1.INTRODUCTION TO SPI PROTOCOL

1.1 Overview of the SPI protocol (simple)

SPI (Serial Peripheral Interface) is a way for electronic chips to talk to each other using a few wires. It's synchronous meaning data is sent in step with a clock signal and usually works in a master–slave setup: one master controls one or more slaves.

Think of it like a teacher (master) calling on students (slaves). The teacher gives a beat (clock) and asks one student at a time to answer (chip select). While the teacher speaks, the student replies both can exchange information at the same time (full-duplex).

Key ideas in plain words:

- It's serial: bits are sent one after another on a wire, not all at once.
- Synchronous: the clock coordinates when bits are sampled.
- Master/Slave: the master starts and controls every transfer; slaves only respond when selected.
- Commonly uses 4 signals (wires):
    - **SCLK** — Serial clock (from master)
    - **MOSI** — Master Out, Slave In (data master → slave)
    - **MISO** — Master In, Slave Out (data slave → master)
    - **CS / SS** — Chip Select or Slave Select (active low usually) — enables a particular slave

(If multiple slaves: additional CS lines: CS1, CS2, ...)

Short summary of how a transfer works:

1. Master pulls a slave's CS low to select it.
2. Master toggles SCLK; on each clock edge a bit moves on MOSI and simultaneously a bit comes back on MISO.
3. After all bits are transferred, master raises CS to finish the transaction.

1.2 History and evolution (simple)

SPI started as a simple, low-overhead serial interface introduced by semiconductor companies (originating from Motorola-era designs) to let microcontrollers talk to peripherals like sensors, ADCs, DACs, and memory chips. Its simplicity made it popular and long-lived.

Over time, several variants and extensions appeared to meet new needs:

- **Basic SPI** — the classic 4-wire form used for many peripherals.

- **Three-wire SPI** — combines MOSI/MISO into a single data line (half-duplex).
- **Dual/Quad SPI (DPI/QPI, QSPI)** — adds multiple data lines to increase throughput for flash memories and high-speed devices.
- **Daisy-chain and multi-slave topologies** — for connecting many devices with fewer pins (but with trade-offs).

Why it evolved: as devices needed faster data rates (e.g., NOR flash for booting), and as pin-count and board layouts became constrained, people extended SPI to run wider (multiple data lines) and to support new bus topologies.

Note: In the detailed report you can add a small timeline (year → milestone) if you want precise dates I can fetch accurate historical dates and references if you want them included.

1.3 Importance of SPI in embedded systems & SoC design (simple)

Why SPI matters in everyday engineering terms:

- **Simple and fast:** Easy to implement in hardware and firmware; often faster than I^2C for short-distance chip-to-chip links.
- **Full-duplex:** Master and slave can exchange bits simultaneously, which helps for some devices (e.g., streaming ADC data).
- **Low overhead:** No device addresses or handshake protocol — fewer cycles per transfer.
- **Flexible:** Works at many clock speeds; many devices support configurable modes (clock polarity/phase, bit order).
- **Widely supported:** Almost every microcontroller and sensor have SPI support makes integrating parts straightforward.

Limitations (so you know the trade-offs):

- **No standard discovery:** The bus doesn't have device addressing or automatic detection you must manage chip select lines.
- **Chip-select pin scaling:** More slaves usually mean more CS pins, which costs IO pins (though daisy-chaining or GPIO expanders can help).
- **No built-in flow control or ACK:** If the slave isn't ready, the master must handle retries at higher level.
- **Short distance:** Best for on-board communication — not suitable for long wiring like RS-485.

Importance for SoC / FPGA / RISC-V projects:

- SPI controllers are common IP blocks in SoCs and easy to integrate with bus bridges (AXI/AHB).
- Good candidate for RTL design practice: a working SPI master/slave teaches clocking, FIFOs, CPOL/CPHA handling, and bus interfacing.
- Flash memory using QSPI is often the boot source for embedded processors so SPI impacts boot-time and system architecture.

2. FUNDAMENTALS OF SPI PROTOCOL

2.1 Full Form and Purpose of SPI

- **Full Form:** SPI = **Serial Peripheral Interface**.

- **Purpose:** It allows a **master device** (usually a microcontroller, SoC, or processor) to exchange data with **peripheral devices** (slaves) such as:
    - Flash memories
    - ADC/DAC chips
    - Displays
    - Sensors (temperature, pressure, motion, etc.)
- SPI is **synchronous** (data is clocked in and out at the same time using a shared clock) and **full-duplex** (data flows both ways at the same time).

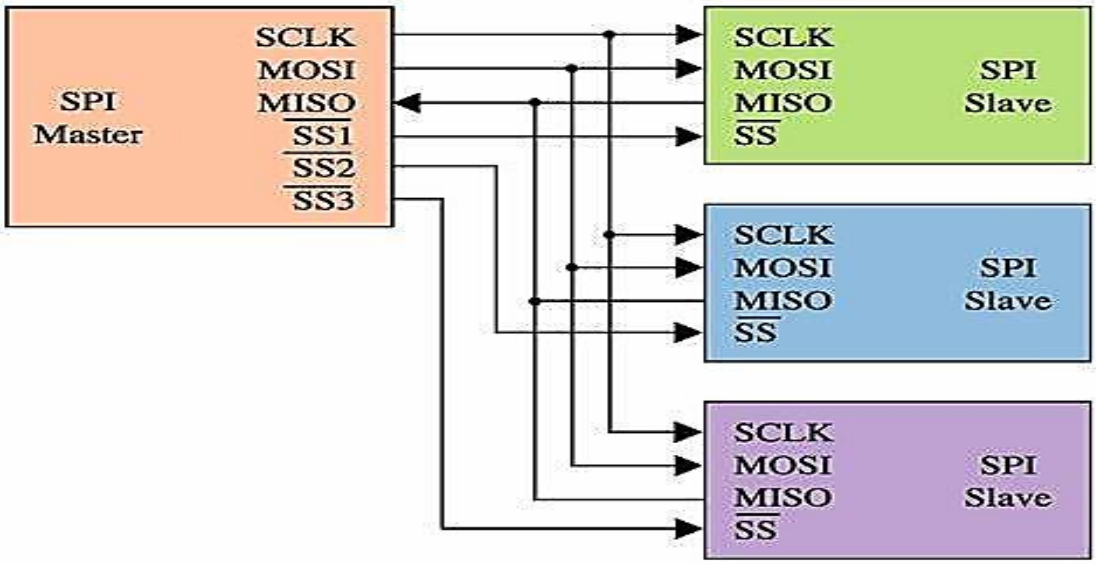
➢ **Key takeaway:** SPI is used where **speed, simplicity, and reliability over short distances** are important.

2.2 SPI Communication Model

The SPI protocol follows a **master–slave communication model**:

- **Master:**
    - Generates the **clock signal (SCLK)**.
    - Initiates and controls all data transfers.
    - Decides which slave is active by controlling **Chip Select (CS)**.
- **Slave(s):**
    - Respond only when their CS pin is active (usually low).
    - Use the master's clock to synchronize data transfer.
    - Can be one or multiple devices on the same SPI bus.

Bus Structure



- **MOSI:** Data from master → slave.
- **MISO:** Data from slave → master.
- **SCLK:** Clock pulses from master.
- **CS (Chip Select):** Dedicated pin for each slave or shared in special setups.

2.3 Serial vs Parallel Communication Basics

Before diving deeper into SPI, it helps to understand how **serial communication** differs from **parallel communication**:

Feature	Serial Communication (SPI style)	Parallel Communication
Data Transfer	One bit at a time	Multiple bits at once
Wires Needed	Few (2–4 lines in SPI)	Many (one per bit + control lines)
Speed	Can be very high with fewer wires	Limited by crosstalk and wire length
Distance	Better for short distances	Poor over long distances
Cost/Complexity	Lower (fewer traces, simpler PCB)	Higher (more traces, more pins)

➢ **Why SPI uses serial:**

- Uses fewer pins → cheaper and easier PCB routing.
- Can achieve very high speeds (tens of MHz).
- Perfect for connecting ICs within the same board.

3. SPI ARCHITECTURE

3.1 Master–Slave Concept

- **Master Device**
  - Controls the communication process.
  - Generates the clock signal (**SCLK**).
  - Activates one slave at a time using **Chip Select (CS)**.
  - Sends and receives data simultaneously.
- **Slave Device**
  - Waits for the master's **CS** signal.
  - Uses the master's clock to shift in and shift out data.
  - Can only communicate when selected.

Key Point:
SPI can have **one master with multiple slaves** or **multiple masters** (rare but possible). The most common is **single-master, multi-slave**.

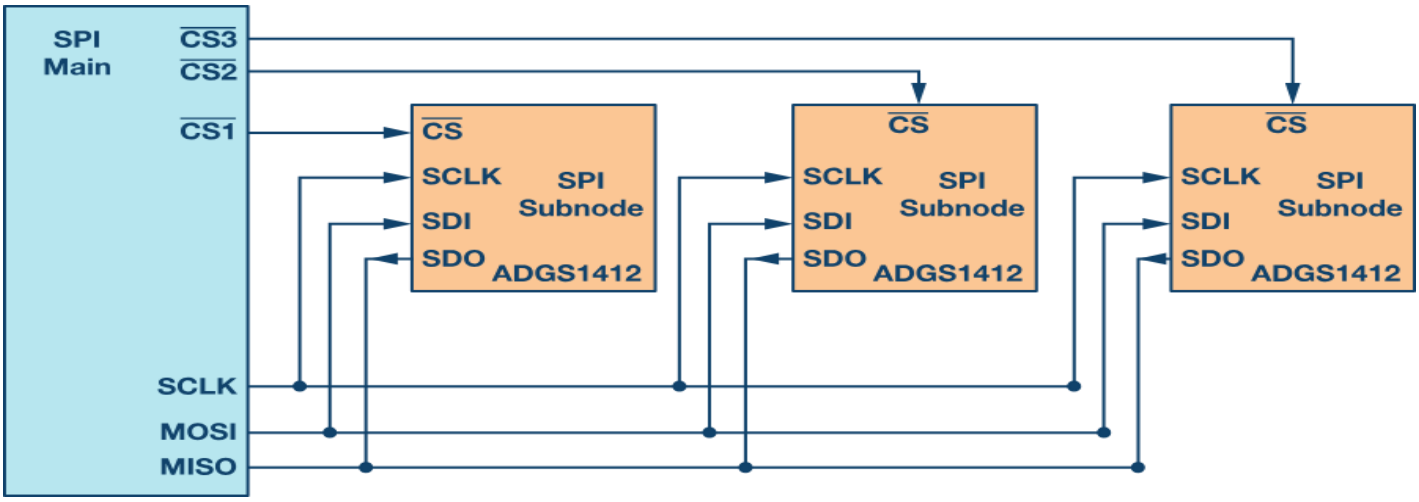
3.2 Key Components

An SPI bus generally consists of **four main signal lines**:

1. **SCLK (Serial Clock)** – Generated by the master to synchronize data.
2. **MOSI (Master Out Slave In)** – Data line from master → slave.
3. **MISO (Master In Slave Out)** – Data line from slave → master.
4. **CS/SS (Chip Select/Slave Select)** – Active-low signal to select which slave communicates.

Diagram: Single-Master, Multi-Slave

3.3 SPI Bus Topologies



There are **different ways to connect multiple devices** on the SPI bus.

3.3.1 Single-Master, Multi-Slave (Most Common)

- Each slave has its own CS line from the master.
- All slaves share SCLK, MOSI, and MISO.
- Only the selected slave drives MISO; others must be in high-impedance (Hi-Z) state.

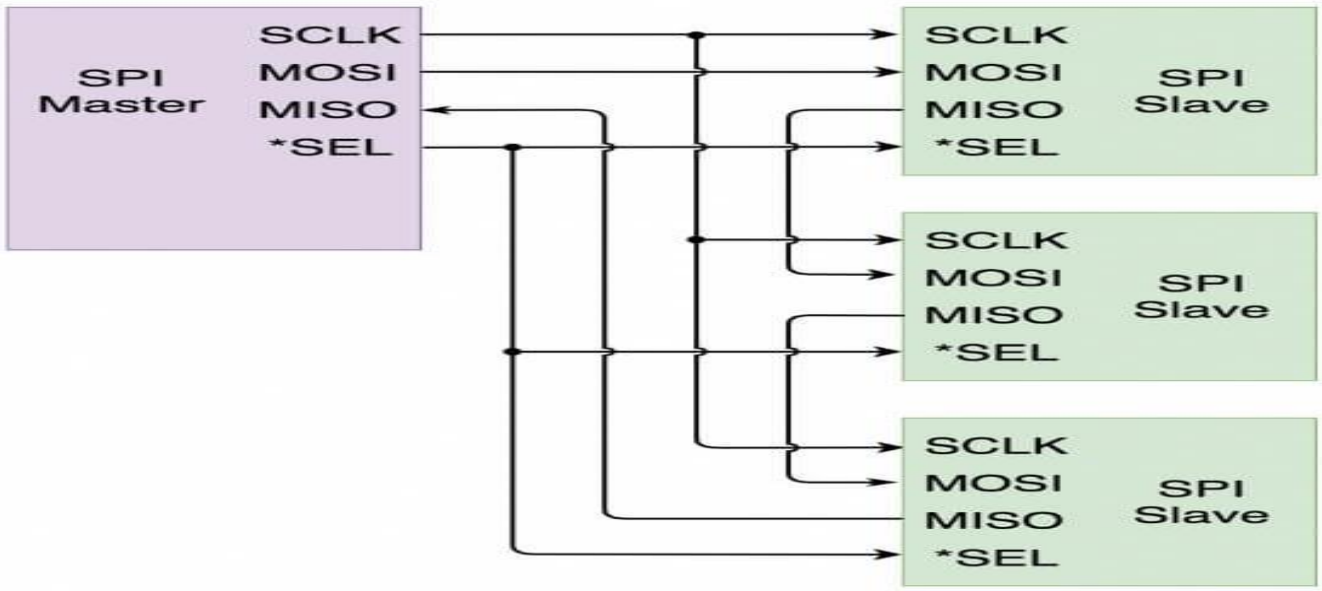
Pros:

- Simple to implement.
- Each slave gets full bandwidth when selected.

Cons:

- Requires one CS pin per slave.

3.3.2 Daisy Chain Topology



- Slaves are connected in series.
- MISO of one device connects to MOSI of the next.
- A single CS line can control the whole chain.

Pros:

- Saves CS pins.
- Good for devices designed for chaining (e.g., shift registers).

Cons:

- All devices shift data even if not addressed.
- Latency increases with more devices.

3.3.3 Multi-Master (Rare)

- More than one master on the bus.
- Requires arbitration to avoid collisions.
- Used in systems where two controllers might need to act as masters.

4. SPI SIGNAL LINES AND THEIR FUNCTIONS

The SPI protocol typically uses **four main signals** (in the standard configuration). Each signal plays a specific role in data exchange.

4.1 MOSI — Master Out, Slave In

- **Direction:** From Master → Slave.
- **Purpose:** Carries data **from the master to the selected slave**.
- **Usage:** Active only when a slave's Chip Select (CS) is active.
- **Timing:** Data is placed on MOSI by the master and is sampled by the slave on the specified clock edge (depends on SPI mode).
- **Electrical Note:** In multi-slave setups, all slaves share MOSI, but only the selected slave reads the data.

4.2 MISO — Master In, Slave Out

- **Direction:** From Slave → Master.
- **Purpose:** Carries data **from the selected slave back to the master**.
- **Usage:** Only the active slave drives MISO; others must be in **high-impedance (Hi-Z)** to avoid bus conflicts.
- **Timing:** Slave outputs data on MISO, which is sampled by the master on the specified clock edge.
- **Electrical Note:** Tri-state buffers are used in slave devices to ensure only one device drives MISO at a time.

4.3 SCLK — Serial Clock

- **Direction:** Always from Master → Slave(s).
- **Purpose:** Synchronizes data transfers on MOSI and MISO lines.
- **Frequency:** Can range from a few kHz to tens of MHz, depending on device capabilities.
- **Polarity & Phase:**
    - **CPOL (Clock Polarity):** Determines whether the clock idles high or low.

- **CPHA (Clock Phase):** Determines whether data is sampled on the leading or trailing clock edge.

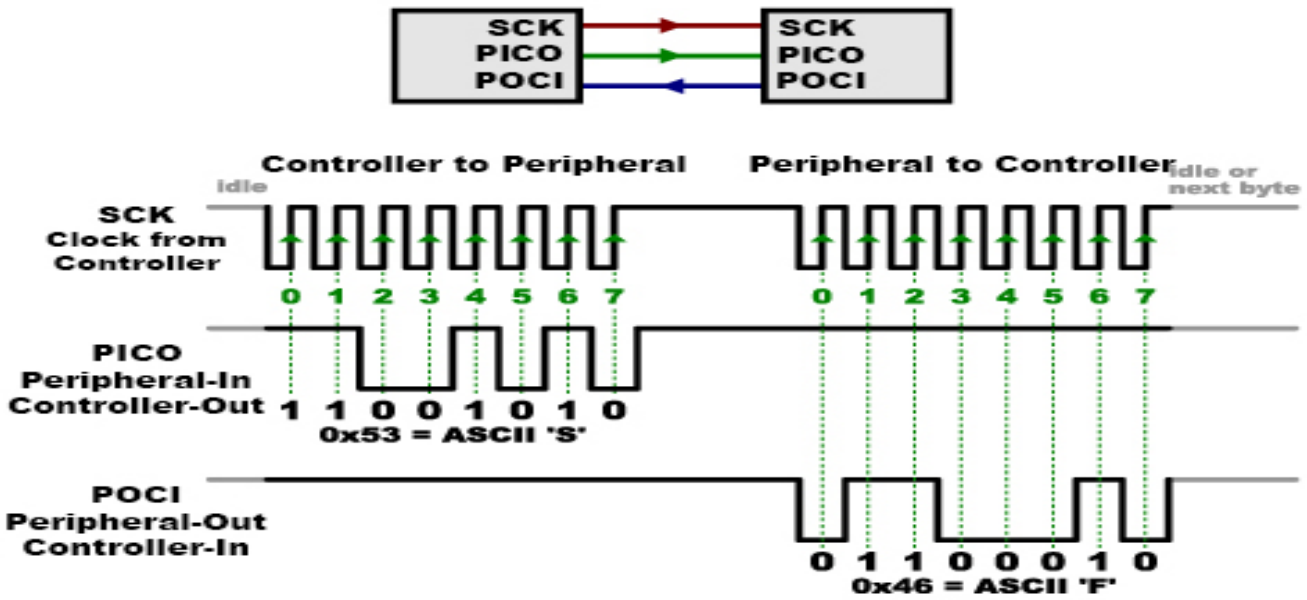
- **Important:** All devices must use the same CPOL and CPHA for communication to work correctly.

4.4 CS / SS — Chip Select / Slave Select

- **Direction:** From Master → Slave(s).
- **Purpose:** Activates a particular slave device.
- **Logic Level:** Usually **active low** (logic 0 = selected, logic 1 = idle).
- **Operation:**
  1. Master pulls CS low → slave becomes active.
  2. Master raises CS high → slave becomes inactive.
- **Multiple Slaves:** Each slave gets its own CS line (in standard multi-slave configuration).

4.5 Combined Signal Operation

When all signals work together:



1. **Idle State:** CS is high, SCLK is idle (CPOL-defined state), MOSI/MISO undefined or pulled high.
2. **Start Transfer:** Master pulls CS low, begins toggling SCLK.
3. **Data Exchange:** On each clock cycle, one bit is shifted out on MOSI (master → slave) and one bit is shifted out on MISO (slave → master).
4. **End Transfer:** Master raises CS high, both sides stop shifting data.

5. SPI MODES OF OPERATION

SPI devices must agree on **two key timing parameters** to exchange data correctly:

1. **Clock Polarity (CPOL)** – determines the idle state of the clock.
2. **Clock Phase (CPHA)** – determines on which clock edge data is sampled.

These two parameters create **four standard SPI modes**.

5.1 Clock Polarity (CPOL)

- **CPOL = 0** → Clock idles low (0) when not transferring data.
- **CPOL = 1** → Clock idles high (1) when not transferring data.

Impact: CPOL only affects the **idle level** of the clock line, not the number of clock pulses.

5.2 Clock Phase (CPHA)

- **CPHA = 0** → Data is **sampled on the first (leading) clock edge** and changed on the second (trailing) edge.
- **CPHA = 1** → Data is **sampled on the second (trailing) clock edge** and changed on the first (leading) edge.

5.3 Four Standard SPI Modes

The combination of CPOL and CPHA creates **4 modes**:

Mode	CPOL	CPHA	Clock Idle	Data Sampled On	Data Changed On
0	0	0	Low	Rising Edge	Falling Edge
1	0	1	Low	Falling Edge	Rising Edge
2	1	0	High	Falling Edge	Rising Edge
3	1	1	High	Rising Edge	Falling Edge

5.4 Timing Diagram for Each Mode

Figure 1 through Figure 5 show an example of communication in four SPI modes. In these examples, the data is shown on the MOSI and MISO line. The start and end of transmission is indicated by the dotted green line, the sampling edge is indicated in orange, and the shifting edge is indicated in blue. Please note these figures are for illustration purpose only. For successful SPI communications, users must refer to the product data sheet and ensure that the timing specifications for the part are met.

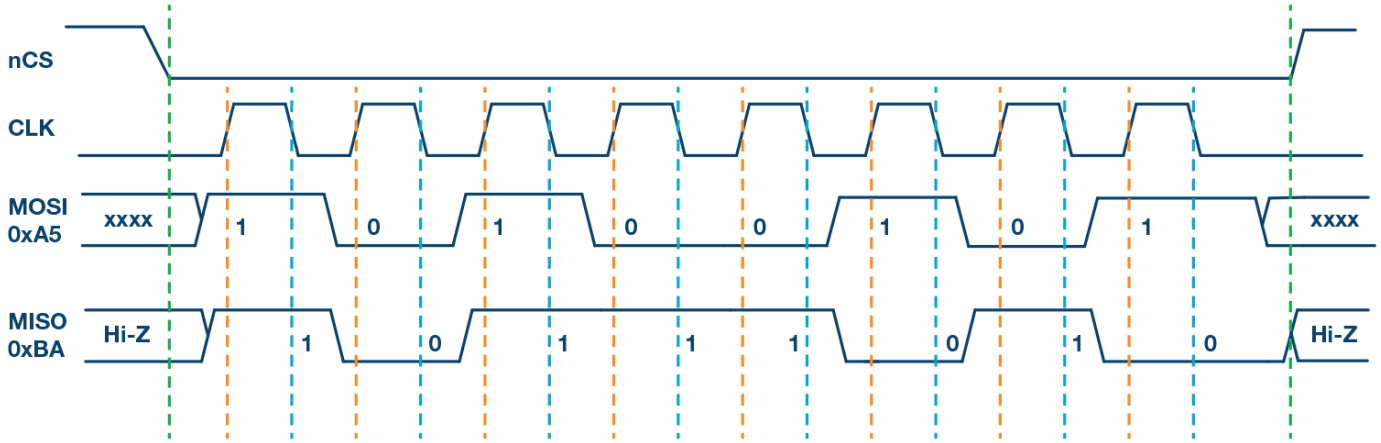


Figure 1. SPI Mode 0, CPOL = 0, CPHA = 0: CLK idle state = low, data sampled on rising edge and shifted on falling edge.

Figure 2 shows the timing diagram for SPI Mode 1. In this mode, clock polarity is 0, which indicates that the idle state of the clock signal is low. The clock phase in this mode is 1, which indicates that the data is sampled on the falling edge (shown by the orange dotted line) and the data is shifted on the rising edge (shown by the dotted blue line) of the clock signal.

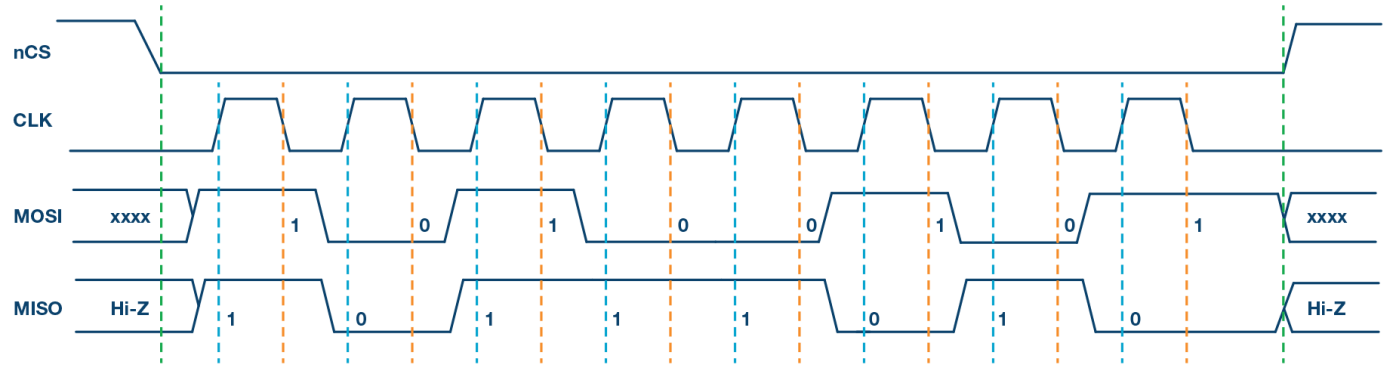


Figure 2. SPI Mode 1, CPOL = 0, CPHA = 1: CLK idle state = low, data sampled on the falling edge and shifted on the rising edge.

Figure 3 shows the timing diagram for SPI Mode 2. In this mode, the clock polarity is 1, which indicates that the idle state of the clock signal is high. The clock phase in this mode is 0, which indicates that the data is sampled on the falling edge (shown by the orange dotted line) and the data is shifted on the rising edge (shown by the dotted blue line) of the clock signal.

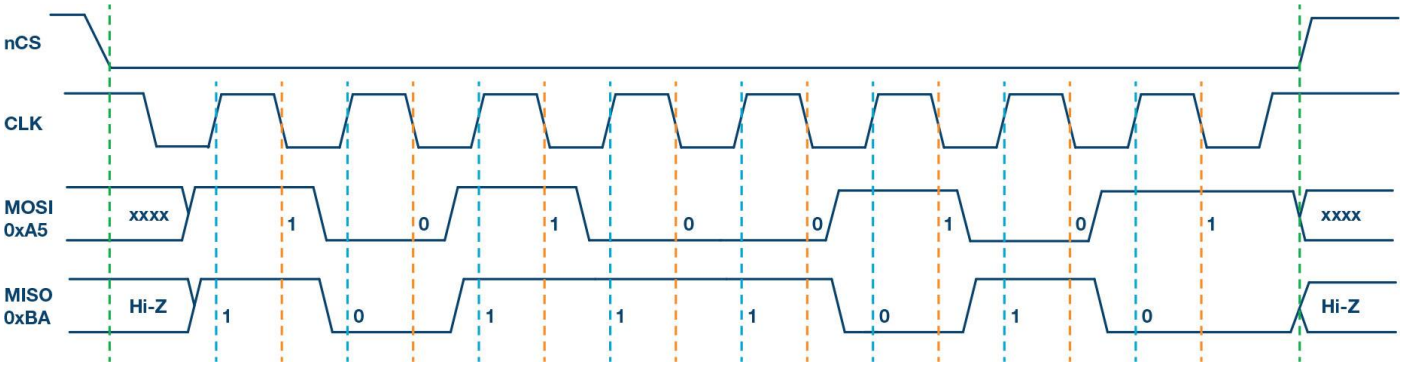


Figure 3. SPI Mode 2, CPOL = 1, CPHA = 0: CLK idle state = high, data sampled on the falling edge and shifted on the rising edge.

Figure 5 shows the timing diagram for SPI Mode 3. In this mode, the clock polarity is 1, which indicates that the idle state of the clock signal is high. The clock phase in this mode is 1, which indicates that the data is sampled on the rising edge (shown by the orange dotted line) and the data is shifted on the falling edge (shown by the dotted blue line) of the clock signal.

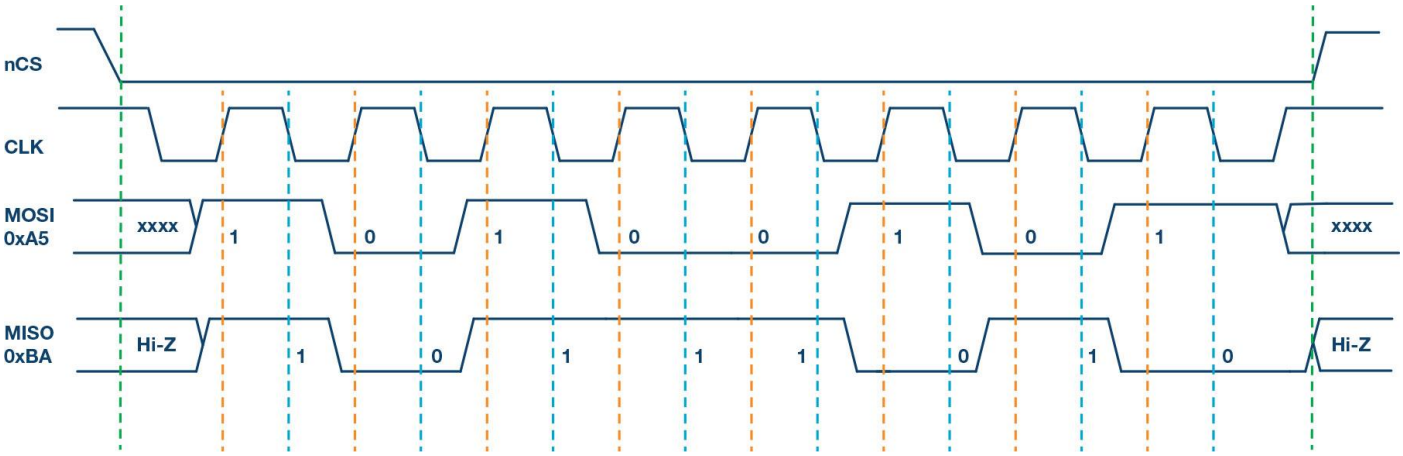


Figure 3. SPI Mode 3, CPOL = 1, CPHA = 1: CLK idle state = high, data sampled on the rising edge and shifted on the falling edge.

6. DATA TRANSMISSION IN SPI

SPI is a **full-duplex** serial protocol — meaning that data is transmitted and received at the same time over MOSI and MISO during each clock cycle.

6.1 Frame Format

A typical SPI transaction frame contains:

1. **Chip Select (CS)** activation (active low).
2. **Data bits** (commonly 8, but can be 4, 16, 32, or other lengths).
3. **Optional padding bits** (depending on device).
4. **Chip Select (CS)** deactivation.

General Sequence:

java

Copy code

CS ↓ → Clock pulses (SCLK) → Data shift (MOSI & MISO) → CS ↑

6.2 Bit Order

- **MSB First:** Most Significant Bit sent first — most common in SPI devices.
- **LSB First:** Least Significant Bit sent first — less common; must be supported by both master and slave.
- **Note:** Bit order is usually a **configurable setting** in the SPI master control register.

6.3 Data Rates and Timing

- SPI speed is determined by **SCLK frequency**.
- Typical embedded system speeds:
  - **Low-speed devices**: 100 kHz – 1 MHz (sensors, EEPROMs)
  - **Standard**: 4 MHz – 10 MHz (displays, flash memories)
  - **High-speed SPI / QSPI**: 50 MHz or more
- Maximum rate is limited by:
  - Slave device's max supported clock
  - PCB trace length and signal integrity
  - Master's clock generation capability

6.4 Example Timing Diagram (8-bit Frame, Mode 0)

markdown

Copy code

```
CS   : _______┌──────────────────────────────┐_______
             │                              │
SCLK : _______|-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_|_______
MOSI :  D7    D6    D5    D4    D3    D2    D1    D0
MISO :  Q7    Q6    Q5    Q4    Q3    Q2    Q1    Q0
```

- **D7 → D0**: Data bits from master to slave.

- **Q7 → Q0**: Data bits from slave to master.
- On **each rising edge**, both sides sample incoming data.

6.5 Full Transaction Example

Example: Master reads a register from a slave device.

1. Master pulls **CS low**.
2. Master sends a **command byte** (e.g., 0x0A) on MOSI.
3. Slave sends a **dummy byte** (ignored) on MISO.
4. Master sends a **dummy byte** (0x00) to generate clock pulses.
5. Slave sends **register data** (e.g., 0x3F) on MISO.
6. Master pulls **CS high** to end transfer.

Step-by-step:

markdown

```
CS   : ________┌──────────────────────────────────────┐________
SCLK : ________|-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-|________
MOSI : 0x0A   (Command)     0x00 (Dummy)
MISO : 0xXX   (Dummy)       0x3F (Register Data)
```

6.6 Half-Duplex vs Full-Duplex

- **Full-duplex**: Data flows on MOSI and MISO simultaneously (standard SPI).
- **Half-duplex**: Same line is used for input/output, but only one direction at a time (common in 3-wire SPI).

6.7 Engineering Notes for RTL Design

- **Shift Register Design:** Both MOSI and MISO can be handled with 8-bit shift registers, loaded at CS low, shifted on each clock pulse.
- **Clock Synchronization:** Sampling must align with CPHA setting.
- **Config Registers:** SPI master should allow setting:
  - Bit order (MSB/LSB)
  - Data frame length (e.g., 8/16/32 bits)
  - SCLK frequency
  - SPI mode (0–3)

6.8 Verification Notes

- Test with different frame sizes (not just 8 bits).
- Verify correct handling of MSB/LSB first modes.
- Check that data is valid and stable for at least half the clock period before sampling.
- Simulate bus contention by having two slaves accidentally drive MISO (should be avoided in design)

7. ADVANTAGES, DISADVANTAGES, AND LIMITATIONS OF SPI

7.1 Advantages of SPI

1. **High Data Transfer Rate**
   - SPI can achieve much higher speeds than I²C or UART (tens of Mbps), making it ideal for high-speed ADC/DAC, displays, and memory chips.
2. **Full-Duplex Communication**
   - Data can be sent and received simultaneously, increasing throughput.
3. **Simple Hardware Interface**
   - No complex addressing or start/stop conditions — just a clock and data lines.
4. **Low Protocol Overhead**
   - No need for framing bytes like UART or ACK/NACK like I²C, which reduces latency.
5. **Flexible Data Word Size**
   - Can transfer 8, 16, 32, or even custom-bit data words.
6. **Multiple Slaves Possible**
   - Using individual chip-select lines for each slave, SPI can handle many devices.

7.2 Disadvantages of SPI

1. **More Pins Required**
   - At least **4 wires** (MOSI, MISO, SCLK, CS) — more for multiple slaves.
2. **No Standardized Acknowledgement**
   - Unlike I²C, SPI has no built-in ACK/NACK — the master must implement error handling.
3. **No Built-In Flow Control**
   - If the slave is not ready, extra logic is needed to avoid data loss.
4. **Limited Slave Management**
   - Requires a dedicated chip-select line per slave unless additional logic is added.
5. **Short Distance Communication**
   - Usually works reliably only over short PCB traces (a few meters max).

7.3 Limitations of SPI

- **Not suited for long-distance communication** due to clock skew and signal degradation.
- **Not multi-master by default** — needs special design if more than one master is required.
- **Lacks addressing mechanism** — the master must explicitly select slaves using hardware pins.
- **Timing critical** — mismatched clock polarity/phase settings (CPOL, CPHA) can break communication.

7.4 SPI vs I²C vs UART — Quick Comparison Table

Feature	SPI	I²C	UART

Data Lines	4+ (MOSI, MISO, SCLK, CS per slave)	2 (SDA, SCL)	2 (TX, RX)
Speed	Very High (up to 50+ Mbps)	Medium (100 kbps – 3.4 Mbps)	Low–Medium (up to ~1 Mbps)
Full Duplex	☑ Yes	❌ No (half-duplex)	☑ Yes
Multi-Master	❌ No (needs custom logic)	☑ Yes	❌ No
Addressing	❌ None (hardware CS)	☑ 7/10-bit address	❌ None
Protocol Overhead	Low	Medium	Medium
Distance	Short (< a few meters)	Short–Medium	Long (depends on baud rate & cable)
Use Cases	Memory, sensors, displays, ADC/DAC	Sensors, EEPROM, IC communication	Serial consoles, GPS modules, modem

8. APPLICATIONS OF SPI

SPI is widely used in embedded systems because of its simplicity, speed, and flexibility. Below are common real-world applications where SPI shines:

8.1 Memory Devices (EEPROM, Flash)

- SPI is the standard interface for many **non-volatile memory chips** like EEPROM and NOR flash.
- **Why SPI?**
  - High data transfer speeds enable fast read/write cycles.
  - Supports multiple memory chips on the same bus via separate chip-selects.
- Used for:
  - Storing firmware and bootloader code in embedded devices.
  - External data storage where fast access is needed.

8.2 Sensors and Actuators

- Many sensors (temperature, pressure, accelerometers) use SPI to communicate measurements.
- SPI allows the master (e.g., microcontroller or SoC) to:
  - Query sensor data quickly.
  - Configure sensor registers via commands.
- Actuators like digital potentiometers also often use SPI for control.

8.3 Display Modules and Peripherals

- SPI is common for driving **LCD and OLED displays** especially small to medium size modules.
- Enables quick sending of image data and commands.
- Also used in devices like **touch controllers, real-time clocks (RTC), and audio codecs**.

8.4 Use in SoC and FPGA Designs

- SPI controllers are often integrated as **IP blocks** in custom SoCs or FPGA designs.
- Common uses include:
  - Interfacing with external flash for code and data storage.
  - Communicating with peripherals like sensors or DACs on the board.
  - Debugging and test access interfaces.
- The protocol's simplicity and configurability make it ideal for embedded processor subsystems like RISC-V cores.

8.5 Emerging and Specialized Applications

- **Quad SPI (QSPI)** for very high-speed flash memory used in advanced embedded systems.
- **SPI for sensor fusion** in IoT and wearable health devices.
- **SPI communication in automotive systems** for reliable, fast sensor data transfer.

9. SPI VARIANTS AND EXTENSIONS

9.1 Quad SPI (QSPI)

- QSPI extends SPI by using **4 data lines instead of 1** for MOSI and MISO combined.
- This **increases throughput up to 4x**, making it ideal for fast flash memory reads/writes.
- Uses **4-bit-wide data buses** and operates with the same clock and chip-select signals as SPI.
- Common in **high-speed NOR flash memories** for booting embedded systems.

9.2 Dual SPI

- Dual SPI uses **2 data lines** for MOSI/MISO instead of one, offering **double the data bandwidth** compared to standard SPI.
- Useful when full quad lanes aren't necessary but speed improvements over SPI are needed.

9.3 Three-wire SPI

- Combines MOSI and MISO into a **single bidirectional data line**, reducing pins from 4 to 3.
- Often used in **half-duplex** communication where data is sent or received but not simultaneously.
- Useful when pin count or wiring complexity needs to be minimized.

10. CONCLUSION AND FUTURE SCOPE

10.1 Summary of Learnings

- SPI is a simple, fast, and flexible synchronous serial protocol widely used in embedded systems.
  - The basic 4-wire SPI supports full-duplex communication with multiple slaves via dedicated chip selects.
- SPI's clock polarity and phase (CPOL, CPHA) create four standard modes that must be matched between devices.
- Variants like QSPI and Dual SPI improve throughput for demanding applications.

- SPI's simplicity makes it ideal for integrating peripherals in SoCs, including RISC-V and FPGA platforms.

10.2 SPI Evolution Trends

- Increasing data rates through multi-lane protocols like QSPI and Octal SPI.
- Integration of SPI controllers into complex SoCs and microcontrollers.
- Adoption of SPI-like protocols optimized for lower power and reduced pin count in IoT devices.
- Potential moves towards standardized flow control and error detection.

10.3 Alternatives and Next-gen Serial Protocols

- **I²C** remains popular for low-speed multi-master applications.
- **UART** for simple asynchronous serial communication.
- Emerging protocols like **MIPI I3C** offer improved features combining the best of I²C and SPI.
- High-speed serial buses like **PCIe and USB** serve very different application domains but influence serial protocol development.

REFERENCES

1. General SPI Specifications

- **Serial Peripheral Interface (SPI) Interface Specification**
  This document provides a comprehensive overview of the SPI interface, detailing its operation and applications.
  Download PDF

2. Quad SPI (QSPI) Specifications

- **Quad SPI Flash Controller Specification – OpenCores**
  OpenCores provides a detailed specification for implementing a Quad SPI Flash Controller, useful for hardware designers.
  Download PDF opencores.org
- **AXI Quad SPI v3.2 LogiCORE IP Product Guide – AMD (Xilinx)**
  This guide offers insights into the AXI Quad SPI core, including its features and integration into FPGA designs.
  Download PDF AMD
- **PMC Quad-SPI Controller Interface – AMD (Versal AI Edge)**
  Provides technical details on the Quad-SPI controller interface, focusing on performance and timing specifications.
  Read Documentation AMD Documentation+1

3. Dual SPI Specifications

- **SC16IS752/SC16IS762: Dual UART with I2C-bus/SPI Interface – NXP**
  Datasheet for dual UARTs with I2C/SPI interfaces, highlighting their capabilities and configurations.
  Download PDF

4. Three-wire SPI Specifications

- **Interfacing with 3-wire SPI – Total Phase**
  An article discussing the implementation and considerations of 3-wire SPI interfaces.
  Read Article totalphase.com+1