PART-2

<u>Basics of stepper motor</u>:

Basic construction

As their name implies, stepper motors move in discrete steps, known as the step angle, which usually ranges from 90 degrees (360° / 90° per step = 4 steps per revolution) to 0.75 degrees (360° /0.75° per step = 500 steps per revolution). Their basic construction consists of an outer stator and an inner rotor. The design of the rotor depends on the type of stepper motor being discussed, but the stator is similar between the various types of stepper motors, with uniform teeth around its perimeter and containing a specified number of poles. Poles are simply magnetic sections of the stator, and each pole has a winding that is connected to the pole opposite it on the stator. Thus, the opposing poles are magnetized with the opposite polarity when current is applied to the windings.

These pole pairs make up the winding phases, with most stepper motors being either 2-phase or 5-phase design.

There are three general stepper motor designs: variable reluctance, permanent magnet, and hybrid.
Points to be considered while using steppers are:

- Size
- Torque rating
- Step count
- Gearing
- Shaft style
- Wiring
- Coil and Phase
    1. Unipolar
    2. Bipolar

The 8-wire unipolar is the most versatile motor of all. It can be driven in several ways:

- **4-phase unipolar** - All the common wires are connected together - just like a 5-wire motor.
- **2-phase series bipolar** - The phases are connected in series - just like a 6-wire motor.
- **2-phase parallel bipolar** - The phases are connected in parallel. This results in half the resistance and inductance - but requires twice the

current to drive. The advantage of this wiring is higher torque and top speed.

https://youtu.be/bngx2dKl5jU

Watch this video for deeper view on stepper motor.

Basic ICs for running motor:

L293D(Motor driver with low current flow)

L293D is a typical Motor driver or Motor Driver IC which allows DC motor to drive on either direction. L293D is a 16-pin IC which can control a set of two DC motors simultaneously in any direction. It means that you can control two DC motor with a single L293D IC. Dual H-bridge *Motor Driver integrated circuit* (*IC*).

Voltage specification:

VCC is the voltage that it needs for its own internal operation 5v; L293D will not use this voltage for driving the motor. For driving the motors it has a separate provision to provide motor supply VSS (V supply). L293d will use this to drive the motor. It means if you want to operate a motor at 9V then you need to provide a Supply of 9V across VSS Motor supply.

The maximum voltage for VSS motor supply is 36V. It can supply a max current of 600mA per channel.Since it can drive motors Up to 36v hence you can drive pretty big motors with this l293d.

VCC pin 16 is the voltage for its own internal Operation. The maximum voltage ranges from 5v and upto 36v.

Concept and working:

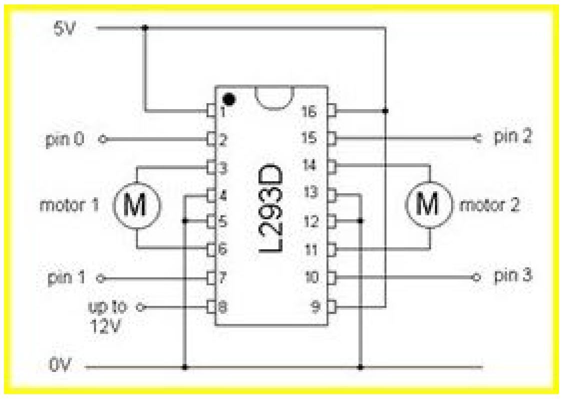
It works on the concept of H-bridge. H-bridge is a circuit which allows the voltage to be flown in either direction. As you know voltage need to change its direction for being able to rotate the motor in clockwise or anticlockwise direction, Hence H-bridge IC are ideal for driving a DC motor.

In a single L293D chip there are two h-Bridge circuit inside the IC which can rotate two dc motor independently.

There are two Enable pins on l293d. Pin 1 and pin 9, for being able to drive the motor, the pin 1 and 9 need to be high. For driving the motor with left

H-bridge you need to enable pin 1 to high. And for right H-Bridge you need to make the pin 9 to high.

Pin Diagram:



The output is controlled by Logic table as follow:

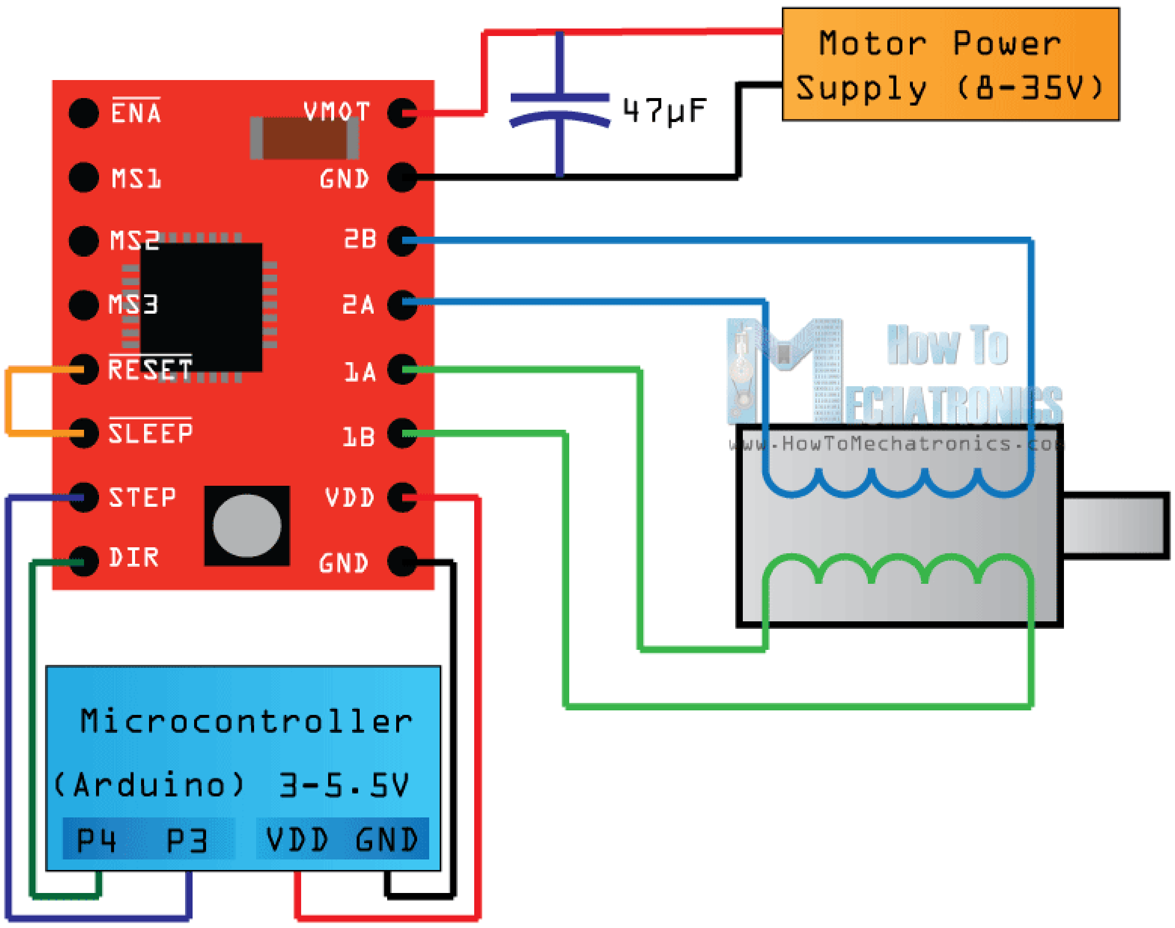
L293D Logic Table:

Let's consider a Motor connected on left side output pins (pin 3,6) and corresponding input in 2,7. For rotating the motor in clockwise direction the input pins has to be provided with Logic 1 and Logic 0.

- **Pin 2** = **Logic 1** and **Pin 7** = **Logic 0** | Clockwise Direction
- **Pin 2** = **Logic 0** and **Pin 7** = **Logic 1** | Anticlockwise Direction
- **Pin 2** = **Logic 0** and **Pin 7** = **Logic 0** | Idle [No rotation] [Hi-Impedance state]
- **Pin 2** = **Logic 1** and **Pin 7** = **Logic 1** | Idle [No rotation]

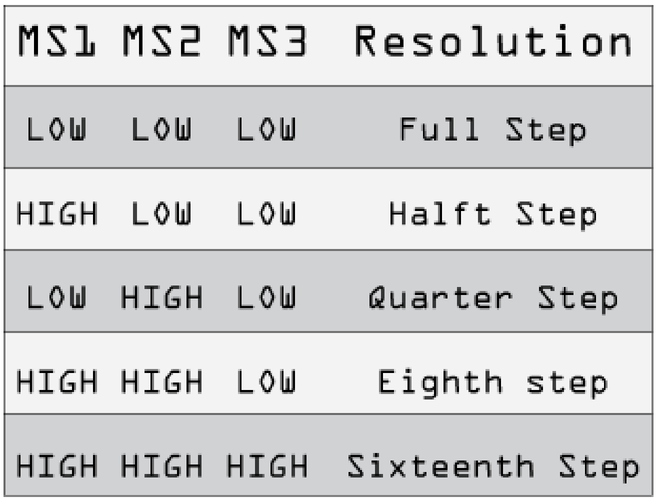
A4988(Stepper driver, Higher value of current flow)

- This breakout board for Allegro's A4988 microstepping bipolar stepper motor driver features adjustable current limiting, overcurrent and over-temperature protection, and five different microstep resolutions (down to 1/16-step). It operates from 8V to 35V and can deliver up to approximately 1 A per phase without a heat sink or forced air flow (it is rated for 2 A per coil with sufficient additional cooling).


Motor Power Supply (8-35V)
47µF
ENA
VMOT
MS1
GND
MS2
2B
MS3
2A
RESET
1A
SLEEP
1B
STEP
VDD
DIR
GND
How To Mechatronics
www.HowToMechatronics.com
Microcontroller (Arduino) 3-5.5V
P4
P3
VDD
GND

With the Step pin we control the mirosteps of the motor and with each pulse sent to this pin the motor moves one step. So that means that we don't need any complex programming, phase sequence tables, frequency control lines and so on, because the built-in translator of the A4988 Driver takes care of everything.
Next is the SLEEP Pin and a logic low puts the board in sleep mode for minimizing power consumption when the motor is not in use.

Next, the RESET pin sets the translator to a predefined Home state. This Home state or Home Microstep Position can be seen from these Figures from the A4988 Datasheet. So these are the initial positions from where the motor starts and they are different depending on the microstep resolution. If the input state to this pin is a logic low all the STEP inputs will be ignored. The Reset pin is a floating pin so if we don't have intention of controlling it with in our program we need to connect it to the SLEEP pin in order to bring it high and enable the board.



MS1	MS2	MS3	Resolution
LOW	LOW	LOW	Full Step
HIGH	LOW	LOW	Halft Step
LOW	HIGH	LOW	Quarter Step
HIGH	HIGH	LOW	Eighth step
HIGH	HIGH	HIGH	Sixteenth Step

The next 3 pins (MS1, MS2 and MS3) are for selecting one of the five step resolutions according to the above truth table. These pins have internal pull-down resistors so if we leave them disconnected, the board will operate in full step mode.

The last one, the ENABLE pin is used for turning on or off the FET outputs. So a logic high will keep the outputs disabled.

Basic loop code for running stepper:

```
digitalWrite(dirPin,HIGH); // Enables the motor to move in a particular direction
// Makes 200 pulses for making one full cycle rotation
 for(int x = 0; x < 200; x++) {
   digitalWrite(stepPin,HIGH);
   delayMicroseconds(500);
   digitalWrite(stepPin,LOW);
   delayMicroseconds(500);
```

Basic of servo motor:

A **servo motor** is an electrical device which can push or rotate an object with great precision. If you want to rotate and object at some specific angles or distance, then you use servo motor. It is just made up of simple motor which run through **servo mechanism**.

Servo Mechanism

It consists of three parts:

1. Controlled device
2. Output sensor
3. Feedback system

It is a closed loop system where it uses positive feedback system to control motion and final position of the shaft. Here the device is controlled by a feedback signal generated by comparing output signal and reference input signal.

Reference input signal is compared to reference output signal and the third signal is produced by feedback system. And this third signal acts as input signal to control device.

Working:

Servo motor works on **PWM (Pulse width modulation)** principle, means its angle of rotation is controlled by the duration of applied pulse to its Control PIN.

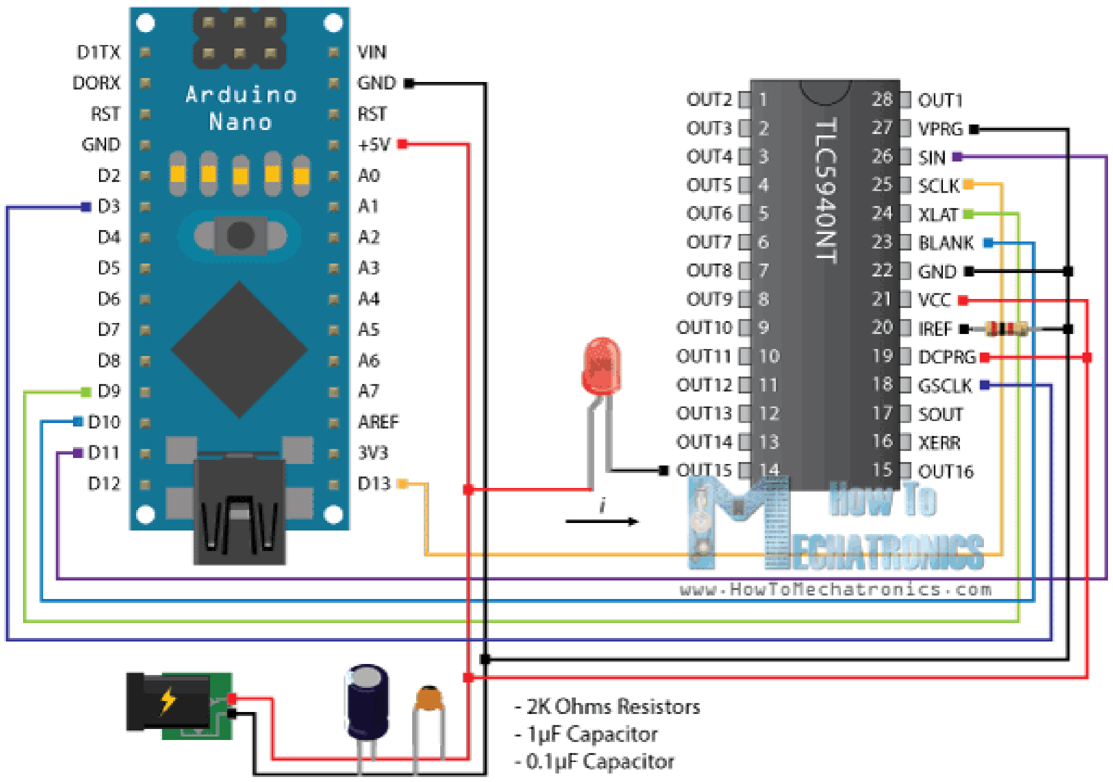
Basic code:

```
// Include the Servo library

#include <Servo.h>
// Declare the Servo pin
int servoPin = 3;
// Create a servo object
Servo Servo1;
void setup() {
  // We need to attach the servo to the used pin number
  Servo1.attach(servoPin);
}
void loop()

{
  // Make servo go to 0 degrees
  Servo1.write(0);
  delay(1000);
  // Make servo go to 90 degrees
  Servo1.write(90);
  delay(1000);
  // Make servo go to 180 degrees
  Servo1.write(180);
  delay(1000);
}
```

TLC5940(LED Driver/ To extend PWM pins of Arduino):

The TLC5940 is a 16-Channel LED Driver which provides PWM outputs and it’s perfect for extending the Arduino PWM capabilities. Not just LEDs, but with this IC we can also control servos, DC Motors and other electronics components using PWM signals.

For controlling the TLC5940 we need to occupy 4 pins of your Arduino Board. As we will use the TLC5940 Arduino Library made by Alex Leone we need to connect the IC to the Arduino according to his library configuration or using the following circuit schematics:



The circuit schematics above is using external power supply for powering the LEDs, but also it can be connected just using the Arduino VCC itself if the total amount of drawn current doesn't exceed the limit of the Arduino (Absolute Maximum Rating, DC Current VCC and GND Pins – 200 mA).

We also need to note that the TLC5940 is a constant-current sink, so the current flow towards the output pins. This means that when connecting LEDs we need to connect the negative lead (Cathode) to the output pin of the IC and the positive lead (Anode) to the 5V VCC.

Best part of using this IC is that we can use more than one IC in series and can extend the same 4 pins of arduino to 32/64 and so one PWM pins.

Basic code:

```
#include "Tlc5940.h"

void setup() {
 Tlc.init(0); // Initiates the TLC5940 and set all channels off
}

void loop() {
 Tlc.set(0,4095); //(Output Pin from 0 to 15,PWM Value from 0 to 4095)
// Note: The previous function doesn't activates the output right away.
The output will be activated when the Tlc.update() function will be
executed!
```

```

 Tlc.update(); // Activates the previously set outputs
 delay(1000);

// For activating all 16 outputs at the same time we can use a for loop for setting all of them to be set to PWM value of 4095. Then  the Tlc.updata() function will active them all at the same time.
 for (int i = 0; i < 16; i++) {
   Tlc.set(i, 4095);
 }
 Tlc.update();
 delay(1000);

//The Tlc.clear() function clears all the outputs, or sets the PWM value of all outputs to 0
 Tlc.clear();
 Tlc.update();
 delay(1000);

// This for loop will active all 16 LEDs one by one
 for (int i = 0; i < 16; i++) {
   Tlc.set(i, 4095);
   Tlc.update();
   delay(200);
   Tlc.clear();
   Tlc.update();
   delay(200);
 }
}
```

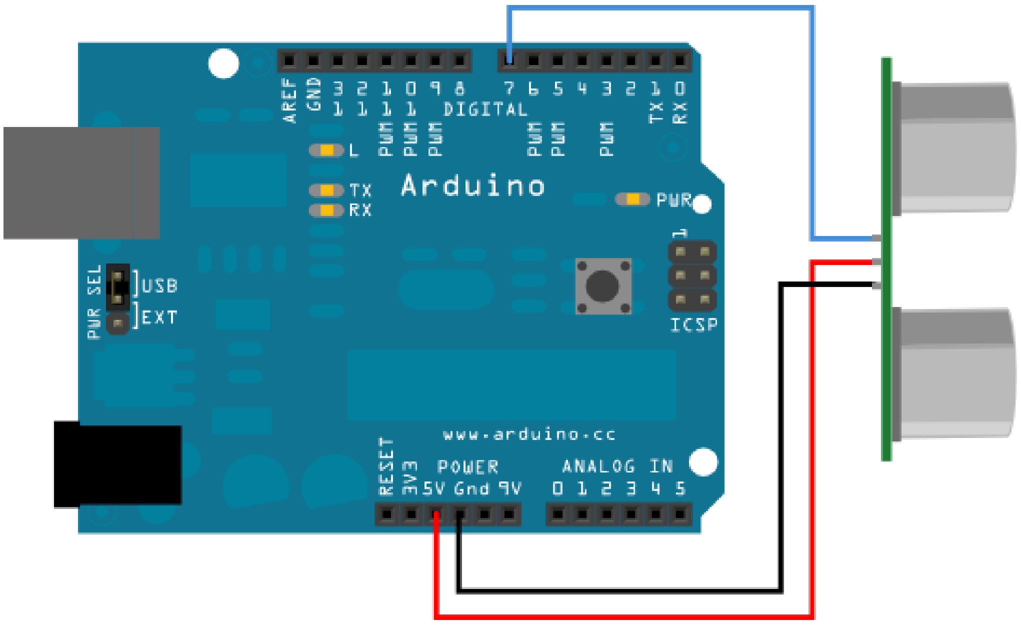
<u>Introducing sensor</u>:

Proximity sensor(Ping Sensor)

The SEN136B5B is an ultrasonic range finder from Seeedstudio. It detects the distance of the closest object in front of the sensor (from 3 cm up to 400 cm). It works by sending out a burst of ultrasound and listening for the echo when it bounces off of an object. It *pings* the obstacles with ultrasound. The Arduino or Genuino board sends a short pulse to trigger the detection, then listens for a pulse on the same pin using the pulseIn() function. The duration of this second pulse is equal to the time taken by the ultrasound to travel to the object and back to the sensor. Using the speed of sound, this time can be converted to distance.

Circuit:

The 5V pin of the SEN136B5B is connected to the 5V pin on the board, the GND pin is connected to the GND pin, and the SIG (signal) pin is connected to digital pin 7 on the board.



Basic code:

```
const int pingPin = 7;

void setup() {

  Serial.begin(9600);
}

void loop() {
  // establish variables for duration of the ping, and the distance result
  // in inches and centimeters:
  long duration, inches, cm;

  // The PING))) is triggered by a HIGH pulse of 2 or more microseconds.
  // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);

  // The same pin is used to read the signal from the PING))): a HIGH pulse
  // whose duration is the time (in microseconds) from the sending of the ping
  // to the reception of its echo off of an object.
  pinMode(pingPin, INPUT);
  duration = pulseIn(pingPin, HIGH);

  // convert the time into a distance
  inches = microsecondsToInches(duration);
  cm = microsecondsToCentimeters(duration);

  Serial.print(inches);
  Serial.print("in, ");
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();

  delay(100);
}

long microsecondsToInches(long microseconds) {
  // According to Parallax's datasheet for the PING))), there are 73.746
  // microseconds per inch (i.e. sound travels at 1130 feet per second).
  // This gives the distance travelled by the ping, outbound and return,
  // so we divide by 2 to get the distance of the obstacle.
  // See: http://www.parallax.com/dl/docs/prod/acc/28015-PING-v1.3.pdf
  return microseconds / 74 / 2;
}

long microsecondsToCentimeters(long microseconds) {
  // The speed of sound is 340 m/s or 29 microseconds per centimeter.
  // The ping travels out and back, so to find the distance of the object we
  // take half of the distance travelled.
  return microseconds / 29 / 2;
}
```

Shift Registers

Serial to Parallel Shifting-Out with a 74HC595

Shifting Out & the 595 chip

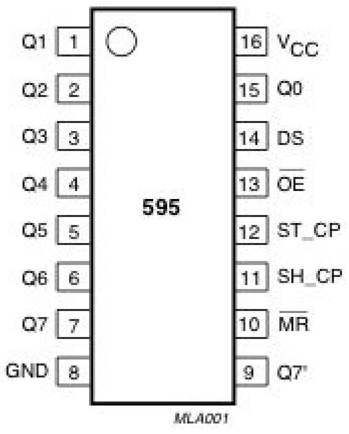
At sometime or another you may run out of pins on your Arduino board and need to extend it with shift registers. This example is based on the 74HC595. The datasheet refers to the 74HC595 as an "8-bit serial-in, serial or parallel-out shift register with output latches; 3-state." In other words, you can use it to control 8 outputs at a time while only taking up a few pins on your microcontroller. You can link multiple registers together to extend your output even more. (Users may also wish to search for other driver chips with "595" or "596" in their part numbers, there are many. The STP16C596 for example will drive 16 LED's and eliminates the series resistors with built-in constant current sources.)

How this all works is through something called "synchronous serial communication," i.e. you can pulse one pin up and down thereby communicating a data byte to the register bit by bit. It's by pulsing second pin, the clock pin, that you delineate between bits. This is in contrast to using the "asynchronous serial communication" of the Serial.begin() function which relies on the sender and the receiver to be set independently to an agreed upon specified data rate. Once the whole byte is transmitted to the register the HIGH or LOW messages held in each bit get parceled out to each of the individual output pins. This is the "parallel output" part, having all the pins do what you want them to do all at once.

The "serial output" part of this component comes from its extra pin which can pass the serial information received from the microcontroller out again unchanged. This means you can transmit 16 bits in a row (2 bytes) and the first 8 will flow through the first register into the second register and be expressed there. You can learn to do that from the second example.

"3 states" refers to the fact that you can set the output pins as either high, low or "high impedance." Unlike the HIGH and LOW states, you can"t set pins to their high impedance state individually. You can only set the whole chip together. This is a pretty specialized thing to do -- Think of an LED array that might need to be controlled by completely different microcontrollers depending on a specific mode setting built into your project. Neither example takes advantage of this feature and you won’t usually need to worry about getting a chip that has it.

Here is a table explaining the pin-outs adapted from the Phillip's datasheet.

 Q1 Q2 Q3 Q4 Q5 Q6 Q7 GND 595 VCC Q0 DS OE ST_CP SH_CP MR Q7" MLA001	PINS 1-7, 15	Q0 " Q7	Output Pins
	PIN 8	GND	Ground, Vss
	PIN 9	Q7"	Serial Out
	PIN 10	MR	Master Reclear, active low
	PIN 11	SH_CP	Shift register clock pin
	PIN 12	ST_CP	Storage register clock pin (latch pin)
	PIN 13	OE	Output enable, active low
	PIN 14	DS	Serial data input
	PIN 16	Vcc	Positive supply voltage

Example 1: One Shift Register

The first step is to extend your Arduino with one shift register.

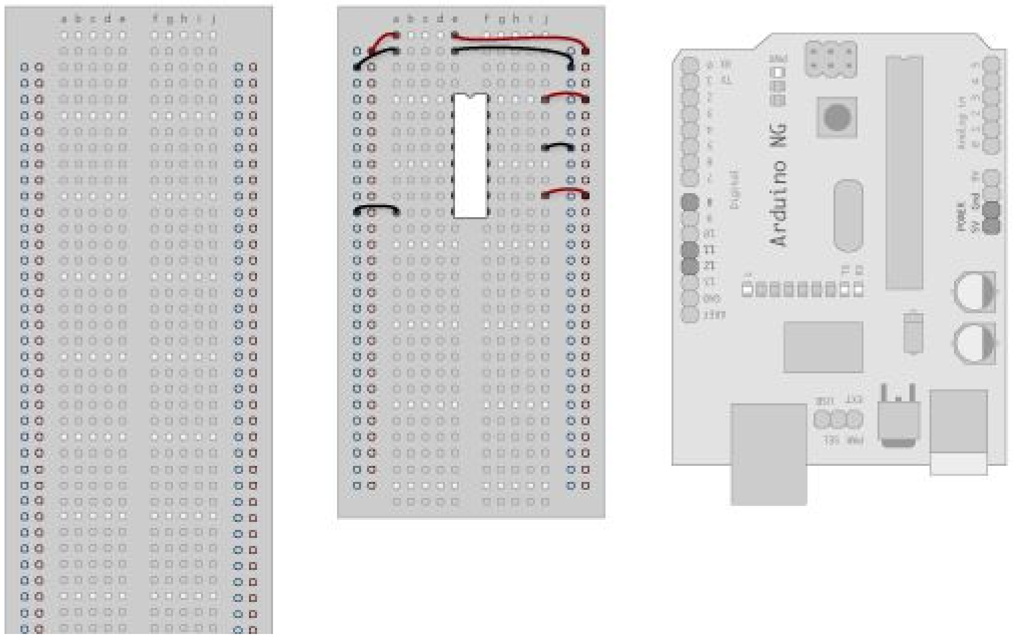
The Circuit

1. Turning it on

Make the following connections:

GND (pin 8) to ground,
Vcc (pin 16) to 5V
OE (pin 13) to ground
MR (pin 10) to 5V

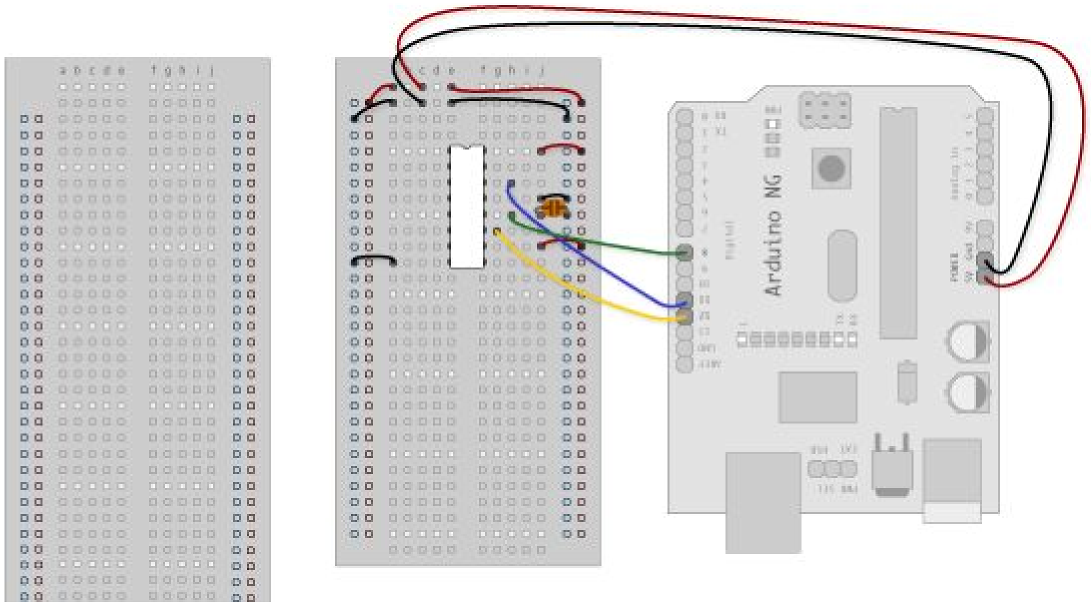
This set up makes all of the output pins active and addressable all the time. The one flaw of this set up is that you end up with the lights turning on to their last state or something arbitrary every time you first power up the circuit before the program starts to run. You can get around this by controlling the MR and OE pins from your Arduino board too, but this way will work and leave you with more open pins.



2. Connect to Arduino

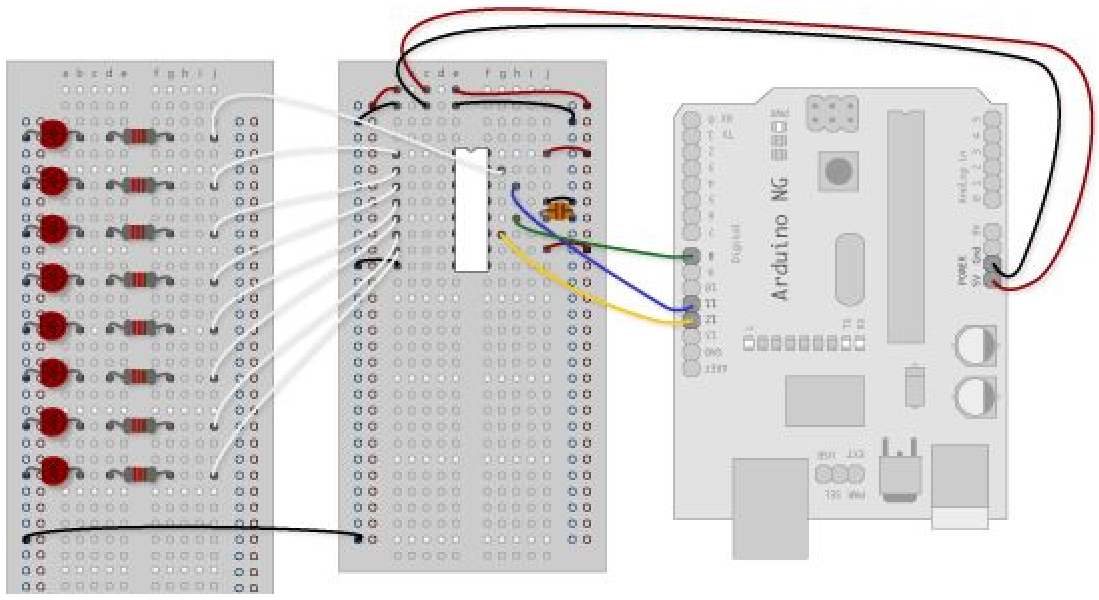
DS (pin 14) to Ardunio DigitalPin 11 (blue wire)
SH_CP (pin 11) to to Ardunio DigitalPin 12 (yellow wire)
ST_CP (pin 12) to Ardunio DigitalPin 8 (green wire)

From now on those will be refered to as the dataPin, the clockPin and the latchPin respectively. Notice the 0.1"f capacitor on the latchPin, if you have some flicker when the latch pin pulses you can use a capacitor to even it out.

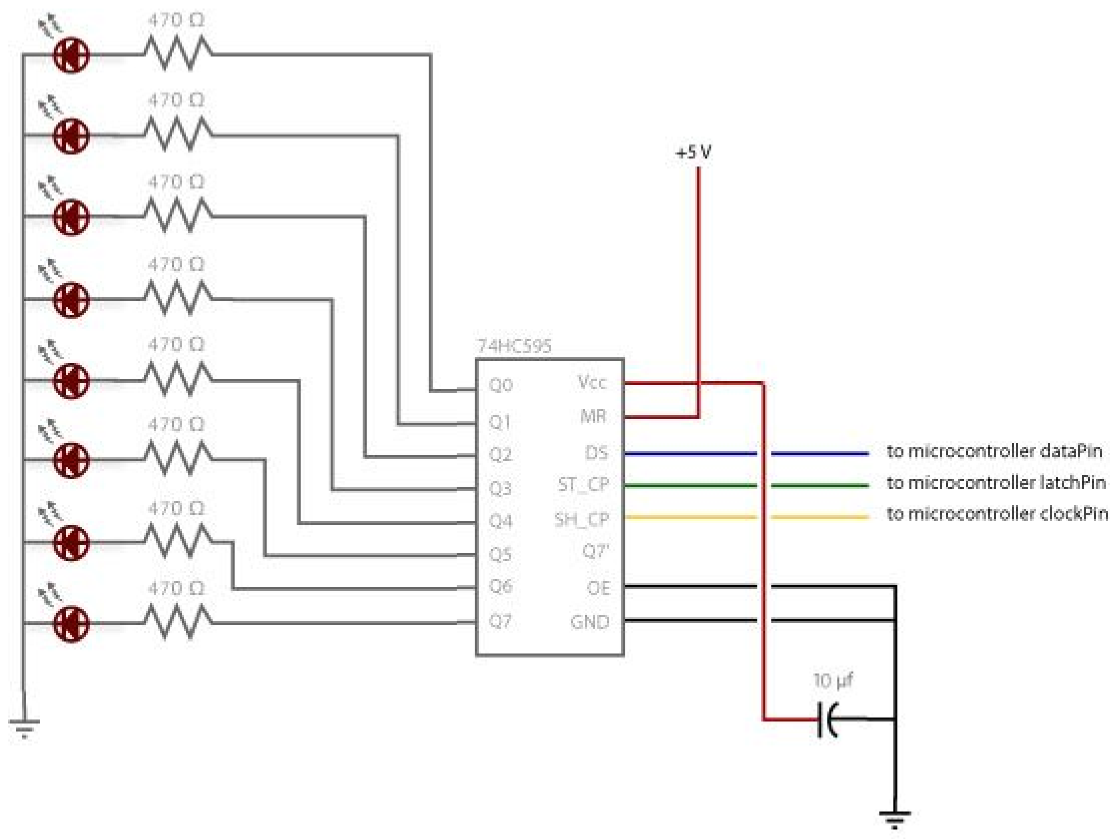


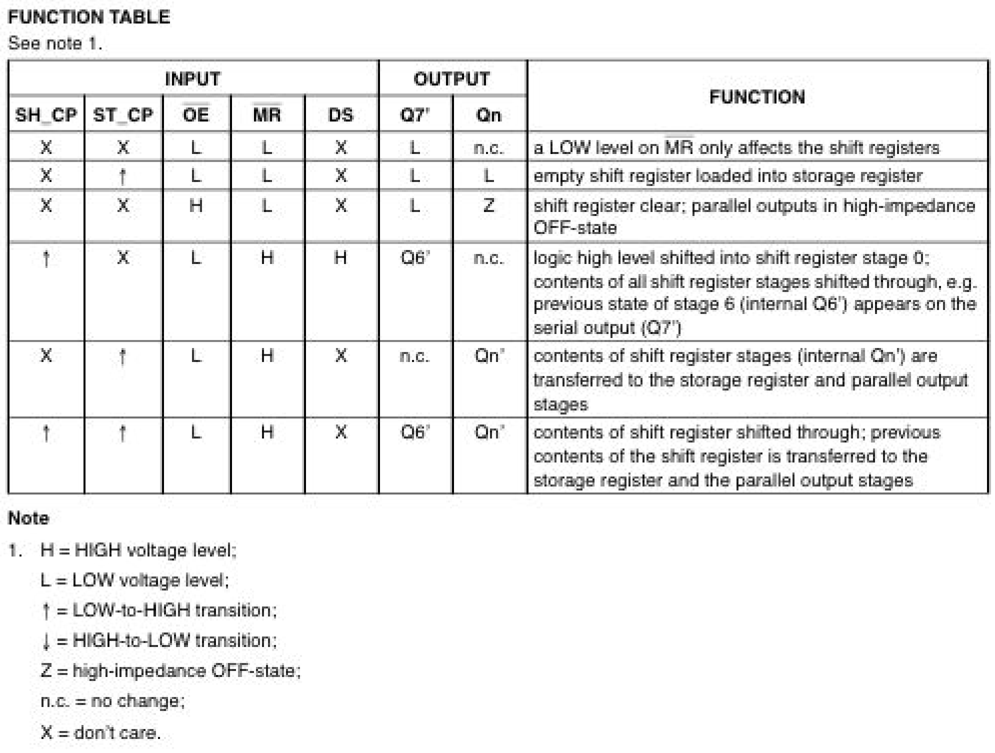
3. Add 8 LEDs.

In this case you should connect the cathode (short pin) of each LED to a common ground, and the anode (long pin) of each LED to its respective shift register output pin. Using the shift register to supply power like this is called sourcing current. Some shift registers can't source current, they can only do what is called sinking current. If you have one of those it means you will have to flip the direction of the LEDs, putting the anodes directly to power and the cathodes (ground pins) to the shift register outputs. You should check the your specific datasheet if you aren't using a 595 series chip. Don't forget to add a 470-ohm resistor in series to protect the LEDs from being overloaded.



Circuit Diagram


470 Ω
+5 V
74HC595
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Vcc
MR
DS
ST_CP
SH_CP
Q7'
OE
GND
to microcontroller dataPin
to microcontroller latchPin
to microcontroller clockPin
10 µf



FUNCTION TABLE

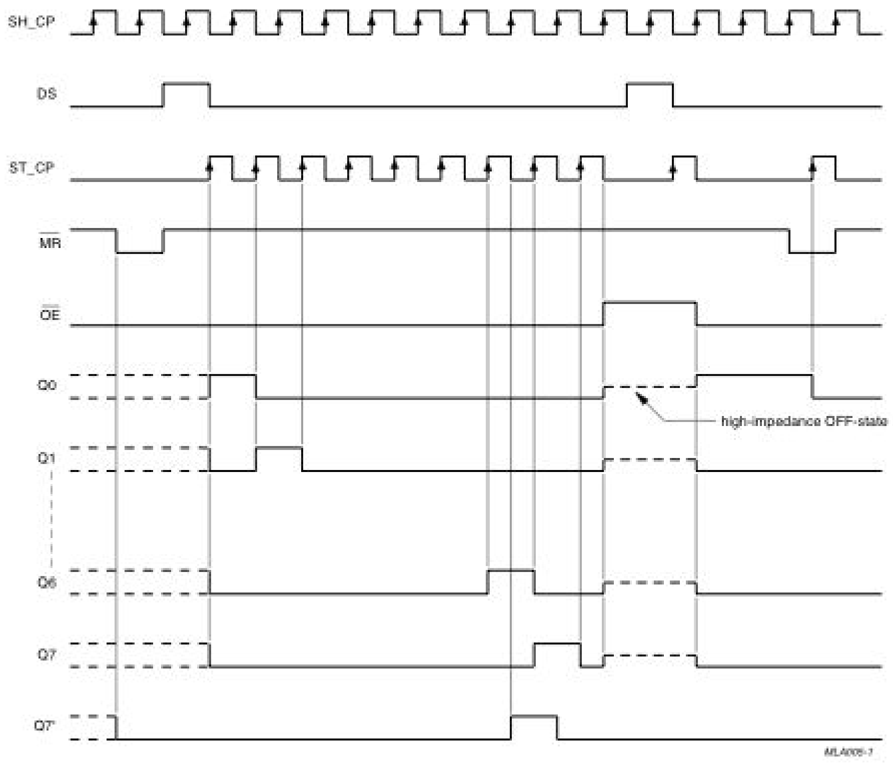
See note 1.

INPUT					OUTPUT		FUNCTION
SH_CP	ST_CP	$\overline{OE}$	$\overline{MR}$	DS	Q7'	Qn	
X	X	L	L	X	L	n.c.	a LOW level on $\overline{MR}$ only affects the shift registers
X	↑	L	L	X	L	L	empty shift register loaded into storage register
X	X	H	L	X	L	Z	shift register clear; parallel outputs in high-impedance OFF-state
↑	X	L	H	H	Q6'	n.c.	logic high level shifted into shift register stage 0; contents of all shift register stages shifted through, e.g. previous state of stage 6 (internal Q6') appears on the serial output (Q7')
X	↑	L	H	X	n.c.	Qn'	contents of shift register stages (internal Qn') are transferred to the storage register and parallel output stages
↑	↑	L	H	X	Q6'	Qn'	contents of shift register shifted through; previous contents of the shift register is transferred to the storage register and the parallel output stages

Note

1. H = HIGH voltage level;
   L = LOW voltage level;
   ↑ = LOW-to-HIGH transition;
   ↓ = HIGH-to-LOW transition;
   Z = high-impedance OFF-state;
   n.c. = no change;
   X = don't care.

595 Logic Table



595 Timing Diagram

The code is based on two pieces of information in the datasheet: the timing diagram and the logic table. The logic table is what tells you that basically everything important happens on an up beat. When the clockPin goes from low to high, the shift register reads the state of the data pin. As the data gets shifted in it is saved in an internal memory register. When the latchPin goes from low to high the sent data gets moved from the shift registers aforementioned memory register into the output pins, lighting the LEDs.

Code Sample 1.1 Hello World

Code Sample 1.2 One by One

Code Sample 1.3 Using an array

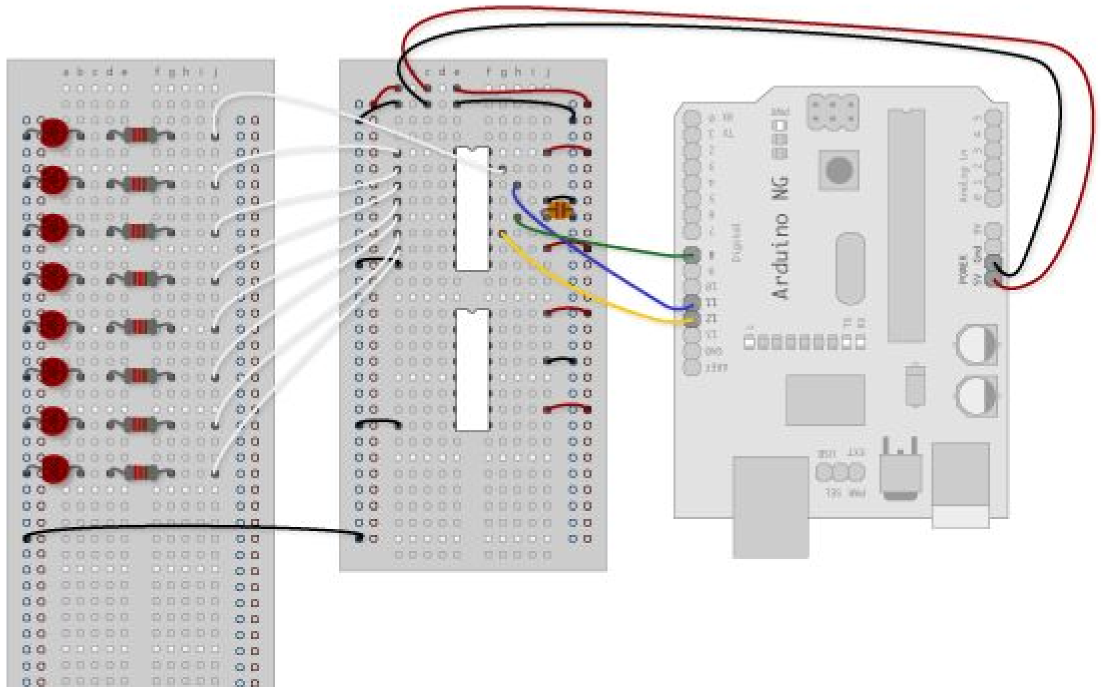
Example 2

In this example you'll add a second shift register, doubling the number of output pins you have while still using the same number of pins from the Arduino.

The Circuit

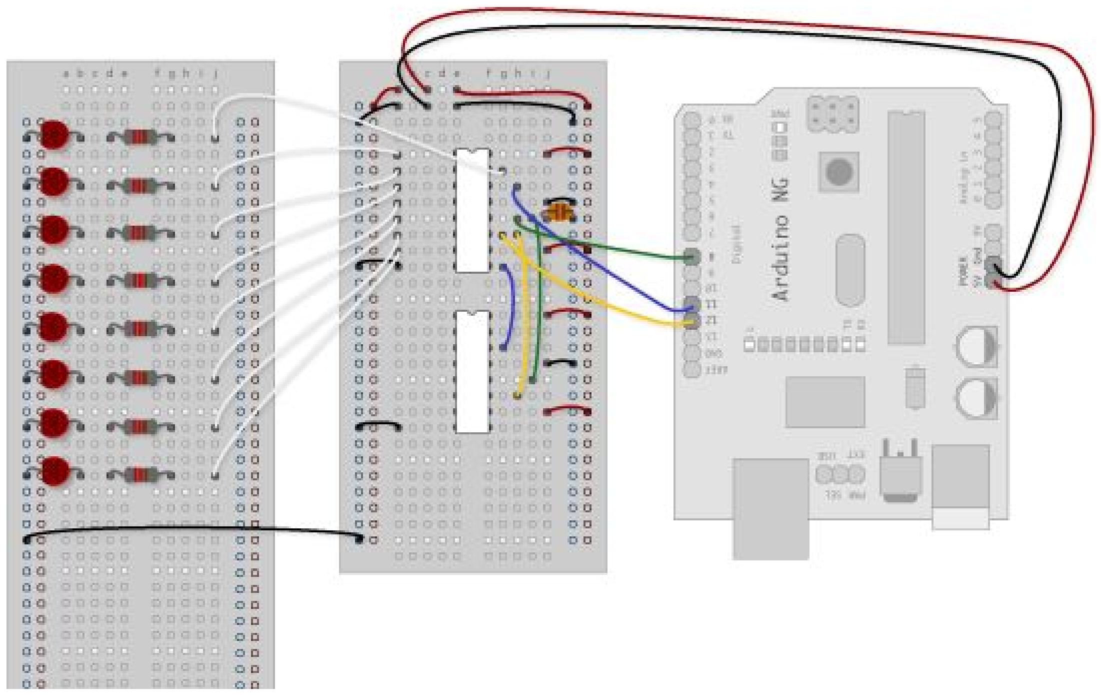
1. Add a second shift register.

Starting from the previous example, you should put a second shift register on the board. It should have the same leads to power and ground.



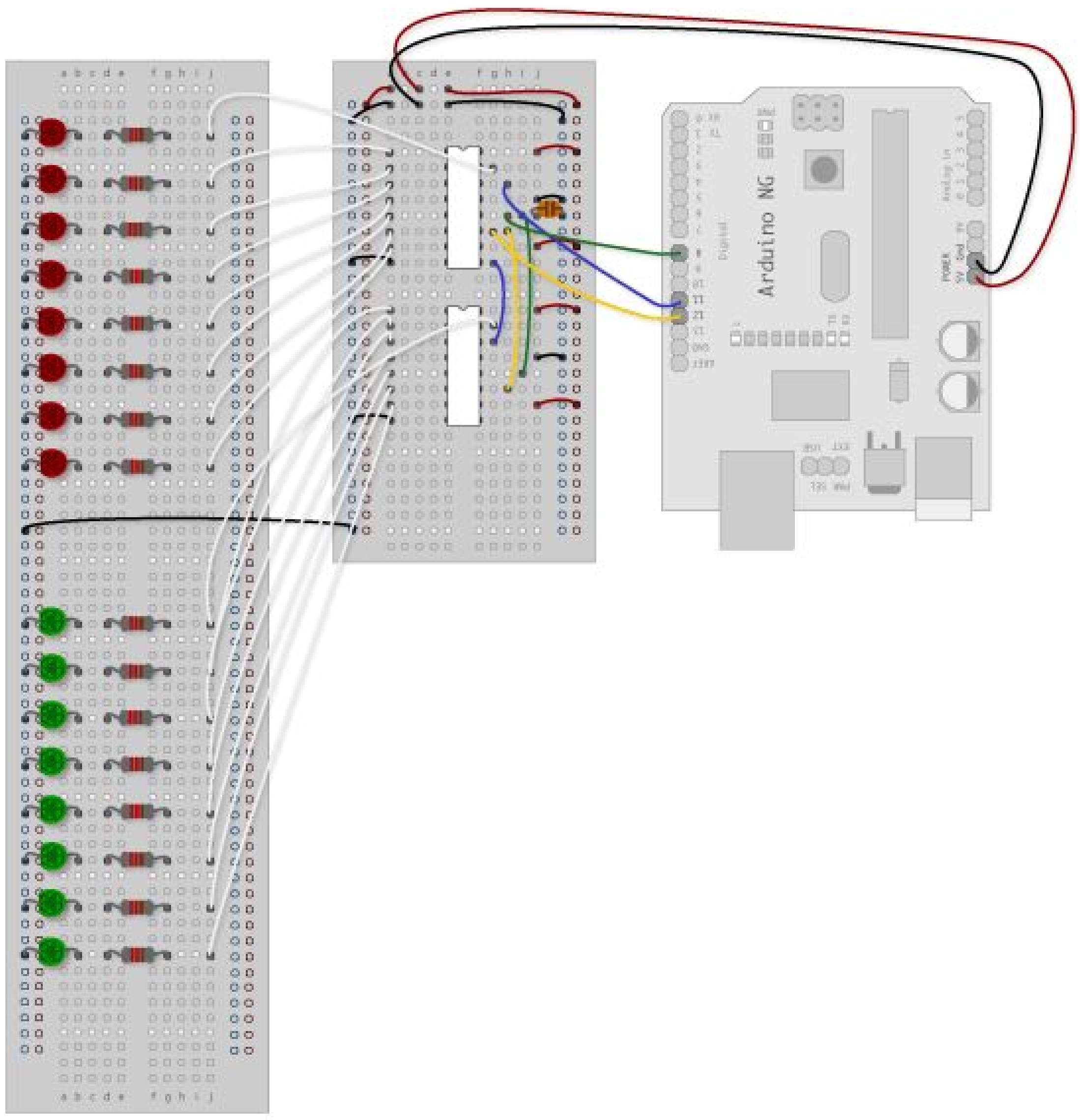
2. Connect the 2 registers.

Two of these connections simply extend the same clock and latch signal from the Arduino to the second shift register (yellow and green wires). The blue wire is going from the serial out pin (pin 9) of the first shift register to the serial data input (pin 14) of the second register.

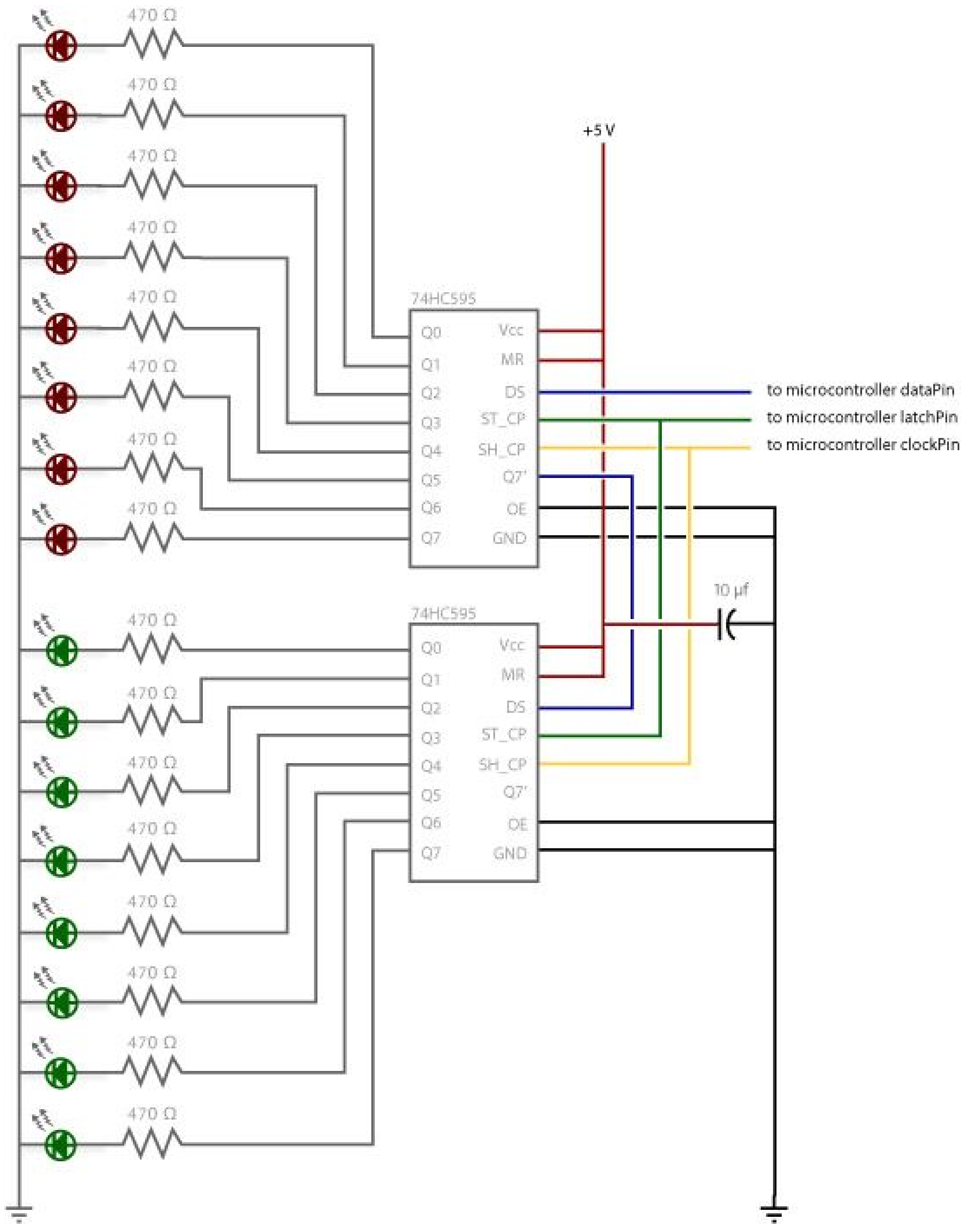


3. Add a second set of LEDs.

In this case I added green ones so when reading the code it is clear which byte is going to which set of LEDs



Circuit Diagram


470 Ω
+5 V
74HC595
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Vcc
MR
DS
ST_CP
SH_CP
Q7'
OE
GND
to microcontroller dataPin
to microcontroller latchPin
to microcontroller clockPin
10 µf

The Code

Here again are three code samples. If you are curious, you might want to try the samples from the first example with this circuit set up just to see what happens.

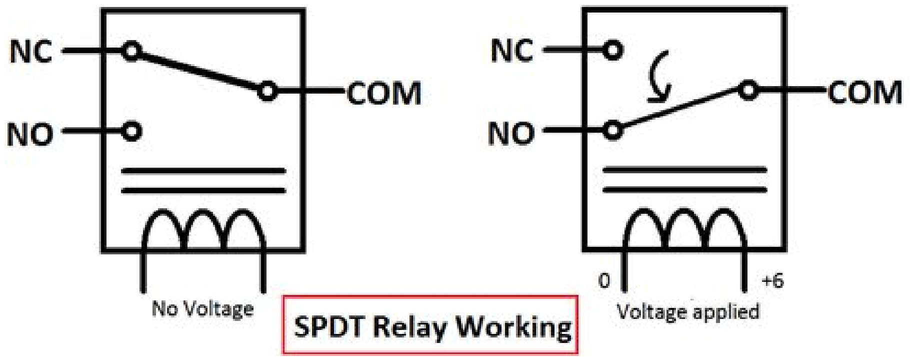
Code Sample 2.1 Dual Binary Counters

There is only one extra line of code compared to the first code sample from Example 1. It sends out a second byte. This forces the first shift register, the one directly attached to the

Arduino, to pass the first byte sent through to the second register, lighting the green LEDs. The second byte will then show up on the red LEDs.
Code Sample 2.2 2 Byte One By One
Comparing this code to the similar code from Example 1 you see that a little bit more has had to change. The blinkAll() function has been changed to the blinkAll_2Bytes() function to reflect the fact that now there are 16 LEDs to control. Also, in version 1 the pulsings of the latchPin were situated inside the subfunctions lightShiftPinA and lightShiftPinB(). Here they need to be moved back into the main loop to accommodate needing to run each subfunction twice in a row, once for the green LEDs and once for the red ones.
Code Sample 2.3 - Dual Defined Arrays
Like sample 2.2, sample 2.3 also takes advantage of the new blinkAll_2bytes() function. 2.3's big difference from sample 1.3 is only that instead of just a single variable called "data" and a single array called "dataArray" you have to have a dataRED, a dataGREEN, dataArrayRED, dataArrayGREEN defined up front. This means that line
data = dataArray[j];
becomes
dataRED = dataArrayRED[j];
dataGREEN = dataArrayGREEN[j];
and
shiftOut(dataPin, clockPin, data);
becomes
shiftOut(dataPin, clockPin, dataGREEN);
shiftOut(dataPin, clockPin, dataRED);

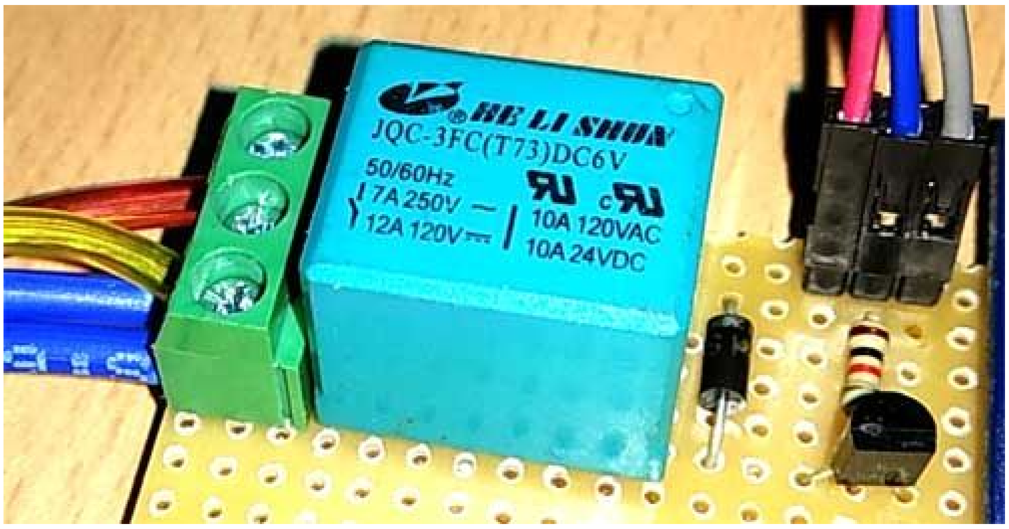
Relay:

Relay is an electromagnetic switch, which is controlled by small current, and used to switch ON and OFF relatively much larger current. Means by applying small current we can switch ON the relay which allows much larger current to flow. A relay is a good example of controlling the AC (alternate current) devices, using a much smaller DC current. Commonly used Relay is **Single Pole Double Throw (SPDT) Relay**, it has five terminals as below:

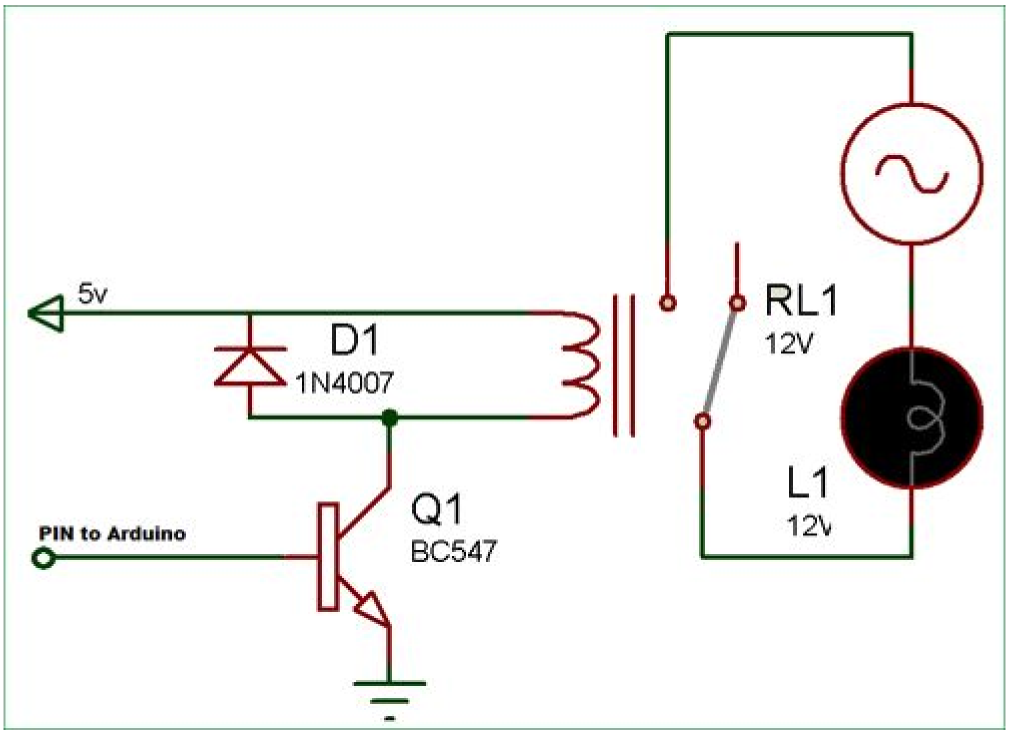


When there is no voltage applied to the coil, COM (common) is connected to NC (normally closed contact). When there is some voltage applied to the coil, the electromagnetic field produced, which attracts the Armature (lever connected to spring), and COM and NO (normally open contact) gets connected, which allow a larger current to flow. Relays are available in many ratings, here we used 6V operating voltage relay, which allows 7A-250VAC current to flow.

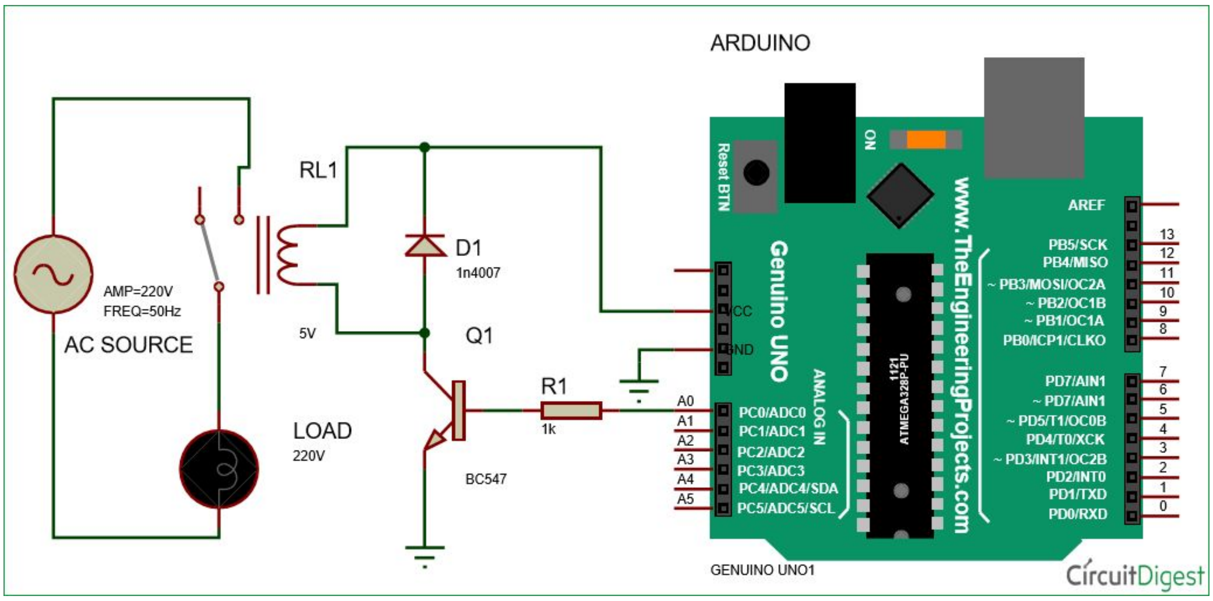
The relay is always configured by using a small **Driver circuit** which consists a Transistor, Diode and a resistor. **Transistor** is used to amplify the current so that full current (from the DC source – 9v battery) can flow through a coil to fully energies it. The **resistor** is used to provide biasing to the transistor. And **Diode** is used to prevent reverse current flow, when the transistor is switched OFF. Every Inductor coil produces equal and opposite EMF when switched OFF suddenly, this may cause permanent damage to components, so Diode must be used to prevent reverse current. A **Relay module** is easily available in the market with all its Driver circuit on the board or you can create it on perf board or PCB like below. Here we have used 6V Relay module.



Here to **turn on the Relay with Arduino** we just need to make that Arduino Pin High (A0 in our case) where Relay module is connected. Below given is **Relay Driver Circuit** to build your own Relay module:



Circuit Diagram and Working:



In this **Arduino Relay Control Circuit** we have used Arduino to control the relay via a BC547 transistor. We have connected transistor base to Arduino pin A0 through a 1k resistor. An AC bulb is used for demonstration. The 12v adaptor is used for powering the circuit.

Working is simple, we need to **make the RELAY Pin (PIN A0) high to make the Relay module ON** and **make the RELAY pin low to turn off the Relay Module**. The AC light will also turn on and off according to Relay.

We just programmed the Arduino to make the Relay Pin (A0) High and Low with a delay of 1 second:

```
void loop()
{
   digitalWrite(relay, HIGH);
   delay(interval);
   digitalWrite(relay, LOW);
   delay(interval);
}
```

Complete code for Arduino Relay Control is given below.

Code:

```
// Arduino Relay Control Code


#define relay A0

#define interval 1000

void setup() {

  pinMode(relay, OUTPUT);

}

void loop()

{

   digitalWrite(relay, HIGH);

   delay(interval);

   digitalWrite(relay, LOW);

   delay(interval);

}
```

Assignment (Level 1):

Batman was getting more and more popular and becoming a role model but he forgot the basic of **justice. "Rules are same for everyone!"**
One day he was roaming around in his batmobile and suddenly traffic police stopped him and told him to pay the penalty of $ 1,000....Batman was not able to digest that a traffic policeman fined him!!! And for what??
Guess what...There were no INDICATOR LEDs in his car! **LOL**
That day he felt ashamed of himself and took a note to install indicators in his car the very next day.
Now coming to the point!
You are the best engineer, Batman has. He told you to fix the issue.
Your task is to use tinkercad and make a car driven by **L293D motor driver** (Both Rear wheels**)** and having 4 LEDs (2 Red LEDs in the back for brake, and one green LED on either side for Left/Right Turn)
Program the microcontroller in such a way that you can control the car using the motor driver and Indicators must glow while turning(Left/Right) and braking.

(Be careful while making the connections, make sure you connect all the GNDs to the arduino GND.Otherwise...Ohh, I forgot that you can’t burn Arduino/Arduinos in simulator!)

Assignment (Level 2):

After fixing the Indicators, Batman wants to automate his car!

Your job is to make the car sensing any obstacle coming closer from right or left and to prevent collision, car should turn in opposite direction.
For this, install PING sensors on the both sides of the car heading 45 degree left and right.
In the program, take input as distances of the obstacles coming from either side and give respective control outputs to turn in opposite direction.