SPI Protocol

Aurel GONTEAN

Introduction - Serial Peripheral Interface (SPI)

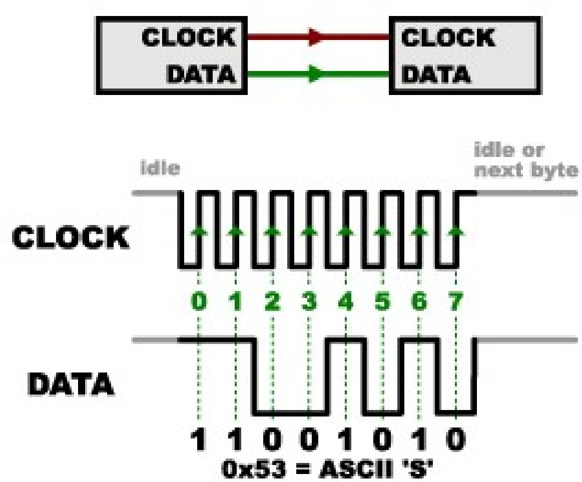
- SPI is an interface bus commonly used to send data between microcontrollers and small peripherals, such as:
  - shift registers, sensors, and SD cards.
- It uses
  - separate clock and data lines,
  - and a select line to choose the device you wish to talk to.

SPI is Synchronous

- SPI is a “synchronous” data bus,
  - it uses separate lines for data and a “clock”
  - that keeps both sides in perfect sync.
- The clock is an oscillating signal that tells the receiver exactly when to sample the bits on the data line.
- This could be the rising (low to high) or falling (high to low) edge of the clock signal;
  - the datasheet will specify which one to use.

SPI is Synchronous

- When the receiver detects that edge,
  - it will immediately look at the data line to read the next bit (see the arrows in the below diagram).
- Because the clock is sent along with the data, specifying the speed isn't important, although devices will have a top speed at which they can operate.



Motivation

- One reason that SPI is so popular is that the receiving hardware can be a simple shift register.
- This is a much simpler (and cheaper!) piece of hardware than the full-up UART (Universal Asynchronous Receiver / Transmitter) that asynchronous serial requires.

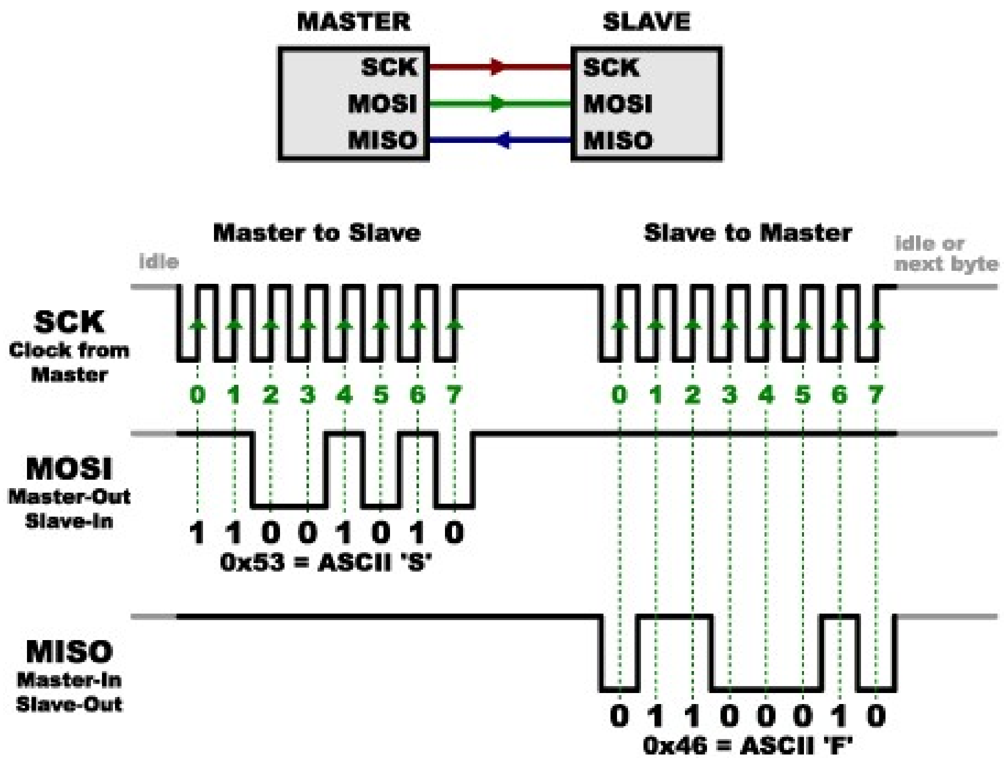
Receiving Data

- 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".
  - There is always only one master (which is almost always your microcontroller),
  - but there can be multiple slaves.

Receiving Data

- When data is sent from the master to a slave,
  - it's sent on a data line called MOSI, for "Master Out / Slave In".
- If the slave needs to send a response back to the master, the master will continue to generate a prearranged number of clock cycles,
  - the slave will put the data onto a third data line called MISO, for "Master In / Slave Out".

SPI Typical Data Transfer



Write is followed by Read

- "prearranged". Because the master always generates the clock signal, it must know in advance when a slave needs to return data and how much data will be returned.
  - This is very different than asynchronous serial, where random amounts of data can be sent in either direction at any time.
- In practice this isn't a problem, as SPI is generally used to talk to sensors that have a very specific command structure.
  - For example, if you send the command for "read data" to a device, you know that the device will always send you, for example, two bytes in return.
  - In cases where you might want to return a variable amount of data, you could always return one or two bytes specifying the length of the data and then have the master retrieve the full amount.

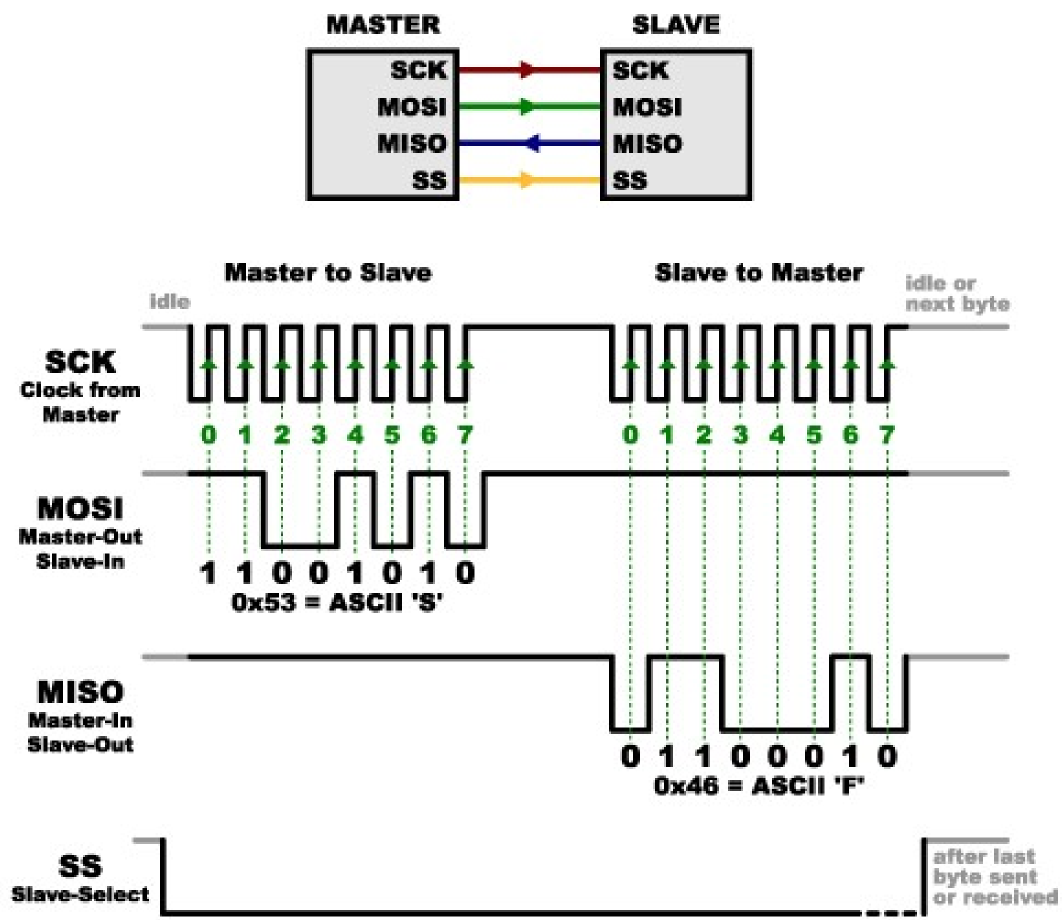
SPI is Full Duplex

- SPI is “full duplex”
  - has separate send and receive lines
- in certain situations, one can transmit and receive data at the same time
  - for example, requesting a new sensor reading while retrieving the data from the previous one.
  - Device’s datasheet will tell if this is possible.

Slave Select ($\overline{SS}$)

- The line tells the slave
  - that it should wake up and receive / send data
- nSS is also used when multiple slaves are present to select the one you'd like to talk to.
- The nSS line is normally held high, which disconnects the slave from the SPI bus[1].
- Just before data is sent to the slave, the line is brought low, which activates the slave[2].
- When you're done using the slave, the line is made high again.

$\overline{\text{Slave Select}}$ ($\overline{\text{SS}}$) based transmission

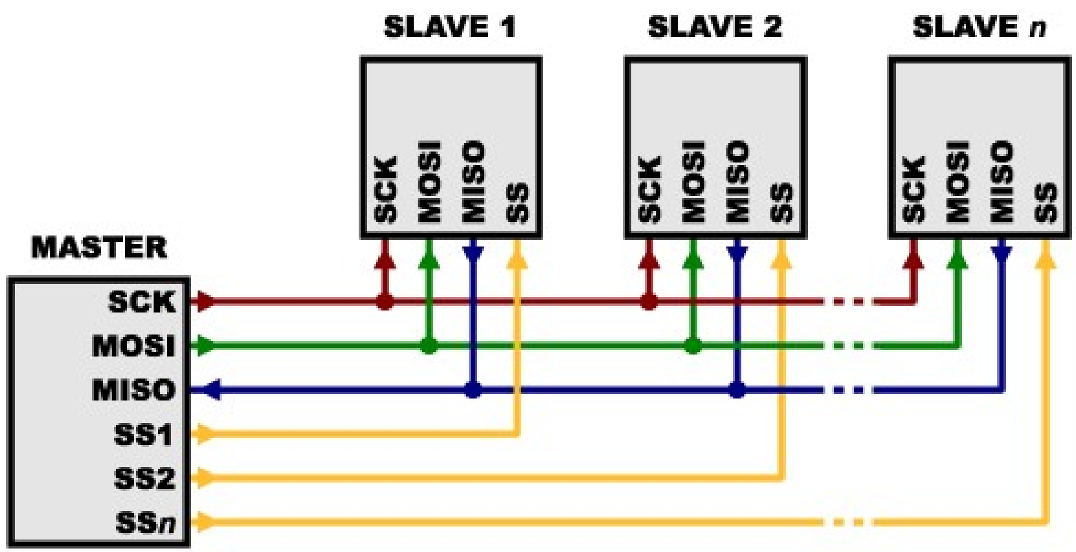


Multiple Slaves Connecting Modes

- There are two ways of connecting multiple slaves to an SPI bus:
  1. Each slave will need a separate nSS line.
  2. Devices are daisy-chained together,
     - with the MISO (output) of one going to the MOSI (input) of the next

Using separate nSS lines

- To talk to a particular slave, you'll make that slave's nSS line low and keep the rest of them high
  - (you don't want two slaves activated at the same time, or they may both try to talk on the same MISO line resulting in garbled data).
- Lots of slaves will require lots of SS lines;
  - if you're running low on outputs, there are binary decoder chips that can multiply your SS outputs.

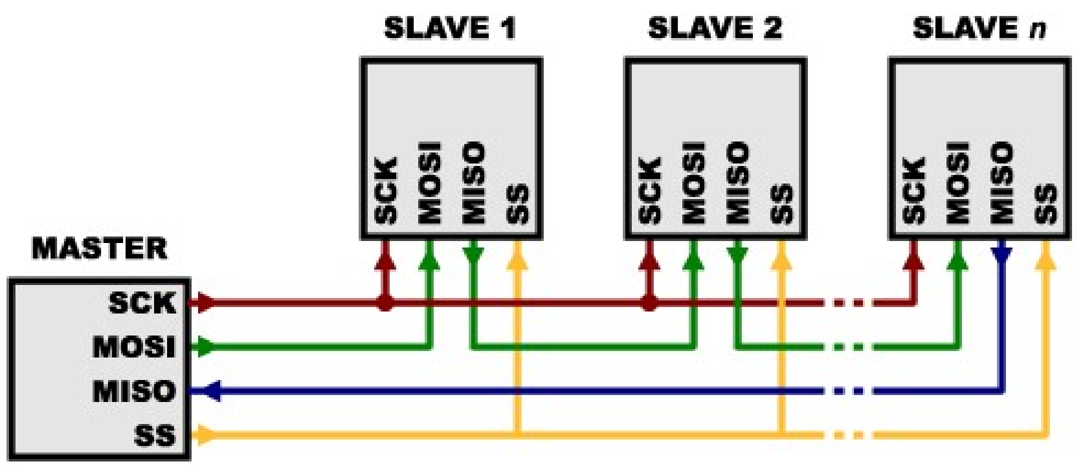


Daisy Chained Slaves

- Some parts prefer to be daisy-chained together
  - with the MISO (output) of one going to the MOSI (input) of the next.
- In this case, a single nSS line goes to all the slaves.
- Once all the data is sent, the nSS line is raised,
  - which causes all the chips to be activated simultaneously.
- This is often used for daisy-chained shift registers and addressable LED drivers.

Daisy Chained Slaves

- Here data overflows from one slave to the next,
  - so to send data to any *one* slave, you'll need to transmit enough data to reach *all* of them.
- Also, the *first* piece of data you transmit will end up in the *last* slave.



Daisy Chained Slaves

- This type of layout is typically used in output-only situations,
  - e.g. driving LEDs where you don't need to receive any data back.
- In these cases you can leave the master's MISO line disconnected.
- However, if data does need to be returned to the master,
  - you can do this by closing the daisy-chain loop (blue wire).
  - In this case the return data from slave 1 will need to pass through *all* the slaves before getting back to the master,
    - be sure to send enough receive commands to get the data you need.