Sistemas Electrónicos Digitales

Tema #3

8. Comunicaciones Serie

Temario

1. Introducción
2. GPIO: General Purpose Input/Output
3. Arquitectura ARM Cortex-M4
4. Interrupciones
5. C en ensamblador
6. Temporizadores (Timers)
7. Direct Memory Access
8. Comunicaciones Serie
9. Conversores A/D y D/A

ÍNDICE

- **Comunicaciones serie**
  - Conceptos
  - Herramientas
  - Software: *polling*, interrupciones y *buffering*

- **Protocolos**:
  - UART
  - SPI
  - I2C

Comunicaciones Serie

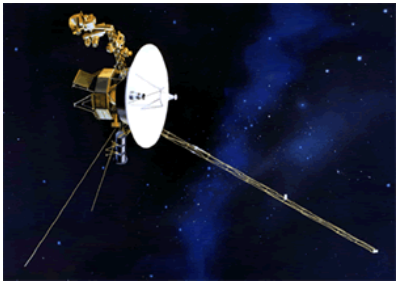
INTRODUCCIÓN

¿Por qué comunicarse de forma serie?

- El tamaño de la palabra nativa es de múltiples bits (8, 16, 32, etc.)

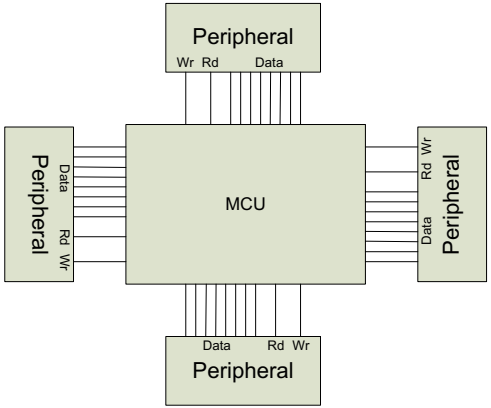
- A menudo no es factible admitir el envío de todos los bits de la palabra al mismo tiempo (de forma **paralela**)
  - Coste y peso: se necesitan **más cables**, se necesitan conectores más grandes
  - Fiabilidad mecánica: más cables => **más contactos** del conector que pueden fallar
  - **Complejidad** de **tiempo**: algunos bits pueden llegar más tarde que otros debido a las variaciones en la capacidad y la resistencia de los conductores
  - **Complejidad** y **potencia** del circuito: es posible que no desee tener 16 transmisores y receptores de radio diferentes en el sistema

Ejemplo de sistema: Voyager Spacecraft



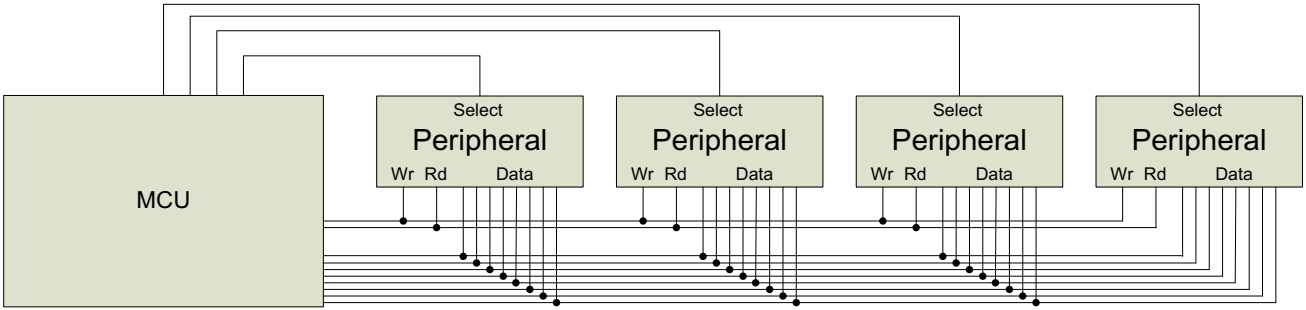
- Lanzado en 1977
- Restricciones: fiabilidad, potencia, tamaño, peso, fiabilidad, fiabilidad (☺) etc.
- "Las comunicaciones de enlace ascendente se realizan a través de la banda S (velocidad de comando de 16 bits/s) mientras que un transmisor de banda X proporciona telemetría de enlace descendente a 160 bits/s normalmente y 1,4 kbps para la reproducción de datos de ondas de plasma de alta velocidad. Todos los datos se transmiten y se reciben en la nave a través de la antena de alta ganancia (HGA) de 3,7 metros " (http://voyager.jpl.nasa.gov/spacecraft/index.html)
  - Enlace ascendente - a la nave espacial
  - Enlace descendente - de la nave espacial

Ejemplo de sistema



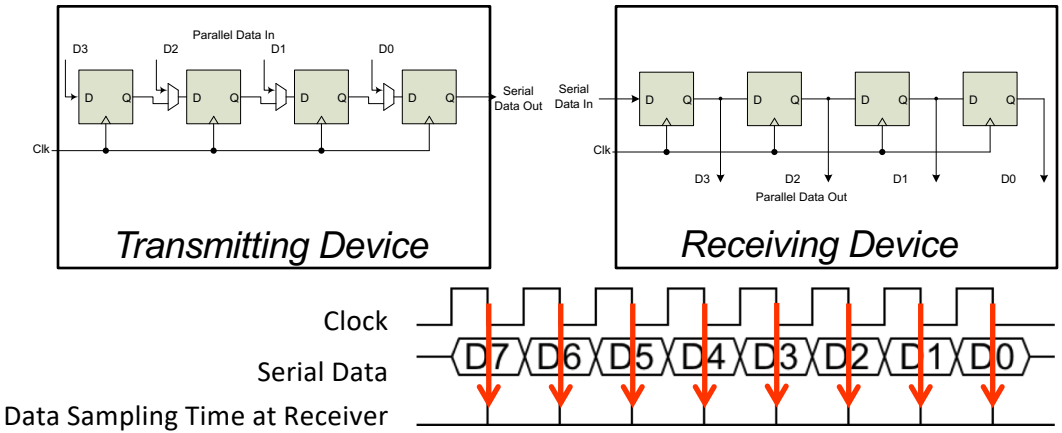
- Conexiones punto a punto dedicadas
  - Líneas de datos paralelas, líneas de lectura y escritura entre MCU y cada periférico
- Rápido, permite transferencias simultáneas.
- Requiere muchas conexiones, área de PCB, difícil escalado

Buses paralelos



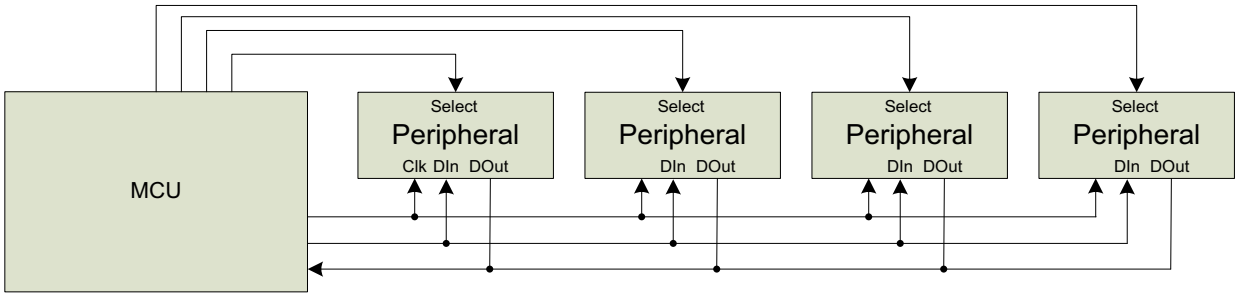
- Todos los dispositivos utilizan buses para compartir datos, leer y escribir señales.
- MCU utiliza líneas de selección individuales para abordar cada periférico
- MCU requiere menos pines para datos, pero aún uno por bit de datos
- MCU puede comunicarse con un solo periférico a la vez

Transmisión de datos serie síncronos



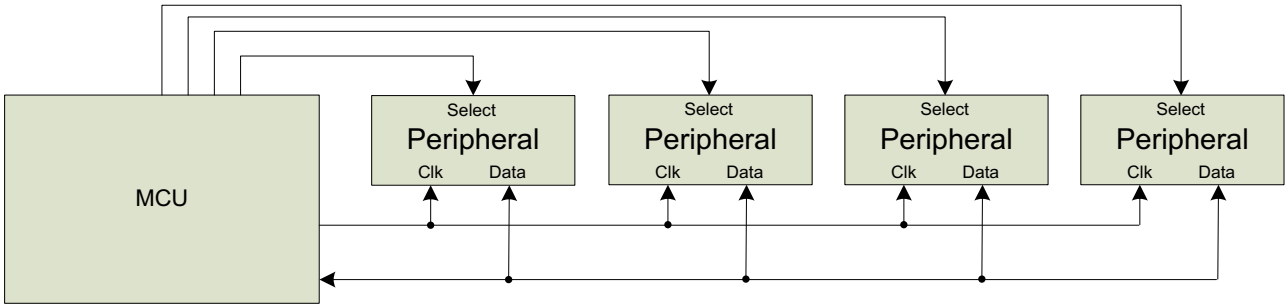
- Utiliza los registros de desplazamiento y una señal de reloj para convertir entre los formatos serie y paralelo
- Síncrono: una señal de reloj explícita va junto con la señal de datos

Comunicaciones Serie **Síncronas** Full-Duplex



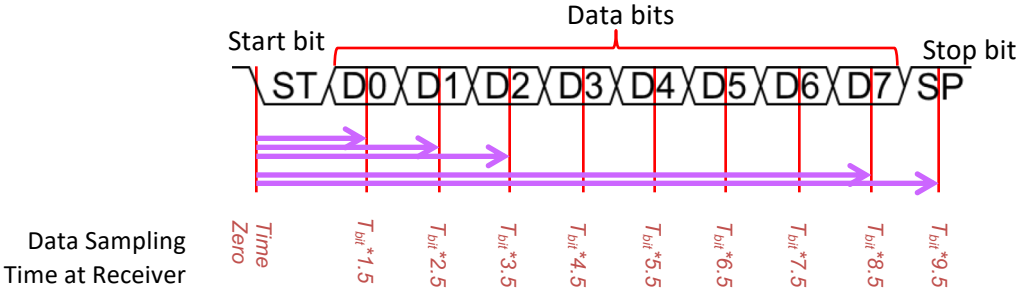
- Usa dos líneas de datos serie: una para **leer** y otra para **escribir**.
- Permite enviar y recibir simultáneamente la comunicación: **full-duplex.**

Comunicaciones Serie **Síncronas** Half-Duplex



- Se comparte la línea de datos serie entre la lectura y la escritura
- No permite el envío y la recepción simultáneos, es una comunicación **half-duplex** (**semidúplex**)

Comunicaciones Serie **Asíncronas**



- ¡**Elimina la línea del reloj**!
- El transmisor y el receptor deben **generar reloj localmente**
- El transmisor debe agregar un **bit de inicio** (siempre el mismo valor) para indicar el inicio de cada *data frame* (trama de datos)
- El receptor detecta el **flanco inicial del bit de inicio**, luego lo usa como una referencia de tiempo para **muestrear** la línea de datos para extraer cada bit de datos N en el momento Tbit * (N + 1.5)
- El **bit de parada** también se usa para detectar algunos errores de tiempo

Comunicaciones serie en detalle

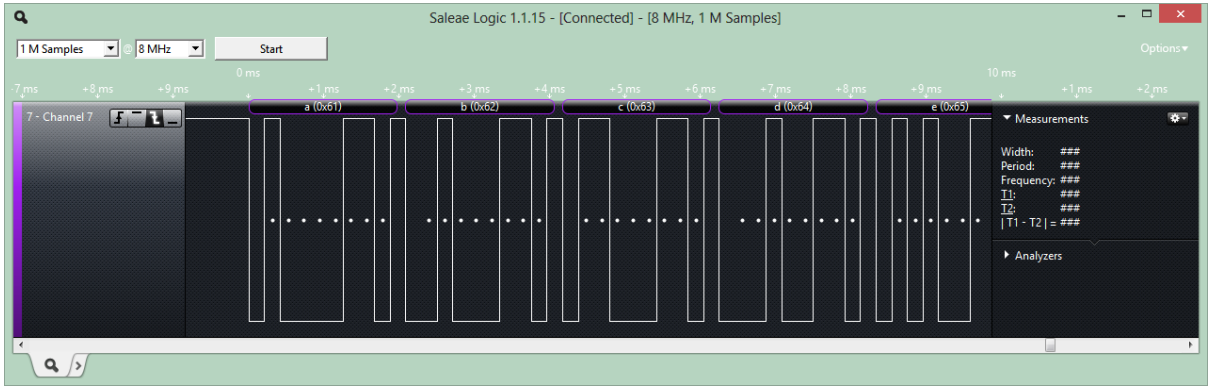
- **Campos de la trama**
  - **Bit de inicio** (un bit)
  - **Datos** (LSB o MSB primero, y tamaño - 7, 8, 9 bits)
  - El bit de **paridad** opcional se usa para hacer que la cantidad total de unidades en los datos sea par o impar
  - **Bit de parada** (uno o dos bits)

- Todos los dispositivos deben utilizar los mismos parámetros de comunicación.
  - Ej. **Velocidad de comunicación** (300 baudios, 600, 1200, 2400, 9600, 14400, 19200, etc.)
- Protocolos de red sofisticados tienen más información en cada trama de datos.
  - **Control de acceso medio**: cuando hay varios nodos en el bus, deben arbitrar para obtener permiso para transmitir
  - Información de **direccionamiento**: ¿para qué nodo está destinado este mensaje?
  - Mayor **carga útil** de datos
  - **Detección de errores** o información de corrección de errores
  - Solicitud de **respuesta** inmediata (”ACK")

Detección de errores

- Envío de **información adicional** para verificar que los datos fueron recibidos correctamente.
- Necesidad de especificar **qué paridad esperar: par, impar o ninguna**.
- El **bit de paridad** se establece de modo que **el número total de bits "1"** en los datos y la paridad **sea par** (para la paridad par) **o impar** (para la paridad impar)
  - 01110111 tiene 6 bits "1", por lo que el bit de paridad será 1 para paridad impar, 0 para paridad par
  - 01100111 tiene 5 bits "1", por lo que el bit de paridad será 0 para paridad impar, 1 para paridad par
- El bit de paridad única detecta si 1, 3, 5, 7 o 9 bits están dañados, pero no detecta un número par de bits dañados
- Existen **códigos de detección de errores más fuertes** (por ejemplo, control de redundancia cíclica) que utilizan múltiples bits (por ejemplo, 8, 16) y pueden detectar muchas más corrupciones.
  - Utilizado para CAN, USB, Ethernet, Bluetooth, etc.

Herramientas para el desarrollo de comunicaciones serie



- Tedioso y lento para depurar protocolos serie con solo un osciloscopio
- En su lugar, use un analizador lógico para decodificar el tráfico del bus
- ¡Vale su peso en oro!

- Analizador lógico de 8 canales Saelae
  - $150 (www.saelae.com)
  - Se conecta al puerto USB de la PC
  - Decodifica SPI, serie asíncrona, I2C, 1-Wire, CAN, etc.
- Cree uno propio: con Logic Sniffer o proyectos de código abierto relacionado

Comunicaciones Serie

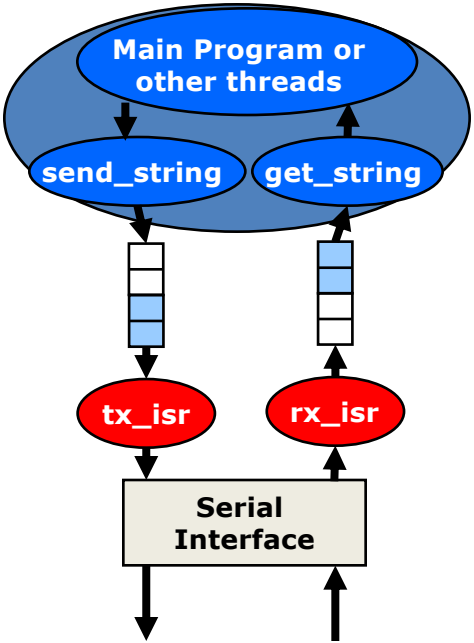
ESTRUCTURA SOFTWARE: COMUNICACIÓN ASÍNCRONA

Estructura Software

- La comunicación es asíncrona al programa.
  - No sé qué código ejecutará el programa ...
    - cuando llegue el siguiente dato
    - cuando se complete la transmisión del dato saliente actual
    - cuando se produzca un error
  - Necesidad de sincronización entre el programa y la interfaz de comunicación serie
- Opciones:
  - **Polling**
    - Espere hasta que los datos estén disponibles
    - Simple pero ineficiente (tiempo de procesador)
  - **Interrupciones**
    - La CPU interrumpe el programa cuando los datos están disponibles
    - Eficiente, pero más complejo.

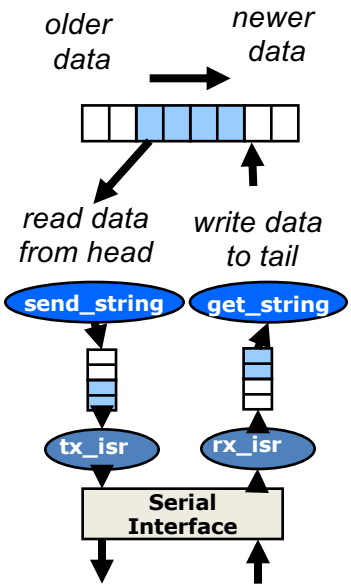
Comunicaciones serie e interrupciones

- Queremos proporcionar múltiples hilos de control en el programa.
  - **Programa principal** (y subrutinas que llama)
  - **ISR Transmisión**: se ejecuta cuando la interfaz serie está lista para enviar otro dato
  - **ISR Recepción**: se ejecuta cuando la interfaz serie recibe un dato
  - **ISRs Error**: se ejecuta si se produce un error
- Necesita una forma de almacenar información entre los hilos
  - Solución: **cola circular** con punteros de cabeza y cola.
  - Uno para TX, otro para RX



Código para implementar colas

- **Meter en cola**: *tail* es el índice de la siguiente entrada libre
- **Sacar desde la cabeza**: *head* es el índice del elemento a eliminar.
- El **tamaño** de la cola **se inicializa** y se almacena en tamaño
- **Una cola por dirección**
  - **TX** ISR descarga tx_q
  - **RX** ISR carga rx_q
- Otros subprocesos (por ejemplo, principal) cargan tx_q y descargan rx_q
- Es necesario "envolver" (wrap) el puntero al final del búfer para hacerlo **circular**,
  - Use el operador % (módulo, resto) si el tamaño de la cola no es potencia de dos
  - Use & (bitwise y) si el tamaño de la cola es una potencia de dos
- La cola está vacía si tail == head
- La cola está llena si (tail + 1) % tamaño == head



Definiendo colas

```
typedef struct {
    uint8_t *data; //!< Array of data, stored on the heap.
    uint32_t head; //!< Index in the array of the oldest element.
    uint32_t tail; //!< Index in the array of the youngest element.
    uint32_t size; //!< Size of the data array.
} Queue;
```

Inicialización y peticiones de estado

```
int queue_init(Queue *queue, uint32_t size) {
    queue->data = (uint8_t*)malloc(sizeof(uint8_t) * size);
    queue->head = 0;
    queue->tail = 0;
    queue->size = size;

    // If malloc returns NULL (0) the allocation has failed.
    return queue->data != 0;
}
int queue_is_full(Queue *queue) {
    return ((queue->tail + 1) % queue->size) == queue->head;
}
int queue_is_empty(Queue *queue) {
    return queue->tail == queue->head;
}
```

Meter y sacar de la cola

```
int queue_enqueue(Queue *queue, uint8_t item) {
    if (!queue_is_full(queue)) {
        queue->data[queue->tail++] = item;
        queue->tail %= queue->size;
        return 1;
    } else {
        return 0;
    }
}

int queue_dequeue(Queue *queue, uint8_t *item) {
    if (!queue_is_empty(queue)) {
        *item = queue->data[queue->head++];
        queue->head %= queue->size;
        return 1;
    } else {
        return 0;
    }
}
```

Usando las colas

- Sending data:

```
queue_enqueue(…, c)
```

- Receiving data:

```
queue_dequeue(…, &c)
```

Comunicaciones Serie

ESTRUCTURA SOFTWARE: TRABAJANDO CON MENSAJES

Decodificando mensajes: tipos de mensajes

- **Datos binarios** reales enviados
  - Primero: identificar el tipo de mensaje
  - Segundo: en función de este tipo de mensaje, copie los datos binarios de los campos del mensaje en variables
    - Es posible que deba usar punteros y conversión para obtener el código para traducir los formatos de manera correcta y segura
- **Caracteres de texto ASCII** que representan datos enviados
  - Primero: identificar tipo de mensaje
  - Segundo: según el tipo de mensaje, convierta (analice) los datos del formato de mensaje ASCII a un formato binario
  - En tercer lugar, copiar los datos binarios en variables

Ejemplo: datos serie binarios (TSIP)

Report Packet (0x84) Data Structure

Type	sizeof(Type)	Item	Units
Double	8	Latitude	Radians; + for north, - for south
Double	8	Longitude	Radians; + for east, - for west
Double	8	Altitude	Meters
Double	8	Clock Bias	Meters
Single	4	Time-of-fix	Seconds

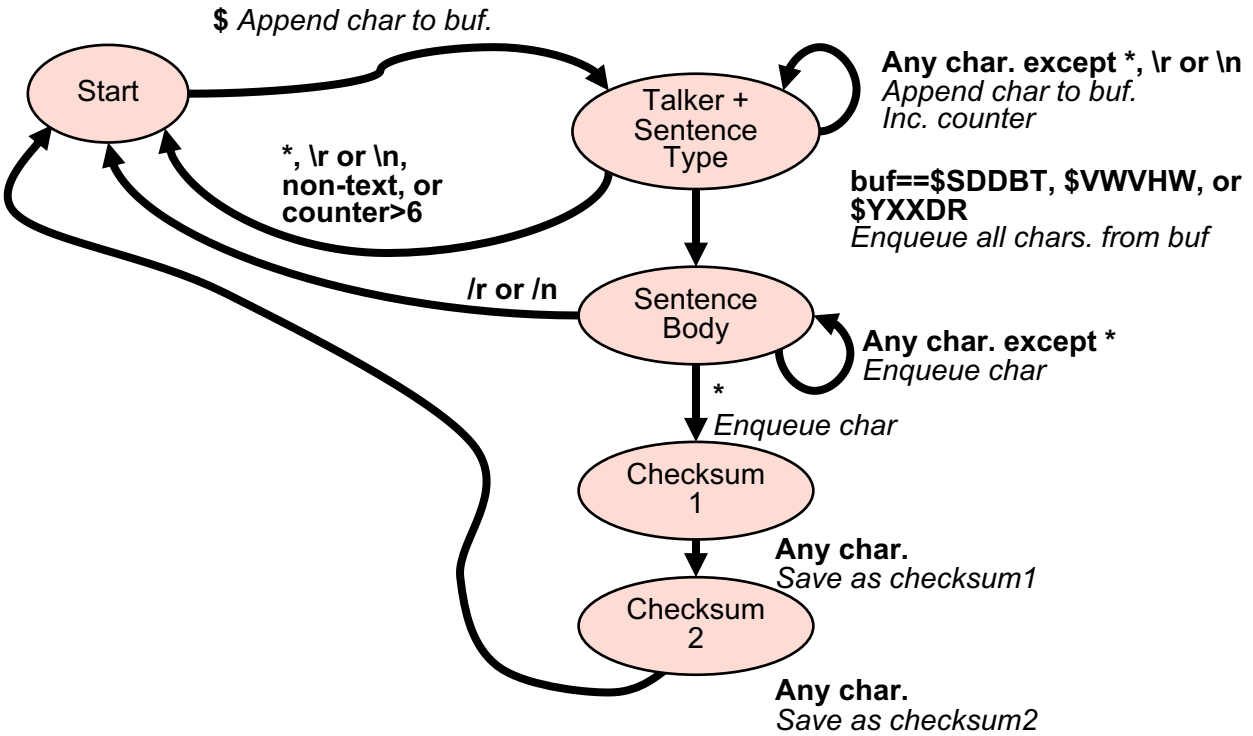
```
switch (id) {
case 0x84:
     lat = *((double *)(&msg[0]));
     lon = *((double *)(&msg[8]));
     alt = *((double *)(&msg[16]));
     clb = *((double *)(&msg[24]));
     tof = *((float  *)(&msg[32]));
     break;
case 0x4A: …

default:
     break;
}
```

Ejemplo: ASCII Serial Data (NMEA-0183)

$IDMSG,D1,D2,D3,D4,…,Dn*CS\r\n

- **$** denota el inicio de un mensaje
- **ID** es un mnemotécnico de dos letras para describir la fuente de datos, por ejemplo, GP significa GPS
- **MSG** es un mnemotécnico de tres letras para describir el contenido del mensaje.
- Las comas se utilizan para separar los campos de datos.
- **Dn** representa cada uno de los campos de datos.
- ***** se utiliza para separar los datos del *checksum* (suma de comprobación).
- **CS** contiene dos caracteres ASCII que representan el valor hexadecimal del checksum.
- **\r \n** es el carácter de retorno de carro seguido del nuevo carácter de línea para indicar el final de un mensaje.

Máquina de estado para analizar NMEA-0183



Parsing

```
switch (parser_state) {
   case TALKER_SENTENCE_TYPE:
       switch (msg[i]) {
           ‘*’:
           ‘\r’:
           ‘\n’:
               parser_state = START;
               break;
           default:
               if (Is_Not_Character(msg[i]) || n>6) {
                   parser_state = START;
               } else {
                   buf[n++] = msg[i];
               }
           break;
       }
       if ((n==6) & … ){
           parser_state = SENTENCE_BODY;
       }
       break;
   case SENTENCE_BODY:
       break;
```

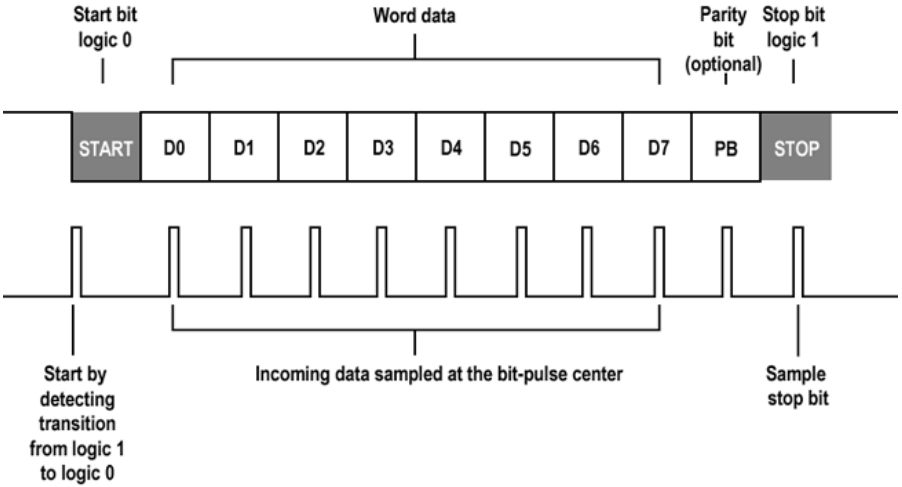
Comunicaciones Serie

COMUNICACIONES SERIE ASÍNCRONAS: UART

UART: concepto

- **Universal Asynchronous Receiver Transmitter**
- Proporciona soporte de hardware para un puerto serie en los procesadores. La señalización es compatible con PC / Mac / Unix serie (RS-232C)
- UART proporciona:
  - Conversión de paralelo a serie y de serie a paralelo
  - Tramas con bit de inicio y parada
  - Generación de paridad
  - Generación de velocidad de transmisión (2400-115.2kbps a 3.686 o 7.37MHz)
  - Interrupciones
    - Transmisión completa
    - Registro de datos de transmisión vacío
    - Recepción completa

UART: datos



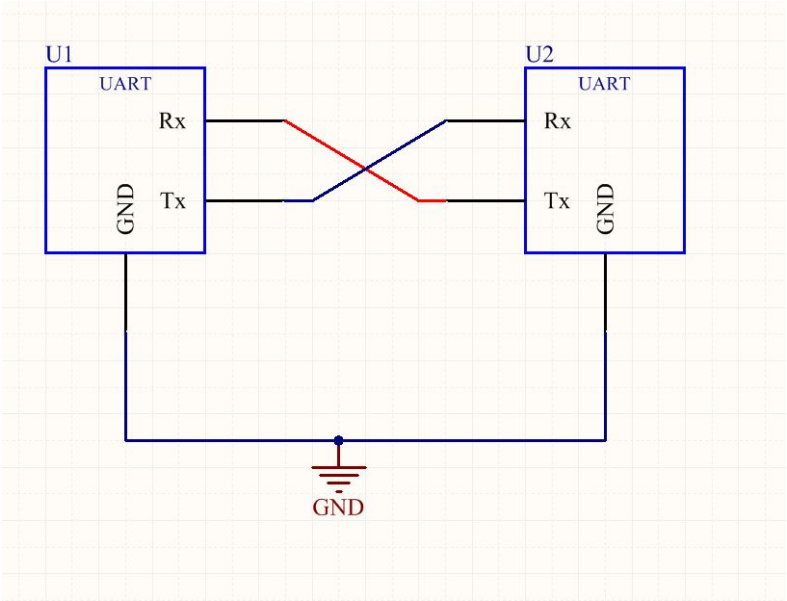
Data

– Start bit
– 6,7,8,9 data bits
– Parity bit opcional (E,O,M,S,N)
– Stop bit

Voltages

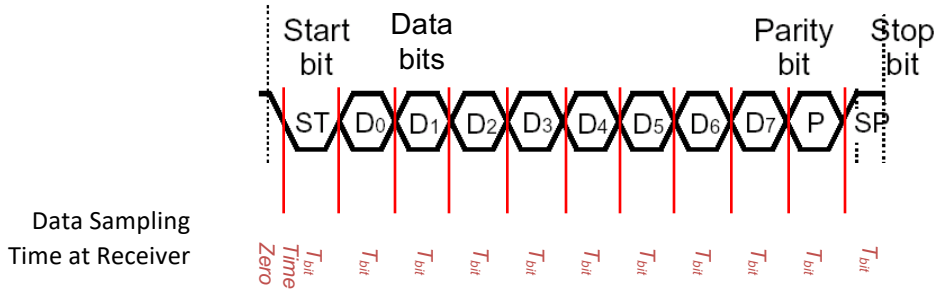
- El procesador proporciona una señal de nivel lógico de 0 / 5V
- RS-232C usa una señal de + 12V / -12V
- IC convertidor de nivel proporcionado en STK500 (MAX202)

UART: conexionado



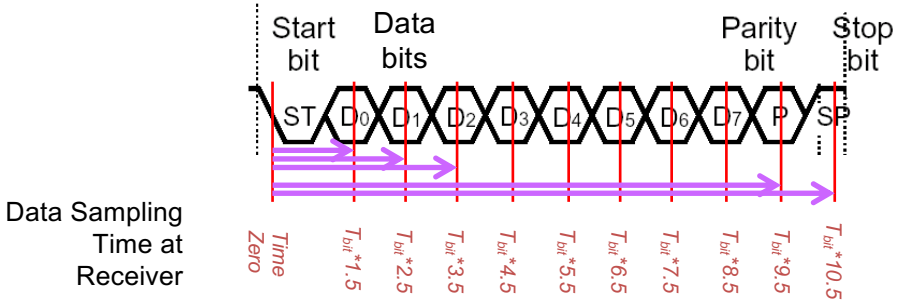
¿¿¿Reloj???

UART: transmisión de datos básica



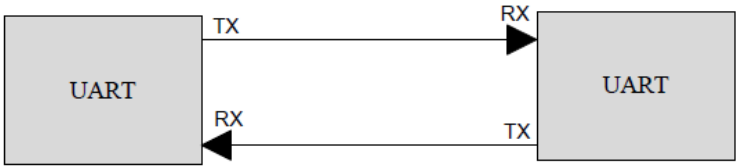
- Si no hay datos para enviar, siga enviando 1 (bit de parada) - línea inactiva
- Cuando hay una palabra de datos para enviar:
  - Enviar un 0 (bit de inicio) para indicar el inicio de una palabra
  - Envíe cada bit de datos en la palabra (use un registro de desplazamiento para el búfer de transmisión)
  - Enviar un 1 (bit de parada) para indicar el final de la palabra

UART: recepción de datos básica



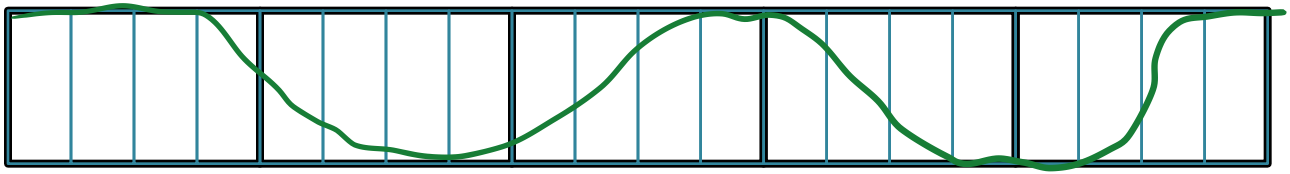
- Esperar a un flanco de bajada (comienzo de un bit de inicio)
- Esperar ½ tiempo de bit (bit time)
- Hacer lo siguiente para cada bit de datos en la palabra
  - Esperar 1 bit time
  - Leer el bit de datos y desplazarlo (shift) a un búfer de recepción (registro de desplazamiento)
- Esperar 1 bit time
- Leer el bit
  - Si es 1 (bit de parada), entonces OK
  - Si es 0, ¡hay un problema!

UART: Para que esto funcione…



- El transmisor y el receptor deben acordar varias cosas (protocolo):
  - Orden de los bits de datos
  - Número de bits de datos
  - Qué es un bit de inicio (1 o 0)
  - Qué es un bit de parada (1 o 0)
  - Cuanto dura un bit:
    - los relojes del transmisor y del receptor deben estar razonablemente cerca, ya que la única referencia de tiempo es el inicio del bit de inicio

Sobremuestreo de datos de entrada



- Al recibir, UART sobremuestrea la línea de datos entrante
  - Las muestras extra permiten hacer medias, mejorando la inmunidad al ruido.
  - Mejor sincronización con los datos entrantes, mejorando la inmunidad al ruido.
- Sobremuestreo configurable de 8x a 32x
  - Coloque el factor de sobremuestreo deseado menos uno en los bits SCB_CTRL, SCB_OVS.

Baud Rate

- Debe dividir el reloj de alta frecuencia a la velocidad de transmisión deseada * factor de sobremuestreo

- Ejemplo
  - 24 MHz -> 4800 baudios con 16x sobremuestreo
  - Factor de división = 24E6 / (4800 * 16) = 312.5. Debe redondear al valor entero más cercano (312 o 313), tendrá un ligero error de frecuencia.

Software para Comunicaciones serie: *polling*

```
void test_polled() {
    uart_init(9600);
    uart_enable();

    while(1) {
        uart_tx(uart_rx()); // echos the received character back
    }
}
```

Software para Comunicaciones serie: interrupciones

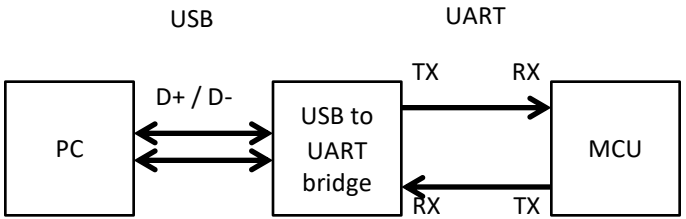
- Usar interrupciones
- **Primero:** inicialice el periférico para generar interrupciones.
- **Segundo:** cree una sola ISR para recibir la *callback*
- **Tercero:** habilitar el periférico

Interrupt Handler

```
Queue rx_queue;
void uart_rx_isr(uint8_t rx) {
        // Store the received character
        queue_enqueue(&rx_queue, rx);
}
int main() {
        queue_init(&rx_queue, 128);
        uart_init(9600);
        uart_set_rx_callback(uart_rx_isr);
        uart_enable();
        …
}
```

USB to UART Interface

Los PCs no tienen interfaces serie asíncronas externas desde hace tiempo, entonces, ¿cómo nos comunicamos con un UART?



- Interfaz USB a UART
  - Conexión USB a PC
  - Nivel lógico (0-3.3V) para UART del microcontrolador (no niveles de voltaje RS232)
- USB01A Adaptador USB a serie
  - http://www.pololu.com/catalog/product/391
  - También puede suministrar 5 V, 3.3 V desde USB

Inconvenientes de comunicaciones asíncronas

- **Inconveniente #1**
  - Las señales de nivel lógico (0 a 1.65 V, 1.65 V a 3.3 V) son sensibles al ruido y la degradación de la señal

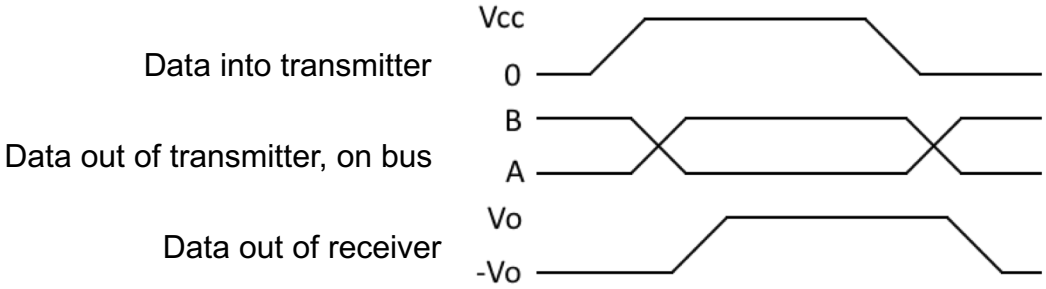
- **Inconveniente #2**
  - La topología punto a punto no admite bien un gran número de nodos
    - Necesita un cable dedicado para enviar información de un dispositivo a otro
    - Necesita un canal UART para cada dispositivo con el que la MCU necesita hablar
    - Transmisor único, receptor único por cable de datos

Solución al ruido: voltajes más altos

- Use voltajes más altos para mejorar el margen de ruido:? +3 a +15 V, -3 a -15 V

- Ejemplo IC (Maxim MAX3232) usa bombas de carga (charge pumps) para generar voltajes más altos desde el riel de suministro de 3.3V

Solución al ruido: señales diferenciales



- Utilizar señales diferenciales:
  - Enviar dos señales: datos del buffer (A), y su complemento (B)
  - El receptor compara las dos señales para determinar si los datos son uno (A > B) o cero (B > A)

Soluciones para la escalabilidad

- **Enfoques**
  - **Multi-drop:** Permitir que **un transmisor** maneje **múltiples receptores**
  - **Multipunto**: Conectar todos los transmisores y todos los receptores a la misma línea de datos.
    - Es necesario agregar una técnica de control de acceso al medio para que todos los nodos puedan compartir el cable

- **Protocolos de ejemplo**
  - **RS-232**: voltajes más altos, punto a punto
  - **RS-422**: voltajes más altos, transmisión de datos diferenciales, multi-drop
  - **RS-485**: voltajes más altos, multipunto

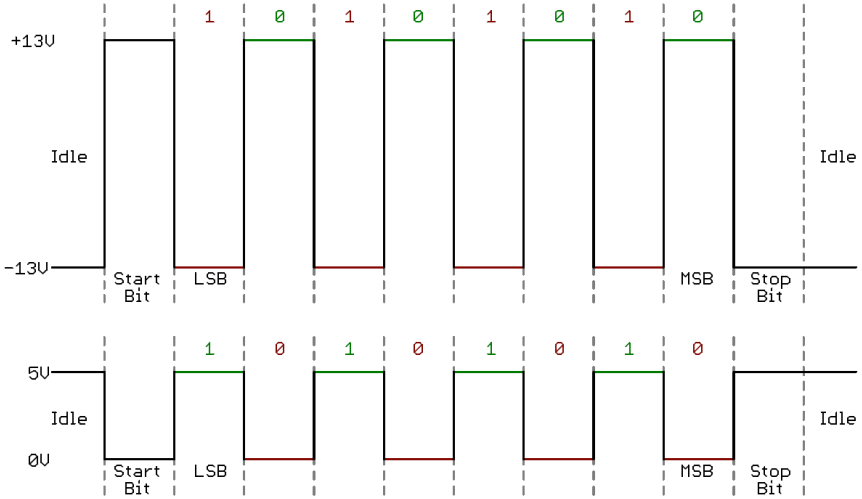
UART y TTL

- La mayoría de los microcontroladores en estos días han incorporado UART para recibir y transmitir datos.
- Este método de comunicación en serie a veces se denomina **TTL (transistor-transistor logic) serie**.
- La comunicación en serie a un nivel TTL siempre permanecerá entre los límites de 0V y Vcc, que a menudo es de 5V o 3.3V.
  - Una **lógica alta** ('1') está representada por **Vcc**, mientras que una **lógica baja** ('0') es **0V**.

RS-232

- El puerto serie del **PC** cumple con el estándar de telecomunicaciones **RS-232**.
- Las señales RS-232 son similares a las señales en serie (TTL) del microcontrolador, ya que transmiten un bit a la vez, a una velocidad de transmisión específica, con o sin paridad y / o bits de parada.
- Los dos **difieren** únicamente a nivel de hardware.
  - Por el estándar RS-232, una lógica alta ('1') está representada por un voltaje negativo, en cualquier lugar de -3 a -25V,
  - Una lógica baja ('0') transmite una tensión positiva que puede estar en cualquier lugar desde +3 a + 25V.
  - En la mayoría de las PC, estas señales oscilan entre -13 y + 13V.

TTL vs RS-232



RS-232

TTL

Conversor RS-232 <-> TTL

DSD TECH USB a TTL convertidor serie CP2102 con 4 PIN Dupont Cable Compatible con Windows 7,8,10, linux, Mac OS

de DSD TECH

★★★★★ 5 opiniones de clientes

Amazon's Choice de "usb ttl"

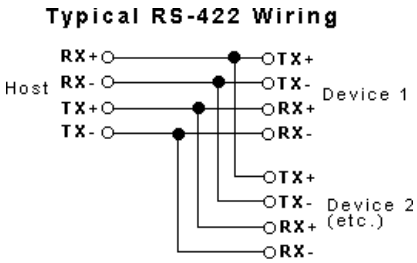
Precio: EUR 6,99 **Envío gratis** (2 días) para clientes Prime
Precio final del producto

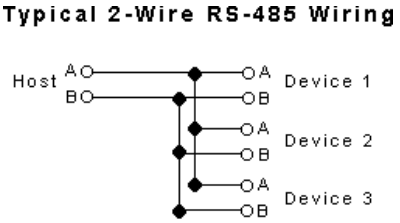
Nuevos: 1 desde EUR 6,99

- Diseñado para ser utilizado para proyectos electrónicos USB a TTL, fácil de conectar a su MCU. LED de estado para TX, RX
- CP2102 USB a la viruta de transferencia de TTL de los laboratorios del silicio, viruta industrial, alta calidad, estupendo estable.
- Fuente de alimentación USB estándar, el pin principal incluye: 3.3V, GND, TXD, RXD, 5V, GND
- Soporta Windows 2000, XP, Vista, Windows 7,8,10, Mac OS 9, Mac OS X y Linux
- GARANTÍA: Volvemos este usb al convertidor del ttl con garantía de 12 meses. Si usted resuelve cualquier pregunta, éntrenos en contacto con por favor, fijaremos su edición en el plazo de 24 horas.

› Ver más detalles

RS-232 RS-422 RS-485





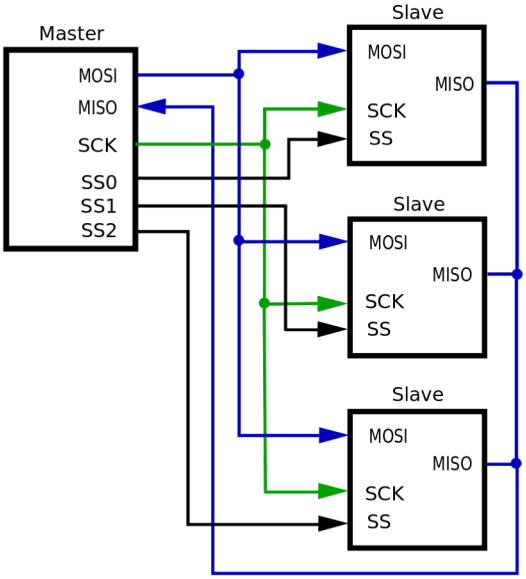
Port name	RS-232	RS-422	RS-485
Transfer type	Full duplex	Full duplex	Half duplex (2 wires), full duplex (4 wires)
Maximum distance	15 meters at 9600 bps	1200 meters at 9600 bps	1200 meters at 9600 bps
Contacts in use	TxD, RxD, RTS, CTS, DTR, DSR, DCD, GND*	TxA, TxB, RxA, RxB, GND	DataA, DataB, GND
Topology	Point-to-Point	Point-to-Point	Multi-point
Max. Number of connected devices	1	1 (10 devices in receive mode)	32 (with repeaters larger, usually up to 256)

Más info: https://www.omega.com/techref/das/rs-232-422-485.html
https://ipc2u.com/articles/knowledge-base/the-main-differences-between-rs-232-rs-422-and-rs-485/

Comunicaciones Serie

COMUNICACIONES SERIE SÍNCRONAS: SPI

Arquitectura hardware



Señales compartidas

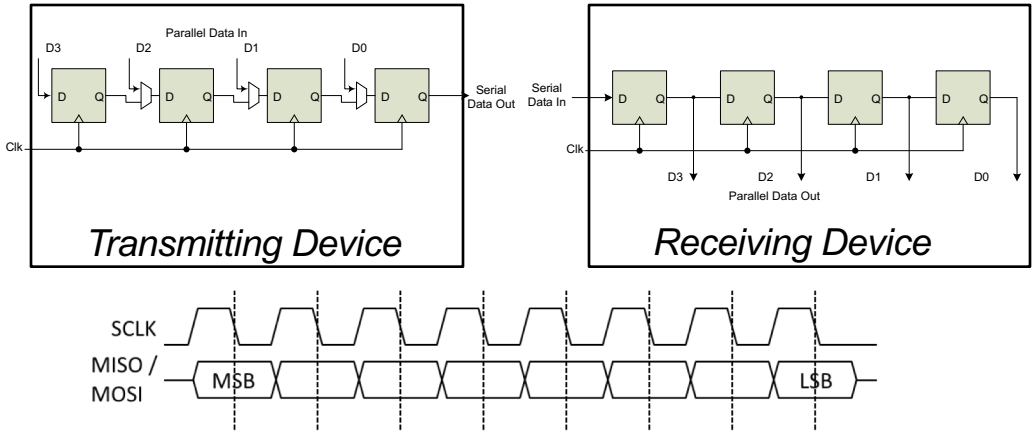
- Reloj SCK
- MOSI (salida maestra, entrada esclava)
- MISO (entrada maestra, salida esclava)

Señales individuales

- SS: Slave (Chip) Select

- ✓ Cada periférico tiene su propia línea de selección de chip (SS)
- ✓ Master (MCU) solo activa la línea SS del periférico con el que se comunica

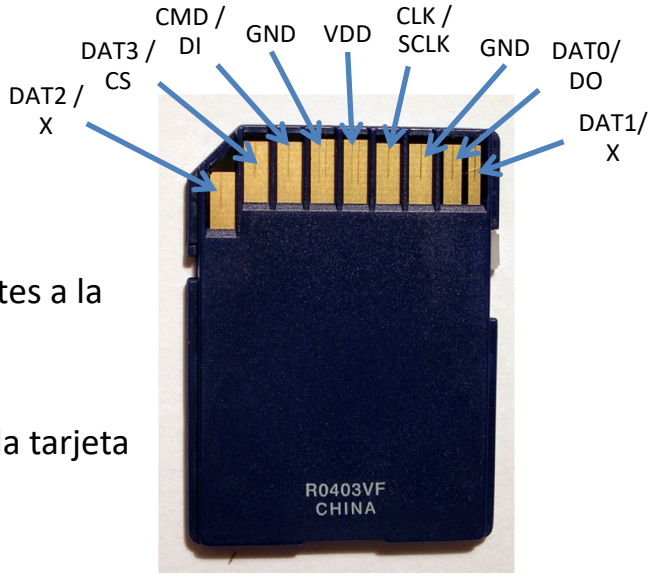
Transmisión de datos serie síncronos



- Utilice los registros de desplazamiento y una señal de reloj para convertir entre los formatos serie y paralelo
- Síncrono: una señal de reloj explícita va junto con la señal de datos

Ejemplo SPI: Secure Digital Card Access

- Las tarjetas SD tienen dos modos de comunicación
  - Nativo de 4 bits
  - SPI heredado (legacy) de 1 bit
- VDD de 2.7 a 3.6 V
- CS: Chip Select (activo a nivel bajo)

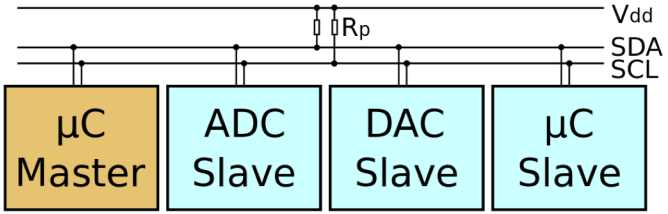


- Host envía un paquete de comando de 6 bytes a la tarjeta
  - Índice, argumento, CRC
- El host lee bytes desde la tarjeta hasta que la tarjeta señala que está listo
  - La tarjeta devuelve
    - 0xff mientras ocupado
    - 0x00 cuando esté listo sin errores
    - 0x01-0x7f cuando se ha producido un error

Comunicaciones Serie

COMUNICACIONES SERIE SÍNCRONAS: I^2C

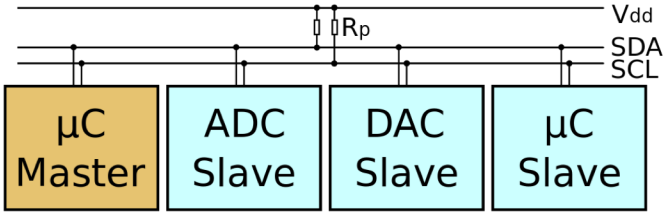
I²C: Descripción general del bus



Author: Colin M.L. Burnett, Source: Wikimedia

- **"Inter-Integrated Circuit" bus**
- **Múltiples dispositivos** conectados por un bus serie compartido.
- El bus es típicamente controlado por el dispositivo **maestro**, los **esclavos** responden cuando se les dirige
- El bus I2C tiene dos líneas de señal:
  - **SCL**: reloj serie
  - **SDA**: datos serie

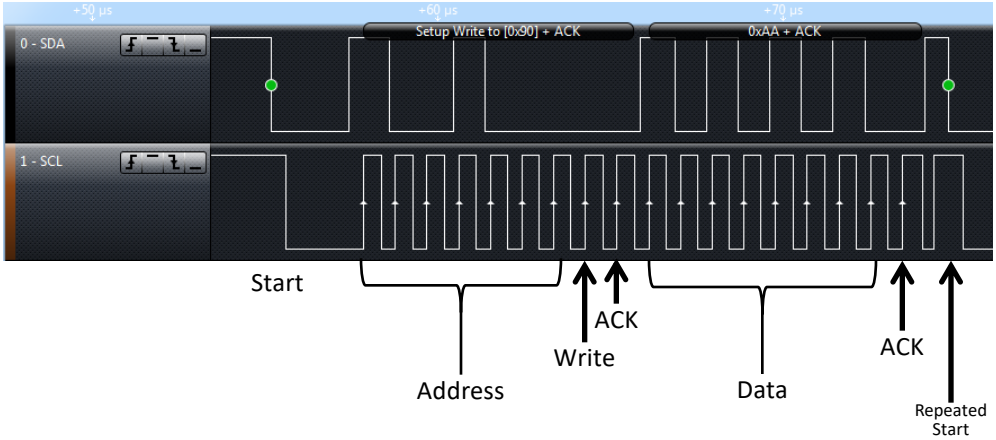
I²C: conexiones



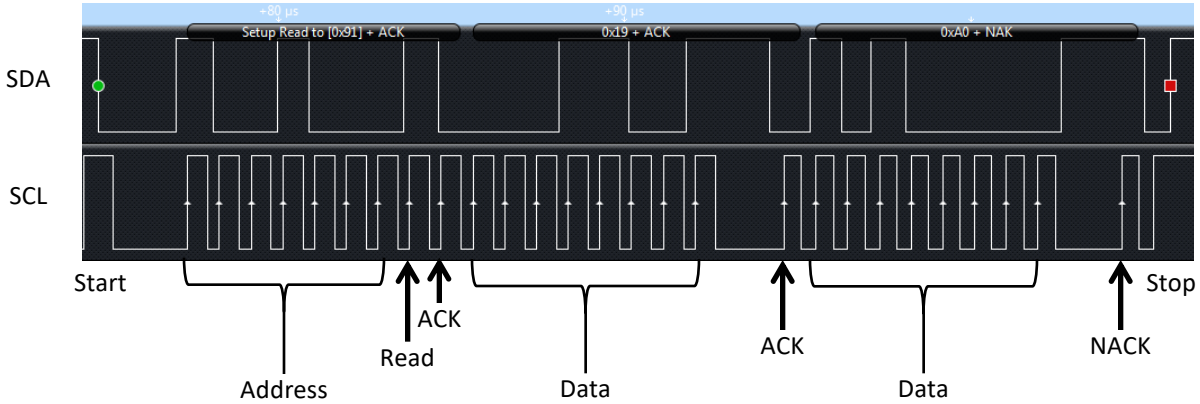
Author: Colin M.L. Burnett, Source: Wikimedia

- Las **resistencias** tiran (pull) de las líneas hacia VDD (up).
- Los **transistores** de drenaje abierto tiran de las líneas hacia tierra.
- El maestro genera la señal de reloj **SCL**
  - Puede alcanzar 400 kHz, 1 MHz o más.

I²C: Maestro escribiendo datos en un esclavo



I²C: Maestro leyendo datos de un esclavo



I²C: direccionamiento

- Cada dispositivo (IC) tiene una **dirección de 7 bits**.
  - Diferentes tipos de dispositivos tienen diferentes direcciones predeterminadas
  - A veces se puede seleccionar una dirección predeterminada secundaria al vincular un pin de dispositivo a un nivel lógico diferente

- ¿Qué sucede si tratamos el primer byte de datos como una dirección de registro?
  - Puede especificar registros dentro de un dispositivo dado: ocho bits

Comunicaciones Serie

COMPARACIÓN DE PROTOCOLOS

Factores a considerar

- **Velocidad de transmisión de los datos**
  - Depende de la tasa de bits en bruto y la sobrecarga de protocolo en la trama
- **¿Cuántas señales de hardware necesitamos?**
  - Puede necesitar línea de reloj, líneas de selección de chip, etc.
- **¿Cómo conectamos múltiples dispositivos (topología)?**
  - Enlace dedicado y hardware por dispositivo - **punto a punto**
  - Un bus para la transmisión maestra / esclavo, un bus para la transmisión esclava / maestro
  - Todos los transmisores y receptores conectados al mismo bus - **multipunto**
- **¿Cómo direccionamos un dispositivo de destino?**
  - Señal de hardware discreta (línea de selección de chip)
  - Dirección incorporada en el paquete, descodificada internamente por el receptor
- **¿Cómo cambian estos factores a medida que agregamos más dispositivos?**

Tabla comparativa de protocolos serie

Protocol	Velocidad	Señales necesarias para comunicación bidireccional con N dispositivos	Direccionamiento de dispositivos	Topología
UART (Point to Point)	Fast – Tens of Mbit/s	2*N (TxD, RxD)	No hay	Point-to-point full duplex
UART (Multi-drop)	Fast – Tens of Mbit/s	2 (TxD, RxD)	Añadido por SW por el usuario	Multi-drop
SPI	Fast – Tens of Mbit/s	3+N SCLK, MOSI, MISO N = 1 SS por dispositivo	Señal de selección de chip de HW por dispositivo	Multi-point full-duplex, multi-drop half-duplex buses
I^2C	Moderate – 100 kbit/s, 400 kbit/s, 1 Mbit/s, 3.4 Mbit/s. Packet overhead.	2 (SCL, SDA)	En el paquete	Multi-point half-duplex bus

Bibliografía

- Documentación STM32F4
- Wikibooks, Serial Programming:
  - https://en.wikibooks.org/wiki/Serial_Programming
- Discovering the STM32 Microcontroller, Geoffrey Brown:
  - https://www.cs.indiana.edu/~geobrown/book.pdf