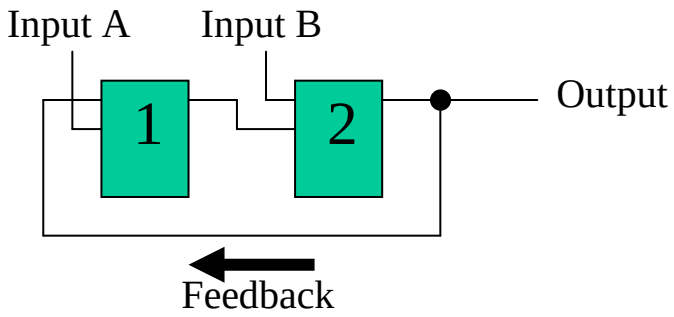
Digital Electronics 2: Sequential Logic

Sequential Logic (it cares about the past): Feedback & Memory

We are now moving on from very simple logic gates – where some "1"s and "0"s at the input give either a "1" or a "0" at the output – to more complicated situations where time becomes important. This will get us to a point where we can build motor controllers and memory elements – both situations where time plays a crucial role. Logic where time plays an important role is called **sequential logic**.

A cartoon of an arbitrary circuit using feedback:

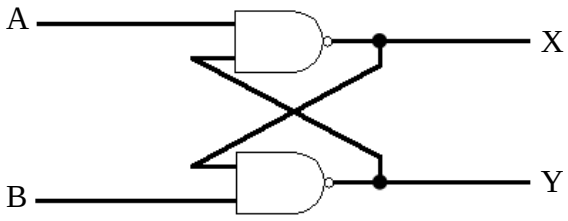


Here the rectangles labelled 1 and 2 are supposed logic gates of some sort. The lines are wires which connect the input to the gates and the gates to each other and to the output. Interesting *behaviour as a function of time* emerges because the output is also one of the inputs. This is called **feedback** and we are going to go through how this works.

As you can probably imagine, delays (due to the *length of wires* or the *speed of response* of a logic gate) cause big problems in circuits that make use of feedback. This is overcome by making use of a **clock**. The clock is a regular pulse which is attached to every logic gate, which triggers it to *re-evaluate its inputs* and to *update its output*. (This is the processor speed that you are used to hearing about for computers.)

Flip-Flops: (H&H, 8.16, p.504)

We are now going to look at a simple circuit with feedback built out of two NAND gates. This circuit is called a basic **Flip-Flop**, a bi-stable unit, which works as a basic memory element. The two inputs A and B go into two separate NAND gates – the two outputs X and Y are the outputs of the two NAND gates. The feedback happens because *each gate feeds into the second input of the other gate*. See the diagram below.



And remember: each NAND gate has this truth table

α	β	Q
0	0	1
0	1	1
1	0	1
1	1	0

OK – a fairly simple circuit. You may notice that there is a symmetry plane along the midline, i.e. that top and bottom halves of the circuit are identical. Now let's take a look at how it behaves.

Rule of thumb: if one of the inputs to a NAND gate is 0 then the output is 1.

Response to A,B=1,0 or A,B=0,1 is sensible.

The NAND gate with a 0 input has to have a 1 output.
Hence the other gate has a 1,1 input, resulting in a 0 output – stabilising the other state.

Response to A,B=1,1 is more difficult.

By symmetry you would think that both outputs must be identical. Wrong!
Changing from A≠B to A=B=1 preserves old X,Y. **Memory!** (The device history has broken the symmetry)

The basic Flip-Flop Operation Table:

A	B	X_{n+1}	Y_{n+1}
0	0	1	1
0	1	1	0
1	0	0	1
1	1	X_n	Y_n

The process of making the value of a memory element equal to 1 (here the X output) is called **setting**. What do you need to input in order to "set" this memory element?

Likewise the process of making the value of a memory element 0 (here the X output) is called **resetting**. What do you need to input in order to "reset" this memory element?

Flip Flopping ≡ Setting & Resetting

What happens if you first enter A,B=0,0 – you see the flip flop respond – and then you enter A,B=1,1? Try it.
Both outputs of the flip flop regularly change states, they **oscillate** in time:

$X_{n+0}, Y_{n+0} = 0, 0$
$X_{n+1}, Y_{n+1} = 1, 1$
$X_{n+2}, Y_{n+2} = 0, 0$
$X_{n+3}, Y_{n+3} = 1, 1$
:

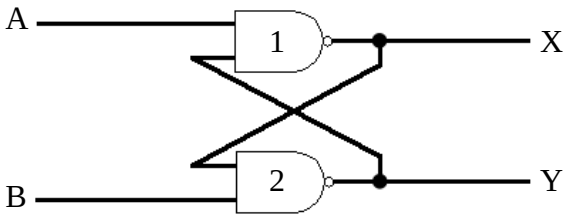
and so on, until you break the cycle.
This sequence of inputs should be avoided.

You are now supposed to understand the most basic operation of a flip flop memory element.

Design flaws of the basic Flip-Flop:

Although this basic flip flop unit provides a simple demonstration of memory the design is ultimately a disaster. There are actually two layers problems. We will unpack each and solve them in turn.

For the purpose of diagnosing the problems we label the two NAND gates "1" and "2"on the diagram and use subscripts to specify time steps:



The first problem we are dealing with is associated with the **lack of precise symmetry** between the two gates. Let's take a flip flop with an example state X_0, Y_0 = 1, 0 and issue a reset command at t_0. In the ideal case the sequence of events should look like this:

time	t_{-1}	t_0	t_{+1}
A	1	1	1
B	1	0	1
X	1	1	0
Y	0	0	1

Colour key:
blue indicates updated settings
red indicates updated evaluations

But what will happen in reality is that after setting B=0 at t_0, due to **finite signal propagation** and **slew rate**, it will take a little time to update Y (i.e. updating the output from X, Y = 1, 0 at t_0 to X, Y = 1, 1 at t_{+1}). After a further while, at t_{+2}, the signal Y will have propagated to the input of gate 1, the output still showing X, Y = 1, 1. After another while the signal X will have been evaluated and updated (i.e. updating the output from X, Y = 1, 1 at t_{+2} to X, Y = 0, 1 at t_{+3}). So, the sequence of events actually is:

time	t_{-1}	t_0	t_{+1}	t_{+2}	t_{+3}
A	1	1	1	1	1
B	1	0	0	0	0
X	1	1	1	1	0
Y	0	0	1	1	1

Disaster! At t_{+1} and t_{+2} the output is ill defined. Similar **transients** can occur if **one gate works a little faster** than the other or the signals to the **inputs arrive at slightly different times** at the flip flop.

The basic Flip-Flop is a failure because *it relies on perfect symmetry* between the components and on *instantaneous signal updates* – both of which are practically impossible. It also relies on the A and B *input signals arriving at exactly the same moment* – which is equally unrealistic.

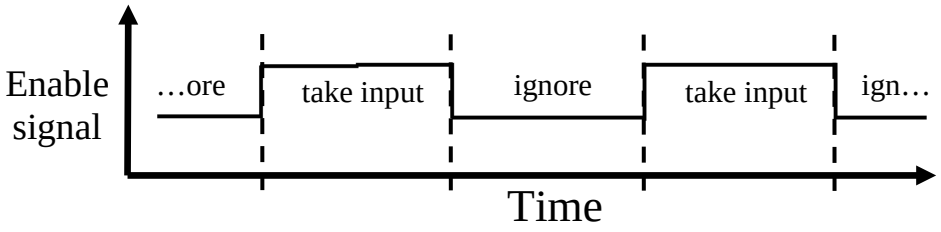
The flip flops have several problems:

- The system is sensitive to the details of the components.
- Some possible input combinations will cause the device to give oscillating outputs for ever (X=Y=1 followed by A=B=1).
- The oscillation condition can be fulfilled just because the input signals arrive at different times.

The first improvement is to exert some **control over when inputs arrive** with the NAND gates. To do this we are going to add some gates to the input side of the flip flop which prevent anything going to the flip flop unless an **enable input** is set to 1.

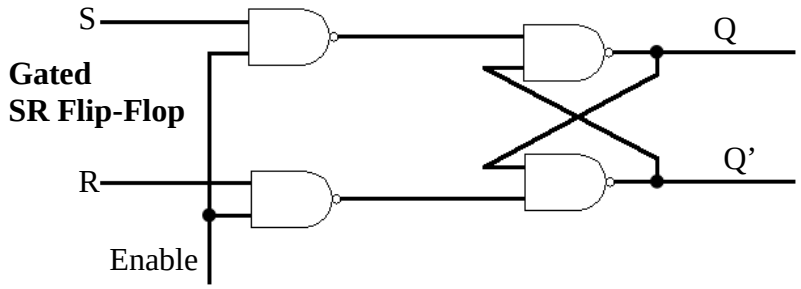
We want to control when the circuit accepts input, in order to control simultaneity and introduce a **clock** which controls the data flow. A clock is a regular sequence of logic "1"s and "0"s. The general from of a clock signal is pictured here:

This allows the flip flop to be *periodically* updated when it only responds to its inputs during one half-cycle of the clock ("take input") , and it is insensitive to the inputs during the other half-cycle ("ignore"), like in this picture:



If properly used, this can prevent the system taking a new input signal while a previous input is still being processed. During the "ignore" half-cycles the inputs can be updated, so that these input signals are passed on simultaneously when the "take input" order comes.

Gated SR Flip-Flop: (H&H, 8.17, p.507)
Here you see the circuit for the extension of a standard flip flop to a **gated SR Flip-Flop** by the use of an **enable signal**:



As desired the system responds only when enable=1. Remember: if either of the inputs to a NAND gate is zero then the output is 1. So when the enable=0 the system is entirely oblivious to the value of the inputs S and R. Also note that only NAND gates are used in this circuit. During the enable=1 being set the operation table can be determined. It looks like this:

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

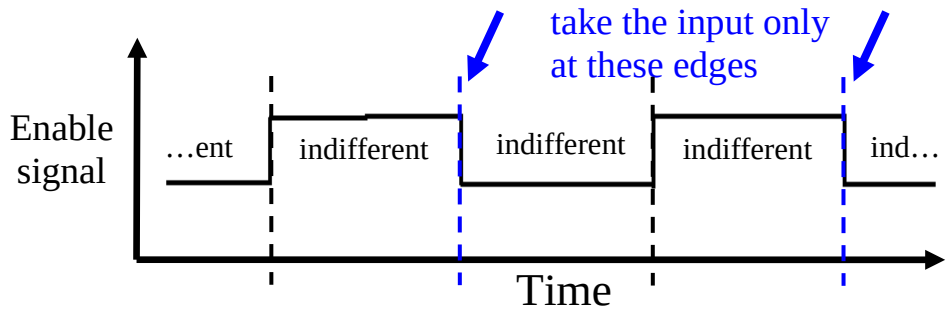
You will notice that the inputs have been labelled S and R corresponding to their "**set**" and "**reset**" roles. The outputs are labelled Q and Q' as they now are expected to be complements.

The S=R=0 input is the memory function i.e. this choice of input at the next enable=1 maintains the existing output value.
The S=R=1 condition must still be avoided so as not to create oscillations. Having input combinations that must be avoided is a hazard to the integrity of the circuit. So further fixing still will be needed.

We are now using the enable circuitry to control when the flip flop will accept input. This circuit is called a **transparent latch**. The name points to another reason why this circuit is still far from ideal. At any time while the enable=1 state is set any number of input changes can pass through to the flip flop. This can lead to oscillating outputs in the same way as with the simple circuit. Bummer!

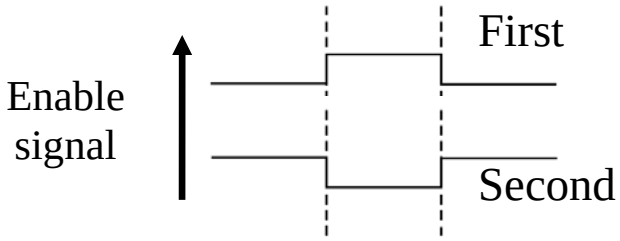
So why bother with this, if the fix doesn't work? Because it points the right way towards the ultimate design of a memory element for which all four input combinations have a properly defined function and where no transition problems occur.

Making the SR Flip-Flop better controlled is equivalent to making the "transparent latch" opaque. We don't want the flip flop to be open to changes in the input for any length of time. The solution can be drawn like this:

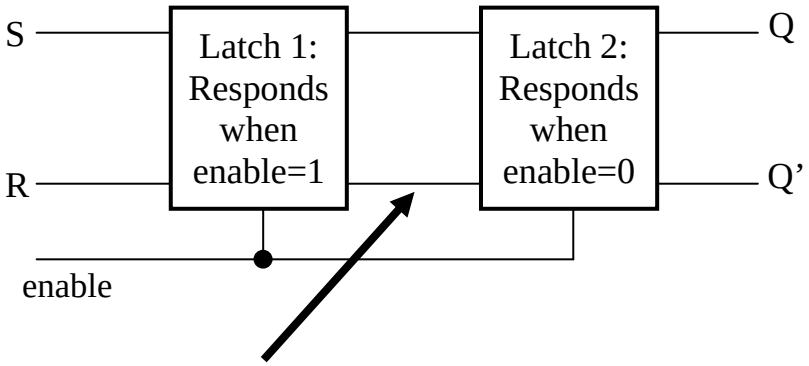


So, we want a circuit that accepts input only at the precise moment when the clock is changing, for example at the **falling edge** from 1 to 0, but not during the clock pulses. This will ensure sufficient simultaneity.

To do this, in essence you need to combine two "transparent latches" – one that accepts input when the clock is high and one which accepts input when the clock is low. So, combine two circuits with opposite phase:

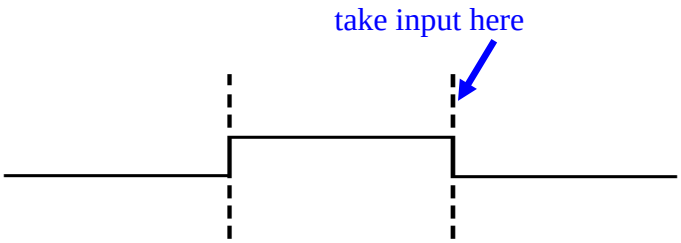


Below is a schematic illustration to make this clearer. Values at the input only make it half way through the circuit initially. When the clock changes from 1 to 0 they are then passed to the second half of the circuit:



The logic values that have reached here at the end of the clock pulse are the ones that are processed at the falling edge of the clock as demanded above.

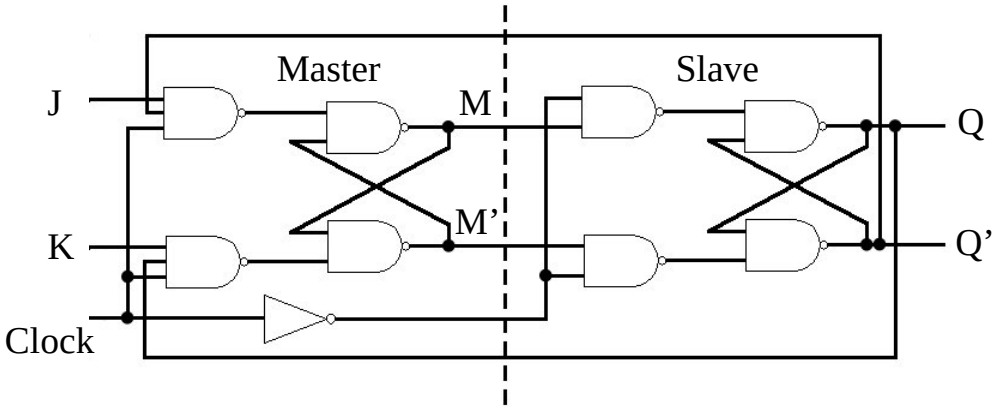
If such a circuit could be constructed then it would give the desired behaviour:



You are now supposed to understand the basic idea of how to make a memory element work such that it doesn't respond to changes in the input at the wrong moment.

Opaque latch: JK Flip-Flop (H&H, 8.17, p507)

The circuit we need to build is not trivial – it goes by the name **JK Flip-Flop**. People rarely go *completely* mad trying to understand how it works, usually just a little bit ☺.



Are you disappointed that you recognise the elements already?
Yes, the basic structure is that of two SR Flip-Flops in sequence: one labelled **Master** and the other labelled **Slave**, each using four NAND gates as introduced before. The output states of the Master are called M and M', those of the Slave are called Q and Q', in line with previous nomenclature. The inputs to the circuit are called J and K by convention.

The enabling of the two SR Flip-Flops is controlled by a *common clock* signal, which gets *inverted* using a NOT gate for the Slave to make it *operate counter-phase* with respect to the Master. The dashed line indicates the point that the input signals reach while the clock pulse is 1. As the clock changes to 0 the signal is processed further by the slave.

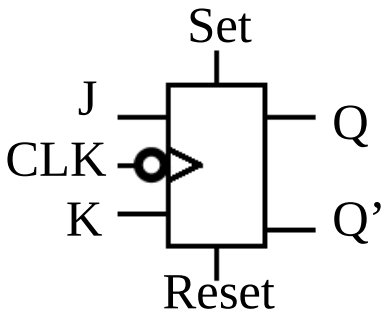
The only new circuit design elements are the **feedback lines** from the output of the Slave to the input of the Master. The lines *cross-feed* as for the basic flip flops. These additional lines also require the **triple-input NANDs**, which you haven't seen before. They follow the same rule-of-thumb as their dual-input brothers: if any input is 0 the output is 1.

The new feedback connections from Q and Q' to K and J ensure that the two outputs always have opposite values, hence the naming convention Q and Q', and that all four input combinations can be used. In addition to the memory state and the set and reset functions you know from the RS Flip-Flop, the input state of J=K=1 toggles the output at the next update, i.e. it inverts the states of Q and Q'.
This is very useful for counting applications. Here is the **operations table** of the JK Flip-Flop:

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	Q_n'

Its **characteristic equation** is: $\mathbf{Q_{n+1} = J.Q_n' + K'.Q_n}$

We now have a well behaved memory element for which all four inputs are used. In practice the details of the JK Flip-Flop are not drawn every time the circuit is used. Instead it has its own symbol:



The input for the clock signal that controls the circuit is labelled "CLK".

J and K can be used to "set" and "reset" the circuit. But in a number of common applications the two inputs J and K are attached together and fixed at a signal value (say 1). In order to still be able to separately initialize the output values separate "set" and "reset" inputs are provided. In the chips you use in the lab these input pins are labelled "preset" and "clear".

You will explore the behaviour of the JK Flip-Flop further in the lab.

Summary

- Feedback makes memory possible, however, it also makes the circuit very sensitive to small differences between apparently identical components.
- Using a clock to control the circuit keeps the behaviour sensible. It prevents important sections of the circuit responding to changes in the input values at unhelpful moments.
- The clock effectively limits feedback

You now have gained a basic idea of how digital signal processing is controlled, e.g. in computer CPUs or in serial data transmission.