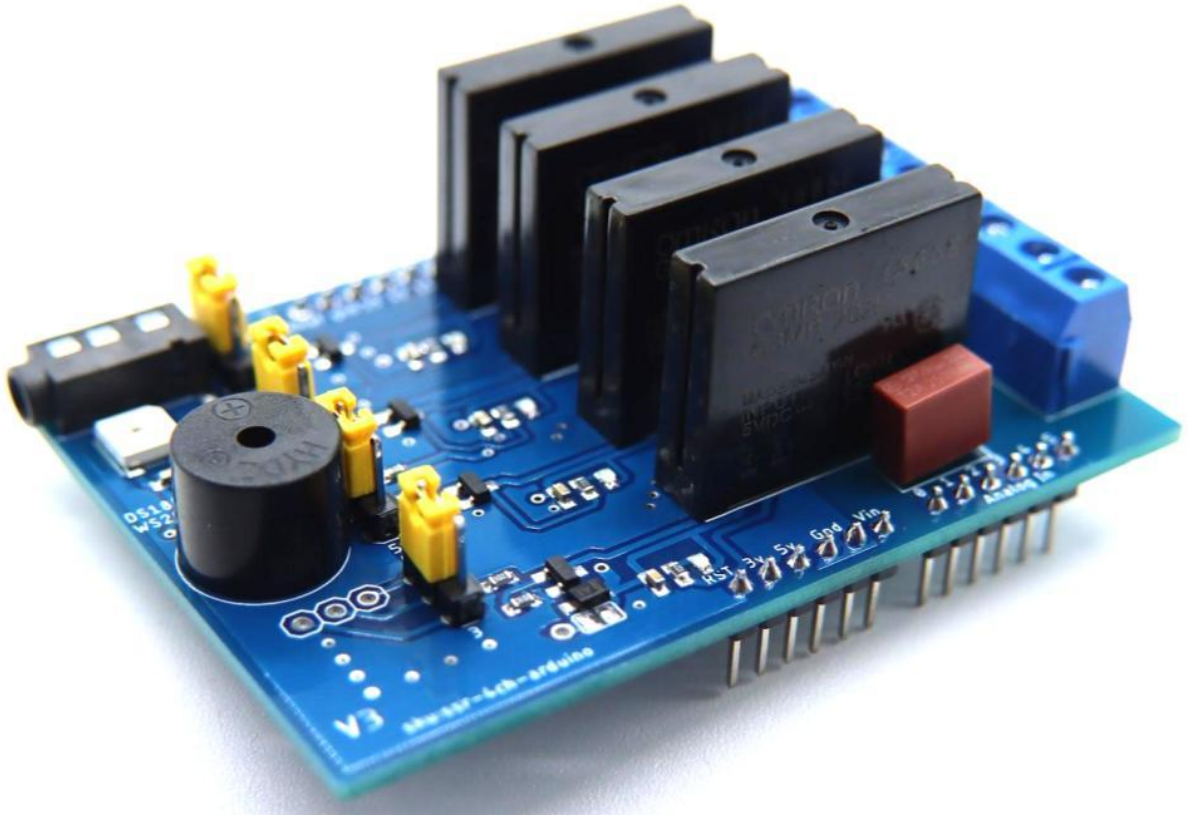

El SSR-4CH-Arduino es una tarjeta montable para Arduino-UNO-R3 la cual cuenta con 4 relevadores de estado sólido modelo G3MB-202P. Este tipo de relevador puede controlar cargas de hasta 240 VAC a 2A. El módulo cuenta con fusibles de 2A como protección contra cortocircuito. El control se hace a través de los pines digitales de Arduino:

- Canal 1: 10 o 9.
- Canal 2: 8 o 7.
- Canal 3: 6 o 5.
- Canal 4. 4 o 3.

El shield de relevadores SSR cuenta adicionalmente con un conector tipo 3.5mm de audio el cual puede ser usado para conectar un sensor de temperatura digital como el DS18B20 (Se vende por separado), o sensores del mismo tipo. Este conector se encuentra conectado al pin 11. También cuenta con un buzzer activo en el pin 13 y un led WS2812B al pin 12.

Especificaciones

- Consumo de corriente: 150mA.
- Voltaje de operación: 5VDC.
- Dimensiones PCB: 73.5mm x 53.5mm x 1.6mm.

Código de Arduino

https://gist.github.com/esmarr58/ab73030bc1c699bc3f63497cce5466ee

```
#include <Adafruit_NeoPixel.h> //Adafruit NeoPiexel version 1.10.4
#include <OneWire.h>  //OneWire by Paul Stoffregen version 2.3.6

#ifdef __AVR__
 #include <avr/power.h> // Required for 16 MHz Adafruit Trinket
#endif

#define PIN        12 // On Trinket or Gemma, suggest changing this to 1
#define relay1 10
#define relay2 8
#define relay3 6
#define relay4 4
#define NUMPIXELS 1 // Popular NeoPixel ring size
#define buzzer 13


Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);

#define DELAYVAL 500 // Time (in milliseconds) to pause between pixels
OneWire  ds(11);  // on pin 10 (a 4.7K resistor is necessary)


void setup() {
  // These lines are specifically to support the Adafruit Trinket 5V 16 MHz.
  // Any other board, you can remove this part (but no harm leaving it):
#if defined(__AVR_ATtiny85__) && (F_CPU == 16000000)
  clock_prescale_set(clock_div_1);
#endif
  // END of Trinket-specific code.

  pixels.begin(); // INITIALIZE NeoPixel strip object (REQUIRED)
  pinMode(3, OUTPUT);
  pinMode(relay1, OUTPUT);
  pinMode(relay2, OUTPUT);
  pinMode(relay3, OUTPUT);
  pinMode(relay4, OUTPUT);
  pinMode(buzzer, OUTPUT);

  Serial.begin(115200);

}

void loop() {
  Serial.println("Hola mundo");
    pixels.clear(); // Set all pixel colors to 'off'
```

```
  pixels.setPixelColor(0, pixels.Color(0, 0, 255));

  pixels.show();   // Send the updated pixel colors to the hardware.

  delay(200); // Pause before next pass through loop

   pixels.clear(); // Set all pixel colors to 'off'

  pixels.setPixelColor(0, pixels.Color(00, 255, 0));

  pixels.show();   // Send the updated pixel colors to the hardware.

  delay(200); // Pause before next pass through loop

   pixels.clear(); // Set all pixel colors to 'off'

  pixels.setPixelColor(0, pixels.Color(255, 0, 0));

  pixels.show();   // Send the updated pixel colors to the hardware.

  delay(200); // Pause before next pass through loop


 digitalWrite(relay1, HIGH);
 digitalWrite(relay2, HIGH);
 digitalWrite(relay3, HIGH);
 digitalWrite(relay4, HIGH);
 digitalWrite(buzzer, HIGH);

 delay(500);
    digitalWrite(buzzer, LOW);

    byte i;
byte present = 0;
byte type_s;
byte data[9];
byte addr[8];
float celsius, fahrenheit;

if ( !ds.search(addr)) {
  Serial.println("No more addresses.");
  Serial.println();
  ds.reset_search();
  delay(250);
  return;
}

Serial.print("ROM =");
for( i = 0; i < 8; i++) {
  Serial.write(' ');
  Serial.print(addr[i], HEX);
}

if (OneWire::crc8(addr, 7) != addr[7]) {
```

```
    Serial.println("CRC is not valid!");
    return;
  }
  Serial.println();

  // the first ROM byte indicates which chip
  switch (addr[0]) {
    case 0x10:
      Serial.println("  Chip = DS18S20");  // or old DS1820
      type_s = 1;
      break;
    case 0x28:
      Serial.println("  Chip = DS18B20");
      type_s = 0;
      break;
    case 0x22:
      Serial.println("  Chip = DS1822");
      type_s = 0;
      break;
    default:
      Serial.println("Device is not a DS18x20 family device.");
      return;
  }

  ds.reset();
  ds.select(addr);
  ds.write(0x44, 1);        // start conversion, with parasite power on at the end

  delay(1000);     // maybe 750ms is enough, maybe not
  // we might do a ds.depower() here, but the reset will take care of it.

  present = ds.reset();
  ds.select(addr);
  ds.write(0xBE);         // Read Scratchpad

  Serial.print("  Data = ");
  Serial.print(present, HEX);
  Serial.print(" ");
  for ( i = 0; i < 9; i++) {           // we need 9 bytes
    data[i] = ds.read();
    Serial.print(data[i], HEX);
    Serial.print(" ");
  }
  Serial.print(" CRC=");
  Serial.print(OneWire::crc8(data, 8), HEX);
  Serial.println();

  // Convert the data to actual temperature
  // because the result is a 16 bit signed integer, it should
  // be stored to an "int16_t" type, which is always 16 bits
  // even when compiled on a 32 bit processor.
  int16_t raw = (data[1] << 8) | data[0];
  if (type_s) {
    raw = raw << 3; // 9 bit resolution default
```

```
    if (data[7] == 0x10) {
      // "count remain" gives full 12 bit resolution
      raw = (raw & 0xFFF0) + 12 - data[6];
    }
  } else {
    byte cfg = (data[4] & 0x60);
    // at lower res, the low bits are undefined, so let's zero them
    if (cfg == 0x00) raw = raw & ~7;  // 9 bit resolution, 93.75 ms
    else if (cfg == 0x20) raw = raw & ~3; // 10 bit res, 187.5 ms
    else if (cfg == 0x40) raw = raw & ~1; // 11 bit res, 375 ms
    //// default is 12 bit resolution, 750 ms conversion time
  }
  celsius = (float)raw / 16.0;
  fahrenheit = celsius * 1.8 + 32.0;
  Serial.print("  Temperature = ");
  Serial.print(celsius);
  Serial.print(" Celsius, ");
  Serial.print(fahrenheit);
  Serial.println(" Fahrenheit");


   digitalWrite(relay1, LOW);
   digitalWrite(relay2, LOW);
   digitalWrite(relay3, LOW);
   digitalWrite(relay4, LOW);


}
```

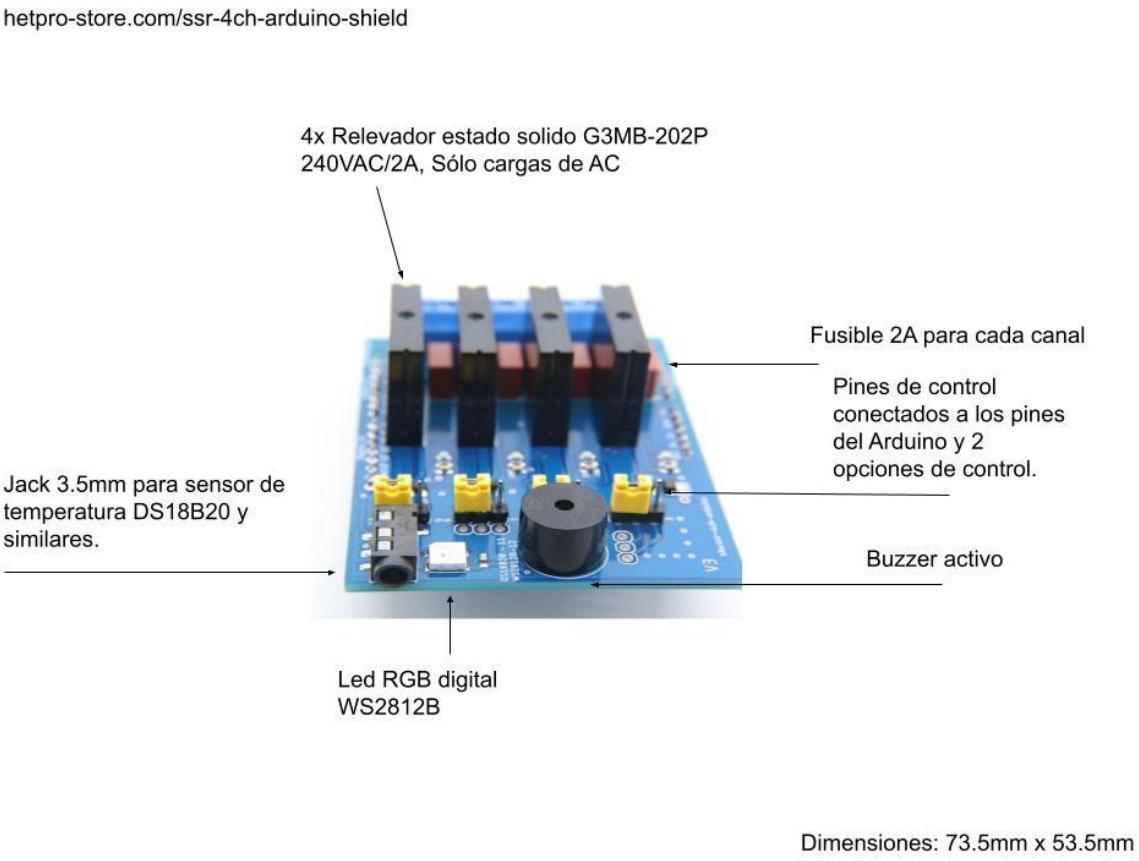

Figura 2. Especificaciones del Shield para Arduino con SSR G3MB-202P de 4 canales.

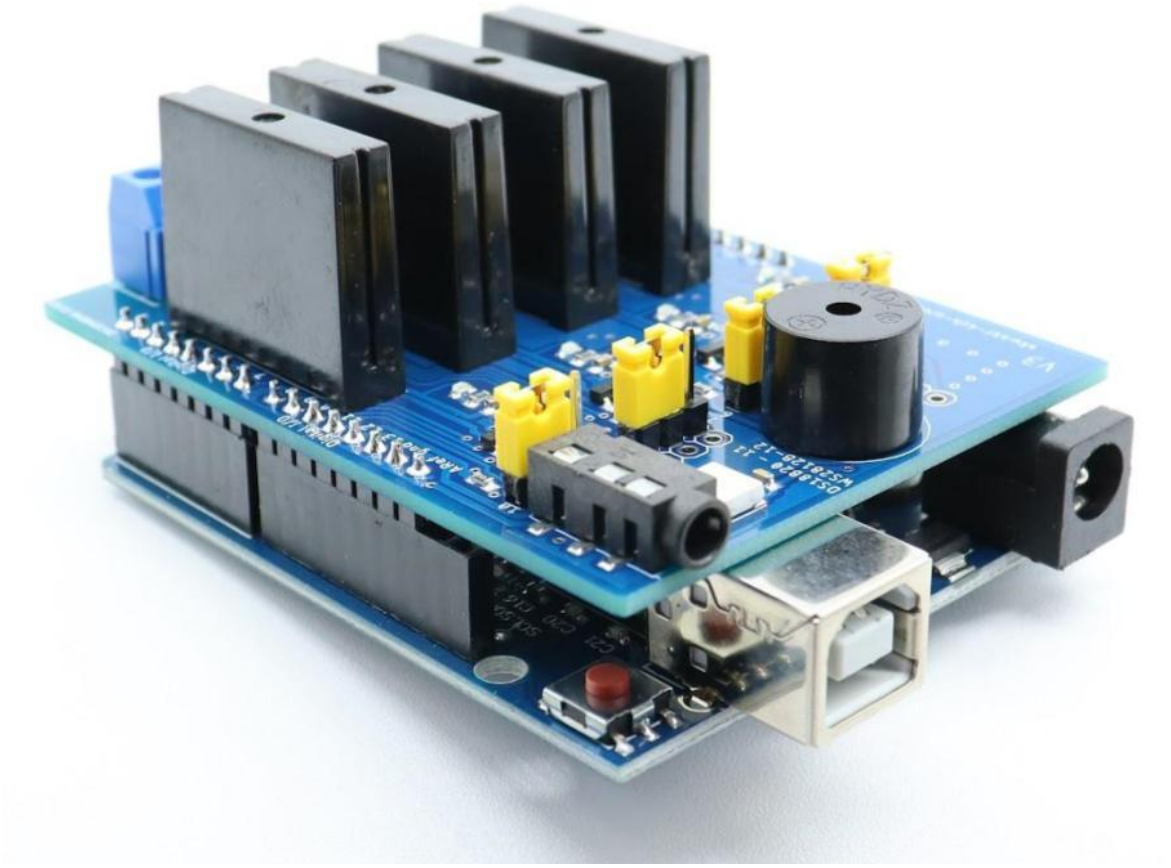

Figura 3. Imagen demostrativa de uso, el producto no incluye el Arduino UNO.