Tema 6 – Comunicaciones con Arduino

(Recopilado de Internet)

Arduino

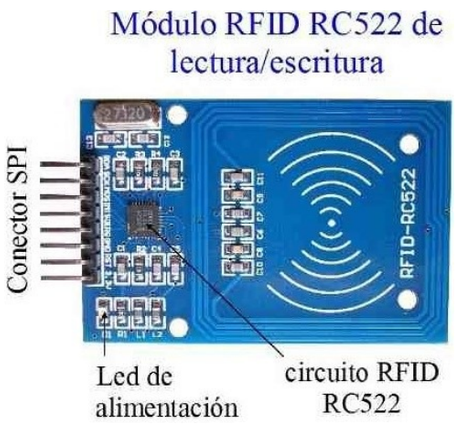
UART recepción-transmisión asíncrona universal

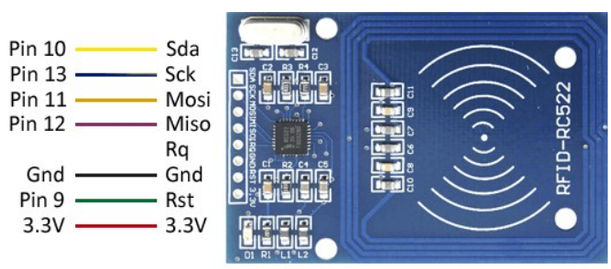
SPI

SCK para el reloj
MOSI para el Maestro
Out Esclavo In
MISO para Maestro
In Esclavo Out.

Para controlar más de un esclavo es preciso utilizar SS (selección de esclavo).

Ej:





I2C Inter-Integrated Circuit

Las líneas se llaman:

- SDA: datos
- SCL: reloj
- GND: tierra

EJ:

Al comenzar a usar Arduino puede resultar algo complejo entender las diferencias entre los diferentes tipos de interfaces de comunicación (y protocolos asociados).

Dentro la comunicación serie tenemos:

UART (recepción-transmisión asíncrona universal) es uno de los protocolos serie más utilizados. La mayoría de los microcontroladores disponen de hardware UART. Usa una línea de datos simple para transmitir y otra para recibir datos. Comúnmente, 8 bits de datos son transmitidos de la siguiente forma: un bit de inicio, a nivel bajo, 8 bits de datos y un bit de parada a nivel alto.

UART se diferencia de SPI y I2C en que es asíncrono y los otros están sincronizados con señal de reloj. La velocidad de datos UART está limitado a 2Mbps

SPI es otro protocolo serie muy simple. Un maestro envia la señal de reloj, y tras cada pulso de reloj envía un bit al esclavo y recibe un bit de éste. Los nombres de las señales son por tanto SCK para el reloj, MOSI para el Maestro Out Esclavo In, y MISO para Maestro In Esclavo Out. Para controlar más de un esclavo es preciso utilizar SS (selección de esclavo).

I2C es un protocolo síncrono. I2C usa solo 2 cables, uno para el reloj (SCL) y otro para el dato (SDA). Esto significa que el maestro y el esclavo envían datos por el mismo cable, el cuál es controlado por el maestro, que crea la señal de reloj. I2C no utiliza selección de esclavo, sino direccionamiento.

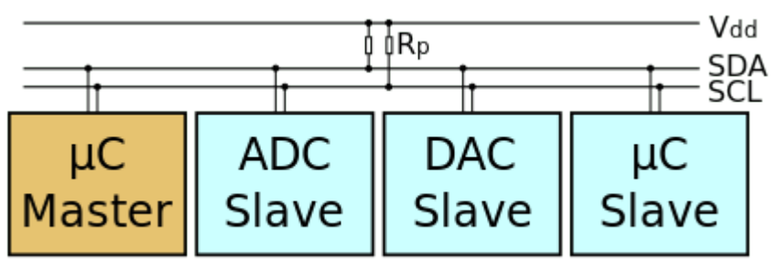
Dos o más señales a través del mismo cable pueden causar conflicto, y ocurrirían problemas si un dispositivo envía un 1 lógico al mismo tiempo que otro envía un 0. Por tanto el bus es “cableado” con dos resistencia para poner el bus a nivel alto, y los dispositivos envían niveles bajos. Si quieren enviar un nivel alto simplemente lo comunican al bus.

I²C es un bus de comunicaciones en serie. Su nombre viene de Inter-Integrated Circuit (Inter-Circuitos Integrados). La versión 1.0 data del año 1992 y la versión 2.1 del año 2000, su diseñador es Philips. La velocidad es de 100 kbit/s en el modo estándar, aunque también permite velocidades de 3.4 Mbit/s. Es un bus muy usado en la industria, principalmente para comunicar microcontroladores y sus periféricos en sistemas integrados (Embedded Systems) y generalizando más para comunicar circuitos integrados entre si que normalmente residen en un mismo circuito impreso.

La principal característica de I²C es que utiliza dos líneas para transmitir la información: una para los datos y otra para la señal de reloj. También es necesaria una tercera línea, pero esta sólo es la referencia (masa). Como suelen comunicarse circuitos en una misma placa que comparten una misma masa esta tercera línea no suele ser necesaria.

Las líneas se llaman:

- SDA: datos
- SCL: reloj
- GND: tierra

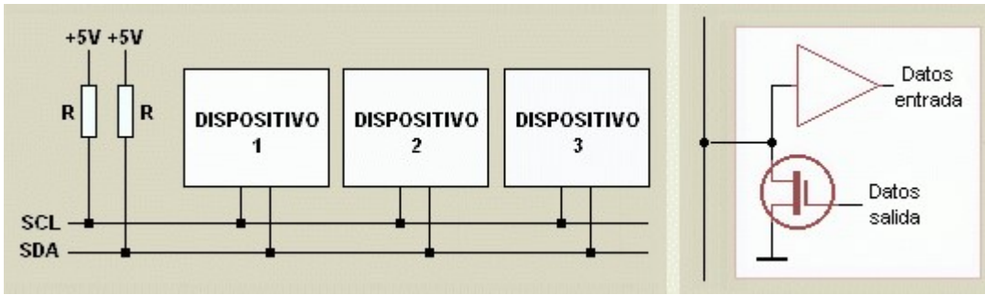


Las transacciones en el bus I2C tienen este formato:

| start | A7 A6 A5 A4 A3 A2 A1 R/W | ACK | … DATA … | ACK | stop | idle |

Los dispositivos conectados al bus I²C tienen una dirección única para cada uno. También pueden ser maestros o esclavos. El dispositivo maestro inicia la transferencia de datos y además genera la señal de reloj, pero no es necesario que el maestro sea siempre el mismo dispositivo, esta característica se la pueden ir pasando los dispositivos que tengan esa capacidad. Esta característica hace que al bus I²C se le denomine bus multimaestro.

Las líneas SDA y SCL son del tipo drenaje abierto, es decir, un estado similar al de colector abierto, pero asociadas a un transistor de efecto de campo (o FET). Se deben polarizar en estado alto (conectando a la alimentación por medio de resistores “pull-up”) lo que define una estructura de bus que permite conectar en paralelo múltiples entradas y salidas.



- El maestro comienza la comunicación enviando un patrón llamado “start condition”. Esto alerta a los dispositivos esclavos, poniéndolos a la espera de una transacción.
- El maestro se dirige al dispositivo con el que quiere hablar, enviando un byte que contiene los siete bits (A7-A1) que componen la dirección del dispositivo esclavo con el que se quiere comunicar, y el octavo bit (A0) de menor peso se corresponde con la operación deseada (L/E), lectura=1 (recibir del esclavo) y escritura=0 (enviar al esclavo).
- La dirección enviada es comparada por cada esclavo del bus con su propia dirección, si ambas coinciden, el esclavo se considera direccionado como esclavo-transmisor o esclavo-receptor dependiendo del bit R/W.
- Cada byte leido/escrito por el maestro debe ser obligatoriamente reconocido por un bit de ACK por el dispositivo maestro/esclavo.
- Cuando la comunicación finaliza, el maestro transmite una “stop condition” para dejar libre el bus.

I2C también se conoce como TWI (Two Wire Interface) y no dispone de un conector estandarizado.

SMBus (Bus de Administración del Sistema) es un subconjunto del protocolo I²C definido por Intel en 1995. Todas las placas bases modernas tienen un bus SMBus al que se conectan la mayor parte de los chips de monitorización del sistema. Estos chips sirven para medir temperaturas de componentes, velocidad de ventiladores, voltajes, etc. Toda clase de información sobre hardware.

La principal diferencia de protocolos como el SPI o el I2C, y la comunicación serie que hemos visto anteriormente es que la usart (uart en este caso) no necesita un pin de señal de reloj entre ambos dispositivos para realizar la comunicación. Cada dispositivo se pone de acuerdo en la velocidad a la que se va a realizar la transmisión (se fija la velocidad), y después de una condición de inicio cada dispositivo muestrea los bits en el tiempo acordado, por lo que sólo son necesarios dos pines para que el micro pueda enviar y recibir datos.

I2C no tiene limitaciones de velocidad, el maestro genera la velocidad de reloj y I2C provee de un mecanismo que si el esclavo es más lento es capaz de ponerse el maestro en modo de espera.

En principio, el número de dispositivos que se puede conectar al bus no tiene límites, aunque hay que observar que la capacidad máxima sumada de todos los dispositivos no supere los 400 pF. El valor de los resistores de polarización no es muy crítico, y puede ir desde 1K8 (1.800 ohms) a 47K (47.000 ohms). Un valor menor de resistencia incrementa el consumo de los integrados pero disminuye la sensibilidad al ruido y mejora el tiempo de los flancos de subida y bajada de las señales. Los valores más comunes en uso son entre 1K8 y 10K.

Lo más común en los dispositivos para el bus I2C es que utilicen direcciones de 7 bits, aunque existen dispositivos de 10 bits. Este último caso es raro.

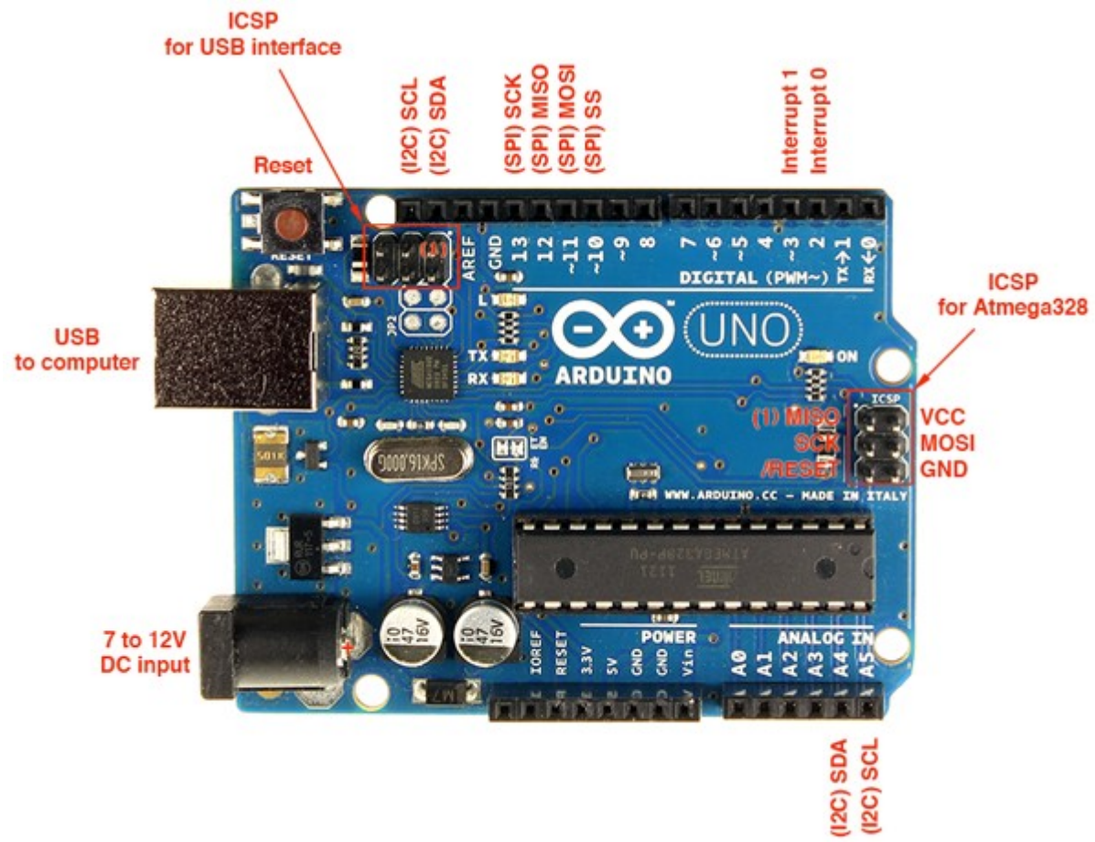
Una dirección de 7 bits implica que se pueden poner hasta 128 dispositivos sobre un bus I2C, ya que un número de 7 bits puede ir desde 0 a 127. Cuando se envían las direcciones de 7 bit, de cualquier modo la transmisión es de 8 bits. El bit extra se utiliza para informarle al dispositivo esclavo si el dispositivo maestro va a escribir o va a leer datos desde él. Si el bit de lectura/escritura (R/W) es cero, el dispositivo maestro está escribiendo en el esclavo. Si el bit es 1 el maestro está leyendo desde el esclavo. La dirección de 7 bit se coloca en los 7 bist más significativos del byte y el bit de lectura/escritura es el bit menos significativo.

Cuando los datos son envaidos por SDA, los pulsos de reloj son enviados por SCL para mantener el maestro y el esclavo soncronicados. Puesto que los datos son enviados como un bit en cada pulso de reloj, la transferencia de datos es un octavo la frecuencia de reloj. La frecuencia del reloj estandar originalmente se puso a 100KHz y la mayoría de los integrados y microcontroladores soportan esta velocidad. En posteriores actualizaciones, se introdujo una fast speed de 400KHz y una high speed de 1.7 a 3.4 MHz. Arduino puede soportar la velocidad estandar y fast speed, BeagleBoard tiene tres buses I2C cada uno a una velocidad distinta y tanto BeagleBoard como Raspberry Pi soportan velocidad estandar y fast speed.

Fast speed corresponde a una velocidad de transferencia de 50Kbytes/sec lo que puede ser una velocidad muy baja para algunas aplicaciones de control. Una opción en ese caso es usar SPI en lugar de I2C.

La librería para manejar el bus I2C en Arduino es Wire: http://arduino.cc/en/reference/wire

Esta librería permite comunicar con I2C / TWI Arduino con otros dispositivos. En las placas Arduino con el diseño R3 (1.0 pinout), la SDA (línea de datos) y SCL (línea de reloj) están en los pines cerca del pin AREF.



El Arduino Due tiene dos interfaces I2C / TWI SDA1 y SCL1 que están cerca del pin AREF y los adicionales en los pines 20 y 21.

En el Arduino Uno los pines I2C son: A4 (SDA), A5 (SCL) y en el Mega son: 20 (SDA), 21 (SCL)

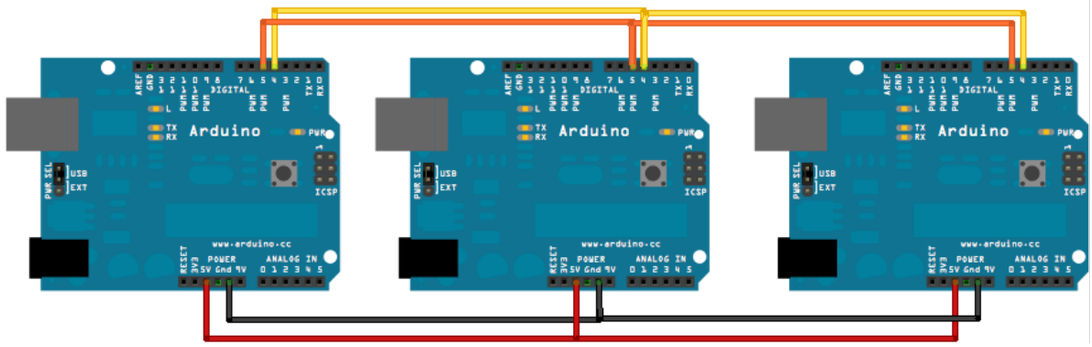
Funciones:

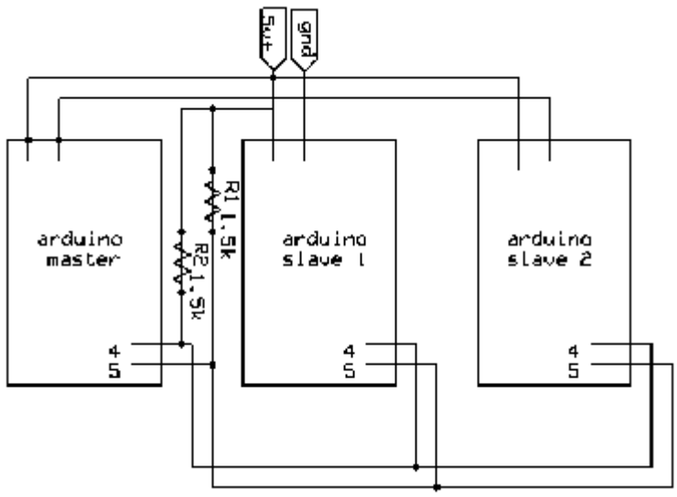
- begin() – Inicia la libreria Wire y especifica si es master o slave
- requestFrom() – Usado por el maestro para solicitar datos del esclavo
- beginTransmission() – Comenzar transmision con esclavo.
- endTransmission() – Finaliza la transmisión que comenzó con un esclavo y transmite los bytes en cola.
- write() – Escribe datos desde un esclavo como respuesta a una petición del maestro o pone en cola la transmisión de un maestro.
- available() – Devuelve el número de bytes para leer
- read() – Lee un byte transmitido desde un esclavo a un maestro o viceversa
- onReceive() – Llama a una función cuando un esclavo recibe una transmision de un maestro.
- onRequest() – Llama a una función cuando un maestro solicita datos de un maestro.

Ejercicio33-I2C: comunicar por I2C tres arduinos

Ejemplo de uso con dos Arduinos:

- http://arduino.cc/en/Tutorial/MasterWriter
- http://arduino.cc/en/Tutorial/MasterReader
- http://arduino.cc/en/Tutorial/DigitalPotentiometer
- http://www.electroensaimada.com/i2c.html





Solución en https://github.com/jecrespo

SPI

SPI fue originalmente desarrollado por Motorola, ahora Freescale.

El Bus SPI (del inglés Serial Peripheral Interface) es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. El bus de interfaz de periféricos serie o bus SPI es un estándar para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj (comunicación sincrónica).

Incluye una línea de reloj, dato entrante, dato saliente y un pin de chip select, que conecta o desconecta la operación del dispositivo con el que uno desea comunicarse. De esta forma, este estándar permite multiplexar las líneas de reloj.

Muchos sistemas digitales tienen periféricos que necesitan existir pero no ser rápidos. La ventajas de un bus serie es que minimiza el número de conductores, pines y el tamaño del circuito integrado. Esto reduce el coste de fabricar montar y probar la electrónica. Un bus de periféricos serie es la opción más flexible cuando se tiene tipos diferentes de periféricos serie. El hardware consiste en señales de reloj, data in, data out y chip select para cada circuito integrado que tiene que ser controlado. Casi cualquier dispositivo digital puede ser controlado con esta combinación de señales. Los dispositivos se diferencian en un número predecible de formas. Unos leen el dato cuando el reloj sube otros cuando el reloj baja. Algunos lo leen en el flanco de subida del reloj y otros en el flanco de bajada. Escribir es casi siempre en la dirección opuesta de la dirección de movimiento del reloj.

El bus SPI se define mediante 4 pines:

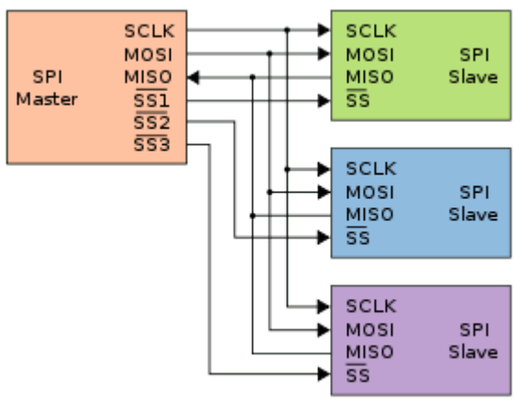
- SCLK o SCK : Señal de reloj del bus. Esta señal rige la velocidad a la que se transmite cada bit.
- MISO(Master Input Slave Output): Es la señal de entrada a nuestro dispositivo, por aquí se reciben los datos desde el otro integrado.
- MOSI(Master Output Slave Input): Transmisión de datos hacia el otro integrado.
- SS o CS: Chip Select o Slave Select, habilita el integrado hacia el que se envían los datos. Esta señal es opcional y en algunos casos no se usa.

A diferencia de otros buses el SPI no implementa el nivel del enlace entre dispositivos, es decir no hay un campo para la dirección ni un campo para ACK, etc. El SPI se comporta como un shift register donde a cada golpe de clock se captura un bit. En parte no es necesaria hacer un direccionamiento de los chips ya que mediante la señal Chip select, habilitamos al integrado al que queremos enviar los datos.

El funcionamiento para un envió de un Master es el siguiente:

- Se habilita el chip al que hay que enviar la información mediante el CS (Opcional).
- Se carga en el buffer de salida el byte a enviar.
- La linea de Clock empieza a generar la señal cuadrada donde normalmente por cada flanco de bajada se pone un bit en MOSI.
- El receptor normalmente en cada flanco de subida captura el bit de la linea MISO y lo incorpora en el buffer.

Se repite el proceso 8 veces y se ha transmitido un byte. Si se ha terminado de transmitir se vuelve a poner la linea CS en reposo. Hay que tener en cuenta que a la vez que el Master esta enviando un dato también lo recibe así que si el Slave ha depositado algún byte en el buffer de salida, este también sera enviado y recibido por el Master.



Ventajas de SPI sobre I2C.

- I2C No es Full-Duplex por lo que no permite envíos y recepciones al mismo tiempo.
- I2C un poco mas complejo que SPI.
- I2C no tiene control de errores, por ejemplo mediante paridad etc. Aunque si se puede realizar por Software.
- Velocidades de comunicación relativamente elevadas. En el caso de Arduino de hasta 8 Mhz.
- Completo control sobre la trama de bits al no exigir direccionamiento ni ACK.

Desventajas de SPI:

- No hay control del flujo por hardware.
- No hay confirmación de la recepción como si ocurre en I2C con el ACK. Es decir no sabemos si el mensaje a llegado al destino.
- Usa mas pines que otros buses, ya que necesita uno por cada esclavo. Eso implica que no hay direccionamiento en la propia trama. A menos que se diseñe por software.
- Funcionamiento a distancias cortas

La mayoría de los microcontroladores modernos tienen soporte HW para SPI como BeagleBoard y Raspberry Pi.

El estandar SPI es un estándar de facto, es decir, ha sido consensuada ni legitimada por un organismo de estandarización al efecto. Por el contrario, se trata de una norma generalmente aceptada y ampliamente utilizada por iniciativa propia de un gran número de interesados.

Por lo tanto cada fabricante puede implementar de forma diferente el SPI y debemos prestar atención al datasheet del dispositivo.

Funcionamiento de SPI en Atmega8

El SPI Master (servidor) inicializa el ciclo de comunicación cuando se coloca en bajo el Selector de Esclavo (SS-Selector Slave)(cliente). Master y Slave(servidor y cliente) preparan los datos a ser enviados en sus respectivos registros de desplazamiento y el Master genera el pulso del reloj en el pin SCK para el intercambio de datos. Los datos son siempre intercambiados desde el Maestro al Esclavo en MasterOut-SlaveIn, MOSI, y desde Esclavo al Maestro en MasterIn-SlaveOut, MISO. Después de

cada paquete de datos el Maestro debe sincronizar el esclavo llevando a ‘alto’ el selector de Esclavo, SS.

Cuando se configure como Maestro, la interfaz SPI no tendrá un control automático de la línea SS. Este debe ser manejado por software antes de que la comunicación pueda empezar, cuando esto es realizado, escribiendo un byte en el registro de la SPI comienza el reloj de la SPI, y el hardware cambia los 8 bits dentro del Esclavo. Después de cambiar un Byte, el reloj del SPI para, habilitando el fin de la transmisión (SPIF). Si la interrupción del SPI está habilitado (SPIE) en el registro SPCR, una interrupción es requerida. El Master podría continuar al cambio del siguiente byte escribiendo dentro del SPDR, o señalizar el fin del paquete colocando en alto el Esclavo seleccionado, línea SS. El último byte llegado se mantendrá en el registro Buffer para luego usarse.

Cuando lo configuramos como un Esclavo, la interfaz SPI permanecerá durmiendo con MISO en tres-estados siempre y cuando el pin SS este deshabilitado. En este estado, por el software se podría actualizar el contenido del registro SPDR, pero los datos no serán desplazados por la llegada del pulso de reloj en el pin SCK hasta que el pin SS no sea habilitado(‘0’). Será visto como un byte completamente desplazado en el fin de la transmisión cuando SPIF se habilite. Si la interrupción SPI, SPIE en SPCR, está habilitada, una interrupción es solicitada. El Esclavo podría continuar para colocar nuevos datos para ser enviados dentro del SPDR antes de seguir leyendo la data que va llegando. El último byte que entra permanecerá en el buffer para luego usarse.

Para usar el bus SPI en Arduino, se usa la librería: http://arduino.cc/en/Reference/SPI

En un primer paso importamos la librería del SPI con #include <SPI.h>. En el setup hay que iniciar y configurar el SPI con SPI.begin() y además hay que definir el pin SS como salida.

Finalmente mediante la función SPI.transfer enviamos el byte que queremos.

Métodos SPI:

- begin() — Inicializa el bus SPI
- end() — Deshabilita el bus SPI.
- setBitOrder() — Configura el orden de los bits enviados como el menos significativo primero o el más significativo primero.
- setClockDivider() — Configura del divisor de reloj en el bus SPI. ES decir configura la velocidad del bus.
- setDataMode() — Configura el modo de dato del bus SPI, es decir, polaridad y fase del reloj.
- transfer() — Transfiere un byte sobre el bus SPI, tanto de envío como de recepción.

For “Microchip PIC” / “ARM-based” microcontrollers:

SPI Mode	Clock Polarity (CPOL/CKP)	Clock Edge (CKE/NCPHA)
0	0	1
1	0	0
2	1	0
3	1	1

La siguiente tabla muestra que pines corresponden a los líneas SPI en cada Arduino:

Arduino Board	MOSI	MISO	SCK	SS (slave)	SS (master)
Uno or Duemilanove	11 or ICSP-4	12 or ICSP-1	13 or ICSP-3	10	–
Mega1280 or Mega2560	51 or ICSP-4	50 or ICSP-1	52 or ICSP-3	53	–
Leonardo	ICSP-4	ICSP-1	ICSP-3	–	–
Due	ICSP-4	ICSP-1	ICSP-3	–	4, 10, 52

Todas las placas con microcontroladores AVR tienen un pin SS que se usa cuando actúa como esclavo. Dado que esta librería solo soporta el estado maestro, este pin debe ponerse siempre como output, sino el interfaz SPI podría ponerse como esclavo.

Sin embargo, es posible usar el pin SS para dispositivos, por ejemplo el pin 4 y 10 son usados para controlar las conexiones de la Ethernet Shield.

En el Arduino Due, el interfaz SPI funciona diferente al resto de Arduinos. Este tiene 3 pines para dispositivos SS.

Uso extendido de la librería SPI en Due: http://arduino.cc/en/Reference/DueExtendedSPI

Uso del bus SPI: https://www.pjrc.com/teensy/td_libs_SPI.html

Dispositivos SPI

Diferentes tipos de periféricos con SPI: http://www.mct.net/faq/spi.html

Aplicaciones: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#Applications

Varios dispositivos con interfaz SPI: https://www.sparkfun.com/search/results?term=spi

Ejemplos de adaptadores SPI:

http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#Host_adapters

Conversor de USB a SPI: http://www.maximintegrated.com/en/products/interface/controllers-expanders/MAX3421E.html

¿Uso SPI para cargar el bootloader?

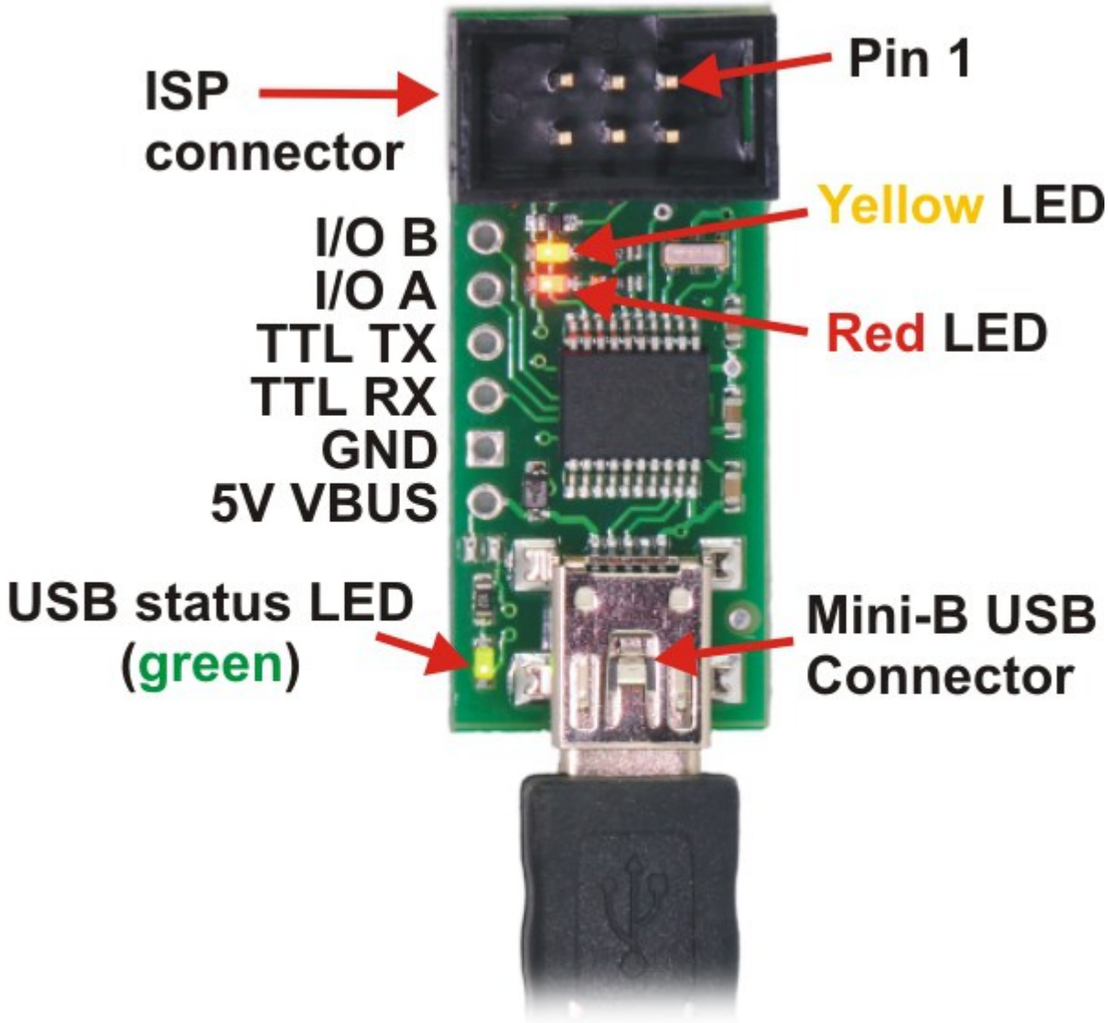
http://www.reprap.org/wiki/Clone_Wars:_Serial_Peripheral_Interface_(SPI)_aplicado_a_In_System_Programing_(ISP)

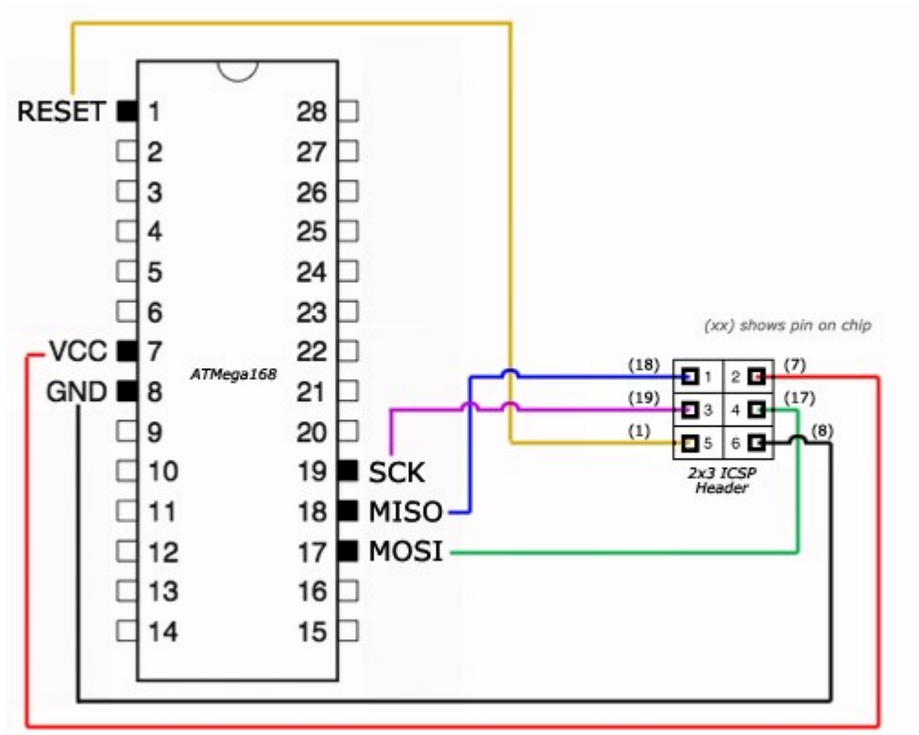
In system programming:

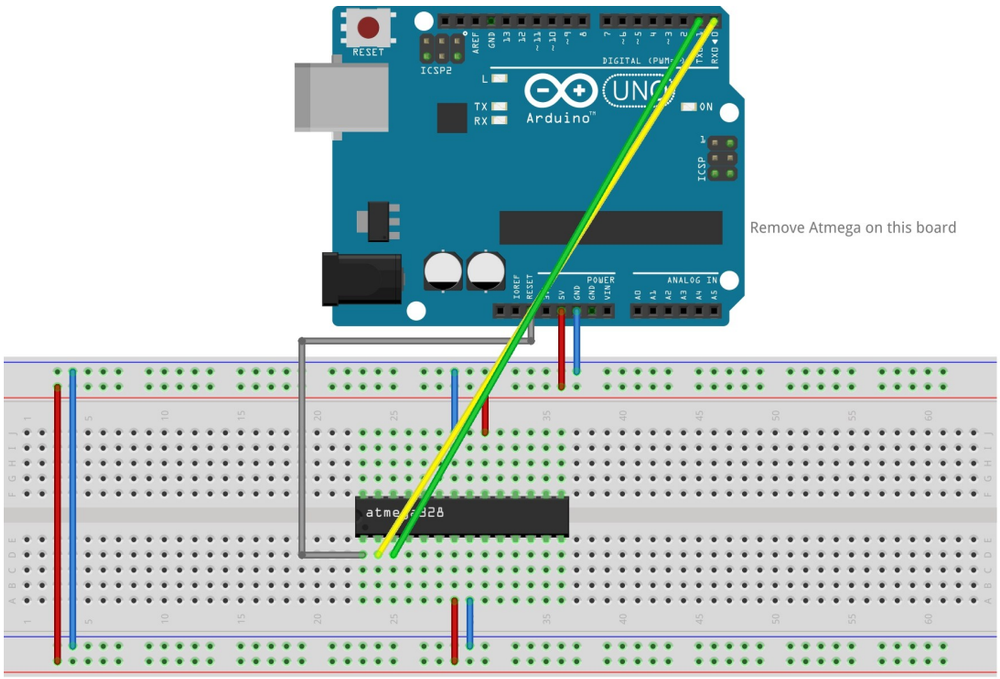
- http://es.wikipedia.org/wiki/Programaci%C3%B3n_en_el_sistema
- http://en.wikipedia.org/wiki/In-system_programming
- http://www.atmel.com/Images/doc0943.pdf → Atmel in system programming, explica como conecto al MCU directamente para programarlo con ISP.

Por ejemplo un ISP: http://www.pololu.com/product/1300

Que lleva este PIC http://ww1.microchip.com/downloads/en/DeviceDoc/41350E.pdf con un firmware para realizar esta función.


ISP
connector
Pin 1
Yellow LED
I/O B
I/O A
TTL TX
TTL RX
GND
5V VBUS
Red LED
USB status LED
(green)
Mini-B USB
Connector


RESET
VCC
GND
ATMega168
SCK
MISO
MOSI
(xx) shows pin on chip
(18)
(19)
(1)
(7)
(17)
(8)
2x3 ICSP
Header



Ejercicio34-SPI: Comunicar dos arduinos por SPI

Solución en https://github.com/jecrespo

Otros ejemplos:

- Ejercicio de controlar un potenciómetro digital por SPI:http://arduino.cc/en/Tutorial/SPIDigitalPot
- Leer un sensor de presión: http://arduino.cc/en/Tutorial/BarometricPressureSensor