Lección 10: Fundamentos del bus i2c

Publicado: 19 Abril 2014 | Escrito por F.J.Alexandre | Imprimir | Visto: 6124

[Proteus versión 8.1]

Como vimos en la lección anterior, nuestro Arduino puede expandirse y aumentar significativamente sus capacidades utilizando el bus i2c. En esta lección vamos a introducirnos en los fundamentos de este bus.

El bus I2C es un estándar diseñado por Philips, que facilita la comunicación entre diferentes microcontroladores, memorias y otros dispositivos con una cierta capacidad de contener lógica. Utiliza una tecnología de tipo bus. Es decir que sólo requiere dos hilos o pistas para intercomunicar todos los componentes entre sí, por los que se trasmiten los datos vía serie. En la imagen siguiente (sacada de un documento original de Philips) se puede observar la topografía de este sistema de comunicaciones en un ejemplo de utilización bastante complejo con muchas de las posibilidades que brinda (incluidos extensores de bus, utilización de múltiples maestros, etc.).

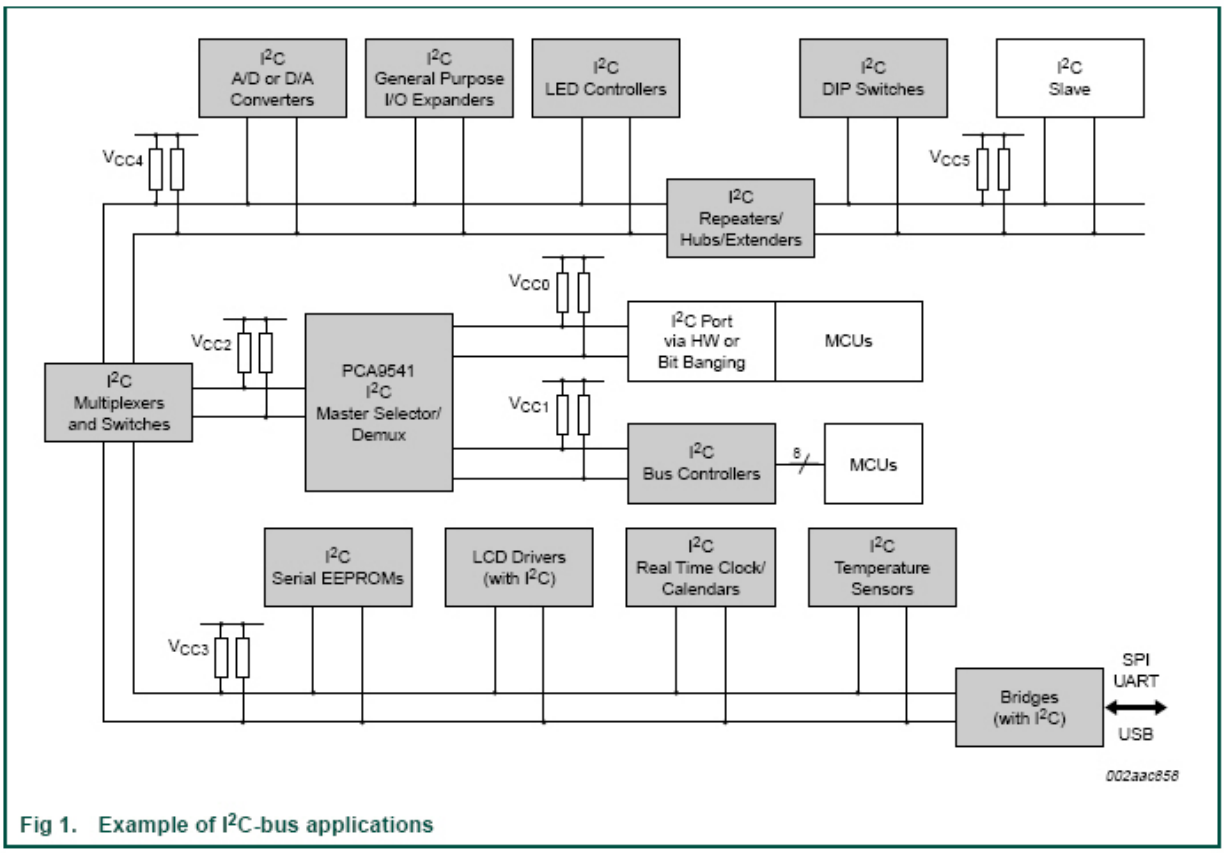


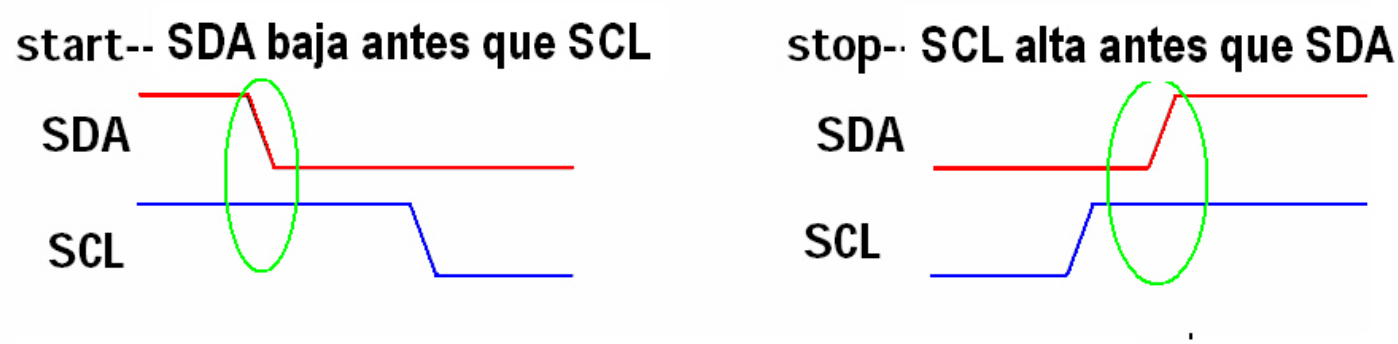
Fig 1. Example of I²C-bus applications

Se utiliza uno de los dos conductores para manejar la sincronización mediante la generación de un tren de pulsos de reloj (SCL) y el otro para intercambiar los datos (SDA). Las dos líneas, SDA (Serial Data) y SCL (Serial Clock), se conectan a la fuente de alimentación a través de dos resistencias de pull-up. Cuando el bus está libre, ambas líneas están en nivel alto. Cuando algún dispositivo quiera utilizar el bus, lo hará poniendo en ambas líneas valores de alto o bajo según las necesidades. Como la línea está en descanso en nivel alto, la forma de actuar cada uno de los dispositivos sobre ella es ponerlo a masa y de esa manera forzar el nivel bajo en esa línea. La línea de reloj sólo la gobierna el maestro (puede haber varios maestros, pero en cada momento sólo un dispositivo actúa como tal). Mientras que la línea de datos la gobiernan tanto el dispositivo maestro como el esclavo que es interrogado en ese momento.

El Maestro es el dispositivo que inicia (y gobierna mientra dura) la transferencia de datos por el bus. El bus está diseñado para permitir la participación de varios Maestros, ya que el protocolo incluye un sistema detector de colisiones. El maestro se encarga de generar la señal de reloj. El esclavo es el dispositivo direccionado que escucha y, si es necesario, responde al requerimiento del maestro. Cada dispositivo tiene una única dirección que lo identifica de modo unívoco dentro del bus, de tal manera que siempre está claro a qué dispositivo interroga el maestro y qué dispositivo debe responder a esa consulta. Si por error, configuramos dos dispositivos con la misma dirección se producirá una colisión en el bús y el sistema no funcionará correctamente.

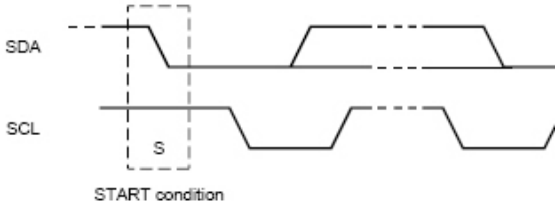
La cantidad de dispositivos que se pueden conectar al bus está limitada, unicamente, por la máxima capacidad permitida, que es de 400 pF y el número máximo de dispositivos que se pueden direccionar. Cada dispositivo puede operar como transmisor o receptor de datos, dependiendo de su función. Por ejemplo, un display es solo un receptor de datos. Y, como otro ejemplo, una memoria es un emisor y receptor de datos.

Los bits de datos se transmiten por la línea SDA. Por cada bit de información se necesita un pulso del reloj en la línea SCL. El dato transmitido por la línea SDA sólo pueden cambiar cuando la línea SCL está a nivel bajo. Están prohibidas las transiciones de la línea SDA mientra la línea SCL esté en nivel alto. La única excepción a esta regla son las señales de arranque (start) y parada (stop).

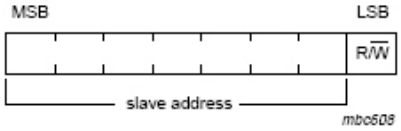


- Start –SDA baja cuando el reloj (SCL) es alto
- Stop – SDA sube cuando SCL es alto
- (Normalmente no hay transición cuando el reloj es alto)

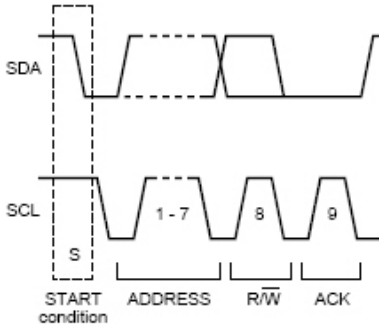
Sólo los maestros pueden iniciar una comunicación. La condición inicial, de bus libre, es cuando ambas líneas están en estado lógico alto. En esta situación cualquier maestro puede disponer del bus. Para hacerlo, establece la condición de inicio (start). Esta condición de inicio se produce cuando un maestro pone en estado bajo la línea de datos (SDA) y deja en nivel alto la línea de reloj (SCL).



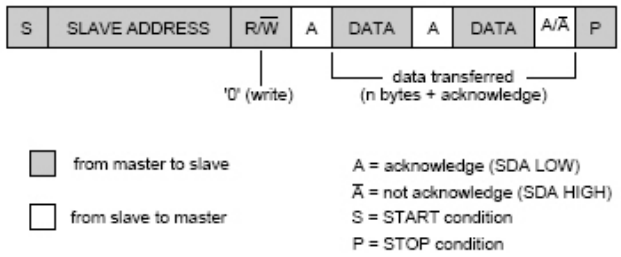
El primer byte que se transmite a continuación de la condición de inicio, está formado por siete bits que contienen la dirección del dispositivo con el que se desea establecer la comunicación, más un octavo bit que indica el tipo de operación que se establecerá con él (lectura o escritura).



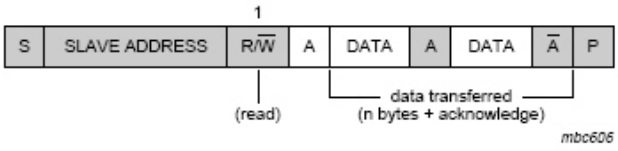
Si el dispositivo cuya dirección se corresponde con la indicada en los siete primeros bits (A0-A6) está presente en el bus, contestará con un bit bajo, ubicado inmediatamente a continuación del octavo bit que ha enviado el dispositivo maestro. Este bit de reconocimiento (ACK) en bajo le indica al dispositivo maestro que el esclavo reconoce la solicitud y está en condiciones de comunicarse.



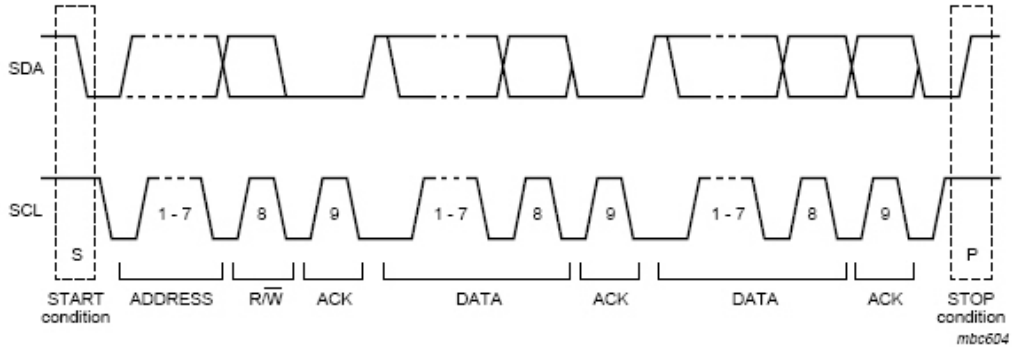
En este momento, en el que ya ha quedado claro quien es el maestro y cúal es el esclavo que debe mantener la comunicación, comienza el intercambio de información entre los dos dispositivos. Si el bit de lectura/escritura (R/W) fue puesto a nivel lógico bajo (escritura), el dispositivo maestro envía datos al dispositivo esclavo. Este proceso se mantiene mientras se reciban señales de reconocimiento por parte del esclavo. La comunicación concluye cuando se hayan transmitido todos los datos.



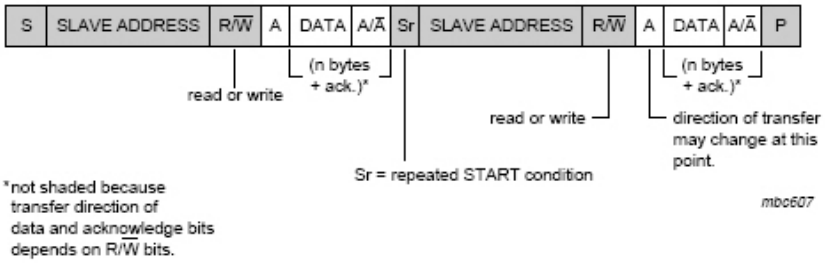
Cuando el bit de lectura/escritura fue puesto a nivel alto (modo lectura), el maestro genera pulsos de reloj para que el dispositivo esclavo pueda enviar los datos. Después de cada byte recibido el maestro que está recibiendo los datos genera un pulso de reconocimiento.



El maestro deja libre el bus generando una condición de parada (stop).



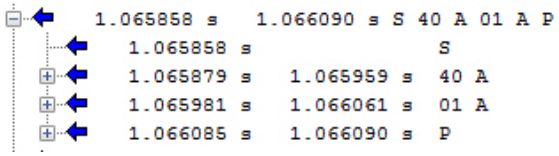
También es posible un modo mixto, donde el maestro envía datos al esclavo y a continuación recibe datos de ese mismo esclavo. En este caso el maestro genera otra condición de inicio en lugar de una condición de parada. Esta nueva condición de inicio se denomina "inicio reiterado" y se puede emplear para direccionar un dispositivo esclavo diferente o para alterar el estado del bit de lectura/escritura.



Lo más común en los dispositivos para el bus I2C es que utilicen direcciones de 7 bits, aunque existen dispositivos capaces de gestionar direcciones de 10 bits. Una dirección de 7 bits implica que se pueden poner hasta 128 dispositivos sobre un bus I2C, ya que un número de 7 bits puede ir desde 0 a 127. La mayoría de los proyectos tienen más que suficiente con esta capacidad de direccionamiento y no es habitual encontrarnos con casos donde nos veamos comprometidos por esta limitación. El hecho de colocar la dirección de 7 bits en los 7 bits más significativos del byte produce confusiones entre quienes comienzan a trabajar con este bus. Si, por ejemplo, se desea escribir en la dirección 21 (hexadecimal), en realidad se debe enviar un 42, que es un 21 desplazado un bit hacia arriba.

La lección de hoy ha sido evidentemente teórica. Pero no debemos asustarnos. Hemos explicado todo esto porque una buena base teórica nos ayudará a entender cómo funciona nuestro bus i2c y, sobre todo, a detectar de donde provienen los problemas cuando se produzcan. Pero no hay que olvidar que las librerías de software disponibles nos facilitan toda la gestión del bus y nos permiten olvidarnos de todos estos entresijos a la hora de programar.

En la lección anterior vimos que era muy fácil utilizar el bus i2c. En ella utilizamos un dispositivo i2c que nos proporcionaba 8 salidas digitales adicionales (el PCF8574). También vimos que podíamos utilizar el depurador de mensajes i2c incluído en Proteus. Con el depurador podíamos visualizar los mensajes que se transmitían por el bus. En concreto transmitíamos una secuencia de órdenes que iban encendiendo y apagando los leds situados en las ocho salidas de forma secuencial. En un momento dado, uno de los mensajes transmitidos era el siguiente:



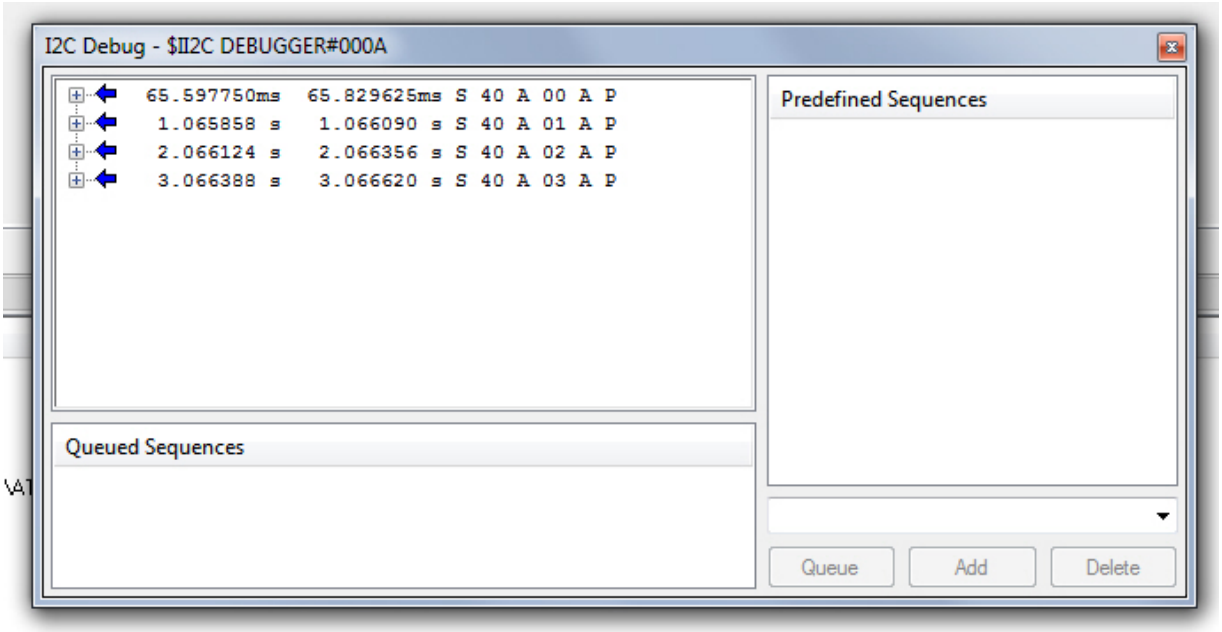
Ahora, a la luz de la teoría vista, ya podemos interpretar este mensaje:

En primer lugar la 'S' simboliza el bit de arranque.

Luego se trasmite el byte de valor 40 (0x40 en hexadecimal) que representa la dirección del esclavo. Si recordáis nuestro componente PCF8574 tenía la dirección 20 en hexadecimal (0x20). 0x20 es 10 0000 en binario. Si escribimos este número en binario en los siete bits superiores y le añadimos en el octavo bit el valor 0 para indicar que es una orden de lectura, tenemos el valor 100 0000 que en hexadecimal se corresponde con el valor 0x40.

La 'A' que aparece a continuación es el pulso de reconocimiento (acknowledgement) que devuelve el esclavo para decir que está listo.

El siguiente byte 00 (0x0 en hexadecimal) es el byte de datos que se transmite. Si visualizamos el programa de la lección anterior, veremos que este valor de byte va aumentando en cada nuevo mensaje para ordenar el encendido secuencial de los leds.



La nueva 'A' que aparece detrás del byte del dato, es el nuevo pulso de reconocimiento que devuelve el esclavo para decir que ha recibido el dato correctamente.

Por último, la 'P' es el bit de parada que manda el maestro para dar por concluido las comunicaciones.

Sencillo ¿verdad? Depurar ahora nuestras comunicaciones i2c y descubrir los posibles errores es un poco más fácil.

En posteriores lecciones aumentaremos nuestro conocimiento y utilizaremos de forma más intensiva las capacidades del bus i2c.

Comparte este artículo en redes sociales

Categoría: Curso de programación de Arduino utilizando Proteus