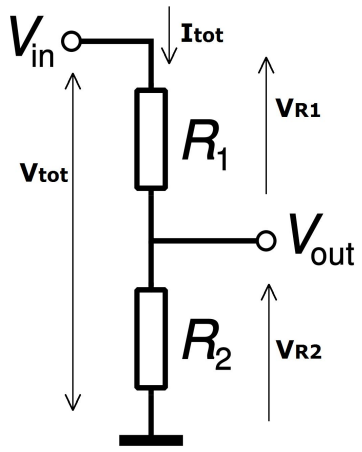
Nel prossimo progetto, che sarà di nuovo un termometro, useremo come sensore di temperatura un termistore, parola derivata dall'unione di termo e resistore. Sappiamo infatti che una resistenza, opponendosi al passaggio della corrente, si riscalda e che c'è relazione tra il riscaldamento e il valore della resistenza. Usando questi principi, sono stati creati i termistori che sono di due tipi:

- **PTC** (coefficiente temperatura positivo) all'aumento della temperatura la resistenza aumenta. Sono usati essenzialmente come protezioni dei circuiti al posto dei fusibili.
- **NTC** (coefficiente temperatura negativo) all'aumento della temperatura la resistenza diminuisce. Sono usati per la costruzione di sonde e termometri.

Quello a nostra disposizione è del secondo tipo cioè NTC.
Per utilizzare questo componente dobbiamo creare un partitore di tensione, aggiungendo in serie al termistore un'altra resistenza di valore uguale al valore di riferimento del termistore (nel nostro caso 10K Ohm), colleghiamo i due lati del partitore all'alimentazione e al centro preleviamo il segnale per l'ingresso analogico di Arduino, con lo scopo di trovare il valore della resistenza del termistore



dobbiamo affrontare un po di matematica per capire come funziona il tutto consideriamo intanto che la resistenza totale $R_{Tot} = R_1 + R_2$

la legge di Ohm ci dice che $I = \frac{V}{R_{TOT}}$ = $I = \frac{V}{R_1 + R_2}$

e la tensione ai capi di ***R1*** sarà $V_{R1} = R_1 * I$

sostituendo la prima formula nella seconda otteniamo

$$V_{R1} = R_1 \frac{V}{R_1 + R_2}$$

che si può scrivere anche $V_{R1} = V \frac{R_1}{R_1 + R_2}$ analogamente la tensione su ***R2*** sarà

$V_{R2} = V \frac{R_2}{R_1 + R_2}$ nel nostro caso la formula avendo R1 un valore noto diventerà:

$V_{R2} = V \frac{R_2}{10.000 + R_2}$ e usando come punto di prelievo della tensione il pin ***+3,3V*** di Arduino, otteniamo

V = 3,3V. Se colleghiamo ***Vout*** all'ingresso analogico di Arduino e misuriamo la tensione otteniamo un valore numerico dato dalla formula

$Valore\ ADC = V_{OUT} * 1023 / A_{Ref}$ siccome ***Vout*** = V_{R2} e V_{in} = ***V*** possiamo scrivere quest'ultima

così: $Valore\ ADC = \frac{R_2}{10\text{K} + R_2} V_{IN} \frac{1023}{A_{Ref}}$

essendo il valore Vin = 3,3V uguale anche ad A_{Ref} = 3,3V perché cosi costruiamo il circuito possiamo semplificare

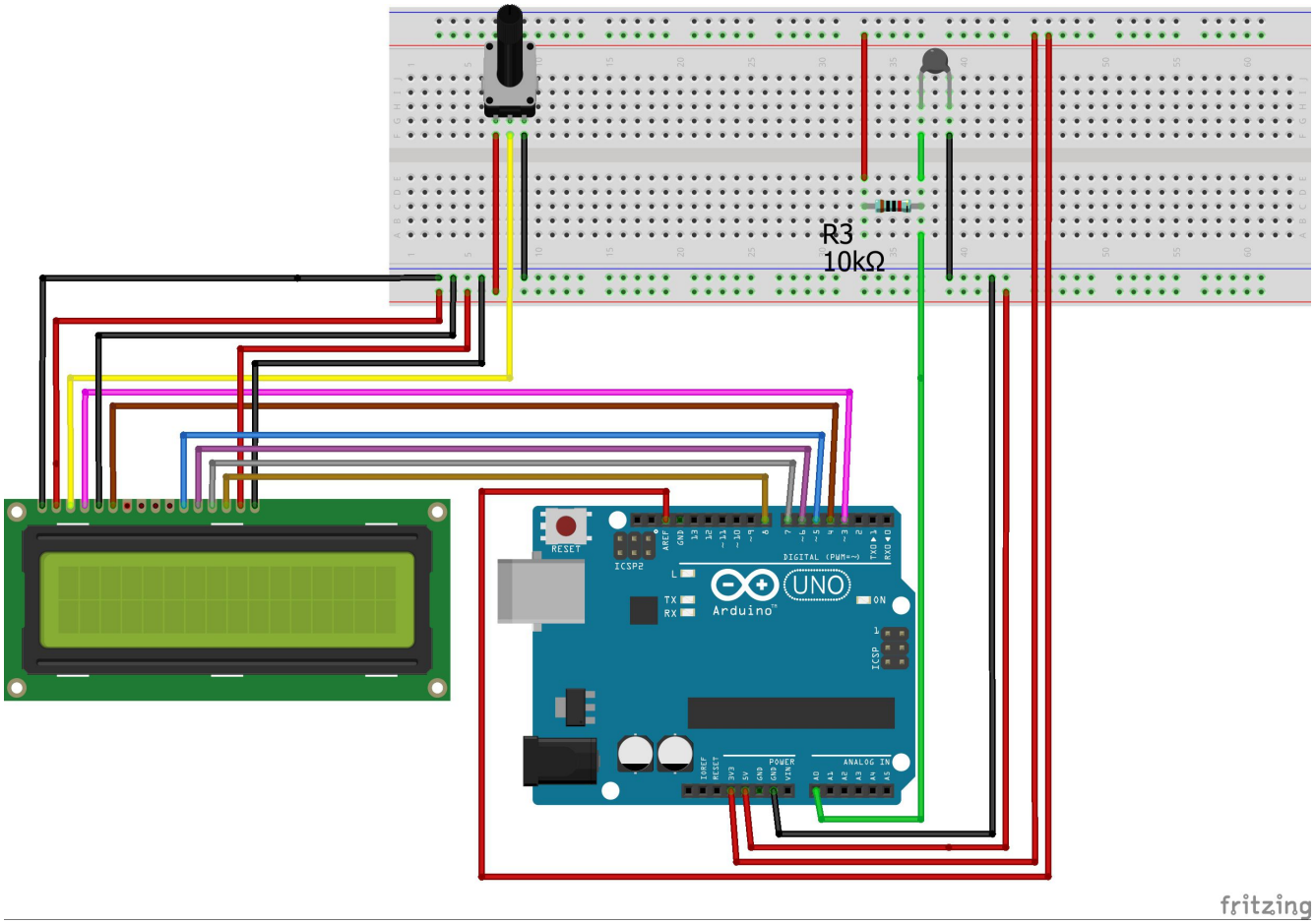
$Val\ ADC = \frac{R_2}{10\text{K} + R_2} 1023$ così otteniamo una formula che è indipendente dalle tensioni usate,

da cui possiamo finalmente calcolare il valore di R_2 : $R_2 = 10K / (1023 / V_{ADC} - 1)$

A questo punto usando l'equazione di Steinhart-Hart possiamo ricavare la temperatura con una precisione soddisfacente:

$\frac{1}{T} = A + B\ln(R) + C(\ln(R))^3$ questa richiede però di conoscere i tre coefficienti A, B, C specifici di ogni termistore che non sono quasi mai disponibili, per cui si preferisce usare la versione semplificata conoscendo solamente il coefficiente B del termistore che è solitamente fornito nel datasheet:

$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B}\ln(\frac{R}{R_0})$ dove T_0 è la temperatura di riferimento usata per calcolare la R_0 del termistore (solitamente 25 °C) espressa però in gradi Kelvin (per cui 25 + 273,15 = 298,15 °K), la resistenza tipica R_0 (nel nostro caso 10K Ohm) e il coefficiente B (compreso tra 3000 e 4000), nel nostro caso 3950.
Prepariamo la nostra breadboard come segue



notare che l'alimentazione a 5V viene usata per l'LCD mentre per il termistore usiamo i 3,3V e la stessa tensione la colleghiamo anche al pin Aref.

Ora scriviamo lo sketch necessario

```
/*  termometro usando termistore in dotazione
 *  alcuni parametri sono dedotti quindi non sicuri...
 *  temperatura di riferimento T0=25 °C
 *  resistenza tipica a 25 °C Rnom.= 10.000 Ohm
 *  coefficiente beta B= 3900
 *
 */
#include <LiquidCrystal.h>

// pin connessione analogica
#define THERMISTORPIN A0
// resistenza a 25 °C
#define THERMISTORNOMINAL 10000
// temperatura di riferimento (quasi sempre 25 °C)
#define TEMPERATURENOMINAL 25
// numero di letture su cui calcolare la media
#define NUMSAMPLES 5
// coefficiente beta del termistore (normalmente 3000-4000)
#define BCOEFFICIENT 3900
// valore dell'altra resistenza in serie: val. nominale 10K Ohm, val. misurato 9950 Ohm
#define SERIESRESISTOR 9950

int samples[NUMSAMPLES];
//collegamenti dell'lcd
//              BS  E  D4 D5 D6 D7
LiquidCrystal lcd(3, 4, 5, 6, 7, 8);

```

```
27
28  void setup(){
29     Serial.begin(9600);
30    lcd.begin(16, 2);
31    // essendo l'alimentazione a 5V relativamente instabile
32    // uso l'alimentazione a 3,3V che è più precisa, come riferimento analogico
33    // e come alimentazione per il termistore, quindi impongo di prelevare il valore
34    // dal contatto esterno ARef
35    analogReference(EXTERNAL);
36  }
37
38  void loop(){
39    uint8_t i;      // dichiarazione equivalente a: unsigned byte i variabile senza segno lunga 8 bit (da 0 a 255)
40    float average;
41
42    // eseguo N letture  di temperatura
43    for (i=0; i< NUMSAMPLES; i++) {
44     samples[i] = analogRead(THERMISTORPIN);
45     delay(10);
46    }
47
48    // calcolo il valore medio delle N letture
49    average = 0;
50    for (i=0; i< NUMSAMPLES; i++) {
51       average += samples[i];
52    }
53    average /= NUMSAMPLES;
54
55    // converto il valore medio in resistenza del termistore
56    // secondo la formula R = 10K / (1023 * VADC - 1)
57    average = 1023 / average - 1;
58    average = SERIESRESISTOR / average;
59
60    Serial.print("Resistenza termistore ");
61    Serial.println(average);
62
63    // ora che conosco R posso, usando la formula
64    // semplificata di Steinhart e Hart, calcolare la temperatura in gradi Kelvin
65    // 1/t = 1/t0 + 1/b * ln(R/R0)
66    float steinhart;
67    steinhart = average / THERMISTORNOMINAL;          // (R/Ro)
68    steinhart = log(steinhart);                       // ln(R/Ro)
69    steinhart /= BCOEFFICIENT;                        // 1/B * ln(R/Ro)
70    steinhart += 1.0 / (TEMPERATURENOMINAL + 273.15); // + (1/To)
71    steinhart = 1.0 / steinhart;                      // inversione
72    // converto i gradi Kelvin in centigradi
73    steinhart -= 273.15;                              // convesione in °C
74
75    Serial.print("Temperatura ");
76    Serial.print(steinhart);
77    Serial.println(" *C");
78
79    // scrivo la temperatura sull'LCD
80    lcd.setCursor(0, 0);
81    lcd.print("Temp        C  ");
82    lcd.setCursor(6, 0);
83    lcd.print(steinhart);
84    delay(1000);
85  }
```

avendo collegato i 3,3V ad Aref su Arduino, bisogna aggiungere nel setup l'istruzione ***analogReference(EXTERNAL)***, se non viene aggiunta, si rischia, anzi è quasi sicuro che si brucerà il pin analogico e probabilmente anche il microcontrollore per cui ATTENZIONE!
analogReference ha tre parametri

1. DEFAULT viene usata la tensione di +5V come riferimento
2. EXTERNAL viene usato il valore di tensione che colleghiamo al pin Aref, la tensione non deve superare i +5V
3. INTERNAL viene usata la tensione interna di +3,3V come riferimento, questa è più stabile rispetto ai +5V di default

Alla riga 23 dello sketch troviamo l'istruzione
int samples[NUMSAMPLES] questa è una nuova istruzione che crea una variabile di tipo array.
Cos'è un array? È una collezione di variabili alle quali si può accedere tramite un indice (che parte da zero!), se può essere più semplice come associazione, è come un cassetto con all'interno un certo numero di scomparti e ogni scomparto è numerato. ad esempio se creo (dichiaro) un array di caratteri di grandezza 6:
char testo[6]
e poi definisco
testo[0] = “S”
testo[1] = ”A”
testo[2] = “L”
testo[3] = ”U”
testo[4] = “T”
testo[5] = ”I”
anche se sembrano avere un senso compiuto se prese in sequenza potrei chiedere di stampare la serie 0,1,2,4,1 ottenendo “SALTA”. La variabile samples verrà riempita con 5 valori letti dal pin analogico di Arduino per poi calcolarne la media.
Nel loop, inizio eseguendo una serie di letture in successione e poi calcolo il valore medio ottenuto. Questo perché il termistore è molto sensibile per cui mediando 5 – 10 letture ottengo un valore più stabile e prossimo al reale.
Per fare questo uso qui una istruzione che fa parte della base della programmazione: il cosiddetto ciclo ***for*** (in italiano “per”) che essendo derivato dalla sintassi del linguaggio C++ si scrive così:

```
for (int i = 0 ; i < NUMERO ; i ++) {
  // istruzioni da eseguire
}
```

l'istruzione si compone di tre parti divise dal punto e virgola, la prima è l'assegnazione del punto di partenza di un contatore, se non dichiarato prima viene dichiarato qui come ***int*** = intero. Nel secondo blocco c'è il limite superiore a cui deve tendere il contatore, può essere un numero oppure una variabile contenete un numero (intero anch'esso). Qui valgono tutti gli operatori logici e le espressioni matematiche per cui potremo avere ***i = 10, i <= 10, i > 10*** o espressioni più complesse.
Il terzo blocco definisce l'incremento del contatore, i ++ è equivalente a ***i = i + 1***, anche qui sono possibili espressioni matematiche diverse anche se solitamente si usa un incremento o decremento lineare di valori.
Tutte le istruzioni racchiuse tra le parentesi graffe verranno ripetute ***i*** volte. Nel nostro caso il ciclo ***for*** verrà ripetuto 5 volte (NUMSAMPLES = 5) e il valore del contatore viene usato come indice dell'array (valori da 0 a 4) per memorizzare 5 diversi valori letti dal pin analogico: ***samples(i) = analogRead(THERMISTORPIN)*** con un intervallo di 10 millisecondi tra le letture.
Successivamente in un nuovo ciclo for riprendo uno alla volta i valori letti e li sommo nella variabile average per avere il totale che poi dividerò per il numero di campionature per avere il valore medio. Su questo valore eseguo tutti i calcoli matematici visti in precedenza per ottenere il dato finale che è la temperatura in °C

Abbiamo visto che collegando sensori e display ad Arduino, si rischia di rimanere a corto di uscite disponibili per cui vediamo come si può ovviare in parte a questo problema senza dover necessariamente cambiare modello di microcontrollore (ricordo che il modello mega 2560 ha 54 ingressi digitali e 16 analogici...). Useremo in questo esempio un convertitore seriale – parallelo che ci consentirà utilizzando solamente tre collegamenti ad Arduino, di indirizzare otto uscite diverse. Si tratta del chip 74HC595 vediamo come è fatto:

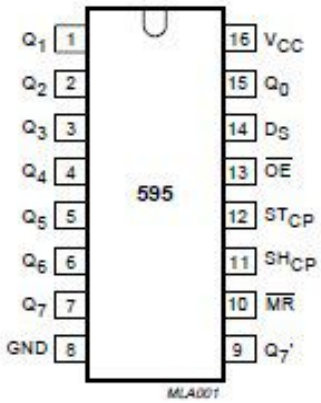


Fig.1 Pin configuration.

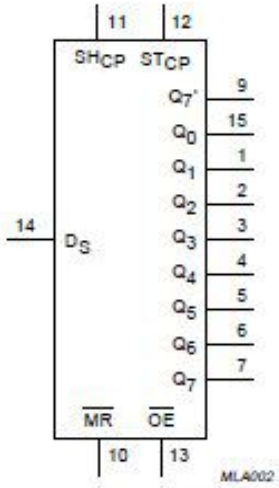


Fig.2 Logic symbol.

I 16 piedini (le 16 porte) del chip hanno le seguenti funzioni

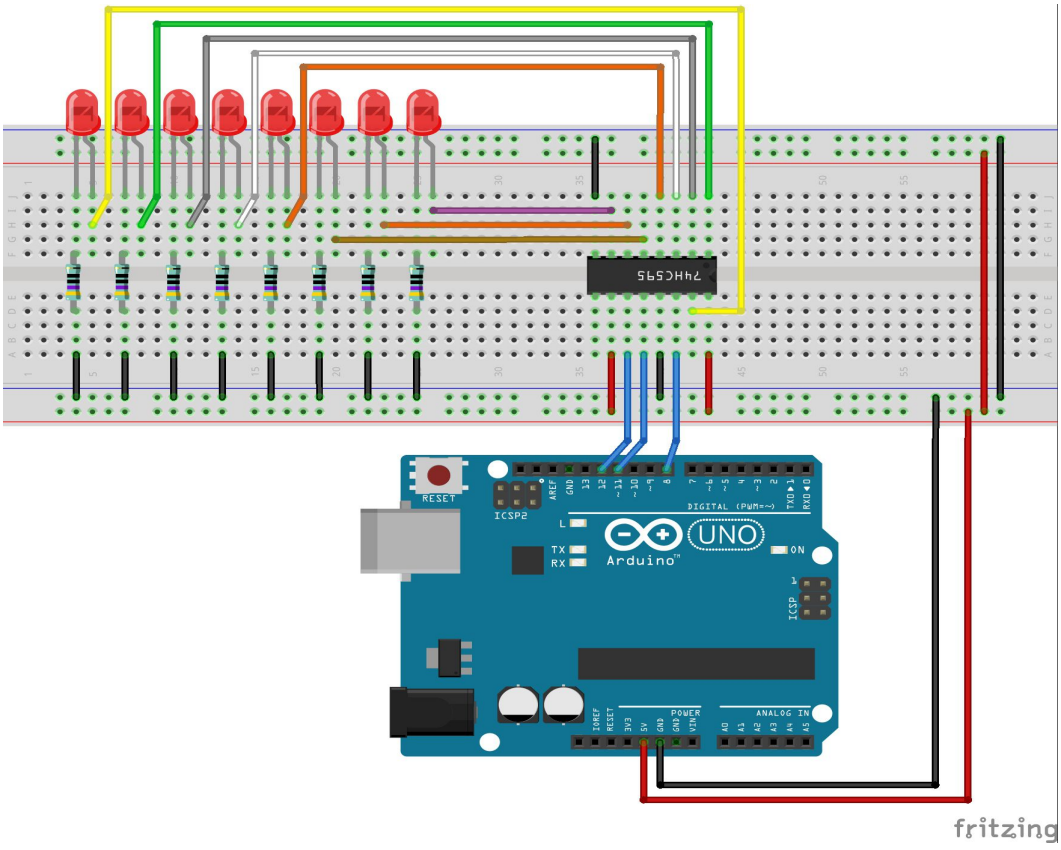
- ***1-7 e 15 (da Q1a Q7e Q0)***: porte di uscita, che vengono attivate o disattivate secondo le istruzioni ricevute da Arduino.
- ***8 (GND ground)***: collegamento di terra.
- ***9 (Q7S, serial out)***: porta di uscita seriale che può essere collegata alla porta di entrata di eventuali altri chip 74HC595 collegabili in cascata.
- ***10 (MR, master reset, active low)***: porta di reset, se la si pone in stato ***LOW*** cancella il byte memorizzato nello shift register. Per evitare problemi di solito viene tenuta ***HIGH*** e quindi alimentata con una tensione di +5V
- ***11 (SH_CP shift register clock pin)***: porta per l'attivazione della fase di trasferimento del byte da Arduino allo shift register.
- ***12 (ST:CP storage register clock pin, detta anche latch pin)***: porta per l'attivazione della fase di trasferimento del byte dallo shift register allo storage register, quando viene dichiarata ***LOW*** viene consentito lo spostamento, mentre quando e' ***HIGH*** viene impedito. E' una specie di interruttore, utilizzato per decidere il momento di spostamento del byte dal registro di entrata al registro di utilizzo. Se poi la porta 13 (la prossima porta OE,) è attiva (e cioè è ***LOW***), il trasferimento dei dati nello shift register coincide con l'attivazione/disattivazione delle porte di uscita.
- ***13 (OE output enable, active low)***: porta che consente l'utilizzo del byte per attivare o disattivare le porte di uscita. Quando e' ***LOW*** consente l'utilizzo del byte mentre quando è ***HIGH*** non ne consente l'utilizzo. Per limitare il numero di porte utilizzate da Arduino, questa porta viene normalmente lasciata attiva e cioè viene mantenuta in stato ***LOW*** e quindi collegata direttamente a terra.
- ***14 (DS data serial input)***: porta sulla quale viene fatto transitare (da Arduino al chip) il byte in formato seriale.

- ***16 (Vcc)*** alimentazione in corrente continua da un minimo di +2V a +6V al massimo

Nel progetto che realizzeremo, collegheremo le uscite a 8 led che si accenderanno in varie sequenze per ottenere dei giochi di luce. In molti progetti che si possono reperire su internet per l'utilizzo di questo integrato, vengono indicate come resistenze da collegare ai led quelle classiche da 220 Ohm.
Controlliamo la quantità totale di corrente totale erogata dal chip in questo modo, usando la solita legge di Ohm che recita:
I = V / R, dove ***V*** = +5V – 1,7 (valore caduta tensione dei led rossi) = 3,3V e ***R*** = 220 Ohm
per cui 3,3 / 220 = 0,015 A, moltiplicato per le 8 uscite disponibili 0,015 * 8 = 0,12A cioè **120 mA**

Se leggiamo il datasheet dell'integrato 74HC595 troveremo che la massima corrente erogabile risulta essere di circa **35 mA per ogni uscita** **ma con un massimo di 70 mA per tutto il chip**
nell'esempio qui sopra usiamo 120 mA: quasi il doppio del massimo erogabile dall'integrato, sicuramente per un po riuscirà a funzionare ma non avrà vita lunga. Nel caso in cui vogliamo avere tutte le uscite attive contemporaneamente, dovremo dividere i 70 mA massimi per le 8 uscite ottenendo 8,75 mA che è il valore che utilizzeremo per il calcolo della resistenza che sarà questa volta sempre secondo la formula della legge di Ohm:
R= V / I dove ***V*** = 3,3V come già visto e ***I*** = 0,00875 (espresso in Ampere), per cui avremo
R = 3,3 / 0,00875 = 377,14 Ohm. Questo è il valore minimo di resistenza che possiamo utilizzare per sfruttare al 100% le possibilità del chip ovviamente per allungargli la vita useremo un valore leggermente più alto, il primo valore disponibile nel nostro kit è 470 Ohm, useremo questo.

Realizziamo ora il circuito in questione



Vediamo ora uno sketch di esempio trovato sul sito ufficiale di Arduino su internet (anche questo indica nello schema resistenze da 220 Ohm!)

```
//************************************************************//
//  Name    : shiftOutCode, Hello World
//  Author  : Carlyn Maw,Tom Igoe, David A. Mellis
//  Date    : 25 Oct, 2006
//  Modified: 23 Mar 2010
//  Version : 2.0
//  Notes   : Code for using a 74HC595 Shift Register          //
//          : to count from 0 to 255
//************************************************************

////Pin connected to DS of 74HC595
int dataPin = 8;
//Pin connected to ST_CP of 74HC595
int latchPin = 11;
//Pin connected to SH_CP of 74HC595
int clockPin = 12;



void setup() {
  //set pins to output so you can control the shift register
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
  // count from 0 to 255 and display the number
  // on the LEDs
  for (int numberToDisplay = 0; numberToDisplay < 256; numberToDisplay++) {
    // take the latchPin low so
    // the LEDs don't change while you're sending in bits:
    digitalWrite(latchPin, LOW);
    // shift out the bits:
    shiftOut(dataPin, clockPin, MSBFIRST, numberToDisplay);

    //take the latch pin high so the LEDs will light up:
    digitalWrite(latchPin, HIGH);
    // pause before next value:
    delay(300);
  }
}
```

Nel ***setup*** si dichiarano i tre contatti come uscite e nel loop un ciclo ***for*** conta da 0 a 255 e ad ogni passaggio viene posto a LOW lo ***STCP*** per permettere il ricevimento di tutti i bit tramite l'istruzione ***shiftOut(DS, SHCP, DirezioneIngressoBit, SerieDiBit)***, quindi viene portato nuovamente ad HIGH lo ***STCP*** per trasferire i dati ricevuti uno dopo l'altro sulle otto uscite contemporaneamente

l'istruzione ***shiftOut(DS, SHCP, DirezioneIngressoBit, SerieDiBit)*** di Arduino ha quattro parametri:

1. numero del pin di Arduino collegato all'ingresso ***DS*** del chip 74HC595
2. numero del pin di Arduino collegato all'ingresso ***SHCP*** del chip 74HC595
3. sono consentiti solamente due valori, definiti dalle costanti interne di Arduino da MSBFIRST (Most Significant Bit first = prima il bit più significativo) e LSBFIRST (Least Significant Bit first= prima il bit meno significativo) che indicano rispettivamente se il primo bit (degli 8) che invieremo andrà mandato sull'uscita ***Q7*** oppure ***Q0*** del chip 74HC595.
4. Un numero che rappresenta gli 8 bit di dati da parallelizzare.