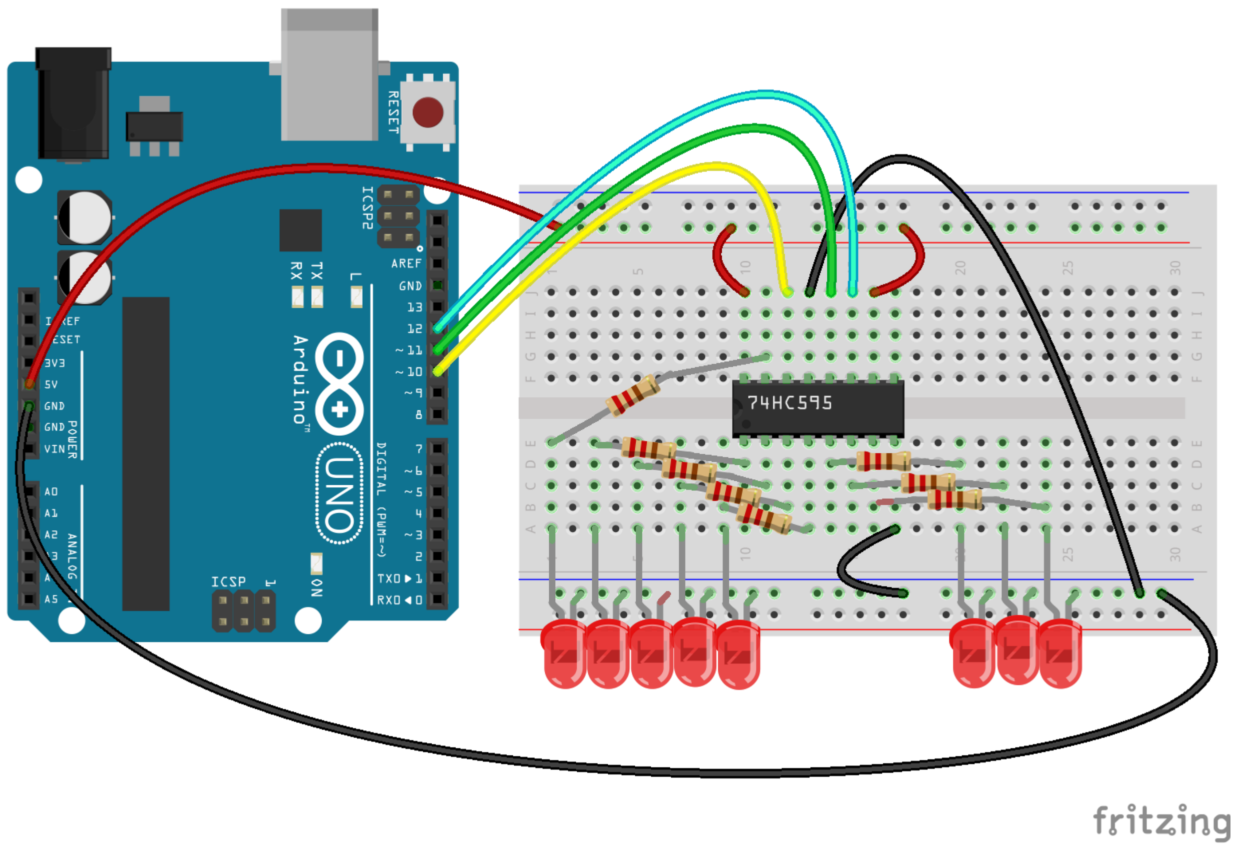
Shift Registers

A shift register is an example of a external **logic IC** (integrated circuit) that is often useful in Arduino project for helping us to increase the number of available digital outputs or inputs.

As you may have noticed in the previous example, we can quite quickly use up pins on the Arduino board, and we only have a limited number of them (19 in total). Therefore a lot of projects use tricks to increase this capability. The one we'll look at is an “output shift register” (74HC595)

Lets build up the following breadboard



Make sure that the notch on the shift register case (which marks the pin one end) is to the left side.

Load up sketch **ShiftRegister1**. You should see each LED is lit in sequence. So how does this work?

If we don't count the power leads, our shift register needs 3 of the Arduino pins. These are called

- The **data** line (connected to pin 10)
- This **shift clock** line (connected to pin 11)
- This **store clock** line (connected to pin 12)

Try not to get too scared by the terminology here; Let me tell you how the shift register works...

A shift register is rather like a clear tube containing coloured balls. If we want to display a specific series of colours (like black-black-white-white etc.) we can build this up in the tube by pushing the colours in from one end, one at a time.

Another approach to building a sequence of coloured balls might be to use a trough instead of a tube and place in all the balls at the same time using several pairs of hands. We would call this loading in **parallel**

Building the sequence up one ball at a time is slower than the parallel load, but it has the advantage we only need to load one colour at a time (If we only had one hand to do the work we could still build a sequence this way). We call this loading in **series**, or a “serial load”

The 74HC595 shift register is rather like this tube of black and white balls.

The **data line** decides the colour of the next ball waiting outside the end of the tube - lets say HIGH for white and LOW for blank. However that is not enough we also need a way to tell the shift register it is time to push in the next colour and shift all the others along. This is the function of the **shift clock**. We tell the shift register to “shift” **when the shift clock line changes from a LOW level to a HIGH level.**

If you think about it, this makes perfect sense - just saying “a HIGH level means shift” is not meaningful enough. If we want to shift twice in a row what can we do other than go back to LOW in between the two HIGHs? So we have to define the signal in terms of a change in the level. There is some common terminology that is useful to know

- A signal that tells a component to “do the next thing” is usually termed a **clock**. Sometimes a clock may happen at a regular frequency that could be timed in real seconds (like a GigaHertz clock in a desktop computer) but the term “clock” is used more generally in electronics and does not have to be about regular timing, or seconds, minutes etc. When we send such a signal to a component we say we are “clocking” it.
- A clock signal that says “do something” when the signal changes from LOW to HIGH is called a **rising edge** clock. If someone talks about something happening on the “rising edge” of some voltage signal, they mean it goes from LOW to HIGH. This is useful to know when you read data sheets for components.
- Many components actually “do something” when a clock signal change from HIGH to LOW. This is called a **falling edge**.

Well you may notice we actually have two clock lines; we also have a **store clock**. Well a useful feature of the 74HC595 shift register is that the outputs do not directly show what is being shifted. Instead we need to separately tell the component to load the contents of the shifting data into the output pins.

Going back to our analogy, this is rather like having two troughs of coloured balls where one of them is visible behind a window and the other is out of view. We can shift in colours to the out of view one using our data and shift clock lines. When we are ready to show what we've got, we signal the store clock. This causes the shifted set of balls to fall into the trough where they can now be seen at the outputs and will stay there even as we load up a new sequence in the first trough.

This is immensely useful - it prevents confusing changing outputs as the next row of data is shifted in behind the scenes. We can separately decide when to update the outputs.

Back to the example; Here we start by setting the data line HIGH and pulsing the shift clock then the store clock. This lets us see the HIGH state we've shifted in. It shows at the first LED position.

Now we set the data line LOW (we only want one LED to be lit) and we pulse the shift and store clocks seven more times, causing the HIGH level to shift along the outputs, with new LOW levels coming in behind. The result is rather like our original LED Scanner project.

Q: Hey - wasn't it much easier to just connect the LEDs directly to output pins?

For this simple example, yes. However you can see that instead of using 8 output pins for 8 LEDs, now we are using just 3.

Even better, we can chain shift registers together - when a value is shifted “off the end” of the register it appears at a special output data pin of the shift register. We can connect this to the data pin of another shift register and let them share the clock lines. Now we can run 16 LEDs with the same 3 output pins on the Arduino.

We can actually go on adding more shift registers this way forever and only need the same 3 pins! The main problem is that serial loading becomes slow, the more data we need to load. However these things are *fast*, so we can get quite a long way without a problem!

Q: In this example we are always signalling the shift and store clock lines together. Can't we just wire them together and save an output pin?

Yes - in simple applications this is a useful and valid technique. The only thing to know is that the store completes before the shift so you will find that an extra clock pulse is needed to show the first bit of data loaded.

Q: What are the other connections on the IC for?

First of all, the shift register needs to be powered. It is an electrical device after all. The connections on pins 8 and 16 are simply power. (IC pins are numbered according to the following standard; With the notch on the placed at the top, pin 1 is top left and other pins are numbers anticlockwise down the left side and back up the right)

The 74HC595 shift register has a couple of pins that do special functions. First of all, pin 10 clears out the shift register content when it has a LOW level. We want to prevent this, so we connect it to a HIGH level. Secondly pin 13 turns off all the outputs when it is at a HIGH level. We want to prevent this too, so we hold it LOW.

The unconnected pin 9 is where the last data bit “falls out”. If we connect an LED to this we have a ninth output. We can also use it to provide input to another shift register if we are chaining the together.

Q: What if I want to shift in the opposite direction?

Shift registers only shift in one direction. However, think about it - because we have separate control over shift and store we can still do this. For example we can shift a HIGH value into the eighth position using 7 shift clocks and giving LLLLLLLH. Then we can signal the store clock. Now we can shift in a sequence LLLLLLHL and store that. From the outputs the HIGH level has shifted one position to the left, even though the shift register can only shift to the right. While it may seem like a lot of work you have to remember these things can go blisteringly fast. It is possible to perform *millions* of shifts per second. Load sketch **ShiftRegister2** to see a two-way LED scanner example.

Shift registers are very useful. In the Arduino world they are often used to expand the number of outputs you have available. More generally this type of shift register is often used for **converting serial data (stream of bits) into parallel data (bytes of data)** in data communications.

The 74HC595 is called a serial-in-parallel-out shift register (SIPO). There are also parallel-input-serial-output (PISO) shift registers which have a set of inputs rather than outputs. They grab the input data and you can shift it out one bit at a time to read it. These are useful for expanding the number of inputs you have available (for example reading a keypad)

Shift registers are just one possibility for expanding input and output capabilities. Some other examples are

- Digital and Analog Multiplexer chips - these let you select directly between a set of inputs or outputs by “dialling up” the one you want.
- Switch matrixes (A technique of arranging switches in a grid and set each column HIGH one at a time and read back from the rows to see which switch is pressed)
- LED matrix (similar technique for driving lots of LEDs arranged in rows and columns)
- Special controller ICs (it is possible to get special IC's that scan a switch matrix or LED matrix for you, your Arduino just tells the IC what to do - or receives updates from it - letting you concentrate on other things!)