Serial peripheral interface (SPI)

Desarrollada por Motorola para proveer comunicación serial síncrona full-duplex entre controladores y periféricos.

Comúnmente utilizada para comunicación con memorias flash, sensores, real-time clocks (RTCs), convertidores análogo-digital, entre otros.

SPI

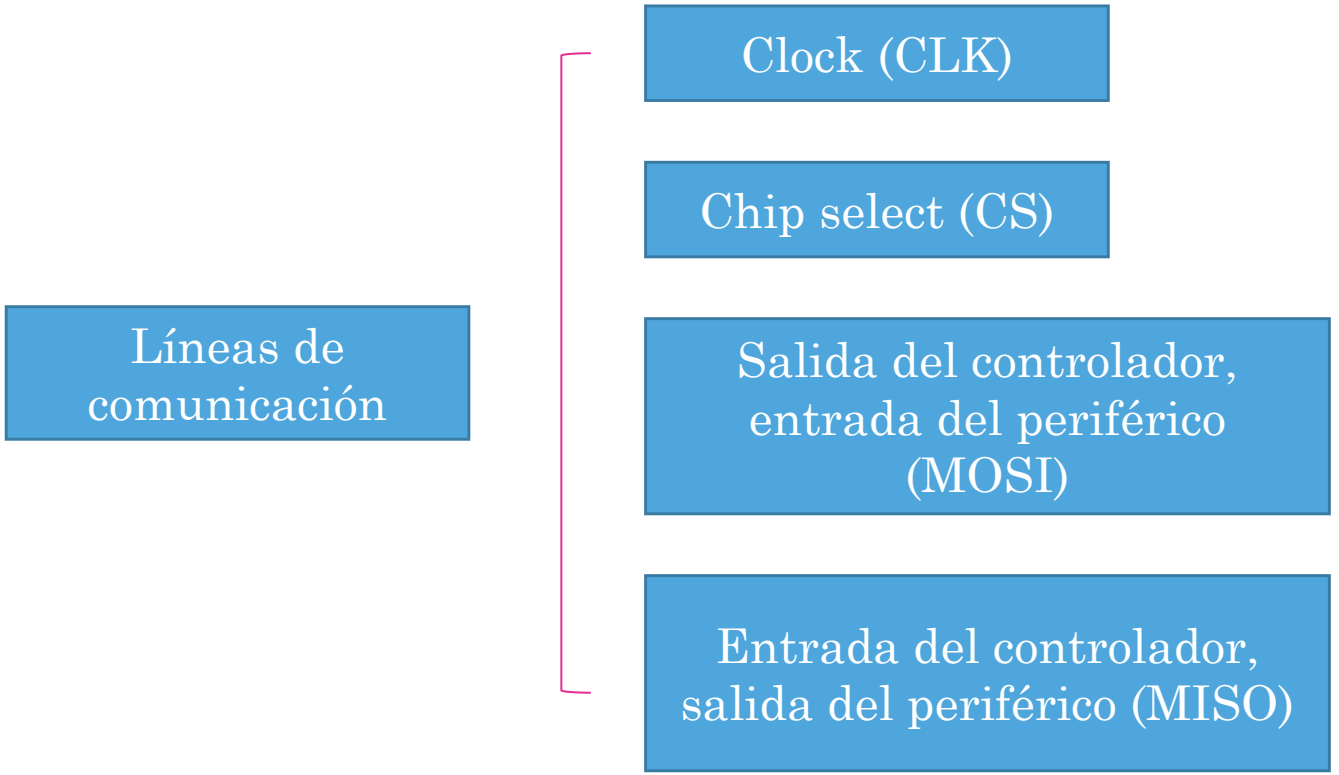
Los datos del controlador y periférico son sincronizados en el flanco ascendente o descendente del reloj.

El controlador y el periférico pueden transmitir datos al mismo tiempo.

La interfaz SPI puede ser de 3 o 4 líneas.

Interfaz de cuatro líneas

Interfaz de cuatro líneas



Interfaz de cuatro líneas

El controlador genera la señal de reloj

Dispositivos SPI soportan frecuencias de reloj más altas comparados con interfaces I2C.

Interfaces SPI solo pueden tener un controlador y uno o más periféricos.

La señal CS es utilizada para seleccionar el periférico. Normalmente se activa en bajo y se pone en alto para desconectar el periférico.

Se requiere una línea individual de CS por cada periférico conectado al controlador.

Interfaz de cuatro líneas

Para iniciar la comunicación, el controlador envía la señal de reloj y selecciona el periférico por medio de CS.

Debido a que SPI es full-duplex, tanto el controlador como el periférico pueden enviar datos al mismo tiempo por medio de MISO y MOSI.

El flanco del reloj serial sincroniza el envío y muestreo de los datos.

SPI permite al usuario seleccionar el flanco ascendente o descendente para el envío o muestreo de los datos.

Polaridad y fase del reloj

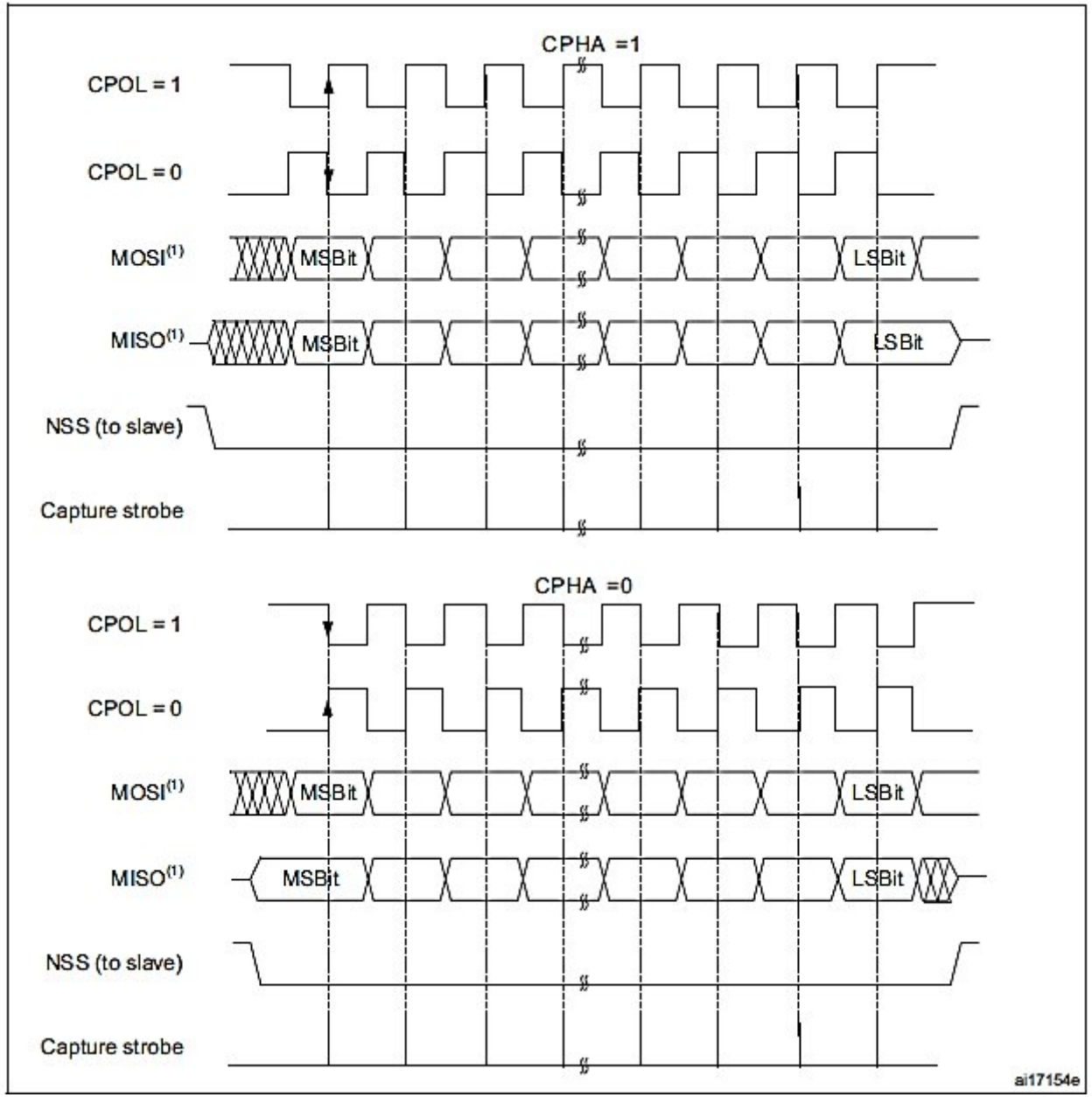
El controlador selecciona la polaridad y fase del reloj dependiendo los requerimientos del periférico.

El bit **CPHA** selecciona la fase del reloj. Decide en cual flanco de reloj (1ro o 2do) el periférico debe muestrear el dato.

El bit **CPOL** configura la polaridad del reloj durante el estado de inactividad.

El estado de inactividad es el periodo en que ~CS está en alto y pasa a bajo en el inicio de la transmisión. Y cuando ~CS está en bajo y pasa a alto al final de la transmisión.

Polaridad y fase del reloj



Polaridad y fase del reloj

Si **CPHA** = 1, el segundo flanco del reloj captura el primer bit de datos transmitido (en el flanco descendente si **CPOL** = 0, o en el flanco ascendente si **CPOl** = 1).

Si **CPHA** = 0, el primer flanco del reloj captura el primer bit de datos transmitido (en el flanco descendente si **CPOL** = 1, o en el flanco ascendente si **CPOl** = 0).

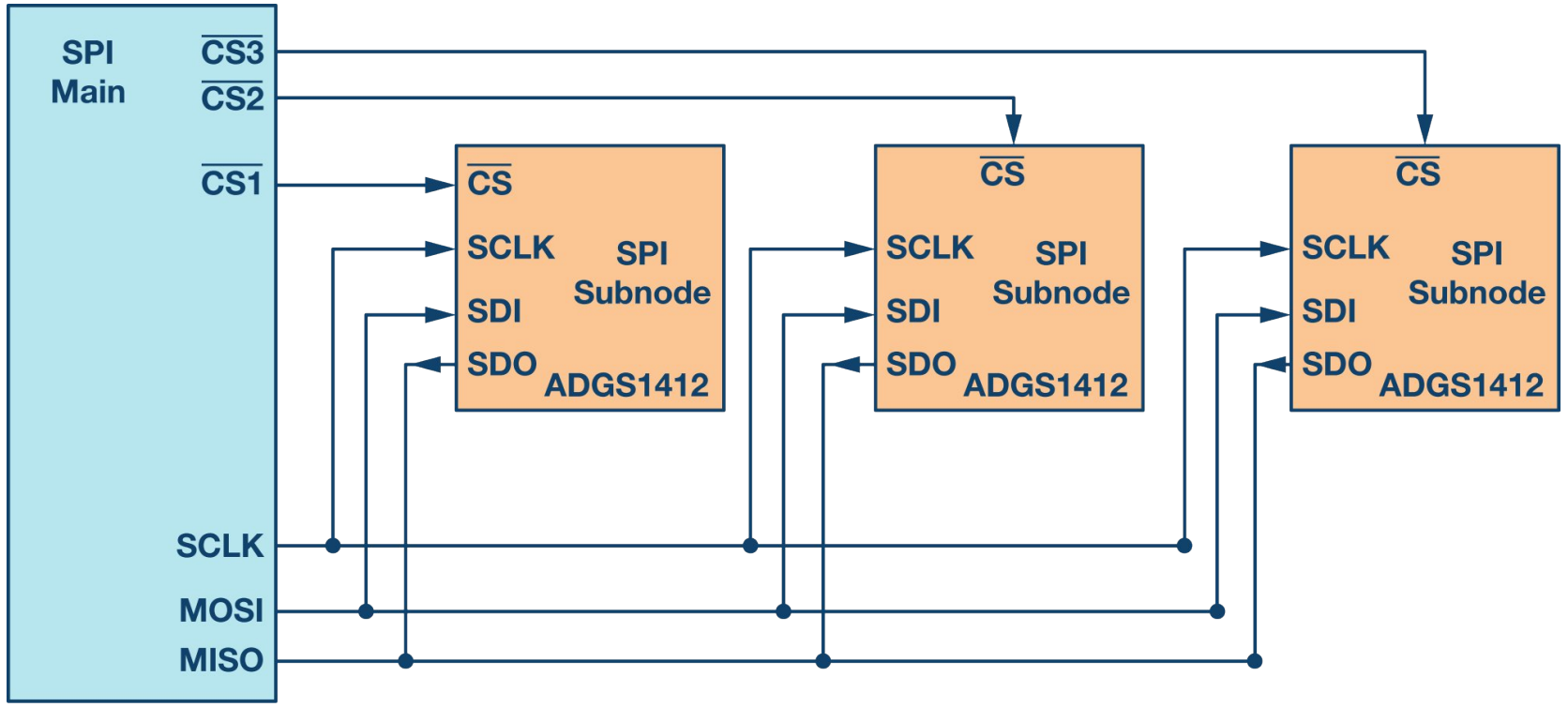
Por default, **CPHA**=0 y **CPOL**=0.

Configuración con múltiples periféricos

Múltiples periféricos se pueden conectar a un controlador.

Los periféricos pueden conectarse en modo regular o en daisy-chain.

Modo regular



Modo daisy-chain

Los periféricos están configurados de manera que la misma línea ~CS está conectada a todos los periféricos y los datos se propagan de un periféricos al siguiente.

Los datos del controlador son conectados directamente al primer periférico y éste provee datos al siguiente periféricos y así sucesivamente.

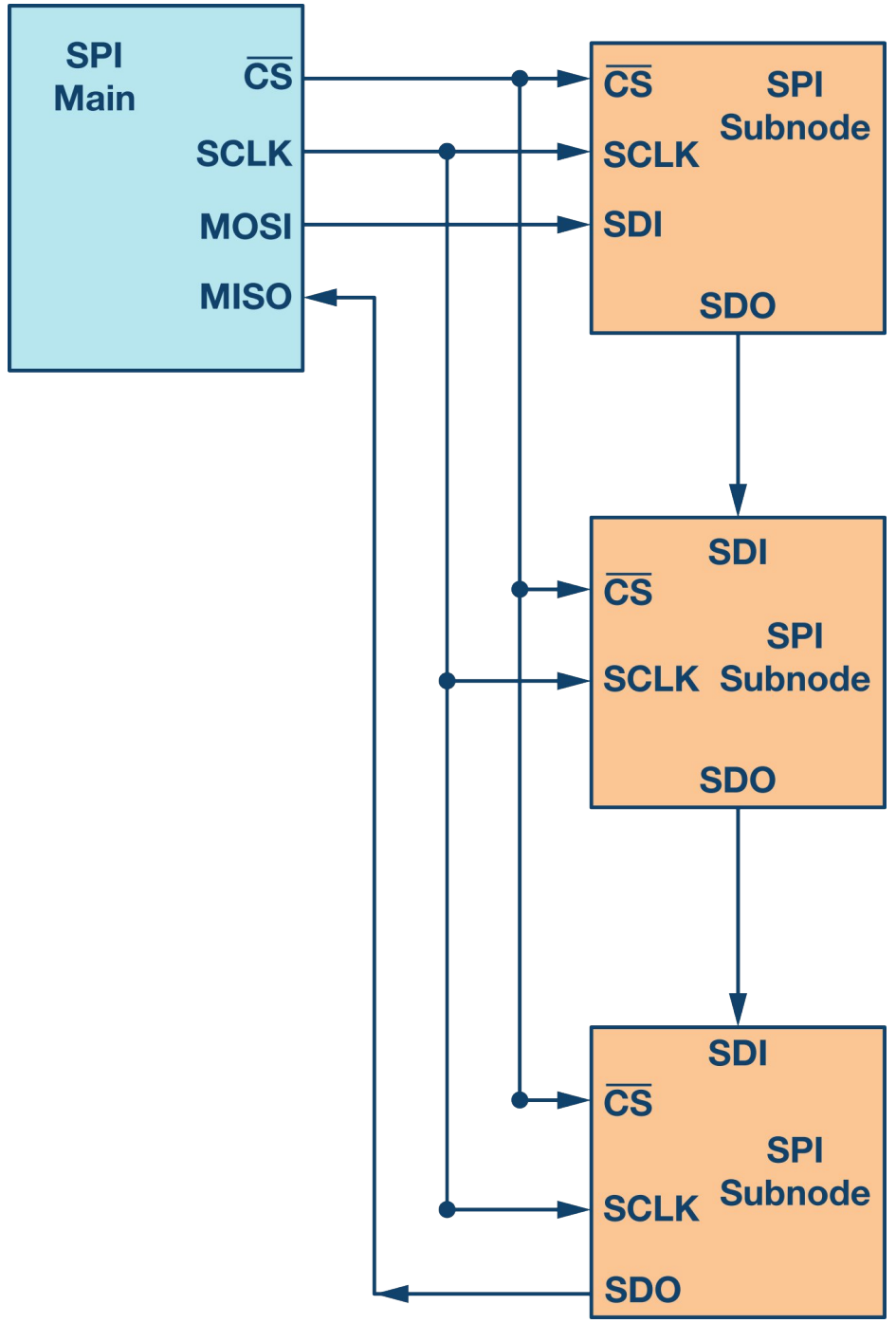
Todos los periféricos reciben el mismo reloj al mismo tiempo.

Modo daisy-chain

Conforme los datos se propagan de un periférico al siguiente, el número de ciclos de reloj requeridos para transmitir datos es proporcional a la posición del periférico en la daisy chain.

En el siguiente diagrama, considerando un sistema de 8 bits, se requieren 24 ciclos de reloj para que los datos estén disponibles en el 3er periférico, comparado con solo ocho pulsos en modo SPI regular.

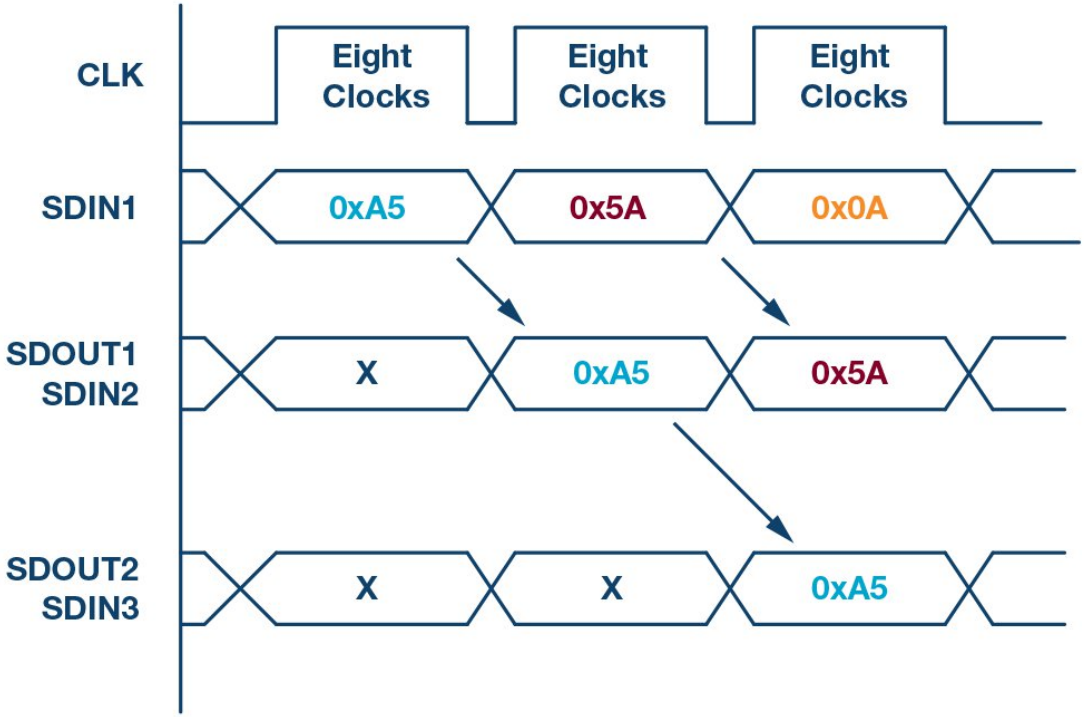
Modo daisy-chain



Modo daisy-chain

El siguiente diagrama muestra los ciclos de reloj y datos propagándose en la daisy chain.

El modo daisy chain no necesariamente es soportado por todos los periféricos SPI.



QUAD SPI (QSPI)

Especialmente diseñado para comunicación con chips de memoria.

El código en la memoria externa se podría ejecutar casi tan rápido como el de la memoria interna.

Útil en aplicaciones que involucran transferencias intensivas de datos de memoria.

Se puede usar para almacenar código en una memoria externa al microcontrolador.

QSPI

Por qué se usa QSPI en lugar de SPI para chips de memoria

Las memorias flash son baratas y una buena opción para sistemas embebidos. Sin embargo, son lentas lo cual puede causar cuellos de botella en la aplicación.

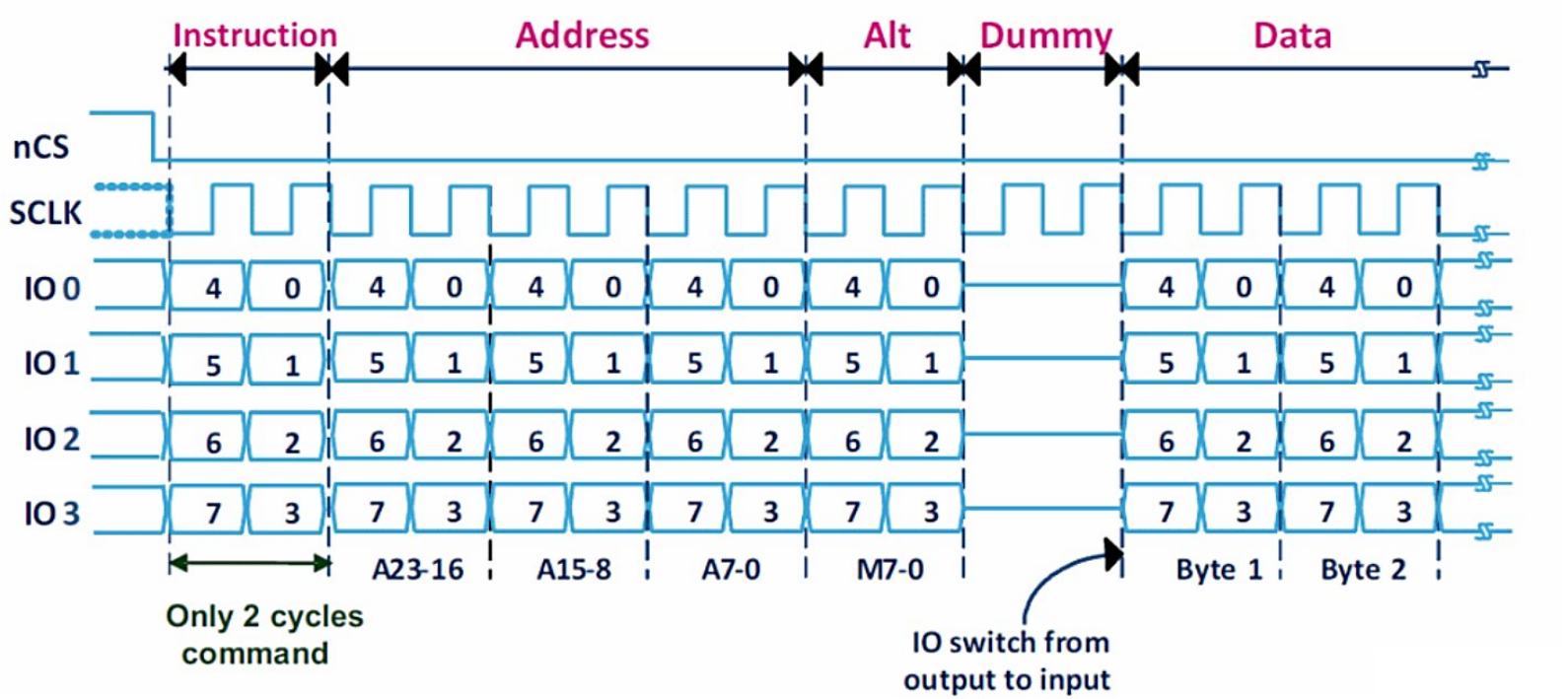
Aunque SPI es veloz pudiendo alcanzar 16Mbps, los chips de memoria flash no son capaces de transferir datos a esa velocidad a través de una sola línea.

QSPI

A diferencia de SPI que usa líneas separadas para las entradas y salidas (MISO y MOSI). QSPI configura las líneas de datos en el aire.

Usa 4 líneas para transferir información: I0, I1, I2 e I3.

QSPI


Instruction
Address
Alt
Dummy
Data
nCS
SCLK
IO 0
IO 1
IO 2
IO 3
A23-16
A15-8
A7-0
M7-0
Byte 1
Byte 2
Only 2 cycles command
IO switch from output to input

- Master
https://docs.espressif.com/projects/esp-idf/en/v5.3.1/esp32/api-reference/peripherals/spi_master.html

- Slave
https://docs.espressif.com/projects/esp-idf/en/v5.3.1/esp32/api-reference/peripherals/spi_slave.html

ESP32 tiene 4 SPI:

- **SPI0** y **SPI1** son dos periféricos comparten el mismo bus SPI (mismas señales y E/S). La diferencia es que CS0 está conectado a la memoria flash principal para el almacenamiento del firmware y CS1 está conectado a PSRAM. No son usables para interfaces SPI generales.

- **SPI2** y **SPI3** son SPI de propósito general que están disponibles para que los utilicen los clientes.

SPI2_HOST → HSPI
SPI3_HOST → VSPI

Controlador	Función	GPIO por default
HSPI	MOSI	GPIO 13
	MISO	GPIO 12
	SCLK	GPIO 14
	CS_0 [1]	GPIO 15
VSPI	MOSI	GPIO 23
	MISO	GPIO 19
	SCLK	GPIO 18
	CS_0 [1]	GPIO 5

[1] Primer dispositivo agregado al bus

MASTER

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/spi_master.h"
#include "driver/gpio.h"
#include "esp_log.h"
#include "string.h"

#define PIN_NUM_MISO 19
#define PIN_NUM_MOSI 23
#define PIN_NUM_CLK  18
#define PIN_NUM_CS   5
#define BUFFER_LEN   16
```

```c
void app_main(void) {
    spi_bus_config_t buscfg = {
        .miso_io_num = PIN_NUM_MISO,
        .mosi_io_num = PIN_NUM_MOSI,
        .sclk_io_num = PIN_NUM_CLK,
        .quadwp_io_num = -1,
        .quadhd_io_num = -1
    };

    ESP_ERROR_CHECK(spi_bus_initialize(HSPI_HOST, &buscfg, SPI_DMA_CH_AUTO));

    spi_device_interface_config_t devcfg = {
        .clock_speed_hz = 1 * 1000 * 1000, /* 1 MHz */
        .mode = 0,                         /* Modo SPI 0 */
        .spics_io_num = PIN_NUM_CS,
        .queue_size = 1,
        .flags = 0,
        .cs_ena_posttrans = 16,
    };

    spi_device_handle_t handle;
    ESP_ERROR_CHECK(spi_bus_add_device(HSPI_HOST, &devcfg, &handle));
```

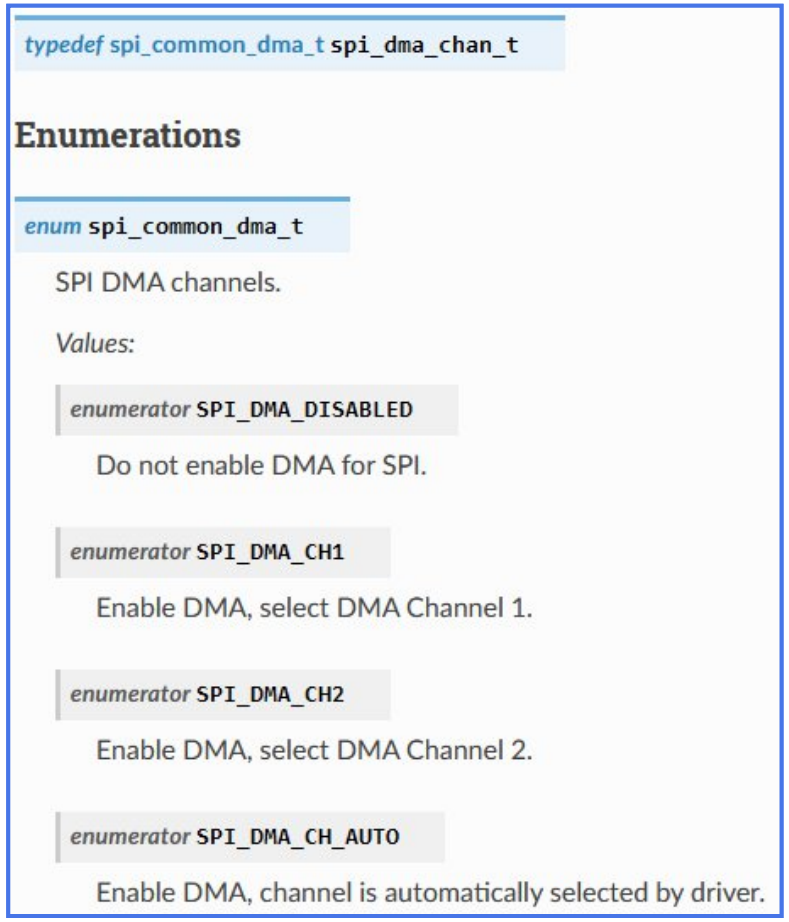
```
    char data[BUFFER_LEN] = {0};
    uint8_t i = 0;
    spi_transaction_t trans;
    memset(&trans, 0, sizeof(trans));
    trans.length = 8 * 8; /* Longitud en bits */
    trans.tx_buffer = data;

    while (1)
    {
        memset(data, 0, BUFFER_LEN);
        snprintf(data, BUFFER_LEN, "Hola %02X!", i++);
        ESP_ERROR_CHECK(spi_device_transmit(handle, &trans));
        ESP_LOGI(TAG, "Datos enviados al periferico: %s", data);
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }

    spi_bus_remove_device(handle);
    spi_bus_free(HSPI_HOST);
}
```

```c
typedef struct {
    union {
      int mosi_io_num;    ///< GPIO pin for Master Out Slave In (=spi_d) signal, or -1 if not used.
      int data0_io_num;   ///< GPIO pin for spi data0 signal in quad/octal mode, or -1 if not used.
    };
    union {
      int miso_io_num;    ///< GPIO pin for Master In Slave Out (=spi_q) signal, or -1 if not used.
      int data1_io_num;   ///< GPIO pin for spi data1 signal in quad/octal mode, or -1 if not used.
    };
    int sclk_io_num;      ///< GPIO pin for SPI Clock signal, or -1 if not used.
    union {
      int quadwp_io_num;  ///< GPIO pin for WP (Write Protect) signal, or -1 if not used.
      int data2_io_num;   ///< GPIO pin for spi data2 signal in quad/octal mode, or -1 if not used.
    };
    union {
      int quadhd_io_num;  ///< GPIO pin for HD (Hold) signal, or -1 if not used.
      int data3_io_num;   ///< GPIO pin for spi data3 signal in quad/octal mode, or -1 if not used.
    };
    int data4_io_num;     ///< GPIO pin for spi data4 signal in octal mode, or -1 if not used.
    int data5_io_num;     ///< GPIO pin for spi data5 signal in octal mode, or -1 if not used.
    int data6_io_num;     ///< GPIO pin for spi data6 signal in octal mode, or -1 if not used.
    int data7_io_num;     ///< GPIO pin for spi data7 signal in octal mode, or -1 if not used.
    int max_transfer_sz;  ///< Maximum transfer size, in bytes. Defaults to 4092 if 0 when DMA enabled, or to
    `SOC_SPI_MAXIMUM_BUFFER_SIZE` if DMA is disabled.
    uint32_t flags;       ///< Abilities of bus to be checked by the driver. Or-ed value of ``SPICOMMON_BUSFLAG_*`` flags.
    intr_cpu_id_t  isr_cpu_id;    ///< Select cpu core to register SPI ISR.
    int intr_flags;       /**< Interrupt flag for the bus to set the priority, and IRAM attribute, see
                           *  ``esp_intr_alloc.h``. Note that the EDGE, INTRDISABLED attribute are ignored
                           *  by the driver. Note that if ESP_INTR_FLAG_IRAM is set, ALL the callbacks of
                           *  the driver, and their callee functions, should be put in the IRAM.
                           */
} spi_bus_config_t;
```

esp_err_t **spi_bus_initialize**(
spi_host_device_t **host_id**, /* Periférico SPI que controla el bus (HSPI_HOST o VSPI_HOST) */
const spi_bus_config_t ***bus_config**, /* Apuntador a spi_bus_config_t que especifica cómo se debe inicializar el host */
spi_dma_chan_t **dma_chan** /* Especifica si deseas utilizar un canal DMA (Direct Memory Access) para transferencias rápidas */
)



typedef spi_common_dma_t **spi_dma_chan_t**

Enumerations

enum **spi_common_dma_t**

SPI DMA channels.

Values:

enumerator **SPI_DMA_DISABLED**

Do not enable DMA for SPI.

enumerator **SPI_DMA_CH1**

Enable DMA, select DMA Channel 1.

enumerator **SPI_DMA_CH2**

Enable DMA, select DMA Channel 2.

enumerator **SPI_DMA_CH_AUTO**

Enable DMA, channel is automatically selected by driver.

```
esp_err_t spi_bus_add_device(
spi_host_device_t host_id, /* Periférico SPI que controla el bus */
const spi_device_interface_config_t *dev_config, /* Configuración para el dispositivo */
spi_device_handle_t *handle /* Apuntador para el manejador del dispositivo */
)
```

Asignar un dispositivo en un bus SPI.

Inicializa las estructuras internas para un dispositivo, además de asignar un pin CS en el periférico maestro SPI y enrutarlo al GPIO indicado. Todos los maestros SPI tienen tres pines CS y, por lo tanto, pueden controlar hasta tres dispositivos.

esp_err_t **spi_device_transmit**(
spi_device_handle_t **handle**, /* Manejador del dispositivo */
spi_transaction_t ***trans_desc** /* Descripción de la transacción a ejecutar */
)

Remueve un dispositivo del bus SPI

esp_err_t **spi_bus_remove_device**(
spi_device_handle_t **handle** /* Manejador del dispositivo */
)

esp_err_t **spi_bus_free**(
spi_host_device_t **host_id** /* Periférico SPI a liberar */
)

Todos los periféricos deben removerse antes de ejecutar esta función

SLAVE

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/spi_slave.h"
#include "driver/gpio.h"
#include "esp_log.h"
#include "string.h"

#define PIN_NUM_MISO 19
#define PIN_NUM_MOSI 23
#define PIN_NUM_CLK  18
#define PIN_NUM_CS   13

static const char* TAG = "SPI_SLAVE";
```

```c
void app_main(void) {
    spi_bus_config_t buscfg = {
        .miso_io_num = PIN_NUM_MISO,
        .mosi_io_num = PIN_NUM_MOSI,
        .sclk_io_num = PIN_NUM_CLK,
        .quadwp_io_num = -1,
        .quadhd_io_num = -1
    };

    /* Configuración del periferico SPI */
    spi_slave_interface_config_t slvcfg = {
        .mode = 0,                          /* Modo SPI 0 */
        .spics_io_num = PIN_NUM_CS,
        .queue_size = 5,
        .flags = 0
    };

    /* Inicializar bus SPI en modo esclavo */
    ESP_ERROR_CHECK(spi_slave_initialize(HSPI_HOST, &buscfg, &slvcfg, SPI_DMA_CH_AUTO));
```

```
    char recvbuf[128] = {0};
    spi_slave_transaction_t trans;
    memset(&trans, 0, sizeof(trans));
    trans.length = 8 * 8; /* Longitud en bits */
    trans.rx_buffer = recvbuf;

    while (1) {
        ESP_LOGI(TAG, "Esperando datos del controlador...");
        ESP_ERROR_CHECK(spi_slave_transmit(HSPI_HOST, &trans, portMAX_DELAY));
        ESP_LOGI(TAG, "Datos recibidos: %s", recvbuf);
        memset(recvbuf, 0, 128);
    }

    spi_slave_free(HSPI_HOST);
}
```

esp_err_t **spi_slave_initialize**(
spi_host_device_t **host**, /* Periférico SPI que controla el bus (HSPI_HOST o VSPI_HOST) */
const spi_bus_config_t ***bus_config**, /* Apuntador a spi_bus_config_t que especifica cómo se debe inicializar el host */
const spi_slave_interface_config_t ***slave_config**, /* Apuntador a spi_slave_interface_config_t que especifica los detalles de la interfaz slave */
spi_dma_chan_t **dma_chan** /* Especifica si deseas utilizar un canal DMA */
)

struct **spi_slave_transaction_t**

This structure describes one SPI transaction

Public Members

uint32_t **flags**

Bitwise OR of SPI_SLAVE_TRANS_* flags.

size_t **length**

Total data length, in bits.

size_t **trans_len**

Transaction data length, in bits.

const void ***tx_buffer**

Pointer to transmit buffer, or NULL for no MOSI phase.

void ***rx_buffer**

Pointer to receive buffer, or NULL for no MISO phase. When the DMA is enabled, must start at WORD boundary (`rx_buffer%4==0`), and has length of a multiple of 4 bytes.

void ***user**

User-defined variable. Can be used to store eg transaction ID.

esp_err_t **spi_slave_transmit**(
spi_host_device_t **host**, /* Periférico SPI que controla el bus */
spi_slave_transaction_t ***trans_desc**, /* Apuntador a la transacción a ejecutar. No es *const* porque es posible que se vaya a escribir el status */
TickType_t **ticks_to_wait** /* Ticks a esperar hasta que regrese un elemento; use portMAX_DELAY para esperar por siempre. */
)

esp_err_t **spi_slave_free**(
spi_host_device_t **host** /* Periférico SPI a liberar */
)

Tarea:

- Revisar la documentación del ESP-IDF sobre SPI.
- Ejecutar los ejemplos vistos en clase.