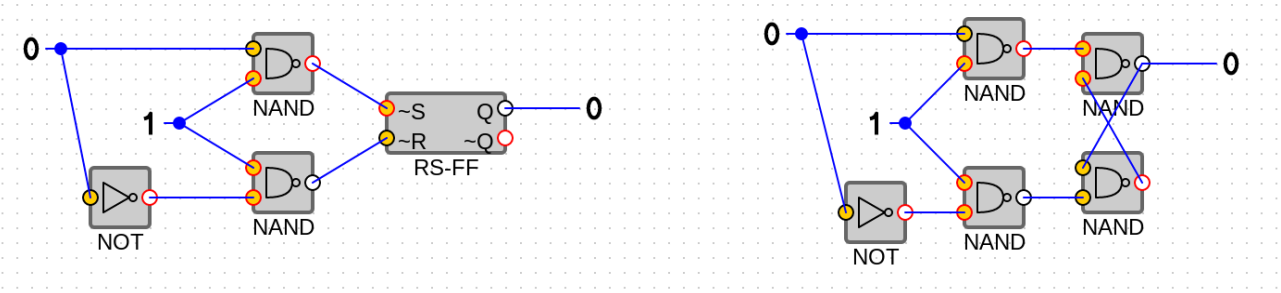
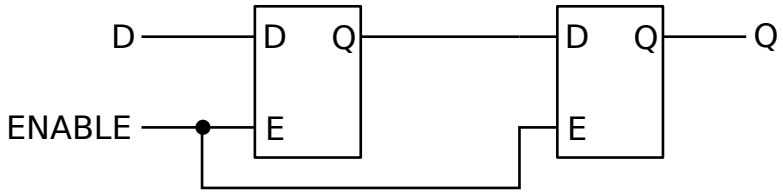
Build a latch in the simulator, using either an RS flip-flop or NAND gates, as below. Verify that it behaves in accordance with the above truth table.

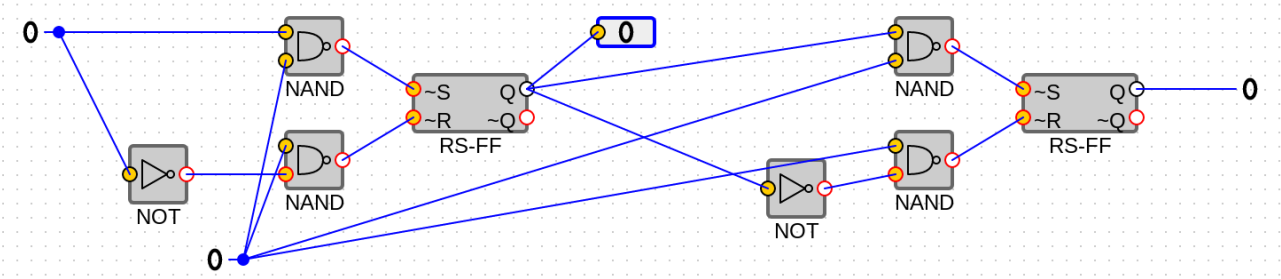


Connecting latches together

Now imagine two latches in series, so the output of the first one drives the input of the second one and they are driven by an enable signal, as below:



What happens when we apply a bit to the input of the first latch and set the enable high to write the bit to the latch? Try it in the simulator as below:



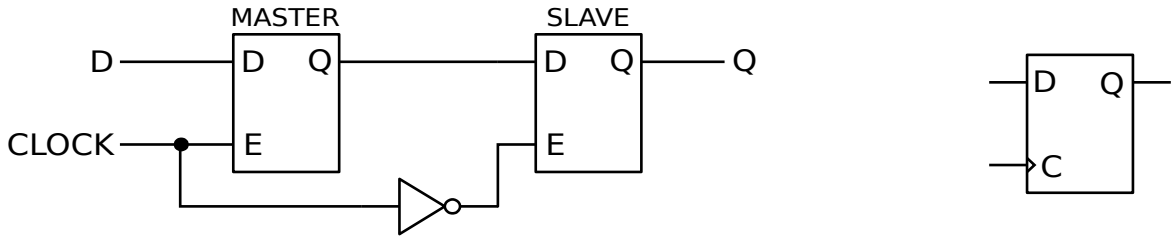
You will see that when the enable is set to 1, the input data bit is written to the first latch, then the second latch immediately updates itself as well. If we have a chain of two or more latches, the same data bit immediately gets written to all the latches when they are enabled. We can only store a single bit of information to a chain of latches, as the outputs of both latches will always the same.

Wouldn't it be better if we could store a different bit to each latch in a chain, meaning we can store more information in the chain? This the behaviour we want in a computer. Rather than rushing through the entire chain in one go, we want the bits in the circuit to “move” only one step down the chain each time the memory cells in the chain are enabled. We learn how to do this in the next section.

4/9/2018 Lesson 11: (Term 3, Week 7)

The Register (Master-Slave flip-flop)

We make a register, or master-slave flip-flop by using two latches in series as below:



The first (master) latch is enabled directly by the clock signal, the other (slave) latch by the inverted clock signal. This means only one of the latches is every enabled at at time: they are never enabled at the same time. This prevents the input going straight through to the output. Rather, the output only updates to the input when the clock *changes*.

When the clock goes high, the master latch is enabled and its output will be the same as its input. During this time, the slave latch is disabled, so its output will not change, but will remain in the earlier state. When the clock goes low, the master latch will be disabled and its output will be "frozen". At the same time, the slave latch will be enabled and the output of the register will be updated to the "frozen" value in the master, where it will remain because its input is not changing. The total effect is that the register's output will be stable and only update when the clock changes from high to low, the master-slave writing cycle being repeated each time the register updates.

The symbol for a register is shown on the right above. The truth table for a register is below.

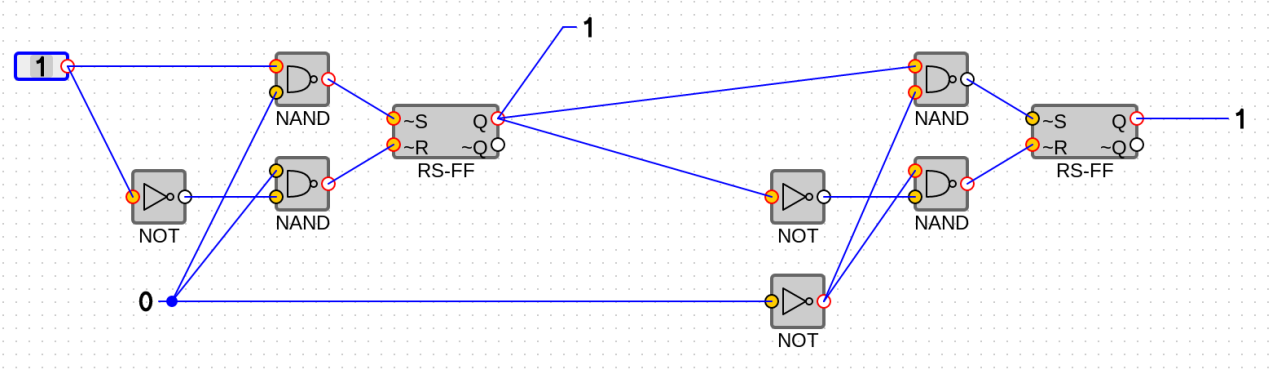
CLOCK	D	Q(t)
0	X	Q(t-1)
1	X	Q(t-1)
↓	0	0
↓	1	1

Note that we have a new symbol in the truth table: an arrow "↓". This arrow indicates that row of the truth table applies then the clock changes from 1 to 0 (high to low). The above truth table says that when the clock changes from 1 to 0, its data input will be stored to its output. When the clock is a stable 0 or 1, the output of the register does not change.

Make a register in simulator as below. We have added a display to the output of the master latch, so you can see the master-slave action in operation. Verify truth table by changing the input with the clock either low or high. You will see that when the data input is changed by itself, the output of the register does not change. The output only changes when the clock changes from 1 to 0, at which point the data input is stored to the output.

As a further exercise, set the clock to 1, then click on the data input multiple times, so it toggles between 1 and 0. You will see that the output of the master latch toggles, but the output of the slave

latch (which is also the output of the register) does not change. This is the master-slave operation in action. When you change the clock to 0, the output of the slave latch will be updated. You will then see that the output of the master latch does not change when the data input is toggled.

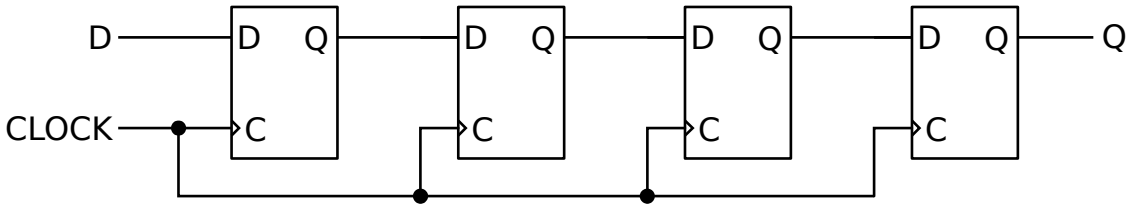


This register is a commonly used component in a computer, so it has its own component in the toolbar of the simulator, called a D-FF (short for D Flip-Flop). Place a D-FF component and verify that its behaviour is the same at the register that you built yourself.

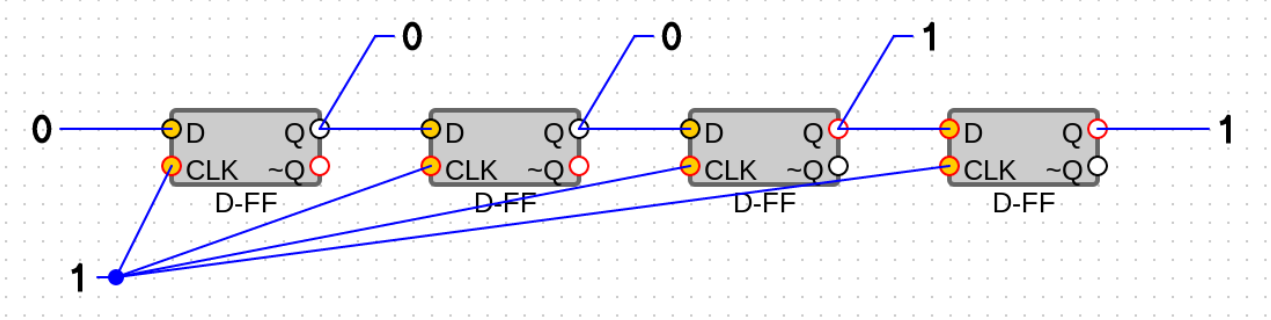
Have you ever heard people talking about the speed of a computer clock? For example, a computer's processor may be described as a 3GHz processor. The “clock” signal being referred to is the clock signal on the register we have just built. It is a measure of how many times per second a computer can store information. In turn, this determines how many things (or operations) a computer can do each second. A 3GHz clock means the above register could store a new information bit 3 billion times per second. That is 3,000,000,000 per second! That is a lot of bits. Computers are simple, but they are fast!

Shift Registers

Now we can make a chain of registers, and compare their behaviour to our previous chain of latches. We have shown a chain of 4 registers below.



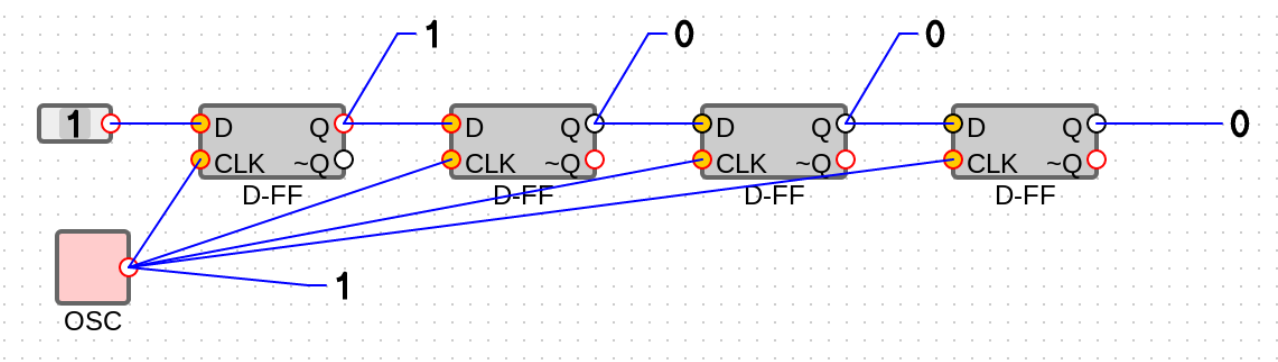
Make a chain of four D-FF in the simulator as below.



Set the data (D) input to 0 and click on the clock input a few times to toggle it between 1 and 0. You will see that the bits all “shift” one register to to the right each time the clock changes from 1 to

0. Once all the registers are set to 0, change the data input to 1 and toggle the clock again until all the registers are set to 1. Unlike the latch, the input bit doesn't just go though the entire chain, but it goes one step at a time. This arrangement of registers is called a **shift register**, because of the shifting (or moving) of bit that you have just observed.

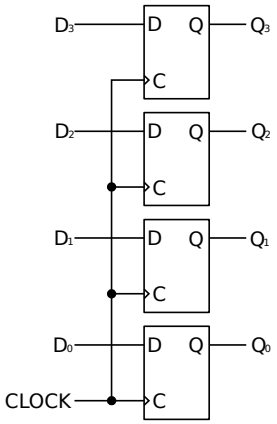
As a further exercise, replace the manual clock input with an oscillator (OSC) component as below.



The output of an oscillator automatically switches between 0 and 1 and back again, in an endless cycle. Now you can watch the shift register work without having to keep clicking on the clock input. Click a few times on the data input and watch the changing data bit propagate down the shift register. The clock in a computer is normally connected to such an oscillator. Our simulated oscillator turns on and off once each second so we can see our circuits in action, but an actual computer's oscillator turns on and off billions of times each second.

Multi-bit Registers

A register can store a 1 or a 0. How do we store a number which has multiple bits? We just use more than one register in parallel, to make a multi-bit register. It looks a bit like a shift register, with all the clocks connected together, but we don't arrange the inputs and outputs registers in a chain. For example, the following register can store a 4-bit binary number, which can have values from 0 to 15.



As usual, try it out in the simulator: