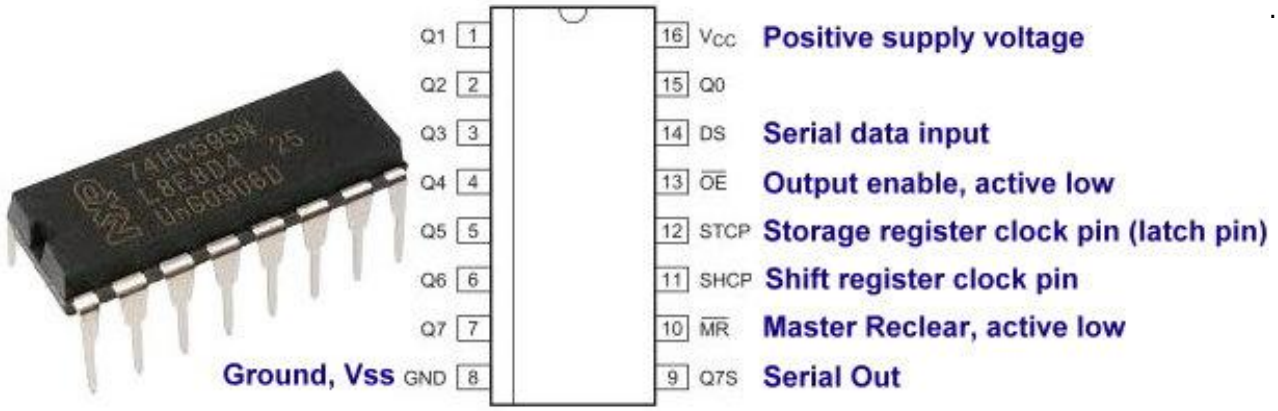
13 – il chip 74HC595, il moltiplicatore di porte – The pins multiplier (some notes at section end)



Nell'esercizio 12 abbiamo sfiorato uno dei limiti di Arduino uno, e cioe' il numero non elevato di porte di input/output. Tale limite ci ha costretto ad utilizzare porte "anomale", come la 0 e la 1 per riuscire a pilotare un display a 4 cifre. Il chip 74HC595 offre una soluzione a questo problema poiche', utilizzando tre sole porte Arduino puo' pilotare ben 8 porte, alle quali e' possibile collegare dei led o degli attuatori a basso assorbimento di energia.

Riducendo ai minimi termini le spiegazioni sul funzionamento di questo chip, e' possibile dire che Arduino invia al chip un byte la cui configurazione binaria (e cioe' gli 8 bit, che lo compongono) rappresentano, per stato (0 oppure 1) e posizione (dalla posizione 0 alla posizione 7), la situazione che si intende creare nelle 8 porte di uscita del chip (da Q0 a Q7). Di fatto Arduino invia un byte al chip che lo memorizza in un registro di entrata (*shift register*). Il byte viene poi trasferito in un registro di utilizzo detto *storage register* e da qui' utilizzato per attivare o disattivare, contemporaneamente, le 8 porte di uscita. I vari passaggi sono ovviamente tutti singolarmente pilotabili da Arduino.

I 16 piedini (le 16 porte) del chip hanno la seguente funzione:

- 1-7 e 15 (da Q1 a Q7 e Q0): porte di uscita, che vengono attivate o disattivate secondo le istruzioni ricevute da Arduino;
- 8 collegamento di terra;
- 9 (Q7S, *serial out*): porta di uscita seriale che puo' essere collegata alla porta di entrata di eventuali altri chip 74HC595 collegabili in cascata;
- 10 (MR, *master reclear, 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 tenuto HIGH e quindi alimentata con una tensione di 5 volt;
- 11(SHCP *shift register clock pin*): porta per l'attivazione della fase di trasferimento del byte da Arduino allo *shif register;*
- 12 (STCP *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 ne 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,) e' attiva (e cioe' e' 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 e' HIGH non ne consente l'utilizzo. Per limitare il numero di porte utilizzate da Arduino, questa porta viene normalmente lasciata attiva e cioe' viene mantenuta in stato LOW e quindi collegata direttamente a terra

- 14(DS, *Serial data input*) porta sulla quale viene fatto transitare (da Arduino al chip) il byte di configurazione
- 16 porta di alimentazione, da collegare ad una tensione di 5 volt cc.

Come gia' detto, per pilotare il chip Arduino puo' limitarsi ad utilizzare tre sole porte:

- una per attivare (o meglio consentire) l'invio del byte da Arduino al chip, e quindi da collegare alla porta numero 11 del chip (SHCP, *shift register clock pin*)
- una per inviare fisicamente al chip il byte di configurazione e quindi da collegare alla porta 14 del chip (DS, *Serial data input*)
- una per trasferire il byte dalla memoria di entrata alla memoria di utilizzo del chip. Deve essere collegata alla *latch pin*, e cioe' allla porta 12 del chip.

Nel nostro esercizio, oltre alle tre porte di cui sopra, da collegare rispettivamente alle porte 13, 10 e 12 di Arduino, dobbiamo collegare a terra le porte 8 e 13 del chip e all'alimentazione di 5 volt cc, le porte 10 e 16. La porta 13 (il cancello) va collegata terra per consentire l'attivazione delle porte di output nel momento in cui la *latch pin* viene dichiarata HIGH mentre la porta 10 va messa sotto tensione per evitare eventuali inaspettati reset.

Sotto l'aspetto della programmazione, il trasferimento del byte da Arduino al chip viene effettuato attraverso l'istruzione shiftOut, cosi' strutturata:

shiftOut (*porta dati, porta latch, modalita' di utilizzo, byte di configurazione****);***

dove:

- porta dati: numero della porta di Arduino collegata alla porta 14 (DS) del chip;
- porta latch: numero della porta di Arduino collegata alla porta 11 (SHCP, oppure LATCH) del chip;
- modalita' di utilizzo: puo' assumere i valori: **MSBFIRST** or **LSBFIRST** (Most Significant Bit First (il bit piu' significativo per primo oppure Least Significant Bit First il bit meno significativo per primo) per scegliere l'ordine di assegnazione dei bit alle porte di output (da 0 a 7 oppure da 7 a 0);
- byte di configurazione: il byte che Arduino trasmette al chip;

In questo esercizio il chip pilotera' 8 led, che fara' accendere e spegnere secondo una configurazione di byte predefiniti contenuti, in forma binaria, in una tabella del programma chiamata "sequenza". Terminata la sequenza iniziale (led accesi uno dopo l'altro da sinistra a destra e da destra a sinistra) arduino prosegue accendendo i led in sequenza casuale

Nota: Questo esercizio e questa nota sono parte di una serie che vede protagonisti Arduino ed alcuni dei componenti ad esso collegabili. Per la maggior parte degli esercizi e' anche disponibile un filmato su youtube.

- Esercizi facenti parte della raccolta
- Filmati presenti su youtube
- Informazioni su arduino e sui componenti collegabili (PDF scaricato nell'area di download)
- Breve manuale di programmazione (PDF scaricato nell'area di download)

Per eventuali chiarimenti o suggerimenti sul contenuto di questa scheda scrivere a giocarduino@libero.it

Here some notes about this project, translated by google translator 

In project 12 (the timer) we have touched one of the Arduino Uno limits: the not high number of input/output pins. This limit has forced us to use "anomalous" pins, such as 0 and 1, to be able to

drive a 4-digit display. The 74HC595 chip offers a solution to this problem because, using three pins, Arduino can drive 8 pins, to which can connect LEDs or low energy absorption actuators.

By reducing to a minimum the explanations about the functioning of this chip, we can say that Arduino sends a byte whose binary configuration (the 8 bits which compose it) represent, for state (0 or 1) and location (from position 0 to position 7), the situation that we intend create on chip output pins (Q0 to Q7). Arduino sends a byte, and chip stores it in a register (*shift register*). The byte is then transferred to a *storage register* and from here used to enable or disable, at the same time, the output pins. The various steps are obviously all individually controllable by Arduino.

Chip 16 legs (16 pins) have the following function:

- 1-7 and 15 (Q1 to Q7 and Q0): output pins, activated or deactivated according to the instructions received from Arduino;
- 8 ground;
- 9 (Q7S, serial out): serial output that can be connected to the input pin of an other 74HC595 chip cascadable;
- 10 (MR, master reclear, active low): reset pin; if is LOW resets the byte stored in shift register. To avoid problems usually is held HIGH and so fed with a 5 volts voltage;
- 11 (SHCP shift register clock pin): to activate the byte transfer phase, from Arduino to the shift register;
- 12 (STCP storage register clock pin, also called pin latch) to activate the byte transfer phase, from shift register to storage register: when is declared LOW is allowed to move, while when is HIGH it is prevented. Is like a switch, used to decide the byte displacement from the input register to the use register. If then the pin 13 (the next pin OE,) is "active" (is LOW), the transfer in shift register coincides with the activation / deactivation of output ports.
- 13 (OE enable output, active low): allows the use of byte to activate or deactivate the output ports. When is LOW allows the use of the byte while when is HIGH does not allow the use. To limit the number of ports used by Arduino, this port is normally activated (and so in LOW state, connected directly to ground)
- 14 (DS, Serial data input) port on which pass (from Arduino to chip) the configuration byte
- 16 power port, to be connected to a 5 volts DC.

As already said, Arduino can drive chip by using only three pins:

- one to activate (or better allow) sending byte from Arduino to the chip, and then to be connected to pin number 11 of the chip (SHCP, shift register clock pin)
- one for physically sent to chip the configuration byte and then be connected to pin 14 on chip (DS, Serial data input)
- one to transfer byte from the input memory to the use memory. It must be connected to the pin latch, pin 12 on chip.

In our project, in addition to the three ports mentioned above, to be connected respectively to the arduino pins 13, 10 and 12, we have to connect to ground the chip pins 8 and 13 and to a 5 volts DC power supply, pins 10 and16. Pin 13 on chip (the gate) should be connected to the ground to allow the activation of output pins at a time when the latch pin is declared HIGH while the pin 10 to be put under tension in order to avoid any unexpected reset.

About programming aspects, the bytes transfer from Arduino to chip is carried out through the shiftOut istruction, structured as follows:

shiftOut *(data pin, latch pin, using mode, configuration byte);*

where:

- data pin: number of Arduino pin connected to pin 14 (DS) on chip;
- latch pin: number of Arduino pin connected to pin 11 (SHCP, or LATCH) on chip;

- using mode: can assume the following values: **MSBFIRST** or **LSBFIRST** (Most Significant Bit First or Least Significant Bit First) to choose the bit assignment order on output pins (from 0 to 7 or from 7 to 0);
- configuration byte: byte from Arduino to chip;

In this project, the chip will drive 8 leds, that will turn on and off according to a configuration of predefined bytes contained, in binary form, in a program array called "sequenza" After the initial sequence (leds lit one after another from left to right and from right to left) Arduino continues turning on the leds in a random mode

Note: This project and this note is part of a series that sees, as main characters, Arduino and some of connectable components. For most projects there is also a video on youtube.

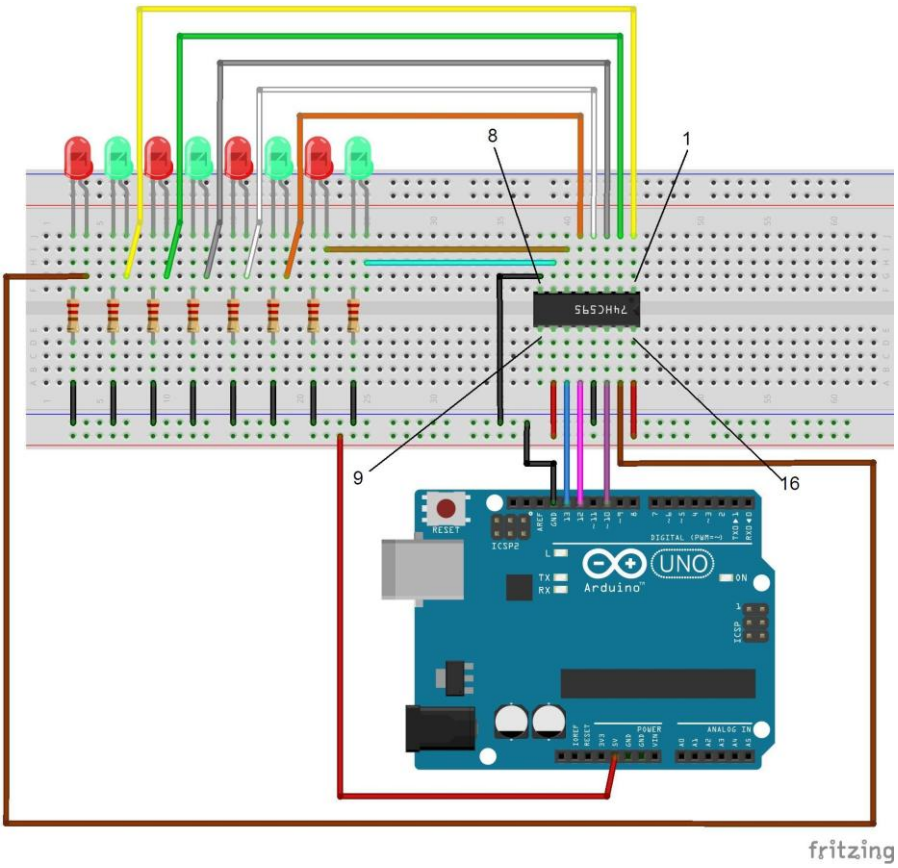
- Projects collection
- Movies on youtube
- About Arduino and components (italian; pdf will be downloaded in your download area
- Quick programming guide (almost english; pdf will be downloaded in your download area)

For any questions or suggestions about this note (and on its english translation), please write to giocarduino@libero.it (simple words and short sentences, please)

Materiali

- una breadboard
- 8 led
- 8 resistenze da 220 ohm
- Un chip 74HC595
- Un po' di cavetteria

Schema



Programma

```
/*  Attenzione: facendo il copia/incolla dal PDF all'IDE si perde la formattazione del testo. Per
rendere piu' facilmente leggibile il programma e' opportuno formattarlo subito dopo il trasferimento
nell'IDE, premendo CTRL+T .
-------------------------------------------------------------------------------------------
Warning: cut&paste from PDF to IDE loses formatting. To restore it press CTRL + T.
-------------------------------------------------------------------------------------------
*/

unsigned char i = 0;           // indice per gestire la sequenza di attivazione/disattivazione porte
unsigned char col = 0;       // indice utilizzato per scorrere la tabella di byte predefiniti
unsigned char configurazione = 0;  // variabile in cui viene inserito il byte di configurazione
// porte utilizzato dalla routine gestisciporte
//
unsigned char sequenza[8] = {        // tabella dei byte per la sequenza iniziale
B00000001, B00000010, B00000100, B00001000, B00010000, B00100000, B01000000, B10000000};
/* nota:   la medesima tabella, con il medesimo contenuto di byte, avrebbe potuto essere definita
anche cosi:   unsigned char sequenza[8] = { 1, 2, 4, 8, 16, 32, 64, 128};
oppure
unsigned char sequenza[8] = { 0x01, 0x02, 0x04, 0x8, 0x10, 0x20, 0x40, 0x80};
//
notes: this array, with the same content of bytes, could also be defined as follows:
unsigned char sequence [8] = {1, 2, 4, 8, 16, 32, 64, 128};
or
unsigned char sequence [8] = {0x01, 0x02, 0x04, 0x8, 0x10, 0x20, 0x40, 0x80};

*/
//Definizione delle variabili con il numero delle porte da usare per collegare il chip ad Arduino:
int latchPin = 12;  //numero della porta da connettere alla porta ST_CP (la 12) del chip 74HC595
int clockPin = 13; //numero della porta da connettere alla porta SH_CP(la 11) del chip 74HC595
int dataPin = 10;  //numero della porta da connettere alla porta DS(la 14) del chip 74HC595
//
//*****************   routine di gestione delle porte ************************
// ************************* pins management routine ***************************
// la routine pilota il chip trasmettendo il byte contenuto in "configurazione"

void gestisciporte(void)
{
  digitalWrite(latchPin, LOW); // attiva (mette in stato di low) la porta di latch per consentire
  // l'invio di un byte
  shiftOut(dataPin, clockPin, MSBFIRST, configurazione); // invia fisicamente il  byte al chip
  digitalWrite(latchPin, HIGH); // disattiva (mette in stato high) la porta di latch per
  // comunicare l'avvenuto invio del byte e predisporre il chip ad una nuova attivazione
  delay(200);  // attende 2 decimi di secondo prima di tornare al punto di programma dal quale e'
  // stata chiamata
}
//
void setup()
{
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  for (i = 0; i < 16; i++)             // ciclo di for per il lancio della sequenza iniziale di byte
  {
    col = i;                                // predispone l'indice di scorrimento della tabella
    if (i > 7) col = 8 - (i -7);       // se e' gia' stata completata la tabella (i>7), predispone
// l'indice per scorrerla all'indietro
    configurazione = sequenza[col]; // inserisce in configurazione il byte da far utilizzare al chip
    gestisciporte ();                 // lancia la routine di gestione delle porte
  }
}
//
void loop()
{   // terminata la sequenza iniziale (lanciata nel setup), il programma prosegue
//  generando byte a caso e quindi illuminando e spegnendo i led senza una sequenza predefinita
  configurazione = random (256);  // genera un numero a caso compreso tra 0 e 255 e cioe'
//  genera un byte a caso
  gestisciporte();                // lancia la routine di gestione delle porte
}
```