URJC Escuela Técnica Superior de Ingeniería de Telecomunicación

GSyC

Sistemas Empotrados y de Tiempo Real

Comunicaciones en Arduino

Grado en Ingeniería de Robótica Software

Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación

Roberto Calvo Palomino
roberto.calvo@urjc.es

Comunicaciones

- UART
- I2C
- SPI
- CANBUS

- Ethernet/WiFi
- MQTT y NodeRed

Comunicación Serie

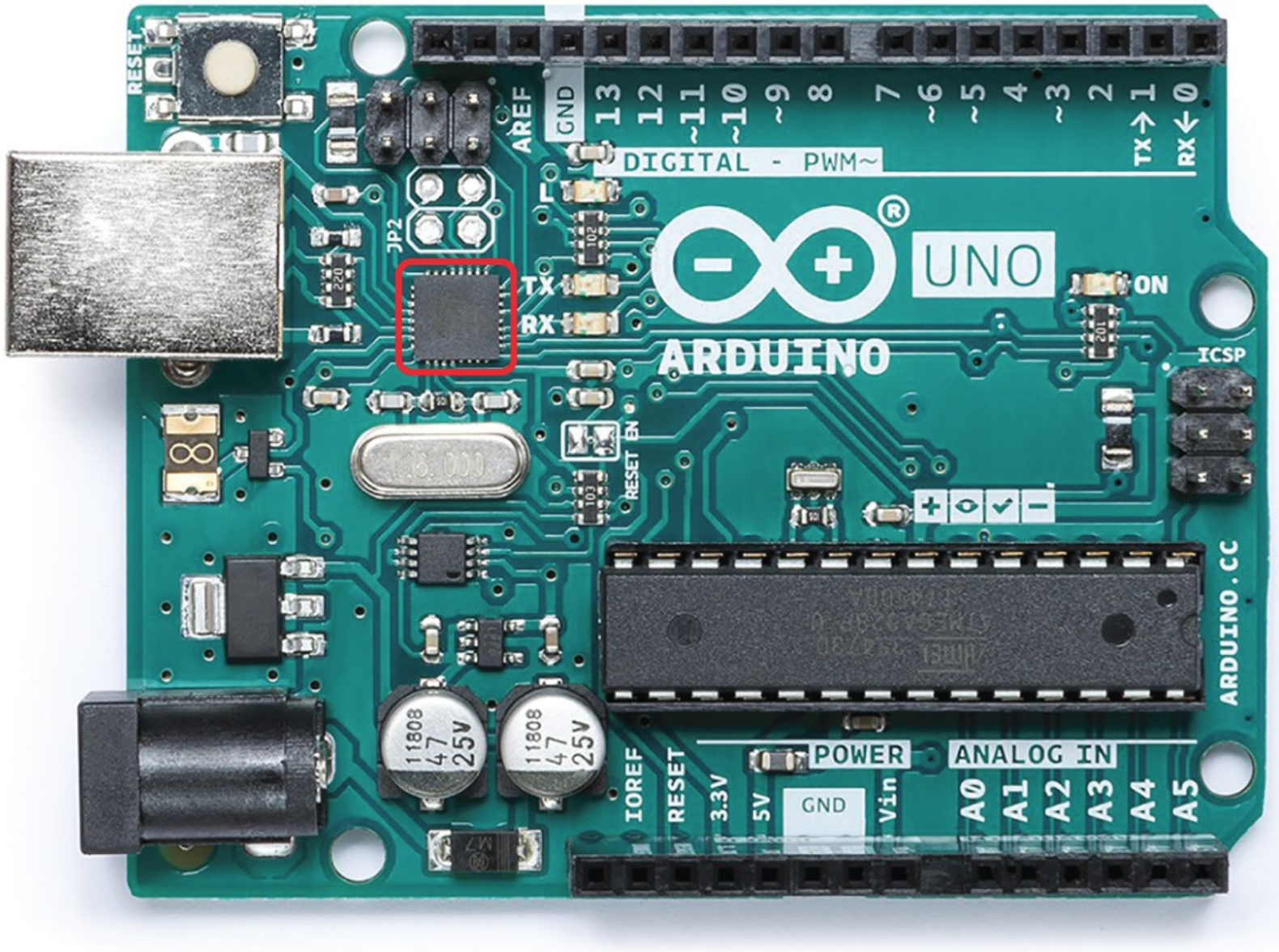
- La **comunicación serie** en telecomunicaciones es el proceso de envío de datos de un bit a la vez, de forma secuencial, sobre un canal de comunicación o bus.
- Multitud de protocolos y sistemas
  - UART, RS-232, I2C, SPI, CAN, USB, JTAG, …
  - ¡Incluso código morse!
- Comunicación puede ser:
  - Asíncrona: No hay reloj común (UART, RS-232)
  - Síncrona: Hay reloj común para la transferencia (I2C, SPI)

Comunicación UART

Comunicación UART

- Todas las placas Arduino contienen al menos un puerto serie para comunicarse con otro ordenador o dispositivos.
- También denominado UART:
  - Universal Asynchronous Receiver-Transmitter
- Inicialmente utilizado para imprimir mensajes y verlos a través del monitor de logs
- También podemos utilizar la comunicación serie para comunicar diferentes Arduinos entre si, o con otros dispositivos.
  - Chips GPS normalmente incorporan puerto serie para la comunicación.

Comunicación UART

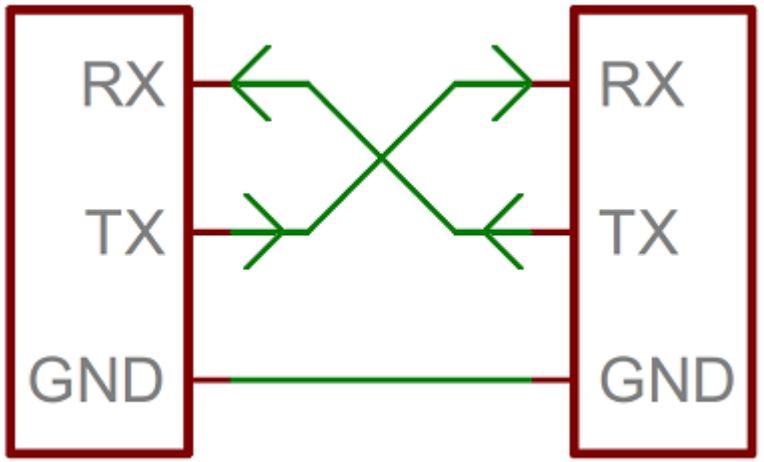


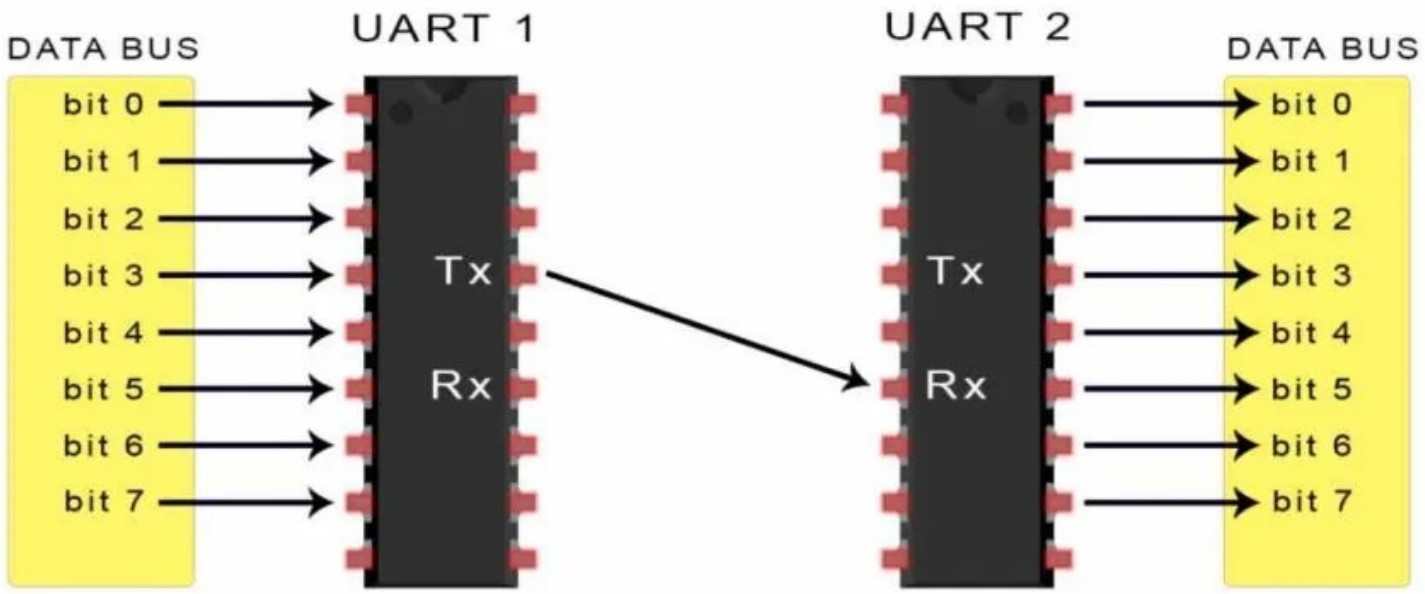
Comunicación UART

- Comunicación serie solo utiliza 2 pines:
  - TX para envío
  - RX para recepción
- No hay señal de reloj
- Los pines de comunicación TX/RX utiliza niveles lógicos TTL(Transistor-Transistor Logic).
  - Arduino utiliza 5V o 3.3V
  - Pero cuidado con RS232, utiliza 12V
- Utilizar TTL-USB para conectar con seguridad.

Comunicación UART

- Siempre conectar:
  - TX #1 con RX #2
  - RX #1 con TX #2





Comunicación UART

- Velocidades (en baudios)
  - 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200
- En telecomunicaciones, **baudios** es una medida utilizada que representa el número de símbolos por segundo en un medio de transmisión digital.
- Cada símbolo puede comprender 1 o más bits, dependiendo del esquema de modulación.
- En el esquema de comunicación serie 1 símbolo = 1bit
- 9600 baudios = 9600 bits/s

Comunicación UART

- Es posible comunicarse con otro Arduino mediante comunicación serie utilizando las siguientes funciones
  - print()-readBytes() y write()-read()

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

void loop() {

  if (Serial.available()) {
    buffer[Serial.readBytesUntil('\n', buffer, 256)] = '\0';

    Serial.println(String("ECHO: ") + buffer);
  }
  delay(1);
}
```

- Referencia a la librería:

  https://www.arduino.cc/reference/en/language/functions/communication/serial/

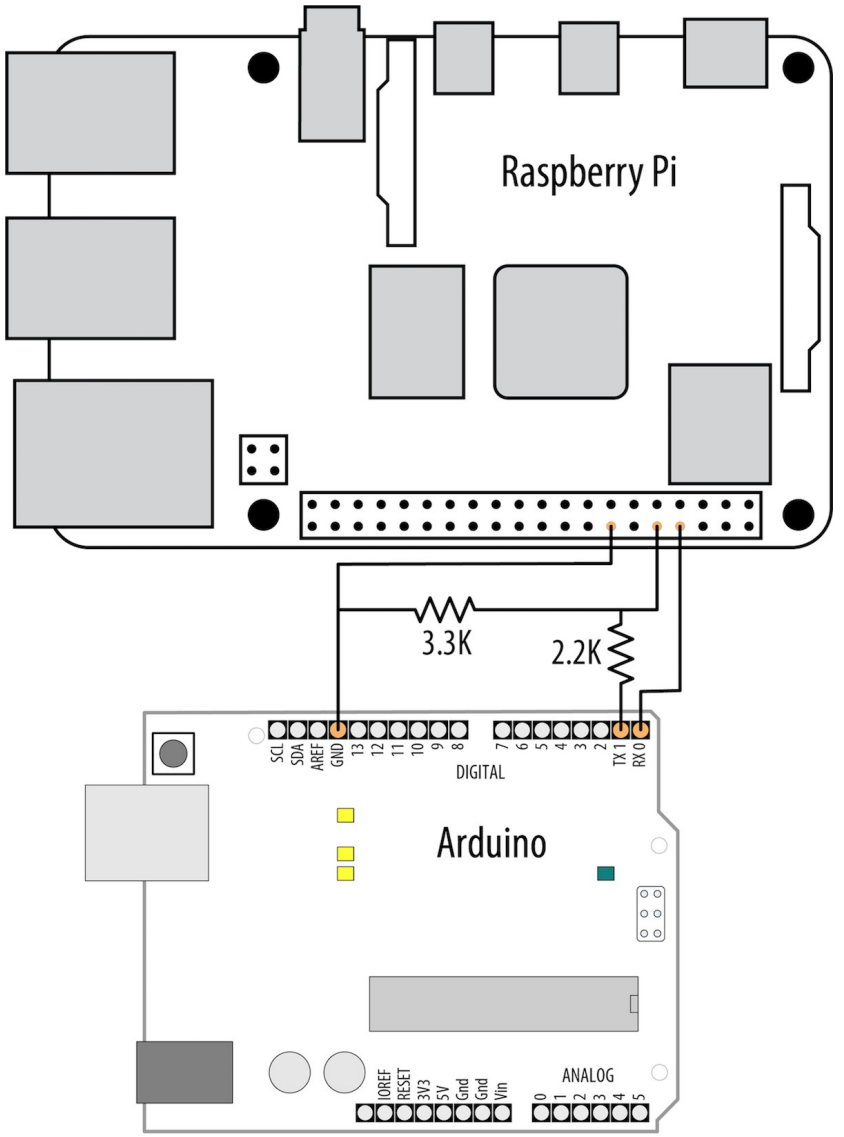
Comunicación UART: Ejemplos

- Comunicar Arduino con RaspberrypPi (python)

```python
#!/usr/bin/env python

import serial
from time import sleep

ser = serial.Serial('/dev/serial0', 9600)
ser.write('P13=1')
sleep(1)
ser.write('P13=0')
```



- Arduino Cookbook (3rd edition)

Comunicación UART: Resumen

- ✔ 2 líneas para envío y recepción
- ✔ Comunicación Asíncrona (no se comparte reloj).
- ✔ Velocidades entre 300 bps y 460 kbps
- ✔ Mecanismo simple

- ✗ Problemas a más de 15 metros.
- ✗ Limitado para la comunicación entre 2 dispositivos
- ✗ La velocidad se debe acordar al inicio, si no es la misma habrá problemas en la integridad de los datos recibidos.

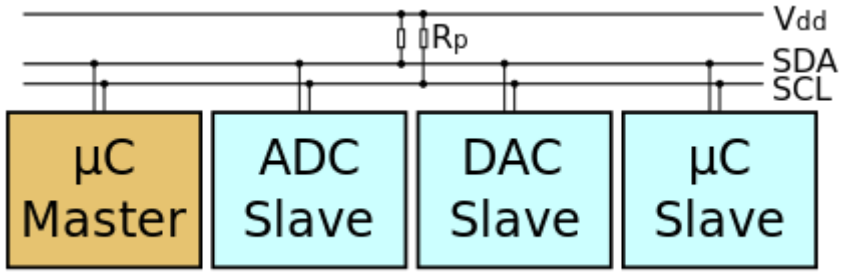
Comunicación I2C

Comunicación I2C

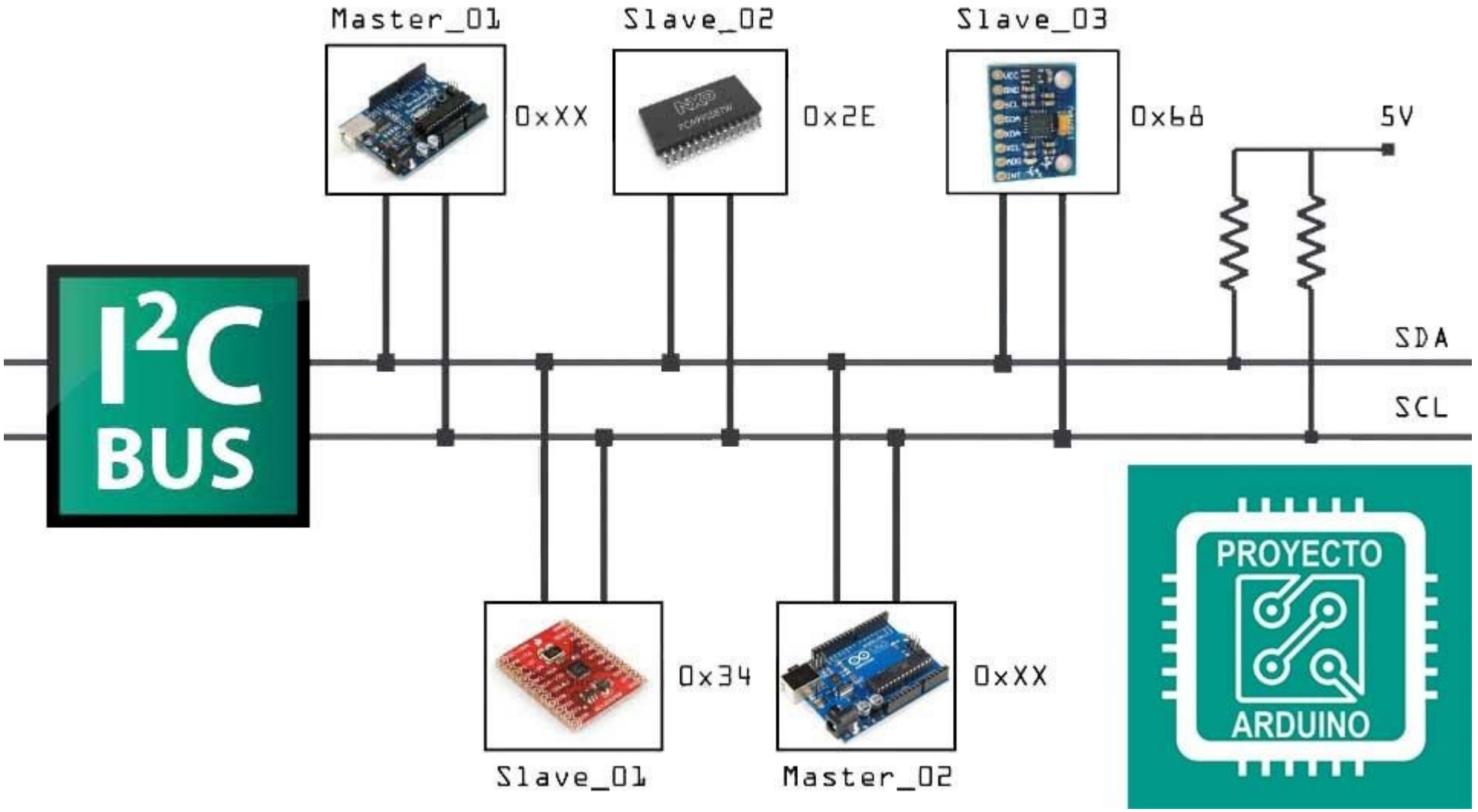
- I2C: Inter-Integrated Circuit
- Protocolo de comunicación serie desarrollado por Philips (1982)
- Muy utilizado para añadir dispositivos de baja velocidad en microcontroladores en una distancia corta.
- Pensado para comunicación dentro de una placa.
- Lider-seguidor (maestro-esclavo)
- Numero ilimitado de lideres y un máximo de 1008 seguidores.
- Es un protocolo **síncrono** y hay una señal de reloj común.
- Velocidades: 100 kbps – 400 kbps (max 5 Mbps)
- Los dispositivos esclavo conectados al bus I2C se identifican por la dirección de chip, definida por el hardware/software

Comunicación I2C

- I2C utiliza sólo 2 pines de comunicación:
  - SDA (Serial Data). Canal de intercambio de información.
  - SCL (Serial Clock). Canal donde viaja la señal de reloj.
- Voltajes típicos son 3.3V o 5V
- El diseño de referencia de I2C tiene direcciones de 7bits



I2C

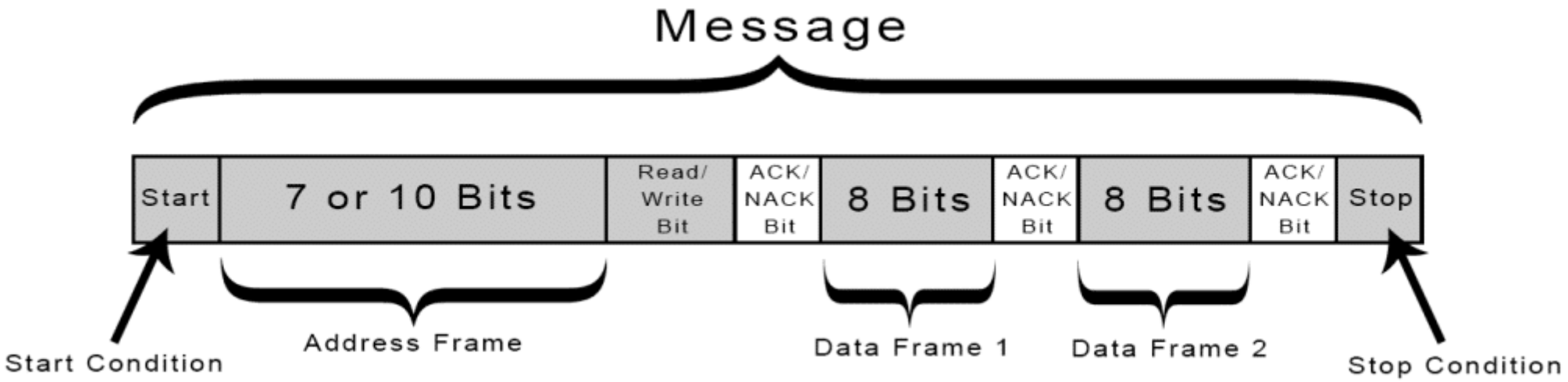

Master_01
0xXX
Slave_02
0x2E
Slave_03
0x68
5V
I²C BUS
SDA
SCL
Slave_01
0x34
Master_02
0xXX
PROYECTO
ARDUINO

Comunicación I2C

- El diseño del bus I2C tiene dos roles diferenciados
  - **Nodo Líder** (máster): Genera la señal de reloj e inicia la comunicación con los nodos seguidores.
  - **Nodo Seguidor** (esclavo): Recibe la señal de reloj y responde cuando es definido por el líder.
- El bus de comunicaciones es un **multi-lider** bus, lo que significa que varios nodos lideres pueden estar presentes.
- Los roles (líder, seguidor) pueden ser intercambiados por ciertos nodos.

Comunicación I2C

- Los mensajes en I2C se componen de 2 partes:
  - **Address frame**: El nodo líder especifica a qué nodo seguidor va dirigido el mensaje.
  - **Data frame**: El mensaje de 8 bits que es enviado desde el nodo líder hacía el nodo seguidor (o viceversa).

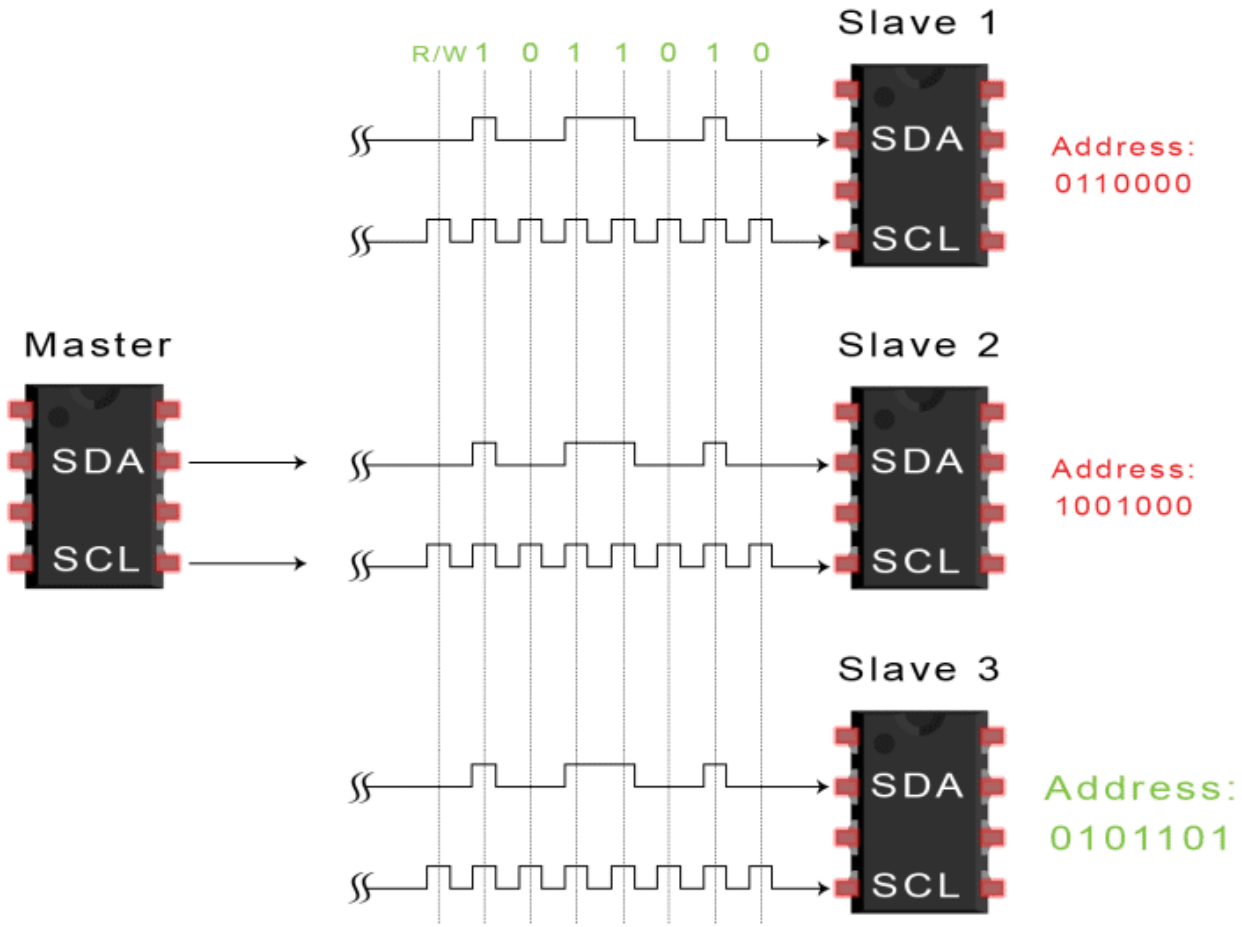


Comunicación I2C

- Para iniciar una comunicación el nodo lider:
  - Mantiene SCL en HIGH
  - Pone SDA en LOW
- Los nodos seguidores se preparan para la comunicación que va a comenzar.
- ¿Qué ocurre si 2 nodos lideres requieren el control del bus al mismo tiempo?
- Obtendrá el control de bus aquel nodo que ponga antes la señal SDA a LOW

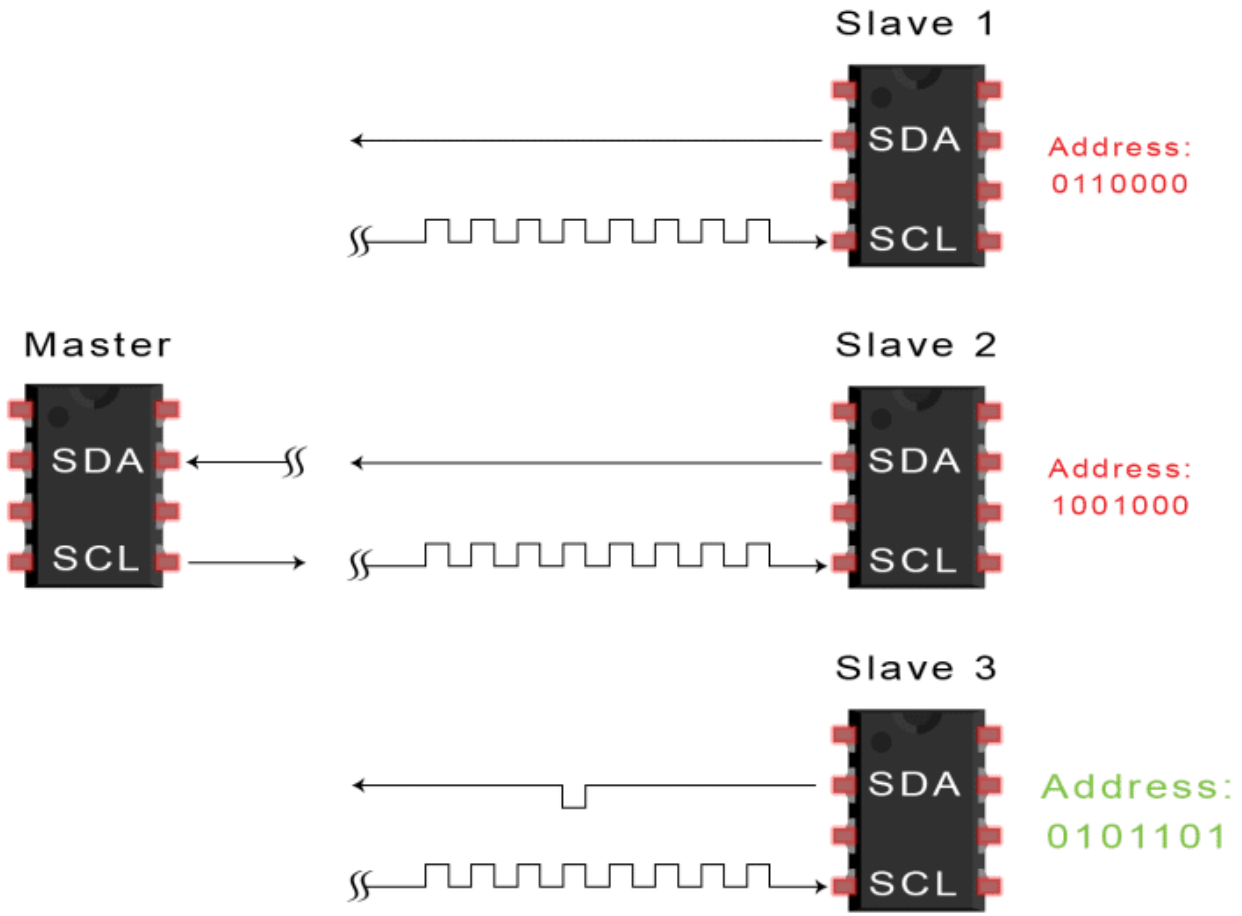
Comunicación I2C

- Una vez iniciada la comunicación con START, el nodo líder envía el “address frame” con la dirección del nodo seguidor.



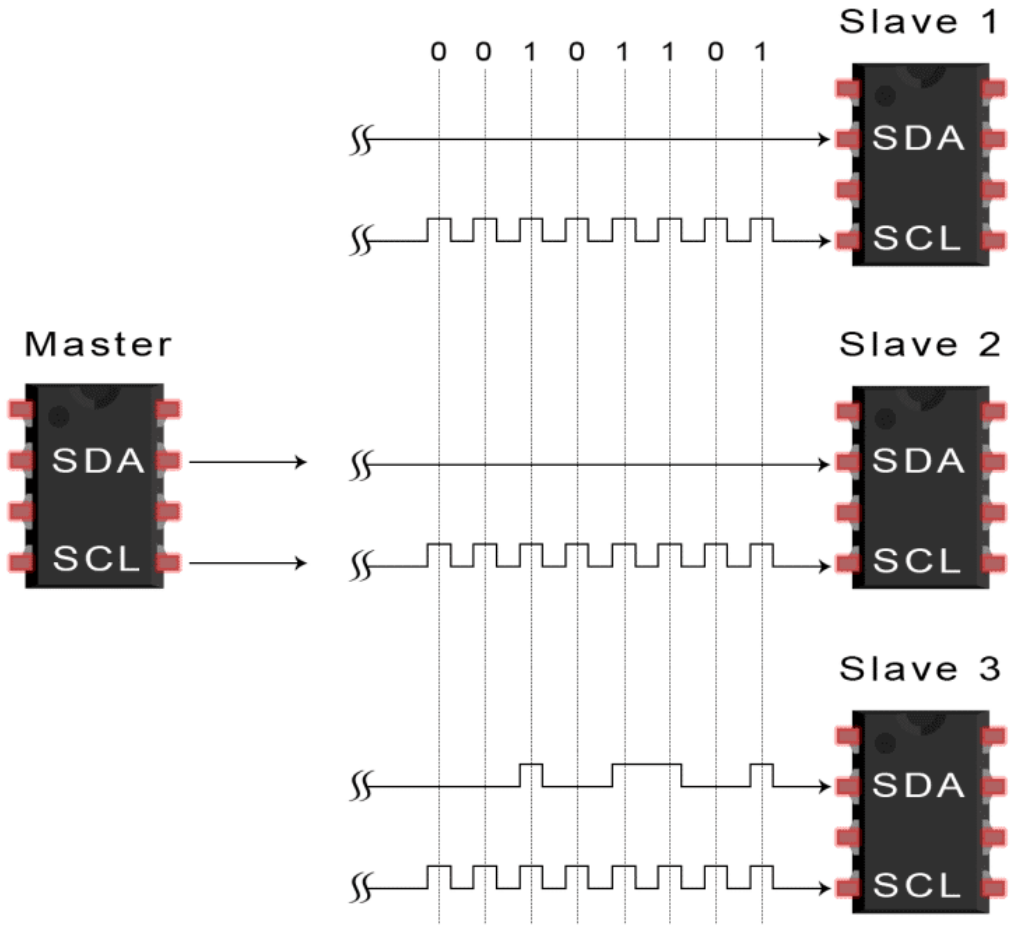
Comunicación I2C

- El nodo seguidor que coincida con la dirección especificada contestará con el bit ACK



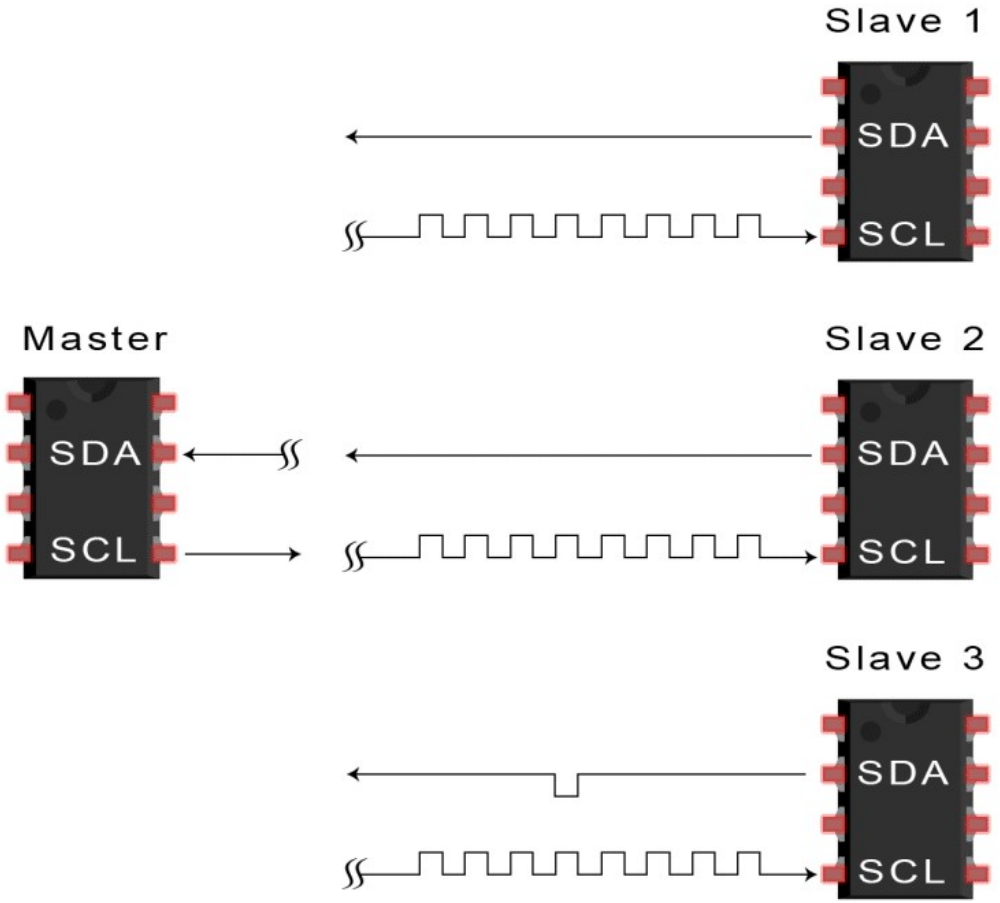
Comunicación I2C

- El nodo líder envía o recibe el data frame:



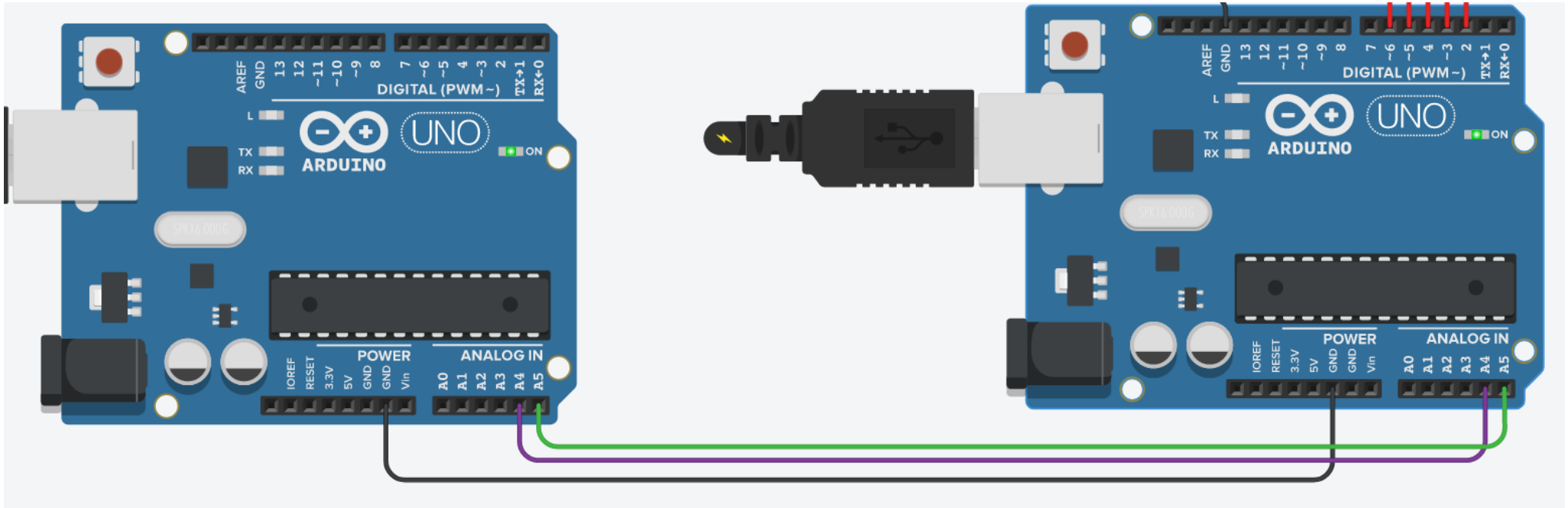
Comunicación I2C

- Después de cada envío, el nodo que recibe envía el ACK al nodo que envía para notificar la recepción correcta del mensaje.



Comunicación I2C: Arduino

- En Arduino disponemos de la librería ‘Wire’ que implementa la comunicación I2C
  - https://www.arduino.cc/en/reference/wire
- En la placa Arduino UNO:
  - A4 corresponde con SDA y A5 corresponde con SCL



Comunicación I2C: Arduino

Master/Lider

```
#include <Wire.h>

const byte I2C_MASTER_ADDR = 0x10;
const byte I2C_SLAVE_ADDR = 0x20;

const byte SEND_PIN = 2;
const byte SEND_STATE = 1;

void setup() {

  // Añadimos este dispositivo al bus
  Wire.begin(I2C_MASTER_ADDR);
}

void loop() {

  Wire.beginTransmission(I2C_SLAVE_ADDR);
  Wire.write(SEND_PIN);
  Wire.write(SEND_STATE);
  Wire.endTransmission();

  delay(1);
}
```

Slave/Seguidor

```
#include <Wire.h>

const byte byte I2C_SLAVE_ADDR = 0x20;

void setup() {

  // Añadimos este dispositivo al bus
  Wire.begin(I2C_SLAVE_ADDR);
  // Registramos evento de recepción
  Wire.onReceive(receiveEvent);
  Serial.begin(9600);
}

void loop() {
  delay(300);
}

void receiveEvent(int howMany) {

  if (Wire.available() == 1) {
     pin = Wire.read();
     Serial.print("PIN: " + String(pin));
  }
  if (Wire.available() == 2) {
     state = Wire.read();
     Serial.print("STATE: " + String(state));
  }
}
```

Comunicación I2C: Resumen

- ✔ Mayor velocidad que comunicación serie
- ✔ Muy extendido
- ✔ Utiliza únicamente 2 líneas
- ✔ Múltiples líderes y seguidores
- ✔ ACK/NACK permite confirmación de la transferencia

- ✗ Más lento que SPI
- ✗ Hardware más complicado
- ✗ Data frame limitado a 8 bits.

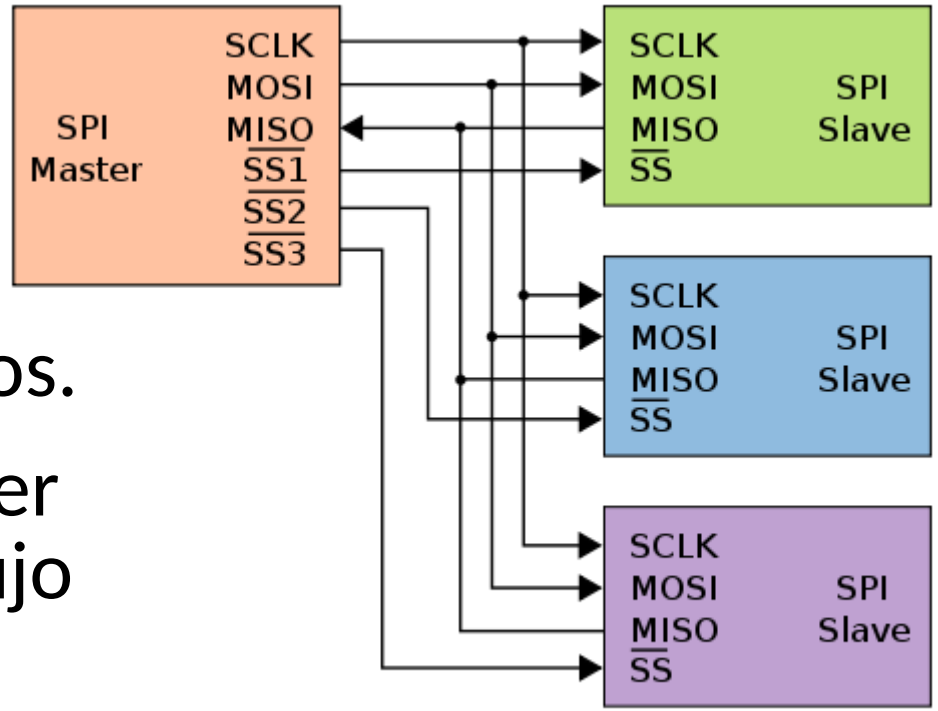
Comunicación SPI

Comunicación SPI

- SPI (Serial Peripheral Interface Bus)
- Desarrollado por Motorola (1980)
- Comunicación sincrona.
- Utilizado para comunicación a distancias cortas (on board)
- Utiliza arquitectura líder-seguidor (maestro-esclavo)
- Permite comunicación full-duplex
- Permite un único líder en el sistema.
- Hace uso de 4 líneas de comunicación.
- Velocidades

Comunicación SPI

- SCLK: señal de reloj para sincronizar los datos.
- MOSI: Master Output Slave Input
- MISO: Master Input Slave Output
- SS: Selección del Slave



- Multitud de seguidores (slaves) están soportados, utilizando una línea individual por cada uno de ellos.
- Cualquier numero de bits pueden ser enviados/recibidos en un mismo flujo continuo.

Comunicación SPI

- Funcionamiento del protocolo

1) El nodo líder configura el reloj (SCLK)

2) El nodo líder selecciona el seguidor con el que quiere comunicarse activando la linea de selección (SS)

3) Durante cada ciclo de SPI, la comunicación full-duplex es permitida.

   El nodo líder manda información (MOSI)

   El nodo seguidor manda información (MISO)

4) La transmisión puede continuar por los ciclos de reloj necesarios.

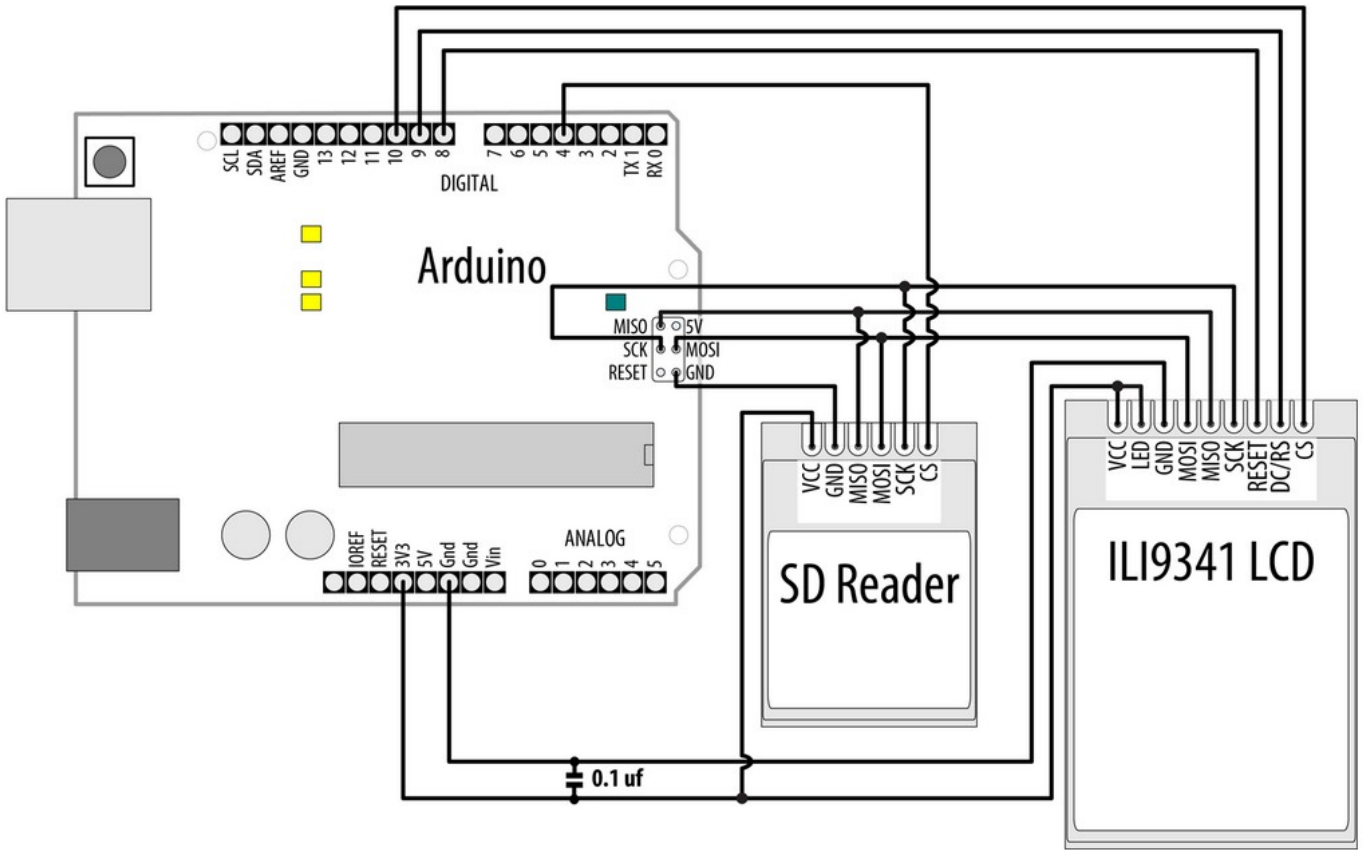
5) Cuando la transmisión está completada, el líder para la señal de reloj y de-selecciona al nodo seguidor.

Comunicación SPI: Arduino

- En Arduino disponemos de la librería ‘SPI’ que implementa la protocolo de comunicación SPI
- En Arduino UNO los pines a utilizar son:
  - SCK: Pin 13
  - MOSI: Pin 11
  - MISO: Pin 12
  - SS: Pin 10
- Muchos de los sensores que usan SPI tienen sus propias librerías para interactuar con ellos.

Comunicación SPI: Ejemplo

- Arduino CookBook (3rd edition)
- Lector SD y LCD funcionan con el protocolo SPI



Comunicación SPI: Resumen

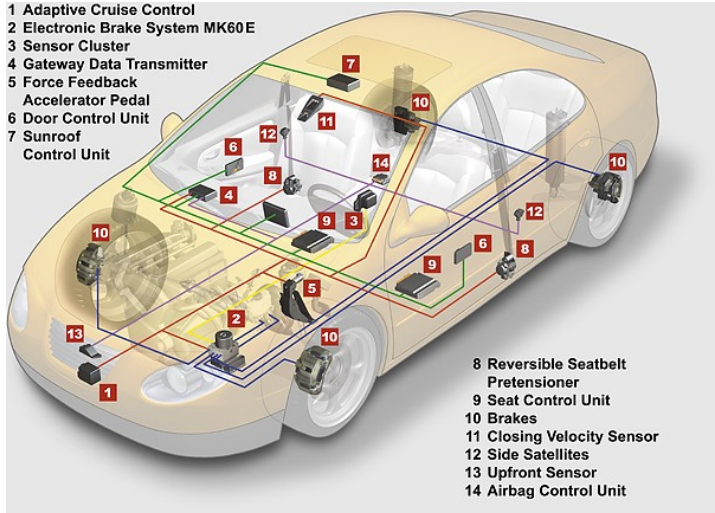
- ✔ Mayor velocidad
- ✔ Full-duplex
- ✔ No está limitado a palabras de 8 bits.
- ✔ Hardware más sencillo, multitud sensores compatibles.

- ✗ Requiere más lineas.
- ✗ Direccionamiento a través de lineas específicas
- ✗ Longitud del mensaje debe ser conocido por ambas partes.
- ✗ No hay ACK en la transmisión.
- ✗ Solo soporta un único nodo líder (máster)

CAN BUS

CAN BUS

- Controller Area Network (CAN)
- Protocolo de comunicaciones
- Creado por BOSH en 1983
- Mercedes fue uno de los primeros en usarlo (1992)
- Primeros estándares ISO en 1993-1994
- Su misión principal es intercomunicar las diferentes ECU (Electronic control units) de un sistema.



CAN BUS

- Conexión entre multitud de microcontroladores y ECU a través de únicamente 1 par de cables.
- Alta velocidad (1MBit/s) y transmisión en tiempo real
- Alta inmunidad a interferencias en un entorno con alto ruido electromagnético
- Bajo coste
- Distancias de BUS de hasta 1 Km.

Bus Length (m)	Signaling Rate (Mbps)
40	1
100	0.5
200	0.25
500	0.10
1000	0.05

CAN BUS

- Es una red cerrada, por lo que no se implementan característica de seguridad, login o cifrado.
- Prioridad de mensajes.
- Garantía de tiempos de latencia.
- Flexibilidad en la configuración.
- Recepción por multidifusión (multicast)
- Sistema multimaestro.
- Retransmisión automática de tramas erróneas
- Funciona a través de señales lógicas CAN_HIGH y CAN_LOW

CAN BUS

- Tramas (mensajes)

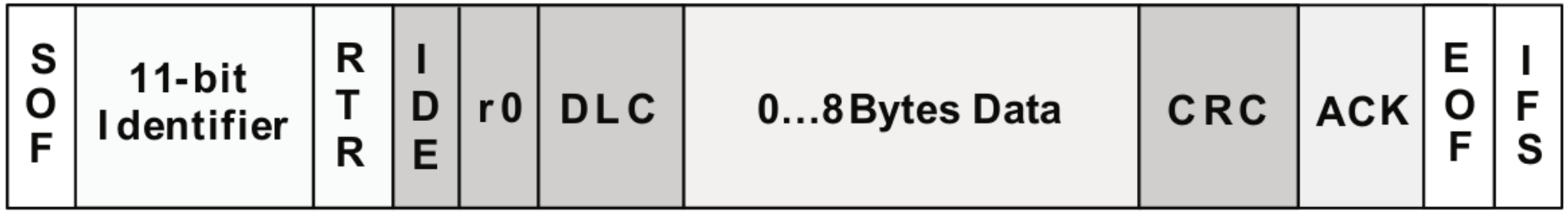


Figure 3. Standard CAN: 11-Bit Identifier

Figure 4. Extended CAN: 29-Bit Identifier

CAN BUS

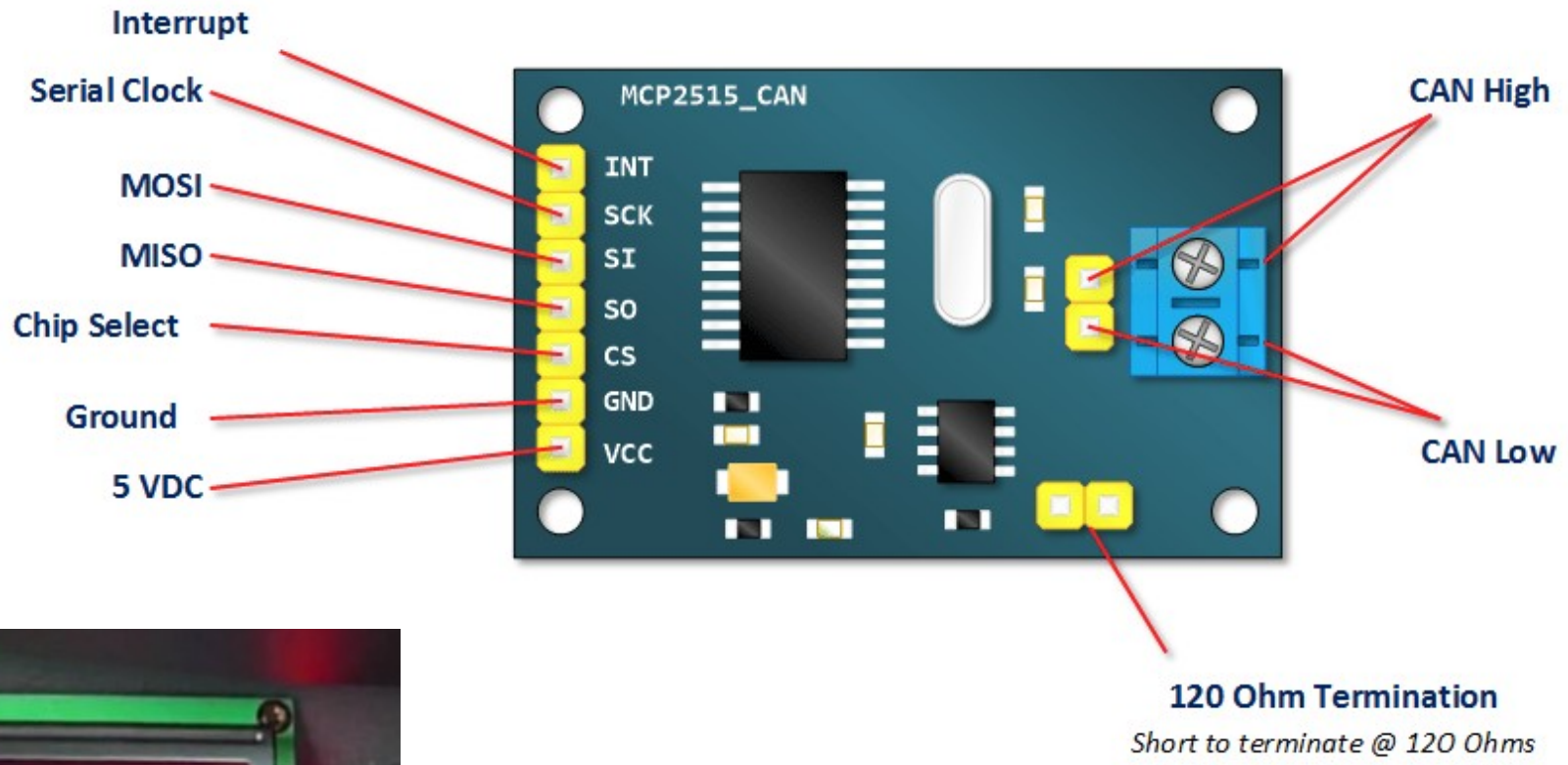
- Normalmente en un automóvil nos encontramos 2 redes CANBUS diferenciadas
  - Una red de alta velocidad (hasta 1 Mbit/s), bajo el estándar ISO 11898-2
    - Destinada para controlar el motor e interconectar las unidades de control electrónico (ECU);
  - Una red de baja velocidad tolerante de fallos (menor o igual a 125 kbit/s), bajo el estándar ISO 11519-2/ISO 11898-3
    - Comunicación de los dispositivos electrónicos internos de un automóvil como son control de puertas, ventanillas, luces y asientos

CAN BUS - OBD

- OBD es un protocolo de diagnóstico a bordo que se comunica con el CAN BUS para realizar diagnosis o detectar fallos en sistemas de automoción.
- OBD y CAN BUS son protocolos estándar, pero cada fabricante puede crear sus propias tramas con códigos de error diferentes.

CAN BUS - Arduino

- MCP2515: Módulo de comunicación CAN BUS



Ethernet/WiFi

Ethernet

Arduino Shield Ethernet

Arduino Ethernet

Ethernet

- Arduino provee de una librería para utilizar la tarjeta de red a alta nivel:
  - https://www.arduino.cc/en/Reference/Ethernet
- Librería diseñada para funcionar con Arduino Ethernet Shield, Arduino Ethernet Shield 2, Leonardo Ethernet, y cualquier otro dispositivo basado en W5100/W5200/W5500
- Utiliza SPI para la comunicación
- Posibilidad de utilizar TCP y UDP

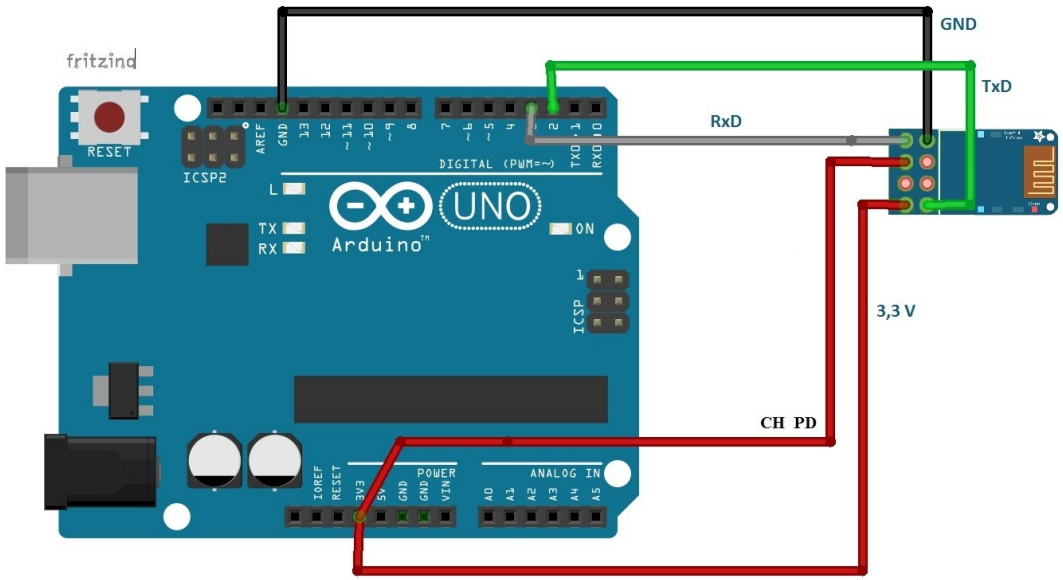
Ethernet

- Ejemplo de servidor web
  - https://www.arduino.cc/en/Tutorial/LibraryExamplesWebServer
- Muy útil para mostrar información adquirida desde los sensores.

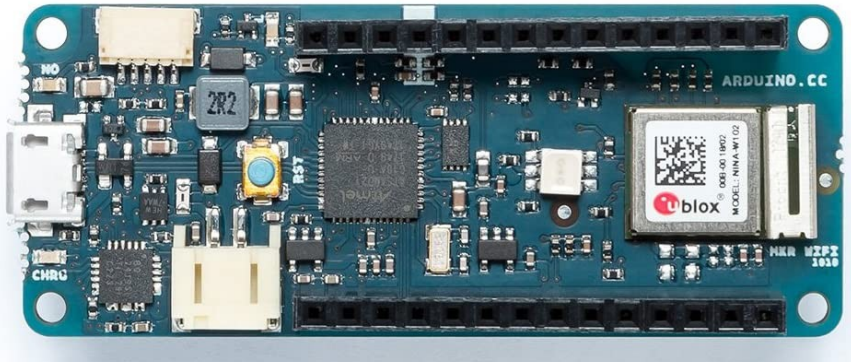
- Consultar el código:
  - https://create.arduino.cc/example/library/ethernet_2_0_0/ethernet_2_0_0%5Cexamples%5CWebServer/WebServer/preview

WiFi

ESP8266 + Arduino



Arduino MKR WiFi 1010
(Arduino Oplà IoT Kit)



WiFi

- Librería
  - https://www.arduino.cc/en/Reference/WiFiNINA
- Ejemplo de servidor web sencillo
  - https://www.arduino.cc/en/Tutorial/WiFiNINAAPSimpleWebServer

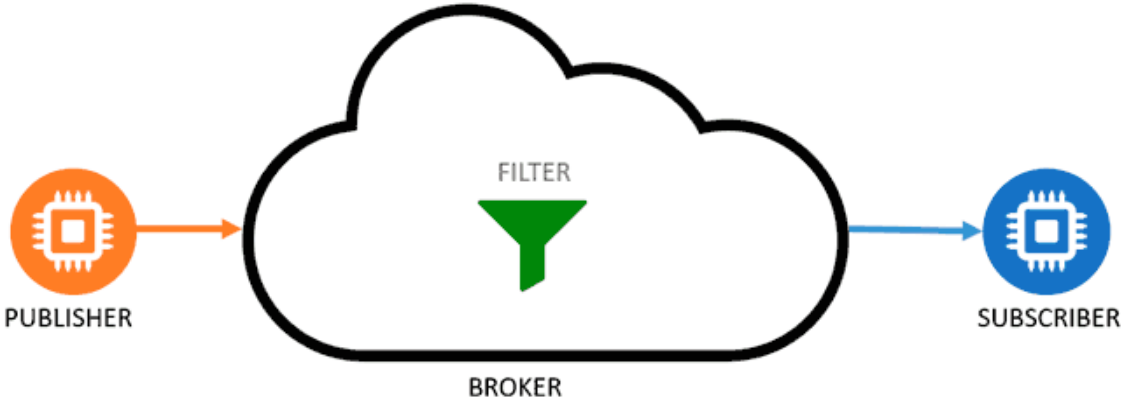
MQTT

MQTT

- Protocolo de comunicación M2M (Machine-to-Machine)
- Basado en Publicador/Subscriptor
- Muy liviano y sencillo de procesar.
- Normalmente ejecuta sobre TCP/IP
- Pensado para comunicar remotamente hardware de bajo coste, sin muchos recursos computacionales.
- Consumo energético bajo.

MQTT

- Se le denomina **protocolo de IoT** ya que se ha extendido su uso en estos dispositivos gracias a su sencillez y ligereza.
- MQTT precisa de un servidor central por donde se encaminan todos los mensajes (**Broker**)
- Hay nodos que publican mensajes a través de topics (**Publisher**)
- Hay nodos que se subscriben a ciertos topis pare recibir mensajes (**Subscriber**)



MQTT

- MQTT dispone de mecanismos de calidad de servicio (QoS).
- Diferentes niveles de QoS para la robustez de envíos:
  - **QoS 0** (at most one): El mensaje solo se envía una vez, en caso de fallo puede que no se reciba correctamente.
  - **QoS 1** (at least one): El mensaje se envía hasta que se garantice la entrega (subscriptor podría recibir mensajes duplicados)
  - **QoS 2** (exactly one): Se garantiza que el mensaje se entrega al subscriptor una única vez.
- Dependiendo de las características y necesidades de nuestro sistema usaremos el nivel apropiado de QoS.

MQTT

- Incluye capa de transporte SSL/TSL para autenticación mediante contraseña o certificados.
- Estándar en IoT
- Seguridad y calidad del servicio (QoS)
- Requiere un ancho de banda mínimo.
- Menor consumo de energía

MQTT - Demo Linux

- Mosquitto es una implementación libre de MQTT y funciona en Debian, Ubuntu, Rpi, Windows, Mac.
  https://mosquitto.org/

- Ejemplos:
  - Subscribir a todos los topics posibles

```
mosquitto_sub -v -h 0.0.0.0 -p 1883 -t '#'
```

  - Subscribir a un topic específico

```
mosquitto_sub -v -h 0.0.0.0 -p 1883 -t '/Robot/odom/'
```

  - Enviar un mensaje en un topic concreto

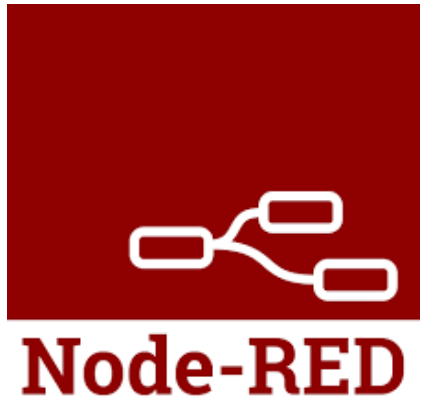
```
mosquitto_pub -h 0.0.0.0 -p 1883 -t /Robot/speed/ -m "speed:12.1"
```

MQTT - Arduino

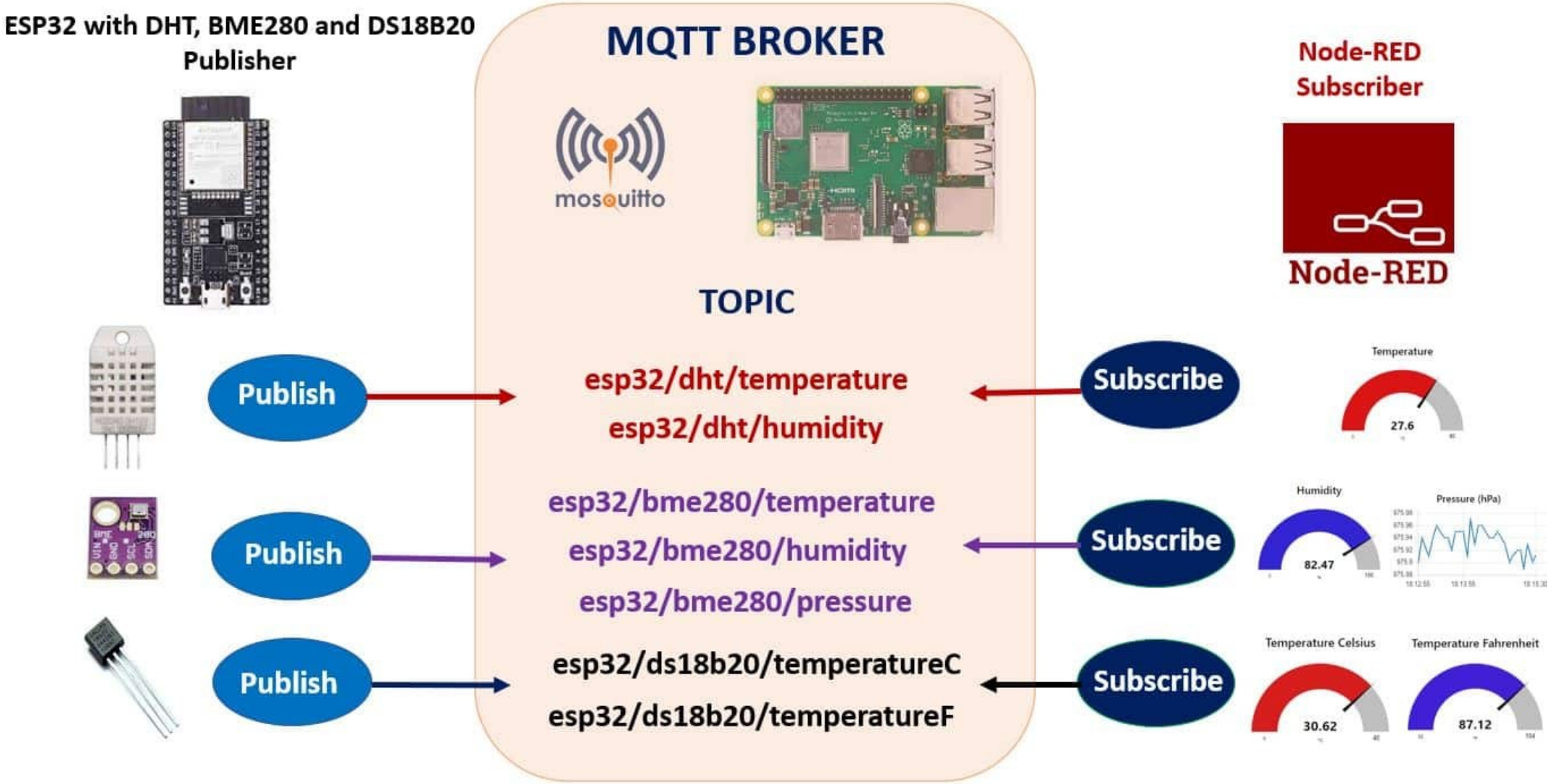
- Existen multitud de librerías para dar soporte MQTT a microcontroladores, entre ellos Arduino.
- Adafruit
  - https://github.com/adafruit/Adafruit_MQTT_Library
- Arduino Client for MQTT
  - https://github.com/knolleary/pubsubclient

MQTT - NodeRed

- Herramienta de programación visual que permite interconectar diferentes dispositivos de nuestro sistema
- Muy utilizado para gestión y procesado de datos en tiempo real y muy unido a proyectos IoT.
- Está desarrollado en NodeJS e internamente los componentes intercambian información a través de JSON.
- https://nodered.org/

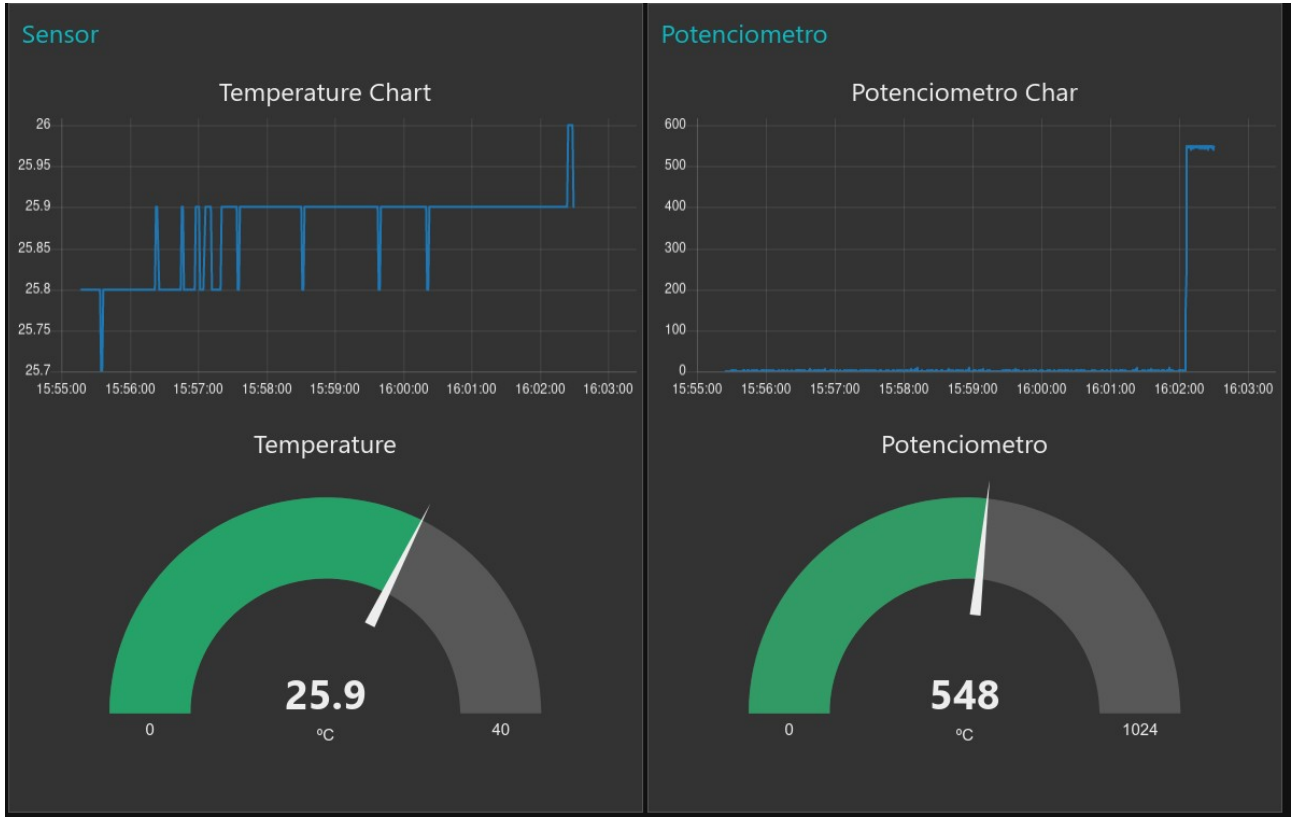


MQTT - NodeRed



NodeRed

- Ejemplo demostración
- https://gitlab.etsit.urjc.es/roberto.calvo/setr/-/tree/main/MQTT-NodeRed



Bibliografia

- Comunicación I2C
  - https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/
- Arduino Communication Peripherals: UART, I2C and SPI
  - https://www.seeedstudio.com/blog/2019/11/07/arduino-communication-peripherals-uart-i2c-and-spi/
- Controller Area Network Physical Layer Requirements
  - https://www.ti.com/lit/an/slla270/slla270.pdf

URJC Escuela Técnica Superior de Ingeniería de Telecomunicación