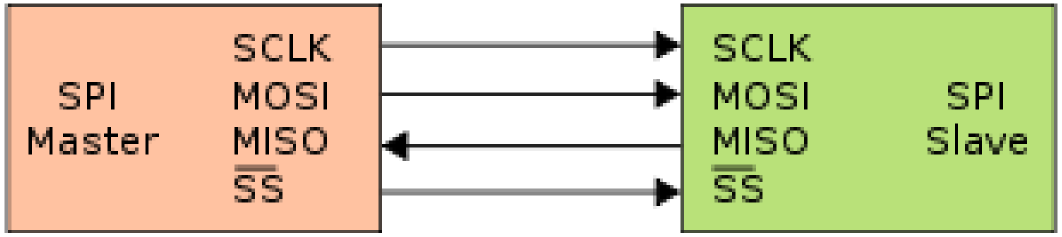
Altera Cyclone V HPS-FPGA

SPI communications

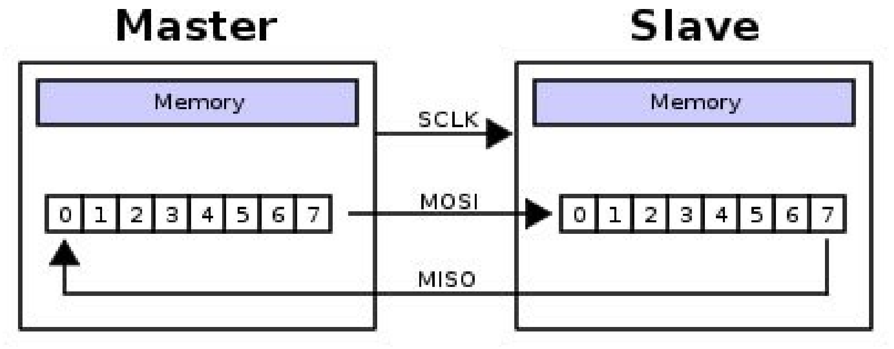
S(erial) P(eripheral) I(nterface)

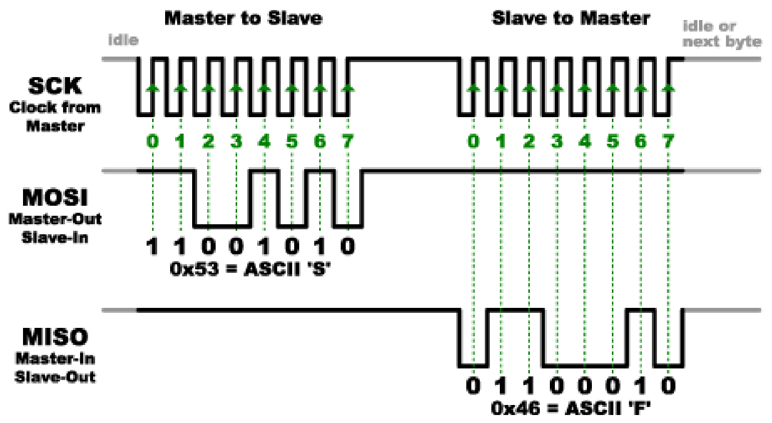
- Generally used to connect external devices
  - Microcontroller world has introduced new standards for peripherals interfaces that can be externally wired or internally wired via PCB tracks: SPI and I2C; SPI developed by Motorola and “de facto” standard.
  - SPI can be considered as a wired bus that can operate with a single master device and with one or more slave devices.
- SPI: OSI Layer 0; SPI uses four wire for exchanging data from peripherals
  - MOSI (Master Out Slave In) [SDO]
  - MISO (Master In Slave Out) [SDI]
  - SCK (Slave Clock) [SCK]
  - SS_n (Slave Select).



Master vs Slave(s)

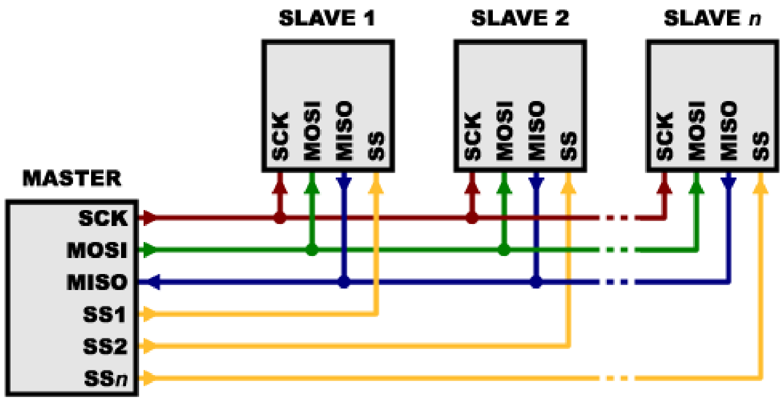
- Master starts the communication (drives the clock, using a frequency supported by the slave device) and selects the Slave (issuing logic level ‘0’ on the select line and possibly waiting for slow peripheral).
- During each SPI clock cycle, a full duplex data transmission occurs.
  - ✓ The master sends a bit on the MOSI line and the slave reads it, while the slave sends a bit on the MISO line and the master reads it.
  - ✓ This sequence is maintained even when only one-directional data transfer is intended.

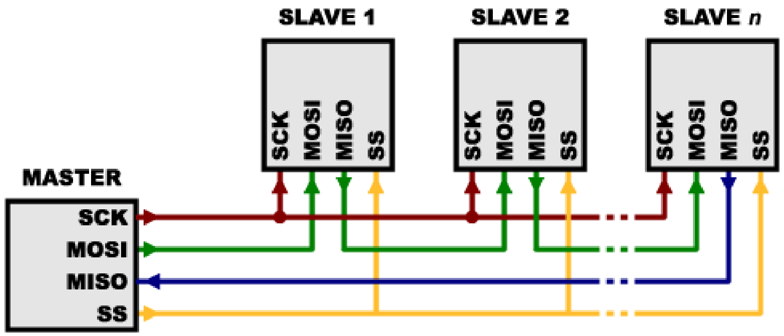




Multi slaves

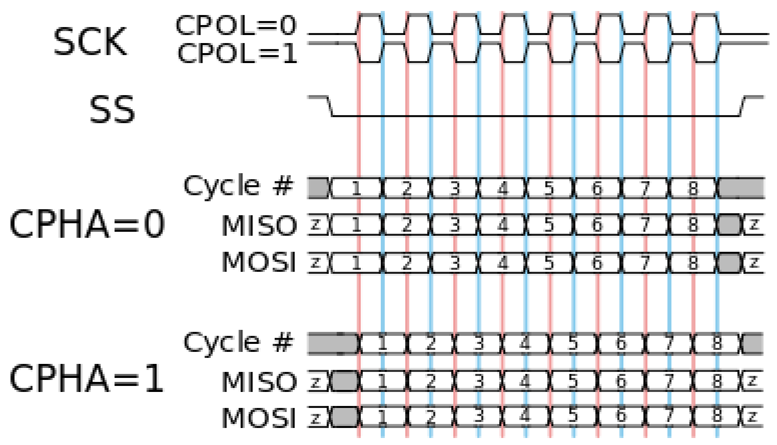
- In general, each slave will need a separate SS line (only one at ‘0’ at a time). Lots of slaves will require lots of SS lines!
- Daisy Chain Multi Slave: the MISO of one going into the MOSI of the next. A single SS line goes to all the slaves. All the devices activated simultaneously.
  - Data stram overflows from one slave to the next: you’ll need to transmit enough data to reach all of them.
  - The first piece of data you transmit will end up in the last slave. Each slave must copy input data to output to propagate them.





Polarity and modes

- The master must also configure the clock polarity and phase (w.r.t. to data).
  - At CPOL=0 the base value of the clock is ‘0’ (active state is 1 and idle state is 0).
    - For CPHA=0, data are captured on the clock's rising edge (low→high transition) and data is output on a falling edge (high→low clock transition).
    - For CPHA=1, data are captured on the clock's falling edge and data is output on a rising edge.
  - At CPOL=1 the base value of the clock is ‘1’ (active state is 0 and idle state is 1).
    - For CPHA=0, data are captured on clock's falling edge and data is output on a rising edge.
    - For CPHA=1, data are captured on clock's rising edge and data is output on a falling edge.



SPI Mode	CPOL	CPHA	Clock Polarity in Idle State
0	0	0	Logic low
1	**0**	**1**	**Logic low**
2	1	1	Logic high
3	1	0	Logic high

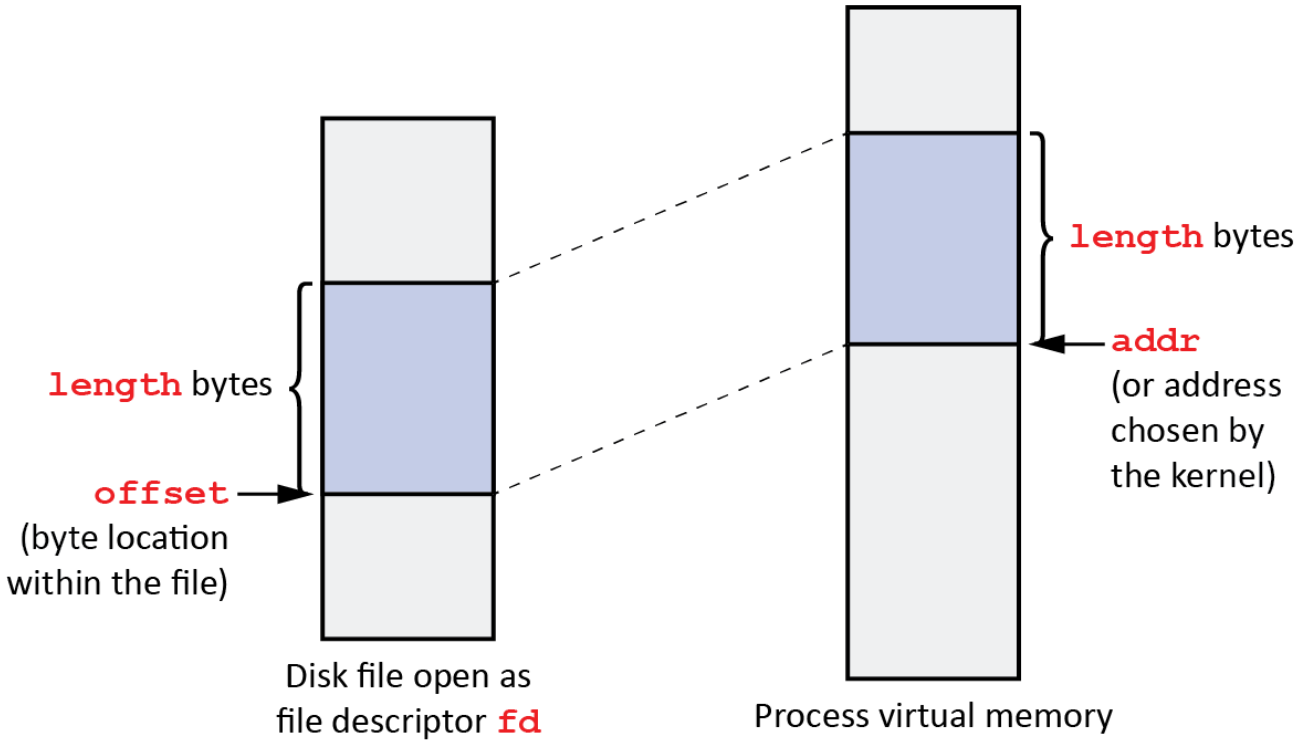
Pros and Cons

- Advantages of SPI:
  - It's faster than asynchronous serial
  - The receive hardware can be a simple shift register
  - It supports multiple slaves
- Disadvantages of SPI:
  - It requires more signal lines (wires) than other communications methods
  - The communications must be well-defined in advance (you can't send random amounts of data whenever you want)
  - The master must control all communications (slaves can't talk directly to each other)
  - It usually requires separate SS lines to each slave, which can be problematic if numerous slaves are needed.

The mmap system call

- mmap() is a system call that can be used by a user process to ask the operating system kernel to map either files or devices into the memory (i.e., address space) of that process.

```
void *mmap(void *addr, size_t length,
    int prot, int flags, int fd, off_t offset)
```



length bytes
offset
(byte location within the file)
Disk file open as file descriptor fd
length bytes
addr
(or address chosen by the kernel)
Process virtual memory

The mmap system call

- mmap() is a system call that can be used by a user process to ask the operating system kernel to map either files or devices into the memory (i.e., address space) of that process.

- Here is the function prototype for mmap():

void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);

- ✓ addr - A hint to the operating system kernel as to the address at which the virtual mapping should start in the virtual memory (i.e., the virtual address space) of the process. The value can be specified as NULL to indicate that the kernel can place the virtual mapping anywhere it sees fit. If not NULL, then addr should be a multiple of the page size.
- ✓ length - The length (number of bytes) for the mapping.
- ✓ prot - The protection for the mapped memory. The value of prot is the bitwise or of various of the following single-bit values:
- ✓ PROT_READ - Enable the contents of the mapped memory to be readable by the process.
- ✓ PROT_WRITE - Enable the contents of the mapped memory to be writable by the process.
- ✓ PROT_EXEC - Enable the contents of the mapped memory to be executable by the process as CPU machine instructions.

The mmap system call

- mmap() is a system call that can be used by a user process to ask the operating system kernel to map either files or devices into the memory (i.e., address space) of that process.

- Here is the function prototype for mmap():

void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);

- ✓ flags - Various options controlling the mapping. The flag value used is described below: MAP_SHARED - Modifications to the mapped memory region are visible to other processes mapping the same file and are eventually reflected in the file.
- ✓ fd - The open file descriptor for the file from which to populate the memory region.
- ✓ offset - If this is not an anonymous mapping, the memory mapped region will be populated with data starting at position offset bytes from the beginning of the file open as file descriptor fd. Should be a multiple of the page size.