Islamic University of Technology

EEE 4483
Digital Electronics & Pulse Techniques

Lecture- 14

Serial-Parallel Communication

Communication between electronic devices is like communication between humans. Both sides need to speak the same language. In electronics, these languages are called communication protocols.

Electronic devices talk to each other by sending bits of data through wires physically connected between devices. A bit is like a letter in a word, except instead of the 26 letters (in the English alphabet), a bit is binary and can only be a **1** or **0**. Bits are transferred from one device to another by quick changes in voltage. In a system operating at 5 V, a **0** bit is communicated as a short pulse of 0 V, and a **1** bit is communicated by a short pulse of 5 V.

The bits of data can be transmitted either in parallel or serial form. In parallel communication, the bits of data are sent all at the same time, each through a separate wire. The following diagram (Fig 2.) shows the parallel transmission of the letter "**C**" [ASCII – 43 –] in binary (**01000011**);

In serial communication, the bits are sent one by one through a single wire. The following diagram (Fig. 1) shows the serial transmission of the letter "**C**" in binary (**01000011**):

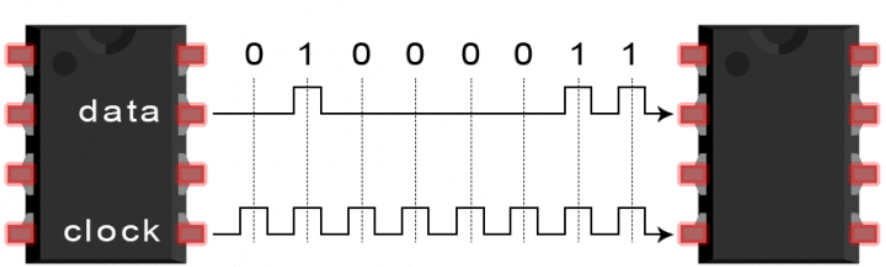


Fig 1. Serial communication

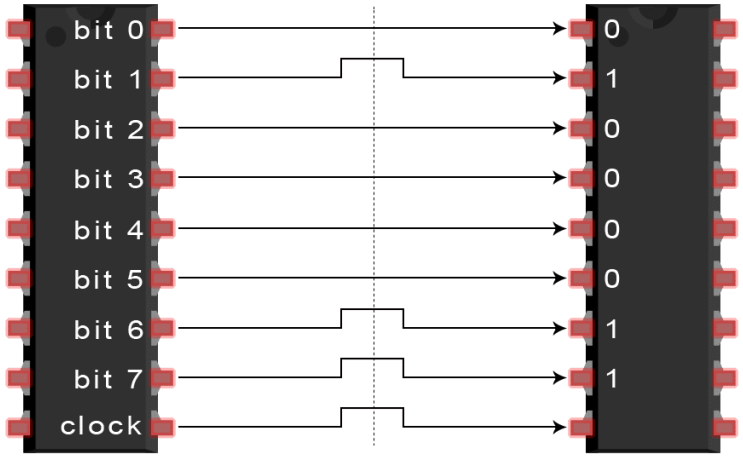
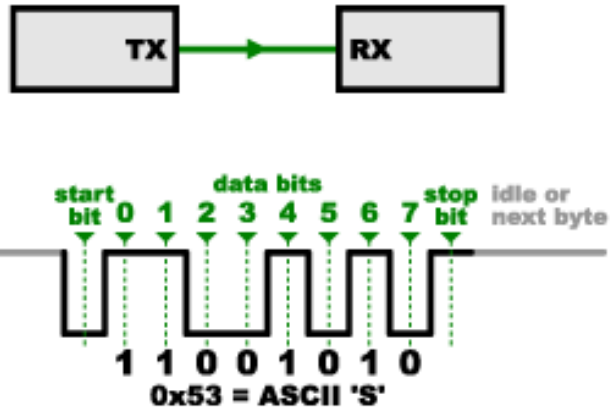


Fig 2. Parallel communication

Difficulties with Serial Ports

A common serial port, the kind with TX and RX lines, is called "*asynchronous*" (not synchronous) because there is no control over when data is sent or any guarantee that both sides are running at precisely the same rate. Since computers normally rely on everything being synchronized to a single "clock" (the main crystal attached to a computer that drives everything), this can be a problem when two systems with slightly different clocks try to communicate with each other.

To work around this problem, asynchronous serial connections add extra start and stop bits to each byte help the receiver sync up to data as it arrives. Both sides must also agree on the transmission speed (such as 9600 bits per second) in advance. Slight differences in the transmission rate aren't a problem because the receiver re-syncs at the start of each byte.



Asynchronous serial works just fine, **but has a lot of overhead in both the extra start and stop bits sent with every byte, and the complex hardware required** to send and receive data.

SPI Communication

The Serial Peripheral Interface (SPI) is used to transfer data between integrated circuits using a reduced number of data lines.

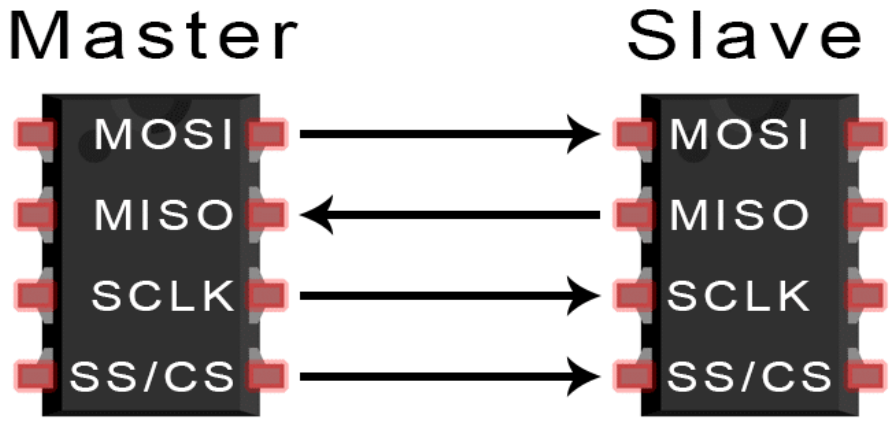
SPI is a common communication protocol used by many different devices. For example, SD card modules, RFID card reader modules, and 2.4 GHz wireless transmitter/receivers all use SPI to communicate with microcontrollers. One unique benefit of SPI is the fact that data can be transferred without interruption. Any number of bits can be sent or received in a continuous stream.

Key Features:

- ❑ Read, write and full-duplex (simultaneous read/write) data transactions;
- ❑ SPI master and SPI slave configuration;
- ❑ Configurable SPI bus clock polarity, phase and frequency;
- ❑ Configurable number of bits to transfer;
- ❑ Communication with serial external devices (ADC and DAC, RTC, temperature and pressure sensors, LCD controller, etc.).

SPI Communication : continued ..

Devices communicating via SPI are in a master-slave relationship. The master is the controlling device (usually a microcontroller), while the slave (usually a sensor, display, or memory chip) takes instruction from the master. The simplest configuration of SPI is a single master, single slave system, but one master can control more than one slave (more on this below).



MOSI (Master Output/Slave Input) – Line for the master to send data to the slave.

MISO (Master Input/Slave Output) – Line for the slave to send data to the master.

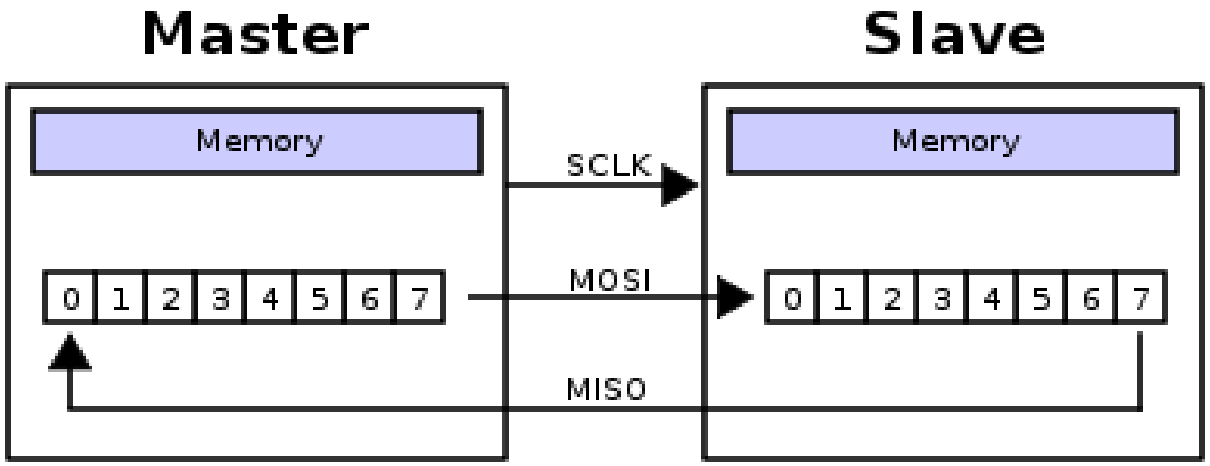
SCLK (Clock) – Line for the clock signal.

SS/CS (Slave Select/Chip Select) – Line for the master to select which slave to send data to.

SPI Working Principle

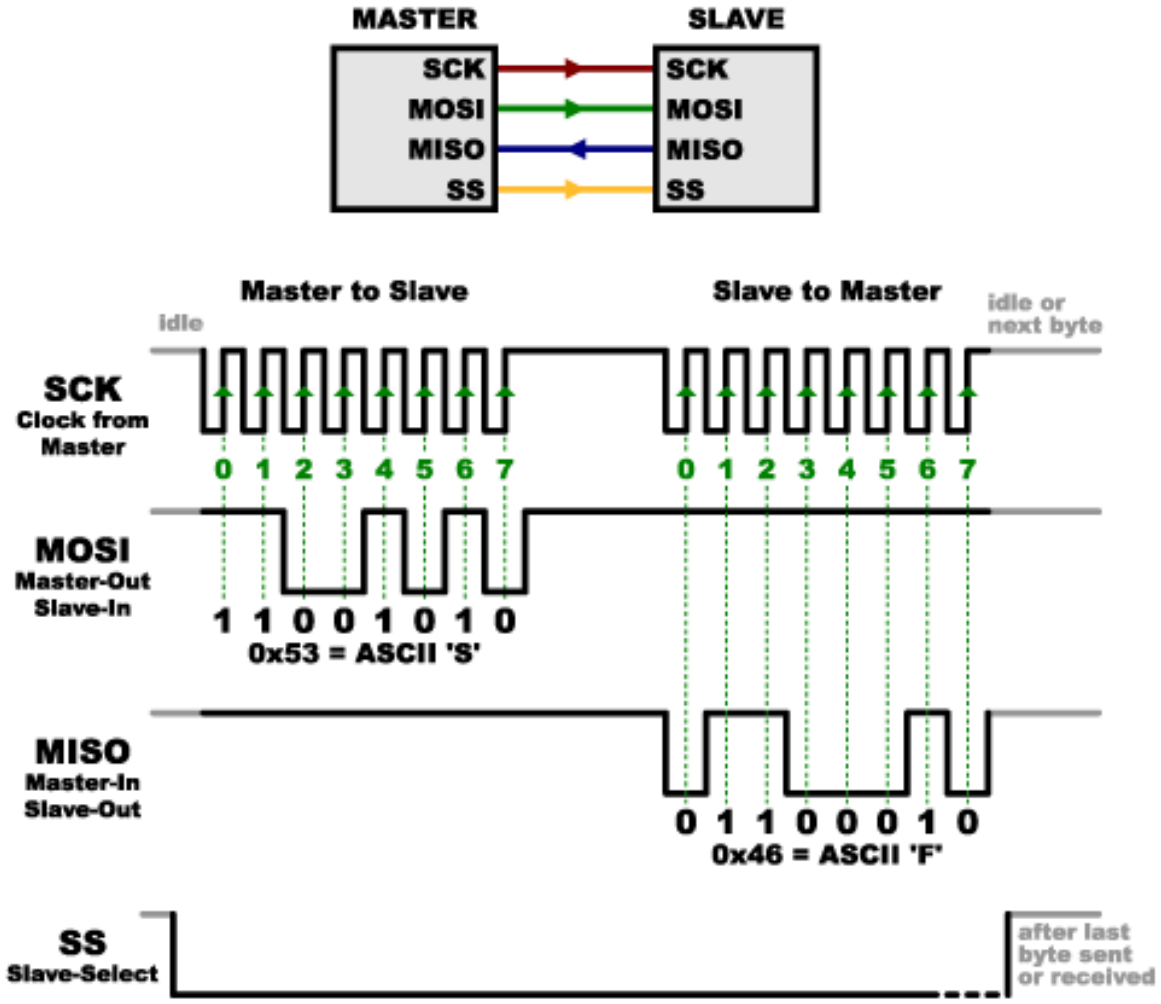
In SPI, only one side generates the clock signal (usually called CLK or SCK for Serial Clock). The side that generates the clock is called the “master”, and the other side is called the “slave”.

The master sends data to the slave bit by bit, in serial through the MOSI line. The slave receives the data sent from the master at the MOSI pin. Data sent from the master to the slave is usually sent with the most significant bit first. The slave can also send data back to the master through the MISO line in serial. The data sent from the slave back to the master is usually sent with the least significant bit first.



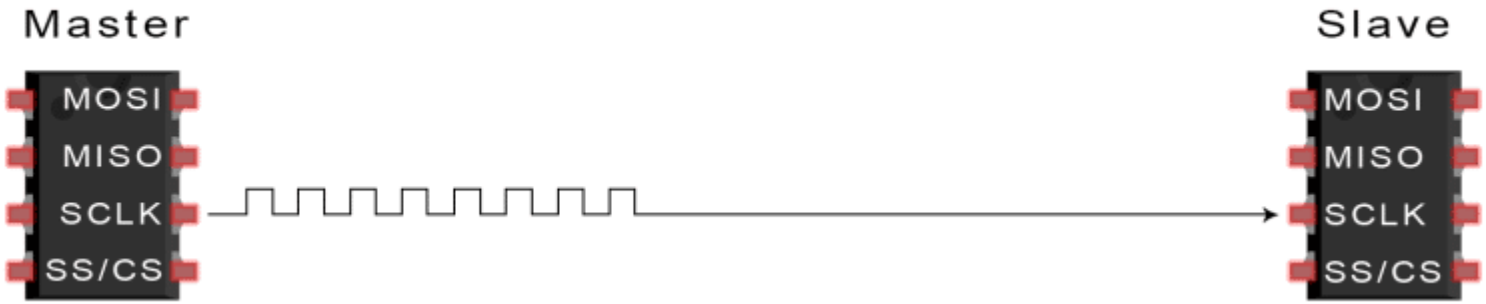
SS Line

The SS line is **normally held high**, which disconnects the slave from the SPI bus. (This type of logic is known as “active low,” and you’ll often see used it for enable and reset lines.) Just before data is sent to the slave, the line is brought low, which activates the slave. When you’re done using the slave, the line is made high again.

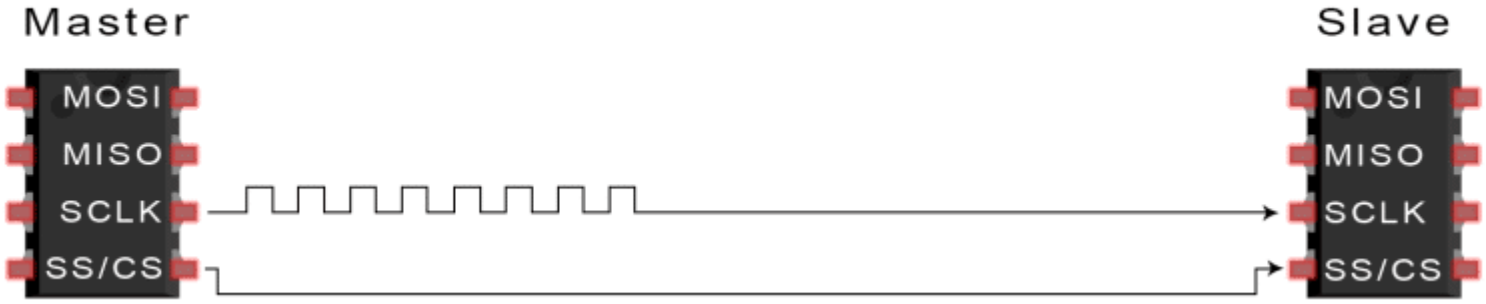


Steps of SPI Data Transmission

- The master outputs the clock signal

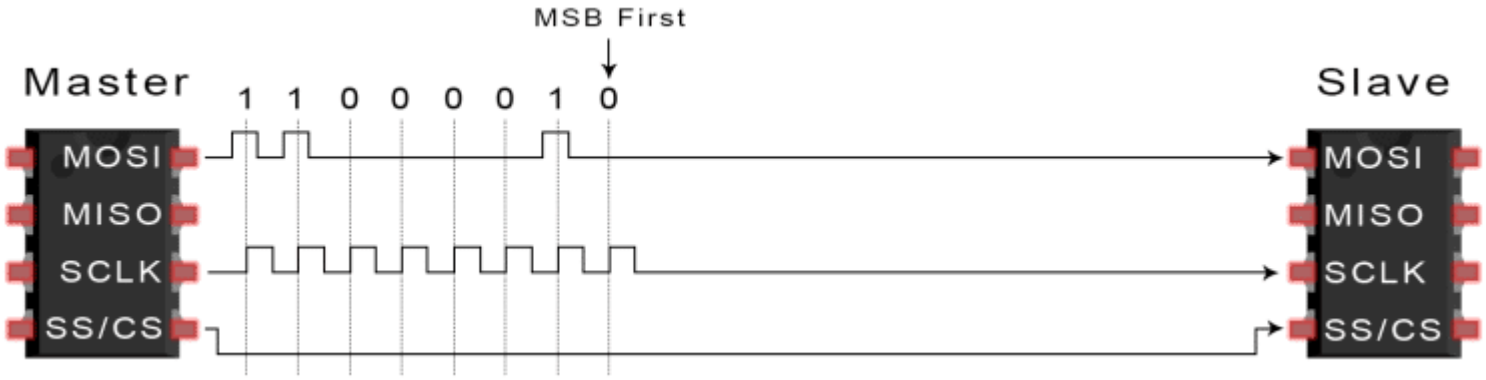


- The master switches the SS/CS pin to a low voltage state, which activates the slave

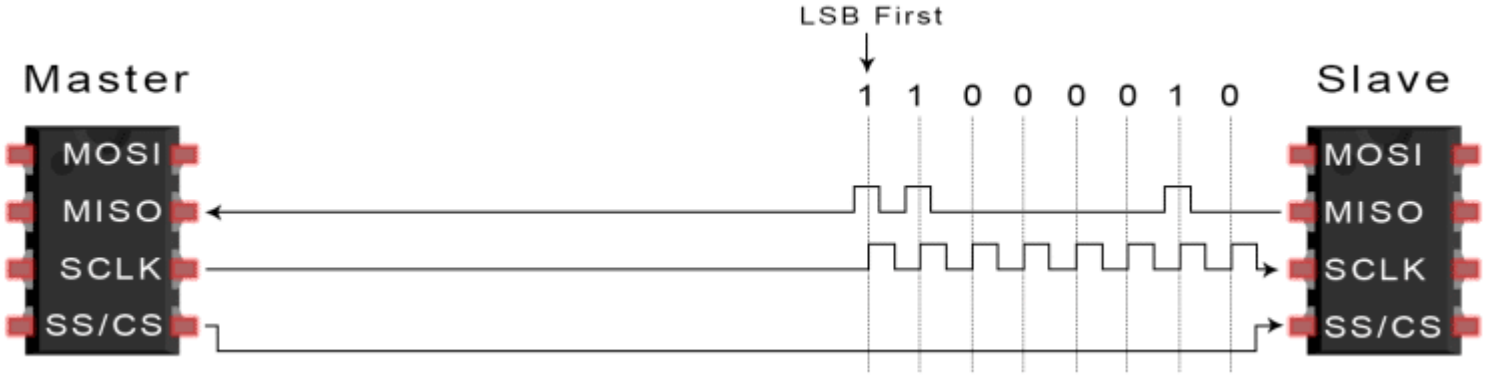


Steps of SPI Data Transmission : continued ..

The master sends the data one bit at a time to the slave along the MOSI line. The slave reads the bits as they are received:

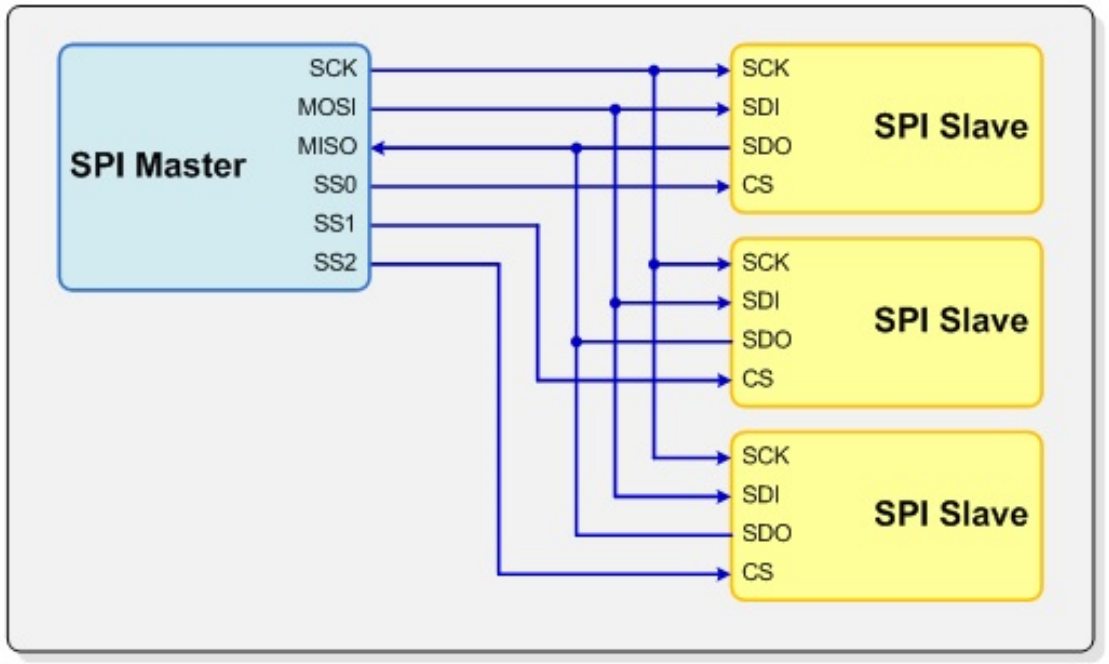


If a response is needed, the slave returns data one bit at a time to the master along the MISO line. The master reads the bits as they are received



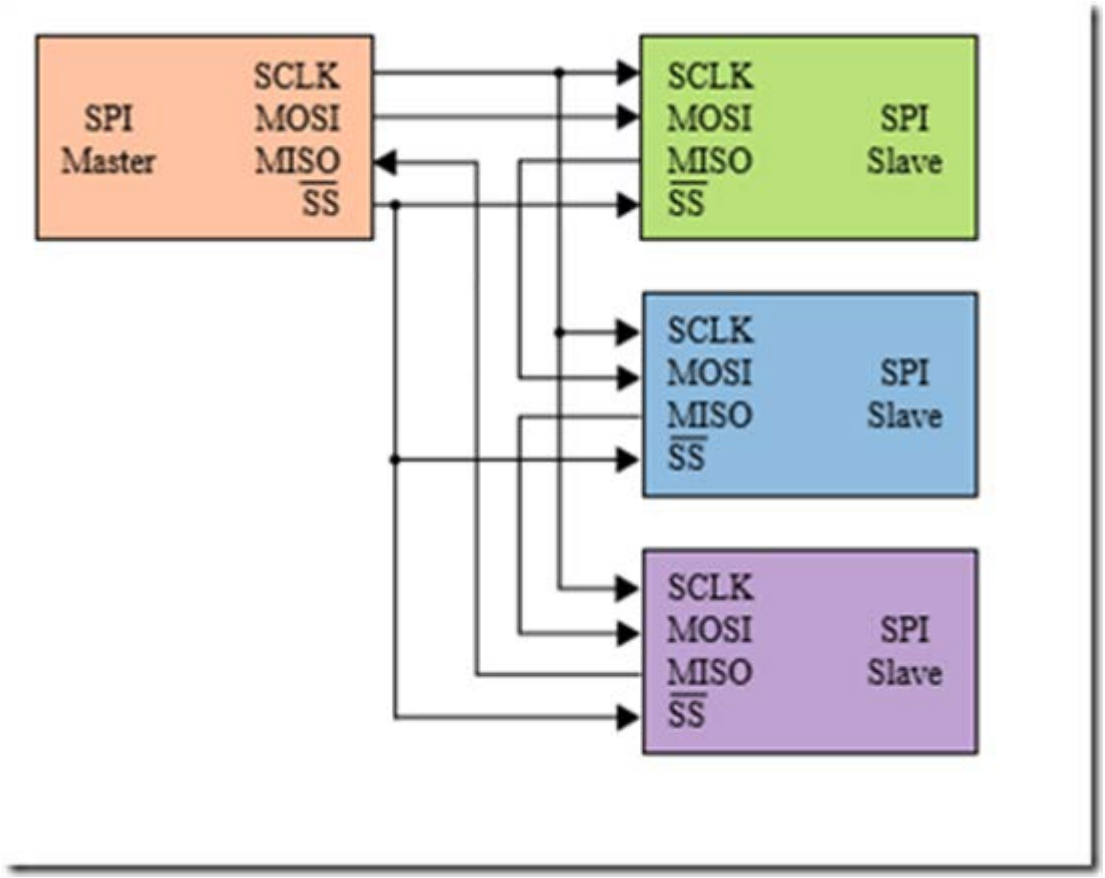
Single Master Multiple Slaves

SPI can be set up to operate with a single master and a single slave, and it can be set up with multiple slaves controlled by a single master. There are two ways to connect multiple slaves to the master. If the master has multiple slave select pins, the slaves can be wired in parallel like this ⇨



Single Master Multiple Slaves : Continued ..

If only one slave select pin is available, the slaves can be daisy-chained like this:



Benefits and Drawbacks

Benefits

- SPI is considered the fastest synchronous serial data transfer interface;
- SPI is a very simple communication protocol;
- Supports full-duplex communication;

Drawbacks

- Requires more traces on the board (X + 3, where X is the number of slave devices);
- No hardware flow control;
- No slave acknowledgment;
- May prone to noise spikes causing faulty communication.

SPI in a Nutshell

0	1	0	0	0	0	0	1
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0

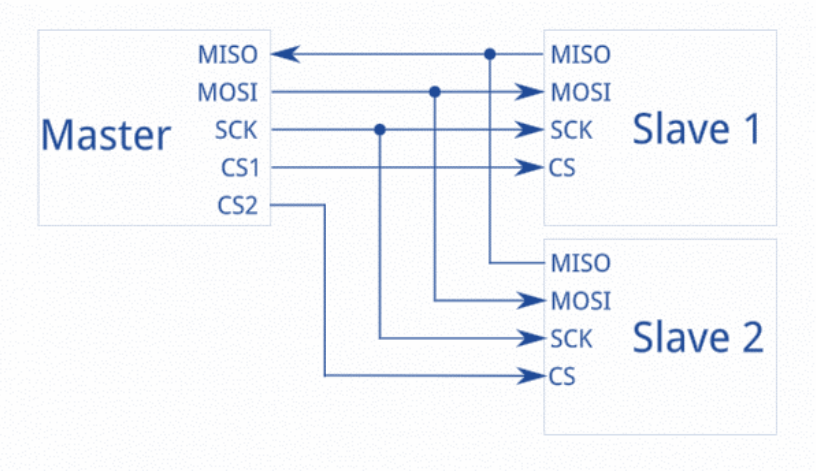
0	1	0	0	0	1	1	1
B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0

Shows data shifted out of Microchip A into Microchip B, and from Microchip B into Microchip A

Shows a virtual 4-channel oscilloscope trace of an SPI transaction between two microchips

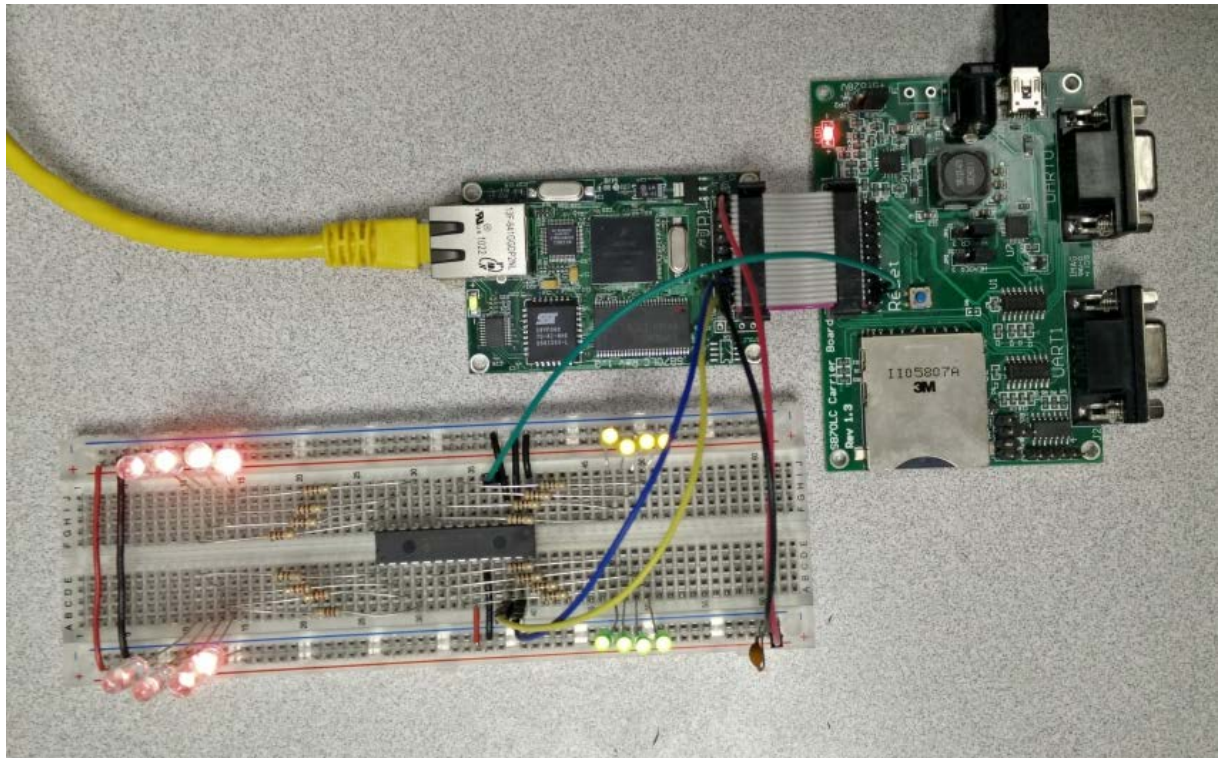
What’s wrong with SPI?

The most obvious drawback of SPI is the number of pins required. Connecting a single master to a single slave with an SPI bus requires four lines; each additional slave requires one additional chip select I/O pin on the master. The rapid proliferation of pin connections makes it undesirable in situations where lots of devices must be slaved to one master. Also, the large number of connections for each device can make routing signals more difficult in tight PCB layout situations. SPI only allows one master on the bus, but it does support an arbitrary number of slaves.



I²C Communication Protocol

I²C is a serial protocol for two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems.



I2C bus is popular because it is simple to use, there can be more than one master, only upper bus speed is defined and only two wires with pull-up resistors are needed to connect almost unlimited number of I2C devices. I2C can use even slower microcontrollers with general-purpose I/O pins since they only need to generate correct Start and Stop conditions in addition to functions for reading and writing a byte.

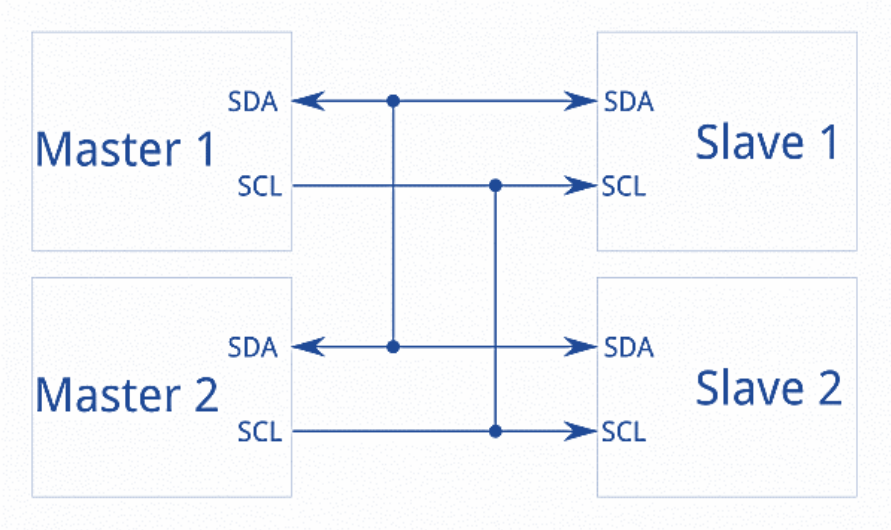
What is I^2C

- **The name stands for “Inter - Integrated Circuit Bus”**
- **A Small Area Network connecting ICs and other electronic systems**
- **Originally intended for operation on one single board / PCB**
  - Synchronous Serial Signal
  - Two wires carry information between a number of devices
  - One wire use for the data
  - One wire used for the clock
- **Today, a variety of devices are available with I^2C Interfaces**
  - Microcontroller, EEPROM, Real-Timer, interface chips, LCD driver, A/D converter

I^2C Bus Characteristics

- **Includes electrical and timing specifications, and an associated bus protocol**
- **Two wire serial data & control bus implemented with the serial data (SDA) and clock (SCL) lines**
  - For reliable operation, a third line is required:
    Common ground
- **Unique start and stop condition**
- **Slave selection protocol uses a 7-Bit slave address**
  - The bus specification allows an extension to 10 bits
- **Bi-directional data transfer**
- **Acknowledgement after each transferred byte**
- **No fixed length of transfer**

I²C Bus Connections



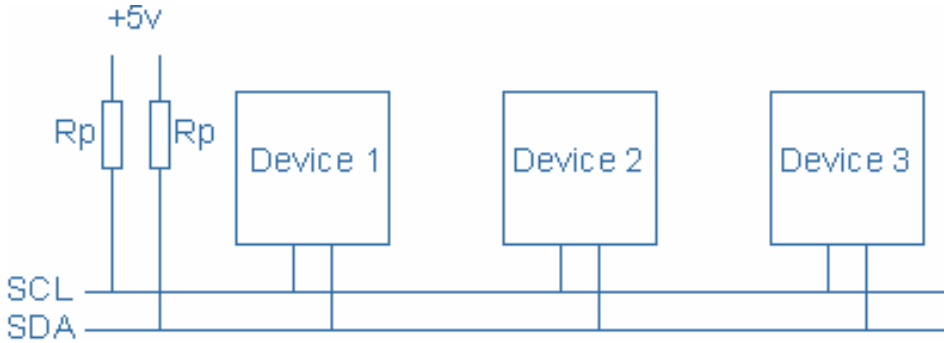
I²C requires a mere two wires, like asynchronous serial, but those two wires can support up to 1008 slave devices. Also, unlike SPI, I²C can support a multi-master system, allowing more than one master to communicate with all devices on the bus.

Data rates fall between asynchronous serial and SPI; most I²C devices can communicate at 100kHz or 400kHz. There is some overhead with I²C; for every 8 bits of data to be sent, one extra bit of meta data (the "**ACK/NACK**" bit, which we'll discuss later) must be transmitted.

The hardware required to implement I²C is more complex than SPI, but less than asynchronous serial. It can be fairly trivially implemented in software.

The Physical I²C Bus

This is just two wires, called SCL and SDA. SCL is the clock line. It is used to synchronize all data transfers over the I²C bus. SDA is the data line. The SCL & SDA lines are connected to all devices on the I²C bus. There needs to be a third wire which is just the ground or 0 volts. There may also be a 5V wire is power is being distributed to the devices. Both SCL and SDA lines are "*open drain*" drivers. What this means is that the chip can drive its output low, but it cannot drive it high. For the line to be able to go high you must provide pull-up resistors to the 5V supply. There should be a resistor from the SCL line to the 5V line and another from the SDA line to the 5V line. You only need one set of pull-up resistors for the whole I²C bus, not for each device, as illustrated below:



If the resistors are missing, the SCL and SDA lines will always be low - nearly 0 volts - and the I2C bus will not work.

I^2C Terminology

Transmitter
This is the device that transmits data to the bus

Receiver
This is the device that receives data from the bus

Master
This is the device that generates clock, starts communication, sends I2C commands and stops communication

Slave
This is the device that listens to the bus and is addressed by the master

Multi-master
I^2C can have more than one master and each can send commands

Arbitration
A process to determine which of the masters on the bus can use it when more masters need to use the bus

Synchronization
A process to synchronize clocks of two or more devices