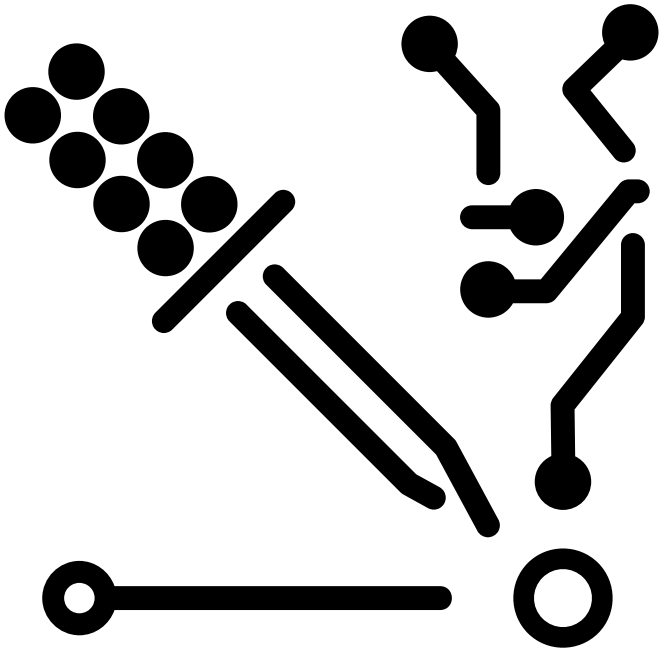


# TALLER DE ROBÓTICA CON ARDUINO

# SOFTWARE SERIAL

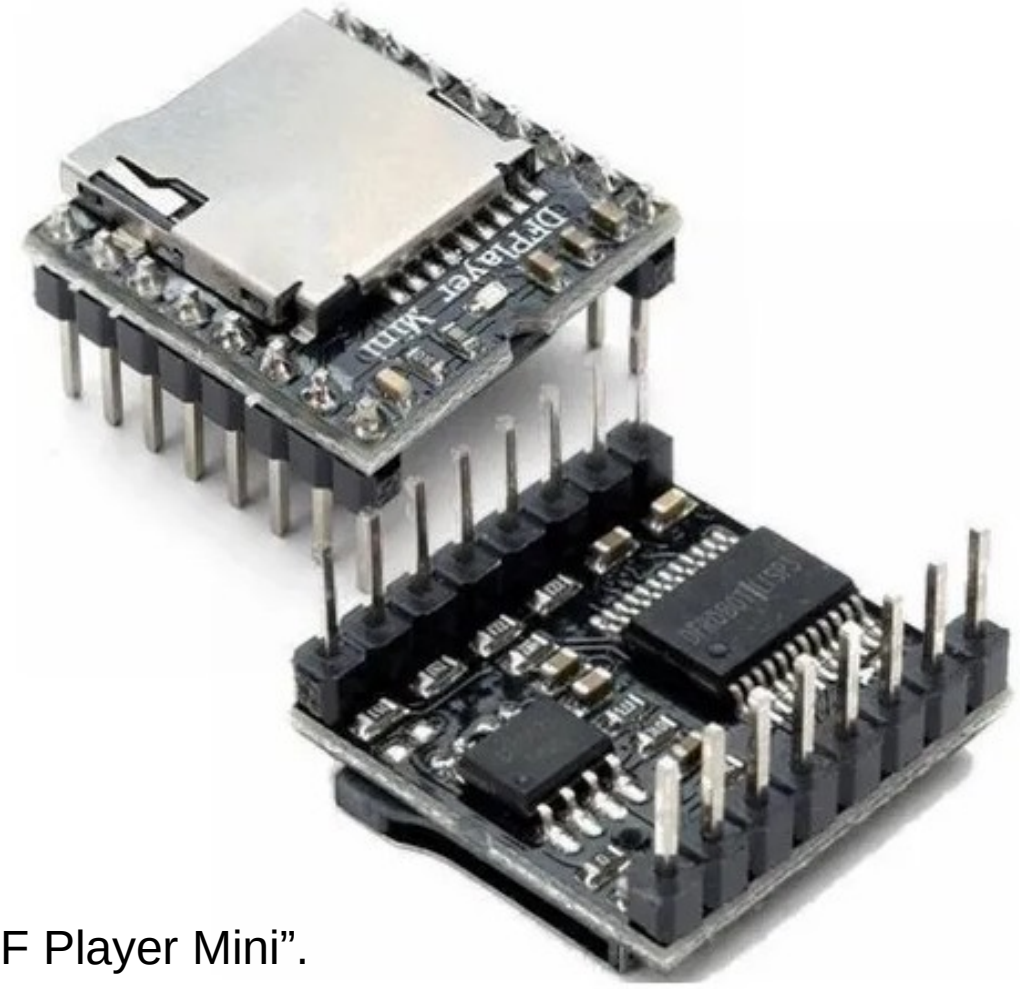
Arduino tiene soporte nativo para la **comunicación en serie**, en los pines 0 y 1, a través de un periférico integrado en el microcontrolador llamado **UART (Universal Asynchronous Receiver-Transmitter)**. Este hardware permite que el chip ATmega reciba comunicación en serie incluso mientras trabaja en otras tareas, siempre que haya espacio en el búfer serie (64 bytes).
La librería **SoftwareSerial** se ha desarrollado para permitir la comunicación en serie en otros pines digitales del Arduino, utilizando software para replicar la funcionalidad. Es posible tener varios puertos serie por software con velocidades de hasta 115200bps.

»»» SoftwareSerial Library

# SOFTWARE SERIAL

```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(2, 3); // RX, TX

void setup(){
    mySerial.begin(38400);
    mySerial.println("Hola Mundo!");
}

void loop(){

}
```

# SOFTWARE SERIAL




Reproductor MP3 “DF Player Mini”.

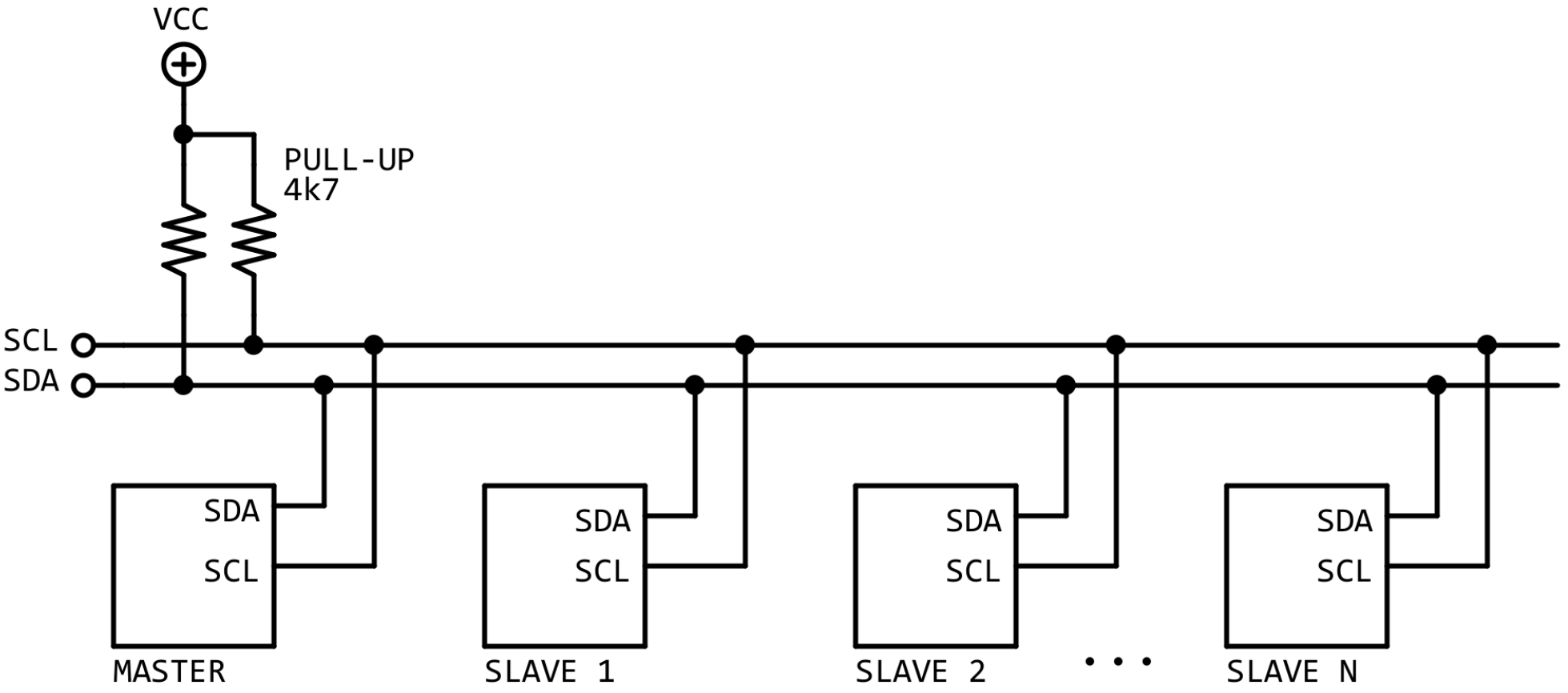
# PROTOCOLO I2C

El estándar **I2C (Inter-Integrated Circuit)** fue desarrollado por Philips en 1982 para la comunicación interna de los circuitos integrados que utilizaba en sus productos. Posteriormente fue adoptado progresivamente por otros fabricantes hasta convertirse en un estándar del mercado.

El bus I2C requiere únicamente dos cables para su funcionamiento, uno para la señal de reloj **(CLK)** y otro para el envío de datos **(SDA)**.

Para usar el bus I2C en Arduino, el IDE proporciona la librería **"Wire.h"**, que contiene las funciones necesarias para controlar el hardware integrado.

# PROTOCOLO I2C



# PROTOCOLO I2C

Las principales características del protocolo son:

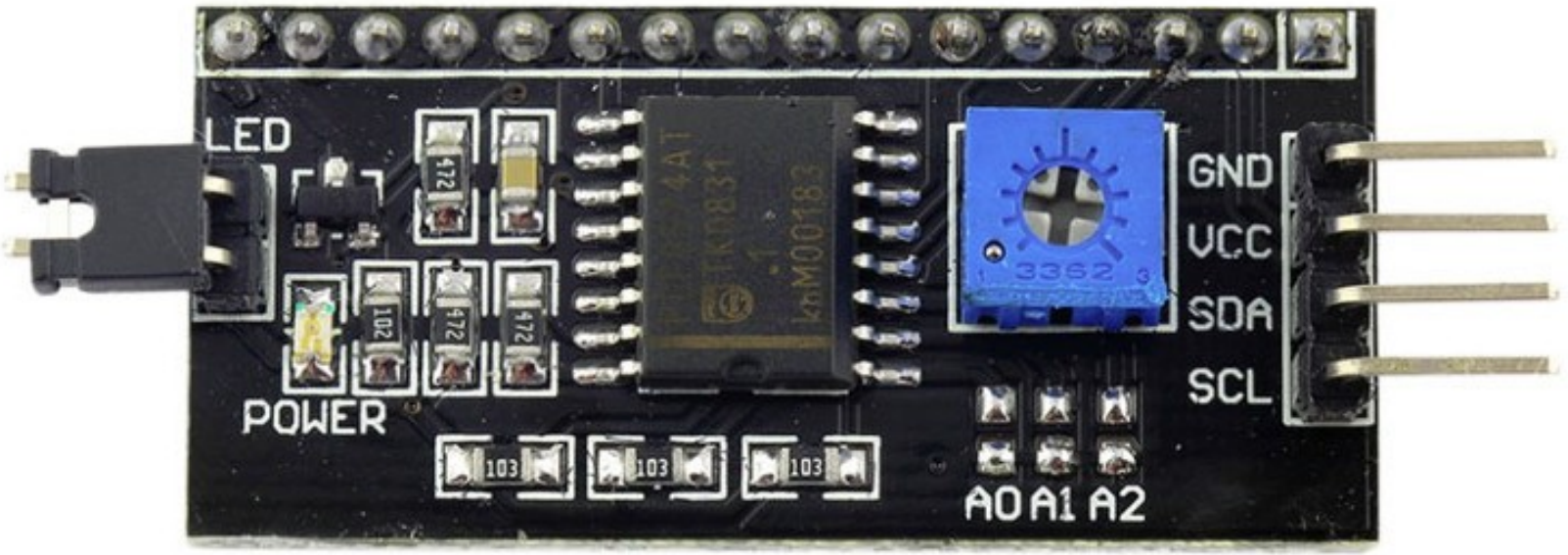
- Posee una **arquitectura de tipo maestro-esclavo**. El dispositivo maestro es el que inicia la comunicación con los esclavos.
- Cada dispositivo dispone de una **dirección única**, que puede ser fijada por hardware o por software.
- Es un bus **síncrono**. El maestro proporciona una señal de reloj, que mantiene sincronizados a todos los dispositivos conectados al bus.
- El protocolo necesita de **resistencias de pull-up** (entre 1k a 4k7).
- Es posible acceder a **112 dispositivos** en un mismo bus (16 direcciones de las 128 direcciones posibles se reservan para usos especiales).
- En general, la **velocidad del bus I2C es reducida**. La velocidad estándar de transmisión es de 100kHz, con un modo de alta velocidad de 400kHz.

# PROTOCOLO I2C

Arduino dispone de **soporte I2C por hardware** vinculado físicamente a ciertos pines del microcontrolador. También es posible emplear cualquier otro grupo de pines como bus I2C a través de sofware, pero en ese caso la velocidad será mucho menor.

| MODELO | SDA | SCK |
|---|---|---|
| Uno | A4 | A5 |
| Nano | A4 | A5 |
| Mini Pro | A4 | A5 |
| Mega | 20 | 21 |

# PROTOCOLO I2C




Módulo LCD PCF8574

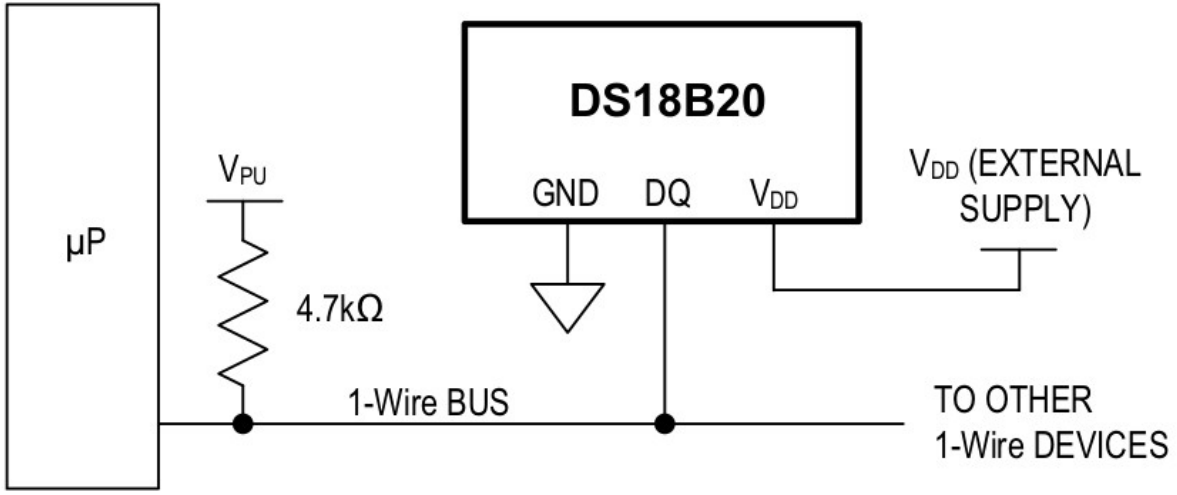
# PROTOCOLO I2C

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27,16,2);

void setup(){
    lcd.init();
    lcd.init();
    lcd.backlight();
    lcd.setCursor(3,0);
    lcd.print("Hola Mundo!");
}

void loop(){

}
```

»»»» LiquidCrystal_I2C

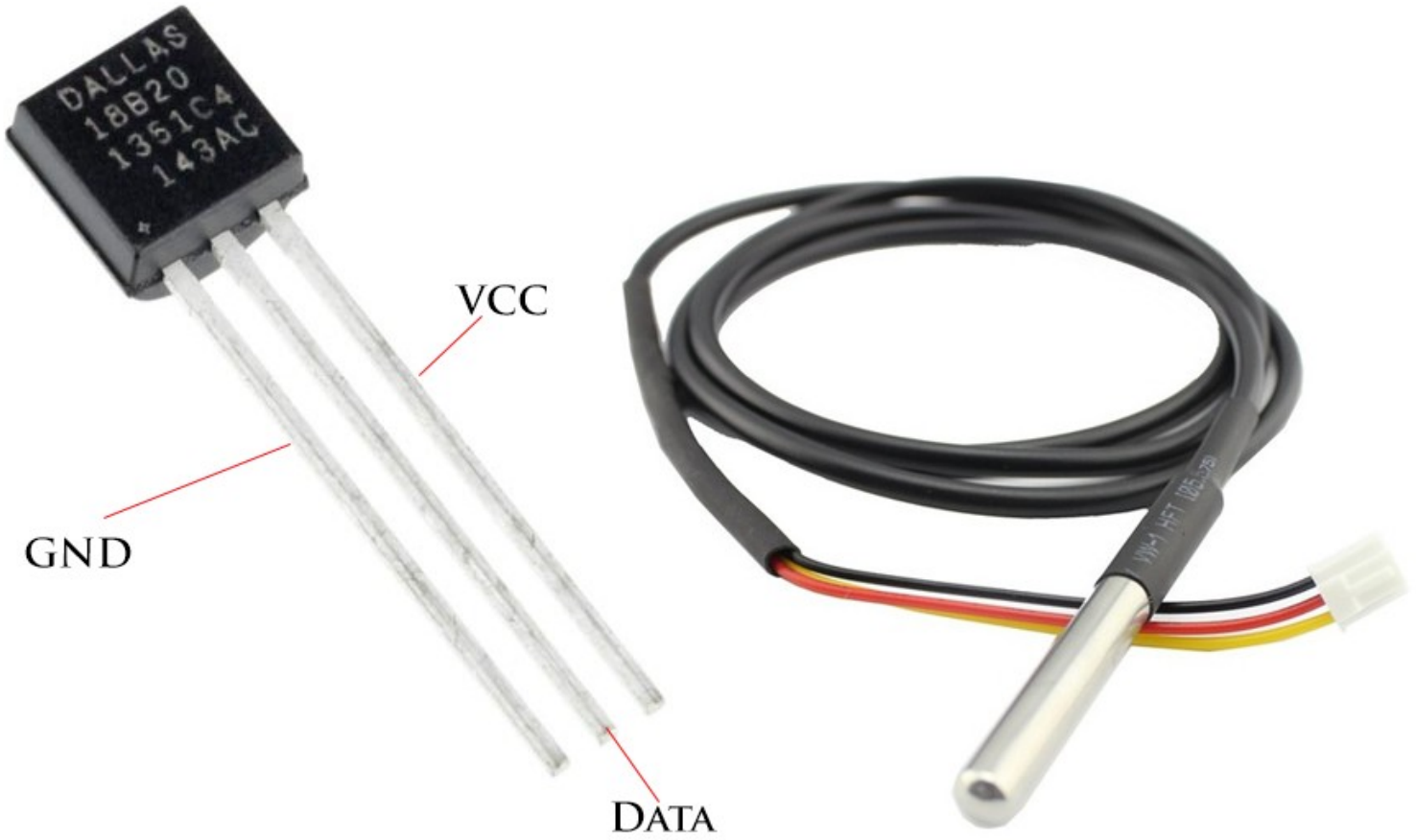
# ONE-WIRE

**1-Wire** es un protocolo de comunicaciones en serie diseñado por Dallas Semiconductor. Está basado en un bus de una sola línea de datos, un maestro y varios esclavos. Por supuesto, necesita una referencia a tierra común a todos los dispositivos.
En el repositorio oficial de Arduino se puede descargar una librería que permite comunicarse con los dispositivos que hagan uso del bus.




»»»» OneWire Library

# ONE-WIRE




Sensor de temperatura “DS18B20”

# ONE-WIRE

```
#include <OneWire.h>
#include <DallasTemperature.h>

OneWire oneWire(2); // Pin del bus
DallasTemperature sensor(&oneWire);

void setup(){
    Serial.begin(9600);
    sensor.begin();
}

void loop(){
    sensor.requestTemperatures();
    float temp = sensor.getTempCByIndex(0);
    Serial.println(temp);
}
```

»»»» Arduino Library for Maxim Temperature Integrated Circuits

# SPI

El bus **SPI (Serial Peripheral Interface)** fue desarrollado por Motorola en 1980, tiene una arquitectura de tipo **maestro-esclavo**. El dispositivo maestro puede iniciar la comunicación con uno o varios dispositivos esclavos, y enviar o recibir datos de ellos. Los dispositivos esclavos no pueden iniciar la comunicación, ni intercambiar datos entre ellos directamente.

En el bus SPI la comunicación de datos entre maestros y esclavo se realiza en dos líneas independientes, una del maestro a los esclavos, y otra de los esclavos al maestro. Por tanto la comunicación es **Full Duplex**, es decir, el maestro puede enviar y recibir datos simultáneamente.

Otra característica de SPI es que es bus **síncrono**. El dispositivo maestro proporciona una señal de reloj, que mantiene a todos los dispositivos sincronizados.

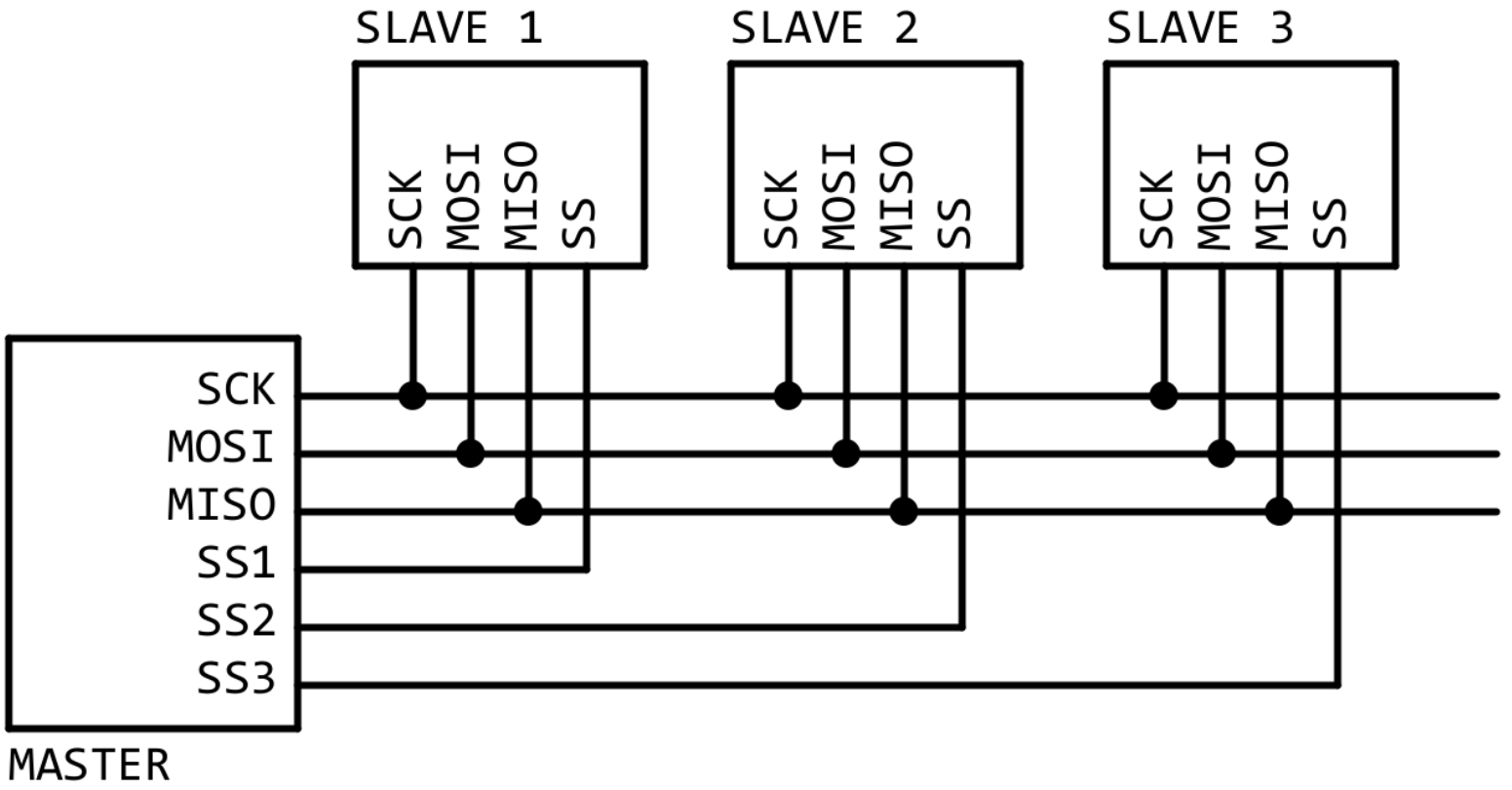
Para usar el puerto SPI en Arduino, el IDE proporciona la librería **"SPI.h"** que contiene las funciones necesarias para controlar el hardware integrado de SPI.

# SPI

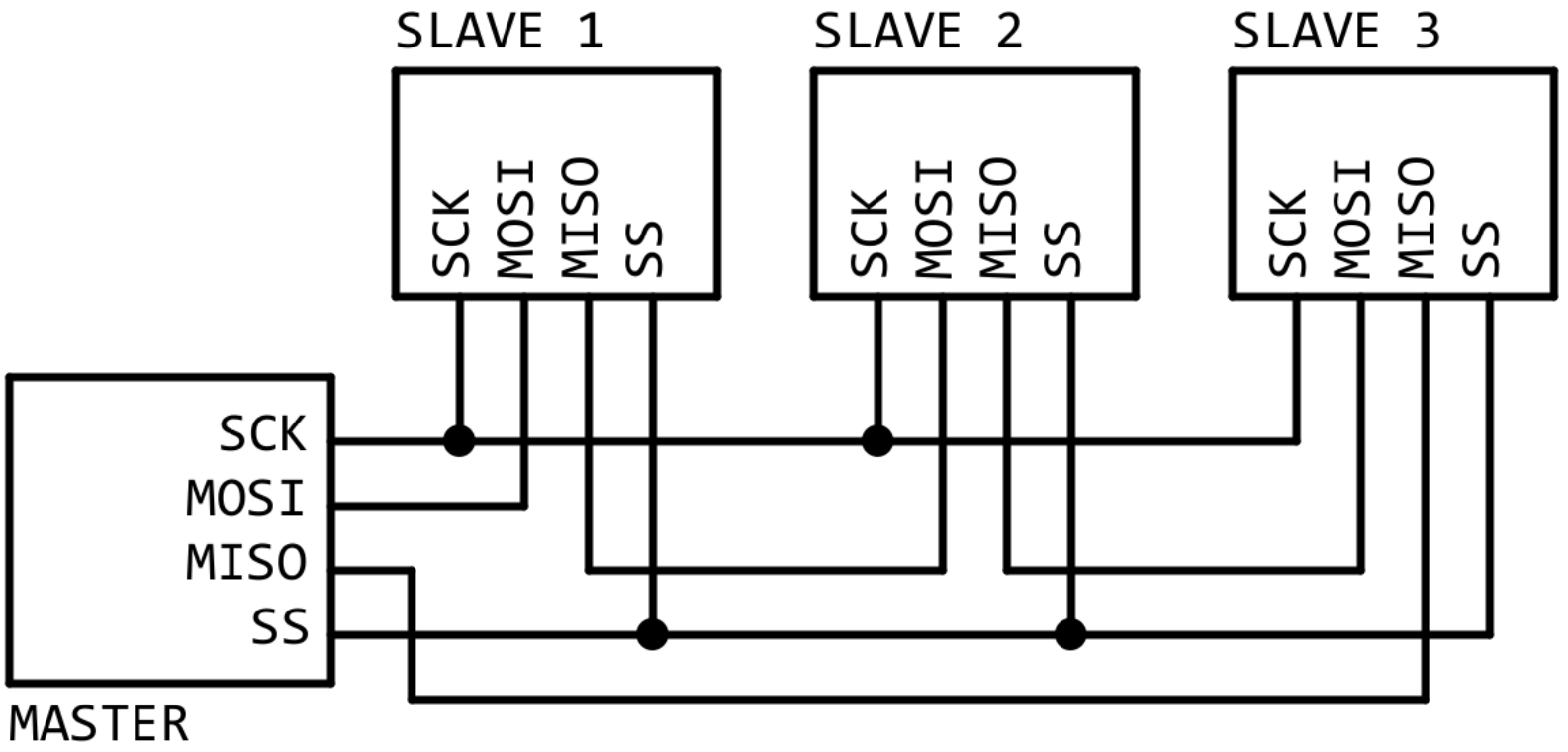
Necesita 3 líneas + 1 línea adicional por cada esclavo:

- **MOSI (Master-out, slave-in)** para la comunicación del maestro al esclavo.
- **MISO (Master-in, slave-out)** para comunicación del esclavo al maestro.
- **SCK (Clock)** señal de reloj enviada por el maestro.
- **SS (Slave Select)** para cada dispositivo esclavo conectado.

# SPI


SLAVE 1
SLAVE 2
SLAVE 3
SCK
MOSI
MISO
SS
SCK
MOSI
MISO
SS
SCK
MOSI
MISO
SS
SCK
MOSI
MISO
SS1
SS2
SS3
MASTER

# SPI




Conexión en cascada.

# SPI

Arduino dispone de **soporte SPI por hardware** vinculado físicamente a ciertos pines. También es posible emplear cualquier otro grupo de pines como bus SPI a través de sofware, pero en ese caso la velocidad será mucho menor.
El pin SS por hardware se emplea al usar Arduino como esclavo. En caso de usar Arduino como maestro, podemos usar cualquier pin como SS, o varios en caso de disponer de varios esclavos.

| MODELO | SS | MOSI | MISO | SCK |
|---|---|---|---|---|
| Uno | 10 | 11 | 12 | 13 |
| Nano | 10 | 11 | 12 | 13 |
| Mini Pro | 10 | 11 | 12 | 13 |
| Mega | 53 | 51 | 50 | 52 |

# SPI

No todos los fabricantes emplean la misma designación para los pines que participan en el bus SPI. Por ejemplo:

| Nombre | Alias | Pin (en Arduino Uno o Nano) | Descripcion |
|---|---|---|---|
| VCC | | | +3.3 ... 5 Volt |
| GND | | Ground | Ground |
| SCLK | CLK/SCK/SCLK | D13 (hardware) | Clock (SPI) |
| MISO | MISO/SDO/DOUT | D12 (hardware) | Master In Slave Out (SPI) |
| MOSI | MOSI/SDI/DIN | D11 (hardware) | Master Out Slave In (SPI) |
| SS | SS/CS/SDA/ | D10 (hardware, solo en esclavo) | Slave/Chip Select (SPI) |
| RES | RST/RES/REST | D9 (variable, se fija por software) | Controller Reset |
| RS | RS/DC | D8 (variable, se fija por software) | Mode: Command/Data |

# CRÉDITOS

**Lucas Martín Treser**
lmtreser@gmail.com – www.automatismos-mdq.com.ar

Atribución-NoComercial 4.0
Internacional (CC BY-NC 4.0)