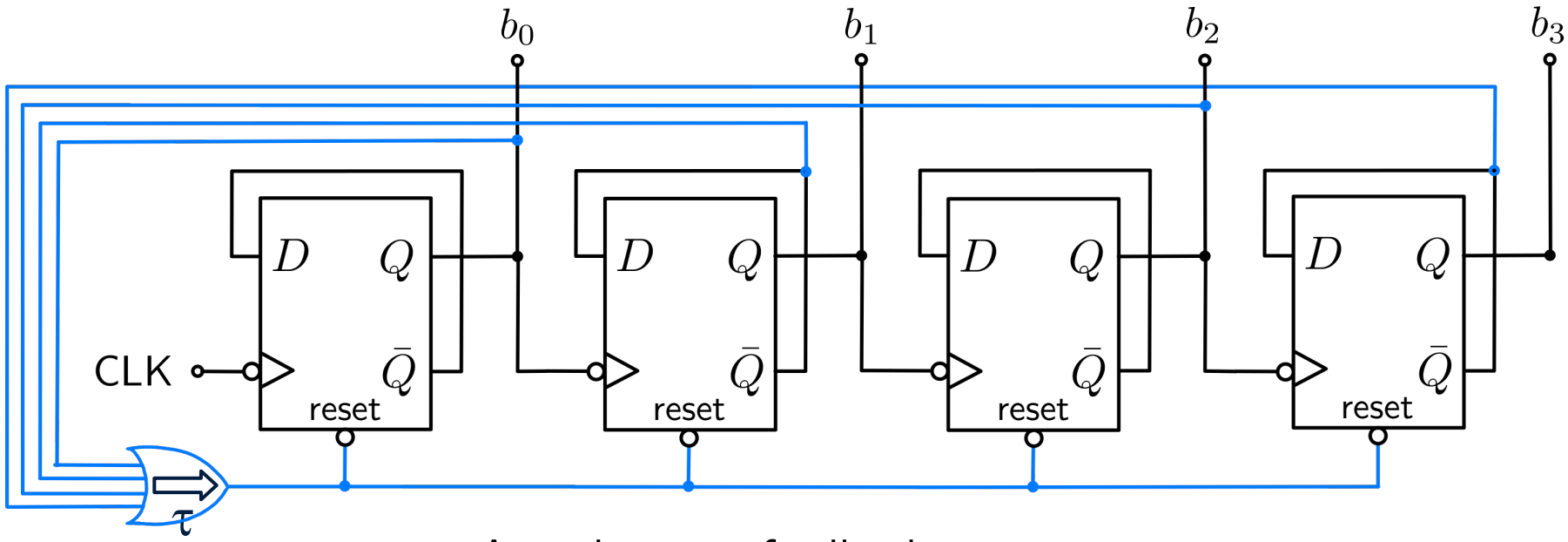


Modulo 10 counter

Modulo $10 = 0, 1, 2, \ldots, 8, 9, 0, 1, \ldots$

Logic is needed to detect when we reach $10_{10} = 1010_2$ and reset counter to zero





Logic function giving 1 when $b_3 b_2 b_1 b_0 = 1010$:

$$R = b_3.\bar{b}_2.b_1.\bar{b}_0$$

$$\Rightarrow \quad \overline{R} = \bar{b}_3 + b_2 + \bar{b}_1 + b_0$$

Asynchronous feedback
Causes timing issues:
glitches or "runt pulses"

Overview of lectures

1. Logical functions and logic gates
2. Low level logic design
3. Binary number representation
4. Binary arithmetic
5. Integration of digital logic components
6. **Memory and sequential circuits**
7. Design of sequential logic
8. Data converters: analogue to digital / digital to analogue

Please send feedback, comments and corrections to mark.cannon@eng.ox.ac.uk