Electrónica Microcontrolada

Docentes: Ing. Jorge E. Morales Téc. Sup. Mecatrónica Gonzalo Vera.

GRUPO NRO 8:

- Schafrik Maria Victoria
- Vera Emilio Andres
- Rojas Jorge Daniel
- Rojo Pedro Omar
- Narvaez Juan Carlos

PraCTICO 4

SHIELDs V1.0

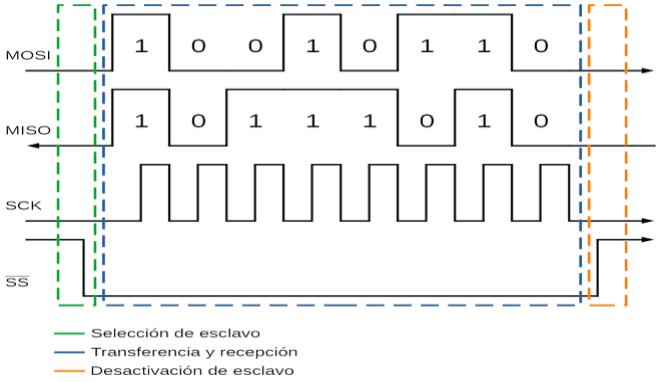
Como probaría si una comunicación SPI funciona correctamente en su laboratorio.

Comunicación SPI con Arduino, etapas de la comunicación SPI Arduino

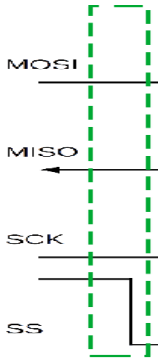
En **SPI** es posible resumir el proceso de comunicación entre el maestro y un esclavo en tres etapas:

- **Etapa 1: activación de esclavo.**
- **Etapa 2: transferencia y/o recepción de información.**
- **Etapa 3: desactivación de esclavo.**

En la siguiente imagen se muestra un ejemplo de comunicación **SPI** donde se han señalado las tres etapas.



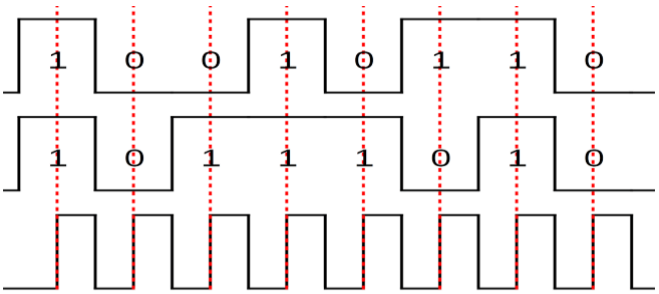
Etapa 1: Activación de esclavo



El primer paso para comunicarse con un esclavo es poner el **pin SS** correspondiente en estado bajo. Una vez realizado esto el esclavo puede leer los datos enviados por el **pin MOSI** y escribir datos en el **pin MISO**.

Solo un esclavo puede tener su **pin SS** en estado bajo al mismo tiempo, de lo contrario es posible que existan errores de comunicación o incluso que alguno de los esclavos resulte dañado. De esta gestión se encarga el maestro.

Etapa 2: Transferencia y/o recepción de información



Esta etapa es donde se intercambia información entre los dispositivos. **Las transferencias se realizan mediante los pines MOSI y MISO en unidades de 8 bits**, es decir, 1 byte. Ojo, que **se pueden transmitir varios bytes, pero siempre en unidades completas**. En otras palabras, no se puede enviar 1 byte y medio, por ejemplo.

Si necesitas enviar 1 byte y medio tendrás que hacerlo en 2 bytes.

Cabe destacar que, tal y como se puede ver en la imagen, un estado alto (HIGH) en los pines de datos representan un 1 lógico; mientras un estado bajo (LOW) representa un 0 lógico.

La señal de reloj juega un papel fundamental en esta etapa. En cada pulso de reloj el dispositivo maestro lee el estado del **pin MISO**; y además, pone la **línea MOSI** en estado alto (HIGH) o bajo (LOW), en dependencia del bit a transmitir. De igual forma el esclavo obtiene el estado del **pin MOSI** (lee el bit enviado por el maestro) y modifica el estado del **pin MISO** (envía información al maestro).

Como puedes ver ambos dispositivos pueden transmitir información al mismo tiempo sin ningún inconveniente. Es por esto que **SPI** también **es una comunicación full-duplex**.

Durante toda esta etapa el **pin SS** debe permanecer en estado bajo (LOW).

Etapa 3: Desactivación de esclavo

Una vez se ha intercambiado toda la información necesaria es preciso indicarle al esclavo que ya no se va a continuar interactuando con él. Para esto es necesario poner el **pin SS** nuevamente en estado alto (HIGH).

Parámetros de SPI Arduino

Como has podido apreciar la comunicación es muy sencilla, sin tramas complejas ni otros mecanismos. Sin embargo, al ser **SPI** un estándar libre, cada dispositivo lo implementa de manera un poco diferente. Esto significa que se debe prestar especial atención a la hoja de características técnicas del dispositivo con el que se desea establecer la comunicación.

A la hora de establecer una comunicación utilizando **SPI** es necesario tener en cuenta tres parámetros:

- La frecuencia de la señal de reloj (pin SCK)
- El orden de transmisión de los bits
- El modo de operación.

Frecuencia de reloj

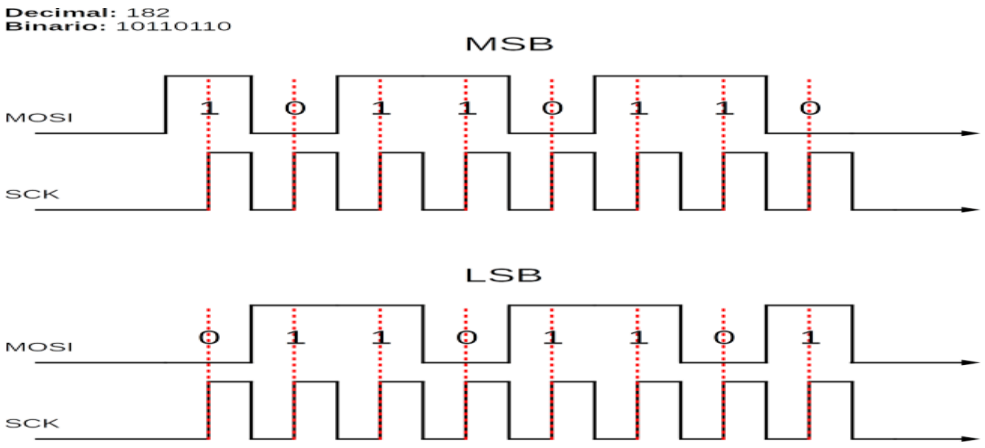
Lo primero que hay que revisar a la hora de utilizar un dispositivo que emplea SPI es su máxima frecuencia de comunicación, es decir, cuál es la máxima frecuencia a la que el maestro puede manejar el pin SCK para enviar o recibir información desde el esclavo. Este parámetro usualmente aparece en las hojas de datos de los componentes o en los manuales de usuario.

En caso de que se emplee una frecuencia de operación mayor a la máxima soportada por el dispositivo no será posible establecer una comunicación con el esclavo.

Orden de transferencia de bits

Otro parámetro a tener en cuenta es **el orden en que se deben enviar y/o leer los bits del esclavo**.

Esto se debe a que algunos dispositivos comienzan transmitiendo los bytes desde el **bit más significativo** (**MSB**), mientras que otros lo realizan desde el **bit menos significativo** (**LSB**). A continuación, puedes ver un ejemplo de cómo varía la etapa de transferencia.



En la figura se muestra como quedaría la transmisión de un byte con valor 182 utilizando ambos métodos. Es importante destacar que si un dispositivo no recibe los bits en el orden correcto puede interpretar de forma errónea dicha información.

Modos de transmisión

De modo general **se definen 4 modos** de transmisión, **que dependen de la polaridad y la fase utilizada para la señal de reloj**.

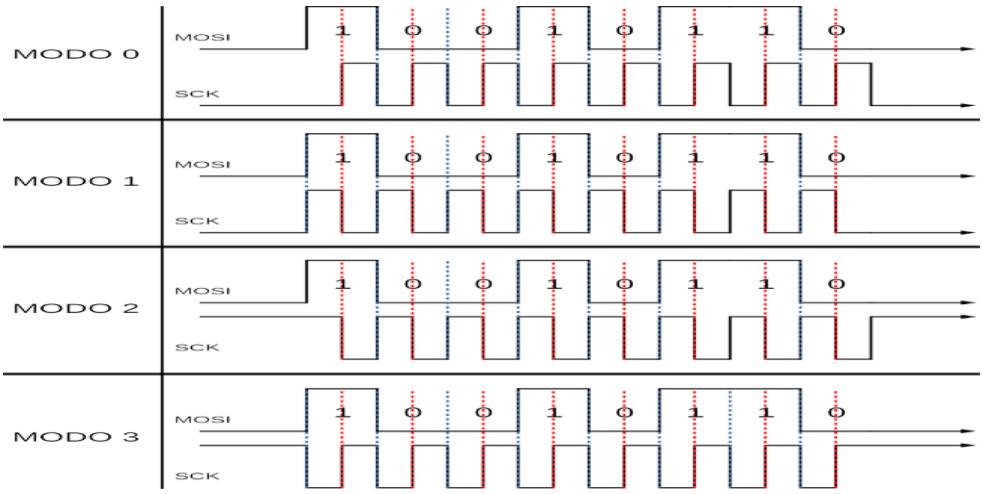
La **polaridad** se refiere al estado en que se debe mantener la señal reloj cuando está inactiva, esta puede ser alta o baja.

La **fase**, por otra parte, define el **momento en que la información es escrita en el pin de salida**. Esto puede ser en el flanco de bajada (cuando va de alto a bajo) o en el flanco de subida (cuando va de bajo a alto) de la señal de reloj.

En la siguiente tabla puedes ver un resumen de los cuatro modos.

MODO	POLARIDAD	FASE
Modo 0	Estado bajo	Flanco de bajada
Modo 1	Estado bajo	Flanco de subida
Modo 2	Estado alto	Flanco de subida
Modo 3	Estado alto	Flanco de bajada

En la siguiente figura puedes ver las señales de reloj y datos para cada uno de estos modos.



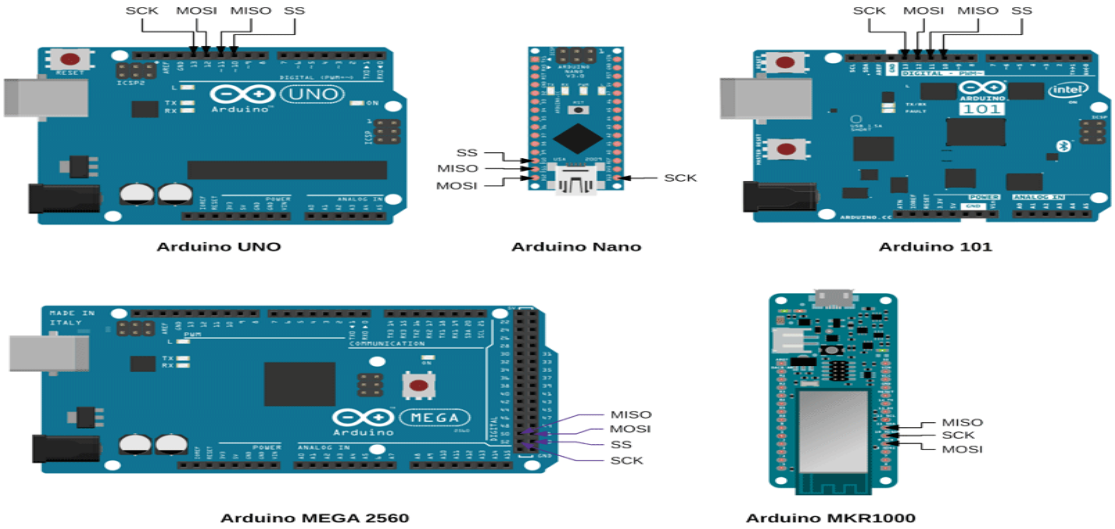
Los flancos en que se actualizan los pines han sido marcados en azul y en rojo se han marcado los flancos en los que el esclavo lee el valor del pin.

Identificar pines SPI con Arduino

Debido a la gran variedad de **placas de Arduino**, la posición de los pines del **SPI** puede variar bastante de un modelo a otro.

En la siguiente tabla puedes encontrar un resumen donde se muestra los pines digitales correspondientes a los **pines SPI** en la mayoría de las placas Arduino.

PLACA ARDUINO	MOSI	MISO	SCK	SS	NIVEL DE VOLTAJE
Arduino UNO (o compatible)	11	12	13	10	5V
Arduino Nano	11	12	13	10	5V
Arduino Mega 2560	51	50	52	53	5V
Arduino 101	11	12	13	10	3.3V
Arduino MKR1000	8	10	9	–	3.3V



Arduino UNO — Arduino Nano — Arduino 101

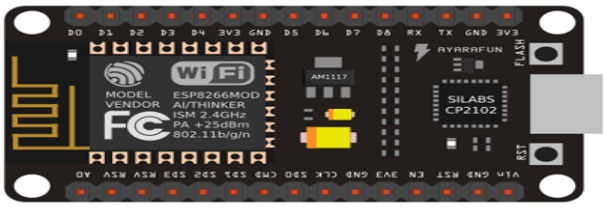
Arduino MEGA 2560 — Arduino MKR1000

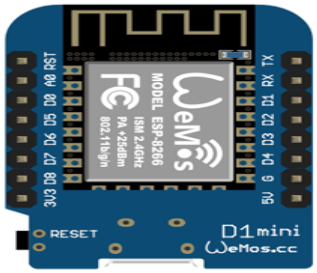
Adicionalmente, los **pines SPI** se encuentran en el **cabezal ICSP** (presente en la mayoría de las placas Arduino), esto ha permitido el diseño de *Shields* que utilicen este protocolo y sean compatibles con los distintos modelos de placas Arduino. Uno de los ejemplos más representativos es el ***Ethernet Shield***.

En la siguiente imagen, obtenida del esquema eléctrico de la **placa Arduino UNO** se puede observar la correspondencia entre las **señales SPI** y los **pines de la cabecera ICSP**.

La conexión de los **pines SPI** en el **cabezal ICSP** es igual para todas las **placas Arduino**.

En caso de que planees utilizar una **placa basada en** ESP8266 es necesario ser cuidadosos al determinar los pines SPI, ya que la mayoría de estas no poseen **conector ICSP**.





En estas placas los pines de SPI tienen la siguiente asignación:

- SCK: D5 (GPIO14)
- MISO: D6 (GPIO12)
- MOSI D7 (GPIO13)
- SS: D8 (GPIO15)

Cabe la posibilidad de utilizar SPI por software utilizando otros pines. Ten en cuenta que al hacerlo así estás perdiendo velocidad en la conexión con el BUS SPI con Arduino.

Librería SPI con Arduino

Para controlar el **bus SPI con Arduino** existe la **librería SPI**, que permite enviar y recibir datos utilizando dicho bus. Siempre y cuando **Arduino** se comporte como maestro. En otras palabras, la librería solo permite programar la placa para que funcione como maestro, y no como esclavo.

Esta librería cuenta con dos clases:

- **SPISettings**: permite establecer los parámetros de las transferencias SPI a realizar.
- **SPI**: permite controlar el bus SPI para enviar y recibir información.

Inicio analizando la clase **SPISettings,** ya que es la más simple.

Clase SPISettings

Un objeto de esta clase puede ser utilizado para configurar el **bus SPI**. Esto implica especificar:

- Velocidad de transferencia máxima.
- Orden de transmisión de bits
- Modo de transmisión

La sintaxis para crear un objeto de este tipo es la siguiente:

```
1 SPISettings mySettting(speedMaximum, dataOrder, dataMode);
```

Donde:

- ***mySettting*:** es el objeto a crear.
- ***speedMaximum***: es la máxima frecuencia a utilizar especificada en herz.
- ***dataOrder***: es el orden en que se transfieren los bits. Su valor puede ser:
- **MSBFIRST**: para comenzar por el bit más significativo.
- **LSBFIRST**: para comenzar por el bit menos significativo.
  dataMode: especifica el modo de transmisión. Puede tomar uno de los siguientes valores: SPI_MODE0, SPI_MODE1, SPI_MODE2 o SPI_MODE3.

Funciones de la Clase SPI

La clase SPI es estática, lo que quiere decir que **no hace falta crear una instancia u objeto de la clase**. Es como las librerías *Serial* o *Wire*.

Directamente escribes la clase (**SPI**) y luego llamas a la función correspondiente. Aquí alguna de estas funciones.

SPI.begin()

Esta función inicializa el **bus SPI** y configura los **pines SCK, MOSI y SS** como salida.

Su sintaxis sería:

```
1 SPI.begin();
```

Después de ejecutar esta función no se deben utilizar las funciones ***digitalWrite()*** o ***analogWrite()*** en los **pines de SPI**.

SPI.end()

Esta función realiza lo contrario a ***SPI.begin()***, es decir, que des-inicializa el **bus SPI**. De esta forma es posible utilizar las funciones ***digitalWrite()*** o ***analogWrite()*** para controlar los **pines de SPI** nuevamente.

Su sintaxis es la siguiente:

```
1 SPI.end();
```

SPI.beginTransaction()

Esta función es utilizada para preparar el **bus SPI** al comienzo de una transferencia y tiene dos propósitos:

1. Establecer los parámetros necesarios para la transferencia a realizar.
2. Indicar que el **bus SPI** está siendo utilizado. De lo contrario otra librería que emplee el **bus SPI** podría intentar utilizarlo, generando errores en la comunicación.

Su sintaxis es:

```
1 SPI.beginTransaction(mySettings);
```

Donde:

- ***mySettings*****:** es un objeto de tipo ***SPISettings*** con los parámetros para la transferencia.

En dependencia de la naturaleza de los parámetros es posible utilizar esta función de dos formas diferentes.

Cuando **los parámetros a utilizar son constantes**, es decir que siempre serán los mismos, es posible declarar el objeto ***SPISettings*** en la propia llamada a la función. Tal y como se muestra en el siguiente ejemplo.

```
1 SPI.beginTransaction(SPISettings(10000000, MSBFIRST, SPI_MODE1));
```

Esta sintaxis provoca que el código obtenido se ejecute más rápido.

En caso de que los parámetros no sean constantes es necesario declarar el objeto ***SPISettings*** y después utilizarlo como parámetro al ejecutar la función. En el siguiente ejemplo lo puedes ver.

```
1 SPISettings spi_sett(speedMax, datOrd, datMod);

2 SPI.beginTransaction(spi_sett);
```

Esta función debe ejecutarse antes de poner el **pin SS** del esclavo a nivel bajo.

SPI.transfer()

Esta función envía y recibe datos con un esclavo previamente seleccionado. Se puede ejecutar de 2 formas diferentes en dependencia de la cantidad de parámetros.

Cuando una función puede ser llamada de diferentes maneras dependiendo del número de parámetros o el tipo de estos, se dice que es una **función o un método sobrecargado**.

La **primera sobrecarga** de la función admite **un parámetro**.

```
1 byte ret = SPI.transfer(byte2send);
```

Donde:

- ***ret*:** es el valor leído desde el pin MISO en esa transacción.
- ***byte2send*:** es el valor a enviar al esclavo (mediante el pin MOSI).

La **segunda sobrecarga** de la función admite **dos parámetros**.

```
1 SPI.transfer(buffer, bufferSize);
```

Donde:

- ***buffer*:** es un arreglo de bytes donde están almacenados los bytes a enviar. Además, es utilizado para almacenar los bytes recibidos en la transferencia.
- ***bufferSize*:** representa la cantidad de elementos a enviar y recibir por el bus SPI.

SPI.endTransaction()

La función ***SPI.endTransaction()*** es la encargada de liberar el **bus SPI** una vez se han terminado las transferencias a realizar. En otras palabras, permite que otras librerías puedan utilizar nuevamente el bus.

Su sintaxis es:

```
1 SPI.endTransaction();
```

Esta función debe ser ejecutada antes de establecer el pin SS del esclavo en estado alto.

Ejemplo comunicación SPI con Arduino y sensor BMP280

En este ejemplo práctico vamos a ver cómo comunicar el sensor de temperatura y presión BMP280 utilizando el bus SPI con Arduino.

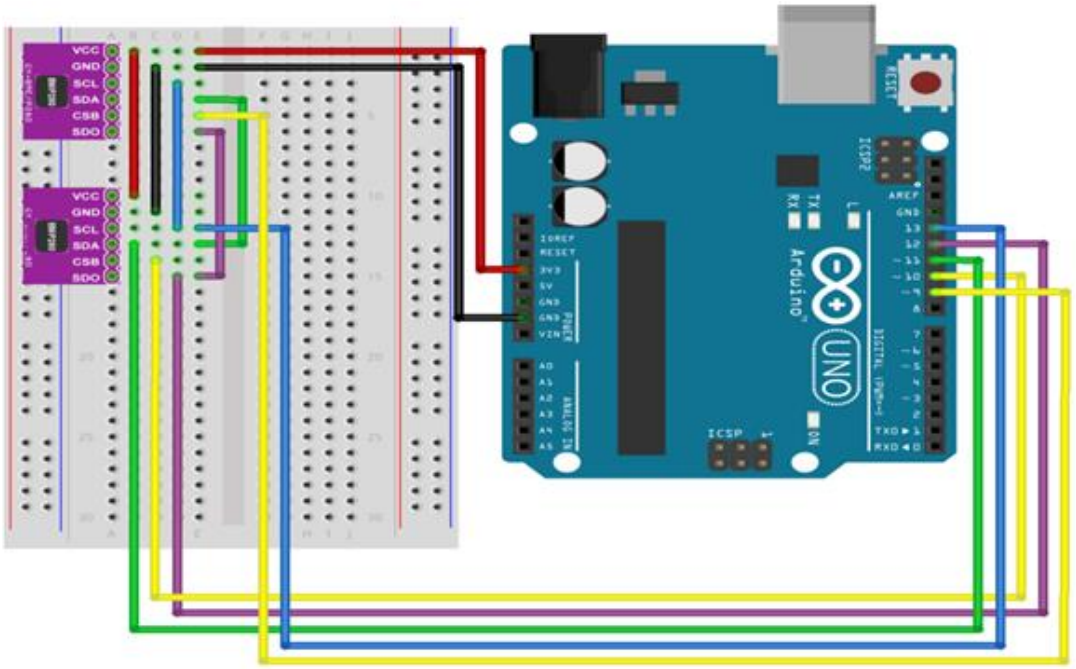
Hay que asegurarse que el sensor BMP280 tenga 6 pines ya que hay sensores con solo 4 pines que solo permiten comunicación por I2C.

Este permite medir tanto la temperatura como la presión atmosférica y gracias a la librería de Adafruit, *Adafruit_BMP280*, vamos a poder trabajar con él mediante el protocolo I2C como con el bus SPI.

En este caso voy a utilizar 2 sensores para poder probar el funcionamiento del bus SPI con Arduino.

Conexión BMP280 con Arduino utilizando SPI

El siguiente esquema eléctrico te muestra cómo conectar dos sensores BMP280 utilizando el bus SPI con Arduino



Cada sensor BMP280 tiene 6 pines.

- **VCC**: pin de alimentación. Admite una tensión de 1,8V a 3,3V
- **GND**: toma de tierra.
- **SCL (SCK)**: pin de señal de reloj en protocolo I2C y señal de reloj bus SPI.
- **SDA (MOSI)**: pin de datos en protocolo I2C y pin MOSI bus SPI (salida maestro).
- **CSB (SS)**: pin SS para seleccionar el esclavo en el bus SPI.
- **SDO (MISO)**: pin MISO bus SPI (entrada maestro).
-

La conexión de los pines es la siguiente

PIN	SENSOR 1	SENSOR 2
VCC	3,3V	3,3V
GND	GND	GND
SCL (SCK)	13	13
SDA (MOSI)	11	11
CSB (SS)	9	10
SDO (MISO)	12	12

Conexión pines de cada sensor con Arduino

Las conexiones los pines SCK, MOSI y MISO de los dos sensores van a los mismos pines de Arduino, es decir, comparte conexión. Sin embargo, para el pin de SS de selección de esclavo cada sensor se conecta con un pin diferente.

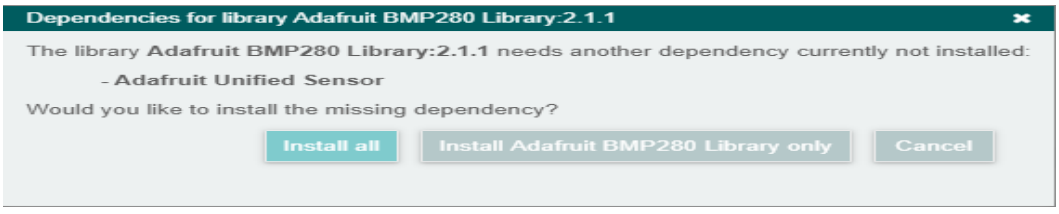
El sensor 1 se conecta al pin 9 y el sensor 2 al pin 10.

Y una vez conectado pasamos a ver el código.

Programación BMP280 SPI con Arduino

Para poder programar este sensor vamos a utilizar la librería de Adafruit *Adafruit_BMP280*. En el gestor de librerías busca la librería y la instalas.

Cuando la intentas instalar te pregunta si quieres instalar otras librerías que son necesarias para que la librería de Adafruit funcione. Haz clic en *Install all*.



Ahora ya está todo listo. El código que voy a utilizar es el siguiente.

```
1  #include <Wire.h>
2  #include <SPI.h>
```

```
#include <Adafruit_BMP280.h>

#define BMP_SCK  (13)
#define BMP_MISO (12)
#define BMP_MOSI (11)
#define BMP_CS_2 (9)
#define BMP_CS_1 (10)

// Instancia de los dos sensores
Adafruit_BMP280 sensor1(BMP_CS_1);
Adafruit_BMP280 sensor2(BMP_CS_2);

//Adafruit_BME280 bme(BME_CS, BME_MOSI, BME_MISO, BME_SCK); //
Cuando se configura por software

void setup() {
  Serial.begin(9600);
  Serial.println(F("Iniciando ejemplo BMP280 SPI"));
  Serial.println("----------------------------");

  if (!sensor1.begin()) {
    Serial.println(F("No se ha encontrado el primer sensor (9)"));
    while (1);
  }

  if (!sensor2.begin()) {
    Serial.println(F("No se ha encontrado el segundo sensor (10)"));
    while (1);
  }

}

void loop() {
    // Mostrar información sensor 1
    Serial.println("Sensor 1");
    mostrarInfoSensor(sensor1);
    Serial.println("Sensor 2");
    mostrarInfoSensor(sensor2);
    Serial.println("-----------------------------------------");

    Serial.println();
    delay(5000);
}

void mostrarInfoSensor(Adafruit_BMP280 sensor){
      Serial.print(F("Temperatura = "));
    Serial.print(sensor.readTemperature());
    Serial.println(" *C");

    Serial.print(F("Presion = "));
    Serial.print(sensor.readPressure());
    Serial.println(" Pa");

    Serial.println();
  }
```

Lo primero es importar todas las librerías necesarias. Luego crea 5 constantes para almacenar los pines SCD, MISO, MOSI y los dos pines SS de selección de esclavo.

Luego hay que crear dos objetos de la clase *Adafruit_BMP280* que se llaman *sensor1* y *sensor2*. Este constructor está sobrecargado y puede admitir 0, 1 o 4 parámetros.

Cuando no admite parámetros estás indicando que vas a trabajar con I2C, cuando indicas un único parámetro le estás diciendo que vas a utilizar el bus SPI con Arduino por hardware. El parámetro indica el pin donde has conectado el SS (selector de esclavo).

Por último, cuando indicas todos los pines del bus SPI con Arduino estás diciendo a la librería que vas a trabajar a través de SPI por software.

En nuestro caso utilizamos la sobrecarga que admite un único parámetro y pasamos el pin SS de cada sensor.

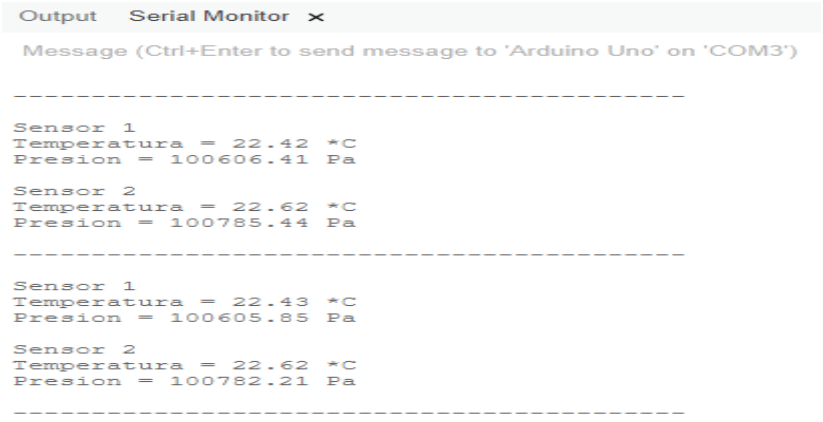
En la función *setup()* llamamos a la función *begin()* para iniciar cada sensor. En caso de no poder iniciar el sensor muestra un mensaje por el monitor serie y no continúa.

En la función es donde vamos a consultar la temperatura y la presión atmosférica. He creado una función que se llama *mostrarInfoSensor(sensor)* que admite un parámetro que es el objeto de la clase *Adafruti_BMP280*.

Dentro de esta función se consulta la temperatura con la función *readTemperature()* y la presión con *readPressure()*. Luego se muestra la información por el monitor serie.

Por último se hace una espera de 5 segundos entre cada medida.

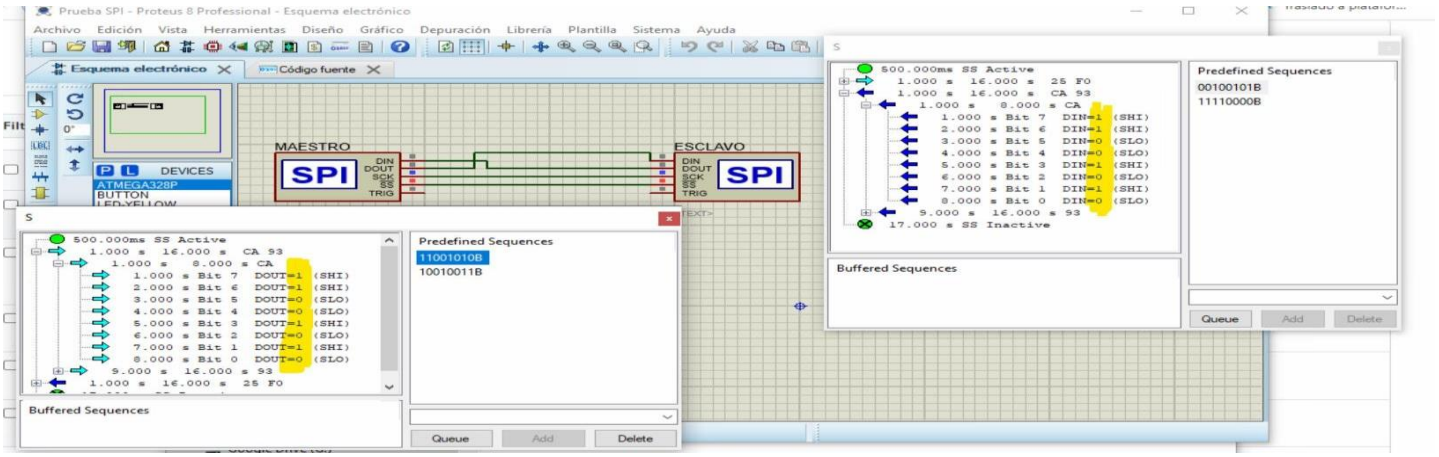
Solo queda una cosa, comprobar que todo funciona correctamente cargando el código a Arduino y abriendo el monitor serie.



Aparecerá algo parecido a la anterior imagen.

También a través del SPI Debugger de Proteus.

Prueba con un dispositivoMaestro y uno Esclavo, el primer dato que envía el maestro (11001010) DOUT (datossalientes) y luego lo vemos como DIN (datos entrantes) en el esclavo aquí vemos cada bit recibido.



Luego en la siguiente imagen podemos ver cuando el esclavo le envía datos al maestro, aquí vemos los bits que envía elesclavo (DOUT) y luego los vemos reflejados en el maestro (DIN).