Protocolos I2C y SPI

Arquitectura 2023

Introducción

¿Porque estudiar estos protocolos de comunicación?

- Ideales para **Sistemas Embebidos**
- **Simples de implementar**
- **Requieren mínima** cantidad de **líneas de datos**
- **Facilitan la comunicación** entre **dispositivos** (memorias, Microcontroladores, sensores, etc)

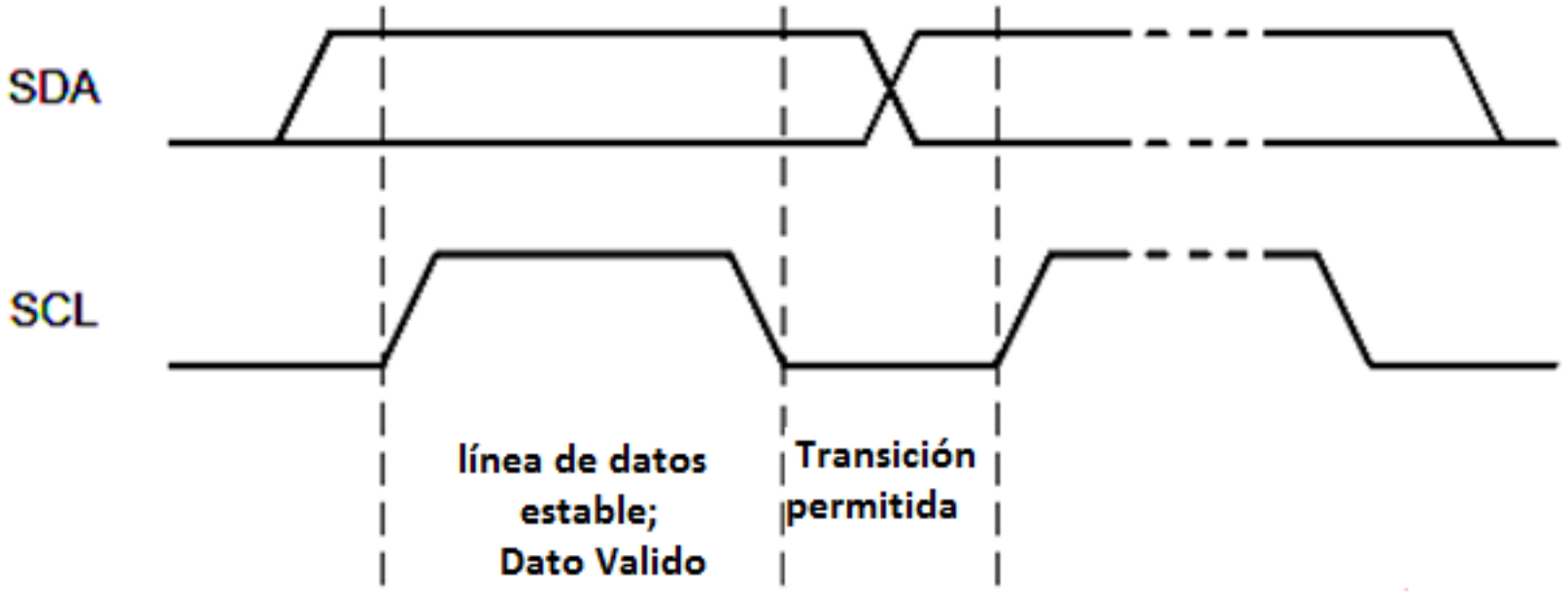
Introducción

- ¿Dónde se utilizan más frecuentemente?
  - Protocolo SPI e I^2C
    - **Comunicación entre Microcontroladores**
    - Entre un **Microcontrolador y un periférico** (memoria Flash, expansores, acelerómetro, tarjetas SD, etc)
    - Los protocolos serie sincrónicos se utilizan para interconexiones en **pequeñas distancias**

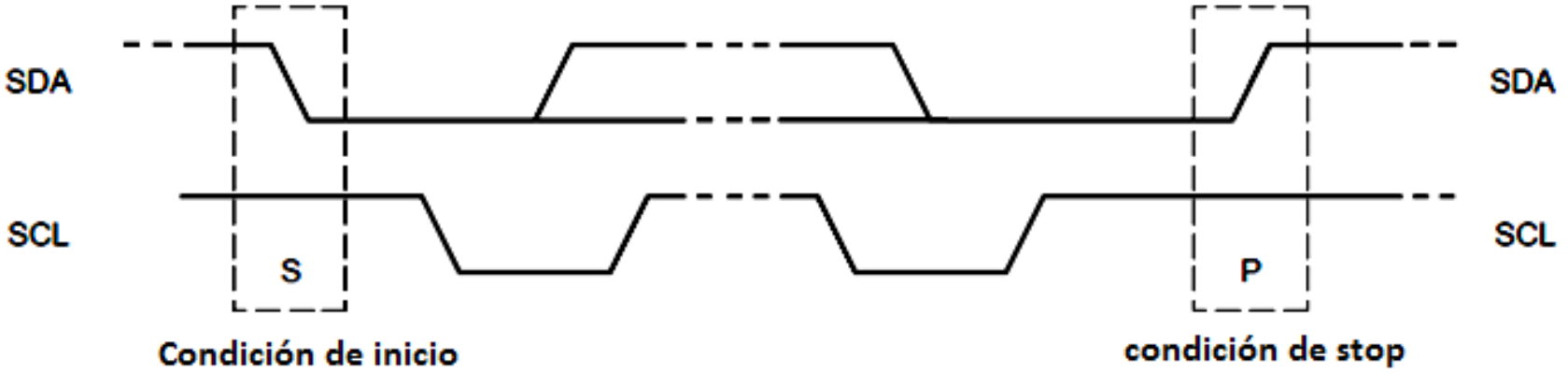
Descripción I^2C

- **2 Líneas de comunicación SCK y SDA conectadas a pull-up**
  - **SDA- línea de entrada salida de Datos**
  - **SCK – Línea de clock de sincronismo**
- **Cada dispositivo se reconoce por una dirección y puede operar como Tx o RX**
- **El master inicia la transferencia de datos en el bus y genera la señal de clock**
- **I^2C permite múltiples masters**

I²C: Transferencia de un bit

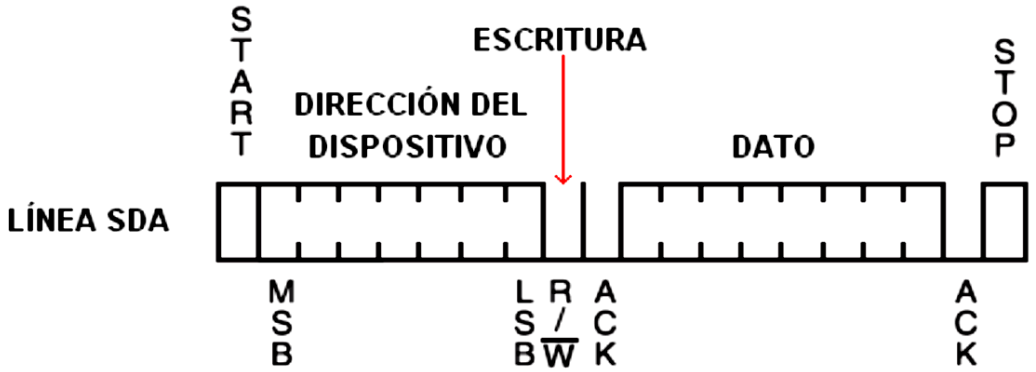


I²C: Inicio y fin de una transferencia

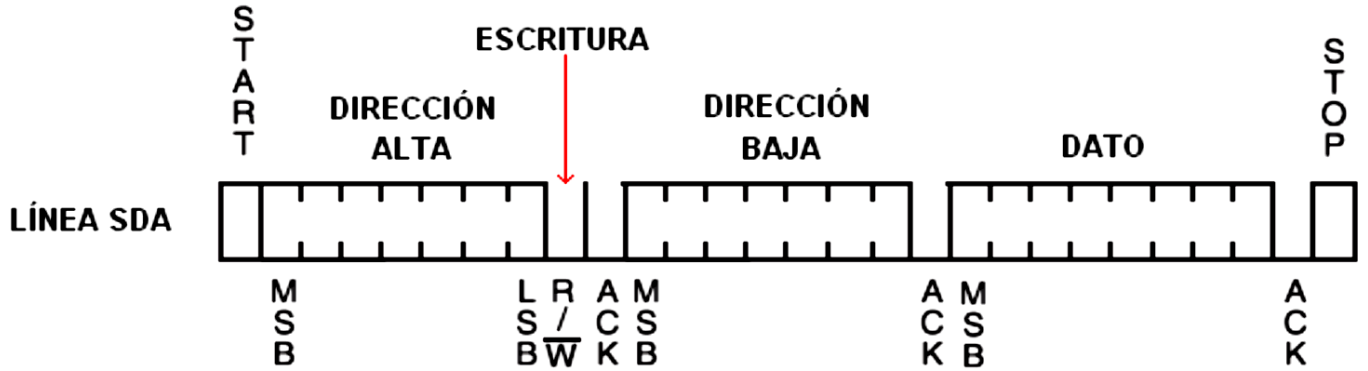


I2C: Transferencia de una trama

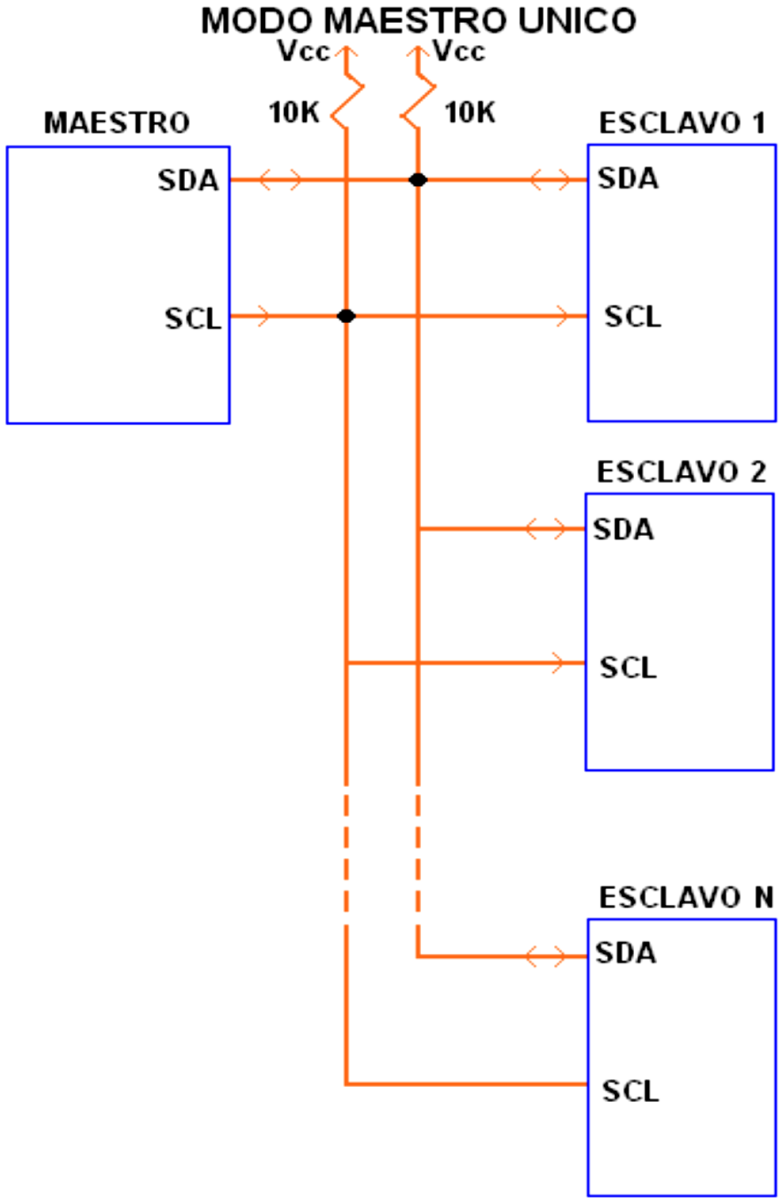
CON DIRECCION EN 8 BITS



CON DIRECCIÓN EN 10 BITS



I2C: Conexión maestro-esclavo



Librería para el Módulo I2C

Los pines I2C se describen como:

#define TW_SCL_PIN PORTC5

#define TW_SDA_PIN PORTC4

Hay 3 funciones disponibles:

Void **tw_init** (twi_freq_mode_t **twi_freq**, bool **pullup_en**);

Inicialice I2C con una frecuencia predefinida y habilite las resistencias pull-up internas. Hay 3 modos de frecuencia para elegir: TW_FREQ_100K, TW_FREQ_250K y TW_FREQ_400K. Entonces seleccione true en pullup_en para habilitar el pull-up interno en los pines SCL y SDA, o seleccione false si usa pull-up externos

ret_code_t **tw_master_transmit** (uint8_t **slave_addr**, uint8_t* **p_data**, uint8_t len, bool **repeat_start**);

Esta función transmite bytes de datos a una dirección esclava deseada. Si el bit repeat_start se configura true, el módulo I2C enviará la condición de START REPETIDA en lugar de la condición de STOP. Esto es útil en un entorno multimaestro donde un maestro envía un comando y espera la respuesta del esclavo sin tener que liberar el bus.

ret_code_t **tw_master_receive** (uint8_t **slave_addr**, uint8_t* **p_data**, uint8_t **len**);

Esta función recibe los bytes de datos de la dirección esclava deseada y devuelve ret_code_t para el manejo de errores o estado de la transmisión (envío correcto, confirmación del esclavo), etc.

Inicialización I2C

```
void tw_init(twi_freq_mode_t twi_freq_mode, bool pullup_en)
{  DDRC  |= (1 << TW_SDA_PIN) | (1 << TW_SCL_PIN);
if (pullup_en)
    {  PORTC |= (1 << TW_SDA_PIN) | (1 << TW_SCL_PIN);
    }
else
    {
       PORTC &= ~((1 << TW_SDA_PIN) | (1 << TW_SCL_PIN));
    }
 DDRC  &= ~((1 << TW_SDA_PIN) | (1 << TW_SCL_PIN));

  switch (twi_freq_mode)
     {
         case TW_FREQ_100K:
/* registro de tasa de bit en 72 y preescaler a 1 obteniendo SCL_freq = 16MHz/(16 + 2*72*1) = 100KHz                */
            TWBR = 72;
             break;

         case TW_FREQ_400K:
/* registro de tasa de bit en 12 y preescaler a 1 obteniendo SCL_freq = 16MHz/(16 + 2*12*1) = 400KHz                */
             TWBR = 12;
              break;

          default: break;
        }  // end switch
} //end tw_init
```

Funciones de transmisión

Desde **tw_master_transmit** se llama a:

```
tw_start();
tw_write_sla(TW_SLA_W(slave_addr));
tw_write(p_data[i]);
tw_stop();
```

Ejemplo de uso I2C

```
#include "twi_master.h"
#include <avr/io.h>
#include <util/delay.h>
#define ADDR            0x68
int main(void)
        {tw_init(TW_FREQ_400K, true);
          // set I2C Frequency, enable internal pull-up
error_code = tw_master_transmit(ADDR, data, sizeof(data), false);
error_code = tw_master_receive(ADDR, data, sizeof(data));
        }
```

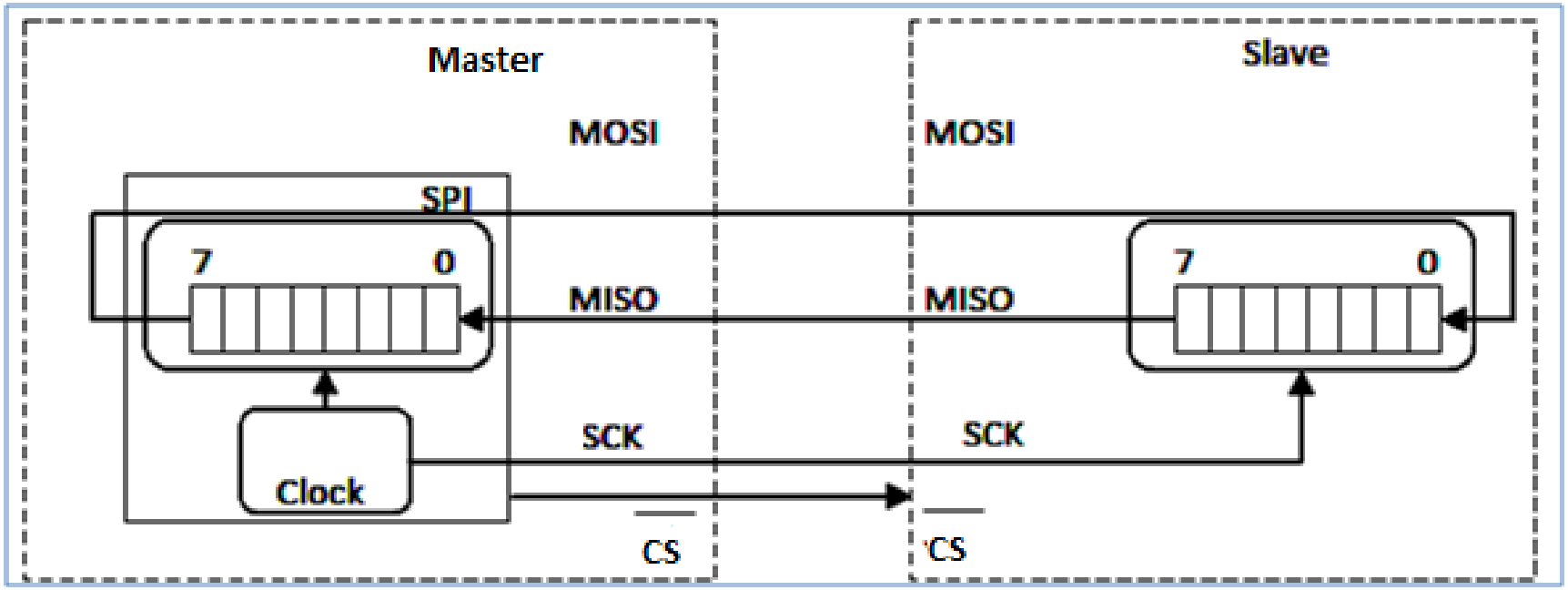
Descripción SPI

Serial Peripheral interface

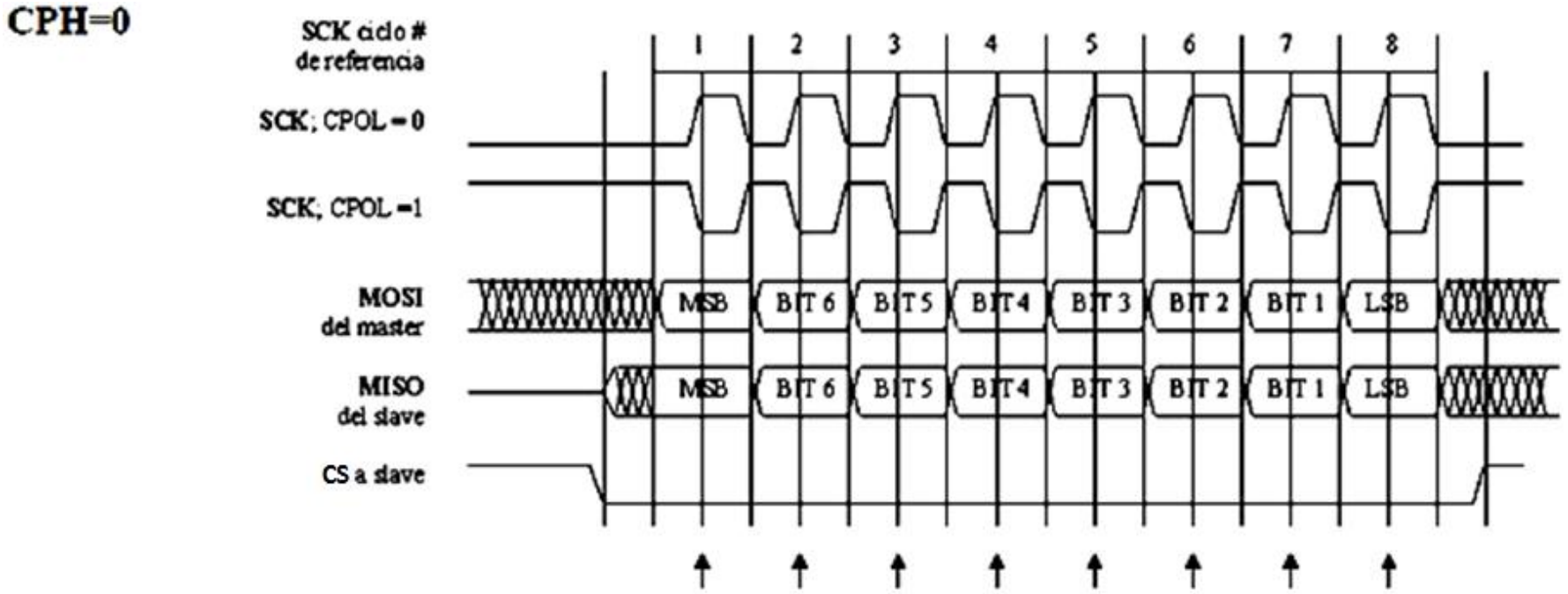
- Interfaz sincrónica maestro-esclavo
- El master genera una señal de clock
- Requiere una línea MISO, una MOSI, una de Clock y uno o varios CS
- Un maestro y uno o varios esclavos
- Un dispositivo maestro puede **intercambiar su rol** y pasar a ser esclavo.

SPI: Diagrama de conexión y modo de transmisión

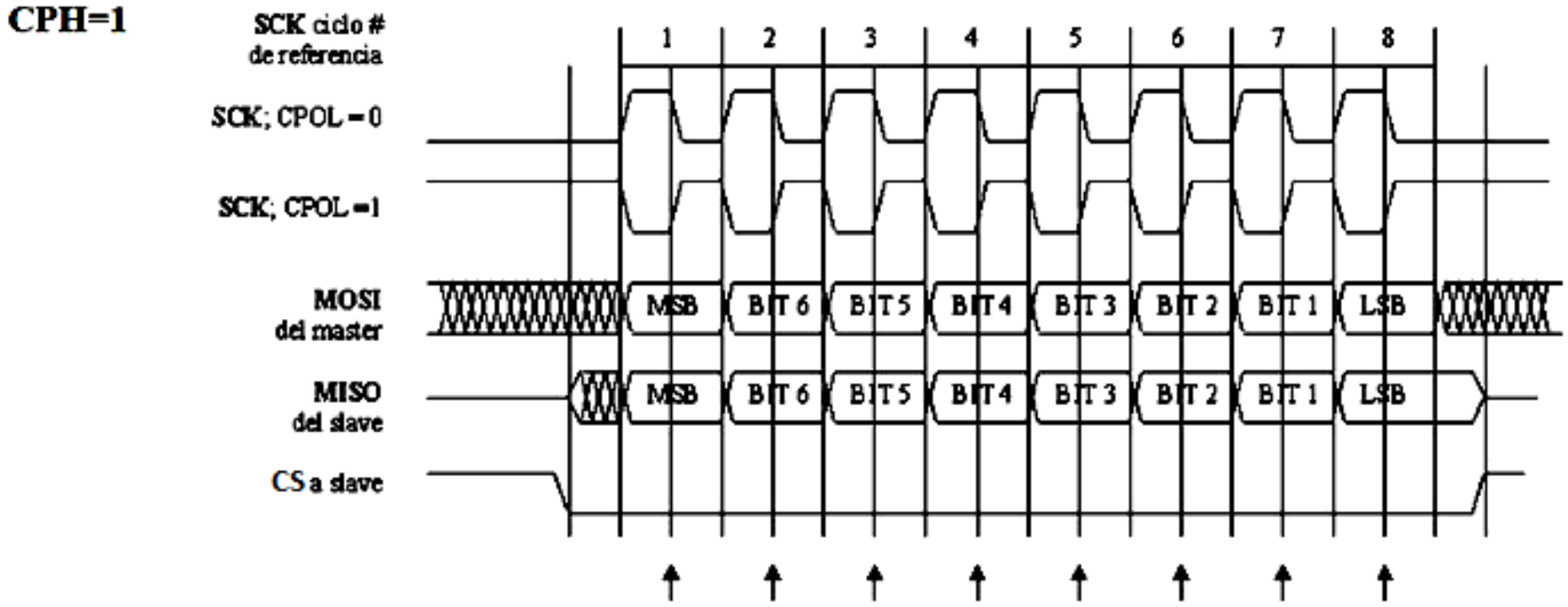
-El modo de transmisión es Full-duplex



SPI: Formato de Transmisión (Fase =0)



SPI: Formato de Transmisión (Fase =1)



Configuración del protocolo SPI por SW

```
#include <avr/ió.h>
.text
 .global sw_spi
.equ spi_port , 0x18 ;PORTB            //Se definen los pines y puertos, usando PB0,1,2
.equ mosi , 0 ;pin PB2
 .equ sck , 1 ;pin PB0
 .equ cs_n , 2 ;pin PB1
sw_spi:                //r18 es un contador que inicia en 8, r24 contiene el byte data recibido como parámetro
        ldi r18,0x08                ;inicia el contador para 8 ciclos de reloj
         cbi spi_port, cs_n          ;setea chip select en bajo
loop:    rol r24                     ;desplaza el byte a izquierda(MSB first); se setea el carry si el bit 7 es uno
          brcc bit_low               ;si el carry es cero, significa que el bit es cero
          sbi spi_port, mosi          ;setea el pin de datos en alto "1"
          rjmp clock                  ;salto al generador de ciclo de reloj
bit_low: cbi spi_port, mosi           ;setea el pin de datos en bajo "0"
clock:    sbi spi_port, sck           ;sck -> uno
          cbi spi_port, sck           ;sck -> cero
          dec r18                     ;decremento el contador
          brne loop                   ;salta a loop si no terminó
          sbi spi_port, cs_n          ;al finalizar el envío setea el chip select en alto
          ret ;
.end
```

Ejemplo con módulo SPI

```
#define F_CPU 16000000UL //16Mhz clock
#include <avr/io.h>
#include <util/delay.h>
//declaración de la rutina SPI escrita en assembler
extern void sw_spi(uint8_t data);
 int main(void)
    {
        DDRB = 0x07; //setea el puerto B bit 1,2,3 como salida
        while(1){
                    sw_spi(0xA5); //alterna los valores a enviar por el puerto spi
                    sw_spi(0x5A);
                   } //fin while
    }    //fin main
```

Referencias

- https://github.com/Sovichea/avr-i2c-library
- Douglas H. Summerville, "Embedded Systems Interfacing for Engineers using the Freescale HCS08 Microcontroller II: Digital and Analog Hardware Interfacing", State University of New York at Binghamton, Morgan y Claypool Publishers, 2009.
- Jan Axelson, "Serial Port Complete", tercera edición, Madison, Lakeview research, 2007.