Arduino + Matriz RGB

Idea

Mostrar una visualización (tipo vúmetro) que pueda cambiar según una entrada de audio, en una matriz de 8x8 LED RGB.

Componentes

- 1 matriz RGB 8x8 Modelo ORM 2088RGB-5
- 4 registros de desplazamiento 74HC595
- 24 resistencias de 220 ohm
- 1 resistencia de 12K ohm

Los registros de desplazamiento

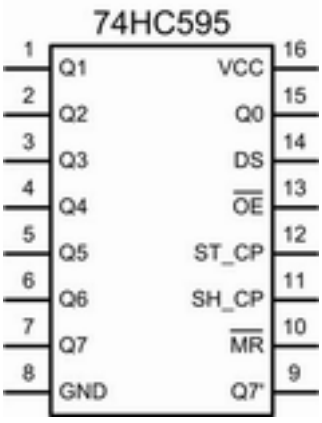
Los registros de desplazamiento son circuitos integrados que permiten controlar más de una salida de datos en paralelo, a través de una sola entrada, en donde los datos van en serie.

Una característica útil es que se pueden encadenar la salida de uno con la entrada de otro y así lograr controlar más salidas. Por ejemplo, si conectamos 3 “74595” encadenados, cada “74595” posee 8 salidas, entonces tenemos 8x3=24 salidas.

* Tutorial de funcionamiento detallado: http://www.protostack.com/blog/2010/05/introduction-to-74hc595-shift-register-controlling-16-leds/

* Especificación completa: http://www.nxp.com/documents/data_sheet/74HC_HCT595.pdf

* Representación de los pines:



Q0 hasta Q7 representan las 8 salidas en ese orden.
VCC es la alimentación positiva, GND (ground) el negativo.
DS (o SER en algunas especificaciones), en el pin 14 es la entrada de datos
El pin 11 (SH_CP) es el pin de clock
El pin 12 (ST_CP or RCLK en algunas especificaciones) es el pin de latch

Q7' (pin 9) es la salida de datos (la que conectada a la entrada de otros registros de desplazamiento permite encadenarlos y duplicar la cantidad de salidas).
OE (pin 13) es "output enable"
MR (pin 10) es "master reset". Conectado a Vcc por defecto.

Cómo es una matriz de leds internamente

La matriz posee 64 LED RGB. En total hay 32 pines, 8 por cada color por cada fila, y 8 pines comunes que representan las columnas. Es de ánodo común, lo que quiere decir que debemos conectar los pines que representan las filas (los comunes) a tensiones positivas, y en donde se envían los datos, tienen que enviarse tensiones negativas para hacer que se enciendan los LED.

La especificación de los pines de la matriz (modelo ORM 2088RGB-5) es la siguiente:

- pin 1 a 8: cátodos color rojo
- pin 9 a 16: cátodos color azul
- pin 17 a 20: ánodos de columnas 1 a 4
- pin 21 a 28: cátodos color verde
- pin 29 a 32: ánodos de columnas 5 a 8

Alternativas para controlarla

- **función shiftOut()**: esta función que provee Arduino (www.arduino.cc/en/Reference/ShiftOut) permite escribir un byte de manera serie en una sola salida. Con esto se podría controlar las filas y columnas y poner un 1 o un 0 en cada una de ellas, y así lograr que se encienda un color determinado, en una fila determinada, en una columna determinada en la matriz. Una desventaja que posee este enfoque es que siempre se asignaría 1 o 0, es decir, máxima intensidad en cada led o nula, con lo cual, sólo se tendrían los 3 colores (rojo, azul, verde) y las mezclas de ellos. No se podrían formar o realizarlos manualmente puede resultar muy complicado.
- **librería ShiftPWM**: (http://www.elcojacobs.com/shiftpwm/) posee varias utilidades para setear salidas que están pensadas para ser utilizadas con registros de desplazamiento mediante PWM (pulse width modulation), lo que resuelve el problema del punto anterior. La dificultad que tiene esta librería es que está diseñada para controlar un led por cada salida de cada registro de desplazamiento. Para el caso de la matriz, es poco práctico ya que tenemos 8x8x3 (192) leds en total, y cada registro de desplazamiento 74595 puede controlar 8 salidas, con lo cual serían necesarios 192/8 (24) registros de desplazamiento. Otro enfoque sería modelar una fila de cada matriz (8*3) leds, 3 registros de desplazamiento, y hacer que desde el programa Arduino se renderice en cada loop cada una de las 8 filas, una por vez, a una velocidad que resulte imperceptible para el ojo humano. Basado en las pruebas que realizamos no logramos esto último, con lo cual descartamos esta alternativa.
- **SPI (Serial Peripheral Interface) y Timers de Arduino**. Esta alternativa es la que elegimos, siguiendo el tutorial (http://www.instructables.com/id/64-pixel-RGB-LED-Display-Another-Arduino-Clone/). Es posible controlar filas, columnas, e intensidad

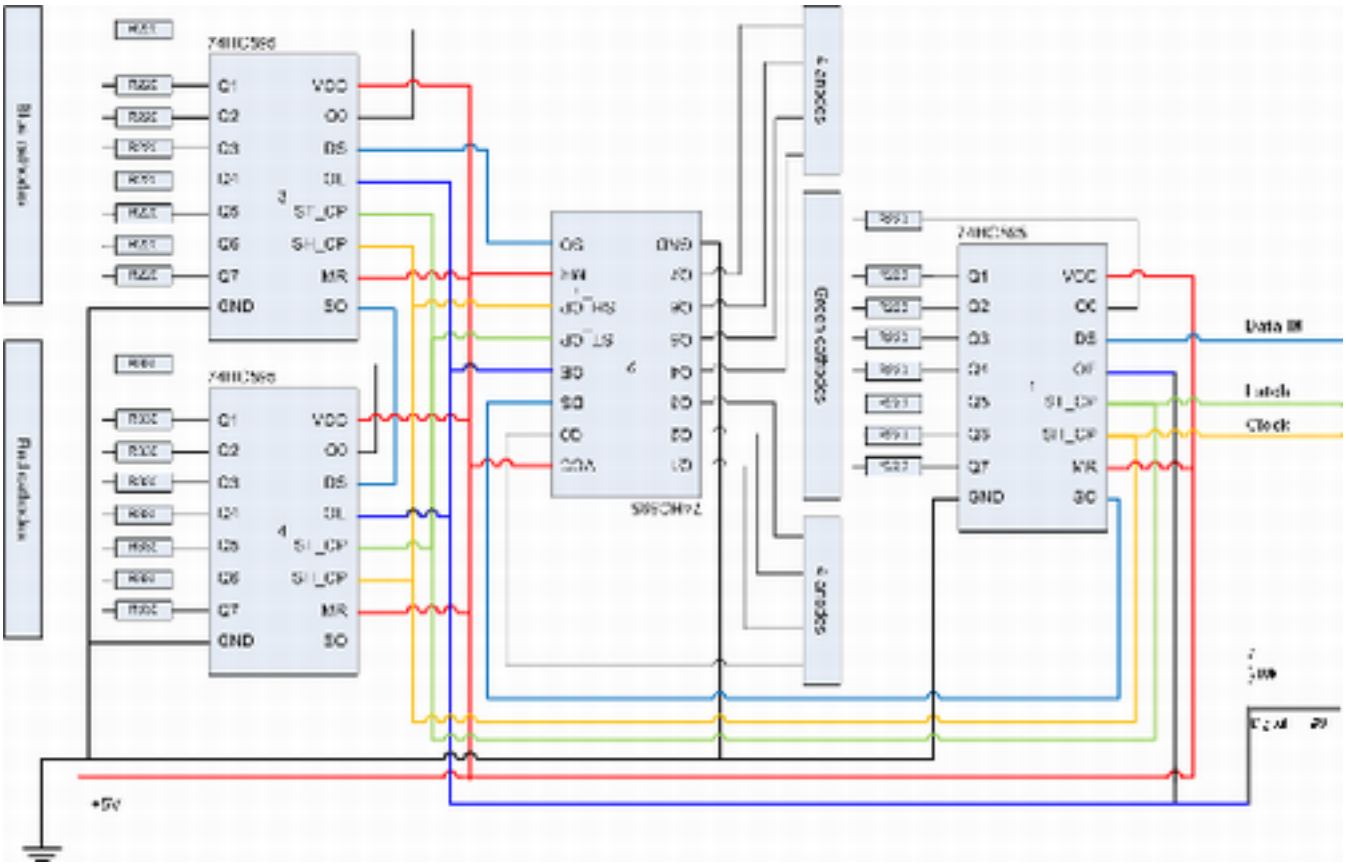
de cada color individualmente. Esto se debe a su implementación en más bajo nivel. Está pensada para ser controlada con 4 registros de desplazamiento (uno para cada color, y uno para las columnas)

Manejo de entrada de audio

Con respecto a la entrada de audio, para mostrar datos que tengan algún sentido en la matriz necesitamos dividir las frecuencias de la señal de audio. Para lo cual observamos dos alternativas:

- utilizar hardware, particularmente el circuito integrado MSEGQ7 (https://www.sparkfun.com/datasheets/Components/General/MSGEQ7.pdf), que genera 7 bandas de salida. La desventaja de utilizar esto era agregar un circuito extra, además de conseguir el integrado.
- utilizar software, por medio de FFT (Fast Fourier Transform), que consiste en utilizar un implementación de esta función matemática. La implementación que utilizamos fue FFFT (http://neuroelec.com/2011/03/fft-library-for-arduino/), que es un *wrapper* para Arduino de una implementación Assembler.

Circuito (esquemático)



Código

```
/*
Vumetro para Matriz 8x8 de LEDs RGB.
```

```
Codigo basado en:
  * Visualizacion de la matriz: http://www.instructables.com/id/64-pixel-RGB-LED-Display-Another-Arduino-Clone/
  * Manejo de la entrada de audio: http://neuroelec.com/2011/03/fft-library-for-arduino/
*/

#include <stdint.h>
#include <ffft.h>

#define  IR_AUDIO  0       // Pin entrada de audio

#define __spi_clock 13     // Pin de clock
#define __spi_latch 10     // Pin de latch
#define __spi_data 11      // Pin de salida de datos
#define __display_enable 9  // Pin 9 (para controlar encendido/apagado de la matriz completa)
#define __rows 8           // Cantidad de filas
#define __max_row __rows-1
#define __columns 8        // Cantidad de columnas
#define __max_led __columns-1
#define __brightness_levels 32 // 0...15 above 28 is bad for ISR ( move to timer1, lower irq freq ! )
#define __max_brightness __brightness_levels-1
#define __fade_delay 4

#define __TIMER1_MAX 0xFFFF // 16 bit CTR
#define __TIMER1_CNT 0x0130 // 32 levels --> 0x0130; 38 --> 0x0157 (flicker)

#include <avr/interrupt.h>
#include <avr/io.h>

byte brightness_red[__columns][__rows];       /* memory for RED LEDs */
byte brightness_green[__columns][__rows];      /* memory for GREEN LEDs */
byte brightness_blue[__columns][__rows];       /* memory for BLUE LEDs */

volatile byte position = 0;
volatile long zero = 0;

int16_t capture[FFT_N];                    /* Wave captureing buffer */
complex_t bfly_buff[FFT_N];                /* FFT buffer */
uint16_t spektrum[FFT_N/2];                /* Spectrum output buffer */

float frequencyAdjustValues[] = {0.8,1.5,1.5,1.5,3,2.5,3,4};

void setup(void) {
  randomSeed(555);
  pinMode(__spi_clock,OUTPUT);
  pinMode(__spi_latch,OUTPUT);
  pinMode(__spi_data,OUTPUT);
```

```
  pinMode(__display_enable,OUTPUT);
  digitalWrite(__spi_latch,HIGH);
  digitalWrite(__spi_data,HIGH);
  digitalWrite(__spi_clock,HIGH);
  setup_hardware_spi();
  delay(10);
  set_matrix_rgb(0,0,0);                    /* set the display to BLACK */
  setup_timer1_ovf();                              /* enable the framebuffer display */

  Serial.begin(57600);
  adcInit();
  adcCalb();
}

void loop(void) {
  vumeter();
}

void vumeter() {
  if (position == FFT_N) {
    fft_input(capture, bfly_buff);      // prepara la funcion FFFT
    fft_execute(bfly_buff);            // ejecuta la funcion FFFT
    fft_output(bfly_buff, spektrum);    // guarda los resultados en el array spektrum (64
valores)

    for (int i = 0; i < __columns; i++) {
     // como tenemos 64 bandas de frecuencia, y 8 columnas, promediamos de a 8 en
cada vuelta
      float bandAverage = ((float)spektrum[8*i] + (float)spektrum[8*i+1] + (float)
spektrum[8*i+2] + (float)spektrum[8*i+3] + (float)spektrum[8*i+4] + (float)
spektrum[8*i+5] + (float)spektrum[8*i+6] + (float)spektrum[8*i+7]) / 8;
     // el promedio de cada banda queda ponderado por unos ajustes para que la escala
sea mas adecuada
      float audioIn = map(bandAverage * frequencyAdjustValues[i], 0, 7, 0, 8);
      Serial.print(bandAverage);
      Serial.print("-");
      for (int j = 0; j < __rows; j++) {
       if (j < audioIn) {
         set_led_rgb(i, j, 255, 0, 0);
       } else {
         set_led_rgb(i, j, 0, 0, 5);
       }
      }
    }
    Serial.println("");
    position = 0;
  }
}
```

```
// Funciones utilitarias para el manejo de LEDs en la matriz

void set_led_red(byte row, byte led, byte red) {
  brightness_red[row][led] = red;
}

void set_led_green(byte row, byte led, byte green) {
  brightness_green[row][led] = green;
}

void set_led_blue(byte row, byte led, byte blue) {
  brightness_blue[row][led] = blue;
}

void set_led_rgb(byte row, byte led, byte red, byte green, byte blue) {
  set_led_red(row,led,red);
  set_led_green(row,led,green);
  set_led_blue(row,led,blue);
}

void set_matrix_rgb(byte red, byte green, byte blue) {
  byte ctr1;
  byte ctr2;
  for(ctr2 = 0; ctr2 <= __max_row; ctr2++) {
    for(ctr1 = 0; ctr1 <= __max_led; ctr1++) {
      set_led_rgb(ctr2,ctr1,red,green,blue);
    }
  }
}

void set_row_rgb(byte row, byte red, byte green, byte blue) {
  byte ctr1;
  for(ctr1 = 0; ctr1 <= __max_led; ctr1++) {
     set_led_rgb(row,ctr1,red,green,blue);
  }
}

void set_column_rgb(byte column, byte red, byte green, byte blue) {
  byte ctr1;
  for(ctr1 = 0; ctr1 <= __max_row; ctr1++) {
     set_led_rgb(ctr1,column,red,green,blue);
  }
}

void set_row_hue(byte row, int hue) {
  byte ctr1;
  for(ctr1 = 0; ctr1 <= __max_led; ctr1++) {
     set_led_hue(row,ctr1,hue);
  }
```

```
}

void set_column_hue(byte column, int hue) {
  byte ctr1;
  for(ctr1 = 0; ctr1 <= __max_row; ctr1++) {
     set_led_hue(ctr1,column,hue);
  }
}
void set_matrix_hue(int hue) {
  byte ctr1;
  byte ctr2;
  for(ctr2 = 0; ctr2 <= __max_row; ctr2++) {
    for(ctr1 = 0; ctr1 <= __max_led; ctr1++) {
      set_led_hue(ctr2,ctr1,hue);
    }
  }
}

void set_led_hue(byte row, byte led, int hue) {
  // see wikipeda: HSV
  float S=100.0,V=100.0,s=S/100.0,v=V/100.0,h_i,f,p,q,t,R,G,B;

    hue = hue%360;
    h_i = hue/60;
    f = (float)(hue)/60.0 - h_i;
    p = v*(1-s);
    q = v*(1-s*f);
    t = v*(1-s*(1-f));

    if      ( h_i == 0 ) {
      R = v;
      G = t;
      B = p;
    }
    else if ( h_i == 1 ) {
      R = q;
      G = v;
      B = p;
    }
    else if ( h_i == 2 ) {
      R = p;
      G = v;
      B = t;
    }
    else if ( h_i == 3 ) {
      R = p;
      G = q;
      B = v;
    }
```

```
    else if ( h_i == 4 ) {
      R = t;
      G = p;
      B = v;
    }
    else                {
      R = v;
      G = p;
      B = q;
    }  set_led_rgb(row,led,byte(R*(float)(__max_brightness)),byte(G*(float)
(__max_brightness)),byte(B*(float)(__max_brightness)));
}

void set_row_byte_hue(byte row, byte data_byte, int hue) {
  byte led;
  for(led = 0; led <= __max_led; led++) {
    if( (data_byte>>led)&(B00000001) ) {
      set_led_hue(row,led,hue);
    }
    else {
      set_led_rgb(row,led,0,0,0);
    }
  }
}

// Funciones relacionadas a hardware SPI y Timers

void setup_hardware_spi(void) {
  byte clr;
  // spi prescaler:
  // SPI2X SPR1 SPR0
  //   0    0    0    fosc/4
  //   0    0    1    fosc/16
  //   0    1    0    fosc/64
  //   0    1    1    fosc/128
  //   1    0    0    fosc/2
  //   1    0    1    fosc/8
  //   1    1    0    fosc/32
  //   1    1    1    fosc/64
  SPCR |= ( (1<<SPE) | (1<<MSTR) ); // enable SPI as master
  //SPCR |= ( (1<<SPR1) ); // set prescaler bits
  SPCR &= ~ ( (1<<SPR1) | (1<<SPR0) ); // clear prescaler bits
  clr=SPSR; // clear SPI status reg
  clr=SPDR; // clear SPI data reg
  SPSR |= (1<<SPI2X); // set prescaler bits
  //SPSR &= ~(1<<SPI2X); // clear prescaler bits
}
```

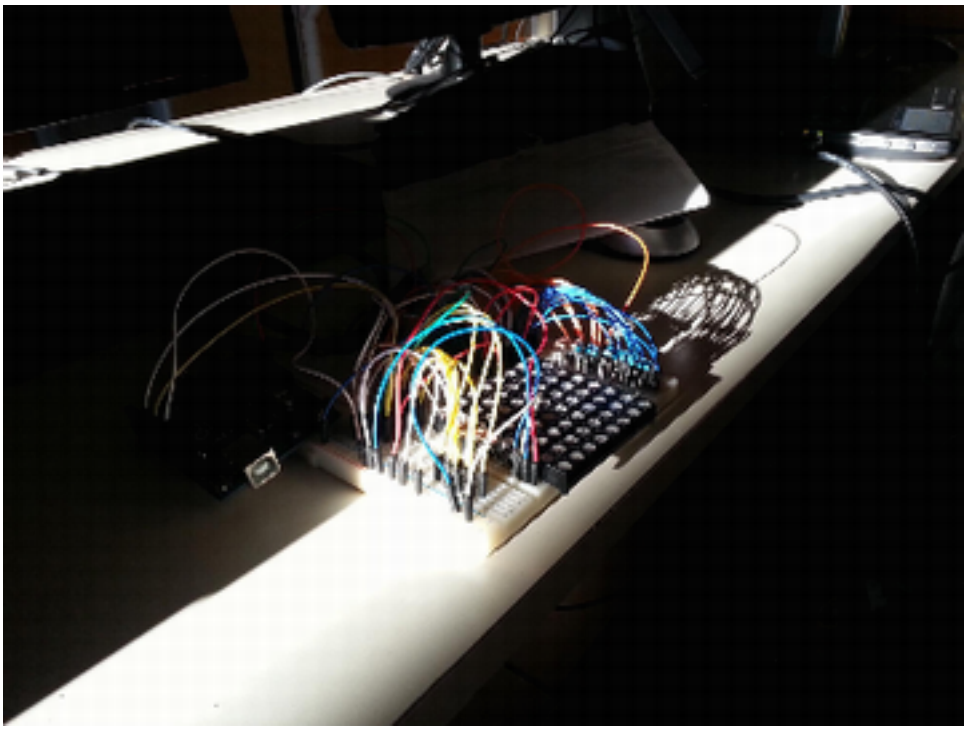
```
void setup_timer1_ovf(void) {
  // Arduino runs at 16 Mhz...
  // Timer1 (16bit) Settings:
  // prescaler (frequency divider) values:   CS12    CS11   CS10
  //                                  0      0     0   stopped
  //                                  0      0     1     /1
  //                                  0      1     0     /8
  //                                  0      1     1     /64
  //                                  1      0     0     /256
  //                                  1      0     1     /1024
  //                                  1      1     0     external clock on T1 pin, falling edge
  //                                  1      1     1     external clock on T1 pin, rising edge
  //
  TCCR1B &= ~ ( (1<<CS11) );
  TCCR1B |= ( (1<<CS12) | (1<<CS10) );
  //normal mode
  TCCR1B &= ~ ( (1<<WGM13) | (1<<WGM12) );
  TCCR1A &= ~ ( (1<<WGM11) | (1<<WGM10) );
  //Timer1 Overflow Interrupt Enable
  TIMSK1 |= (1<<TOIE1);
  TCNT1 = __TIMER1_MAX - __TIMER1_CNT;
  // enable all interrupts
  sei();
}


ISR(TIMER1_OVF_vect) { /* Framebuffer interrupt routine */
  TCNT1 = __TIMER1_MAX - __TIMER1_CNT;
  byte cycle;

  digitalWrite(__display_enable,LOW); // enable display inside ISR

  for(cycle = 0; cycle < __max_brightness; cycle++) {
    byte led;
    byte row = B00000000;  // row: current source. on when (1)
    byte red;              // current sinker, on when (0)
    byte green;                    // current sinker, on when (0)
    byte blue;             // current sinker, on when (0)

    for(row = 0; row <= __max_row; row++) {

      red = B11111111;             // off
      green = B11111111;    // off
      blue = B11111111;            // off

      for(led = 0; led <= __max_led; led++) {
        if(cycle < brightness_red[row][led]) {
          red &= ~(1<<led);
        }
```

```
        if(cycle < brightness_green[row][led]) {
          green &= ~(1<<led);
        }
        if(cycle < brightness_blue[row][led]) {
          blue &= ~(1<<led);
        }
      }

      digitalWrite(__spi_latch,LOW);
      // aca se controla el orden de los registros de desplazamiento:
      // para nuestro circuito es: verde -> rojo -> azul -> columnas
      spi_transfer(B00000001<<row);
      spi_transfer(blue);
      spi_transfer(red);
      spi_transfer(green);
      digitalWrite(__spi_latch,HIGH);
    }
  }
  digitalWrite(__display_enable,HIGH);    // disable display outside ISR
}

byte spi_transfer(byte data) {
  SPDR = data;                    // Start the transmission
  while (!(SPSR & (1<<SPIF))) {}  // Wait the end of the transmission
  return SPDR;                    // return the received byte. (we don't need that here)
}

// Funciones para inicializar FFFT

void adcInit(){
  /*  REFS0 : VCC use as a ref, IR_AUDIO : channel selection, ADEN : ADC Enable,
ADSC : ADC Start, ADATE : ADC Auto Trigger Enable, ADIE : ADC Interrupt Enable,
ADPS : ADC Prescaler  */
  // free running ADC mode, f = ( 16MHz / prescaler ) / 13 cycles per conversion
  ADMUX = _BV(REFS0) | IR_AUDIO; // | _BV(ADLAR);
  //ADCSRA = _BV(ADSC) | _BV(ADEN) | _BV(ADATE) | _BV(ADIE) | _BV(ADPS2) |
_BV(ADPS1) //prescaler 64 : 19231 Hz - 300Hz per 64 divisions
  ADCSRA = _BV(ADSC) | _BV(ADEN) | _BV(ADATE) | _BV(ADIE) | _BV(ADPS2) |
_BV(ADPS1) | _BV(ADPS0); // prescaler 128 : 9615 Hz - 150 Hz per 64 divisions, better
for most music
  sei();
}

void adcCalb(){
  Serial.println("Start to calc zero");
  long midl = 0;
  // get 2 meashurment at 2 sec
  // on ADC input must be NO SIGNAL!!!
  for (byte i = 0; i < 2; i++)
```

```
  {
    position = 0;
    delay(100);
    midl += capture[0];
    delay(900);
  }
  zero = -midl/2;
  Serial.println("Done.");
}

// free running ADC fills capture buffer
ISR(ADC_vect) {
  if (position >= FFT_N)
    return;

  capture[position] = ADC + zero;
  if (capture[position] == -1 || capture[position] == 1)
    capture[position] = 0;

  position++;
}
```

Fotos

Todo conectado. Ahora, como prendo una fila?



Y un dia, se hizo la luz...

Dificultades

- La especificación de la matriz según hojas de datos consultadas en Internet no coincidía con la real, por lo que se tuvo que probar cada uno de los pines manualmente
- Idas y vueltas con el programa, sobre el uso o no de librerías
- Varios de los circuitos consultados conectaban los pines 13 (output enable) de cada registro de desplazamiento a GND, pero para el programa que hacía uso de SPI y timers (la solución elegida) esto hacía que una fila de la matriz se muestre más brillante que las otras. La solución estaba en los comentarios del siguiente post (http://blog.spitzenpfeil.org/wordpress/2008/09/08/8x8-rgb-matrix-2nd-arduino-project/) y consistía en usar un pin del Arduino (pin 9) conectado a todos los pines 13 de los registros, con una resistencia de *pull-up* (12K ohm) conectada a Vcc.

Referencias

- http://www.solderlab.de/index.php/led-projects/rgb-matrix-12x8
- http://blog.fraktus.com/?p=702
- http://www.instructables.com/id/64-pixel-RGB-LED-Display-Another-Arduino-Clone/
- http://francisshanahan.com/index.php/2009/how-to-build-a-8x8x3-led-matrix-with-pwm-using-an-arduino/
- http://www.arduteka.com/2012/02/tutorial-arduino-0008-matriz-led-8x8-bicolor-74ch595/
- http://tronixstuff.wordpress.com/2010/09/28/moving-forward-with-arduino-chapter-18-rgb-led-matrix/
- http://blog.spitzenpfeil.org/wordpress/projects/8x8-rgb-led-matrix/
- http://blog.spitzenpfeil.org/wordpress/2008/09/08/8x8-rgb-matrix-2nd-arduino-project/
- http://neuroelec.com/2011/03/fft-library-for-arduino/

- http://code.google.com/p/neuroelec/downloads/detail?name=ffft_library_for_Arduino_1.0.zip&can=2&q=