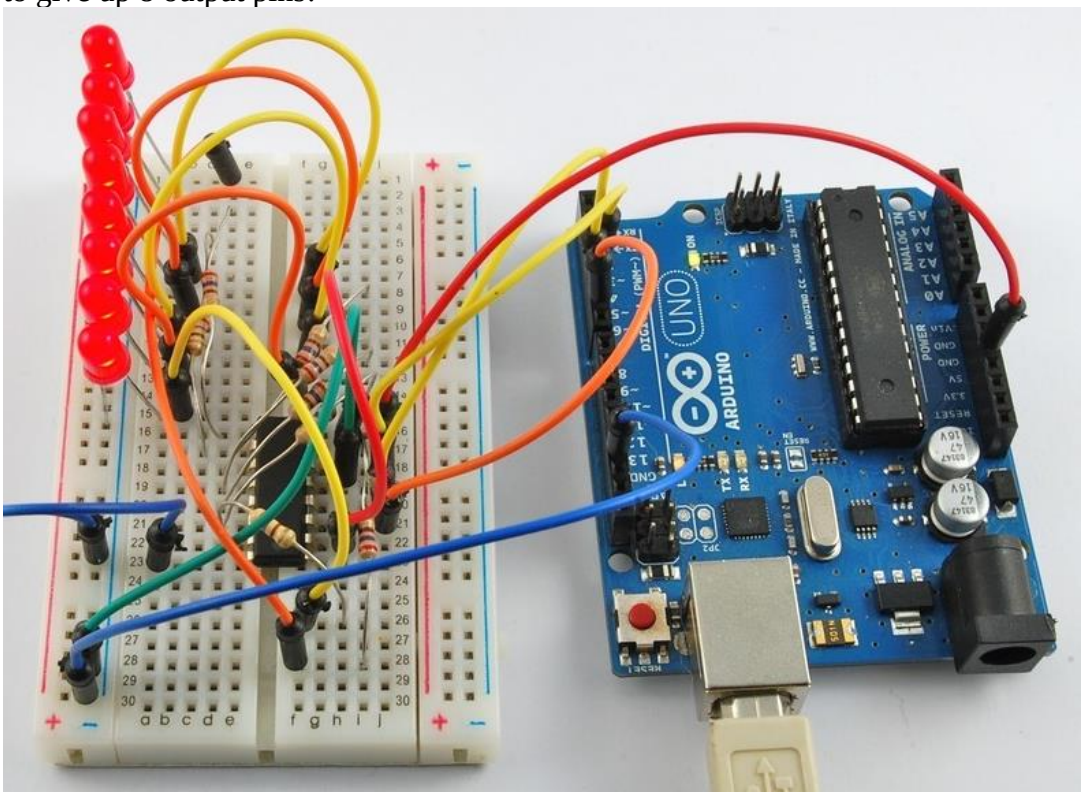
Eight LEDs and Shift Register

Overview

In this lesson, you will learn how to use eight large red LEDs with an Arduino without needing to give up 8 output pins!

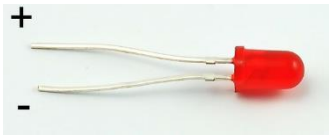


Although you could wire up eight LEDs each with a resistor to an Arduino pin (like we did for an RGB LED in Lesson 2) you would rapidly start to run out of pins on your Arduino. If you don't have a lot of stuff connected to your 'duino it's OK to do so - but often times we want buttons, sensors, servos, etc and before you know it you've got no pins left. So, instead of doing that, you are going to use a chip called the **74HC595 Serial to Parallel Converter**. This chip has eight outputs (perfect) and three inputs that you use to feed data into it a bit at a time.

This chip makes it a little slower to drive the LEDs (you can only change the LEDs about 500,000 times a second instead of 8,000,000 a second) but it's still really really fast, way faster than humans can detect, so it's worth it!

Parts

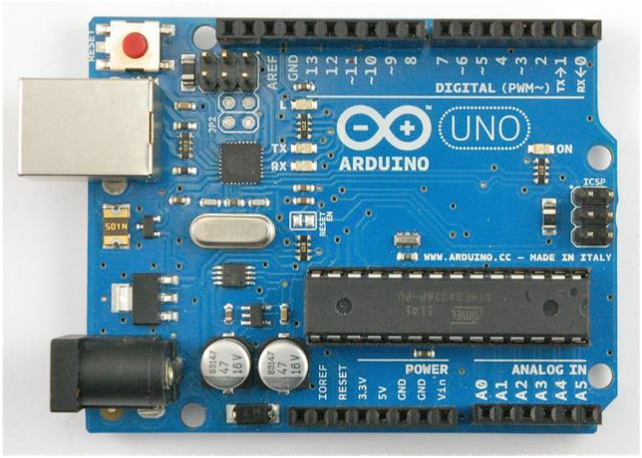
To build the project described in this lesson, you will need the following parts.



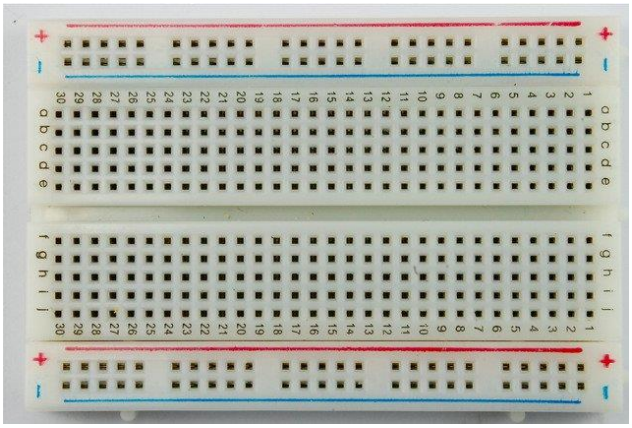
5 mm Red LED - 8

1 220 ohm resistor (red, red, black, black, brown) - 1

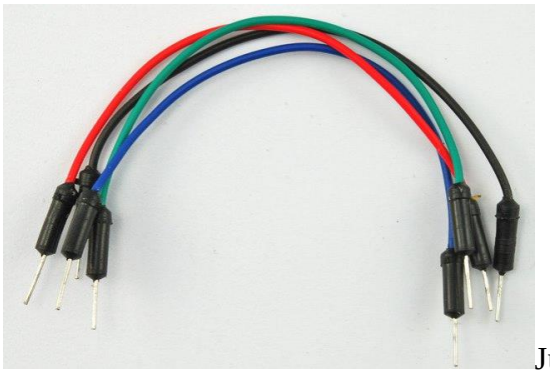
74HC595 Shift Register - 1



Arduino UNO R3 - 1



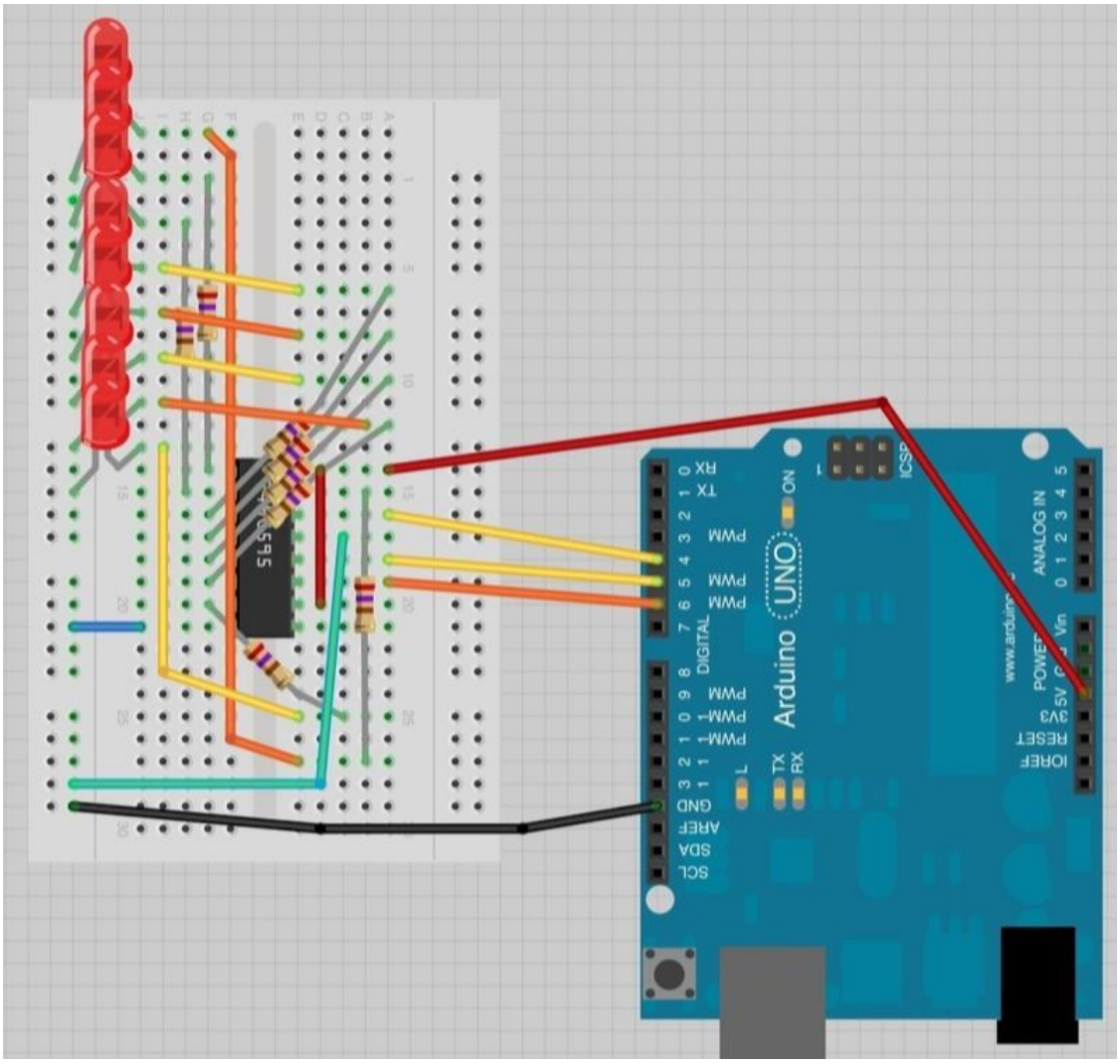
breadboard - 1



Jumper wires - 12

Breadboard Layout

As we have eight LEDs and eight resistors to connect up, there are actually quite a few connections to be made.



It is probably easiest to put the 74HC595 chip in first, as pretty much everything else connects to it. Put it so that the little U-shaped notch is towards the top of the breadboard. Pin 1 of the chip is to the left of this notch.

- Digital 4 from the arduino goes to pin #14 of the shift register
- Digital 5 from the arduino goes to pin #12 of the shift register
- Digital 6 from the arduino goes to pin #11 of the shift register

All but one of the outputs from the '595 are on the left hand side of the chip, hence, for ease of connection, that is where the LEDs are too.

After the chip, put the resistors in place. You need to be careful that none of the leads of the resistors are touching each other. You should check this again, before you connect the power to your Arduino. If you find it difficult to arrange the resistors without their leads touching, then it helps to shorten the leads so that they are lying closer to the surface of the breadboard.Next, place the LEDs on the breadboard.

The longer positive LED leads must all be towards the chip, whichever side of the breadboard they are on.

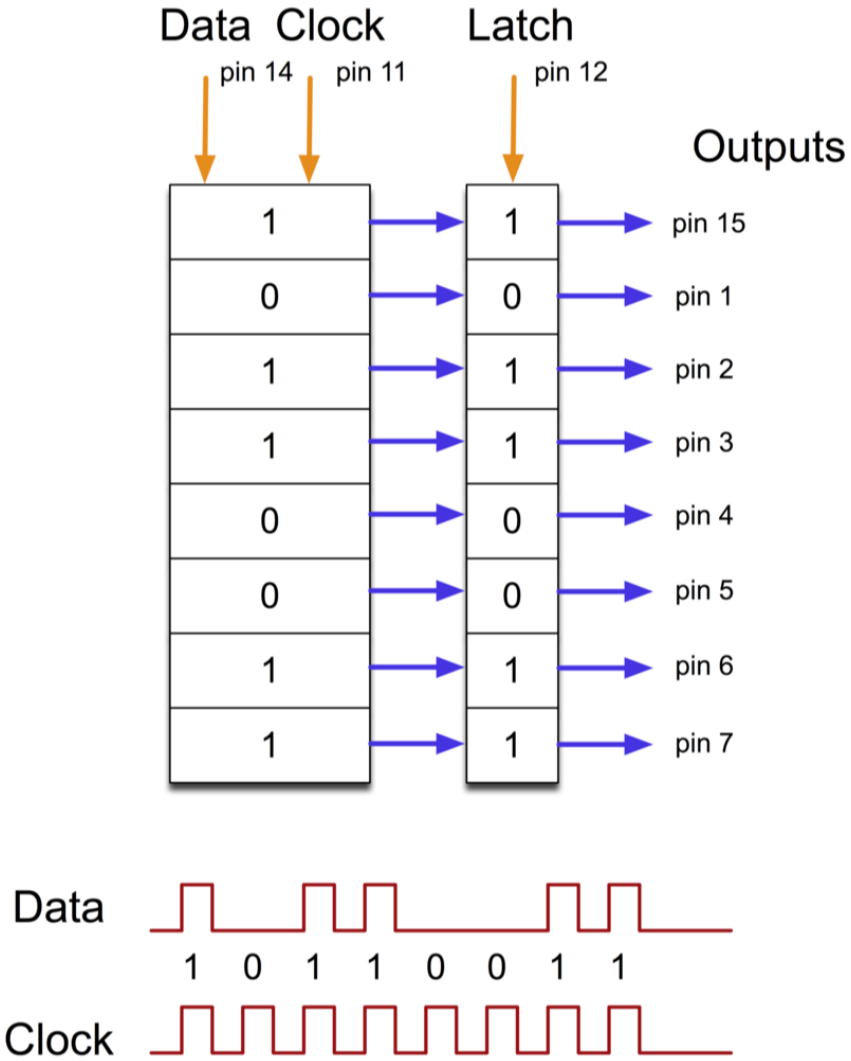
It now just remains to attach the jumper leads as shown above. Do not forget the one that goes from pin 8 of the IC to the GND column of the breadboard.

Load up the sketch listed a bit later and try it out. Each LED should light in turn until all the LEDs are on, and then they all go off and the cycle repeats.

The 74HC595 Shift Register

Before I go through the code, let's have a quick look at what the chip is doing, so that we can understand what the code has to do.

The chip is of a type called a shift register.



The shift register holds what can be thought of as eight memory locations, each of which can be a 1 or a 0.

To set each of these values on or off, we feed in the data using the 'Data' and 'Clock' pins of the chip.

The clock pin needs to receive eight pulses. At the time of each pulse, if the data pin is high, then a 1 gets pushed into the shift register. Otherwise, it is a 0. When all eight pulses have been received, then enabling the 'Latch' pin copies those eight values to the latch register. This is necessary, otherwise the wrong LEDs would flicker as the data was being loaded into the shift register.

The chip also has an OE (output enable) pin, this is used to enable or disable the outputs all at once. You could attach this to a PWM capable Arduino pin and use 'analogWrite' to control the brightness of the LEDs. This pin is active low, so we tie it to GND.

Arduino Sketch

Get a copy of the sketch from your teacher. Copy this code into the Arduino program.
Arduino includes a special function called 'shiftOut' that is designed specifically for sending data to shift registers.
The first thing we do is define the three pins we are going to use. These are the Arduino digital outputs that will be connected to the latch, clock and data pins of the 74HC595.
Next, a variable called 'leds' is defined. This will be used to hold the pattern of which LEDs are currently turned on or off. Data of type 'byte' represents numbers using eight bits. Each bit can be either on or off, so this is perfect for keeping track of which of our eight LEDs are on or off.
The 'setup' function just sets the three pins we are using to be digital outputs.
The 'loop' function initially turns all the LEDs off, by giving the variable 'leds' the value 0. It then calls 'updateShiftRegister' that will send the 'leds' pattern to the shift register so that all the LEDs turn off. We will deal with how 'updateShiftRegister' works later.

The loop function pauses for half a second and then begins to count from 0 to 7 using the 'for' loop and the variable 'i'. Each time, it uses the Arduino function 'bitSet' to set the bit that controls that LED in the variable 'leds'. It then also calls 'updateShiftRegister' so that the leds update to reflect what is in the variable 'leds'.

There is then a half second delay before 'i' is incremented and the next LED is lit.
The function 'updateShiftRegister', first of all sets the latchPin to low, then calls the Arduino function 'shiftOut' before putting the 'latchPin' high again. This takes four parameters, the first two are the pins to use for Data and Clock respectively.

The third parameter specifies which end of the data you want to start at. We are going to start with the right most bit, which is referred to as the 'Least Significant Bit' (LSB).

The last parameter is the actual data to be shifted into the shift register, which in this case is 'leds'.
f you wanted to turn one of the LEDs off rather than on, you would call a similar Arduino function (bitClear) on the 'leds' variable. This will set that bit of 'leds' to be 0 and you would then just need to follow it with a call to 'updateShiftRegister' to update the actual LEDs.

Brightness Control

One pin of the 74HC595 that I have not mentioned is a pin called 'Output Enable'. This is pin 13 and on the breadboard, it is permanently connected to Ground. This pin acts as a switch, that can enable or disable the outputs - the only thing to watch for is it is 'active low' (connect to ground to enable). So, if it is connected to 5V, all the outputs go off. Whereas if it is connected to Ground, those outputs that are supposed to be on are on and those that should be off are off.

We can use this pin along with the 'analogWrite' function, that we used back in Lesson 3, to control the brightness of the LEDs using PWM (also see Lesson 3).

To do this, all you need to do, is to change the connection to pin 13 of the 74HC595 so that instead of connecting it to Ground, you connect it to pin 3 of the Arduino.

Get a copy of the sketch from your teacher that will, once all the LEDs have been lit, gradually fade them back to off.