Ejercicio #2

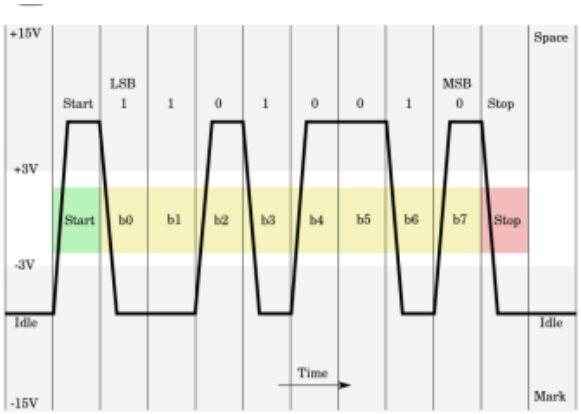
a) Que es el protocolo SPI y cuales son sus características.

Fue desarrollado por Motorola allá por los 80’s, y que es un protocolo muy utilizado para comunicarse con distintos módulos de hardware. Este tipo de comunicación se ve comúnmente en el mundo de los microcontroladores, utilizado para comunicarse con otros dispositivos como memorias, pantallas, tarjetas de sonido, y dispositivos de interfaz serie-paralelo, lo cual nos brinda la oportunidad de crear cosas muy robustas e interesantes.

El problema

La comunicación serial del tipo RX y TX (USART/UART) es una comunicación asíncrona, que no posee control de los datos que se envían sobre su bus y sin garantía de que ambos dispositivos estén funcionando siempre a la misma velocidad. Esto causa un problema ya que los dispositivos a conectar no siempre corren a la misma frecuencia, dificultando la comunicación entre ambos.

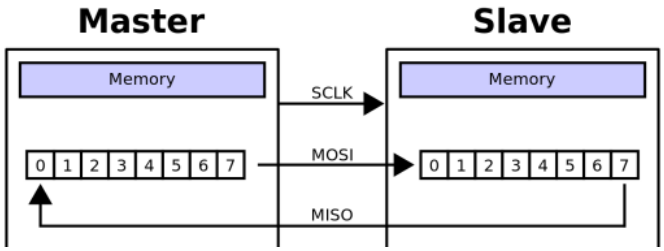
Para solucionar este error de sincronización, se agrega un bit de inicio (start) y un bit de parada(stop) al final de cada dato enviado, con lo cual el receptor sincroniza su reloj para la comunicación en cada bit de inicio.



La solución

La solución a este problema es el protocolo SPI. Es un protocolo con un bus síncrono, lo que significa que utiliza líneas separadas para datos y reloj, lo que dota a ambos dispositivos de perfecta sincronización.

El reloj le indica al dispositivo receptor el momento exacto en que puede tomar el bit de la línea de datos enviado por el transmisor. El pulso de reloj puede ser tanto de subida (rising) como de bajada (fallin). Cuando el receptor recibe este pulso, inmediatamente toma el dato de la línea y lo almacena en un registro de corrimiento. Un dato importante a tener en cuenta son los límites de velocidad del reloj del dispositivo receptor, ya que la frecuencia de la señal generada por el reloj del transmisor, puede ser mas alta que la soportada por el receptor, causando problemas en la comunicación.



Una de las razones por las que el protocolo SPI es muy popular es por que el hardware que lo compone es un simple registro de desplazamiento (shift register), el cual tiene un costo mucho más bajo que los chips USART/UART.

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.

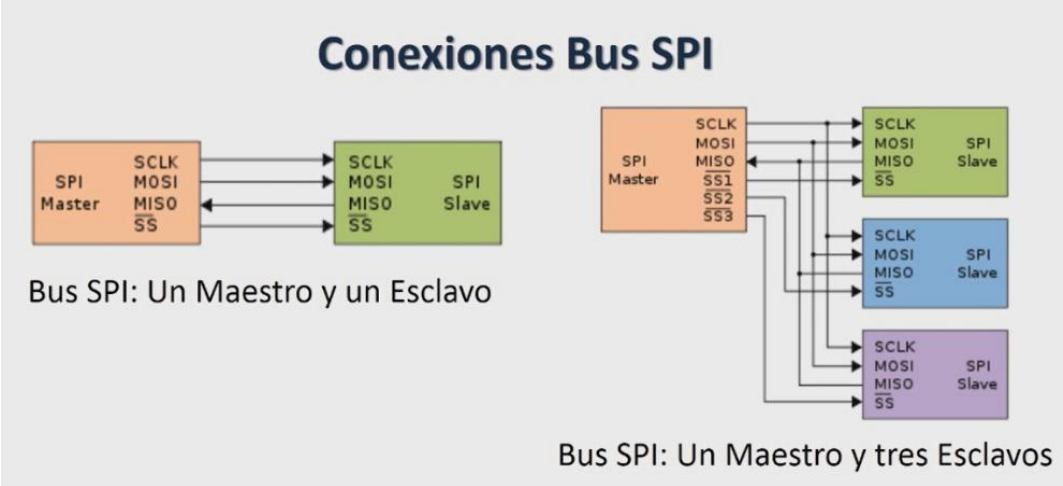
El bus SPI se define mediante 4 pines:

- 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 línea 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 línea 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.



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 7 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 tresestados 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 . En el setup hay que iniciar y configurar el SPI con SPI.begin() y además hay que definir el pin SS como salida.

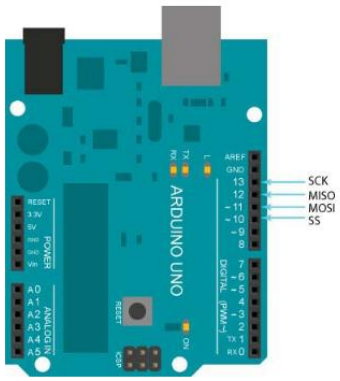
Finalmente mediante la función SPI.transfer enviamos el byte que queremos.

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.

En las placas **Arduino UNO R3** o **Arduino Duemilanove**, los pines utilizados son:

- **SS**: digital 10. Puede usar otros pines digitales, pero 10 es generalmente el valor predeterminado. Si se piensa usar el bus SPI en modo esclavo, este pin es de uso obligatorio.
- **MOSI**: digital 11
- **MISO**: digital 12
- **SCK**: digital 13



Inicialización del bus SPI

```
Inicialización de bus SPI                                   Arduino
1 SPI.begin();            // Inicializa el bus SPI
2 SPI.transfer(dato);     // Envia un byte
3 SPI.attachInterrupt(); // Activa las interrupciones para recibir datos
```

Configuración el orden de transmisión de bits del bus SPI

```
Configuracion del orden de transmisión en bus SPI           Arduino
1 SPI.setBitOrder(LSBFIRST); // Bit menos significativo primero
2 SPI.setBitOrder(MSBFIRST); // Bit mas signigicativo primero
```

Configurar la polaridad del reloj del bus SPI

```
Configuracion polaridad del reloj en bus SPI                Arduino
1 SPI.setDataMode(SPI_MODE0); // Reloj normalmente bajo, muestreo en flanco subida
2 SPI.setDataMode(SPI_MODE1); // Reloj normalmente bajo, muestreo en flanco bajada
3 SPI.setDataMode(SPI_MODE2); // Reloj normalmente alto, muestreo en flanco subida
4 SPI.setDataMode(SPI_MODE3); // Reloj normalmente alto, muestreo en flanco bajada
```

b) Cuáles son las ventajas y desventajas del protocolo SPI

Ventajas de SPI sobre I2C.

- I2C No es Full-Duplex por lo que no permite envíos y recepciones al mismo tiempo.
- I2C un poco más complejo que SPI.
- I2C no tiene control de errores, por ejemplo mediante paridad etc. Aunque 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.
- Se requiere un hardware sencillo (Barato)

- Requiere un menor consumo y menor electrónica de conexión que el I2C
- Como el Clock lo proporciona el master, los esclavos no necesitan osciladores (más barato).
- Alta velocidad de trasmisión (hasta 8 Mhz en Arduino) y Full Duplex
- Los dispositivos necesarios son sencillos y baratos, lo que hace que esté integrado en muchos dispositivos.
- Puede mandar secuencias de bit de cualquier tamaño, sin dividir y sin interrupciones.

Desventajas de SPI:

- No hay control del flujo por hardware.
- Las comunicaciones tiene que estar perfectamente establecidas de antemano. No puedes enviar mensajes de diferentes longitudes cuando convenga.
- No hay confirmación de la recepción como ocurre en I2C con el ACK. Es decir no sabemos si el mensaje a llegado al destino.
- Usa más 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
- Master único y casi sin posibilidad de master múltiple
- Se requiere 3 cables (SCK, MOSI y MISO) + 1 cable adicional (SS) por cada dispositivo esclavo.
- Solo es adecuado a corta distancias (unos 30cm)s
- No se dispone de ningún mecanismo de control, es decir, no podemos saber si el mensaje ha sido recibido y menos si ha sido recibido correctamente.
- La longitud de los mensajes enviados y recibidos tiene que ser conocida por ambos dispositivos.

Tanto el bus SPI como el I2C son llamados buses de tarjeta, es decir están pensados para trabajar a distancias pequeñas del entorno de una tarjeta, en caso de necesitar un bus serie a larga distancia hay que ir a buses de campo como RS485, CAN, etc…

El canal SPI fue diseñado para aplicaciones de transmisión de datos a velocidades altas (10 Mbps) y distancias cortas, del orden de 10 a 20 cms, ó bien dentro de un mismo PCB (circuito impreso), entre 2 circuitos integrados como podrían ser un microcontrolador y otro dispositivo, por ejemplo, un circuito integrado con la función RFID. Las señales de transmisión de datos y control del canal SPI, usan niveles de voltaje TTL ó bien 3.3 volts, dependiendo de la tecnología de fabricación del dispositivo.