Serial Peripheral Interface (SPI) is the most common multi-wire serial data transmission formats that have been in use for decades, developed by Motorola in the mid of 1980 and has become a de facto standard (de facto means widely used).

SPI is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances, primarily in embedded systems. It is used to transfer data between integrated circuits using a reduced number of data lines such as shift registers, SD cards, EEPROM, RTC (Real Time Clock), ADC (Analog – to – Digital Converters), DAC (Digital – to – Analog Converters), displays like LCDs, Audio ICs, sensors like temperature and pressure, memory cards like MMC or even other microcontrollers. It can also be used for communication between two microcontrollers.  
  
  
  
  
  
  
  
  
**SPI Working Methodology**

SPI devices communicate in full duplex mode using a master-slave architecture with a single master. The master device originates the frame for reading and writing. Multiple slave devices are supported through selection with individual slave select (SS) lines. Sometimes SPI is called a *four-wire* serial bus, contrasting with three-, two-, and one-wire serial buses. The SPI may be accurately described as a synchronous serial interface,but it is different from the Synchronous Serial Interface (SSI) protocol, which is also a four-wire synchronous serial communication protocol.

In SPI protocol, the devices are connected in a Master – Slave relationship in a multi – point interface. In this type of interface, one device is considered the Master of the bus (usually a Microcontroller) and all the other devices (peripheral ICs or even other Microcontrollers) are considered as slaves.

In SPI protocol, there can be only one master but many slave devices.

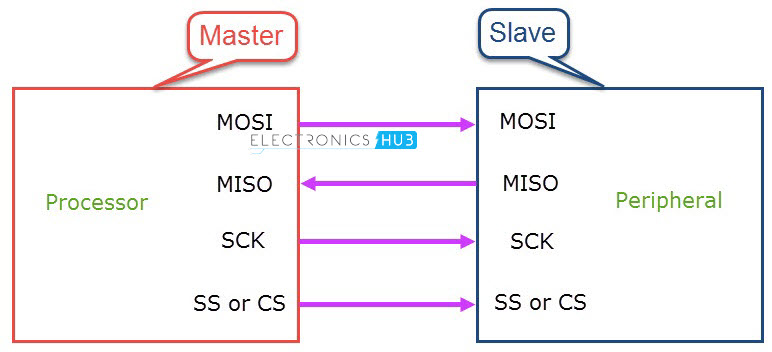
The SPI bus consists of 4 signals or pins. They are

* Master – Out / Slave – In (MOSI)
* Master – In / Slave – Out (MISO)
* Serial Clock (SCLK) and
* Chip Select (CS) or Slave Select (SS)

**NOTE:** Different manufacturers uses different nomenclature for the SPI bus. Refer the data sheet for exact information.

Since, the SPI bus is implemented using 4 signals or wires, it is sometimes called as Four Wire Interface. Let us first see a simple interface between a single master and single slave that are connected using SPI protocol and then we will explain about the 4 wires.

The following image depicts a Master (Processor) connected to a Slave (Peripheral) using SPI bus.



**Master – Out / Slave – In** or MOSI, as the name suggests, is the data generated by the Master and received by the Slave. Hence, MOSI pins on both the master and slave are connected together. Master – In / Slave – Out or MISO is the data generated by Slave and must be transmitted to Master.

**MISO** pins on both the master and slave are ties together. Even though the Signal in MISO is produced by the Slave, the line is controlled by the Master. The Master generates a clock signal at SCLK and is supplied to the clock input of the slave. Chip Select (CS) or Slave Select (SS) is used to select a particular slave by the master.

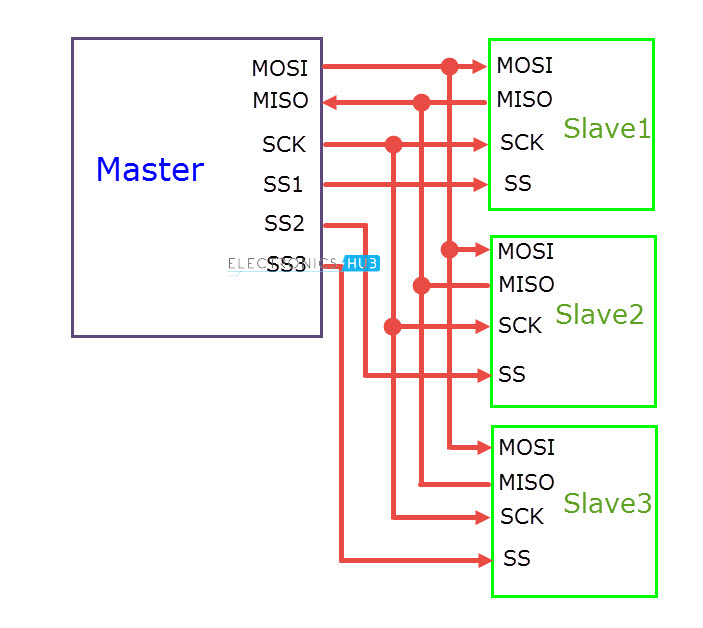
Since the clock is generated by the Master, the flow of data is controlled by the master. For every clock cycle, one bit of data is transmitted from master to slave and one bit of data is transmitted from slave to master.

This process happen simultaneously and after 8 clock cycles, a byte of data is transmitted in both directions and hence, SPI is a full – duplex communication.

If the data has to be transmitted by only one device, then the other device has to send something (even garbage or junk data) and it is up to the device whether the transmitted data is actual data or not.

This means that for every bit transmitted by one device, the other device has to send one bit data i.e. the Master simultaneously transmits data on MOSI line and receive data from slave on MISO line.

If the slave wants to transmit the data, the master has to generate the clock signal accordingly by knowing when the slave wants to send the data in advance. If more than one slave has to be connected to the master, then the setup will be something similar to the following image.

Even though multiple slaves are connected to the master in the SPI bus, only one slave will be active at any time. In order to select the slave, the master will pull down the SS (Slave Select) or CS (Chip Select) line of the corresponding slave.

Hence, there must by a separate CS pin on the Master corresponding to each of the slave device. We need to pull down the SS or CS line to select the slave because this line is active low.  
  
**SPI Modes of Operation**

We have already seen that it is the job of the Master device to generate the clock signal and distribute it to the slave in order to synchronise the data between master and slave. The work of master doesn’t end at generating clock signal at a particular frequency.

In fact, the master and slave have to agree on certain synchronization protocols. For this, two features of the clock i.e. the Clock Polarity (CPOL or CKP) and Clock Phase (CPHA) come in to picture.

Clock Polarity determines the state of the clock. When CPOL is LOW, the clock generated by the Master i.e. SCK is LOW when idle and toggles to HIGH during active state (during a transfer). Similarly, when CPOL is HIGH, SCK is HIGH during idle and LOW during active state.

Clock Phase determines the clock transition i.e. rising (LOW to HIGH) or falling (HIGH to LOW), at which the data is transmitted. When CPHA is 0, the data is transmitted on the rising edge of the clock. Data is transmitted on the falling edge when CPHA is 1.

Depending on the values of Clock Polarity (CPOL) and Clock Phase (CPHA), there are 4 modes of operation of SPI: Modes 0 through 3.

#### **Mode 0:**

Mode 0 occurs when Clock Polarity is LOW and Clock Phase is 0 (CPOL = 0 and CPHA = 0). During Mode 0, data transmission occurs during rising edge of the clock.

#### **Mode 1:**

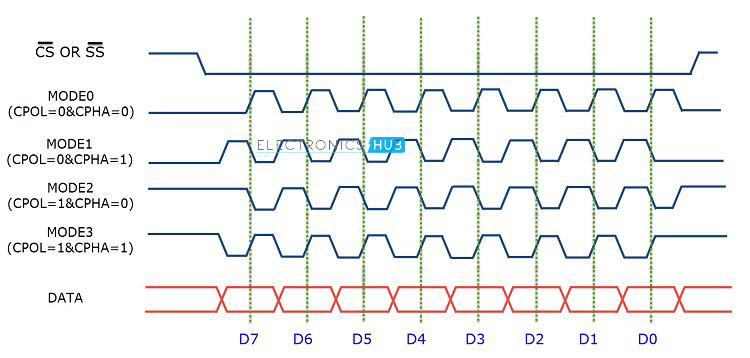
Mode 1 occurs when Clock Polarity is LOW and Clock Phase is 1 (CPOL = 0 and CPHA = 1). During Mode 1, data transmission occurs during falling edge of the clock.

#### **Mode 2:**

Mode 2 occurs when Clock Polarity is HIGH and Clock Phase is 0 (CPOL = 1 and CPHA = 0). During Mode 2, data transmission occurs during rising edge of the clock.

#### **Mode 3:**

Mode 3 occurs when Clock Polarity is HIGH and Clock Phase is 1 (CPOL = 1 and CPHA = 1). During Mode 3, data transmission occurs during rising edge of the clock.

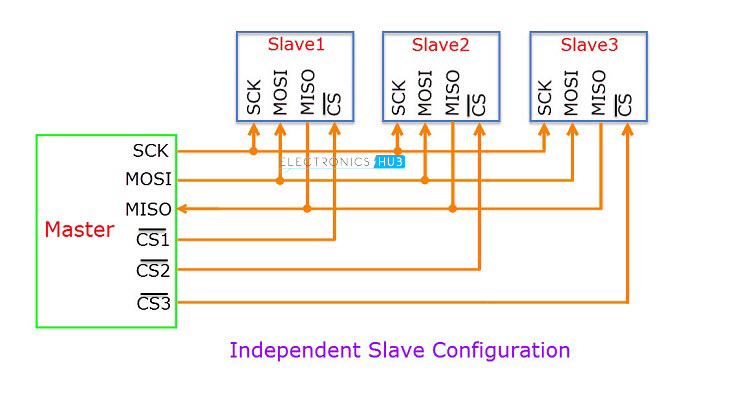


### **SPI Configurations**

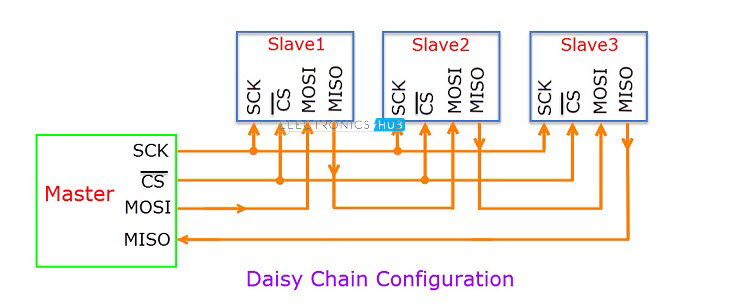
There are two types of configurations in which the SPI devices can be connected in an SPI bus. They are Independent Slave Configuration and Daisy Chain Configuration.

In Independent Slave Configuration, the master has dedicated Slave Select Lines for all the slaves and each slave can be selected individually. All the clock signals of the slaves are connected to the master SCK.

Similarly, all the MOSI pins of all the slaves are connected to the MOSI pin of the master and all the MISO pins of all the slaves are connected to the MISO pin of the master.

In Daisy Chain Configuration, only a single Slave Select line is connected to all the slaves. The MOSI of the master is connected to the MOSI of slave 1. MISO of slave 1 is connected to MOSI of slave 2 and so on. The MISO of the final slave is connected to the MISO of the master.

Consider the master transmits 3 bytes of data in to the SPI bus. First, the 1st byte of data is shifted to slave 1. When the 2nd byte of data reaches slave 1, the first byte is pushed in to slave 2.

Finally, when the 3rd byte of data arrives in to the first slave, the 1st byte of data is shifted to slave 3 and the second byte of data is shifted in to second slave.If the master wants to retrieve information from the slaves, it has to send 3 bytes of junk data to the slaves so that the information in the slaves comes to the master.

### **Applications of SPI**

* Memory: SD Card , MMC , EEPROM , Flash
* Sensors: Temperature and Pressure
* Control Devices: ADC , DAC , digital POTS and Audio Codec.
* Others: Camera Lens Mount, touchscreen, LCD, RTC, video game controller, etc.

### **Advantages**

* SPI is very simple to implement and the hardware requirements are not that complex.
* Supports full – duplex communication at all times.
* Very high speed of data transfer.
* No need for individual addresses for slaves as CS or SS is used.
* Only one master device is supported and hence there is no chance of conflicts.
* Clock from the master is configured based on speed of the slave and hence slave doesn’t have to worry about clock.

### **Disadvantages**

* Each additional slave requires an additional dedicated pin on master for CS or SS.
* There is no acknowledgement mechanism and hence there is no confirmation of receipt of data.
* Slowest device determines the speed of transfer.
* There are no official standards and hence often used in application specific implementations.
* There is no flow control.

**I²C** (**Inter-Integrated Circuit**), pronounced *I-squared-C*, is a synchronous, multi-master, multi-slave, packet switched, single-ended, serial computer bus invented in 1982 by Philips Semiconductor (now NXP Semiconductors). It is widely used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication. Alternatively I²C is spelled **I2C**(pronounced I-two-C) or **IIC** (pronounced I-I-C).

Since October 10, 2006, no licensing fees are required to implement the I²C protocol. However, fees are required to obtain I²C slave addresses allocated by NXP.

Several competitors, such as Siemens AG (later Infineon Technologies AG, now Intel mobile communications), NEC, Texas Instruments, STMicroelectronics (formerly SGS-Thomson), Motorola (later Freescale, now merged with NXP),Nordic Semiconductor and Intersil, have introduced compatible I²C products to the market since the mid-1990s.

SMBus, defined by Intel in 1995, is a subset of I²C, defining a stricter usage. One purpose of SMBus is to promote robustness and interoperability. Accordingly, modern I²C systems incorporate some policies and rules from SMBus, sometimes supporting both I²C and SMBus, requiring only minimal reconfiguration either by commanding or output pin use.

The original communication speed was defined with a maximum of 100 kbit per second and many applications don’t require faster transmissions. For those that do there is a 400 kbit fast mode and – since 1998 – a high speed 3.4 Mb option available. Recently, fast mode plus a transfer rate between this has been specified. Beyond this, there is the ultra fast mode UFM, but it frankly is no real I2C bus.

I2C is not only used on single boards but also to connect components which are linked via cable. Simplicity and flexibility are key characteristics that make this bus attractive to many applications.

**Most significant features include:**

* Only two bus lines are required
* No strict baud rate requirements like for instance with RS232, the master generates a bus clock
* Simple master/slave relationships exist between all components
* Each device connected to the bus is software-addressable by a unique address
* I2C is a true multi-master bus providing arbitration and collision detection

## 

## 

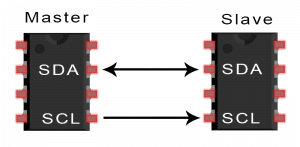
## 

## 

## 

I2C combines the best features of SPI and UARTs. With I2C, you can connect multiple slaves to a single master (like SPI) and you can have multiple masters controlling single, or multiple slaves. This is really useful when you want to have more than one microcontroller logging data to a single memory card or displaying text to a single LCD.

Like UART communication, I2C only uses two wires to transmit data between devices:

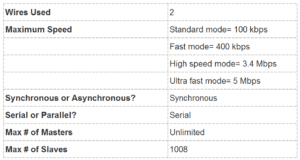


**SDA (Serial Data)** – The line for the master and slave to send and receive data.

**SCL (Serial Clock)** – The line that carries the clock signal.

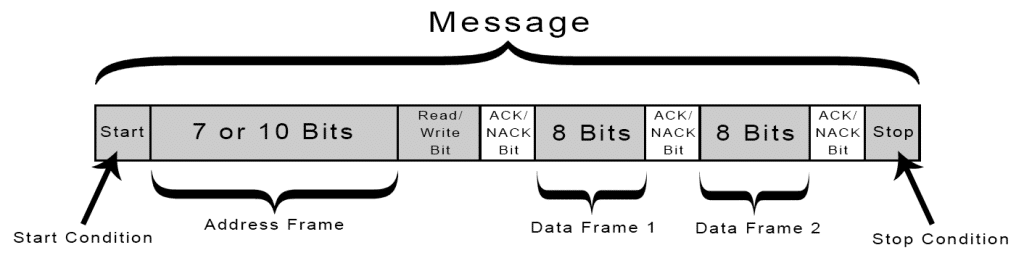
I2C is a serial communication protocol, so data is transferred bit by bit along a single wire (the SDA line).

Like SPI, I2C is synchronous, so the output of bits is synchronized to the sampling of bits by a clock signal shared between the master and the slave. The clock signal is always controlled by the master.



## **I²C Working Methodology**

With I2C, data is transferred in *messages.* Messages are broken up into *frames* of data. Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted. The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame:



**Start Condition:** The SDA line switches from a high voltage level to a low voltage level *before* the SCL line switches from high to low.

**Stop Condition:** The SDA line switches from a low voltage level to a high voltage level *after* the SCL line switches from low to high.

**Address Frame:** A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it.

**Read/Write Bit:** A single bit specifying whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).

**ACK/NACK Bit:** Each frame in a message is followed by an acknowledge/no-acknowledge bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device.

### **ADDRESSING**

I2C doesn’t have slave select lines like SPI, so it needs another way to let the slave know that data is being sent to it, and not another slave. It does this by *addressing*. The address frame is always the first frame after the start bit in a new message.

The master sends the address of the slave it wants to communicate with to every slave connected to it. Each slave then compares the address sent from the master to its own address. If the address matches, it sends a low voltage ACK bit back to the master. If the address doesn’t match, the slave does nothing and the SDA line remains high.

### 

### **READ/WRITE BIT**

The address frame includes a single bit at the end that informs the slave whether the master wants to write data to it or receive data from it. If the master wants to send data to the slave, the read/write bit is a low voltage level. If the master is requesting data from the slave, the bit is a high voltage level.

### **THE DATA FRAME**

After the master detects the ACK bit from the slave, the first dataframe is ready to be sent.

The data frame is always 8 bits long, and sent with the most significant bit first. Each data frame is immediately followed by an ACK/NACK bit to verify that the frame has been received successfully. The ACK bit must be received by either the master or the slave (depending on who is sending the data) before the next data frame can be sent.

After all of the data frames have been sent, the master can send a stop condition to the slave to halt the transmission. The stop condition is a voltage transition from low to high on the SDA line after a low to high transition on the SCL line, with the SCL line remaining high.

## 

## 

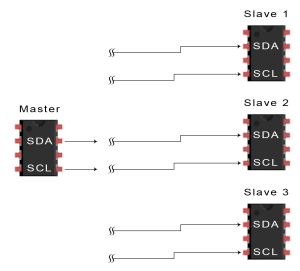
## 

## 

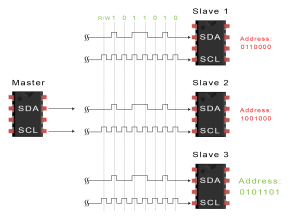
## 

## **STEPS OF I2C DATA TRANSMISSION**

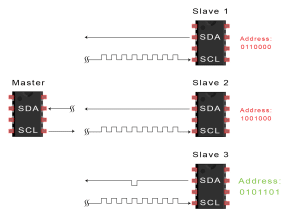
1. The master sends the start condition to every connected slave by switching the SDA line from a high voltage level to a low voltage level *before* switching the SCL line from high to low:



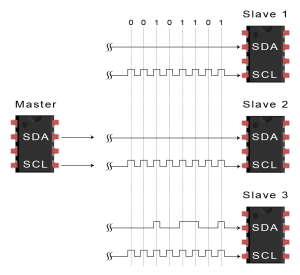
2. The master sends each slave the 7 or 10 bit address of the slave it wants to communicate with, along with the read/write bit:



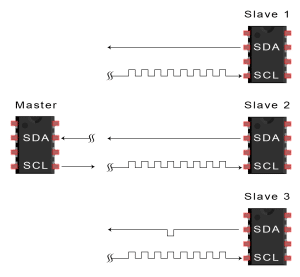
3. Each slave compares the address sent from the master to its own address. If the address matches, the slave returns an ACK bit by pulling the SDA line low for one bit. If the address from the master does not match the slave’s own address, the slave leaves the SDA line high.



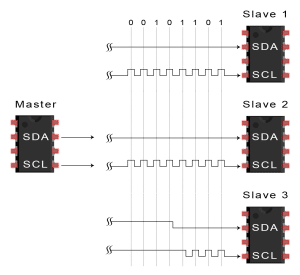
4. The master sends or receives the data frame:



5. After each data frame has been transferred, the receiving device returns another ACK bit to the sender to acknowledge successful receipt of the frame:

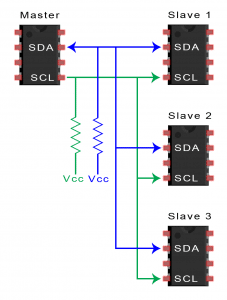


6. To stop the data transmission, the master sends a stop condition to the slave by switching SCL high before switching SDA high:



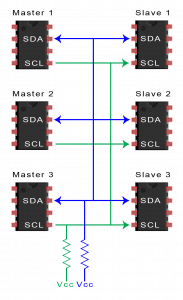
## **SINGLE MASTER WITH MULTIPLE SLAVES**

Because I2C uses addressing, multiple slaves can be controlled from a single master. With a 7 bit address, 128 (27) unique address are available. Using 10 bit addresses is uncommon, but provides 1,024 (210) unique addresses. To connect multiple slaves to a single master, wire them like this, with 4.7K Ohm pull-up resistors connecting the SDA and SCL lines to Vcc:



## **MULTIPLE MASTERS WITH MULTIPLE SLAVES**

Multiple masters can be connected to a single slave or multiple slaves. The problem with multiple masters in the same system comes when two masters try to send or receive data at the same time over the SDA line. To solve this problem, each master needs to detect if the SDA line is low or high before transmitting a message. If the SDA line is low, this means that another master has control of the bus, and the master should wait to send the message. If the SDA line is high, then it’s safe to transmit the message. To connect multiple masters to multiple slaves, use the following diagram, with 4.7K Ohm pull-up resistors connecting the SDA and SCL lines to Vcc:



## 

## 

## **ADVANTAGES AND DISADVANTAGES OF I2C**

There is a lot to I2C that might make it sound complicated compared to other protocols, but there are some good reasons why you may or may not want to use I2C to connect to a particular device:

### **ADVANTAGES**

* Only uses two wires
* Supports multiple masters and multiple slaves
* ACK/NACK bit gives confirmation that each frame is transferred successfully
* Hardware is less complicated than with UARTs
* Well known and widely used protocol

### **DISADVANTAGES**

* Slower data transfer rate than SPI
* The size of the data frame is limited to 8 bits
* More complicated hardware needed to implement than SPI

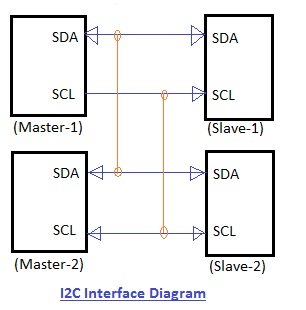
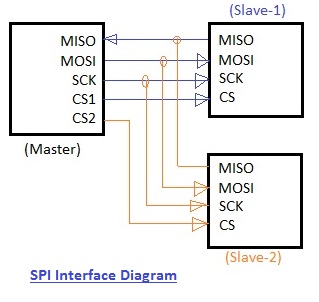
#### **The difference between I2C and SPI Communication Protocol**

|  |  |
| --- | --- |
| **I2C** | **SPI** |
| I2C can be multi-master and multi-slave, which means there can be more than one master and slave attached to the I2C bus | SPI can be multi-save but does not a multi-master serial protocol, that means there can be only one master attached to SPI bus. |
| I2C is half-duplex communication protocol. | SPI is a full duplex communication protocol. |
| I2C has the feature of clock stretching, that means if the slave cannot able to send fast data as fast enough then it suppresses the clock to stop the communication. | Clock stretching is not the feature of SPI. |
| I2C is used only two wire for the communication, one wire is used for the data and the second wire is used for the clock. | SPI needs three or four wire for communication ((depends on requirement), MOSI, MISO, SCL and Chip-select pin. |
| I2C is slower than SPI. | In comparison to I2C, SPI is faster. |
| I2C draws more power than SPI. | Draws less power as compared to I2C. |
| I2C is less susceptible to noise than SPI | SPI is more susceptible to noise than I2C. |
| I2C is cheaper to implement than the SPI communication protocol. | Costly as compare to I2C. |
| I2C work on wire and logic and it has a pull-up resistor. | There is no requirement of pull-up resistor in case of the SPI. |
| In I2C communication we get the acknowledgment bit after each byte. | Acknowledgment bit is not supported by the SPI communication protocol. |
| I2C ensures that data sent is received by the slave device. | SPI does not verify that data is received correctly or not. |
| I2C support the multi-master communication. | SPI does not support multi -master communication. |
| I2C is a multi-master communication protocol that’s why it has the feature of arbitration. | SPI is not a multi-master communication protocol, so it does not consist the properties of arbitration. |
| I2C is the address base bus protocol, you have to send the address of the slave for the communication. | In case of the SPI, you have to select the slave using the slave select pin for the communication. |
| I2C has some extra overhead due to start and stop bits. | SPI does not have a start and stop bits. |
| I2C supports multiple devices on the same bus without any additional select lines (work on the basis of device address). | SPI requires additional signal (slave select lines) lines to manage multiple devices on the same bus. |
| I2C is better for long distance. | SPI is better for the short distance. |
| I2C is developed by NXP. | SPI is developed by Motorola. |

difference between SPI and I2C interfaces.

|  |  |  |
| --- | --- | --- |
| **Specifications** | **SPI** | **I2C** |
| Number of Bus lines | Four ( SCLK, CS, MISO, MOSI) | Two ( SDA, SCL) |
| data rate | Higher, about 10 MHz or more | Lower, about 100 KHz or 400 KHz |
| Run current | Lower, about 200 µA at 4 Mbps, hence less power consumption | Higher, about 400 µA at 400 Kbps, hence more power consumption |
| Preferable application | Perform well in single master and single slave configuration | Perform well in multi-master and multi-slave configuration |
| Device addressing | Does not support | Support |
| Acknowledgement (ACK) mechanism | Does not have mechanism to support receipt of the data | It has mechanism to support receipt of the data using ACK |
| Overhead in point to point connection | Less | More |
| Application | applications requiring continuous data stream transmission | communication between devices which does not require continuous transmission i.e. requiring occasional communication |

### **SPI Interface I2C interface**

****