Info

<https://www.youtube.com/watch?v=6fVbJbNPrEU>

<http://bildr.org/2011/02/74hc595/>

<https://www.electronics-tutorials.ws/transistor/tran_2.html>

<https://electronics.stackexchange.com/questions/91307/which-resistor-for-npn-transistor-base>

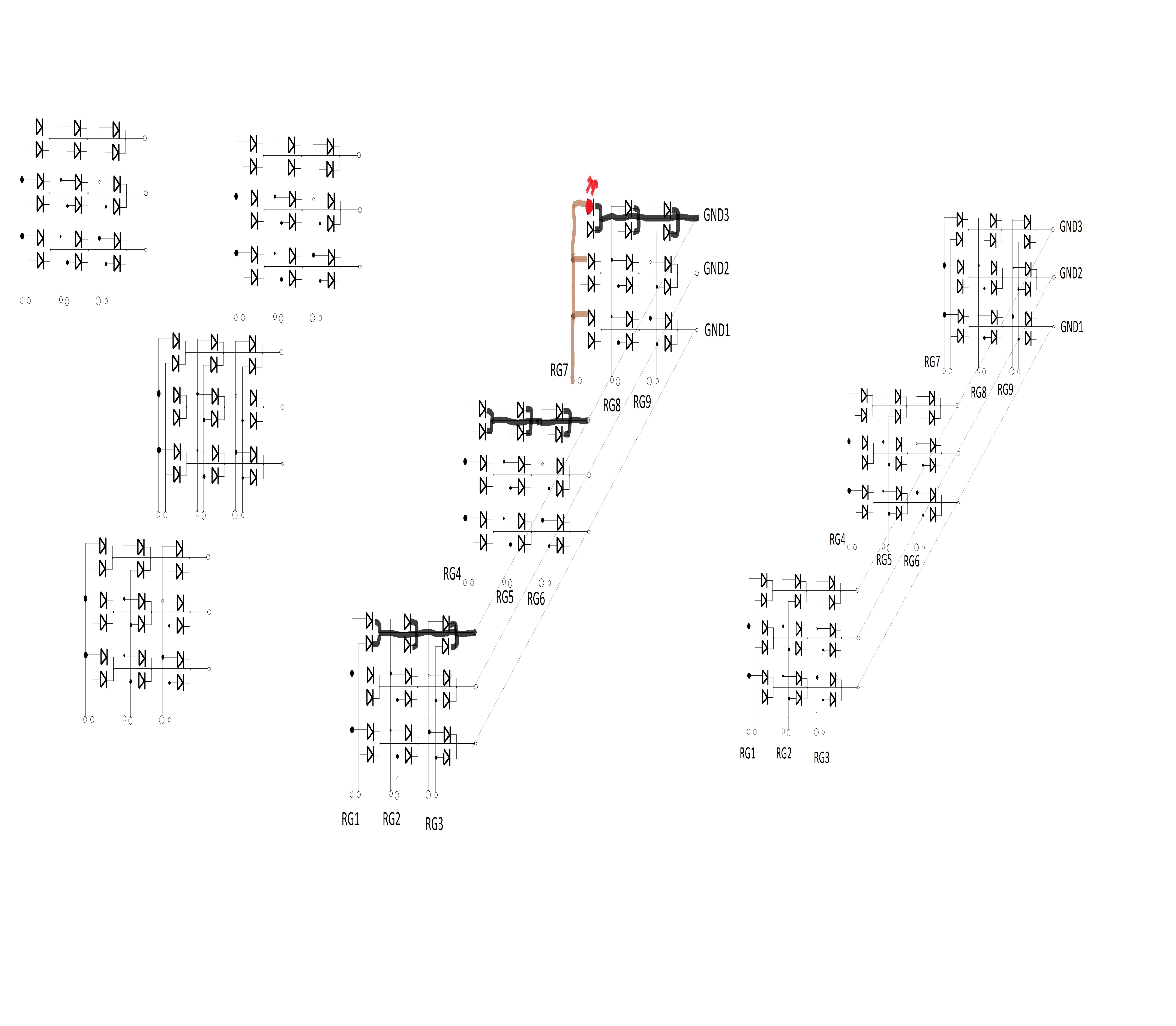
<https://www.instructables.com/id/LED-Cube-and-Arduino-Lib/>

# Information about LED cubes

First off I think we should know how this devices work. They are made of panels of LED diodes connecting them to a common GND or Vcc+. When we connect LEDs in common cathode (-) mode, all shorter legs are soldered together (or just united in some way) and when applying voltage to the other “free leg” it will light up only one LED. To light all the LEDs we would have to give power to each of the anodes one by one or at the same time.

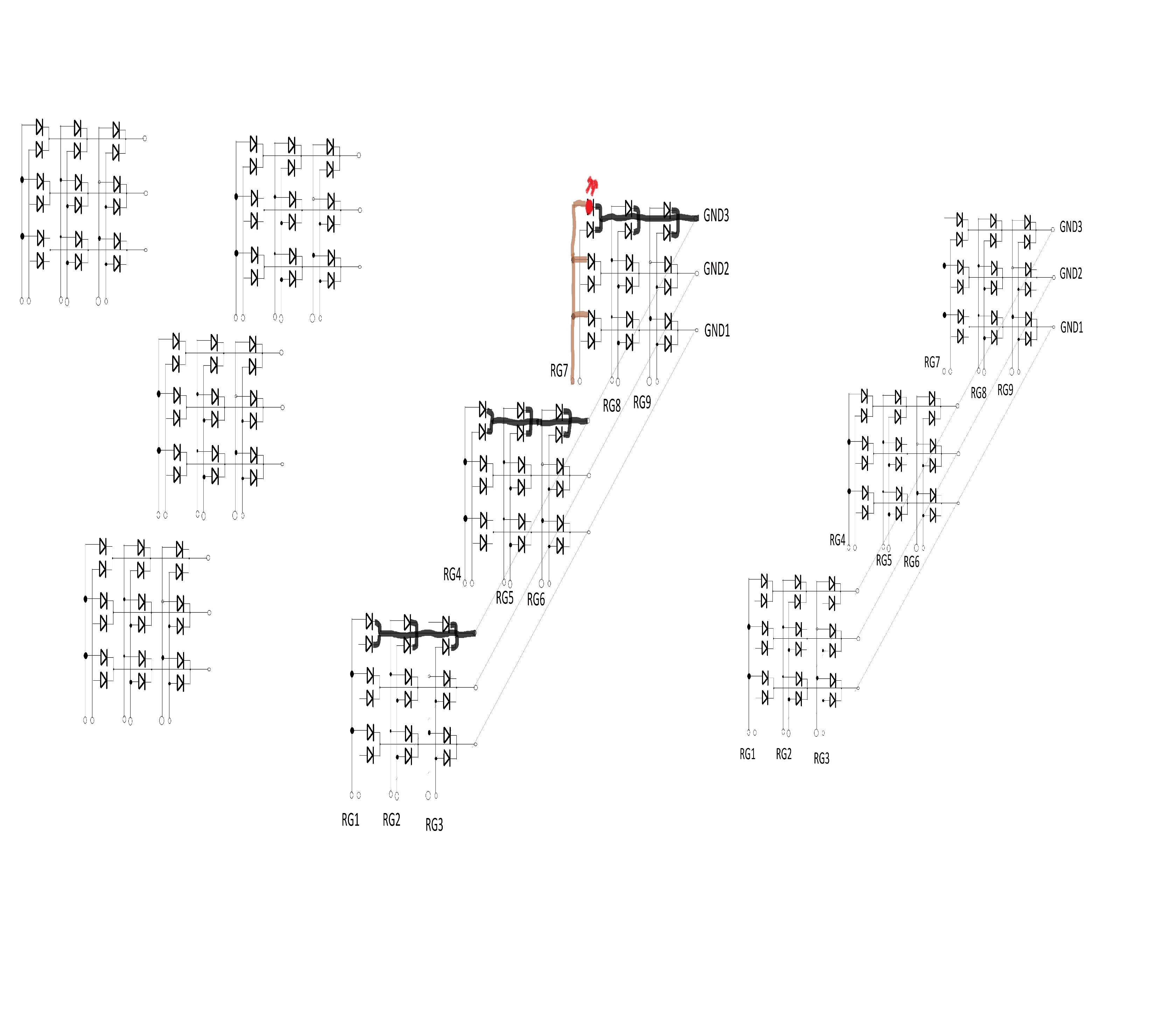
# Dual color LED cube working

We decided to use bi colour LEDs to show different information easily. These LEDs work like if they were two LEDs per each actual LED. Therefore for each column we will have two anodes, one that connects the red LED and the other the green LED. They are common cathode, meaning the gnd is shared.

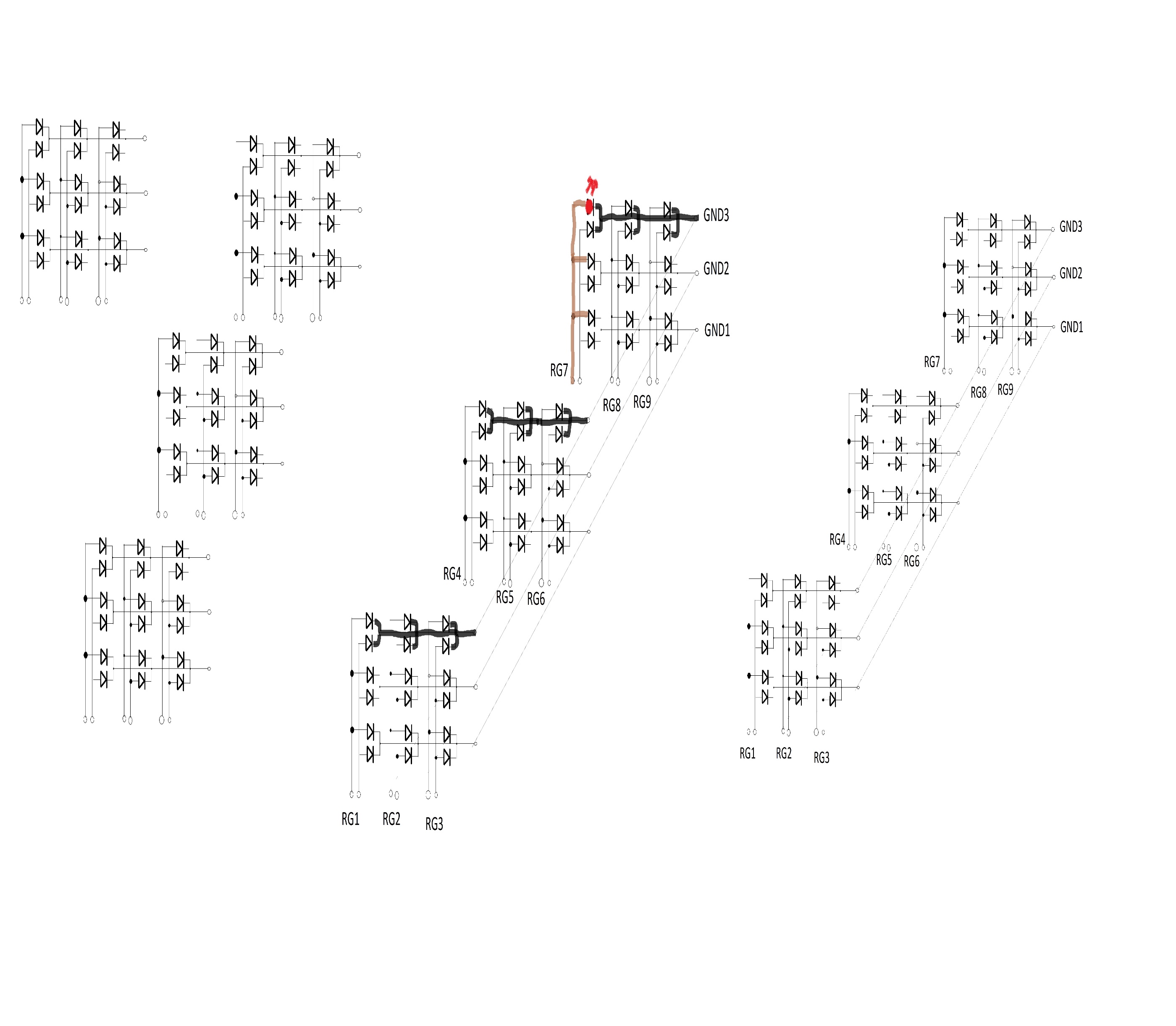
A 3x3x3 cube can be separated in 3 vertical slices by 3 horizontal layers. To simplify the explanation, let’s take a look into one of the vertical slices.

In the picture above, the vertical lines (columns) are the two anodes we get and the horizontal lines (rows) are the common GND. In this way, a particular LED will light up only when it’s row is connected to GND and the column set to HIGH (or connected to the power supply). To light up two different rows we would have to connect each row to GND, but by doing so we would light up the LED corresponding to that row at the column we light up before and we might not want that.

To avoid this, we use multiplexing. We send the data for what LED we want to turn on in the first row and connect the first row to GND. After some miliseconds we change the row connected to GND and send new information, now about which LEDs we want to turn on from the next row. And so on, repeating again and again depending on the code we had uploaded.

Then, we would need 2 control pin for each column to control the LED turned on and the color, and 1 control pin for each row. For only this one slice that would make 6 control pin (columns) + 3 control pins (rows) = 9 pins from out microcontroller. That seems feasible. However we are not trying to make a 3x1x3 LED cube (if we would call that a cube) so two more slices have to go in.

Now we have something that starts to look like a cube… sort of. I hope the picture above is clear enough to understand, we simply added to more vertical slices to the previous one. Also, a connection between the GND from each of the rows has been made, creating a layer (GND1, GND2, GND3). Now, by connecting each layer to ground we get the same result that we got for the row in 2D explanation but in 3D. So to light up the red LED of the upper left corner of the last vertical slice, we would have to set HIGH R7 and LOW GND3.

By doing so we are also connecting the corresponding layer (upper layer) entirely connected to GND (as the picture can hopefully show) and the 7th column to HIGH. However only our red LED on the upper left corner is turned on, since it is the only one completing the circuit. If we would want to connect the middle green LED from the same slice, the one above it and the one to it’s left would light up too. Again, as in 2D we would have to multiplex between the layers.

What about the number of pins needed now? Has it changed? Well for the layer/row control it remains the same: 3 control pin one for each layer. However, we have 6 more columns, with two new pins each and those need an individual connection for themselves, meaning 12 new control pins for LED and color. Now we need 18 pins for LED control (column control) and 3 pins for layer control (GND). We intend on adding a sensor to serve as an input of data for the cube to change its color and animation, therefore we need 2 pins for I2C control. In total 23 pins of a microcontroller. The Arduino Uno (microcontroller to be used) has 14 digital I/O pins and 6 analog I/O, a total of 20 pins which is 3 less than what we need. Then we would have to add another microcontroller for the last 3 pins which, let’s say, control the GND multiplexing. OR we could add shift registers that can take 3 pins of the Arduino and control 8, 16 or more outputs (depending on the shift register). This would get the number of used pins low enough to be handled by one single Arduino UNO, which is what we have in hand (also, buying a shift register is cheaper than a new microcontroller).

A new question arises then, what is a shift register, what does it do and how does it work with the cube?

Why connect transistors in layers (GND)?

Each of the layers are connected to a transistor that enables the cube to turn on and off the flow of current through each layer.

So we can control when the layer is Connected to gnd. Otherwise it would be always Connected to gnd or we would have to send a LOW output from arduino I/O but that is not gnd, it is just LOW output (less than 2’5V I believe).

You have to use those transistors because the arduino can only sink up to 40mA per pin, with 5 transistors you only need a limited amount of current from the arduino pins to the base of the transistor to switch it to gnd.

Shift registers workings:

Shift in all the data (bytes of desired output) and then latch them (“open them to the outputs”). So basically you load the animation/configuration you want and sen dit, then load next and sen dit and so on. The data is shifted inside storage by clk and once all is layed out it is shifted to the outputs by the latch.

We use three SIPO shift registers daisy chained to transfer our data from the arduino to the LED cube. It is needed to reduce the number of I/O pins used from the arduino to the mínimum (Data, latch and clk) This way we can manage 24 outputs with only 3 pins. This is possible by sending the data of which outputs we want to turn on and off in binary. Each SR has 8 outputs, so 1 byte, 8 bits. Sending 00000001 will make the first output turn 1 and the rest 0 (since we are shifting to the right). Lets clarify. In the code, to send this one byte of data (telling which outputs to turn on or off) we make an AND operation with a mask 1<<7 (10000000). The result can be either 1 if the MSB from the data byte is 1 or 0 if it is 0. If result equals 1, then we set the Data shift pin of the arduino, otherwise we clear it. Later we cycle the clock and this bit that has been transfered to the data shift pin will be set and stored in the shift registre in the first output position (Qa for the 74HC595). During the clock, a shift to the right inside the shiftregister also occurs, creating space for the new Data shift value to be stored in Qa and transferint/shifting the previous one into Qb.

However our data byte has 7 more bits to transfer, therefore we have to make an AND operation for every bit of our data byte. Before making the next operation though, a shift in our data byte is needed. Since the mask is 1<<7, the shifting will be to the left by one bit. With the new value of our Data byte we repeat all the steps again and again until we do the entire byte (8 bits).

Example-   
Data byte 01101110  
Mask 1000000  
Data shift pin –  
Qa: - ; Qb: - ; Qc: - ; Qd: - ; Qe: - ; Qf: - ; Qg: - ; Qh: - ;

1st operation  
01101110  
AND  
10000000  
result = 00000000 -> Data shift pin clear (0)  
Clk change -> shifting the stored data (no data so far, no changes)  
Qa: - ; Qb: - ; Qc: - ; Qd: - ; Qe: - ; Qf: - ; Qg: - ; Qh: - ;   
sending Data shift value to Qa. Qa=0  
Qa: 0 ; Qb: - ; Qc: - ; Qd: - ; Qe: - ; Qf: - ; Qg: - ; Qh: - ;  
  
Shift Data byte one to the left -> Data byte: 11011100

2nd operation  
11011100  
AND  
10000000  
result = 10000000 -> Data shift pin set (1)  
Clk change -> shifting the stored data to the right (Qb=Qa) & sending Data shift value to Qa. Qa=1  
Qa: 1 ; Qb: 0 ; Qc: - ; Qd: - ; Qe: - ; Qf: - ; Qg: - ; Qh: - ;

Shift previous Data byte one to the left -> Data byte: 10111000

3rd operation  
10111000  
AND  
10000000  
result = 10000000 -> Data shift pin set (1)  
Clk change -> shifting the stored data to the right (Qb=Qa; Qc=Qb) & sending Data shift value to Qa. Qa=1  
Qa: 1 ; Qb: 1 ; Qc: 0 ; Qd: - ; Qe: - ; Qf: - ; Qg: - ; Qh: - ;

Shift previous Data byte one to the left -> Data byte: 01110000

...

8th operation  
00000000  
AND  
10000000  
result = 00000000 -> Data shift pin clear (0)  
Clk change -> shifting the stored data to the right (Qb=Qa; Qc=Qb; Qd=Qc;...) & sending Data shift value to Qa. Qa=0  
Qa: 0 ; Qb: 1 ; Qc:1 ; Qd: 1 ; Qe: 0 ; Qf: 1 ; Qg: 1 ; Qh: 0 ;

Shift previous Data byte one to the left -> Data byte: 10111000

Now that we have our data byte loaded into the shift register we can proceed to send it to the outputs with the latch pin signal. From the exemple, the outputs devices Connected to Qb, Qc, Qd, Qf and Qg will be the ones to receive HIGH State.

First try, only one anode Connected

Since we could not get the shift registers ready on time for the lab testing we had to use only one of the anodes of the LEDs to be able to connect and try the functionality of the cube. Using both anodes without SR would mean an amount of I/O pins from the microcontroller that we did not have.

Therefore we connected our 9 columns of anodes and the 3 GND layers to the Arduino UNO and programmed the coding to light up and down the different pins with different layers, creating animations. As a prove of concept, only two animations were done:

1st- All LEDs turned on and layer shifting, impression of going up.

2nd- All LEDs from one slice (vertical layer) tunred on and shifting to the next slice, impression of moving to the side.

Once the code was working properly and the animations perceived as they were supposed to, the implementation of the temperature sensor was done.

For this we needed two more pins, specifically A4 and A5 that correspond to SDA (data) and SCL (clock) respectively of the *twi* from the Arduino UNO.

The code was expanded as the instruction to read from the sensor had to be added. Then, a condition to make one animation or the other after reading the data from the temperature sensor. It was established that up until 28ºC we would perceive the animation of LEDs going up and beyond 28ºC a change in animation would happen.

At first we wanted to use interrupts to call the reading of the temperature. By dping this we could have control on how fast the reading would take place (by changing the different pre scale). For a reason we could not find, this idea would not work correctly so we moved the calling to the main while loop. Since it worked as we told it to our first attempt was finished and successful.

Some problems along the way:

* Not adding a delay between the turn off and turn on status, making the LEDs on all time.
* Buffer size of transmition had to be changed again to send the sensor data correctly. From 32 bytes to 64 bytes.
* Use of NPN transistors not working according to theory. As we found out, using three NPN BJT transistors to connect the GND layers to the actual ground would make it safer for the board and controllable from I/O pins. Since it didn’t work, we used the LOW state for the GND pin to connect the specific layer to GND. It worked.
* Resistors of unknown value made us use some resistors that were too big, impeding the LEDs of the column to light up.
* Implementation of sensor and LEDs not possible through interrupts.

Code:

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* **@file** main.c

\* **@author** Alexander J. Magnusson & José Cuevas, based off examples from Tomas Fryza

\* Originally from Tomas Fryza, Brno University of Technology, Czechia

\* **@version** V1.1

\* **@date** Nov 29, 2018

\* **@brief** Light up a 3x3x3 single anode LED cube in two different animations

depending on the data received from the DHT12 temperature sensor.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\* Includes ------------------------------------------------------------------\*/

#include "settings.h"

#include <avr/io.h>

#include <avr/interrupt.h>

#include <stdio.h>

#include <stdlib.h>

#include <util/delay.h>

#include "twi.h"

#include "uart.h"

/\* Constants and macros ------------------------------------------------------\*/

/\*\*

\* **@brief** Define UART buad rate.

\*/

#define UART\_BAUD\_RATE 9600

#define DHT12 0x5c

struct values**{**

uint8\_t temperature\_integer**;**

uint8\_t temperature\_decimal**;**

**};**

struct values Meteo\_values**;**

/\* Function prototypes -------------------------------------------------------\*/

/\*\*

\* **@brief** Initialize UART, TWI, and Timer/Counter1.

\*/

void setup**(**void**);**

/\*\*

\* **@brief** TWI Finite State Machine transmits all slave addresses.

\*/

void fsm\_twi\_scanner**(**void**);**

/\* Global variables ----------------------------------------------------------\*/

**typedef** enum **{**

IDLE\_STATE **=** 1**,**

SLA\_W\_STATE**,**

ACK\_STATE**,**

TEMPERATURE\_STATE**,**

UART\_STATE**,**

**}** state\_t**;**

/\* FSM for scanning TWI bus \*/

state\_t twi\_state **=** IDLE\_STATE**;**

/\* Functions -----------------------------------------------------------------\*/

/\*\*

\* **@brief** Main function.

\*/

int main**(**void**)**

**{**

/\* Initializations \*/

setup**();**

/\* Enables interrupts by setting the global interrupt mask \*/

sei**();**

/\* Forever loop \*/

**while** **(**1**)** **{**

fsm\_twi\_scanner**();**

**if** **(**Meteo\_values**.**temperature\_integer **<=** 28**)**

**{**

TestAnimation1**();**

**}**

**else**

**{**

TestAnimation2**();**

**}**

**}**

**return** 0**;**

**}**

/\*\*

\* **@brief** Setup all peripherals.

\*/

void setup**(**void**)**

**{**

/\* Initialize UART: asynchronous, 8-bit data, no parity, 1-bit stop \*/

uart\_init**(**UART\_BAUD\_SELECT**(**UART\_BAUD\_RATE**,** F\_CPU**));**

/\* Initialize TWI \*/

twi\_init**();**

/\* Set LEDs as outputs \*/

DDRB **|=** \_BV**(**PB5**);**

DDRB **|=** \_BV**(**PB4**);**

DDRB **|=** \_BV**(**PB3**);**

DDRB **|=** \_BV**(**PB2**);**

DDRB **|=** \_BV**(**PB1**);**

DDRB **|=** \_BV**(**PB0**);**

DDRD **|=** \_BV**(**PD7**);**

DDRD **|=** \_BV**(**PD6**);**

DDRD **|=** \_BV**(**PD5**);**

/\* and GND layer pins \*/

DDRD **|=** \_BV**(**PD4**);**

DDRD **|=** \_BV**(**PD3**);**

DDRD **|=** \_BV**(**PD2**);**

/\* Turn all LEDs off \*/

PORTB **&=** **~**\_BV**(**PB5**);**

PORTB **&=** **~**\_BV**(**PB4**);**

PORTB **&=** **~**\_BV**(**PB3**);**

PORTB **&=** **~**\_BV**(**PB2**);**

PORTB **&=** **~**\_BV**(**PB1**);**

PORTB **&=** **~**\_BV**(**PB0**);**

PORTD **&=** **~**\_BV**(**PD7**);**

PORTD **&=** **~**\_BV**(**PD6**);**

PORTD **&=** **~**\_BV**(**PD5**);**

/\* and GND layer pins. When cleared, they close the circuit

\* connecting the layer to GND and making it possible for

\* the LEDs of the specific layer to turn on. \*/

PORTD **|=**\_BV**(**PD4**);**

PORTD **|=**\_BV**(**PD3**);**

PORTD **|=**\_BV**(**PD2**);**

**}**

/\* LED cube animation of turned on LEDs layer shifting up \*/

/\*\*

\* function animation test 1

\*/

void TestAnimation1**()**

**{**

PORTD **&=** **~**\_BV**(**PD4**);**

PORTD **|=**\_BV**(**PD3**);**

PORTD **|=**\_BV**(**PD2**);**

PORTB **|=**\_BV**(**PB5**);**

PORTB **|=**\_BV**(**PB4**);**

PORTB **|=**\_BV**(**PB3**);**

PORTB **|=**\_BV**(**PB2**);**

PORTB **|=**\_BV**(**PB1**);**

PORTB **|=**\_BV**(**PB0**);**

PORTD **|=**\_BV**(**PD7**);**

PORTD **|=**\_BV**(**PD6**);**

PORTD **|=**\_BV**(**PD5**);**

\_delay\_ms **(**200**);**

PORTB **&=** **~**\_BV**(**PB5**);**

PORTB **&=** **~**\_BV**(**PB4**);**

PORTB **&=** **~**\_BV**(**PB3**);**

PORTB **&=** **~**\_BV**(**PB2**);**

PORTB **&=** **~**\_BV**(**PB1**);**

PORTB **&=** **~**\_BV**(**PB0**);**

PORTD **&=** **~**\_BV**(**PD7**);**

PORTD **&=** **~**\_BV**(**PD6**);**

PORTD **&=** **~**\_BV**(**PD5**);**

/\* Change of layer \*/

PORTD **&=** **~**\_BV**(**PD3**);**

PORTD **|=**\_BV**(**PD4**);**

PORTD **|=**\_BV**(**PD2**);**

PORTB **|=**\_BV**(**PB5**);**

PORTB **|=**\_BV**(**PB4**);**

PORTB **|=**\_BV**(**PB3**);**

PORTB **|=**\_BV**(**PB2**);**

PORTB **|=**\_BV**(**PB1**);**

PORTB **|=**\_BV**(**PB0**);**

PORTD **|=**\_BV**(**PD7**);**

PORTD **|=**\_BV**(**PD6**);**

PORTD **|=**\_BV**(**PD5**);**

\_delay\_ms **(**200**);**

PORTB **&=** **~**\_BV**(**PB5**);**

PORTB **&=** **~**\_BV**(**PB4**);**

PORTB **&=** **~**\_BV**(**PB3**);**

PORTB **&=** **~**\_BV**(**PB2**);**

PORTB **&=** **~**\_BV**(**PB1**);**

PORTB **&=** **~**\_BV**(**PB0**);**

PORTD **&=** **~**\_BV**(**PD7**);**

PORTD **&=** **~**\_BV**(**PD6**);**

PORTD **&=** **~**\_BV**(**PD5**);**

/\* Change of layer \*/

PORTD **&=** **~**\_BV**(**PD2**);**

PORTD **|=**\_BV**(**PD4**);**

PORTD **|=**\_BV**(**PD3**);**

PORTB **|=**\_BV**(**PB5**);**

PORTB **|=**\_BV**(**PB4**);**

PORTB **|=**\_BV**(**PB3**);**

PORTB **|=**\_BV**(**PB2**);**

PORTB **|=**\_BV**(**PB1**);**

PORTB **|=**\_BV**(**PB0**);**

PORTD **|=**\_BV**(**PD7**);**

PORTD **|=**\_BV**(**PD6**);**

PORTD **|=**\_BV**(**PD5**);**

\_delay\_ms **(**200**);**

PORTB **&=** **~**\_BV**(**PB5**);**

PORTB **&=** **~**\_BV**(**PB4**);**

PORTB **&=** **~**\_BV**(**PB3**);**

PORTB **&=** **~**\_BV**(**PB2**);**

PORTB **&=** **~**\_BV**(**PB1**);**

PORTB **&=** **~**\_BV**(**PB0**);**

PORTD **&=** **~**\_BV**(**PD7**);**

PORTD **&=** **~**\_BV**(**PD6**);**

PORTD **&=** **~**\_BV**(**PD5**);**

**}**

/\* LED cube animation of turned on LEDs slices shifting to the side \*/

/\*\*

\* function animation test 2

\*/

void TestAnimation2**()**

**{**

PORTD **&=** **~**\_BV**(**PD4**);**

PORTD **&=** **~**\_BV**(**PD3**);**

PORTD **&=** **~**\_BV**(**PD2**);**

PORTB **|=**\_BV**(**PB3**);**

PORTB **|=**\_BV**(**PB0**);**

PORTD **|=**\_BV**(**PD5**);**

\_delay\_ms **(**400**);**

PORTB **&=** **~**\_BV**(**PB3**);**

PORTB **&=** **~**\_BV**(**PB0**);**

PORTD **&=** **~**\_BV**(**PD5**);**

PORTB **|=**\_BV**(**PB4**);**

PORTB **|=**\_BV**(**PB1**);**

PORTD **|=**\_BV**(**PD6**);**

\_delay\_ms **(**400**);**

PORTB **&=** **~**\_BV**(**PB4**);**

PORTB **&=** **~**\_BV**(**PB1**);**

PORTD **&=** **~**\_BV**(**PD6**);**

PORTB **|=**\_BV**(**PB5**);**

PORTB **|=**\_BV**(**PB2**);**

PORTD **|=**\_BV**(**PD7**);**

\_delay\_ms **(**400**);**

PORTB **&=** **~**\_BV**(**PB5**);**

PORTB **&=** **~**\_BV**(**PB2**);**

PORTD **&=** **~**\_BV**(**PD7**);**

**}**

void fsm\_twi\_scanner**(**void**)**

**{**

/\* Static variable inside a function keeps its value between callings \*/

//static uint8\_t slave\_address = 0;

uint8\_t twi\_status**;**

char uart\_string3**[**10**];**

char uart\_string4**[**10**];**

**switch** **(**twi\_state**)** **{**

**case** IDLE\_STATE**:**

twi\_state **=** TEMPERATURE\_STATE**;**

**break;**

**case** TEMPERATURE\_STATE**:**

twi\_status **=** twi\_start**((**DHT12**<<**1**)** **+** TWI\_WRITE**);**

**if** **(**twi\_status**==**0**){**

twi\_write **(**0x02**);**

twi\_stop **();**

twi\_start**((**DHT12**<<**1**)** **+** TWI\_READ**);**

Meteo\_values**.**temperature\_integer **=** twi\_read\_ack**();**

Meteo\_values**.**temperature\_decimal **=** twi\_read\_nack**();**

twi\_stop **();**

twi\_state **=** UART\_STATE**;**

**}**

**else** **{**

uart\_puts**(**"Not connected T"**);**

twi\_state **=** IDLE\_STATE**;**

**}**

**break;**

**case** UART\_STATE**:**

uart\_puts**(**"\r\n---Temperature values---:\r\n"**);**

itoa **(**Meteo\_values**.**temperature\_integer**,** uart\_string3**,** 10**);**

itoa **(**Meteo\_values**.**temperature\_decimal**,** uart\_string4**,** 10**);**

uart\_puts**(**uart\_string3**);**

uart\_puts**(**"."**);**

uart\_puts**(**uart\_string4**);**

twi\_state **=** IDLE\_STATE**;**

**break;**

**default:**

twi\_state **=** IDLE\_STATE**;**

**}** /\* End of switch (twi\_state) \*/

**}**

/\* END OF FILE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/