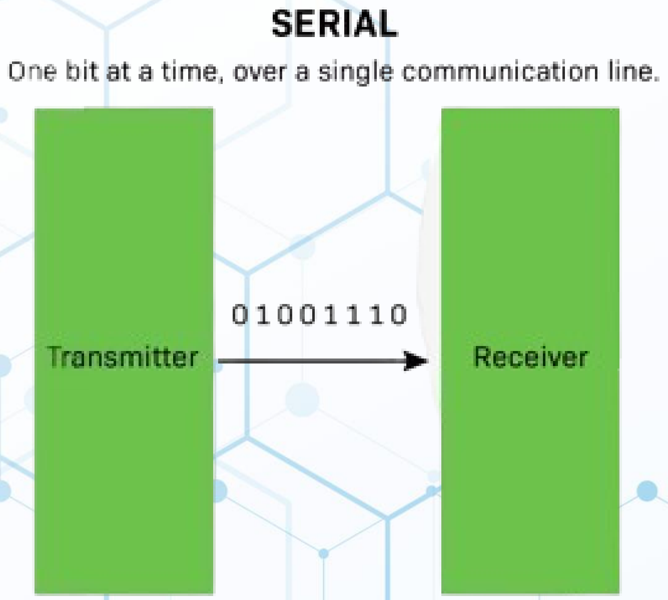
COMMUNICATION PROTOCOLS

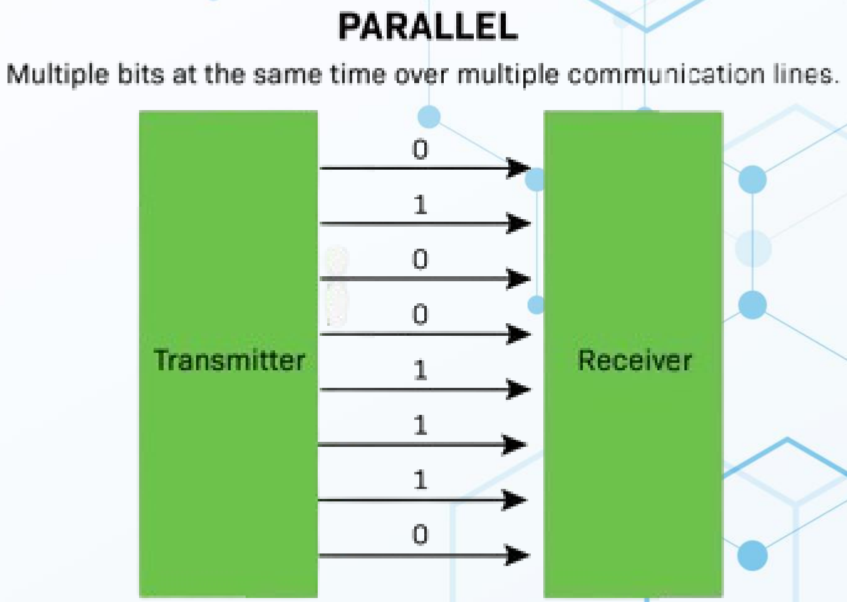
Anoop Jayaram

Embedded systems are specialized computing systems that perform dedicated functions within larger systems. Communication between different components in an embedded system—such as microcontrollers, sensors, actuators, and other peripherals—is crucial for the system's overall functionality. This communication is enabled by a set of rules and conventions known as communication protocols.

Communication protocols define how data is transmitted and received between devices, ensuring that information is correctly understood and processed by the receiving end. These protocols are essential for data exchange, synchronization, and interoperability among devices, making them a fundamental aspect of embedded system design.

SERIAL AND PARALLEL COMMUNICATION




PARALLEL
Multiple bits at the same time over multiple communication lines.
Transmitter
0
1
0
0
1
1
1
0
Receiver

In **serial communication**, data is transmitted **bit by bit** over a single or a few wires, one bit at a time. This is in contrast to parallel communication, where multiple bits are transmitted simultaneously over multiple wires.

Advantages of Serial Communication:

- **Fewer wires:** Since data is sent one bit at a time, fewer physical connections are needed.
- **Long-distance communication:** Serial communication is more reliable for long-distance data transmission, as fewer wires reduce the risk of signal degradation and interference.
- **Lower cost:** Fewer wires and simpler connectors result in lower costs.
- **Less crosstalk:** Reduced risk of crosstalk between lines, especially in high-speed applications.

Advantages of Serial Communication:

- **Crosstalk and Signal Degradation**: High-speed parallel communication is prone to signal integrity issues like crosstalk and skew, especially over long distances.
- **More Wires and Higher Costs**: Requires more data lines, making the physical connections more complex and expensive.
- **Shorter Range**: Not suitable for long-distance communication due to signal degradation.

In **parallel communication**, multiple bits are transmitted simultaneously over multiple wires. For instance, in an 8-bit parallel communication setup, 8 data lines would carry the bits at once. Parallel communication is used in situations where speed is critical and short distances are involved.

Advantages of Parallel Communication:

- **Faster data transfer**: Multiple bits are sent simultaneously, making parallel communication faster than serial communication.
- **Higher data throughput**: It is especially suitable for applications where large amounts of data need to be transferred quickly.

Disadvantages of Parallel Communication:

- **More wires**: Parallel communication requires more physical connections (e.g., 8 or 16 data lines), making it cumbersome and expensive for long-distance communication.
- **Signal degradation**: Over long distances, signals in parallel lines can suffer from skew (where bits arrive at different times due to varying delays in the wires) and crosstalk (interference between adjacent wires).
- **Higher costs**: More wires and connectors lead to increased cost and complexity.

Types of Communication Protocols

Communication protocols in embedded systems can be broadly classified into the following categories:

Serial Protocols Serial communication protocols transmit data one bit at a time over a single communication line. These protocols are simple and widely used for point-to-point communication.

1. **UART (Universal Asynchronous Receiver-Transmitter)**: UART is a widely used serial communication protocol that allows for asynchronous data transmission. It is commonly used for direct communication between two devices, such as a microcontroller and a sensor.

2. **SPI (Serial Peripheral Interface)**: SPI is a synchronous serial communication protocol used for high-speed data transfer between a master device and one or more slave devices. It is commonly employed in short-distance communication between microcontrollers and peripherals like sensors, displays, and memory devices.

3. **I2C (Inter-Integrated Circuit)**: I2C is a multi-master, multi-slave, synchronous serial communication protocol. It is widely used for communication between components on the same circuit board, such as microcontrollers, EEPROMs, and sensors. I2C is particularly useful for applications requiring communication with multiple devices using a minimal number of pins.

•**Parallel Protocols** Parallel communication protocols transmit multiple bits simultaneously across multiple lines. These protocols are used for high-speed data transfer but require more physical connections compared to serial protocols.

•**GPIO (General-Purpose Input/Output)**: GPIO pins on a microcontroller are used for basic parallel communication. These pins can be configured as inputs or outputs to control and interact with external devices. GPIOs are commonly used for simple tasks like turning devices on or off, reading sensor values, or generating control signals.

•**Wireless Protocols** Wireless communication protocols enable data exchange between devices without the need for physical connections. These protocols are increasingly important in modern embedded systems, particularly in the context of the Internet of Things (IoT).

•**Bluetooth**: Bluetooth is a short-range wireless communication protocol designed for low-power, low-cost data exchange between devices. It is commonly used in applications like wireless headphones, wearables, and smart home devices.

•**Wi-Fi**: Wi-Fi is a wireless communication protocol that allows devices to connect to the internet or a local network. In embedded systems, Wi-Fi enables remote control, data logging, and cloud connectivity.

•**Zigbee**: Zigbee is a low-power, wireless mesh networking protocol used in IoT devices. It is designed for applications requiring low data rates, low power consumption, and secure communication, such as home automation, smart lighting, and industrial monitoring.

SERIAL PERIPHERAL INTERFACE (SPI)

Serial Peripheral Interface (SPI) is a protocol that enables short-distance, synchronous serial communication between integrated circuits and other devices. It is used in many devices, including:

- SD cards
- RFID card readers
- 2.4GHz wireless transmitters and receivers
- Flash memory
- Sensors
- Real-time clocks (RTCs)
- Analog-to-digital converters

Basic Concepts of SPI

1. **Synchronous Communication**

- SPI operates in a synchronous mode, meaning that data is transmitted in sync with a clock signal shared between the communicating devices. This allows for fast and reliable data exchange.

2. **Master-Slave Architecture**

- SPI follows a master-slave architecture, where one device (the master) controls the communication process, while one or more devices (the slaves) respond to the master's commands. The master generates the clock signal, which dictates the timing of data transmission.

3. **Full-Duplex Communication**

- SPI supports full-duplex communication, meaning that data can be transmitted and received simultaneously. This is achieved using separate data lines for transmitting (MOSI) and receiving (MISO) data.

SPI Communication Lines

SPI communication typically involves four main lines:

1. **MOSI (Master Out Slave In)**

The MOSI line is used for sending data from the master to the slave.

2. **MISO (Master In Slave Out)**

The MISO line is used for sending data from the slave to the master.

3. **SCLK (Serial Clock)**

The SCLK line carries the clock signal generated by the master. This clock signal synchronizes the data transmission between the master and the slave.

4. **SS (Slave Select)**

The SS line is used by the master to select which slave device it wants to communicate with. The SS line is active low, meaning that a slave device is selected when the SS line is pulled low.

SPI Connection Diagram

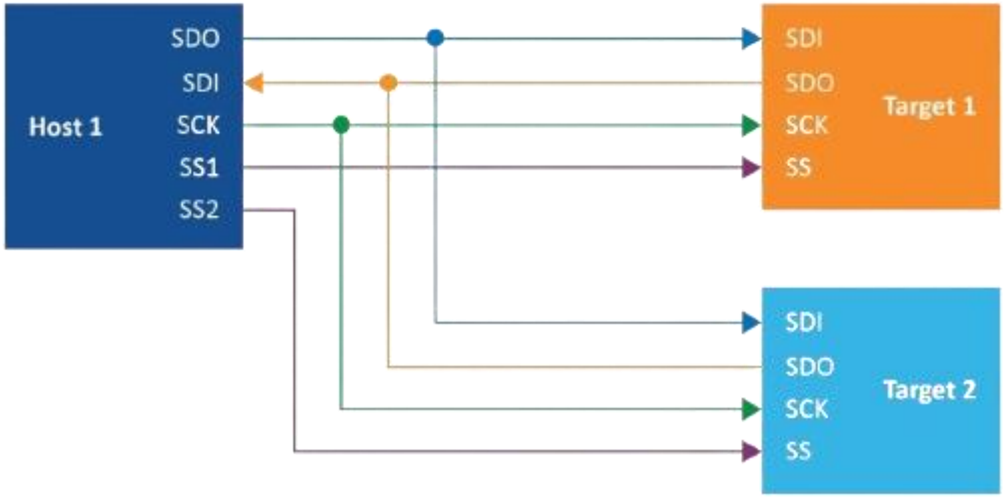
Alternative Pins

SDO – Serial Data Out

SDI – Serial Data In

SCK –Serial Clock

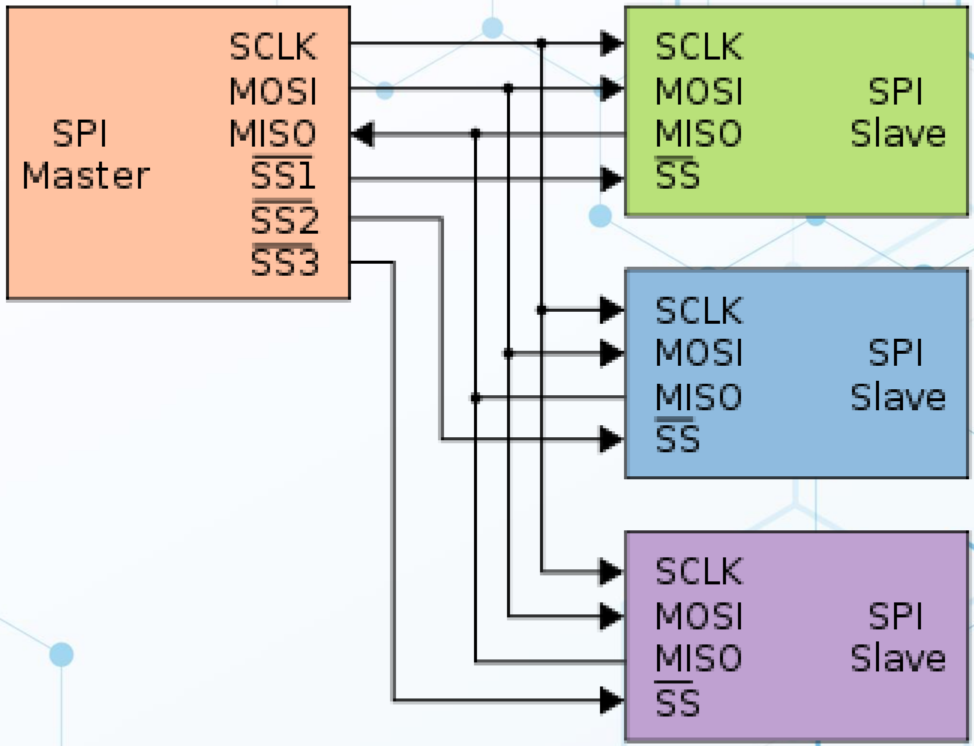
SS – Slave Select



Some peripherals have Pins named

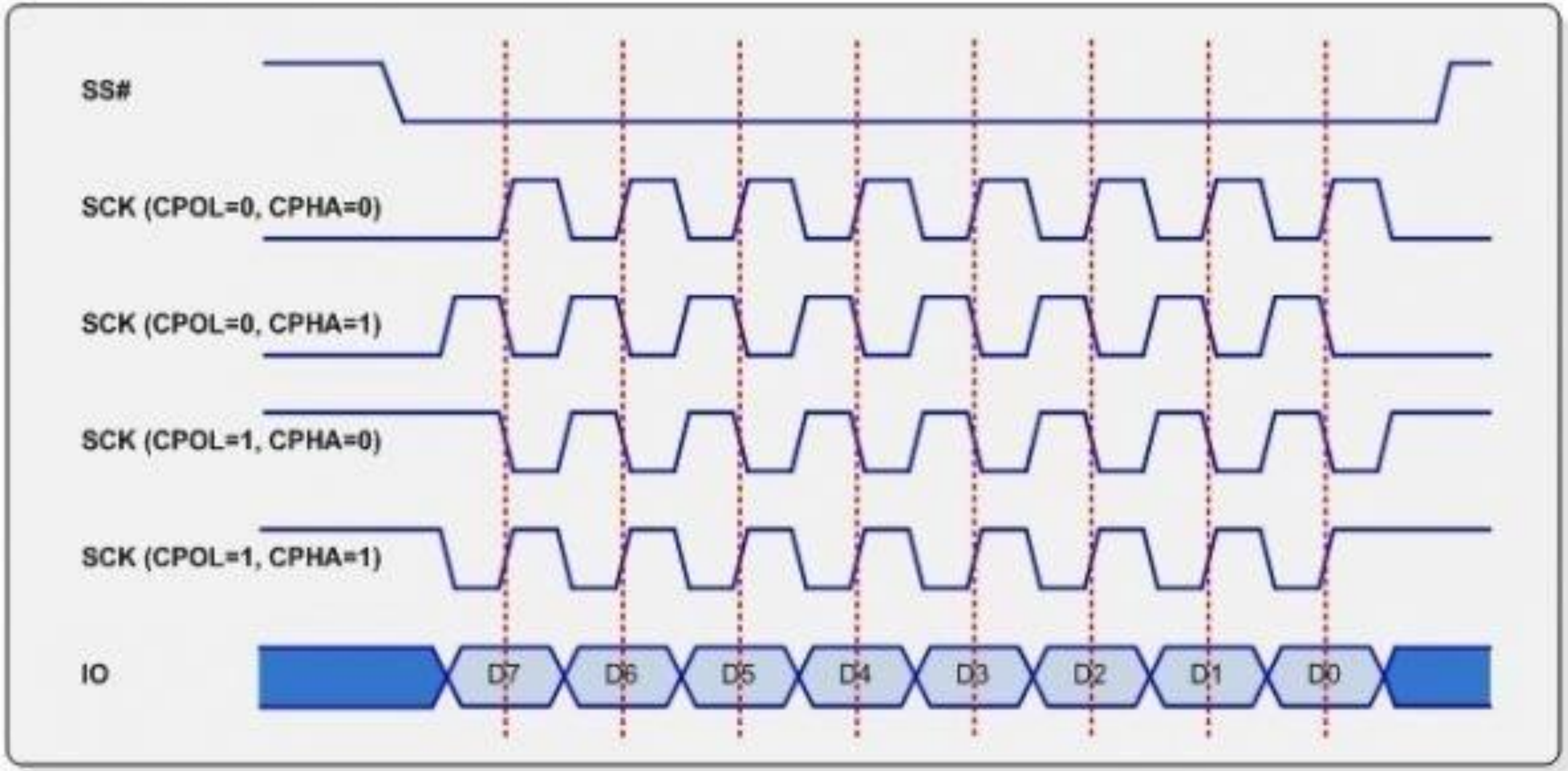
MISO – Master In Slave Out
MOSI – Master Out Slave In

Similar to SDO and SDI.



SPI Modes

SPI supports multiple communication modes, defined by two parameters: clock polarity (CPOL) and clock phase (CPHA).



These parameters determine when data is sampled and transmitted relative to the clock signal. The four possible SPI modes are:

1. **Mode 0 (CPOL = 0, CPHA = 0):**
Clock polarity is low when idle.
Data is sampled on the rising edge of the clock.

2. **Mode 1 (CPOL = 0, CPHA = 1):**
Clock polarity is low when idle.
Data is sampled on the falling edge of the clock.

3. **Mode 2 (CPOL = 1, CPHA = 0):**
Clock polarity is high when idle.
Data is sampled on the falling edge of the clock.

4. **Mode 3 (CPOL = 1, CPHA = 1):**
Clock polarity is high when idle.
Data is sampled on the rising edge of the clock.
The choice of SPI mode depends on the specific requirements of the devices involved in the communication.

Advantages of SPI

1. **High Speed:**

SPI offers high data transfer rates, making it suitable for applications requiring fast communication, such as driving high-resolution displays or reading data from high-speed sensors.

2. **Simple Hardware Interface:**

The hardware implementation of SPI is relatively simple, requiring fewer pins compared to parallel communication protocols.

3. **Full-Duplex Communication:**

SPI's full-duplex capability allows simultaneous data transmission and reception, improving communication efficiency.

4. **No Arbitration Required:**

Unlike I2C, SPI does not require arbitration since the master directly controls which slave is active using the SS line.

Disadvantages of SPI

- **No Acknowledgment Mechanism:**
  - SPI does not have a built-in acknowledgment mechanism to confirm successful data reception, which can be a drawback in some applications.
- **Limited Number of Slaves:**
  - The number of slave devices that can be connected to a single SPI bus is limited by the number of available SS lines on the master device.
- **High Pin Count:**
  - For systems with multiple slaves, the requirement for a separate SS line for each slave can result in a high pin count, complicating the hardware design.
- **Short-Distance Communication:**
  - SPI is typically used for short-distance communication due to its susceptibility to noise over longer distances.

Applications of SPI

SPI is used in a wide range of embedded systems applications, including:

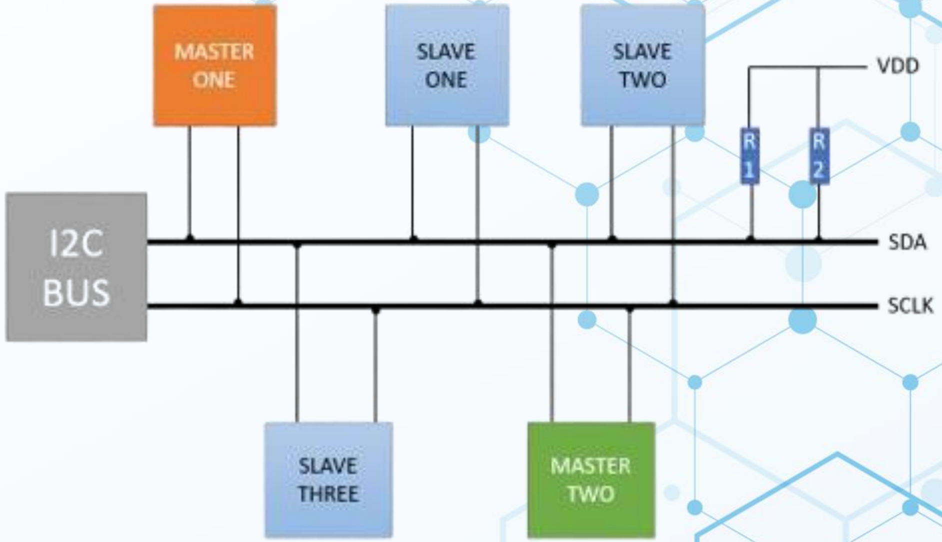
- **Sensor Interfaces:**
  - SPI is commonly used to interface with sensors that require fast data acquisition, such as accelerometers, gyroscopes, and temperature sensors.
- **Memory Devices:**
  - SPI is often used to communicate with non-volatile memory devices, such as EEPROMs, flash memory, and SD cards.
- **Displays:**
  - High-resolution displays, such as TFT LCDs and OLEDs, often use SPI for fast image data transmission.
- **Communication with Other Microcontrollers:**
  - SPI can be used for communication between multiple microcontrollers in a system, allowing for efficient data exchange in multi-processor environments.

INTER INTEGRATED CIRCUIT (I2C)

The Inter-Integrated Circuit (I2C) protocol is a widely used serial communication protocol designed for communication between multiple integrated circuits on the same circuit board.

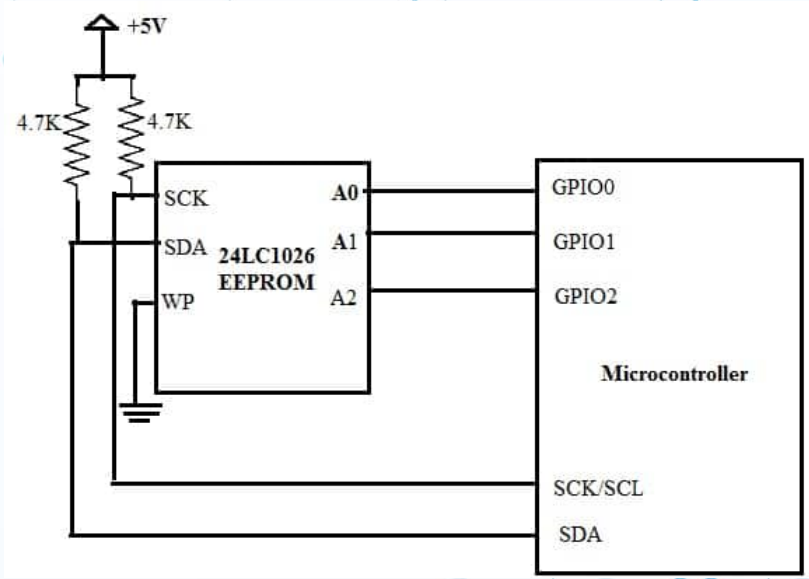
It was developed by Philips (now NXP Semiconductors) in the 1980s and has since become a standard in embedded systems for its simplicity, versatility, and efficient use of wiring.

I2C is particularly useful in applications where communication between multiple devices, such as sensors, microcontrollers, and memory chips, is required.

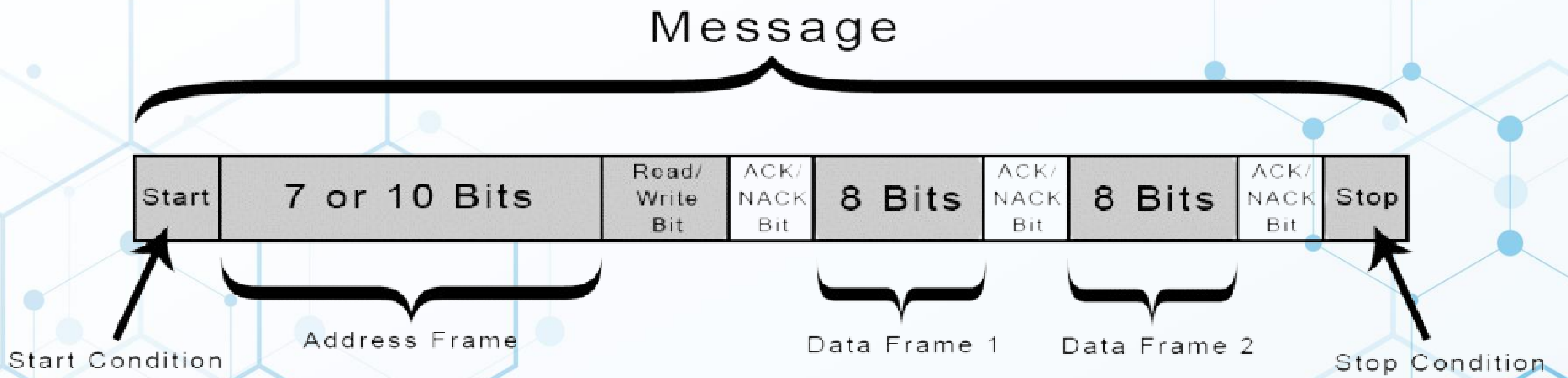


Example connection diagram of interfacing External EEPROM IC with microcontroller.

The data can be read and stored in the EEPROM using I2C.



FRAME FORMAT



The I2C communication protocol transfers data in a structured frame format consisting of several key components: start condition, address frame, data frames, and stop condition.

1. Start Condition (S)

- The **start condition** signals the beginning of a communication session.
- It occurs when the **SDA** (Serial Data Line) transitions from **high** to **low** while **SCL** (Serial Clock Line) remains **high**.
- It alerts all devices on the I2C bus that communication is about to start.

2. Address Frame

- After the start condition, an **address frame** is sent to identify the intended slave device.
- The address frame consists of:
  - **7-bit address** (or 10-bit in some cases) to uniquely identify the slave.
  - **1-bit Read/Write (R/W)** bit to indicate the operation type:
    - **0** for a write operation (master sending data to the slave).
    - **1** for a read operation (master requesting data from the slave).
- The addressed slave device will respond to this frame.

4. Data Frames

- After the address frame, multiple **data frames** (8 bits each) are transmitted, either from the master to the slave (write) or from the slave to the master (read).
- Each data frame is followed by an **ACK/NACK** bit.
- The data can be a command, configuration, or actual data to be transmitted or received.

5. Stop Condition (P)

- The **stop condition** signals the end of communication.
- It occurs when the **SDA** line transitions from **low** to **high** while **SCL** remains **high**.
- It releases the bus for other devices to initiate communication.

Summary of I2C Frame Components:

- **Start condition (S)** – Initiates communication.
- **Address frame** – 7-bit address + 1-bit R/W.
- **Acknowledge (ACK/NACK)** – Ensures data receipt.
- **Data frames** – 8-bit data blocks sent with ACK.
- **Stop condition (P)** – Ends communication.

This format allows I2C to be reliable and supports multi-device communication using minimal lines.

Pull-up resistors play a crucial role in the functioning of the I2C (Inter-Integrated Circuit) bus. The I2C protocol relies on open-drain or open-collector outputs, meaning that the devices connected to the bus can only pull the bus lines (SDA and SCL) low; they cannot drive them high. This design necessitates the use of pull-up resistors to ensure that the lines can return to a high state when no device is pulling them low.

Function of Pull-Up Resistors in I2C

- **Maintaining Logic High Level:**
  - The pull-up resistors connect the SDA (Serial Data Line) and SCL (Serial Clock Line) lines to the supply voltage (Vcc). When no device on the bus is pulling a line low, the pull-up resistors ensure that the line is pulled to a high logic level (close to Vcc). This allows the I2C bus to function correctly, as the devices rely on detecting the high or low states of the bus lines to communicate.
- **Ensuring Proper Signal Timing:**
  - In I2C communication, the timing of signals is critical. Pull-up resistors help define the rise time of the signals on the bus lines. The value of the pull-up resistors determines how quickly the lines transition from a low state to a high state when released by a device. Properly chosen pull-up resistors ensure that the rise times meet the I2C specification for the desired speed mode (e.g., standard mode, fast mode).
- **Preventing Bus Contention:**
  - Since multiple devices can be connected to the same I2C bus, pull-up resistors help prevent bus contention by ensuring that the bus lines are not left floating when no device is driving them low. This is important for avoiding undefined states on the bus, which could lead to communication errors.

Basic Concepts of I2C

- **Multi-Master, Multi-Slave Architecture**

  I2C supports a multi-master, multi-slave architecture, meaning that multiple master devices can initiate communication, and multiple slave devices can be connected to the same bus. However, only one master can control the bus at any given time.

- **Two-Wire Interface**

  I2C uses only two bidirectional lines for communication:

  - **SDA (Serial Data Line):** Carries the data between devices.
  - **SCL (Serial Clock Line):** Carries the clock signal generated by the master to synchronize the data transfer.

  The simplicity of the two-wire interface makes I2C particularly attractive for systems where minimizing the number of connections is critical.

- **Addressing Mechanism**

  Each device on the I2C bus is assigned a unique address. During communication, the master device sends the address of the target slave device, ensuring that only the intended device responds. I2C supports both 7-bit and 10-bit addressing schemes, allowing up to 127 or 1023 unique addresses, respectively.

- **Synchronous Communication**

  – I2C is a synchronous protocol, meaning that data transfer is synchronized with the clock signal provided by the master. This ensures that data is reliably transmitted and received, even over longer distances compared to some other protocols.

I2C Communication Process

The I2C communication process follows a structured sequence:

- **Start Condition:**
  Communication on the I2C bus begins with a start condition, initiated by the master. This is indicated by a transition from high to low on the SDA line while the SCL line is high. The start condition signals all devices on the bus that a transmission is about to begin.

- **Address Frame:**
  After the start condition, the master sends the 7-bit or 10-bit address of the target slave device, followed by a single read/write bit. The read/write bit indicates whether the master intends to read from (1) or write to (0) the slave device.

- **Acknowledgment (ACK/NACK):**
  After receiving the address frame, the targeted slave device responds with an acknowledgment (ACK) by pulling the SDA line low during the next clock pulse. If no device recognizes the address, the SDA line remains high, indicating a negative acknowledgment (NACK).

- **Data Transfer:**
  Data is then transferred between the master and the slave in 8-bit frames. Each byte of data is followed by an acknowledgment bit. The master can continue sending or receiving multiple bytes of data in this manner.

- **Stop Condition:**
  - Communication ends with a stop condition, initiated by the master. This is indicated by a transition from low to high on the SDA line while the SCL line is high. The stop condition releases the bus, allowing other devices to initiate communication.

I2C Speed Modes

I2C supports various speed modes to accommodate different data transfer requirements:

- **Standard Mode:**

  Operates at a maximum clock speed of 100 kHz. It is suitable for most general-purpose applications.

- **Fast Mode:**

  Operates at a maximum clock speed of 400 kHz. It is used in applications where higher data rates are required.

- **Fast Mode Plus (Fm+):**

  Operates at a maximum clock speed of 1 MHz. It is used in high-speed applications, such as displays and high-resolution sensors.

•**High-Speed Mode (Hs):**
Operates at a maximum clock speed of 3.4 MHz. It is used in very high-speed applications, though it requires special handling and compatible devices.

•**Ultra-Fast Mode:**
Operates at a maximum clock speed of 5 MHz. This mode is rarely used and is intended for specialized, high-performance applications.

Advantages of I2C

- **Simplicity and Efficiency:**

  I2C’s two-wire interface reduces the complexity of circuit design, requiring fewer connections compared to other protocols like SPI.

- **Multi-Master and Multi-Slave Support:**

  The ability to connect multiple master and slave devices on the same bus makes I2C highly versatile for complex systems.

- **Flexibility in Device Count:**

  With support for up to 127 or 1023 devices on a single bus (depending on the addressing scheme), I2C can easily accommodate a wide range of peripherals.

- **Error Checking:**

The acknowledgment mechanism allows for basic error checking during communication, ensuring data integrity.

Disadvantages of I2C

- **Slower Data Rates:**
  - Compared to protocols like SPI, I2C has lower maximum data transfer rates, which can be a limitation in applications requiring high-speed communication.
- **Bus Contention:**
  - Since I2C is a shared bus, bus contention can occur if multiple masters attempt to communicate simultaneously. While I2C includes arbitration mechanisms to resolve such conflicts, this adds complexity to the protocol.
- **Limited Bus Length:**
  - Due to the reliance on pull-up resistors and the susceptibility to noise, I2C is generally limited to short-distance communication, typically within the same circuit board.
- **Complexity in Multi-Master Systems:**
  - Implementing a multi-master system can be complex, as it requires careful management of arbitration and bus control.

Applications of I2C

I2C is used in a wide range of embedded systems applications, including:

- **Sensor Networks:**

  I2C is commonly used to interface with a variety of sensors, such as temperature sensors, pressure sensors, and accelerometers, allowing multiple sensors to share the same communication bus.
- **Real-Time Clocks (RTCs):**

  I2C is frequently used to interface with RTCs, which keep track of time in embedded systems even when the main system is powered down.
- **Memory Devices:**

  I2C is often used to communicate with EEPROMs and other non-volatile memory devices for storing configuration settings or calibration data.
- **LCD and OLED Displays:**

  I2C is commonly used to control character and graphic displays, particularly in applications where a simple and low-pin-count interface is required.
- **Microcontroller Interfacing:**

  I2C can be used for communication between multiple microcontrollers, enabling data sharing and coordination in multi-processor systems.