CIRCUITOS DIGITALES Y MICROCONTROLADORES 2025

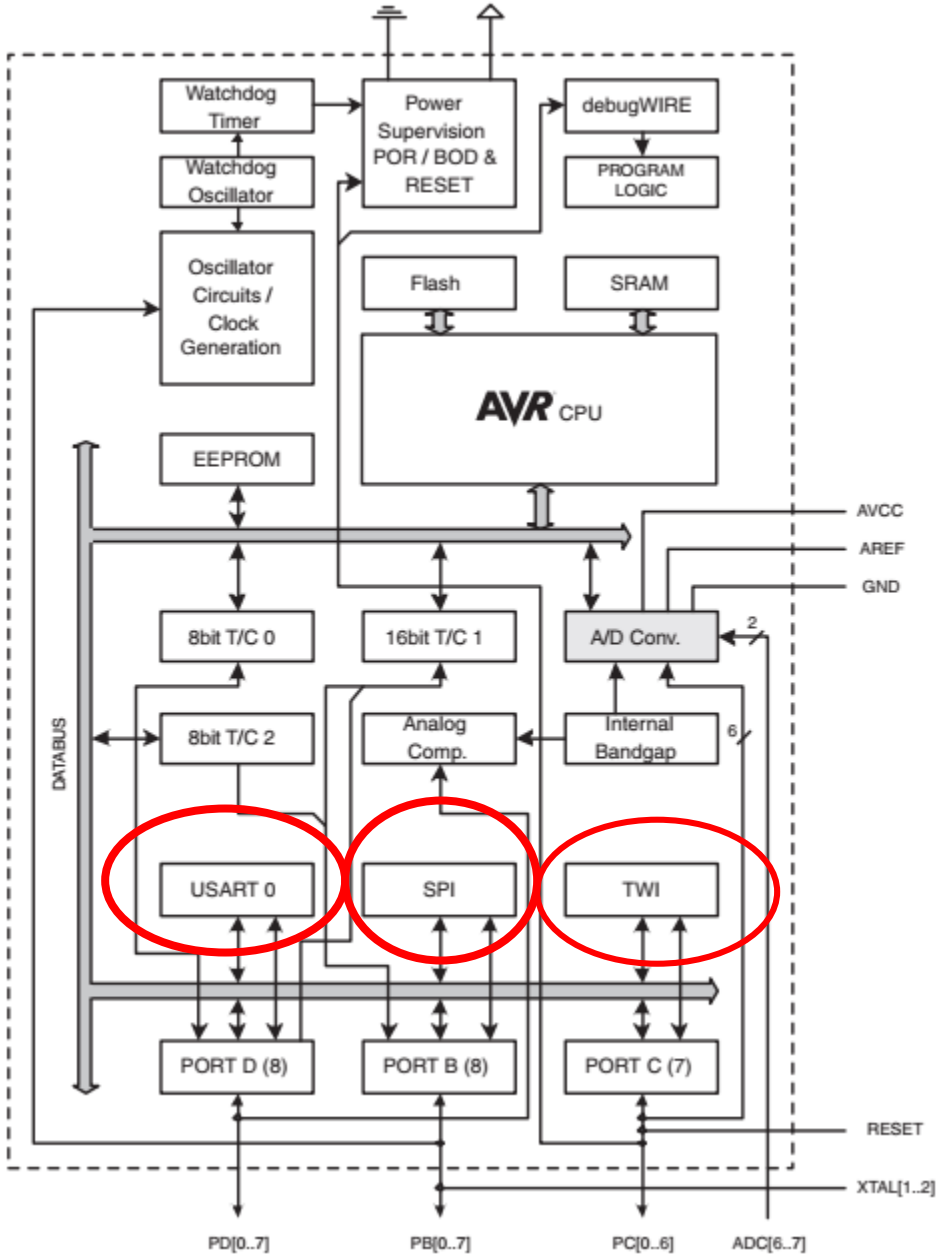
Facultad de Ingeniería UNLP

Interfaces de comunicación serie

SPI

Ing. José Juárez

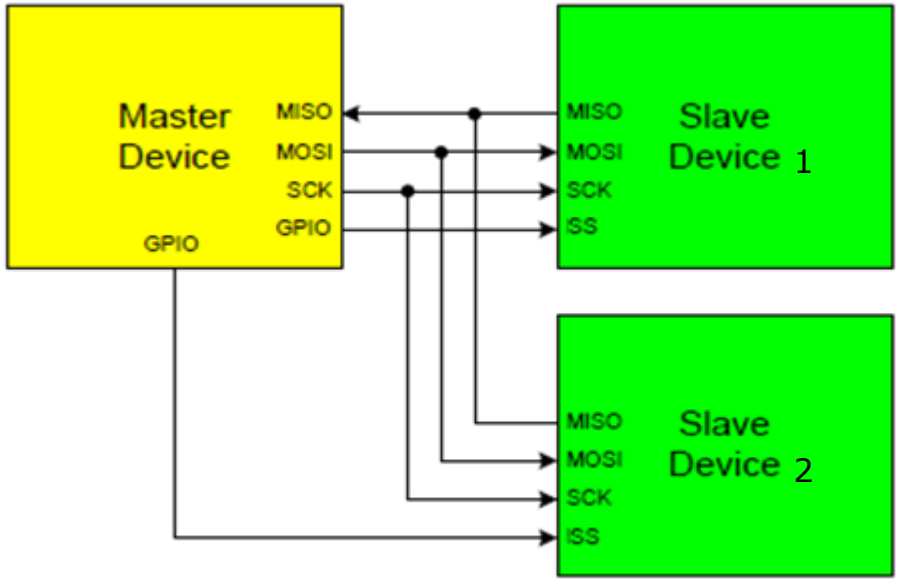
Periféricos de comunicación serie



Source: Atmega328p Data Sheet

SPI (Serial Peripheral Interface)

- Esta interfaz, desarrollada por Motorola (1991), permite una comunicación serie full dúplex, sincrónica con dispositivos periféricos u otros MCU y con tasas de trasferencia superiores al Mbps (hasta 10Mbits/s en algunos MCU).
- Por ejemplo, algunos dispositivos periféricos que utilizan SPI son: memoria FLASH externa, acelerómetros, conversores AD/DA, tarjetas SD, sistemas ISP (In System Programming).
- Utiliza una configuración Master/Slave donde el Master genera y distribuye la señal de reloj (sincrónico)
- Utiliza una línea de selección de chip por cada dispositivo esclavo (es un bus con lógica de 3 estados).



Las señales que intervienen son:

MISO: Master Input – Slave Output

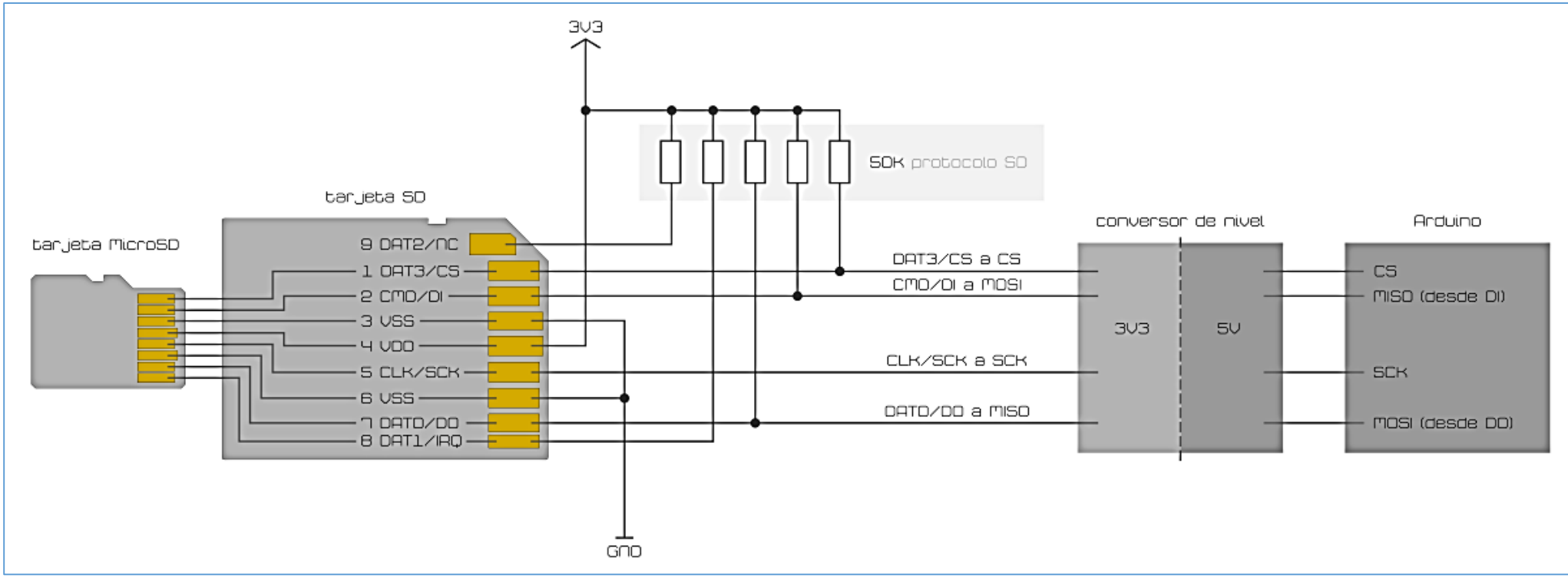
MOSI: Master Output – Slave Input

SCK: Serial Clock

SS: Slave Select (en cada slave)

SPI (Serial Peripheral Interface)

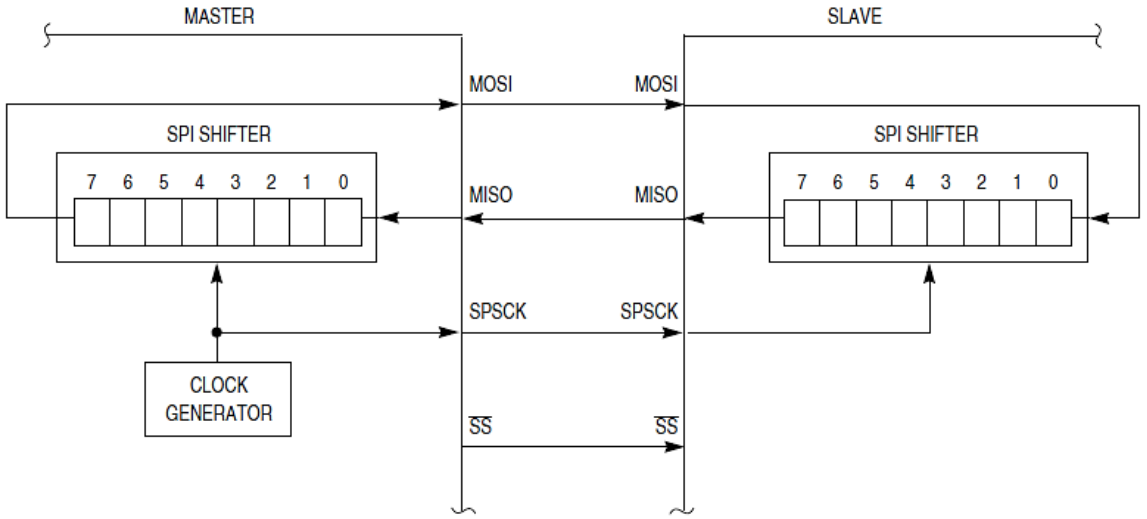
Ejemplo de conexión de Tarjeta SD por interfaz SPI



Los dispositivos de almacenamiento masivo SD utilizan lógica de 3.3V por lo tanto es necesario un conversor de niveles lógicos bidireccional.

SPI (Serial Peripheral Interface)

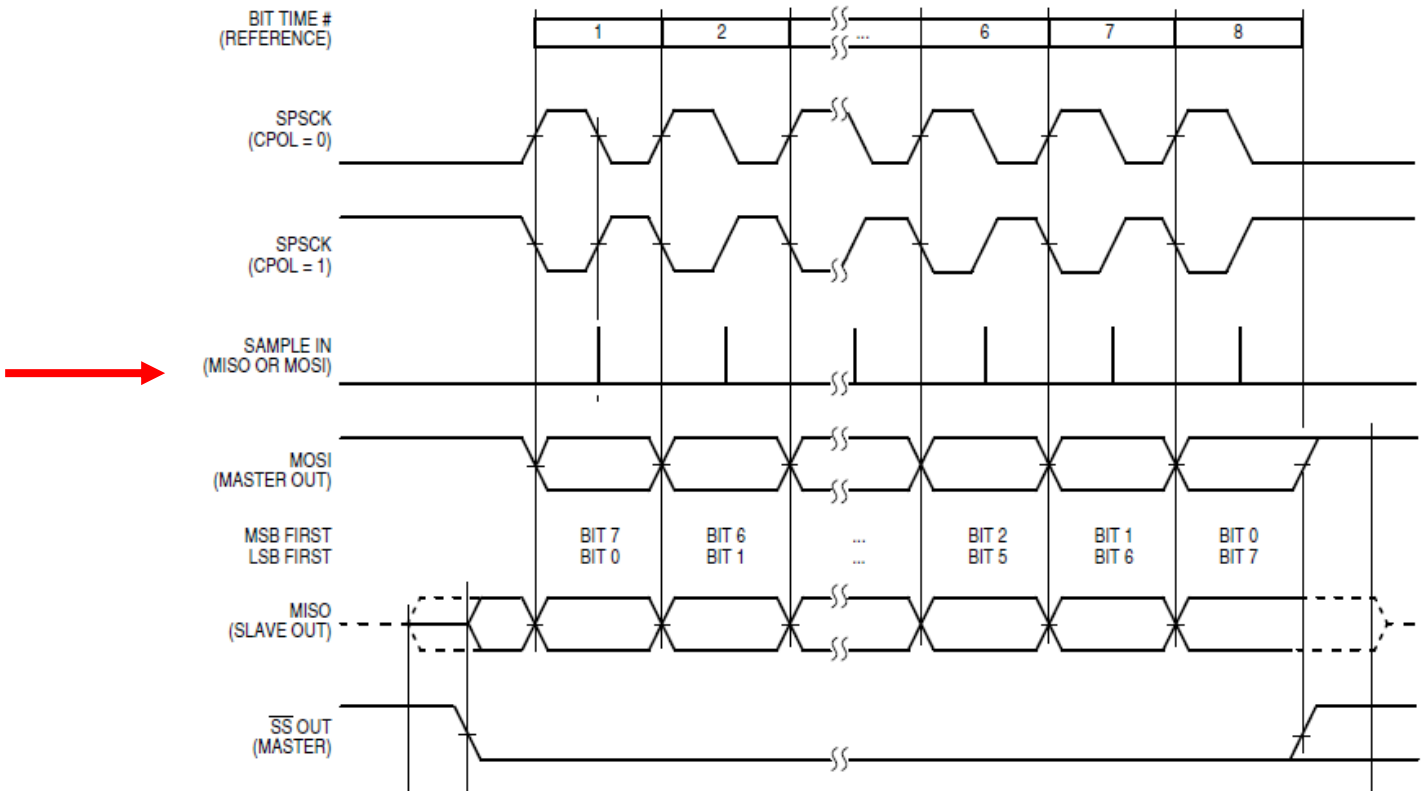
Esquema de interconexión y Funcionamiento:



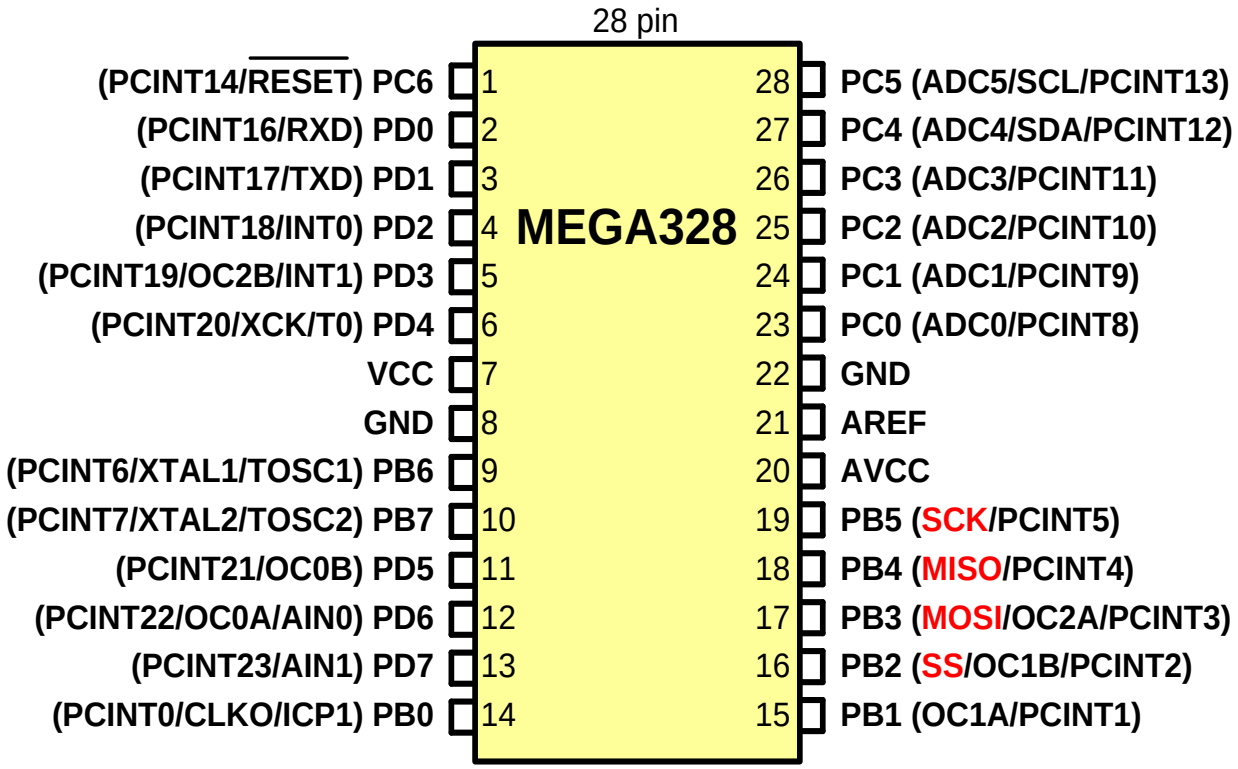
- Básicamente son 2 registros de desplazamientos de 8 bits interconectados
- Cuando se realiza una transferencia de datos, con cada pulso de reloj los datos son "intercambiados" entre ambos registros de desplazamiento.
- Los datos son actualizados a la salida de cada registro de desplazamiento en el flanco de reloj opuesto al que son ingresados al mismo registro.
- Los pulsos de reloj pueden ser programados de manera que la transmisión del bit se realice en 4 modos diferentes, a estos bits de control se llama polaridad (CPOL) y fase de la transmisión (CHPA)
- CPOL controla el nivel del reloj en el estado IDLE del protocolo
- CHPA controla en que instante se transfiere o reciben los bits

SPI (Serial Peripheral Interface)

- Con CPHA=0 el dispositivo muestrea la entrada en el 1er, 3er, 5to… flanco de reloj
- Con CPHA=1 el dispositivo muestrea la entrada en el 2do, 4to, 6to… flanco de reloj
- Veamos un Ejemplo de transferencia con CPHA=1

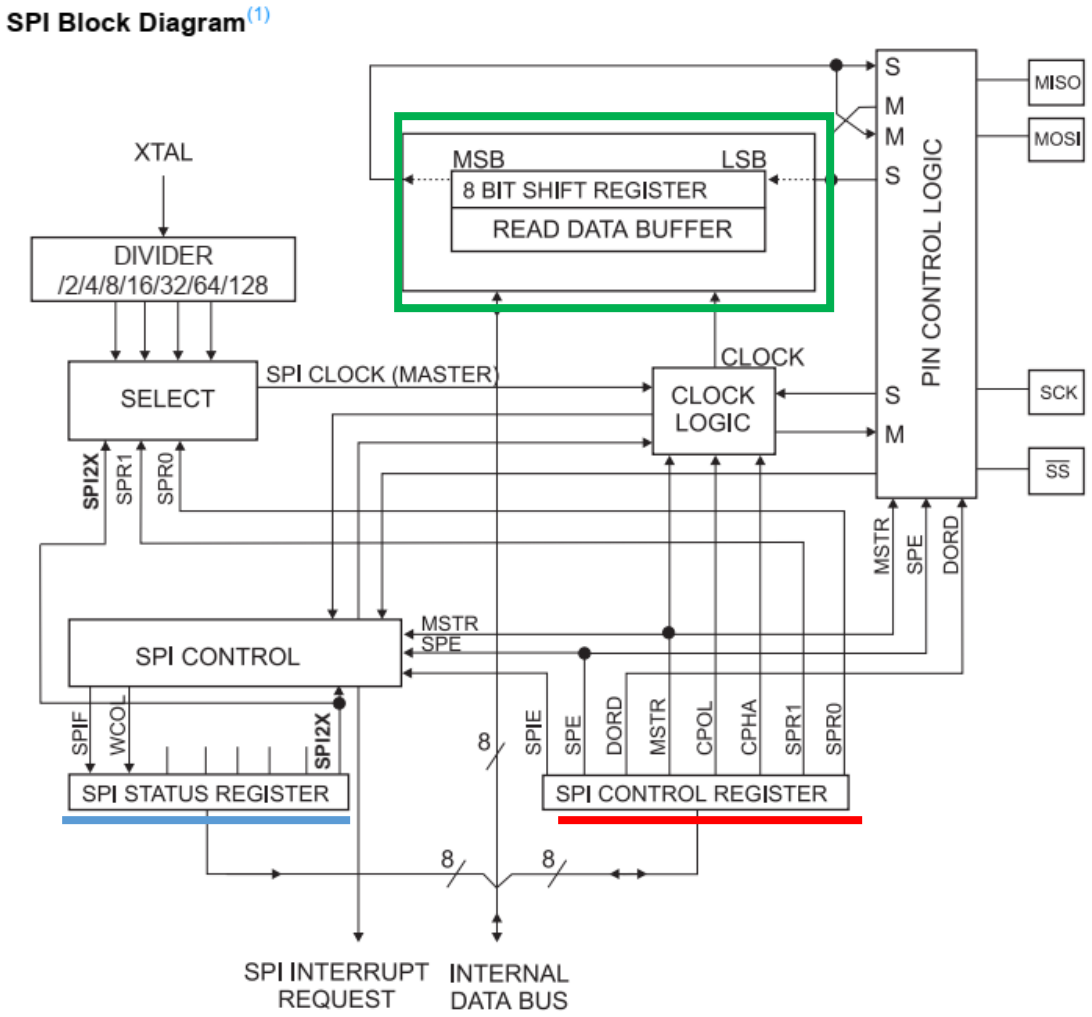


SPI en el Atmega 328p



SPI en el Atmega 328p

- Control register:
  - SPCR (SPI Control Register)
- Status Register:
  - SPSR (SPI Status Register)
- Data Register:
  - SPDR (SPI Data Register)



Source: CH 19 Atmega328p Data Sheet

SPSR (SPI Status Register)

- SPIF (SPI Interrupt Flag)
  - La transferecia del dato fue completada (transmit & receive)
- WCOL (Write Collision)
  - El bit WCOL se activa cuando el SPI Data Register (SPDR) se escribe durante una transferencia en curso.
- SPI2X (Double SPI Speed)
  - Cualdo se setea a 1 la frecuencia del reloj (SCK Frequency) se duplica (solo en Master mode)

SPCR:

SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

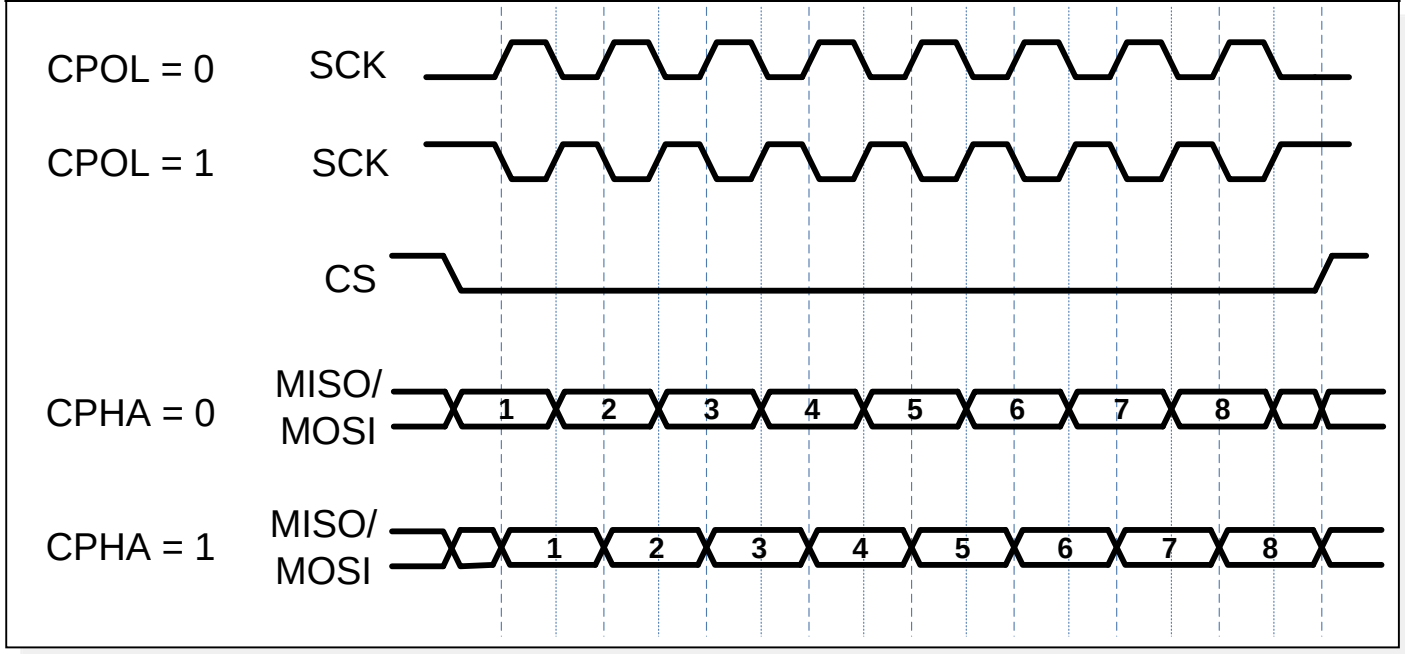
- SPIE (SPI Interrupt Enable)
- SPE (SPI Enable)
- DORD (Data Order)
- MSTR (Master)
- CPOL (Clock Polarity)
- CPHA (Clock Phase)
- SPR1, SPR0 :SPI Clock Rate

SPI2X	SPR1	SPR0	SCK Freq.
0	0	0	Fosc/4
0	0	1	Fosc/16
0	1	0	Fosc/64
0	1	1	Fosc/128
1	0	0	Fosc/2 (not recommended)
1	0	1	Fosc/8
1	1	0	Fosc/32
1	1	1	Fosc/64

SPCR:

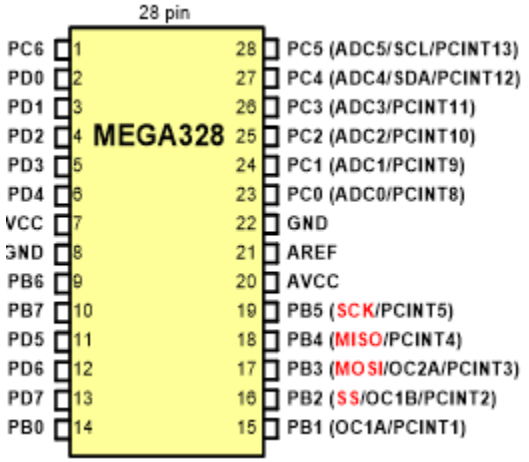
SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

CPOL	CPHA	Data Read and Change Time	SPI Mode
0	0	Read on rising edge, changed on a falling edge	0
0	1	Read on falling edge, changed on a rising edge	1
1	0	Read on falling edge, changed on a rising edge	2
1	1	Read on rising edge, changed on a falling edge	3



Programa 1: Enviar ‘G’ por SPI como master

```
#include <avr/io.h>

#define MOSI 3
#define SCK 5
#define SS 2

int main (void)
{
 DDRB = (1<<MOSI)|(1<<SCK)|(1<<SS); //MOSI and SCK are output
 DDRD = 0xFF; //Port D is output
 SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0); //enable SPI as master, SCLK fosc/16, modo 0
 while(1) {
  PORTB &= ~(1<<SS); //enable slave device
  SPDR = 'G'; //start transmission
  while(!(SPSR & (1<<SPIF))); //wait transfer finish
  PORTD = SPDR; //move received data to PORTD
  PORTB |= (1<<SS); //disable slave device
}
 return 0;
}
```



Cuando el SPI es master
Debemos controlar PB2 para activar al dispositivo slave

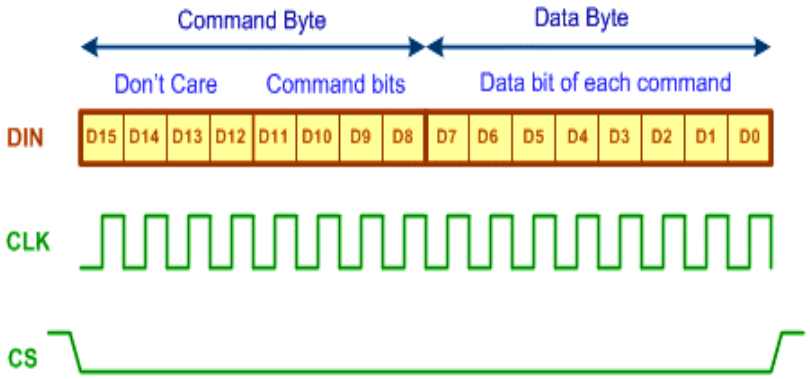
Programa 2: Enviar ‘G’ por SPI como slave

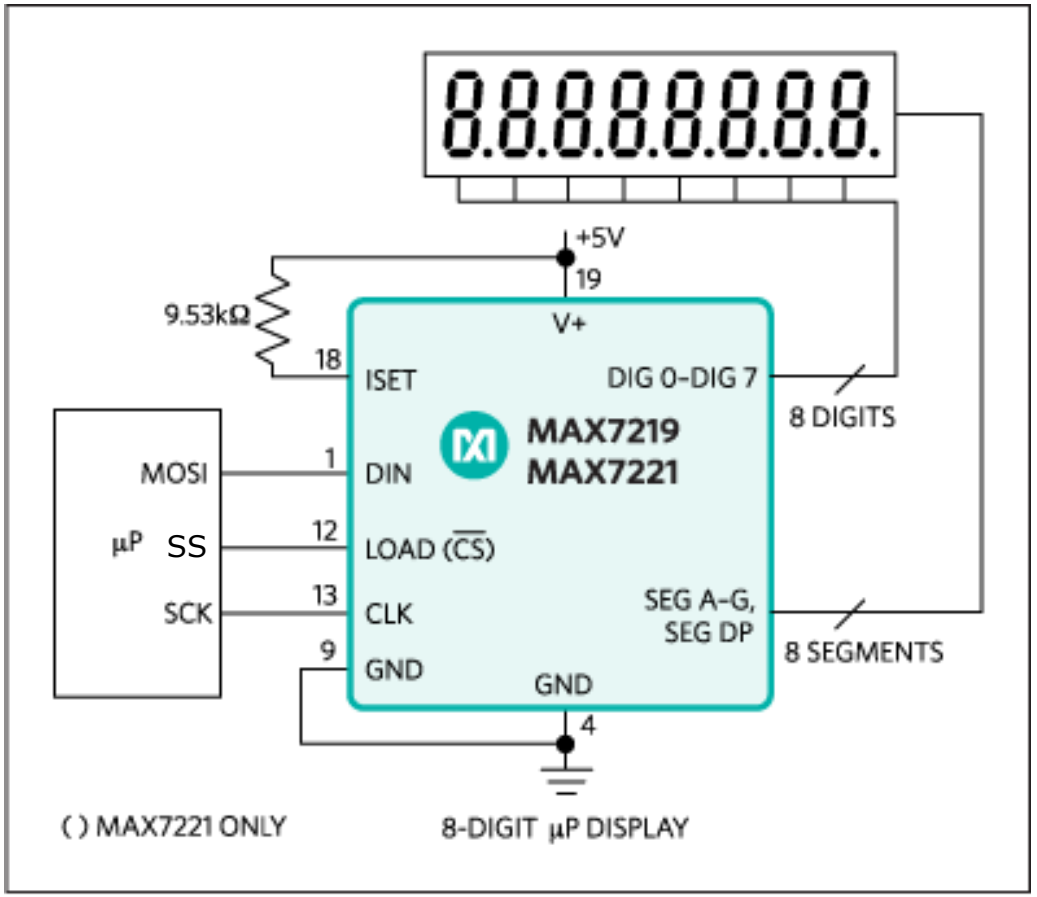
```
#include <avr/io.h>

#define MISO 4

int main (void)
{
 DDRD = 0xFF; //Port D is output
 DDRB = (1<<MISO); //MISO is output
 SPCR = (1<<SPE); //enable SPI as slave, modo 0
 while(1)
 {
  SPDR = 'G';
  while(!(SPSR &(1<<SPIF,))); //wait for transfer finish
  PORTD = SPDR; //move received data to PORTD
 }
 return 0;
}
```

Cuando el SPI es slave
SS es input. Si SS=1 todos los terminales son entradas Hi-z.
Si SS=0 slave envía datos según indica el SCLK del master

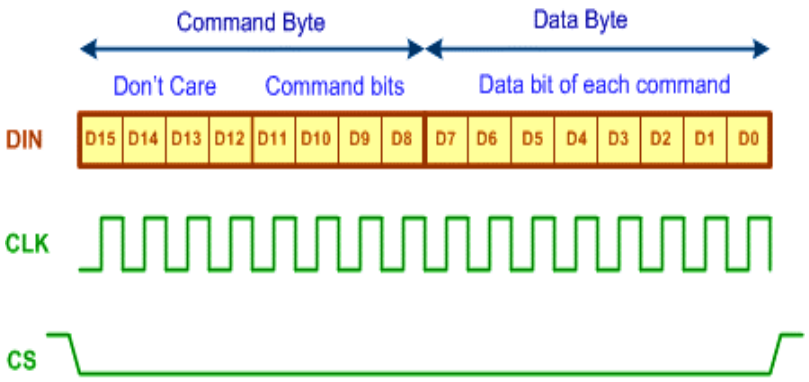
Programa 3: transferencia de 16 bits como master

- Es el caso del controlador de LEDS MAX7221 (slave)





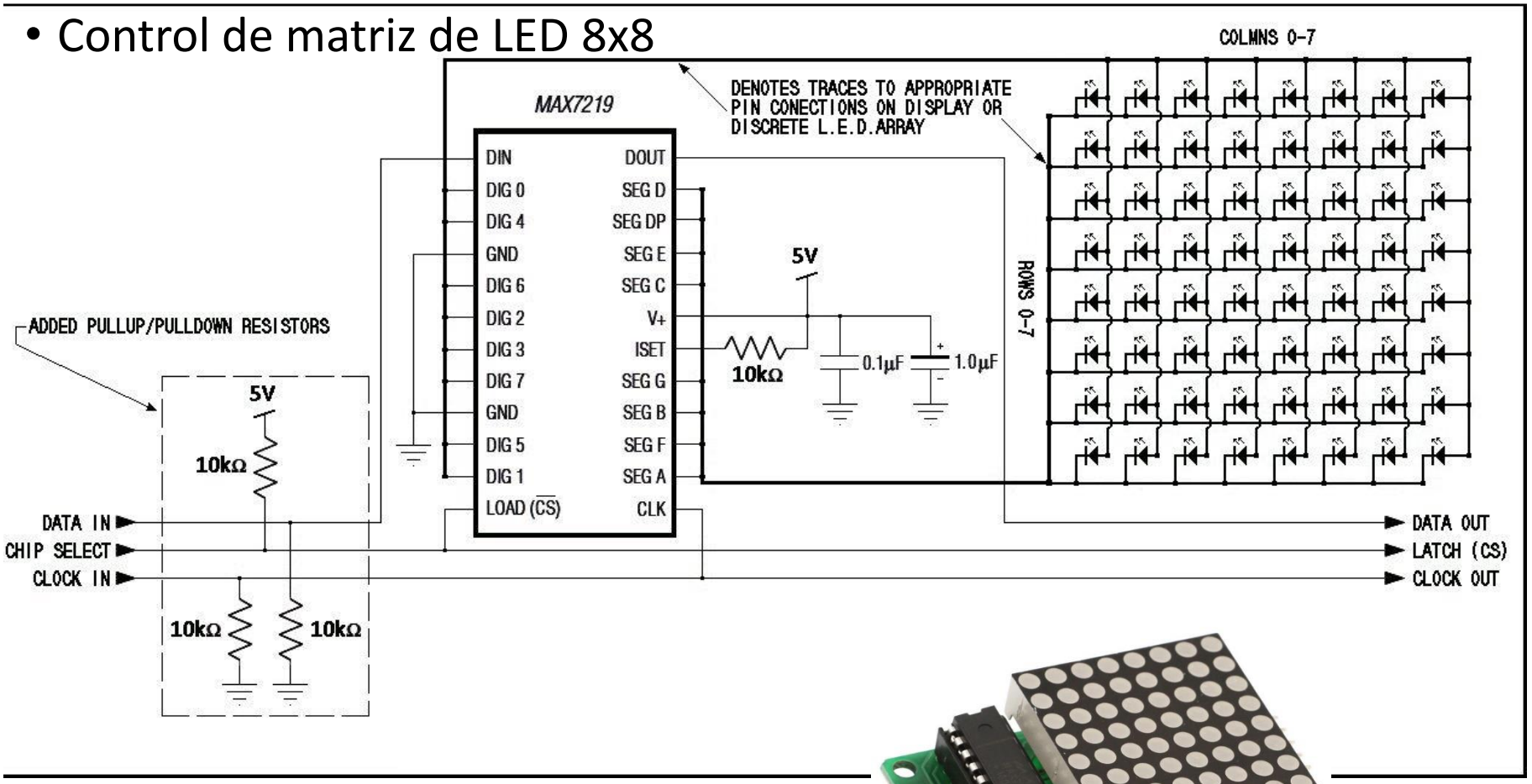
Programa 3: transferencia de 16 bits como master




```
#include <avr/io.h>

#define MOSI 3
#define SCK 5
#define SS 2

int main (void)
{
 DDRB = (1<<MOSI)|(1<<SCK)|(1<<SS); //MOSI SCK and SS are output
 SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0); //enable SPI as master, SCLK fosc/16
 while(1) //do for ever
 {
  PORTB &= ~(1<<SS); //enable slave device
  SPDR = 'A'; //start transmission first byte
  while(!(SPSR & (1<<SPIF))); //wait transfer finish
  SPDR = 'B'; //start transmission second byte
  while(!(SPSR & (1<<SPIF))); //wait transfer finish
  PORTB |= (1<<SS); //disable slave device
 }
 return 0;
}
```

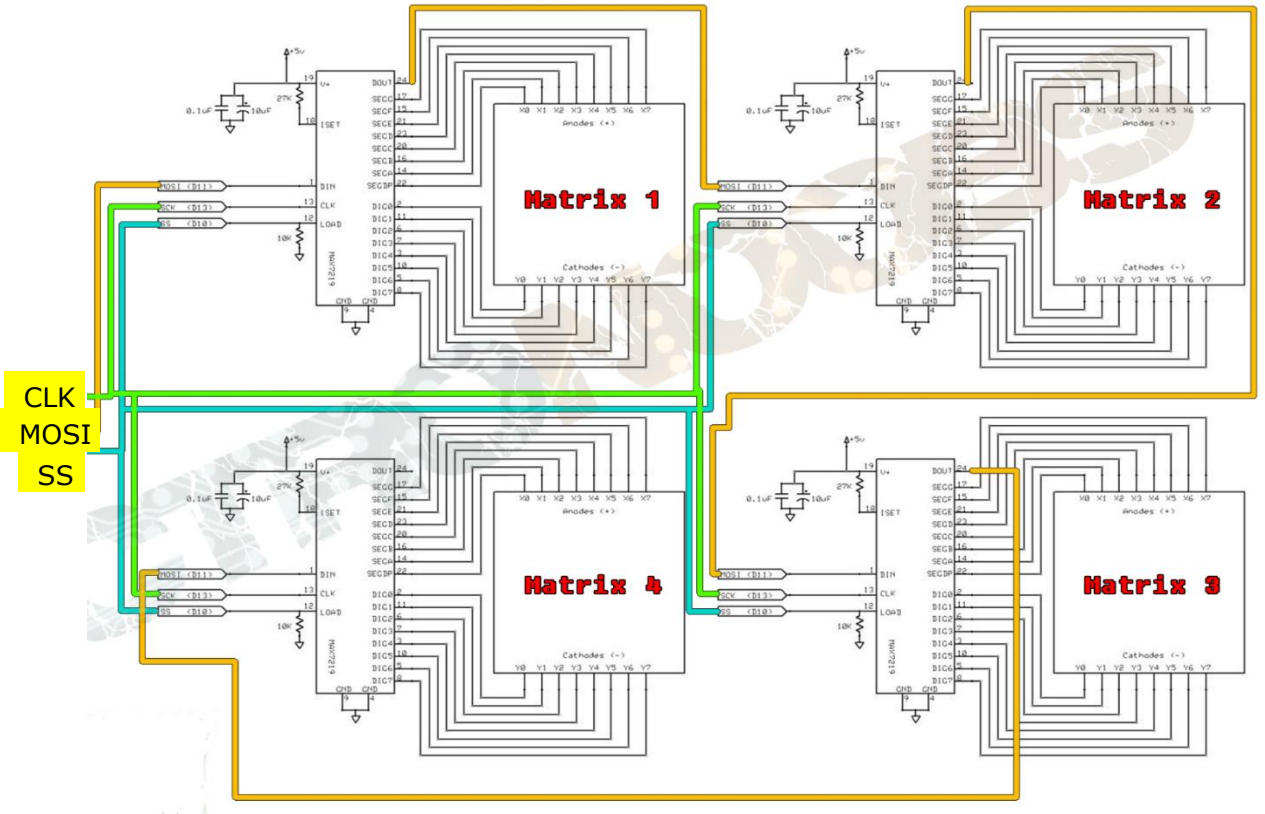
Ejemplo de aplicación del MAX7221

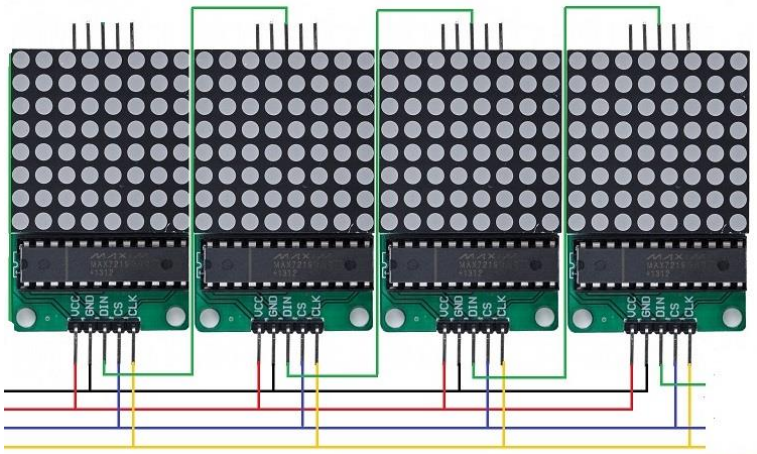
- Control de matriz de LED 8x8



Ejemplo de aplicación del MAX7221

- Control de 4 matrices de LEDS https://www.youtube.com/watch?v=6GRTEznTWcs



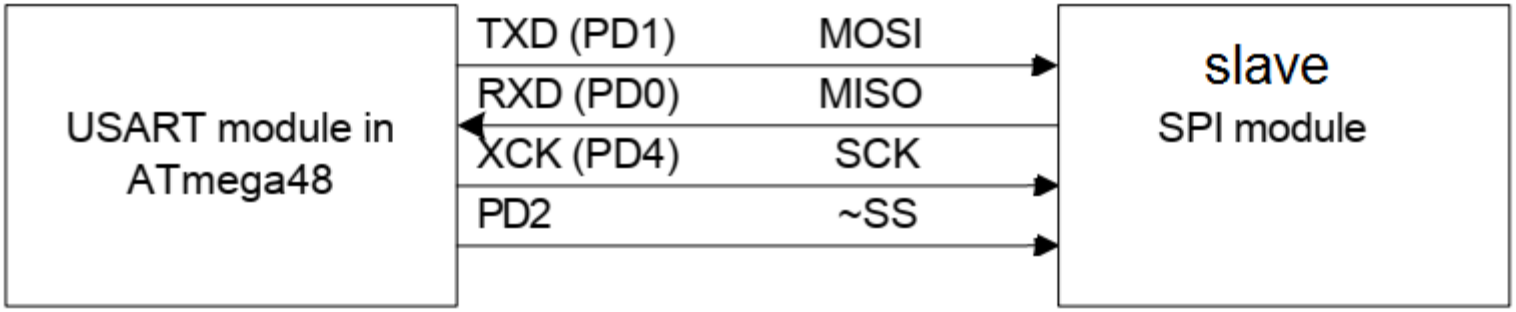


SPI (Serial Peripheral Interface)

- Ventajas
  - Protocolo flexible
    - No está limitado a la transferencia de bloques de 8 bits, ni al orden de transmisión de los bits
    - Libre elección del tamaño de la trama de bits, de su significado y su propósito
    - Permite seleccionar cuatro modos de operación (polaridad-fase)
  - Su implementación en hardware es simple
    - No es necesario arbitraje o mecanismos de respuesta ante fallos
    - Los dispositivos *esclavos* usan el reloj que envía el *maestro*, no necesitan por tanto su propio reloj y la frecuencia de este puede configurarse según la aplicación.
    - un dispositivo puede utilizarse para solo transmitir, sólo recibir o ambas cosas a la vez.
  - Existen versiones Dual SPI, Quad SPI, QPI para aumentar la eficiencia de la transferencia
- Desventajas
  - Require más terminales de interconexión que I^2C (TWI)
  - El direccionamiento se hace mediante líneas de selección de chip (una por cada slave)
  - No posee control de flujo por hardware
  - No posee mecanismos de reconocimiento (el maestro podría estar enviando información sin que estuviesen conectados los esclavos).
  - No permite tener más de un maestro conectado al bus.

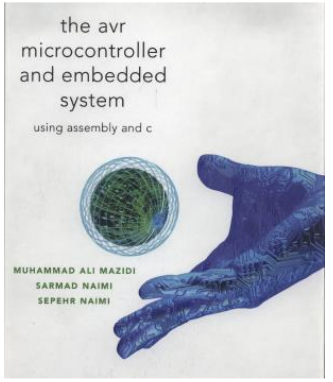
USART0 en modo SPI

- El periférico serie USART0 del Atmega 328 puede utilizarse en modo SPI solo Master



- Esto es posible gracias a la capacidad del hardware de realizar transferencias tanto asincrónicas como sincrónicas.
- Los detalles y ejemplos de como configurar la USART0 en modo SPI están en el capítulo 21 de la hoja de datos.

Bibliografía:



- *The AVR microcontroller & Embedded Systems*. Mazidi,Naimis, CH17
- Libro Digital de Felipe Espinosa: CH6.2 y CH6.3
- Hoja de datos: ATMEGA, MAX7219-MAX7221
- Wiki: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface
- Videos ARDUINO:
  - Lector de tarjeta SD
  - Pantalla gráfica TFT