

Unidad 2.3

Comunicación serie Asincrónica y Sincrónica

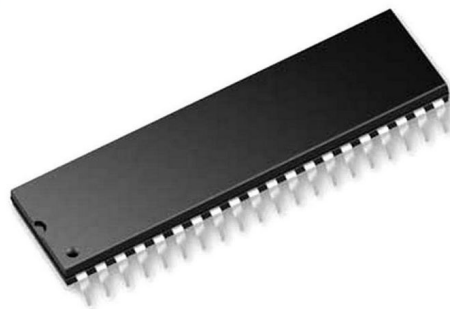
UART - SPI - I2C

V1.0.0


► Universal Asynchronous Receiver Transmitter

UART

Un **UART (Universal Asynchronous Receiver/Transmitter)** es un componente de hardware que se utiliza para facilitar la comunicación serie en una computadora u otro dispositivo electrónico. Actúa como un intermediario entre el procesador o microcontrolador y los dispositivos periféricos o externos que utilizan una interfaz de comunicación serie.



El UART se encarga de **convertir los datos paralelos del procesador en una secuencia de bits serie** para su transmisión y recibir de la misma forma y convertirlos de nuevo en datos paralelos comprensibles para el procesador. **Esto permite la transmisión y recepción de datos de manera secuencial a través de un solo cable**, lo que es útil para la comunicación entre dispositivos a larga distancia o en el caso que existan limitaciones en el número de pines disponibles.



Las aplicaciones del UART son variadas. Como ejemplos de su uso podemos mencionar:

1. Comunicación entre microcontroladores o microprocesadores y sensores externos, como sensores de temperatura, acelerómetros, giroscopios, etc.
2. Comunicación entre sistemas embebidos y dispositivos externos, como pantallas LCD, módulos de comunicación inalámbrica, controladores de motores, etc.
3. Programación y depuración de microcontroladores o dispositivos electrónicos durante el desarrollo y el proceso de prueba.
4. Transferencia de datos en sistemas de comunicación en serie, como redes RS-232, RS-485.

Para establecer una comunicación serie entre dispositivos, hay varios parámetros a tener en cuenta:

1. **Velocidad de transmisión (baud rate):** Es la cantidad de bits que se transmiten por segundo. Se mide en baudios (baudios). Ambos dispositivos deben estar configurados con la misma velocidad de transmisión para garantizar una comunicación correcta.
2. **Bits de datos (data bits):** Indica la cantidad de bits utilizados para transmitir los datos. Es común utilizar 7 u 8 bits de datos.
3. **Bit de paridad (parity bit):** Es un bit que puede generar paridad par o impar. La paridad se verifica en el dispositivo receptor para detectar errores de transmisión. Este bit es opcional.
4. **Bits de parada (stop bits):** Indica el número de bits que se agregan al final de cada byte de datos transmitido.

El **baudio** es una unidad de medida que indica la velocidad de transmisión en una **comunicación serie**. A menudo se utiliza como sinónimo de "**baud rate**" o "velocidad de transmisión". Un baudio **representa la cantidad de bits que se transmiten por segundo**. Existen velocidades standard de comunicación que parten desde los 150 baudios (150 bits por segundo) a 115200 baudios.

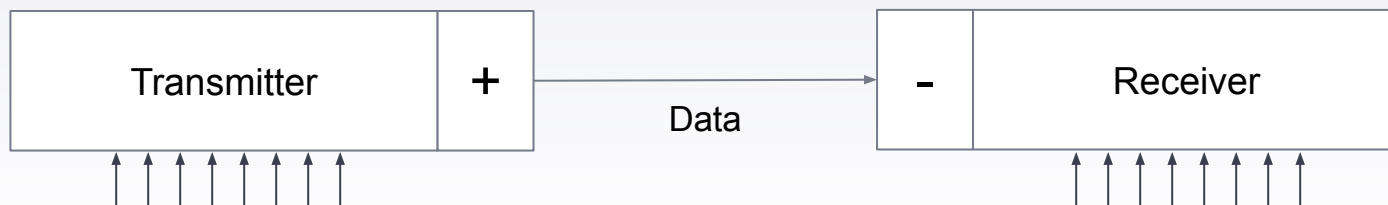
Un ejemplo de una velocidad muy utilizada es 9600 baudios. Esto implica que se transmiten 9600 bits en un segundo.

¿Cuál es la duración en tiempo de cada bit?

$T_{\text{bit}} (\text{tiempo de bit}) = 1 / \text{baud rate} \Rightarrow 1/9600 = 0,000104167 \text{ Segundos.}$

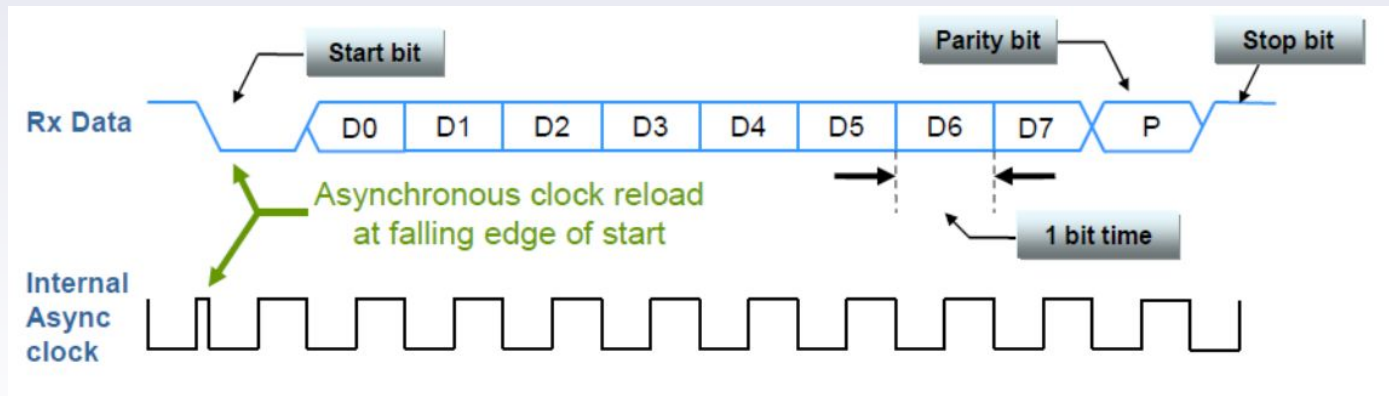
Si lo pasamos a microsegundos son 104,167 microsegundos por bit.

Comunicación serie asincrónica



En este gráfico vemos el diagrama simplificado de una comunicación serie asincrónica. A simple vista parecen ser dos shift registers, pero la diferencia radica en que **no hay señal de "clock" o "sincronismo"**. Debido a esta causa recibe el nombre de **comunicación "asincrónica"**.

Si no hay señal de sincronismo, ¿Cómo hace el receptor para saber cuando está recibiendo un dato?.. O ¿Cuándo sabe el receptor si está recibiendo todos los bits en "uno" o todos los bits en "cero"?



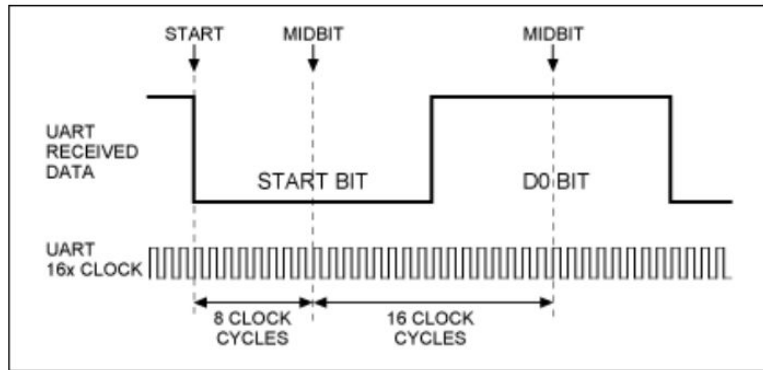
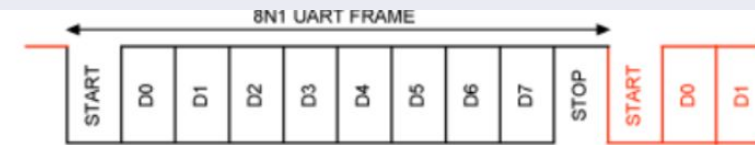
Como primera medida es importante aclarar que **el estado “idle” o el estado de “reposo”** (cuando no hay actividad en la línea de transmisión) **es un “uno”**.

Bit de start (bit de inicio): Es el primer bit de cada byte transmitido. Se establece en un nivel lógico “cero”, para indicar el inicio de un byte de datos. Este bit de inicio permite al receptor sincronizarse con la transmisión y prepararse para recibir los siguientes bits.

Bits de datos: a continuación del “Start bit” vienen los bits de datos, que pueden ser 7 u 8, que son los que contienen la información transmitida.

Luego puede venir el **“Parity Bit” (opcional)** y por último **el o los “bits de stop”**.

Pero ¿Cómo sabe el receptor cuando debe leer cada bit?

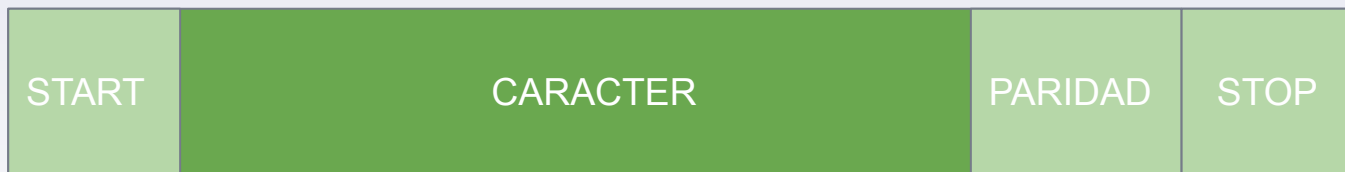


Como se dijo anteriormente el estado "idle" es un nivel alto en la línea de recepción. El receptor espera un cambio de nivel alto a bajo. En ese instante habilita la cuenta de un contador que tiene como clock la frecuencia del baud rate por 16 (16 veces mayor). Es decir que cada bit va a tener un ancho de 16 clock de esa señal.

Ese contador cuenta hasta 8, para asegurarse de estar en el "centro" del Start Bit.

Si detecta un "cero" implica que el start bit es válido y está recibiendo un dato.

A partir de ese momento ese contador cuenta hasta 16 (para asegurarse de estar en el centro del próximo bit) y el receptor lee la línea de datos. Ese estado que lee será el valor del primer bit de datos. Luego repite el proceso hasta completar la cantidad de bit programados en el formato de palabra.



El formato de palabra es el que se muestra en la figura. **Siempre existe un bit de start cuyo nivel es cero**, luego los bits de datos o el caracter transmitido, luego un bit de paridad (que es opcional) y **por último uno o dos bits de stop que siempre su nivel lógico es uno**.

El formato de palabra normalmente se describe con una sigla del tipo:

8N1 => 8 bits de datos, Ninguno de paridad y 1 bit de stop. Esto suma 9 bits, pero en realidad son 10 debido a que el STARTBIT está implícito y no lo debemos olvidar.

702 = 7 bits de datos paridad "impar" (ODD) 2 bits de stop. Total $1+7+1+2 = 11$ bits.

Recuerden esta regla nemotécnica ODD en inglés es impar. ODD tiene 3 letras (IMPAR). EVEN en inglés es PAR. EVEN tiene 4 letras (PAR).

8E1 ocho bits de datos paridad PAR, 1 bit de stop. Total $1+8+1+1 = 11$ bits.

Noten que siempre se agrego un bit en la suma al inicio que corresponde al bit de start.

Si sumamos todos esos bits y lo multiplicamos por el Tbit obtenemos la duración en tiempo de la transmisión de un caracter.

Ejemplo: Duración en tiempo de la transmisión de un caracter en formato 8N1 a 9600 baudios.

Tbit (tiempo de bit) = $1 / \text{baud rate} \Rightarrow 1/9600 = 0,000104167$ Segundos.

$8N1 = 1+8+1 = 10$ bits $10 * 0,000104167$ Segundos = $0,00104167$ Segundos. 1,04167 milisegundos por caracter transmitido.

Si un UART transmite a 9600 baudios (bits por segundo) y su formato de transmisión es 8N1, ¿cuántos caracteres por segundo se transmitirán?

Si el formato es 8N1 => 10 bits por caracter, como vimos anteriormente.

Entonces si la velocidad es de **9600 bits por segundo / 10bits** del formato del caracter => 960 caracteres por segundo.

Esta es la manera en la que determinamos la cantidad de "caracteres por segundo" que se transmiten.

Al determinar la cantidad de caracteres que se transmiten por segundo, tenemos la posibilidad de determinar qué porcentaje de esos bits son de "sincronismo o seguridad" y cuales son los de datos puros.

En el caso del ejemplo 8 de los 10 bits son de información pura y solo 2 de "sincronismo o seguridad". Por lo tanto el 80% del tiempo de transmisión se transmite información pura.

A la **cantidad de bits puros de información por segundo** la llamamos **“Bits de información por segundo”** y la denominamos **“bps”**.

Ejemplo I:

Indicar ¿cuál es la velocidad de transmisión (V_t) en bauds de un canal de comunicaciones que transmite en formato serie asincrónico, si la frecuencia del reloj de control del UART correspondiente es de 28800 Hz?

Suponiendo que el UART está configurado para transmitir caracteres de 7 bit + paridad y un bit de stop, indicar cual es la velocidad medida en bps y en caracteres por segundo.

Resolución:

Frecuencia del receptor serie $f_{ck} = 28800$ Hz.

$V_t = f_{ck} / 16 = 1800$ baud.

1 bit start + 7 bit datos + 1 bit paridad + 1 bit stop = 10 bit/caracter

$1800 \text{ baud} / 10 \text{ bit/caracter} = 180 \text{ caracteres/segundo}$

$180 \text{ caracteres/segundo} * 7 \text{ bit inf} = 1260 \text{ bits de información /seg (bps)}$

Nótese la diferencia entre bauds y bps: **el cociente bps/bauds es la eficiencia del canal.** En este caso: $1260/1800 = 0,7$

Ejemplo II

Indicar en cuánto tiempo se puede transmitir un documento de texto simple de 3 MB, utilizando un canal serie asincrónico de 33600 bauds, si cada caracter se transmite en ASCII de 7 bits + paridad y dos bits de stop

Tamaño documento = 3 MByte

$V_t = 33600$ baud

1 bit start + 7 bit datos + 1 bit paridad + 2 bit stop = 11 bit/caracter

El documento es de 3 Mbyte = $3 * 1024 * 1024$ bytes (palabras de 8 bits).

** El UART está configurado para transmitir palabras ASCII de 7 bits. El problema es que no se puede "desarmar" cada palabra de 8 bits del documento: el UART carga las palabras en paralelo -aunque transmite en serie-. Por lo tanto cada carácter ASCII se transmitirá en 7 bits, con la consiguiente pérdida de información.*

$33600 \text{ baud} / 11 \text{ bit/carácter} = 3054 \text{ caracteres/seg}$ (nunca con decimales)

$3 \text{ MB} / 3054 \text{ caracteres/seg} = (3 * 1024 * 1024) \text{ bytes} / 3054 \text{ car/seg} = 1031 \text{ s} \sim 18 \text{ min}$

¿Cuánto se pierde? Observe una tabla ASCII y vea qué caracteres no se recibirán correctamente.



Serial Peripheral Interface

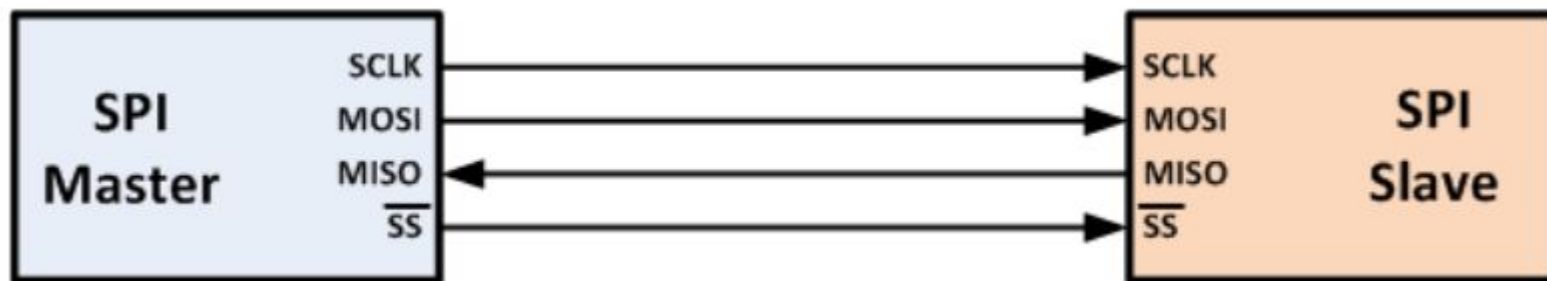
SPI

SPI (Serial Peripheral Interface) es un **protocolo de comunicación serial** utilizado para la transferencia de datos entre un maestro (**master**) y uno o varios dispositivos esclavos (**slaves**).

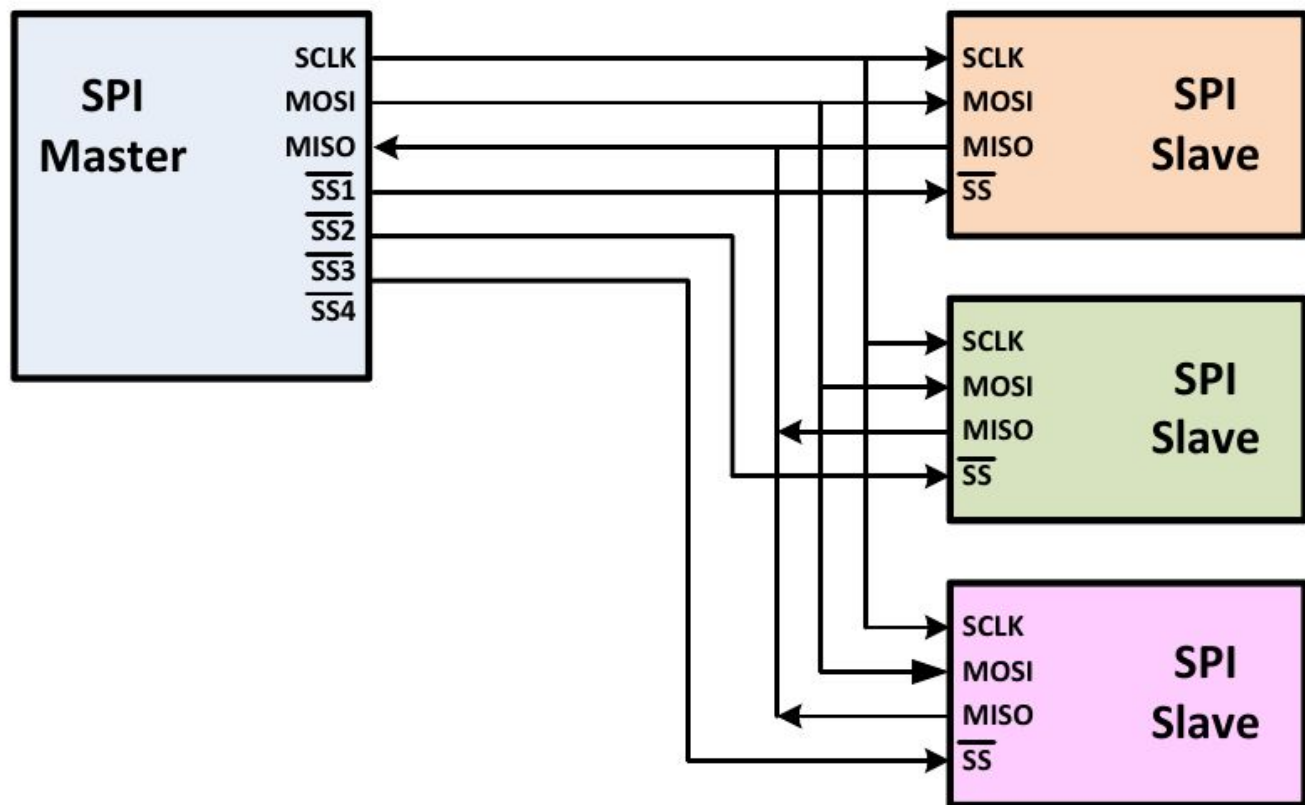
El bus SPI consta de cuatro líneas principales:

1. **SCLK (Serial Clock)**: Es la línea de reloj que sincroniza la transferencia de datos entre el master y los dispositivos slaves. En todos los casos el master es el que genera la señal de reloj.
2. **MOSI (Master Output, Slave Input)**: Es la línea por donde el master envía los datos hacia los slaves.
3. **MISO (Master Input, Slave Output)**: Es la línea por donde los dispositivos slaves envían los datos al master.
4. **SS (Slave Select)**: Es una línea individual por cada dispositivo slave en el bus para establecer la comunicación con dicho dispositivo en particular, mientras los demás permanecen inactivos.

Interconexión de dos dispositivos.



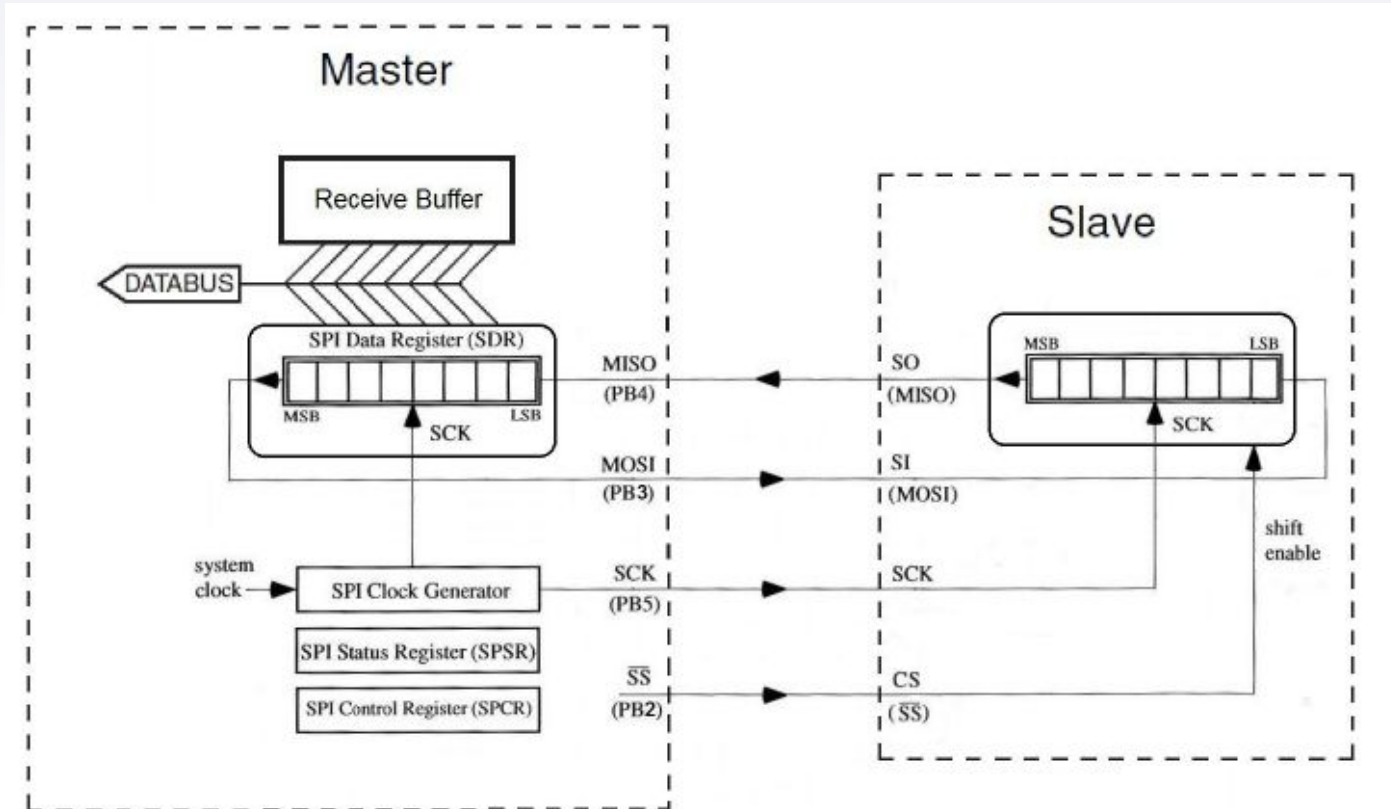
Interconexión de varios dispositivos.



El flujo de la comunicación en un bus SPI **es controlado por el master**. El proceso típico de transferencia de datos es el siguiente:

1. El master activa la línea SS correspondiente al dispositivo slave con el que desea comunicarse, seleccionándolo.
2. El master genera pulsos en la línea SCLK para sincronizar la transferencia de datos.
3. En cada pulso de SCLK, el master coloca un bit de datos en la línea MOSI y el dispositivo slave lo recibe en su línea MOSI.
4. Al mismo tiempo, el dispositivo slave coloca un bit de datos en su línea MISO y el master lo recibe en su línea MISO.
5. La transferencia continúa en cada pulso de SCLK hasta que se haya transmitido la cantidad de bits necesaria.
6. El master desactiva la línea SS del dispositivo slave, finalizando la comunicación.

Circuito interno simplificado en un microcontrolador





Inter Integrated Circuit

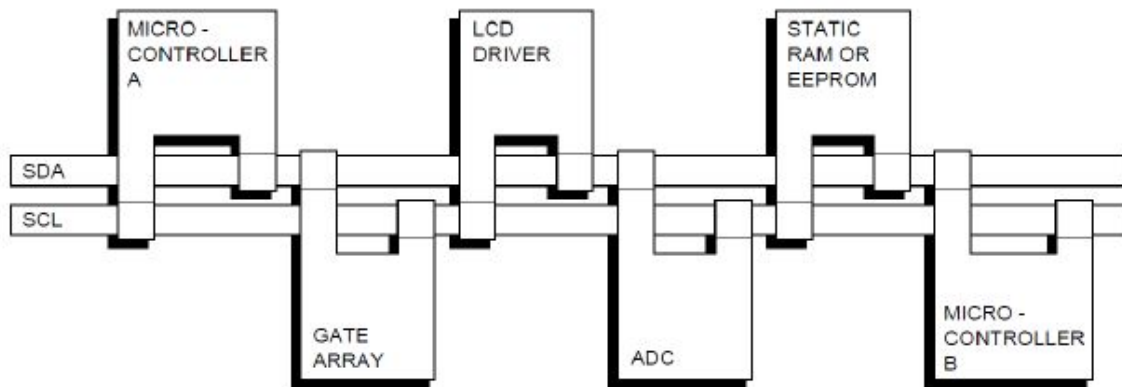
I2C

El bus **I2C (Inter-Integrated Circuit)** es un protocolo de comunicación serie que permite la **transferencia de datos entre dispositivos electrónicos a través de solo dos cables: una línea de datos (SDA) y una línea de reloj (SCL).**

Fue un desarrollo de la empresa Philips en los años 80's para comunicar diferentes chips dentro de televisores y radios que ya comenzaban a incluir microcontroladores para su control.

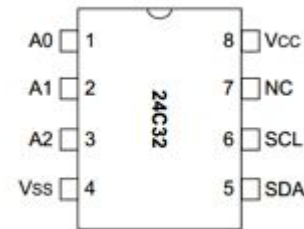
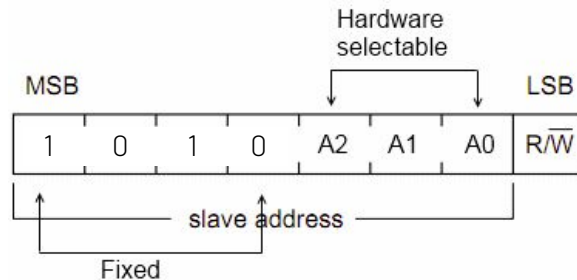
Se transformó en un standard de la industria en los 90's.


Como se trata de un bus, permite interconectar muchos dispositivos solo con dos cables.



El funcionamiento del bus I2C se basa en un enfoque master/slave. Un dispositivo master, como un microcontrolador, inicia y coordina las transferencias de datos en el bus, mientras que los dispositivos slaves, como sensores, memorias o controladores, responden a las solicitudes del maestro.

El protocolo I2C **utiliza un esquema de direccionamiento para identificar a cada dispositivo slave en el bus.** Cada dispositivo tiene una dirección única asignada por el fabricante de cada chip, que puede ser de 7 bits o de 10 bits. Varios chips del mismo tipo pueden convivir en el bus (con el mismo "fixed address") ya que el fabricante deja accesibles por fuera del chip unos bits de la palabra de dirección llamado normalmente "hardware selectable" address.



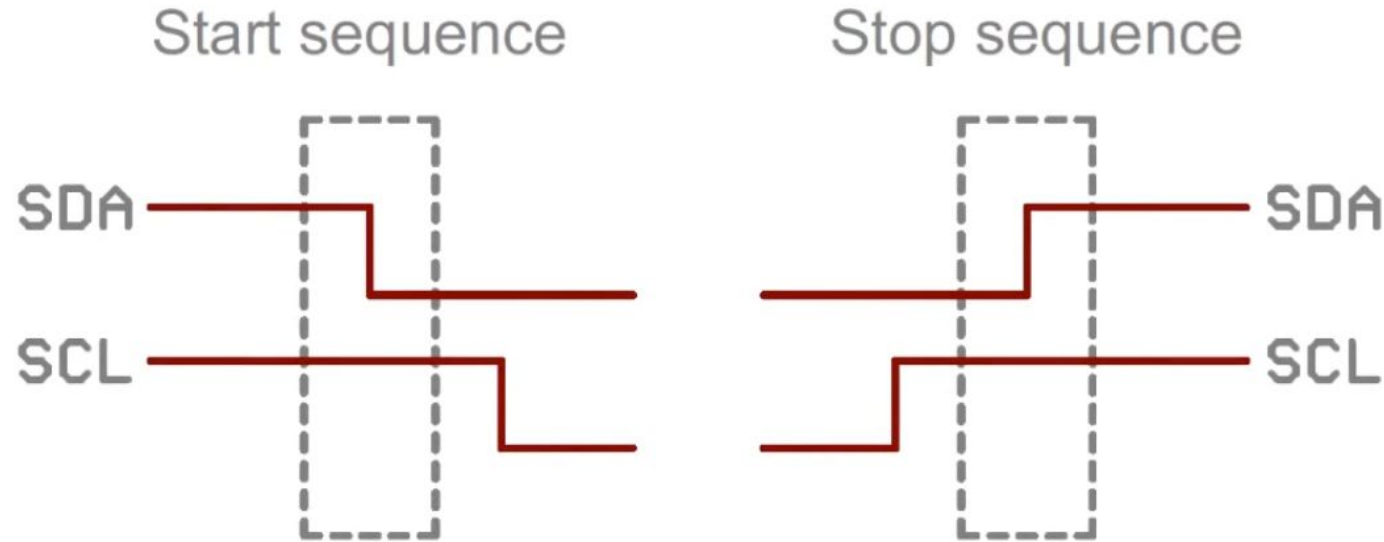


Para iniciar una comunicación, el master envía una señal de inicio al bus I2C, indicando que desea comunicarse con un dispositivo slave específico. A continuación, transmite la dirección del dispositivo slave al que desea acceder. Si el dispositivo slave coincide con la dirección enviada, responde con un ACK (acknowledge) al maestro.

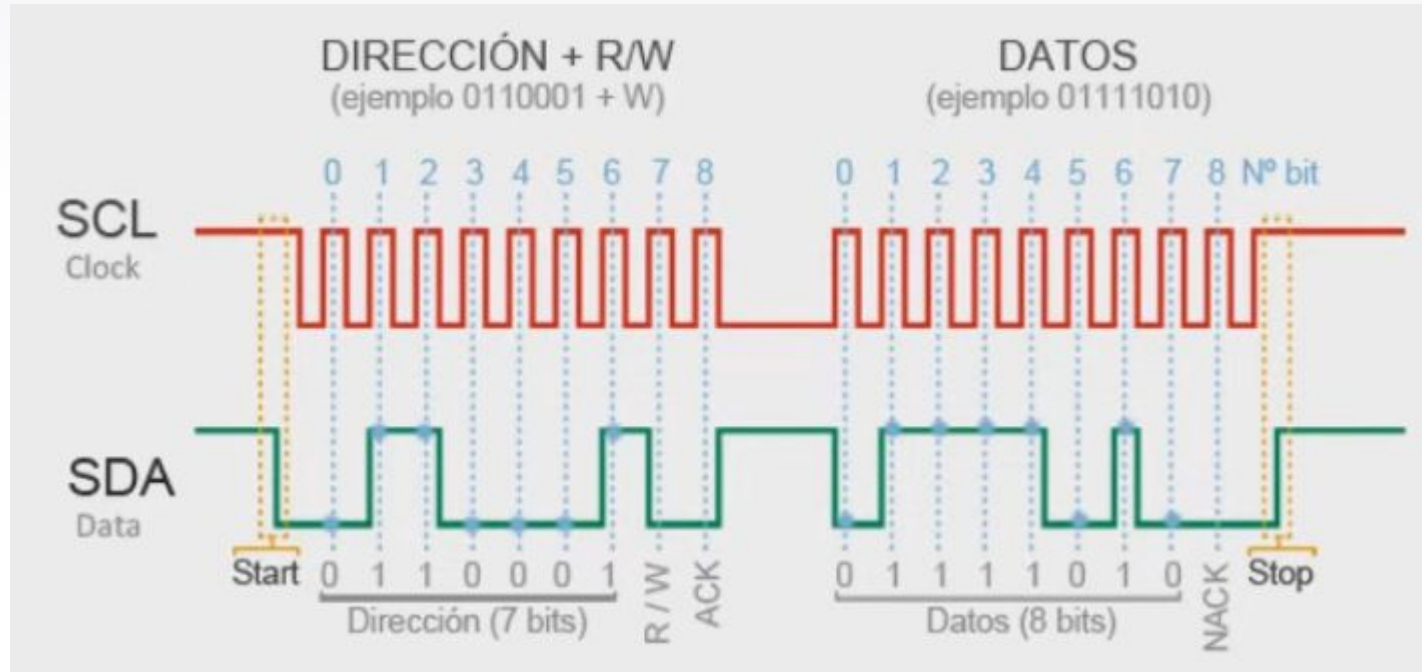
Una vez establecida la comunicación, el master y el slave pueden intercambiar datos. Los datos se transfieren en paquetes de 8 bits, donde cada paquete se confirma con un ACK o un NACK (no acknowledge) después de su recepción. El master puede enviar o recibir múltiples bytes de datos en una secuencia continua.

Al finalizar la comunicación, el master envía una señal de parada al bus I2C, indicando que ha finalizado la transferencia. Esta señal de parada es reconocida por todos los dispositivos en el bus y libera las líneas de datos y reloj para su uso por otros dispositivos.

Al existir solo dos cables, las señales de START y STOP son generadas por estados excepcionales en los pines de SDA y SCL. Es decir se mueve la línea de dato con la línea de clock en alto.



Los datos se transmiten en secuencias de ocho bits y siempre se mueve la línea SDA con la línea SCL en "bajo" (lo opuesto solo ocurre en los estados de start y stop).



Por cada byte transmitido el chip que oficia de slave envía al master un bit de reconocimiento llamado "ACKNOWLEDGE" (ACK), cuyo estado es cero para indicar que recibió de manera correcta el dato.

En la figura se muestra la secuencia de escritura de una posición de memoria en un dispositivo 24C32 (memoria eeprom).

