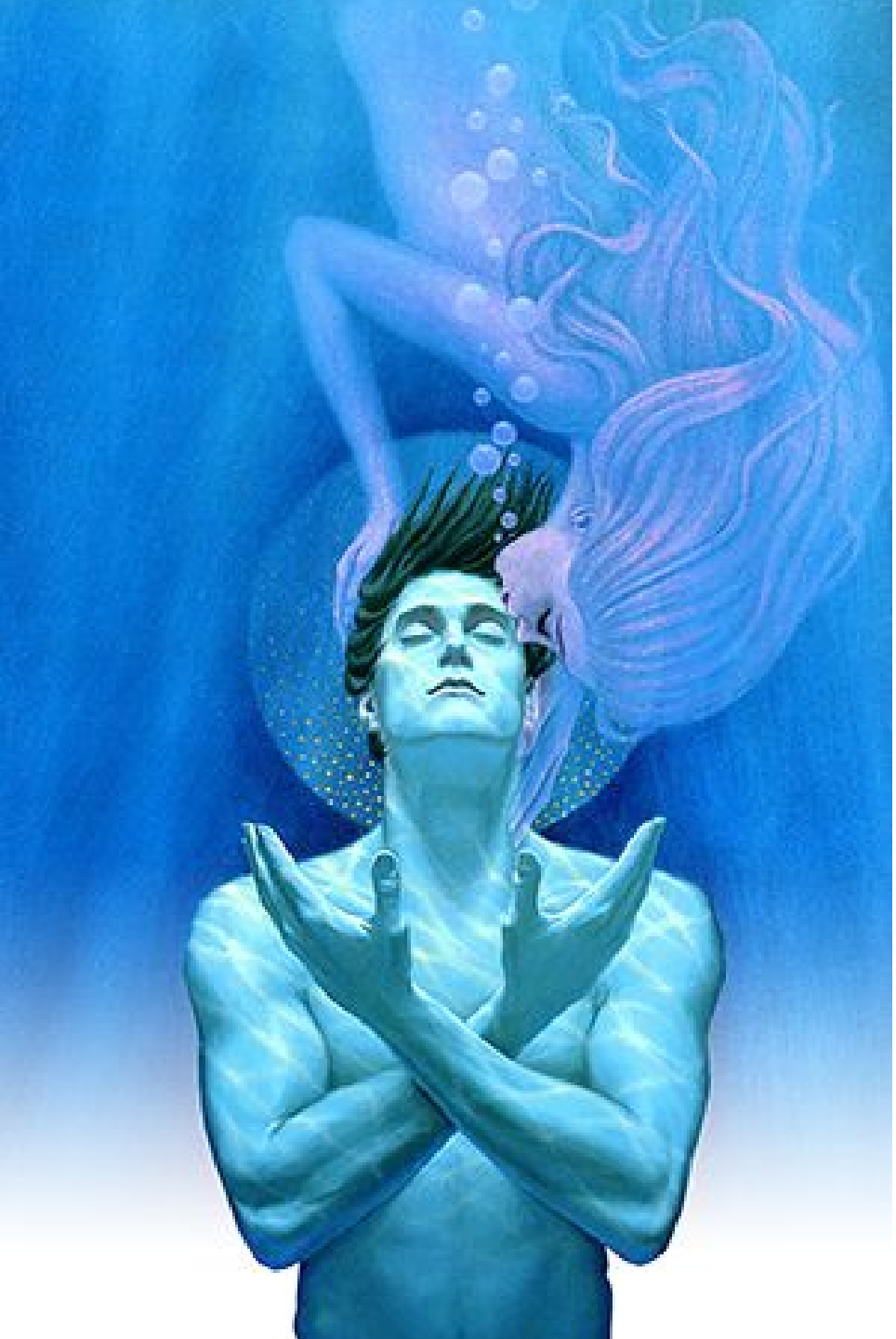
Groking the Linux SPI Subsystem

FOSDEM 2017

Matt Porter

Konsulko Group



Obligatory geek reference deobfuscation

grok ***(/gräk/)***

verb

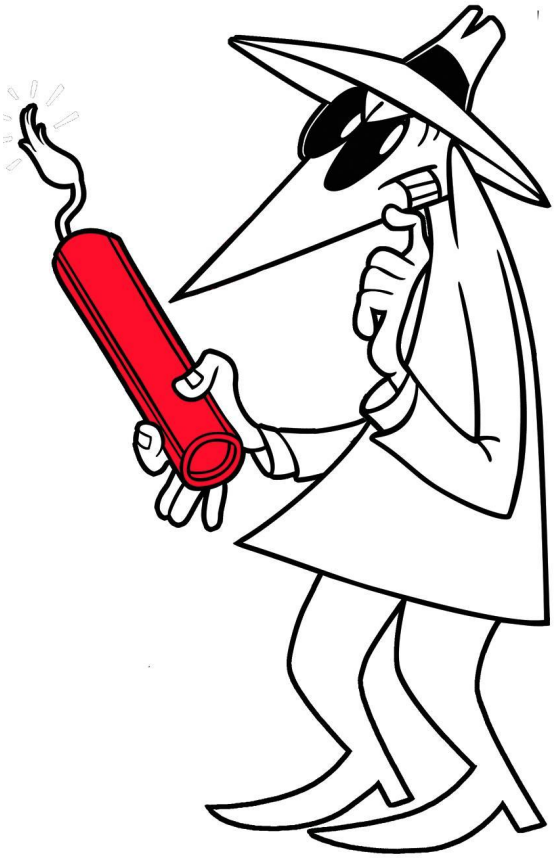
to understand intuitively or by empathy, to establish rapport with.

Overview

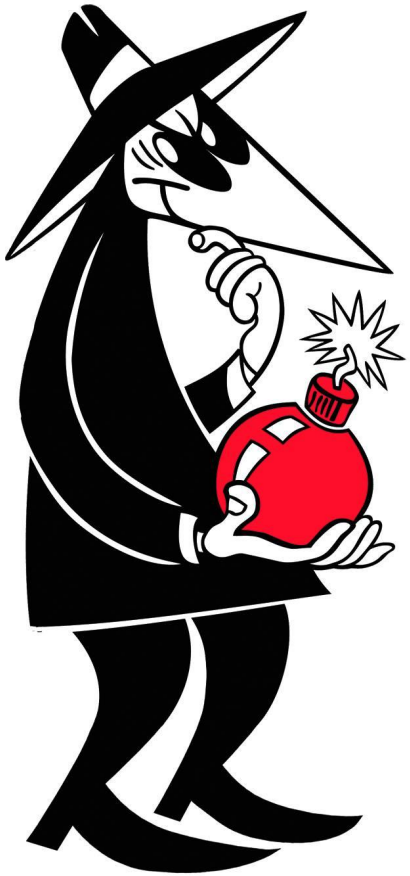
- What is SPI?
- SPI Fundamentals
- Linux SPI Concepts
- Linux SPI Use cases
  - Add a device
  - Protocol drivers
  - Controller drivers
  - Userspace drivers
- Linux SPI Performance
- Linux SPI Future

What is SPI?

What is SPI?



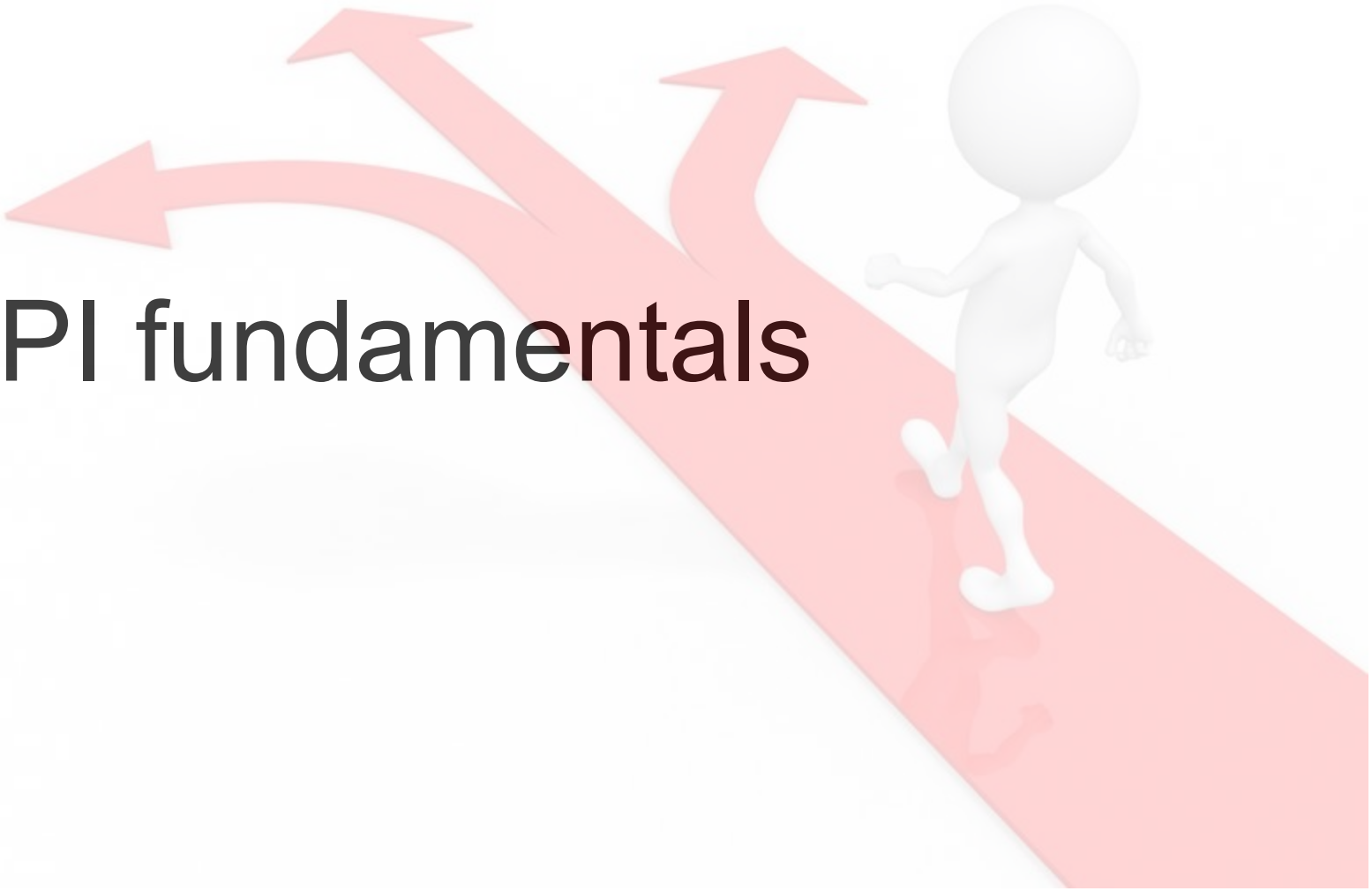
- Serial Peripheral Interface
- Motorola
- de facto standard
- master-slave bus
- 4 wire bus
  - except when it's not
- no maximum clock speed
- http://wikipedia.org/wiki/Serial_Peripheral_Interface
- "A glorified shift register"



Common uses of SPI

- Flash memory
- ADCs
- Sensors
  - thermocouples, other high data rate devices
- LCD controllers
- Chromium Embedded Controller

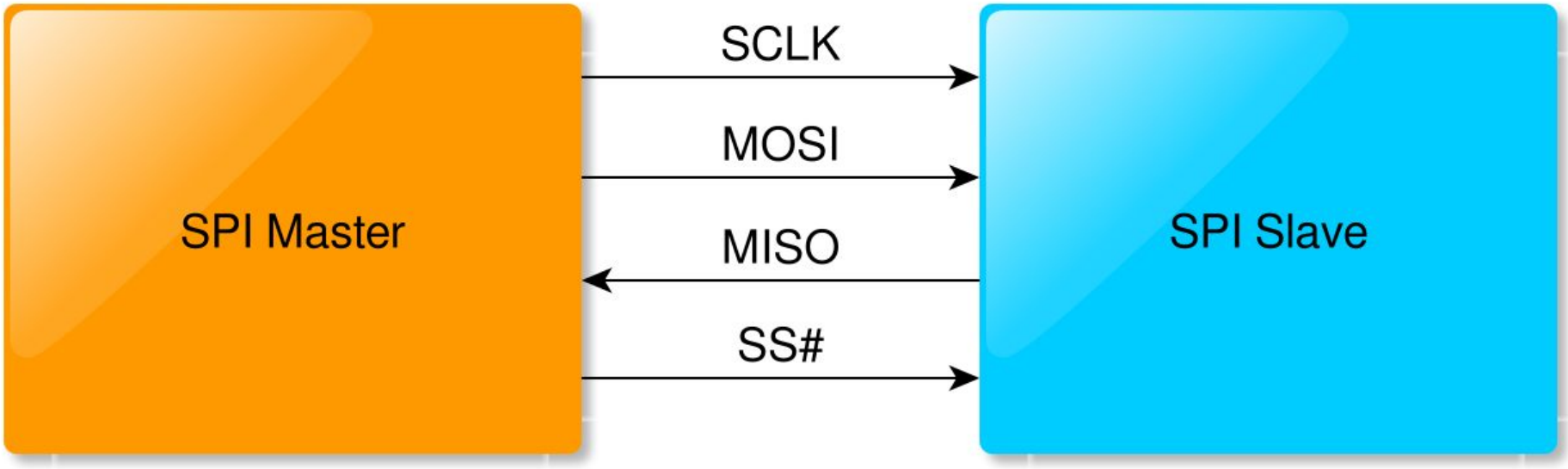
SPI fundamentals



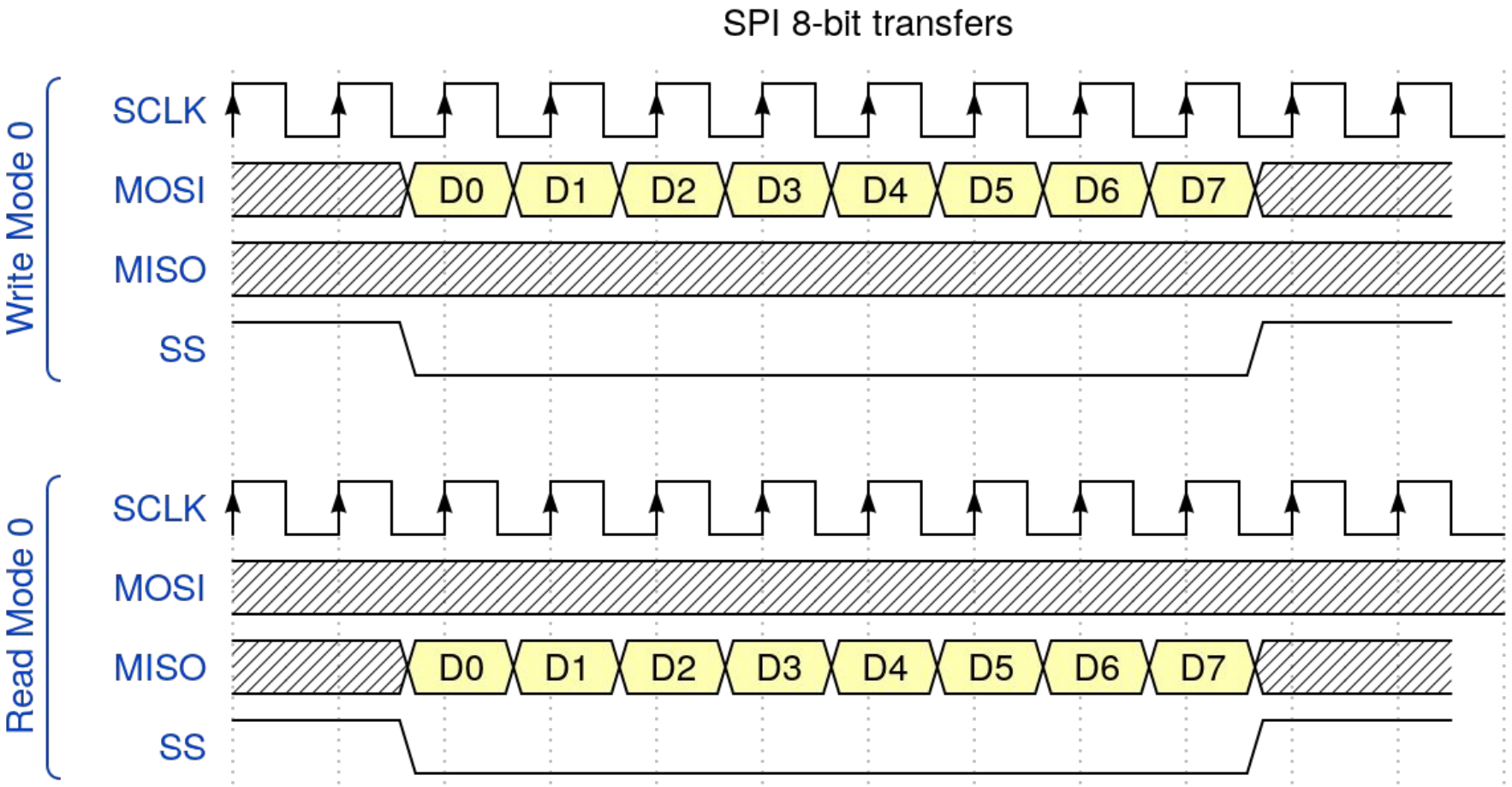
SPI Signals

- MOSI - Master Output Slave Input
  - SIMO, SDI, DI, SDA
- MISO - Master Input Slave Output
  - SOMI, SDO, DO, SDA
- SCLK - Serial Clock (Master output)
  - SCK, CLK, SCL
- $\overline{SS}$ - Slave Select (Master output)
- CSn, EN, ENB

SPI Master and Slave



Basic SPI Timing Diagram

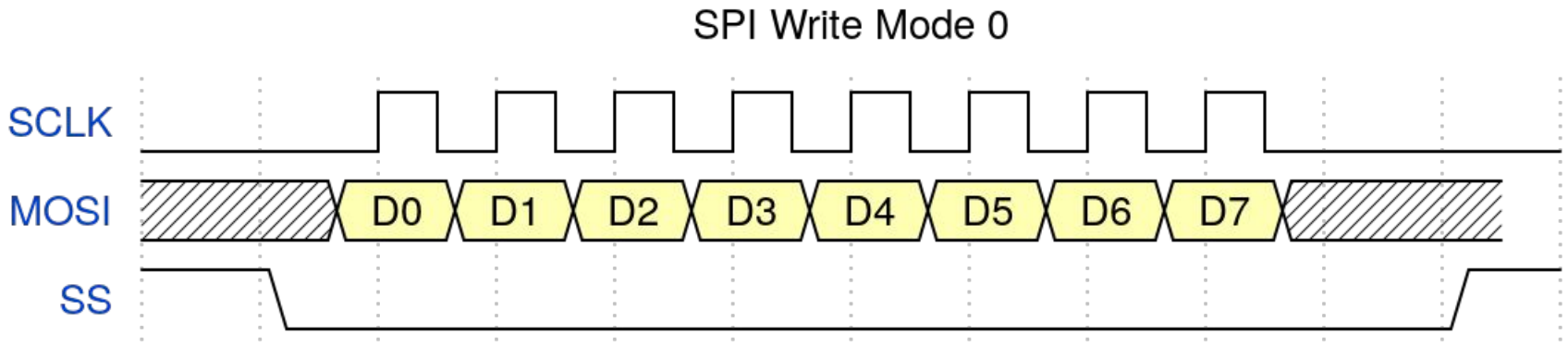


SPI Modes

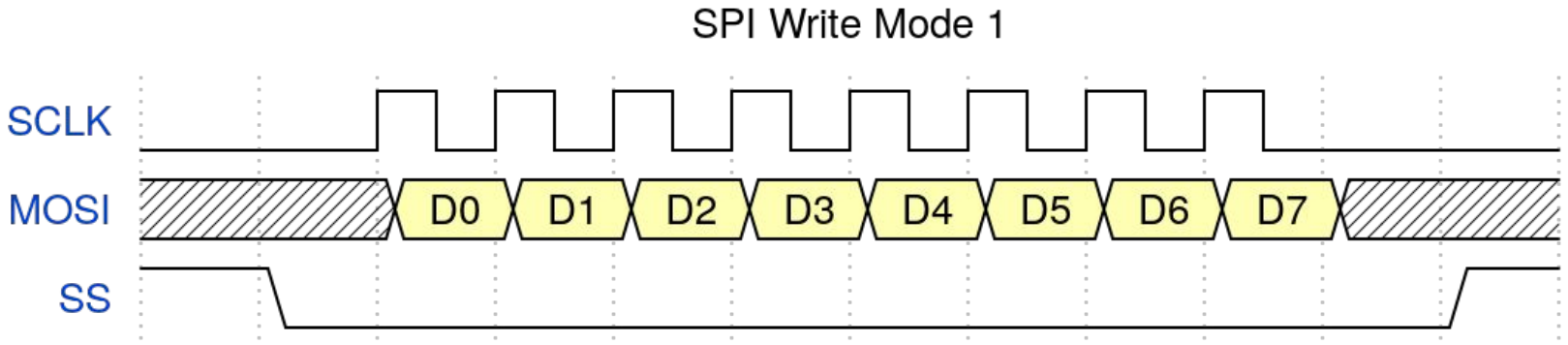
- Modes are composed of two clock characteristics
- CPOL - clock polarity
  - 0 = clock idle state low
  - 1 = clock idle state high
- CPHA - clock phase
  - 0 = data latched falling, output rising
  - 1 = data latched rising, output falling

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

SPI Mode Timing - CPOL 0

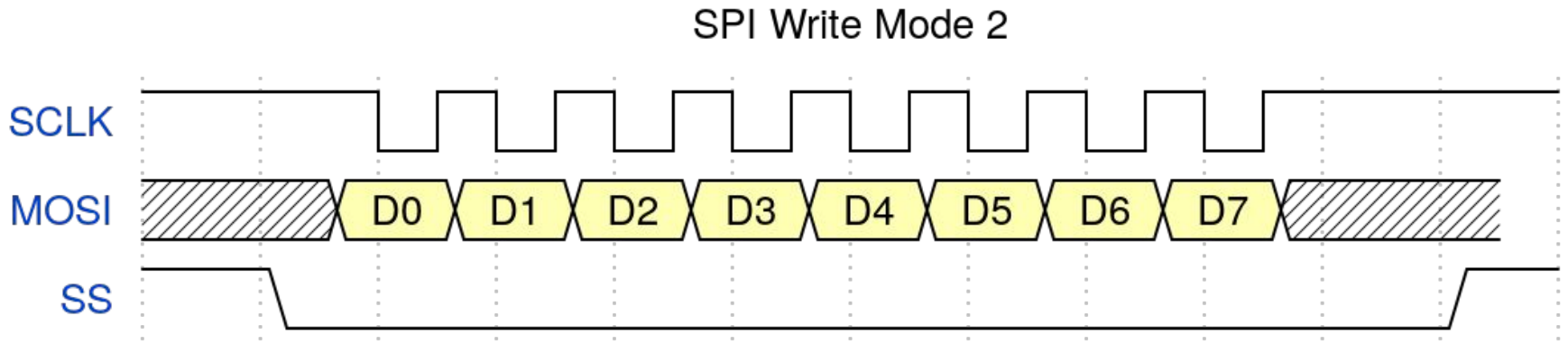


Clock idle low, data latched on rising edge

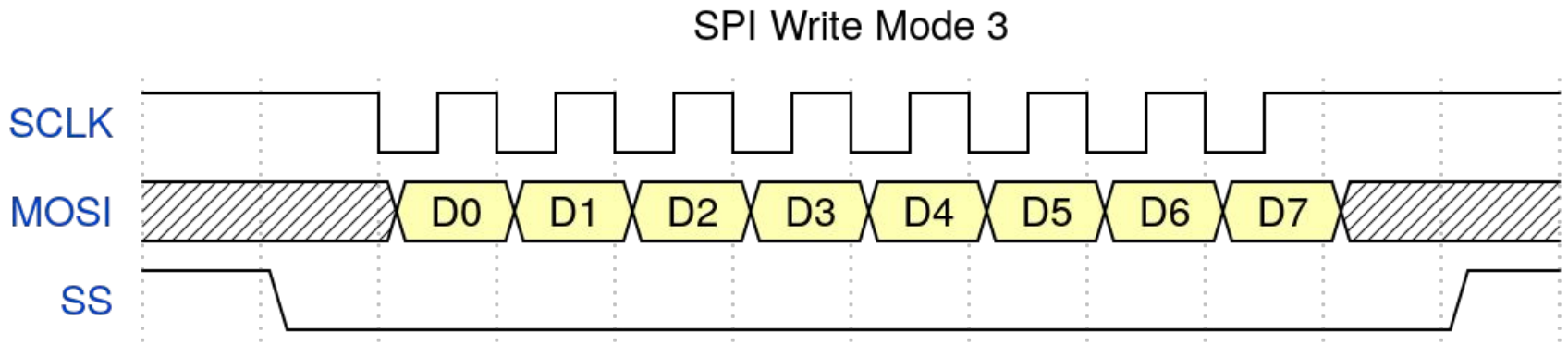


Clock idle low, data latched on falling edge

SPI Mode Timing - CPOL 1



Clock idle high, data latched on falling edge

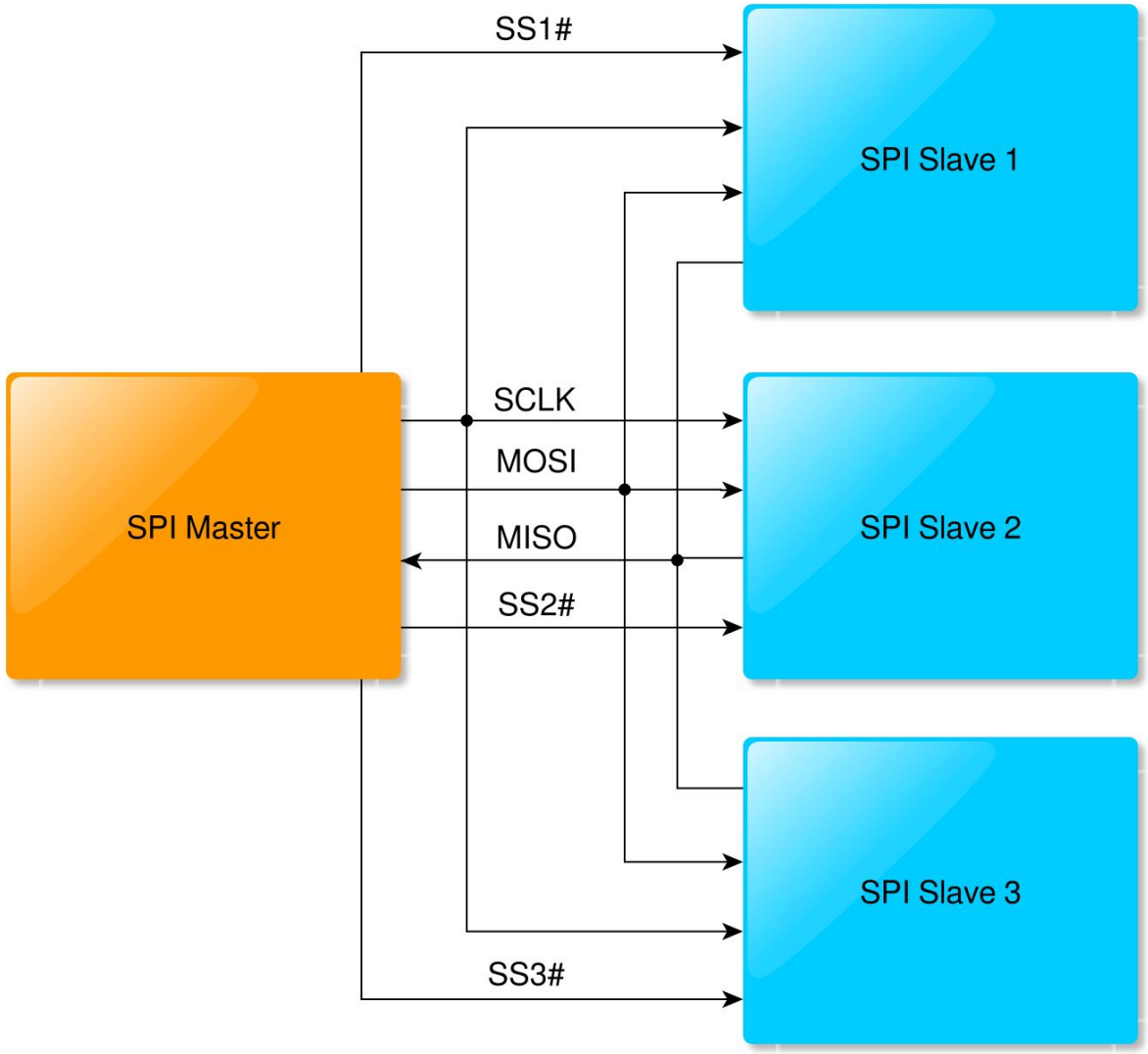


Clock idle high, data latched on rising edge

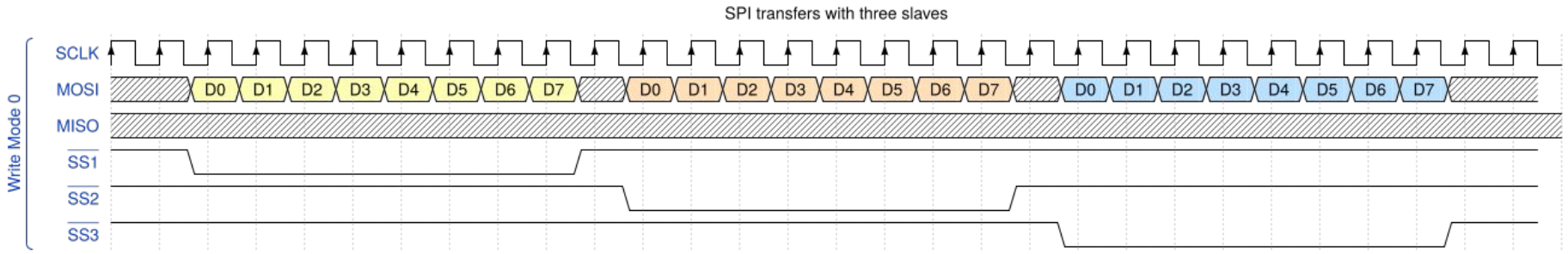
SPI can be more complicated

- Multiple SPI Slaves
  - One chip select for each slave
- Daisy Chaining
  - Inputs to Outputs
  - Chip Selects
- Dual or Quad SPI (or more lanes)
  - Implemented in high speed SPI Flash devices
  - Instead of one MISO, have N MISOs
  - N times bandwidth of traditional SPI
- 3 Wire (Microwire) SPI
  - Combined MISO/MOSI signal operates in half duplex

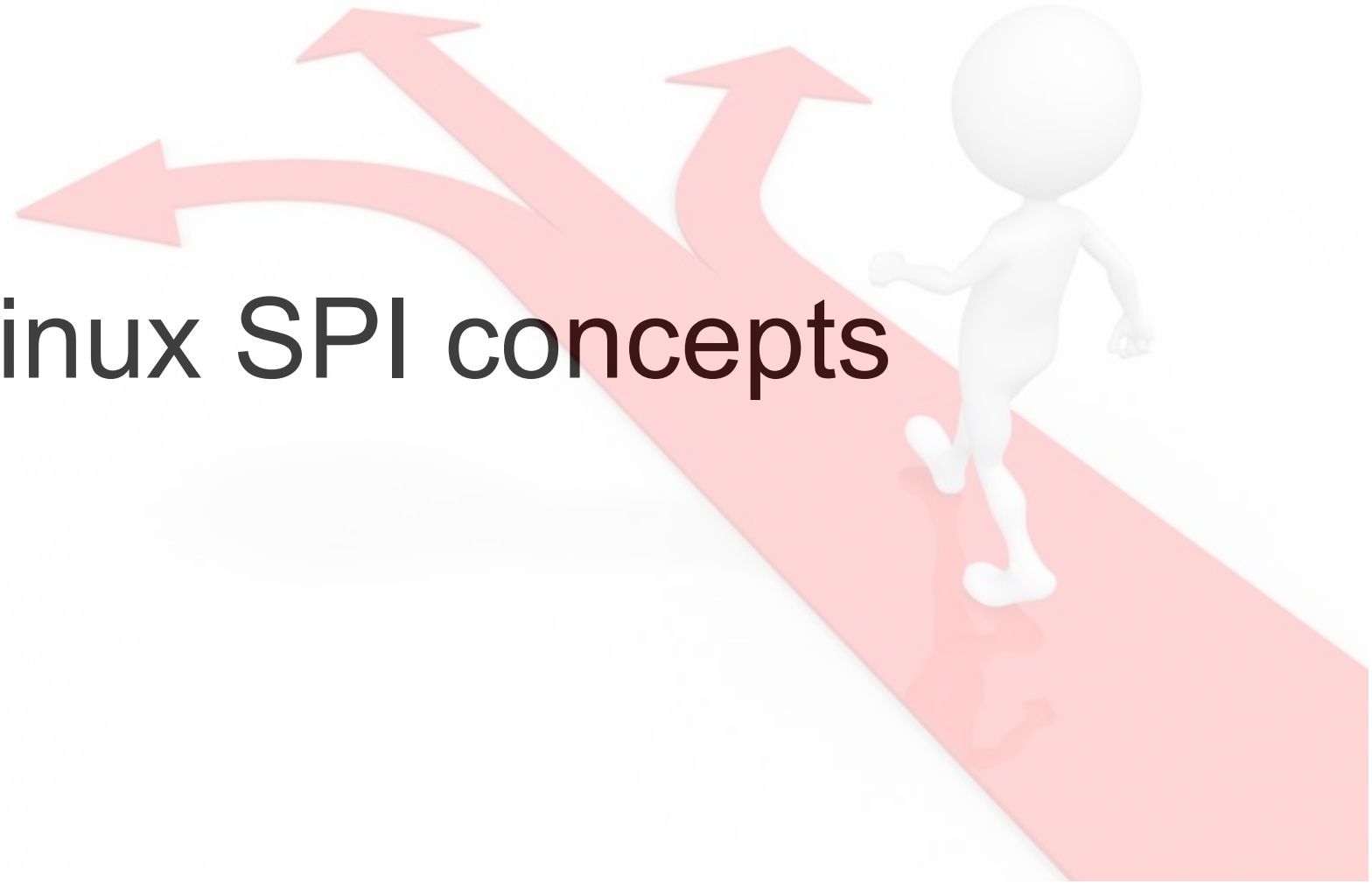
Multiple SPI Slaves



SPI Mode Timing - Multiple Slaves



Linux SPI concepts



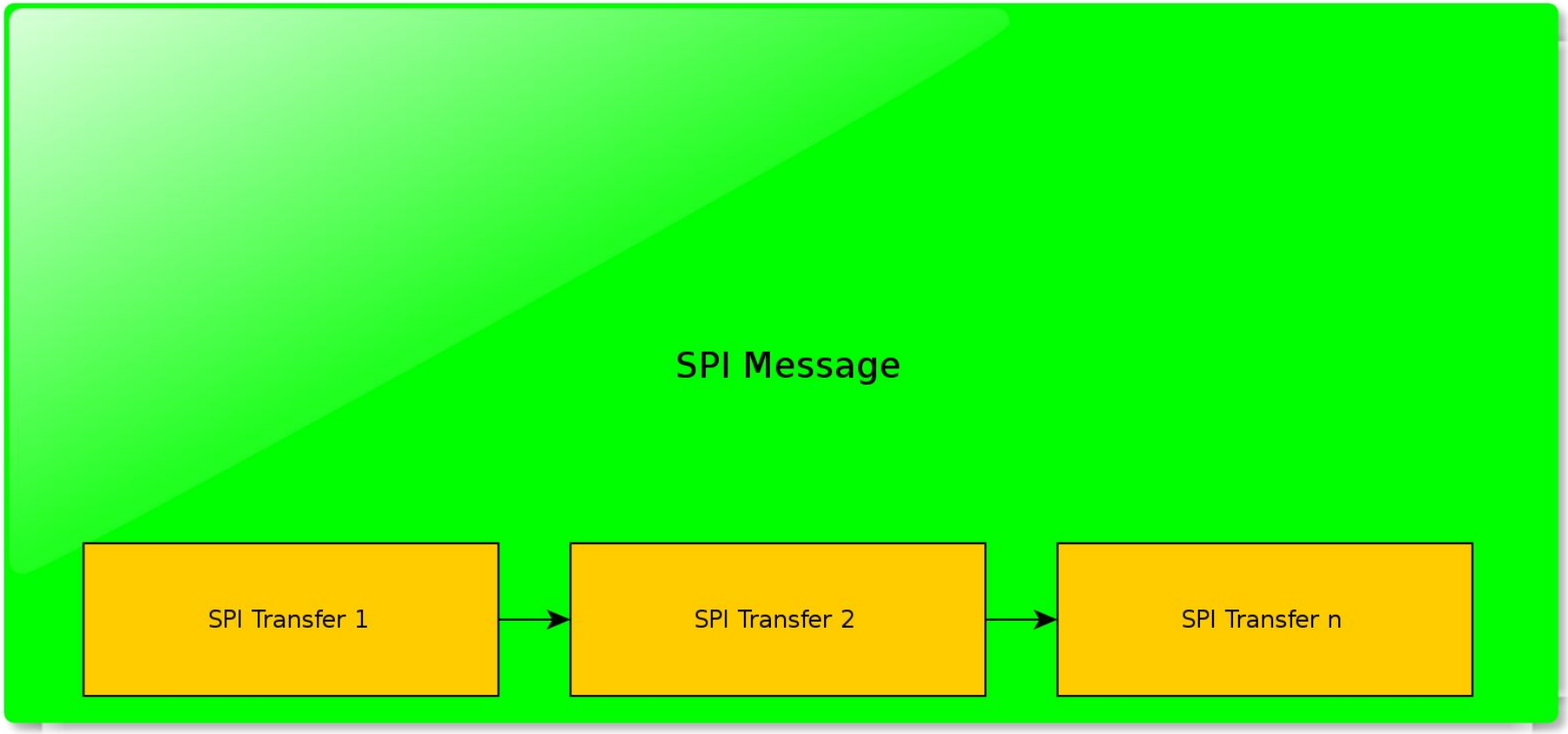
Linux SPI drivers

- Controller and Protocol drivers only (so far)
  - Controller drivers support the SPI master controller
    - Drive hardware to control clock and chip selects, shift data bits on/off wire and configure basic SPI characteristics like clock frequency and mode.
    - e.g. spi-bcm2835aux.c
  - Protocol drivers support the SPI slave specific functionality
    - Based on messages and transfers
    - Relies on controller driver to program SPI master hardware.
    - e.g. MCP3008 ADC

Linux SPI communication

- Communication is broken up into transfers and messages
- Transfers
  - Defines a single operation between master and slave.
  - tx/rx buffer pointers
  - optional chip select behavior after operation
  - optional delay after operation
- Messages
  - Atomic sequence of transfers
  - Fundamental argument to all SPI subsystem read/write APIs.

SPI Messages and Transfers



Linux SPI use cases

Exploring via use cases

- I want to hook up a SPI device on my board that already has a protocol driver in the kernel.
- I want to write a kernel protocol driver to control my SPI slave.
- I want to write a kernel controller driver to drive my SPI master.
- I want to write a userspace protocol driver to control my SPI slave.

Adding a SPI device to a system

- Know the characteristics of your slave device!
  - Learn to read datasheets
- Three methods
  - Device Tree
    - Ubiquitous
  - Board File
    - Deprecated
  - ACPI
    - Mostly x86

Reading datasheets for SPI details - ST7735

The ST7735 is a single-chip controller/driver for 262K-color, graphic type TFT-LCD. It consists of 396 source line and 162 gate line driving circuits. This chip is capable of connecting directly to an external microprocessor, and accepts Serial Peripheral Interface (SPI), 8-bit/9-bit/16-bit/18-bit parallel interface. Display data can be stored in the on-chip display data RAM of 132 x 162 x 18 bits. It can perform display data RAM read/write operation with no external operation clock to

8.2 Serial interface characteristics (3-line serial)

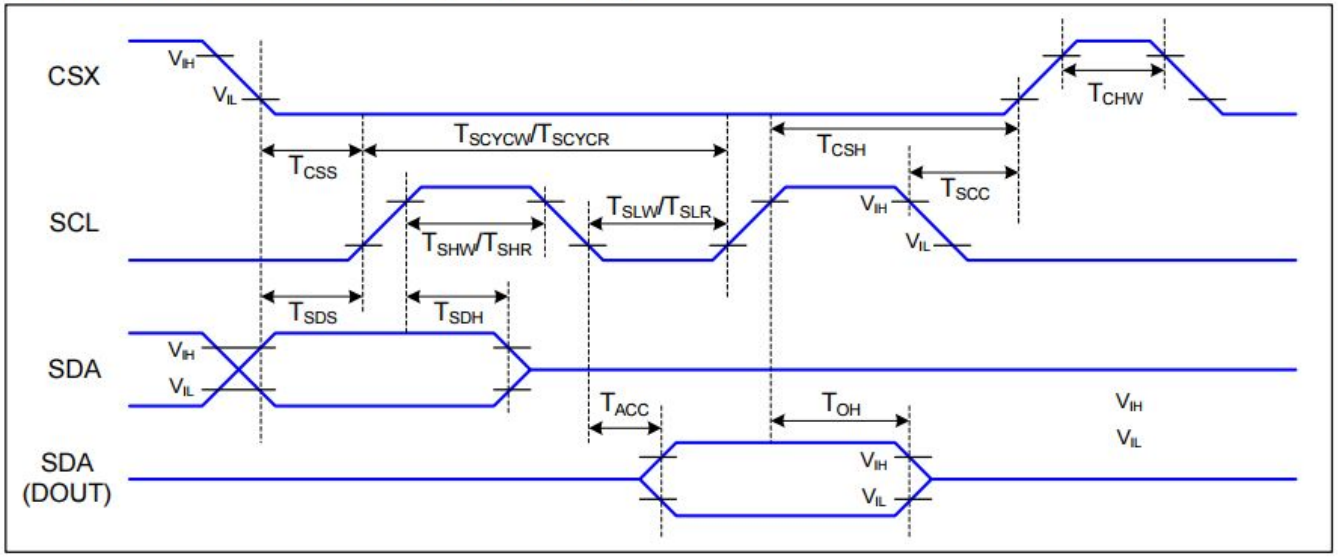
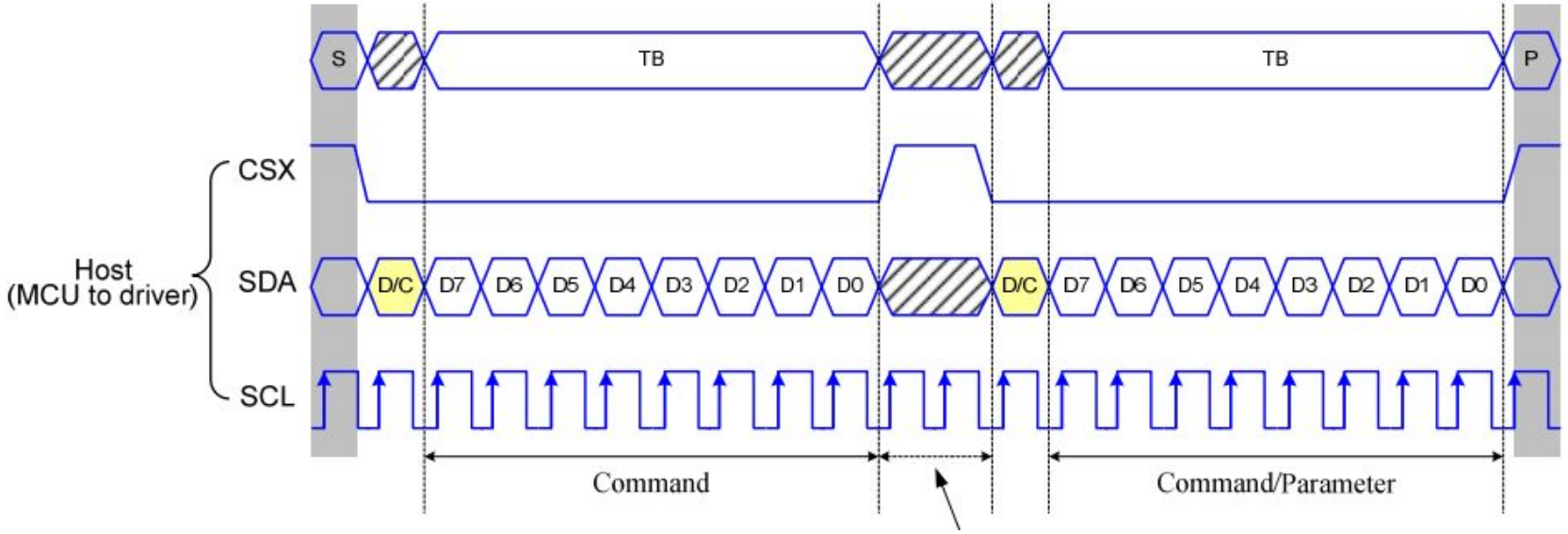


Fig. 8.2.1 3-line serial interface timing

Signal	Symbol	Parameter	Min	Max	Unit	Description
CSX	TCSS	Chip select setup time (write)	15		ns	
	TCSH	Chip select hold time (write)	15		ns	
	TCSS	Chip select setup time (read)	60		ns	
	TSCC	Chip select hold time (read)	65		ns	
	TCHW	Chip select “H” pulse width	40		ns	

Reading datasheets for SPI details - ST7735

SCL	TSCYCW	Serial clock cycle (Write)	66		ns	
	TSHW	SCL “H” pulse width (Write)	30		ns	
	TSLW	SCL “L” pulse width (Write)	30		ns	
	TSCYCR	Serial clock cycle (Read)	150		ns	
	TSHR	SCL “H” pulse width (Read)	60		ns	
	TSLR	SCL “L” pulse width (Read)	60		ns	



Reading datasheets for SPI details - MCP3008

- On-chip sample and hold
- SPI serial interface (modes 0,0 and 1,1)
- Single supply operation: 2.7V - 5.5V
- 200 ksps max. sampling rate at V_{DD} = 5V
- 75 ksps max. sampling rate at V_{DD} = 2.7V

single-ended inputs. Differential Nonlinearity (DNL) and Integral Nonlinearity (INL) are specified at ±1 LSB. Communication with the devices is accomplished using a simple serial interface compatible with the SPI protocol. The devices are capable of conversion rates of up to 200 ksps. The MCP3004/3008 devices operate

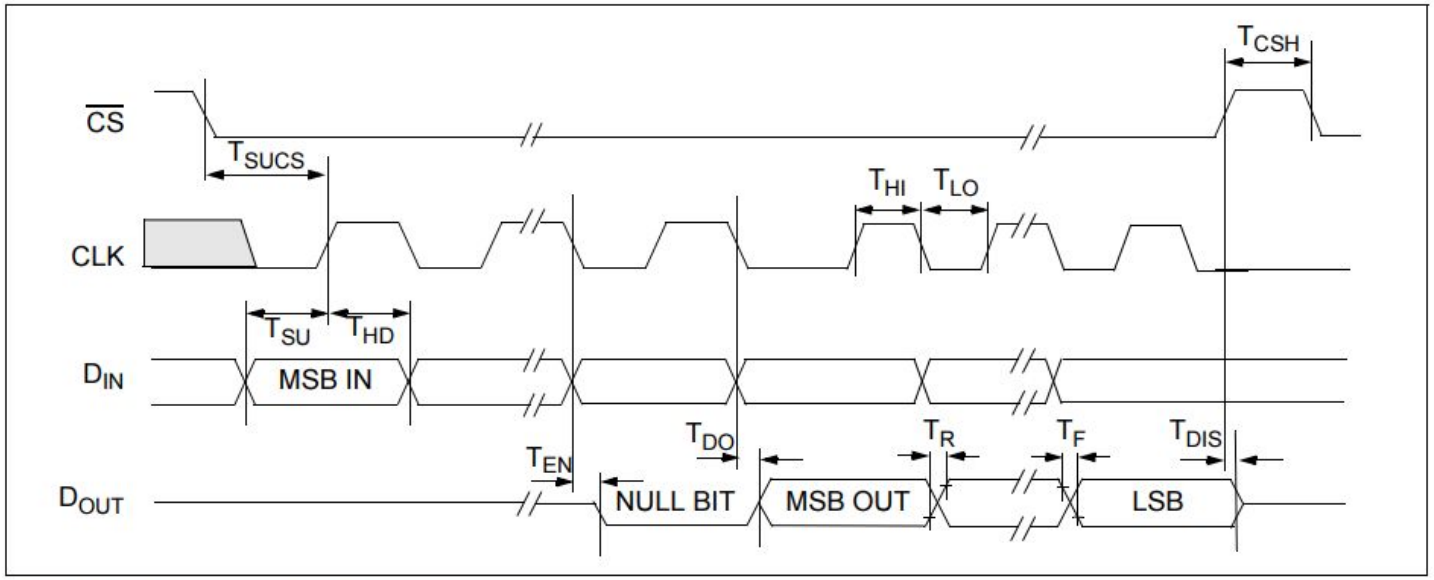
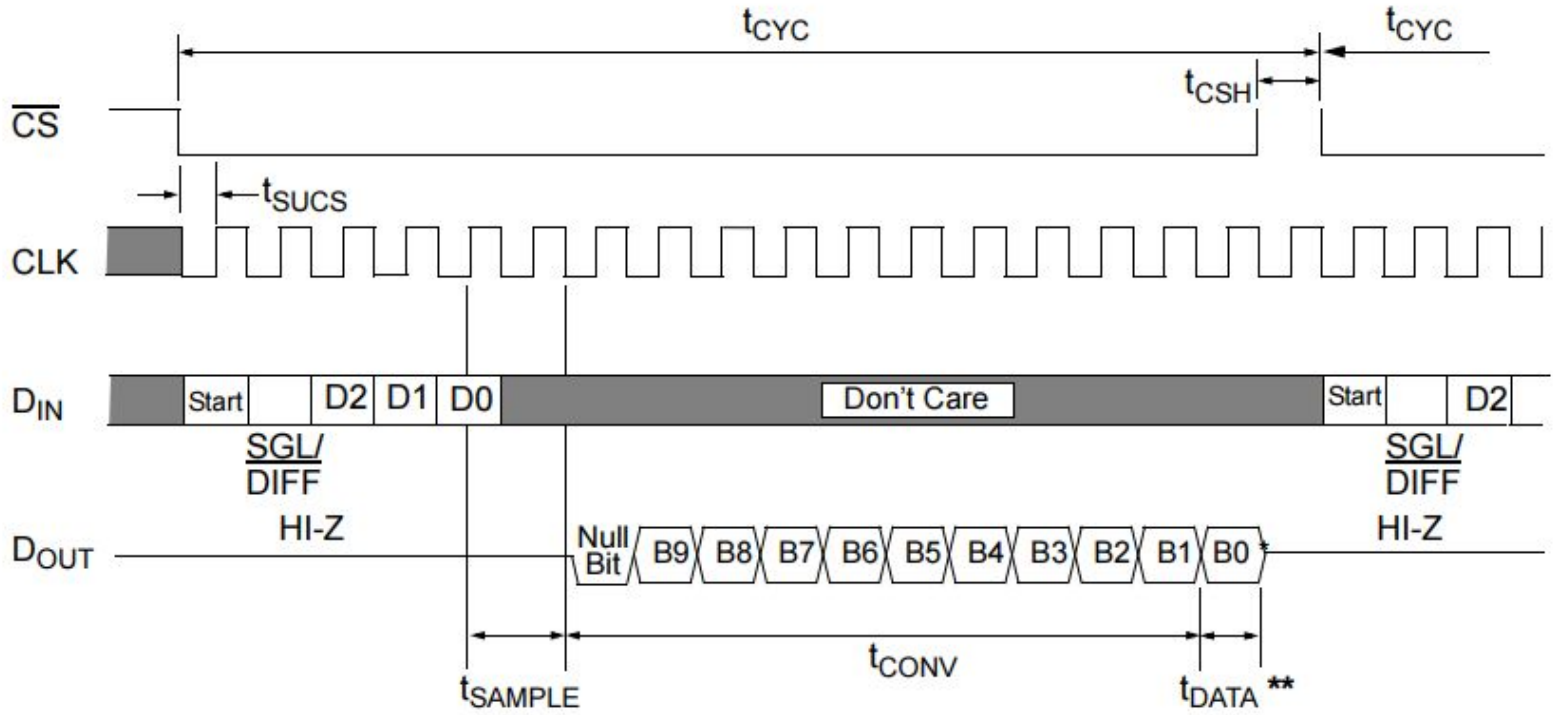


FIGURE 1-1: *Serial Interface Timing.*

Clock High Time	t_{HI}	125	—	—	ns	
Clock Low Time	t_{LO}	125	—	—	ns	
$\overline{CS}$ Fall To First Rising CLK Edge	t_{SUCS}	100	—	—	ns	
$\overline{CS}$ Fall To Falling CLK Edge	t_{CSD}	—	—	0	ns	

Reading datasheets for SPI details - MCP3008



* After completing the data transfer, if further clocks are applied with $\overline{CS}$ low, the A/D converter will output LSB first data, then followed with zeros indefinitely. See Figure 5-2 below.

** t_{DATA}: during this time, the bias current and the comparator powers down while the reference input becomes a high-impedance node.

FIGURE 5-1: *Communication with the MCP3004 or MCP3008.*

MCP3008 via DT

```
* Microchip Analog to Digital Converter (ADC)

The node for this driver must be a child node of a SPI controller, hence
all mandatory properties described in

        Documentation/devicetree/bindings/spi/spi-bus.txt

must be specified.

Required properties:
        - compatible:   Must be one of the following, depending on the
                        model:
                                ...
                                "microchip,mcp3008"
                                ...

Examples:
spi_controller {
        mcp3x0x@0 {
                compatible = "mcp3002";
                reg = <0>;
                spi-max-frequency = <1000000>;
        };
};
```

MCP3008 via DT

- DTS fragment

```
    fragment@1 {
        target = <&spi0>;
        __overlay__ {
            #address-cells = <1>;
            #size-cells = <0>;
            mcp3x0x@0 {
                compatible = "microchip,mcp3008";
                reg = <0>;
                spi-max-frequency = <4000000>;
            };
        };
    };
~
~
```

MCP3008 via board file

- C code fragment

```
static struct spi_board_info my_board_info[] __initdata = {
{
        .modalias       = "mcp320x",
        .max_speed_hz   = 4000000,
        .bus_num        = 0,
        .chip_select    = 0,
},
};

spi_register_board_info(spi_board_info, ARRAY_SIZE(my_board_info));
```

MCP3008 via ACPI

```
Scope (\_SB.SPI1)
{
        Device (MCP3008)
        {
                Name (_HID, "PRP0001")
                Method (_CRS, 0, Serialized) {
                        Name (UBUF, ResourceTemplate () {
                                SpiSerialBus (0x0000, PolarityLow, FourWireMode, 0x08,
                                        ControllerInitiated, 0x003D0900, ClockPolarityLow,
                                        ClockPhaseFirst, "\\_SB.SPI1", 0x00, ResourceConsumer)
                        })
                        Return (UBUF)
                }

                Method (_STA, 0, NotSerialized)
                {
                        Return (0xF)
                }
        }
}
~
```

Protocol Driver

- Standard LInux driver model
- Instantiate a struct spi_driver
  - .driver =
    - .name = “my_protocol”,
    - .pm = &my_protocol_pm_ops,
  - .probe = my_protocol_probe
  - .remove = my_protocol_remove
- Once it probes, SPI I/O may take place using kernel APIs

Kernel APIs

- spi_async()
  - asynchronous message request
  - callback executed upon message complete
  - can be issued in any context
- spi_sync()
  - synchronous message request
  - may only be issued in a context that can sleep (i.e. not in IRQ context)
  - wrapper around spi_async()
- spi_write()/spi_read()
  - helper functions wrapping spi_sync()

Kernel APIs

- spi_read_flash()
  - Optimized call for SPI flash commands
  - Supports controllers that translate MMIO accesses into standard SPI flash commands
- spi_message_init()
  - Initialize empty message
- spi_message_add_tail()
  - Add transfers to the message’s transfer list

Controller Driver

- Standard LInux driver model
- Allocate a controller
  - spi_alloc_master()
- Set controller methods
  - setup() - configure SPI parameters
  - cleanup() - prepare for driver removal
  - prepare_transfer_hardware() - msg arriving soon
  - unprepare_transfer_hardware() - no msgs pending
  - transfer_one_message() - dispatch one msg and queue
  - transfer_one() - dispatch one transfer and queue
- Register a controller
  - spi_register_master()

Userspace Driver

- spidev
- Slave devices bound to the spidev driver yield:
  - /sys/class/spidev/spidev[bus].[cs]
  - /dev/spidev[bus].[cs]
- Character device
  - open()/close()
  - read()/write() are half duplex
  - ioctl()
    - SPI_IOC_MESSAGE - raw messages, full duplex and chip select control
    - SPI_IOC_[RD|WR]_* - set SPI parameters

Userspace Help

- Docs
  - *Documentation/spi/spidev*
- Examples
  - *tools/spi/spidev_fdx.c*
  - *tools/spi/spidev_test.c*
- Helper libaries
  - https://github.com/jackmitch/libsoc
  - https://github.com/doceme/py-spidev

Linux SPI Performance

Performance considerations

- Be aware of underlying DMA engine or SPI controller driver behavior.
  - e.g. OMAP McSPI hardcoded to PIO up to 160 byte transfer
- sync versus async API behavior
  - async may be suitable for higher bandwidth where latency is not a concern (some network drivers)
  - sync will attempt to execute in caller context (as of 4.x kernel) avoiding sleep and reducing latency

Performance considerations

- Use cs_change wisely. Note the details from include/linux/spi/spi.h:

```
* All SPI transfers start with the relevant chipselect active.  Normally
* it stays selected until after the last transfer in a message.  Drivers
* can affect the chipselect signal using cs_change.
*
* (i) If the transfer isn't the last one in the message, this flag is
* used to make the chipselect briefly go inactive in the middle of the
* message.  Toggling chipselect in this way may be needed to terminate
* a chip command, letting a single spi_message perform all of group of
* chip transactions together.
*
* (ii) When the transfer is the last one in the message, the chip may
* stay selected until the next transfer.  On multi-device SPI busses
* with nothing blocking messages going to other devices, this is just
* a performance hint; starting a message to another device deselects
* this one.  But in other cases, this can be used to ensure correctness.
* Some devices need protocol transactions to be built from a series of
* spi_message submissions, where the content of one message is determined
* by the results of previous messages and where the whole transaction
* ends when the chipselect goes intactive.
*
```

Performance tools

- Debug/visibility tools critical to any hardware focused work
- Logic analyzer
  - http://elinux.org/Logic_Analyzers
  - https://sigrok.org/wiki/Supported_hardware#Logic_analyzers
- *drivers/spi/spi-loopback-test*
- SPI subsystem statistics
  - /sys/class/spi_master/spiB/spiB.C/statistics
    - messages, transfers, errors, timedout
    - spi_sync, spi_sync_immediate, spi_async
    - transfer_bytes_histo_*

Linux SPI Future

Slave Support

- Hard real time issues on Linux due to full duplex nature of SPI.
- Useful if considering limited use cases
  - Pre-existing responses
  - Commands sent to slave
- RFC v2 patch series
  - https://lkml.org/lkml/2016/9/12/1065
- Registering a controller works just like a master
  - spi_alloc_slave()

Slave Support

- /sys/class/spi_slave/spiB/slave for each slave controller
- slave protocol drivers can be bound via sysfs
  - echo slave-foo > /sys/class/spi_slave/spi3/slave
- Two slave protocol drivers provided as an example
  - spi-slave-time (provides latest uptime to master)
  - spi-slave-system-control (power off, reboot, halt system)

Questions?