
SPI

Introduction to SPI (Serial Peripheral Interface)

SPI (Serial Peripheral Interface) is a synchronous serial communication protocol used in embedded systems. It enables a master device (e.g., a microcontroller) to communicate with slave devices such as sensors, EEPROMs, SD cards, displays, and wireless modules.

Other communication protocols include CAN, I2C, Ethernet, USB, RS232, and RS485, but SPI is widely used due to its simplicity and speed.

SPI Pins

SPI communication requires four main pins:

MISO (Master In Slave Out) – Used for data transmission from slave to master.

MOSI (Master Out Slave In) – Used for data transmission from master to slave.

SCK (Serial Clock) – The clock signal generated by the master to synchronize communication.

SS (Slave Select) – Used to select a specific slave device when multiple slaves are connected.

Key Points

SPI is synchronous, meaning communication is controlled by a clock signal.

The master generates the clock, and the slave follows it.

If SPI communication is not working, the first debugging step is to check whether the master is generating the clock signal.

The SS pin is only required when multiple slave devices are present.

Slave Selection:

The master selects the slave by pulling the Slave Select (SS) pin LOW (0) using a GPIO pin.

This is crucial because the slave remains inactive (high impedance state) until SS is pulled low.

Data Transmission:

Once selected, the slave activates its MISO and MOSI lines.

The master sends data along with a clock signal (SCK) to synchronize communication.

Pulling SS LOW is mandatory; otherwise, the slave remains inactive.

SPI data transfer is synchronized with the master's clock signal.

The program must control the SS pin to enable proper communication.

Minimal SPI Bus Configuration

SPI communication typically involves four pins (MOSI, MISO, SCK, SS), but in some applications, it can be reduced to just two wires depending on the communication requirements.

Minimal SPI Setup (2-Wire Configuration)

In some cases, only MOSI (Master Out, Slave In) and SCK (Clock) are needed.

If the slave only receives data (does not send any data back), the MISO line is not required.

The SS (Slave Select) pin can be permanently grounded if only one slave is used.

SPI pins can be customized based on application needs.

Not all applications require all four SPI pins;

Example:

2-wire SPI: Only MOSI and SCK are used when the slave only receives data.

3-wire SPI: MOSI, MISO, and SCK (no SS if only one slave).

4-wire SPI: Standard SPI with all signals.

The number of SPI lines depends on how many slaves are present and how data is exchanged.

How SPI Hardware Works

SPI communication is based on shift registers, which enable synchronous data exchange between the master and slave devices.

How Data is Transferred in SPI?

Both the master and slave have an 8-bit shift register.

The MOSI (Master Out, Slave In) and MISO (Master In, Slave Out) lines connect the devices.

The serial clock (SCK), generated by the master, synchronizes data transfer.

Step-by-Step Data Exchange Process

The master loads data (e.g., A0-A7) into its shift register.

After one clock cycle, the LSB (A0) is pushed out on MOSI and moves to the slave's MSB.

Simultaneously, the slave's LSB (B0) moves out on MISO and enters the master's MSB.

With each clock cycle, bits shift one position in both registers.

After 8 clock cycles, the master's data is fully transferred to the slave, and the slave's data is fully transferred to the master.

SPI works on shift registers, where data moves bit-by-bit with each clock pulse.

SPI always exchanges data – when the master sends data, it also receives data (if the MISO line is connected).

The process is full-duplex, meaning both transmission and reception happen simultaneously.

Different SPI bus configurations.

SPI communication can be configured in different ways depending on the application requirements and hardware capabilities of the microcontroller.

1. Full Duplex Communication (Default Mode)

SPI is naturally full duplex, meaning data can be transmitted and received simultaneously.

Two dedicated data lines:

MOSI (Master Out, Slave In) → Master sends data to Slave.

MISO (Master In, Slave Out) → Slave sends data to Master.

Clock (SCK) synchronizes the shift registers of both master and slave.

Used in high-speed data exchange applications.

2. Half Duplex Communication

Uses only one data line (instead of two) to save GPIO pins.

MOSI (Master) is connected to MISO (Slave) via a resistor (e.g., $1k\Omega$).

No dedicated transmit or receive lines – data direction is switched based on configuration:

When Master is in Transmit mode, Slave must be in Receive mode.

When Master is in Receive mode, Slave must be in Transmit mode.

SS (Slave Select) is optional when only one slave is used.

Not all microcontrollers support Half Duplex SPI, so it's important to check the reference manual of the MCU.

3. Simplex Communication (One-Way Data Transfer)

Two types:

Transmit-Only Mode: Master only sends data, Slave only receives.

Receive-Only Mode: Master only receives data, Slave only transmits.

Unnecessary lines are disconnected (e.g., MISO is unused in Transmit-Only mode).

The unused pins can be configured as GPIOs for other functions.

Useful for applications where data flow is strictly one-directional.

Full Duplex: Two-way communication (default mode, uses MOSI & MISO).

Half Duplex: Saves GPIO pins, requires direction control.

Simplex: One-way data transfer (Transmit-Only or Receive-Only).

Understanding the SPI Functional Block Diagram

SPI communication in STM32F411x microcontrollers follows a structured hardware design, which includes shift registers, buffers, and control logic. The same principle applies to other microcontrollers, though register names and details may vary.

SPI Block Diagram Overview

The SPI peripheral consists of the following key components:

1. SPI Interface Pins

NSS (Slave Select)

SCK (Serial Clock)

MOSI (Master Out, Slave In)

MISO (Master In, Slave Out)

2. Shift Register

Used for data exchange between Master and Slave.

The shift register width is up to 16 bits, supporting both 8-bit and 16-bit data transfers.

MOSI and MISO are directly connected to the shift register.

3. Baud Rate Generator

Generates the clock signal (SCK) when SPI operates in Master mode.

The baud rate can be configured through SPI control registers.

4. TX Buffer (Transmit Buffer)

Data to be sent is first written into the TX buffer.

If the shift register is free, the data moves from TX buffer to the shift register, and transmission begins.

A TX Buffer Empty interrupt is triggered when the TX buffer is ready for new data.

5. RX Buffer (Receive Buffer)

Data received is first stored in the shift register.

Once a full frame (8 or 16 bits) is received, the data is moved to the RX buffer.

A RX Buffer Full interrupt is triggered when new data is available for reading.

6. APB Bus Interface

SPI interacts with the APB (Advanced Peripheral Bus) for data transfer.

The CPU accesses SPI registers through this interface.

Transmission Path:

Write data to the TX Buffer.

If the shift register is free, data moves from TX Buffer → Shift Register → MOSI Line.

Transmission is synchronized with the serial clock (SCK).

Reception Path:

Data received over MISO enters the Shift Register.

When a full frame (8 or 16 bits) is received, it moves to the RX Buffer.

An interrupt notifies the application to read the received data.

Interruptions in SPI Communication

TX Buffer Empty Interrupt → Signals when the TX buffer is ready for new data.

RX Buffer Full Interrupt → Signals when new data is received.

Shift registers handle SPI data transfer, shifting bits on each clock pulse.

TX and RX buffers manage data flow, ensuring smooth transmission and reception.

Baud rate control allows adjusting SPI clock speed.

Interrupts simplify data handling, avoiding CPU polling.

Software Slave Management (SSM)

Controlled internally by software instead of an external NSS pin.

Enabled by setting the SSM bit = 1 in the SPI Control Register (CR1).

The NSS state is controlled by the SSI bit:

SSI = 0 → NSS is internally grounded (Slave is always selected).

$\text{SSI} = 1 \rightarrow \text{NSS}$ is internally pulled high (Slave is deselected).

Hardware Slave Management (HSM)

NSS is physically used to select slaves in the SPI network.

$\text{SSM} = 0$ (Software slave management is disabled).

Slave only communicates when NSS is pulled LOW by the master.

The master must set NSS as an output pin to control slaves.

SPI Communication Formats: CPHA, CPOL, and Data Frame Format

SPI communication depends on three important factors:

CPOL (Clock Polarity)

CPHA (Clock Phase)

Data Frame Format

For successful communication, the master and slave must use the same settings; otherwise, data corruption can occur.

1. Data Frame Format

Defines how many bits are transmitted in one SPI frame.

Common formats:

8-bit frame (default)

16-bit frame (for wider data transfers)

Both master and slave must have the same frame size.

2. Clock Polarity (CPOL)

CPOL defines the idle state of the serial clock (SCK) when no data is being transferred.

Example:

$\text{CPOL} = 0$: SCK remains LOW when inactive.

$\text{CPOL} = 1$: SCK remains HIGH when inactive.

3. Clock Phase (CPHA)

CPHA controls when data is sampled relative to the clock edge.

Important Rule:

If CPHA = 0, data is captured on the first clock edge.

If CPHA = 1, data is captured on the second clock edge.

4. SPI Modes (Based on CPOL and CPHA)

SPI has four different modes, determined by CPOL and CPHA settings.

Simplified Rule:

If CPHA = 0 → Data sampled on the leading edge.

If CPHA = 1 → Data sampled on the trailing edge.

When to Use Different SPI Modes?

Mode 0 (CPOL = 0, CPHA = 0) → Default mode, works for most applications.

Mode 1 (CPOL = 0, CPHA = 1) → Used when slave needs more time to stabilize data.

Mode 2 (CPOL = 1, CPHA = 0) → Used when idle clock should be HIGH.

Mode 3 (CPOL = 1, CPHA = 1) → Used for specific peripherals that require different clock timing.

Understanding SPI Serial Clock in ST

The maximum SPI clock speed depends on two factors:

The APB bus frequency on which the SPI peripheral is connected.

The internal SPI prescaler value configured in the SPI baud rate control register.

1. Finding the Maximum SPI Clock Speed

To determine the maximum SPI serial clock (SCK) speed, follow these steps:

Identify which APB bus the SPI peripheral is connected to.

Find the clock frequency of that APB bus.

Apply the internal SPI prescaler to determine the final SCK frequency.

2. SPI Peripherals and Their Bus Frequencies

The SPI peripherals are mapped as follows:

However, in our application, we are using the internal 16 MHz HSI oscillator, meaning:

APB1 Bus runs at 16 MHz → SPI2 & SPI3 max SCK = 8 MHz

APB2 Bus runs at 16 MHz → SPI1 max SCK = 8 MHz

3. SPI Prescaler and Serial Clock Calculation

The SPI peripheral has an internal prescaler that further divides the APB clock to set the SPI clock speed.

Default SPI prescaler = 2, meaning:

APB1 at 16 MHz → SPI SCK = $16 \text{ MHz} / 2 = 8 \text{ MHz}$

APB2 at 16 MHz → SPI SCK = $16 \text{ MHz} / 2 = 8 \text{ MHz}$

The prescaler can be increased to lower the SPI clock speed:

/4 → 4 MHz

/8 → 2 MHz

/16 → 1 MHz, etc.

If the APB1 clock is increased to 42 MHz, SPI2 & SPI3 can achieve up to 21 MHz.

If the APB2 clock is increased to 84 MHz, SPI1 can achieve up to 42 MHz.

3. User-Configurable Parameters

The user should be able to configure:

SPI Mode – Master or Slave.

Communication Mode – Full Duplex, Half Duplex, Simplex.

Data Frame Format – 8-bit or 16-bit.

Clock Polarity (CPOL) and Clock Phase (CPHA) – Determines clock behavior.

Slave Select Management – Software or Hardware NSS.

SPI Speed (Baud Rate Prescaler) – Controls the SPI clock speed.

SPI Register Definition Structure

```
typedef struct
{
    volatile uint32_t CR1;      // SPI Control Register 1
    volatile uint32_t CR2;      // SPI Control Register 2
    volatile uint32_t SR;       // SPI Status Register
    volatile uint32_t DR;       // SPI Data Register
    volatile uint32_t CRCPR;    // SPI CRC Polynomial Register
    volatile uint32_t RXCRCR;   // SPI RX CRC Register
    volatile uint32_t TXCRCR;   // SPI TX CRC Register
} SPI_RegDef_t;
```

Step for SPI Init

1. Configure the Device Mode (Master/Slave)
2. Configure the Bus Configuration
3. Configure the SPI Clock Speed (Baud Rate)
4. Configure the Data Frame Format (8-bit or 16-bit)
5. Configure the Clock Polarity and Phase
6. Configure Software Slave Management (SSM)

Step for Implementation of SPI Send Data - Blocking Mode Transmission

2. Implementing SPI_SendData()

Step 1: Wait for TXE flag to be set

TXE = 1 → SPI is ready for data transmission.

TXE = 0 → SPI is still busy, wait until TXE becomes 1.

Step 2: Check DFF Bit (Data Frame Format)

if (pSPIx->CR1 & (1 << SPI_CR1_DFF))

DFF = 0 → 8-bit mode → Load 1 byte into DR.

DFF = 1 → 16-bit mode → Load 2 bytes into DR.

Step 3: Load Data into the Data Register (DR)

pSPIx->DR = *pTxBuffer; // For 8-bit transmission

pSPIx->DR = *((uint16_t*)pTxBuffer); // For 16-bit transmission

For 8-bit: Load one byte from pTxBuffer into DR.

For 16-bit: Load two bytes from pTxBuffer into DR.

Step 4: Update Buffer Pointer and Length

pTxBuffer++; // Move to the next byte (8-bit mode)

pTxBuffer += 2; // Move to the next word (16-bit mode)

len--; // Reduce length for 8-bit mode

len -= 2; // Reduce length for 16-bit mode