
Cyber-Physical Systems

Communication



UNIVERSITY
AT ALBANY
State University of New York

IECE 553/453– Fall 2021

Prof. Dola Saha

Why do we need Communication?

- Connect different systems together
 - Two embedded systems
 - A desktop and an embedded system
- Connect different chips together in the same embedded system
 - MCU to peripheral
 - MCU to MCU



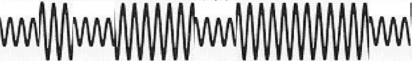
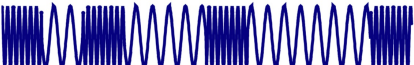
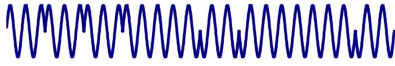
How much can we transmit?

$$C = W \log_2 \left(\frac{S + N}{N} \right)$$

- Shannon's noisy channel coding theorem
 - Says you can achieve error-free communicate at any
- Rate up to the *channel capacity*, and can't do any better
 - C: channel capacity, in bits / s
 - W: bandwidth amount of frequency “real estate”, in Hz (cycles / s)
 - S: Signal power
 - N: Noise power

Communication Methods

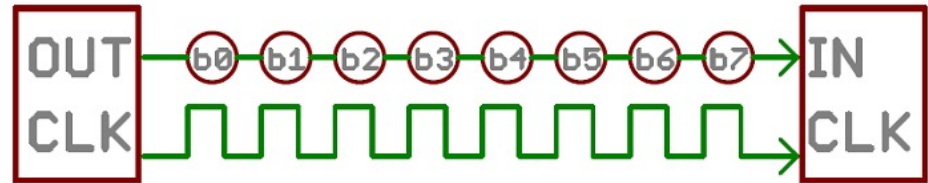
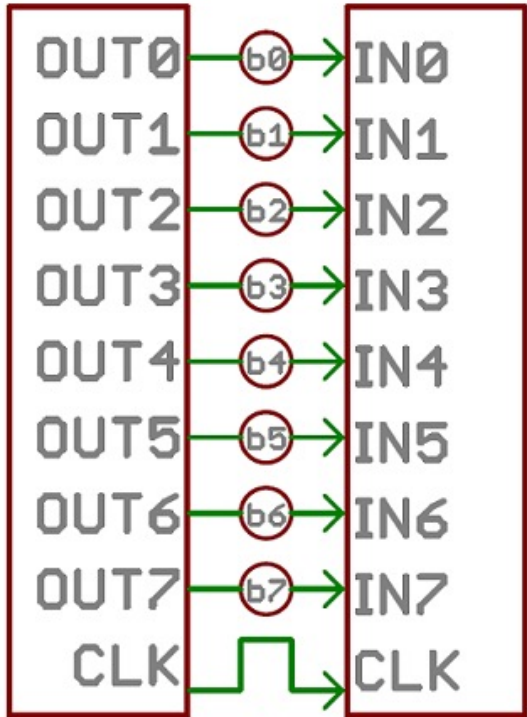
- Different physical layers methods: wires, radio frequency (RF), optical (IR)
- Different encoding schemes: amplitude, frequency, and pulse-width modulation

Modulation Technique	Waveform
No modulation (Baseband)	
On-Off Keying (OOK)	
Amplitude Modulation	
Frequency Shift Keying (FSK)	
Binary Phase Shift Keying (BPSK)	
Direct Sequence Spread Spectrum (DSSS), etc	

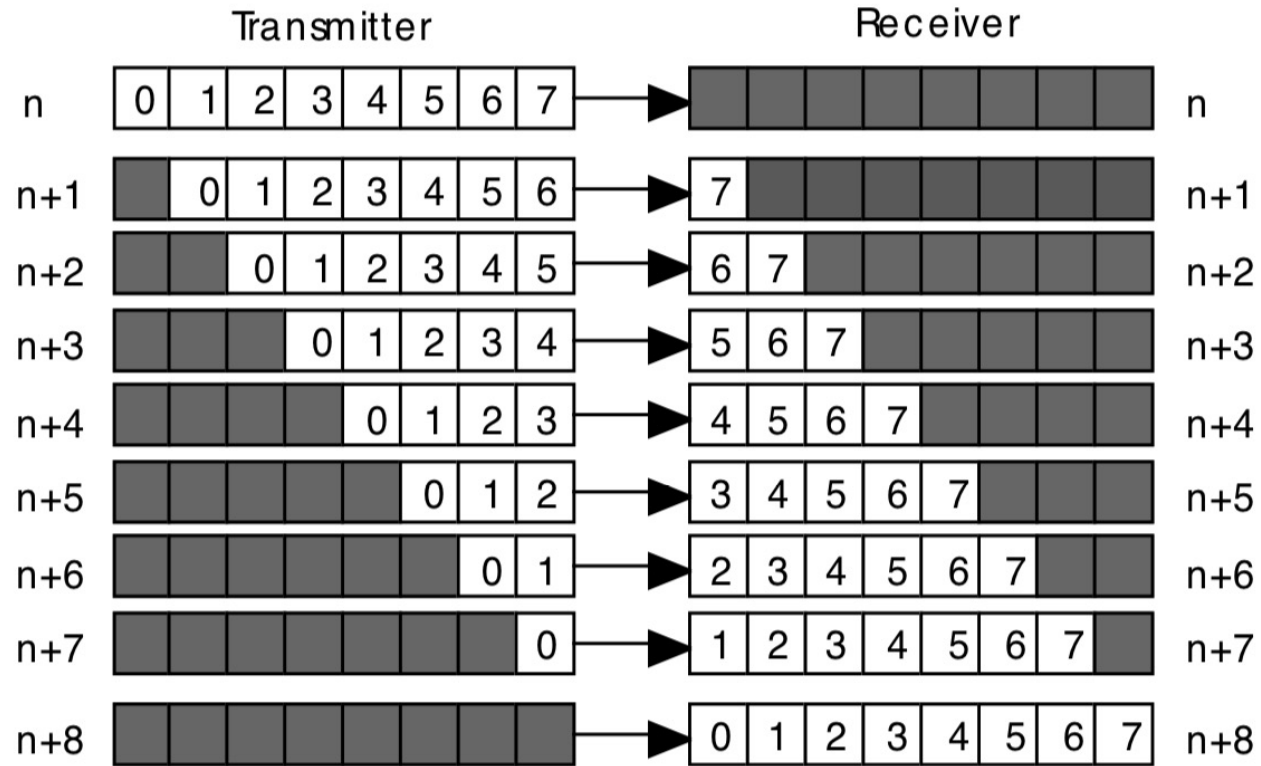
Dimensions to consider

- bandwidth – number of wires – serial/parallel
- speed – bits/bytes/words per second
- timing methodology – synchronous or asynchronous
- number of destinations/sources
- arbitration scheme – daisy-chain, centralized, distributed
- protocols – provide some guarantees as to correct communication

Parallel and Serial Bus



Serial

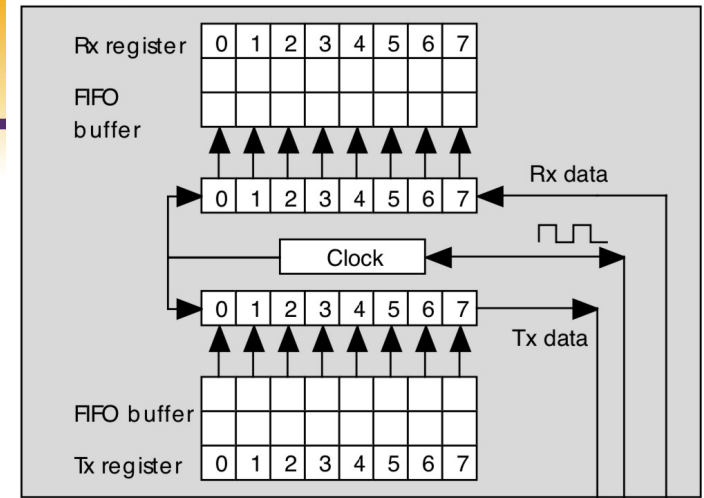


Interrupt:
transmitter empty

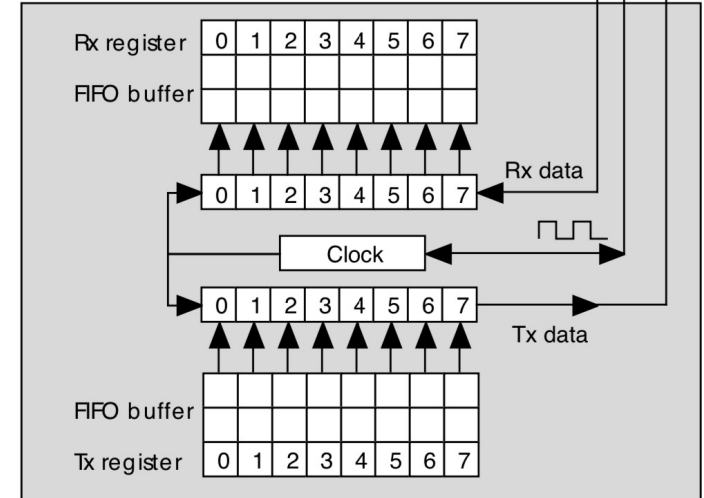
Interrupt:
receiver full

Serial Comm with buffer

Processor



Peripheral



Parallel and Serial Communication

➤ Serial

- Single wire or channel to transmit information one bit at a time
- Requires synchronization between sender and receiver
- Sometimes includes extra wires for clock and/or handshaking
- Good for inexpensive connections (e.g., terminals)
- Good for long-distance connections (e.g., LANs)

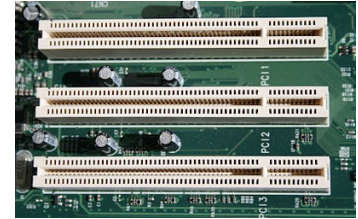
➤ Parallel

- Multiple wires to transmit information one byte or word at a time
- Good for high-bandwidth requirements (CPU to disk)
- Crosstalk creates interference between multiple wires
 - Length of link & Higher frequency of transmission increases crosstalk
- More expensive wiring/connectors/current requirements

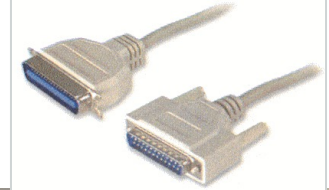
Parallel vs. Serial Digital Interfaces

- Parallel (one wire per bit)
 - ATA: Advanced Technology Attachment
 - PCI: Peripheral Component Interface
 - SCSI: Small Computer System Interface
- Serial (one wire per direction)
 - RS-232
 - SPI: Serial Peripheral Interface bus
 - I2C: Inter-Integrated Circuit
 - USB: Universal Serial Bus
 - SATA: Serial ATA
 - Ethernet, IrDA, Firewire, Bluetooth, DVI, HDMI
- Mixed (one or more “lanes”)
 - PCIe: PCI Express

PCI



SCSI



USB



RS-232

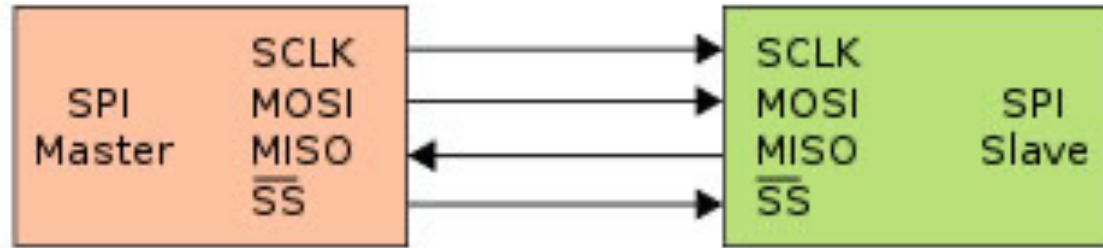


Parallel vs Serial Digital Interfaces

- Parallel connectors have been replaced by Serial
 - Significant crosstalk/inter-wire interference for parallel connectors
 - Maintaining synchrony across the multiple wires
 - Serial connection speeds can be increased by increasing transmission freq, but parallel crosstalk gets worse at increased freq

Serial Peripheral Interface (SPI)

- Synchronous full-duplex communication
- Can have multiple slave devices
- No flow control or acknowledgment
- Slave cannot communicate with slave directly.



Serial Peripheral Interface

http://upload.wikimedia.org/wikipedia/commons/thumb/e/ed/SPI_single_slave.svg/350px-SPI_single_slave.svg.png

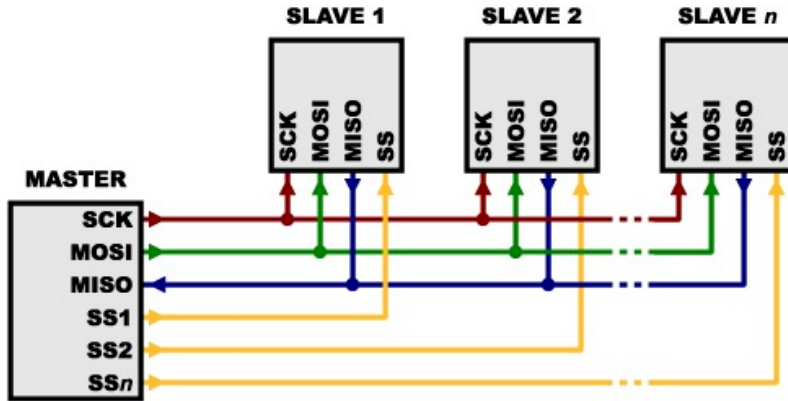
SCLK: serial clock

MOSI: master out slave in

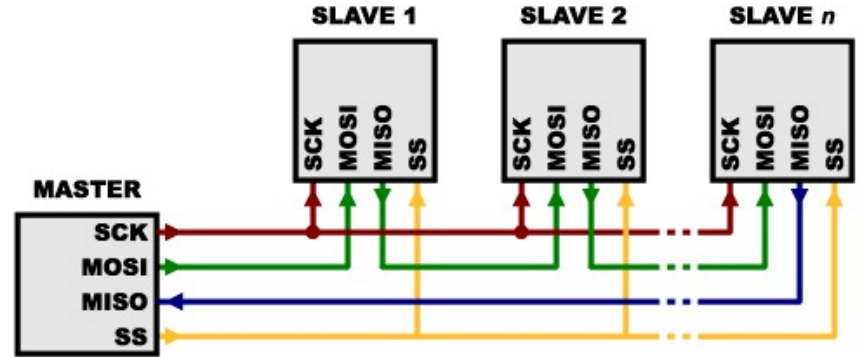
SS: slave select (active low)

MISO: master in slave out

SPI – Point-to-point and Daisy Chain



Point-to-point



Daisy Chain

SCLK: serial clock

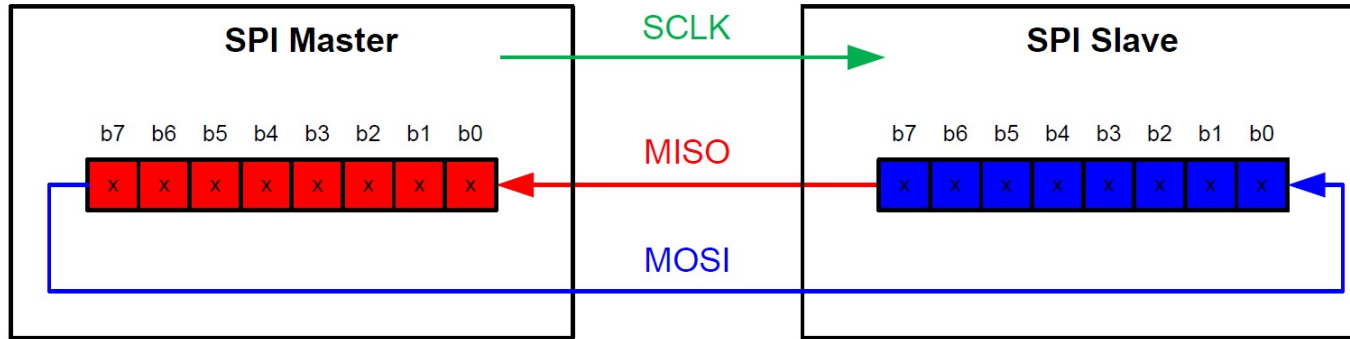
MOSI: master out slave in

SS: slave select (active low)

MISO: master in slave out

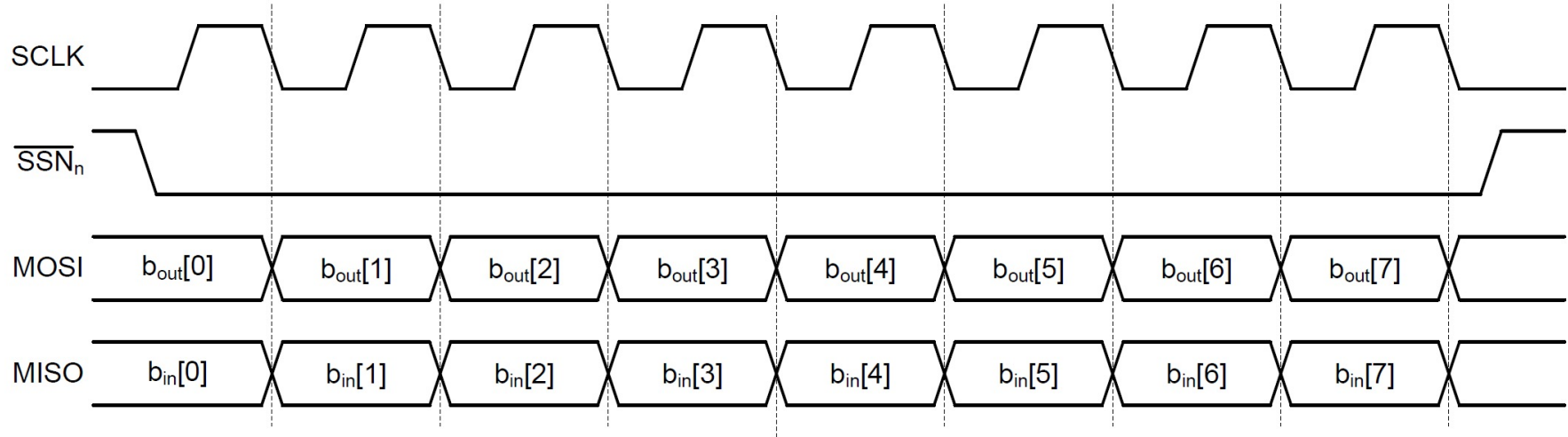
Pictures: <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/>

Data Exchange



- Master has to provide clock to slave
- Synchronous exchange: for each clock pulse, a bit is shifted out and another bit is shifted in at the same time. This process stops when all bits are swapped.
- Only master can start the data transfer

Clock



Clock Phase and Polarity

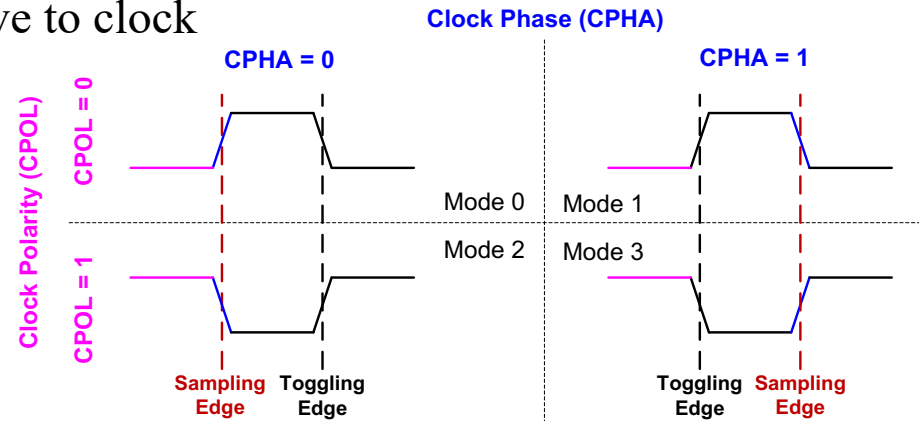
➤ CPHA (Clock PHase)

- determines when data goes on bus relative to clock
- = 0 data Tx edge active to idle
- = 1 data Tx edge idle to active

➤ CPOL (Clock POLarity)

- sets polarity of Clk during idle state
- =0 clock idles low between transfers
- =1 clock idles high between transfers

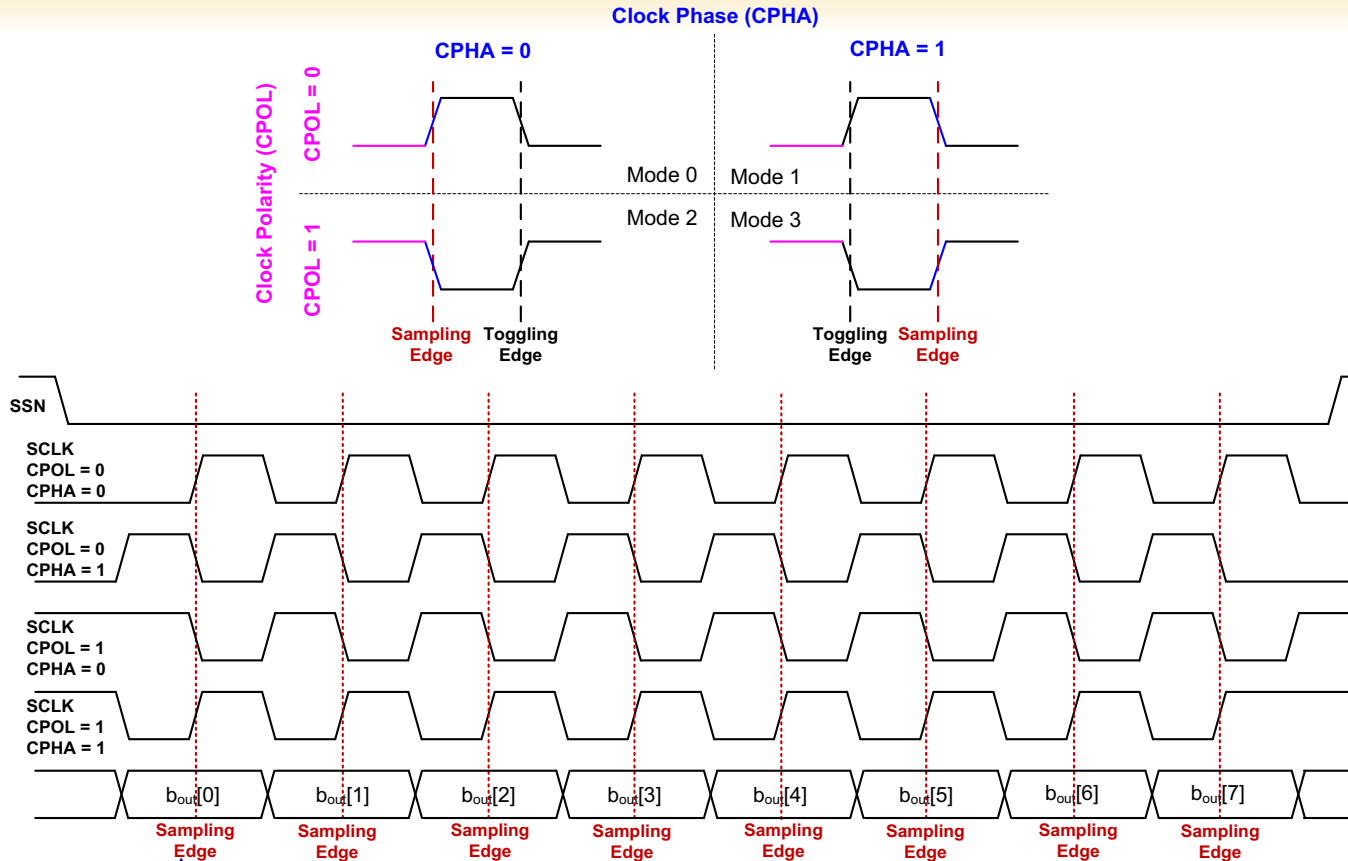
- Combination of CPOL and CPHA determines the clock edge for transmitting and receiving.



SPI Modes

SPI Mode	CPOL	CPHA	Clock Polarity in Idle State	Clock Phase Used to Sample and/or Shift the Data
0	0	0	Logic low	Data sampled on rising edge and shifted out on the falling edge
1	0	1	Logic low	Data sampled on the falling edge and shifted out on the rising edge
2	1	1	Logic high	Data sampled on the falling edge and shifted out on the rising edge
3	1	0	Logic high	Data sampled on the rising edge and shifted out on the falling edge

Clock Phase and Polarity



SPI: Pros and Cons

➤ Pros

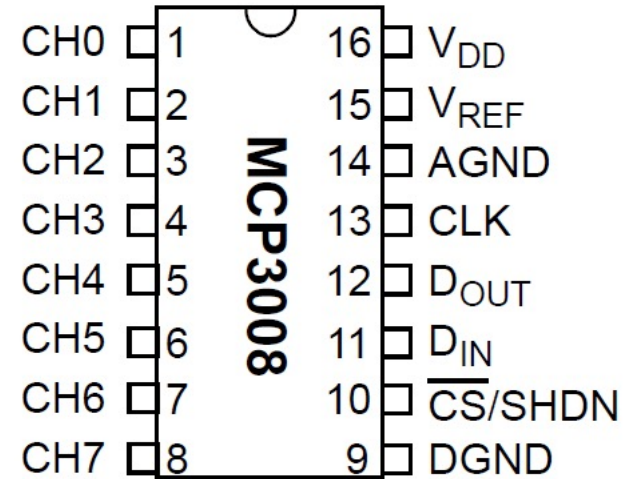
- Simplest way to connect 1 peripheral to a micro
- Fast (10s of Mbits/s, not on MSP) because all lines actively driven, unlike I2C
- Clock does not need to be precise
- Nice for connecting 1 slave

➤ Cons

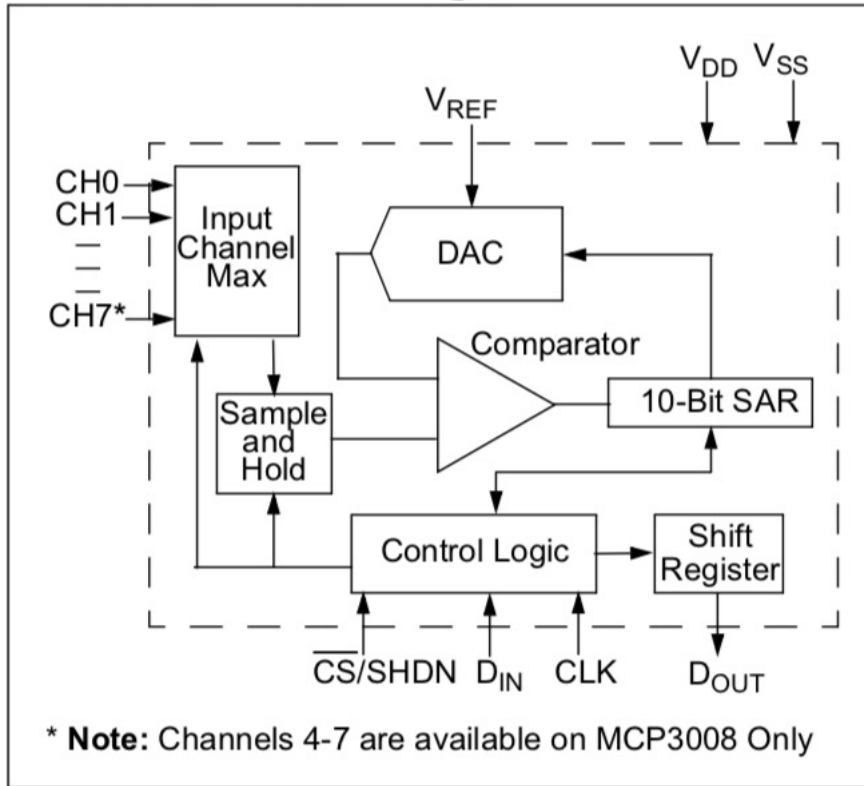
- No built-in acknowledgement of data
- Not very good for multiple slaves
- Requires 4 wires
- 3 wire variants exist...some get rid of full duplex and share a data line, some get rid of slave select

Analog to Digital Converter

- DGND : digital ground pin for the chip
- CS : chip select.
- DIN : data in from the MC itself.
- DOUT: data out pin.
- CLK: clock pin.
- AGND: analog ground and obviously connects to ground.
- VREF: analog reference voltage. You can change this if you want to change the scale. You probably want to keep it the same so keep this as 3.3v.
- VDD: positive power pin for the chip.



MCP 3008



$$\text{Digital Output Code} = \frac{1024 \times V_{IN}}{V_{REF}}$$

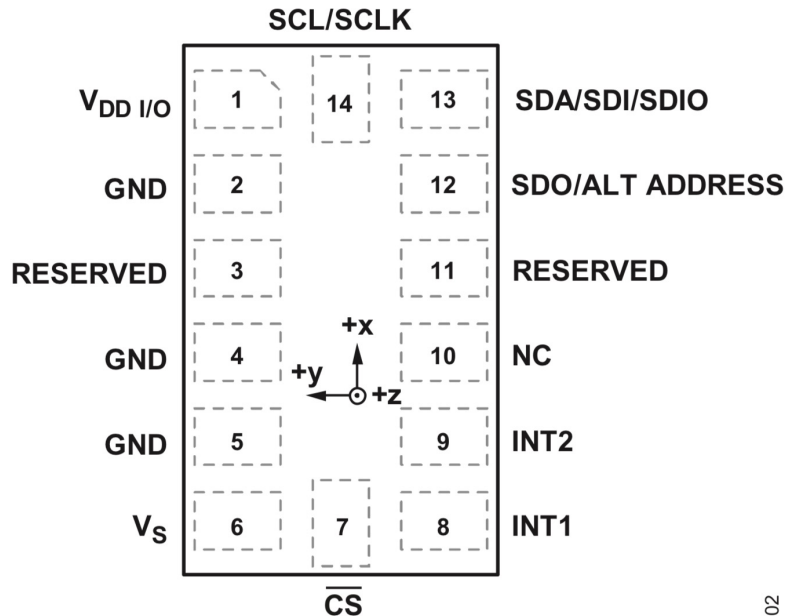
Where:

V_{IN} = analog input voltage

V_{REF} = analog input voltage

ADXL345

**ADXL345
TOP VIEW
(Not to Scale)**



NOTES
1. NC = NO INTERNAL CONNECTION.

Serial Data Input (SDI)

07925-002

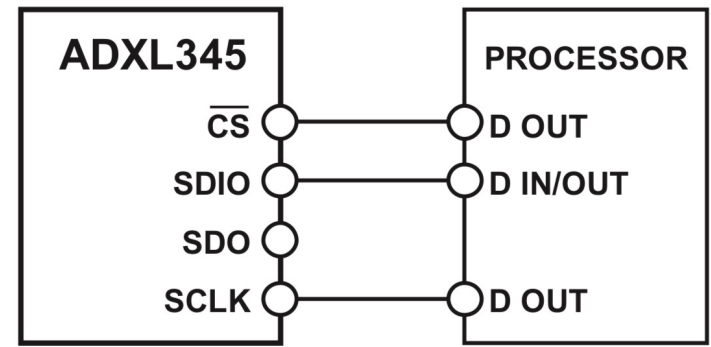


Figure 34. 3-Wire SPI Connection Diagram

07925-004

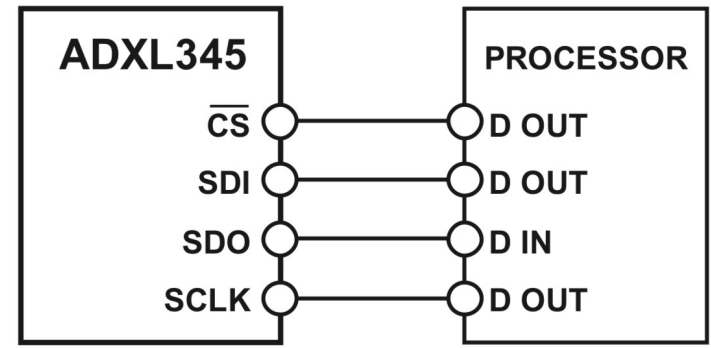
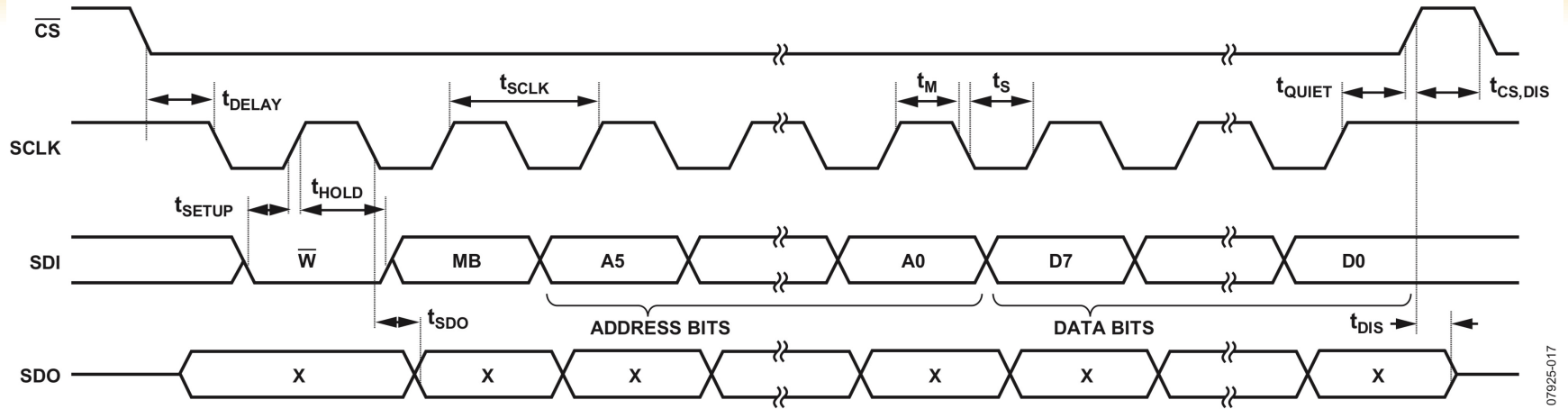


Figure 35. 4-Wire SPI Connection Diagram

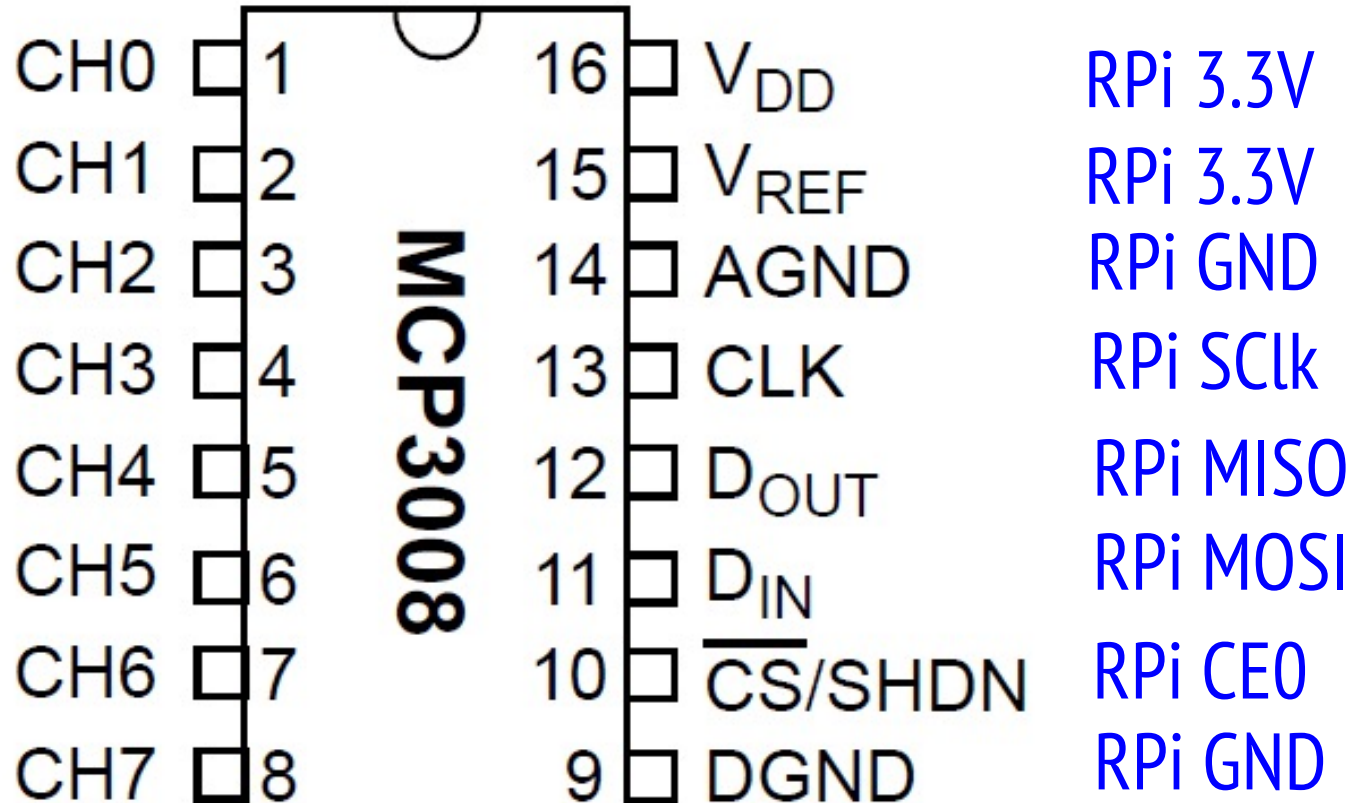
07925-003

Communication

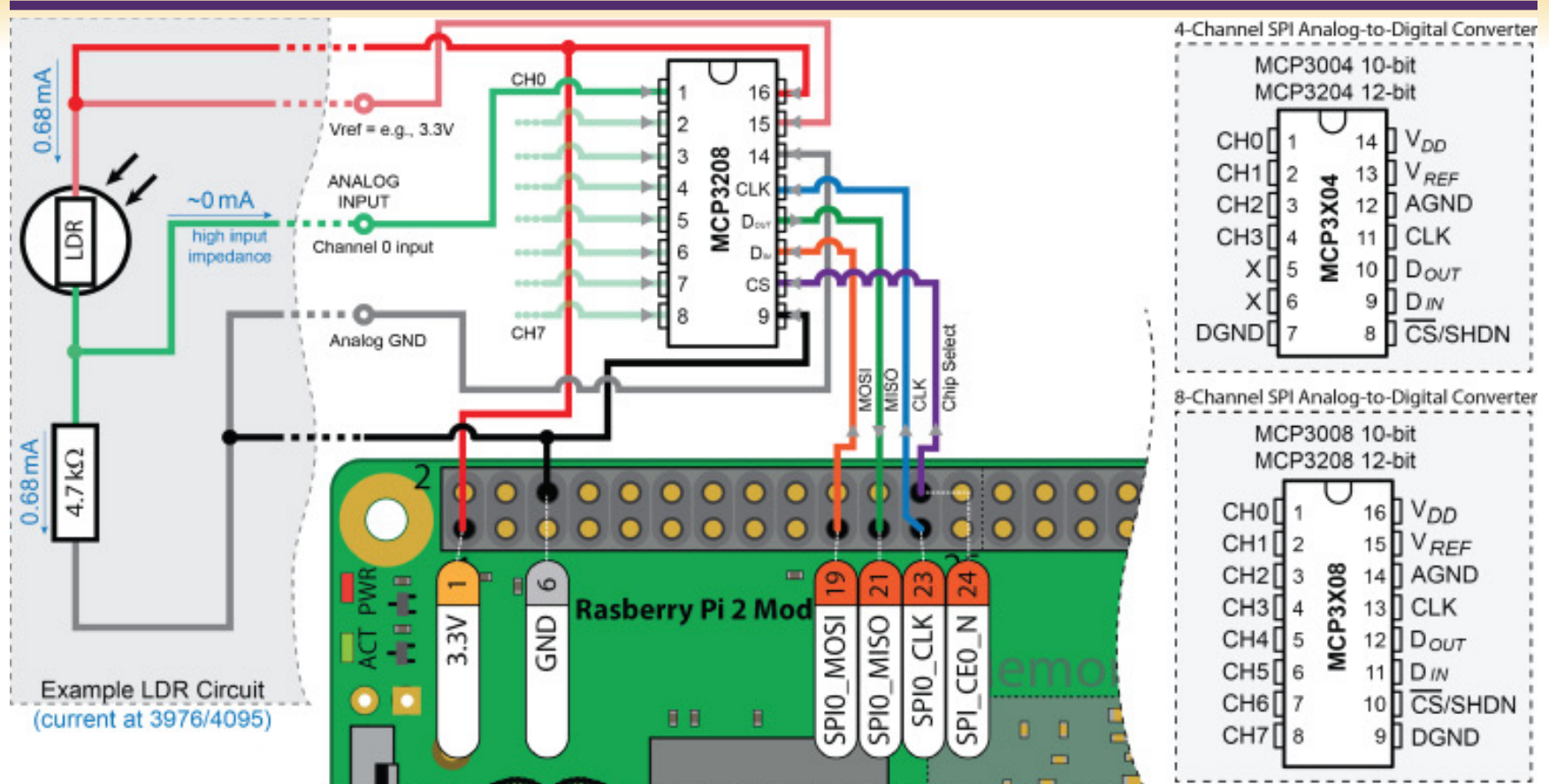


07925-017

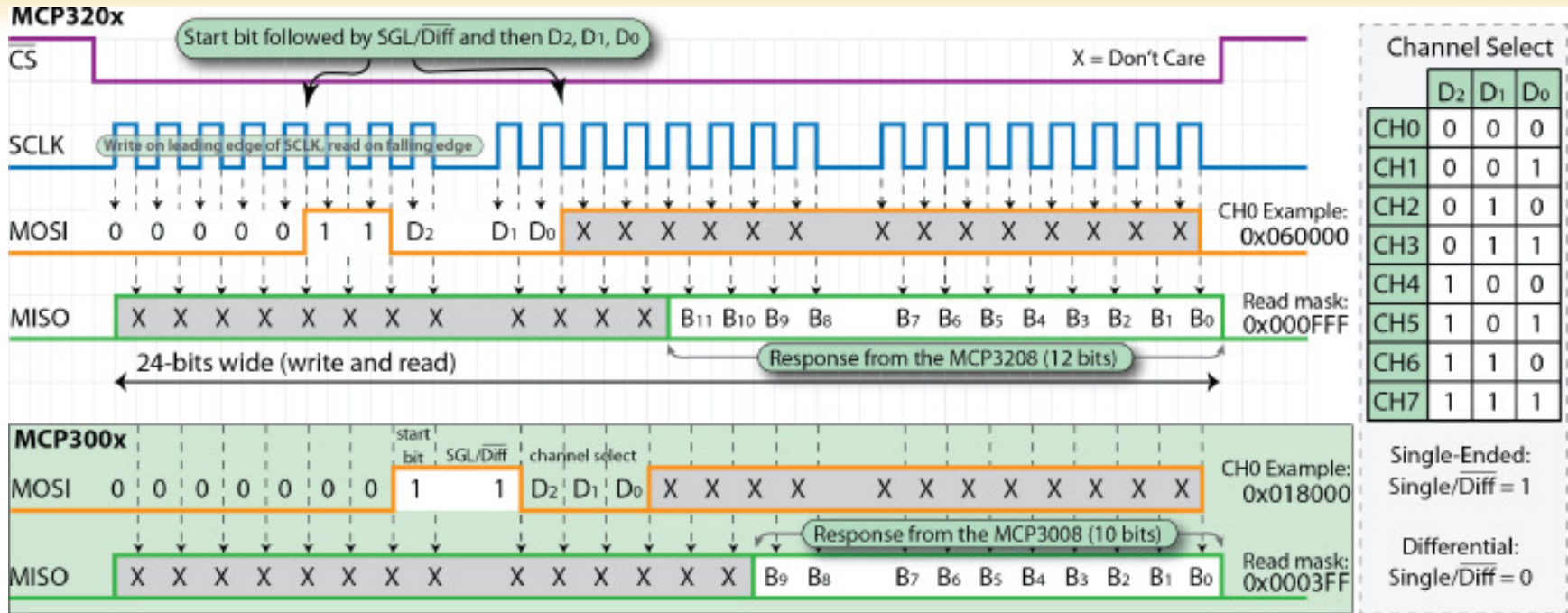
Analog to Digital Converter



Connect a Sensor



Channel Select



- The device will begin to sample the analog input on the fourth rising edge of the clock after the start bit has been received. The sample period will end on the falling edge of the fifth clock following the start bit.

Enable SPI in Raspberry PI

- sudo raspi-config
- 5 Interfacing Options
- P4 SPI
- Would you like the SPI interface to be enabled?
 - Select Yes
- The SPI interface is enabled
 - Select OK
- Finish

Has SPI been really enabled?

- `sudo ls /dev/spi*`
- `/dev/spidev0.0 /dev/spidev0.1`

SPI Bus on Linux

- `lsmod | grep spi` It formats the contents of the file `/proc/modules`, which contains information about the status of all currently-loaded LKMs.
- `modprobe spidev` **modprobe** intelligently adds or removes a module from the Linux kernel
- `modprobe spi_bcm2835`
- `dmesg | grep spi` display messages from the linux kernel ring buffer

SPI Using User->Kernel Modules

➤ ioctl

- `/usr/include/asm-generic/ioctl.h`

➤ spidev

- `/usr/include/linux/spi/spidev.h`
- <https://github.com/raspberrypi/tools/blob/master/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian/arm-linux-gnueabihf/libc/usr/include/linux/spi/spidev.h>

➤ Kernel Module

- <https://github.com/raspberrypi/linux/blob/rpi-3.12.y/drivers/spi/spi-bcm2835.c>

ioctl() – Input/Output Control

- **int ioctl(int *fd*, unsigned long *request*, ...);**
- The **ioctl()** system call manipulates the underlying device parameters of special files.
- Input Arguments
 - *fd* – File Descriptor
 - *request* – Device dependent request code
 - Third Argument – Integer value of a pointer to data for transfer
- Return
 - 0 on success.
 - -1 on error.

spi_ioc_transfer structure

```
struct spi_ioc_transfer {
    __u64      tx_buf;
    __u64      rx_buf;

    __u32      len;
    __u32      speed_hz;

    __u16      delay_usecs;
    __u8       bits_per_word;
    __u8       cs_change;
    __u8       tx_nbits;
    __u8       rx_nbits;
    __u16      pad;

    /* If the contents of 'struct spi_ioc_transfer' ever change
     * incompatibly, then the ioctl number (currently 0) must change;
     * ioctls with constant size fields get a bit more in the way of
     * error checking than ones (like this) where that field varies.
     *
     * NOTE: struct layout is the same in 64bit and 32bit userspace.
     */
};
```


SPI Dev Interface

- <https://www.kernel.org/doc/Documentation/spi/spidev>
- `/dev/spidevB.C` (B=bus, C=slave number).
 - On RPi it is `/dev/spidev0.0`
- To open the device:
 - `fd=open("/dev/spidev0.0",O_RDWR);`

SPI Dev Interface

➤ To set the mode:

- `int mode=SPI_MODE_0;`
- `result = ioctl(spi_fd , SPI_IOC_WR_MODE , &mode);`

➤ To set the bit order:

- `int lsb_mode =0;`
- `result = ioctl(spi_fd, SPI_IOC_WR_LSB_FIRST, &lsb_mode);`

SPI Dev Interface

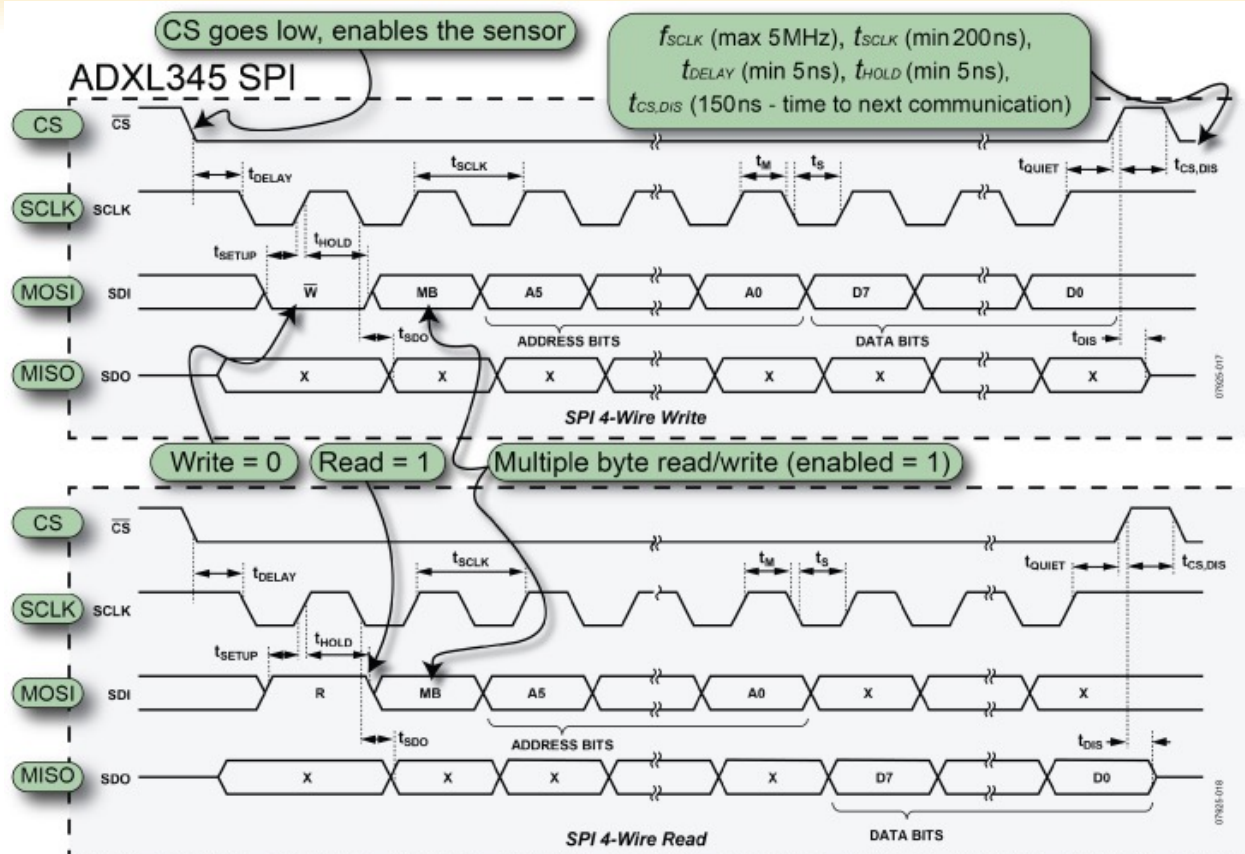
➤ To transfer:

- `ret = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);`

➤ To close:

- `close(fd);`

ADXL 345



Underlying image courtesy of Analog Devices, Inc.

MCP 3008 Data Transfer

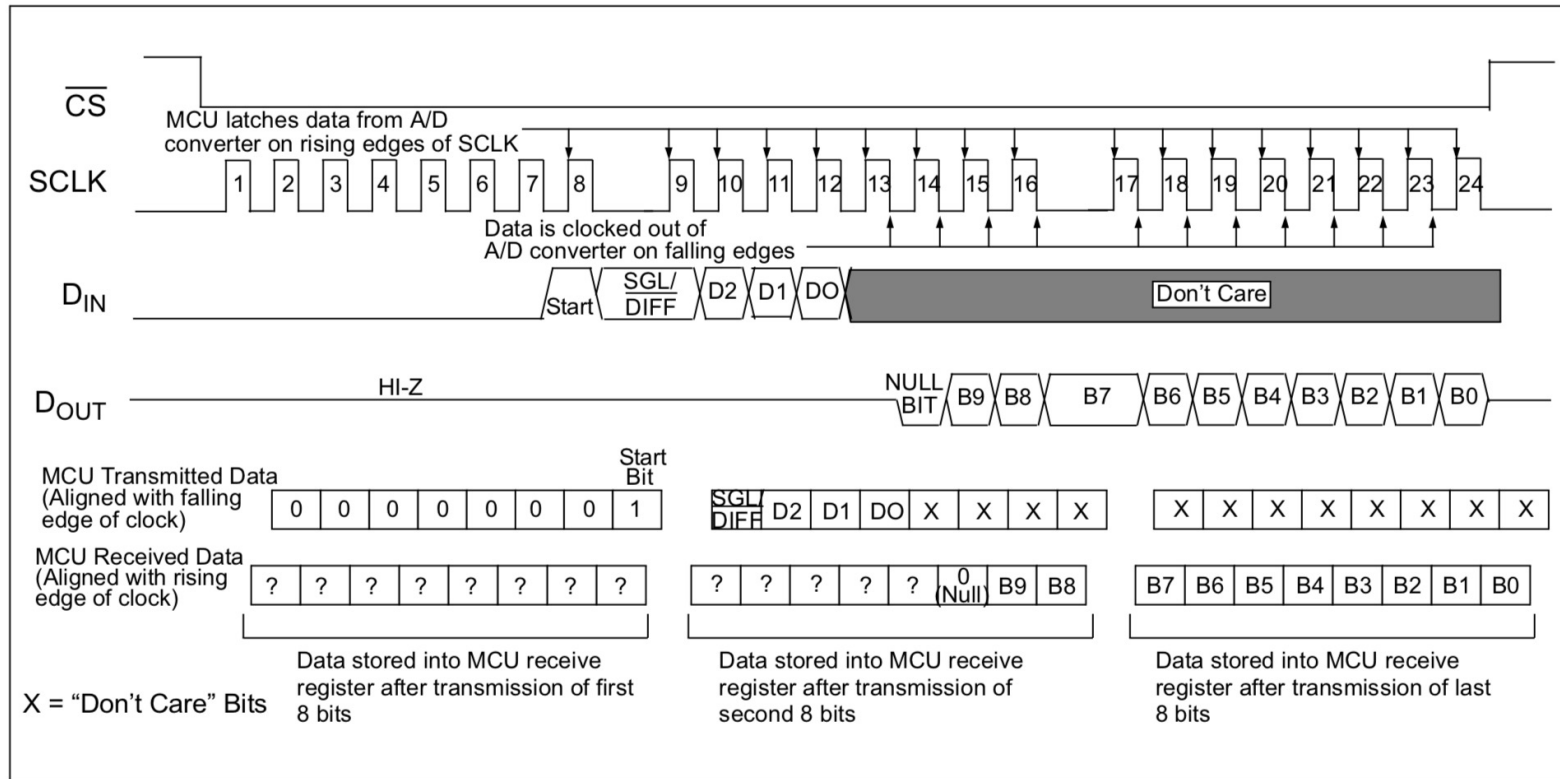


FIGURE 6-1: SPI Communication with the MCP3004/3008 using 8-bit segments (Mode 0,0: SCLK idles low).

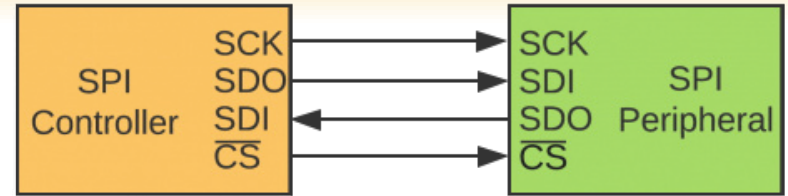
Name Change for SPI Ports

➤ Adafruit:

- MOSI: main output, secondary input
- MISO: main input, secondary output

➤ Sparkfun:

- **SDO** – Serial Data Out.
- **SDI** – Serial Data In.
- **CS** – Chip Select.
- **COPI** (controller out / peripheral in).
- **CIPO** (controller in / peripheral out).
- **SDIO** – Serial Data In/Out. A bi-directional serial signal.



Deprecated signal names:

- ✗ **MOSI** – Master Out Slave In
- ✗ **MISO** – Master In Slave Out
- ✗ **SS** – Slave Select
- ✗ **MOMI** Master Out Master In
- ✗ **SOSI** Slave Out Slave In

<https://hackaday.com/2020/06/29/updating-the-language-of-spi-pin-labels-to-remove-casual-references-to-slavery/>

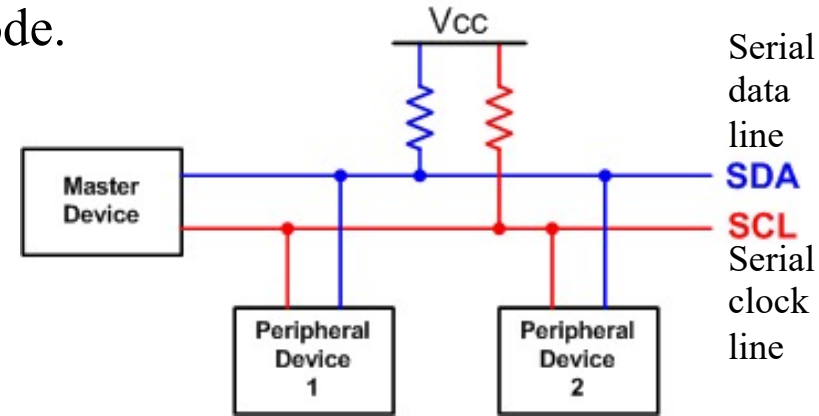
https://www.sparkfun.com/spi_signal_names

Inter-Integrated Circuit (I2C)

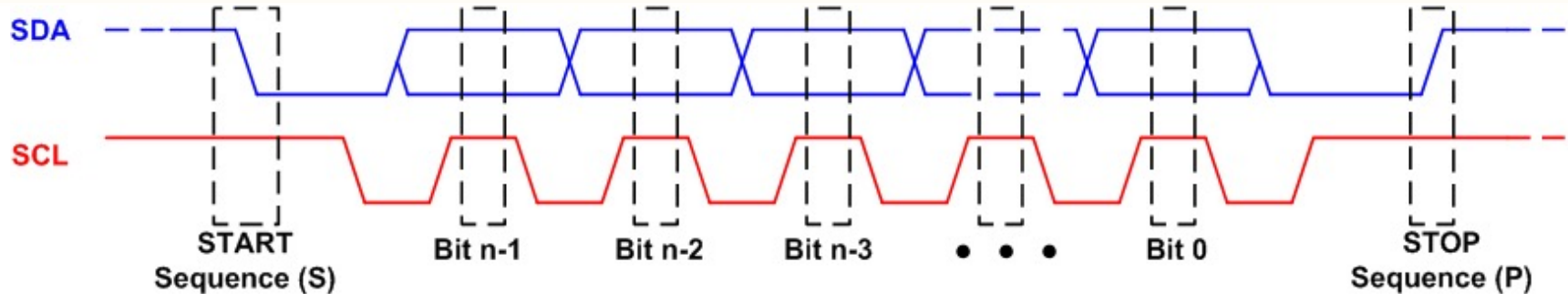
- Designed for low-cost, medium data rate applications by Philips in the early 1980's
 - Original purpose: connect a CPU to peripheral chips in a TV-set
 - Today: a de-facto standard for 2-wire communications
 - Since October 10, 2006, no licensing fees are required to implement the I²C protocol. However, fees are still required to obtain I²C slave addresses allocated by NXP (acquired Philips).
- Characteristics
 - Serial, byte-oriented
 - Multi-master, multi-slave
 - Two bidirectional open-drain lines, plus ground
 - Serial Data Line (SDA)
 - Serial Clock Line (SCL)
 - SDA and SCL need to pull up with resistors

Inter-Integrated Circuit (I2C)

- A master device, such as the RPi, controls the bus, and many addressable slave devices can be attached to the same two wires.
- Up to 100 kbit/s in the standard mode, up to 400 kbit/s in the fast mode, and up to 3.4 Mbit/s in the high-speed mode.
- SDA and SCL have to be open-drain
 - Connected to positive if the output is 1
 - In high impedance state if the output is 0
- Each Device has a unique address (7, 10 or 16 bits). Addr 0 for broadcast



Timing Diagram

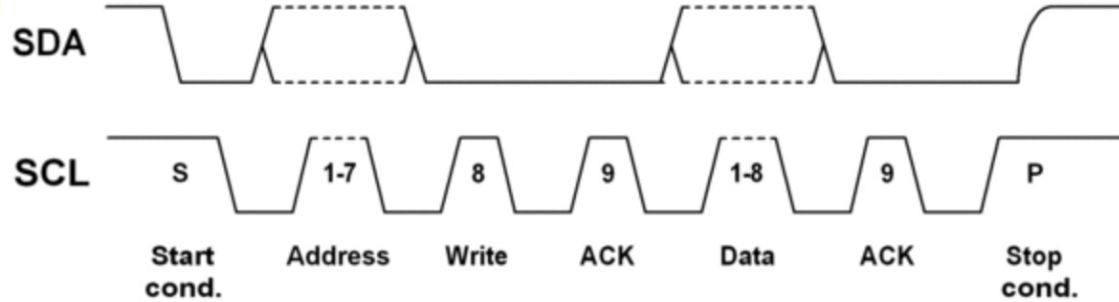


- A **START** condition is a high-to-low transition on SDA when SCL is high.
- A **STOP** condition is a low to high transition on SDA when SCL is high.
- The address and the data bytes are sent most significant bit first.
- Master generates the clock signal and sends it to the slave during data transfer

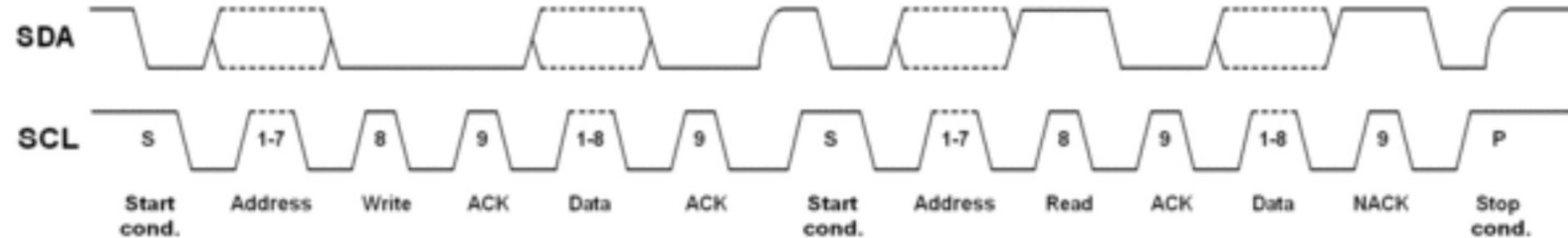
Example: Write 1 byte to device register

- Master sends a *start bit* (i.e., it pulls SDA low, while SCL is high).
- While the clock toggles, the *7-bit slave address* is transmitted one bit at a time.
- A *read bit* (1) or *write bit* (0) is sent, depending on whether the master wants to read or write to/from a slave register.
- The slave responds with an *acknowledge bit* (ACK = 0).
- In write mode, the master sends a byte of data one bit at a time, after which the slave sends back an ACK bit. To write to a register, the register address is sent, followed by the data value to be written.
- Finally, to conclude communication, the master sends a *stop bit* (i.e., it allows SDA to float high, while SCL is high).

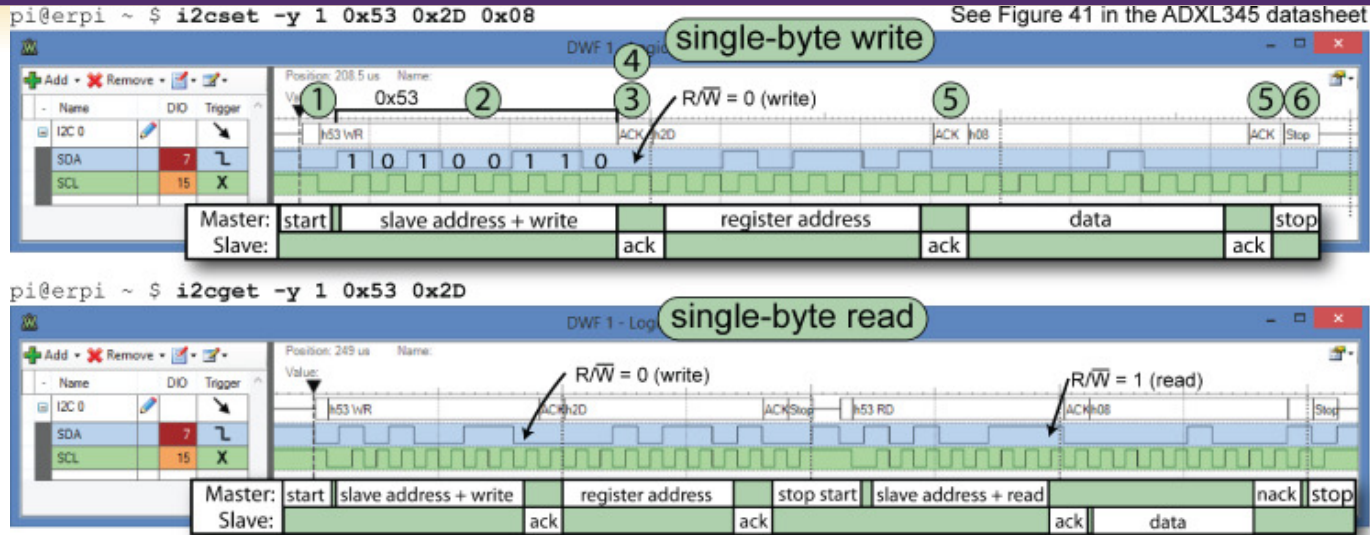
I2C Addressing



Repeated Start Condition



Example Use

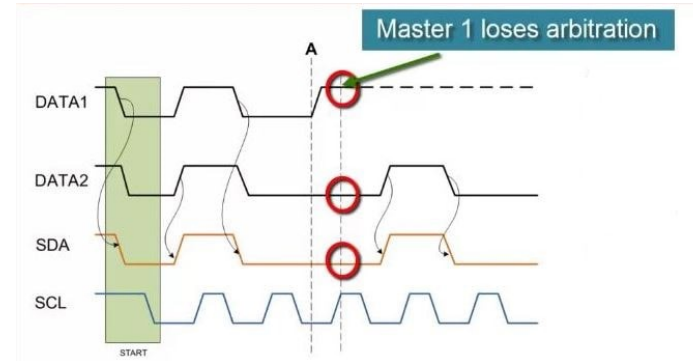


1. The master sends a *start bit* (i.e., it pulls SDA low, while SCL is high)
2. While the clock toggles, the 7-bit slave address is transmitted one bit at a time
3. A read bit (1) or write bit (0) is sent, depending on whether the master wants to read or write
4. The slave responds with an *acknowledge bit* (ACK = 0).
5. In write mode, the master sends a byte of data one bit at a time, after which the slave sends back an ACK bit. To write to a register, the register address is sent, followed by the data value to be written.
6. The master sends a *stop bit* (i.e., it allows SDA to float high, while SCL is high)



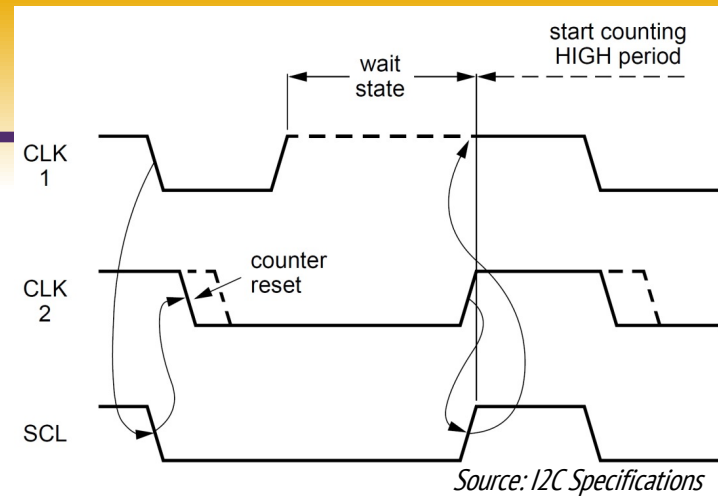
Multiple Masters

- “**Wired-AND**” bus: A sender can pull the lines to low, even if other senders are trying to drive the lines to high
- In single master systems, arbitration is not needed.
- **Arbitration** for multiple masters:
 - During data transfer, the master constantly checks whether the SDA voltage level matches what it has sent.
 - When two masters generate a START setting concurrently, the first master which detects SDA low while it has actually intended to set SDA high will lose the arbitration and let the other master complete the data transfer.
 - Losing Master goes to Slave Mode
 - Retries transmission after the STOP Bit



Clock Synchronization

- Clock synchronization is needed when there are multiple masters.
- **Wired-AND** connection for clock synchronization
 - Each master has a counter. Counter resets if SCL goes LOW. When the counter counts down to zero, the master releases SCL and thus SCL goes high.
 - SCL remains LOW if any master pulls it LOW.
 - When all masters concerned have counted off their LOW period, the clock line is released and goes HIGH.
 - After going high, all masters start counting their HIGH periods. The first master to complete its HIGH period pulls the SCL line LOW again.



WIRED AND: *When one Master pulls SCL Low, no other Master can pull it high.*

A synchronized signal on SCL is obtained, where the slowest device determines the length of the low period and the fastest device determines the length of the high period.

Working Modes

➤ Master-sender

- Master issues START and ADDRESS, and then transmits data to the addressed slave device

➤ Master-receiver

- Master issues START and ADDRESS, and receives data from the addressed slave device

➤ Slave-sender

- Master issues START and the ADDRESS of the slave, and then the slave sends data to the master

➤ Slave-receiver

- Master issues START and the ADDRESS of the slave, and then the slave receives data from the master.

Is it better than SPI?

- SPI requires 4 lines
- SPI allows only one Master
- SPI allows high data rate (clock rate up to 10MHz in some devices) full duplex connections
- In SPI, the slave devices are not addressable (CS line used)
- More Information:
 - <https://www.i2c-bus.org/specification/>

Enable I2C in Raspberry Pi

- Similar to enabling SPI
- Use `sudo raspi-config`

I2C Timing

SINGLE-BYTE WRITE										
MASTER	START	SLAVE ADDRESS + WRITE		REGISTER ADDRESS		DATA		STOP		
SLAVE			ACK		ACK			ACK		

MULTIPLE-BYTE WRITE										
MASTER	START	SLAVE ADDRESS + WRITE		REGISTER ADDRESS		DATA		DATA		STOP
SLAVE			ACK		ACK			ACK		

SINGLE-BYTE READ										
MASTER	START	SLAVE ADDRESS + WRITE		REGISTER ADDRESS		START ¹	SLAVE ADDRESS + READ		NACK	STOP
SLAVE			ACK		ACK			ACK	DATA	

MULTIPLE-BYTE READ										
MASTER	START	SLAVE ADDRESS + WRITE		REGISTER ADDRESS		START ¹	SLAVE ADDRESS + READ		ACK	
SLAVE			ACK		ACK			ACK	DATA	DATA

Detect I2C Devices

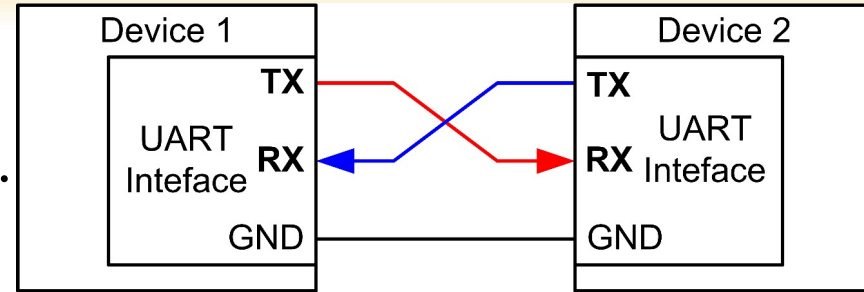
➤ `i2cdetect -y -r 1`

- Indicates one device with address 0x18

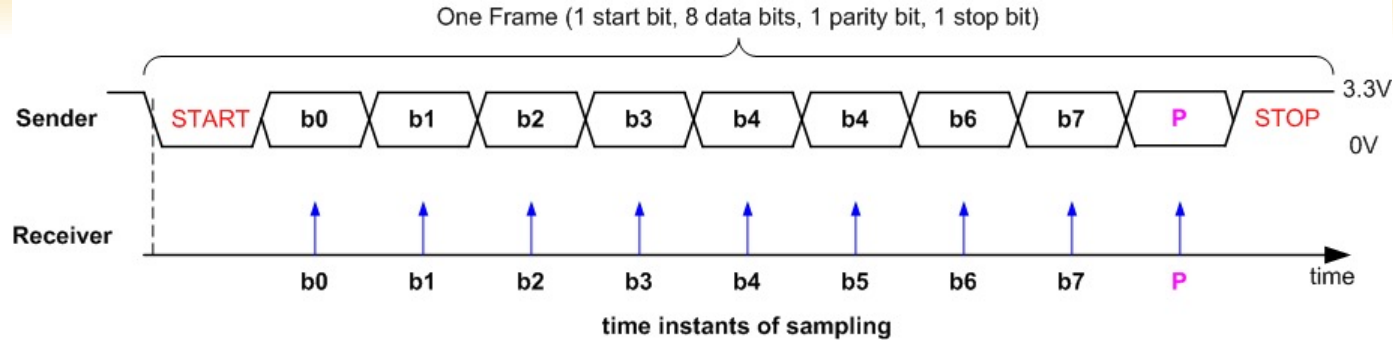
```
[dsaha@sahaPi:~] $ sudo i2cdetect -y -r 1
   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- 18 -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- --
```

Universal Asynchronous Receiver and Transmitter

- Universal
 - Programmable format, speed, etc.
- Asynchronous
 - Sender provides no clock signal to receivers
- Half Duplex
- Any node can initiate communication
- Two lanes are independent of each other



Data Frame



- Sender and receiver uses the same transmission speed (10% clock shift/difference is tolerated)
- Data frame
 - One start bit
 - Data (LSB first or MSB, and size of 7, 8, 9 bits)
 - Optional parity bit
 - One or two stop bit

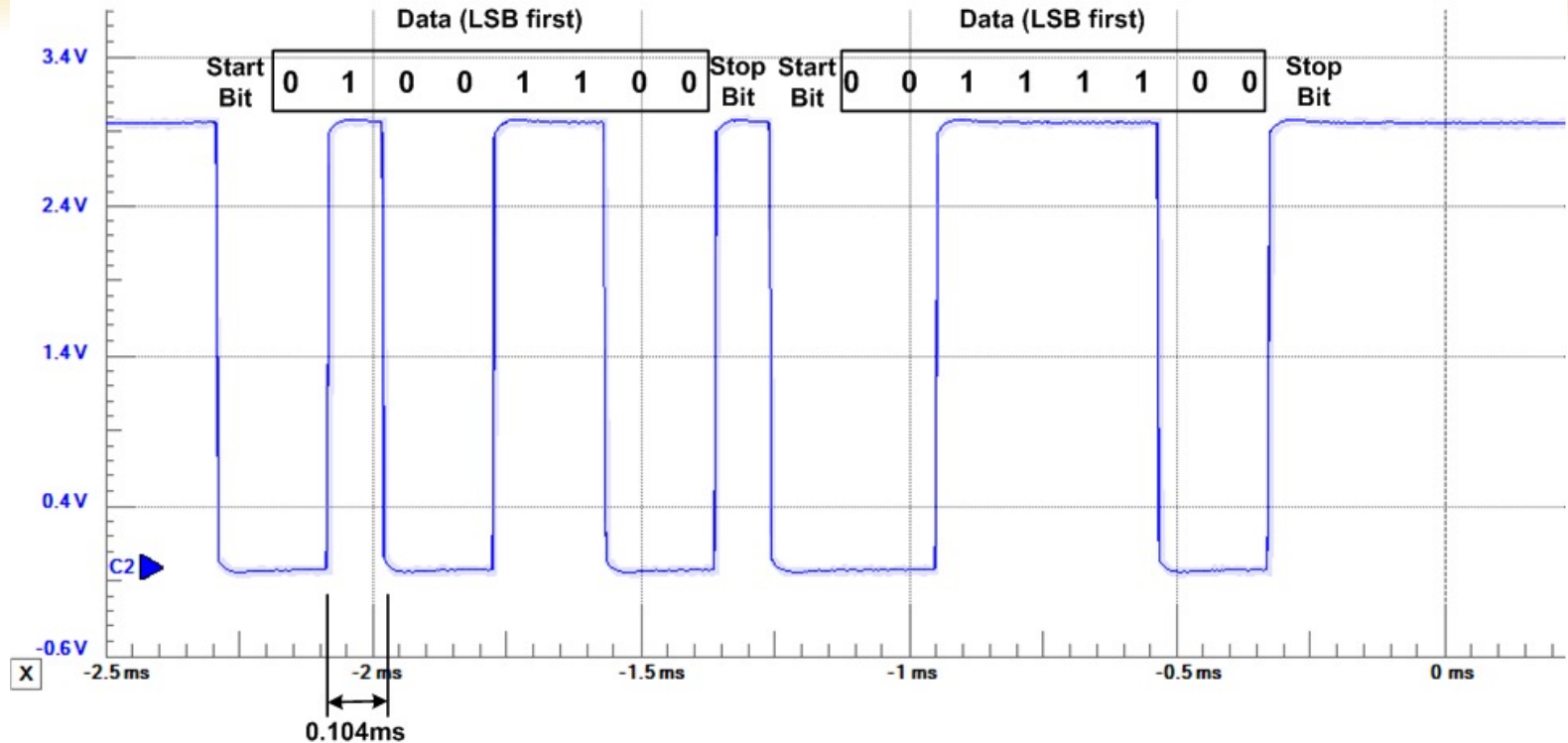
Baud Rate

- Historically used in telecommunication to represent the number of pulses physically transferred per second
- In digital communication, baud rate is the number of bits physically transferred per second
- Example:
 - Baud rate is 9600
 - each frame: a start bit, 8 data bits, a stop bit, and no parity bit.
 - Transmission rate of actual data
 - ~~$9600/8 = 1200$ bytes/second~~
 - $9600/(1 + 8 + 1) = 960$ bytes/second
 - The start and stop bits are the protocol overhead

Error Detection

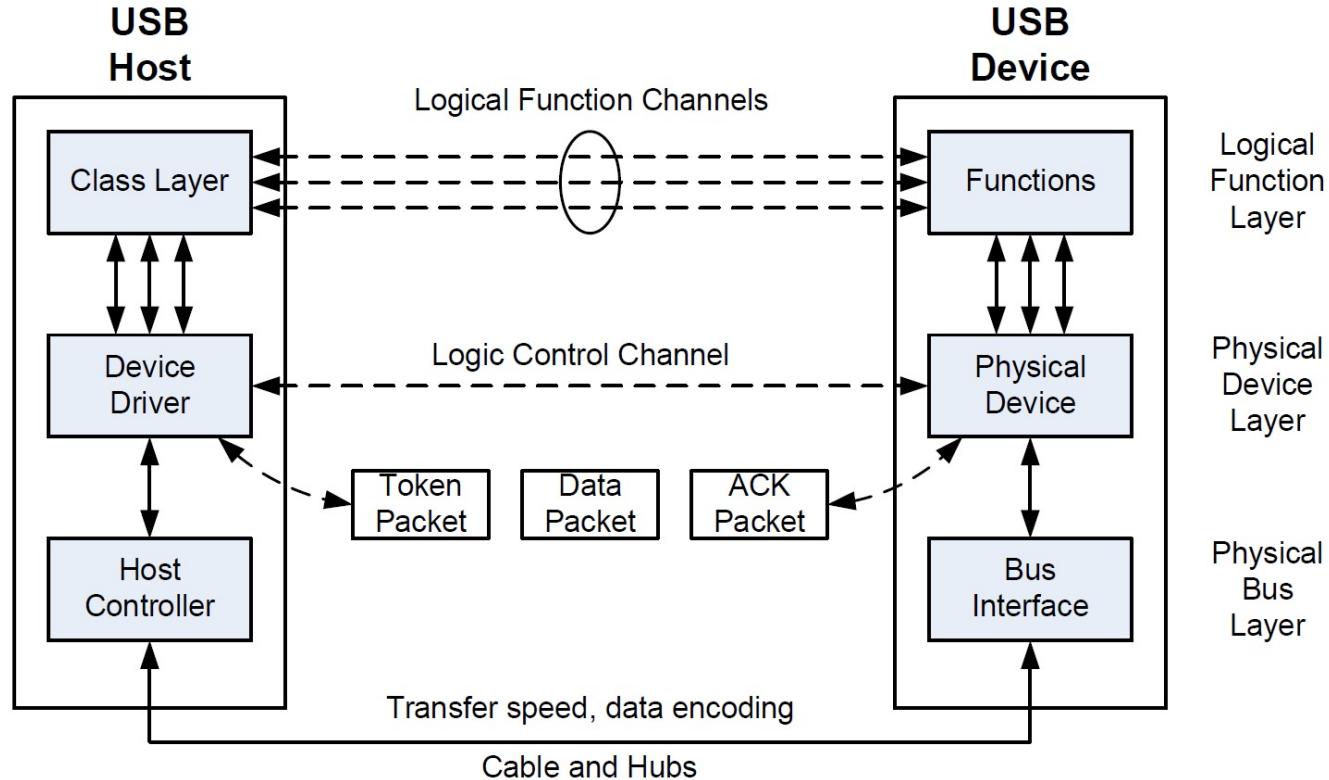
- **Even Parity:** total number of “1” bits in data and parity is even
- **Odd Parity:** total number of “1” bits in data and parity is odd
- Example: Data = 10101011 (five “1” bits)
 - The parity bit should be 0 for odd parity and 1 for even parity
- This can detect single-bit data corruption

Transmitting 0x32 and 0x3C

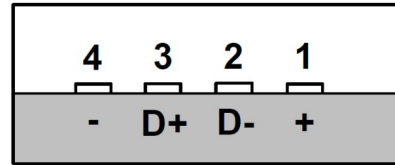


1 start bit, 1 stop bit, 8 data bits, no parity, baud rate = 9600

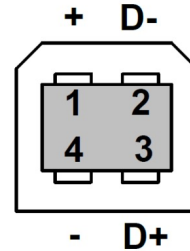
USB Layers



USB Connection



Standard A



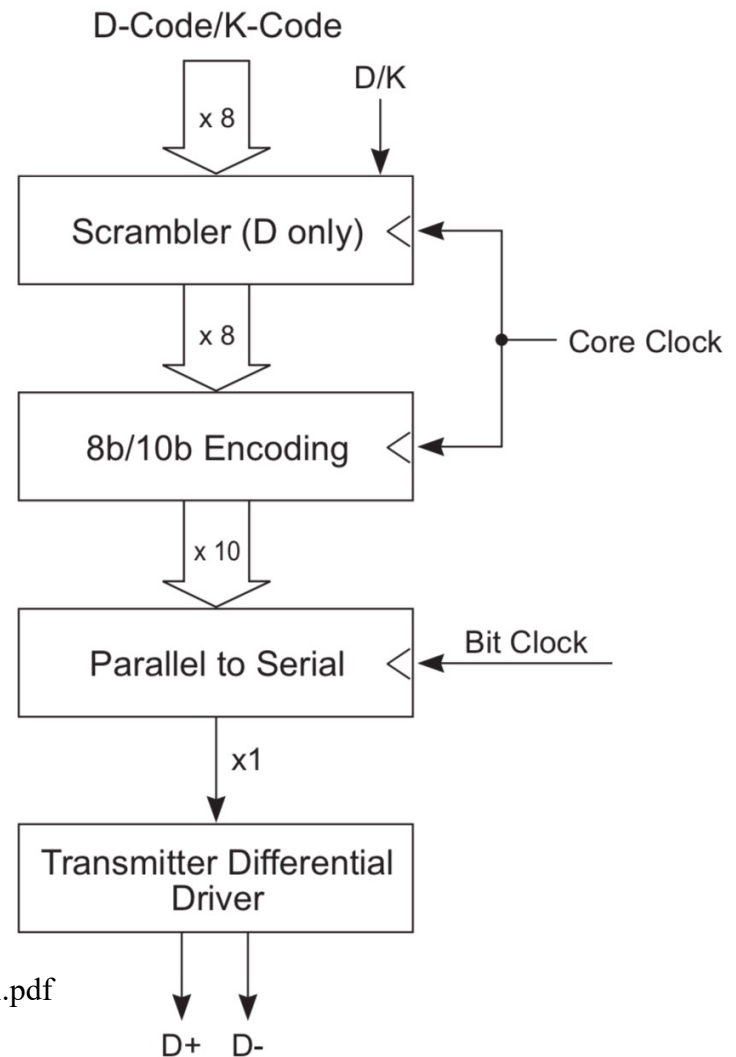
Standard B

- Four shielded wires: two for power (+5V, ground), two for data (D+, D-)
- D+ and D- are twisted to cancel external electromagnetic interference



USB Physical Layer

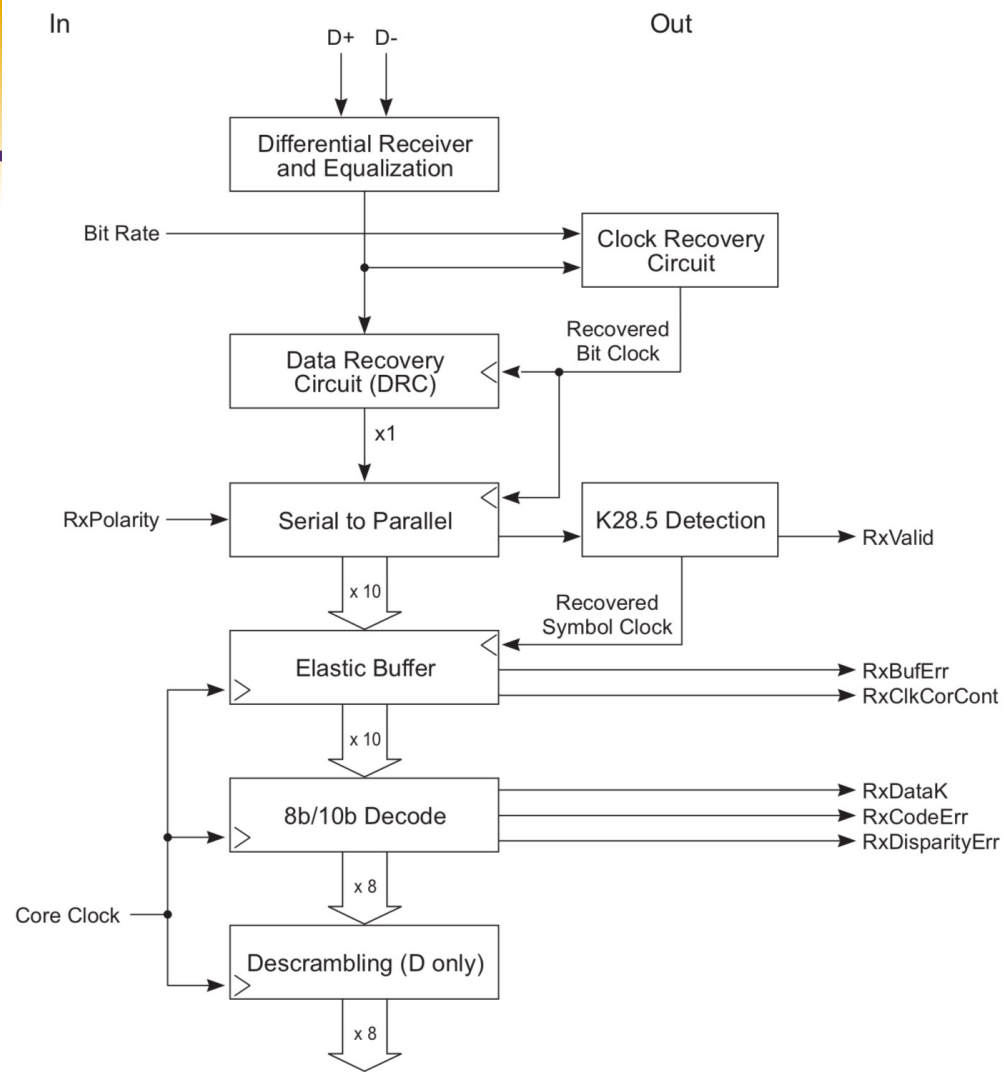
- Transmitter Block Diagram
- Separate CRCs for control and data fields of each packet



[https://www.usb3.com/whitepapers/USB%203%200%20\(11132008\)-final.pdf](https://www.usb3.com/whitepapers/USB%203%200%20(11132008)-final.pdf)

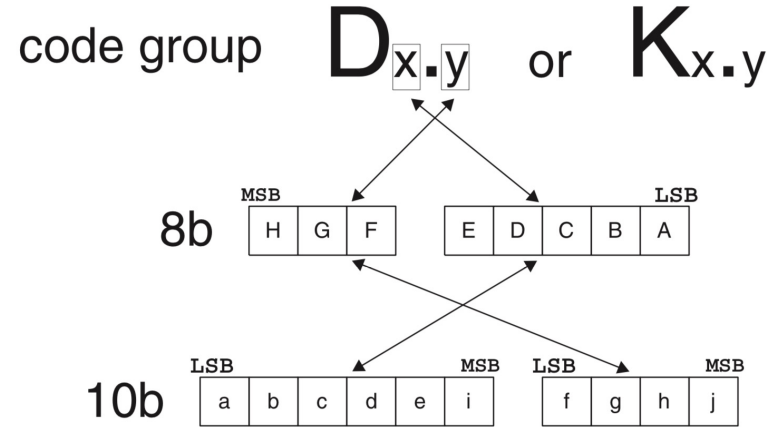
USB PHY

➤ Receiver Block Diagram



8b/10b Encoding

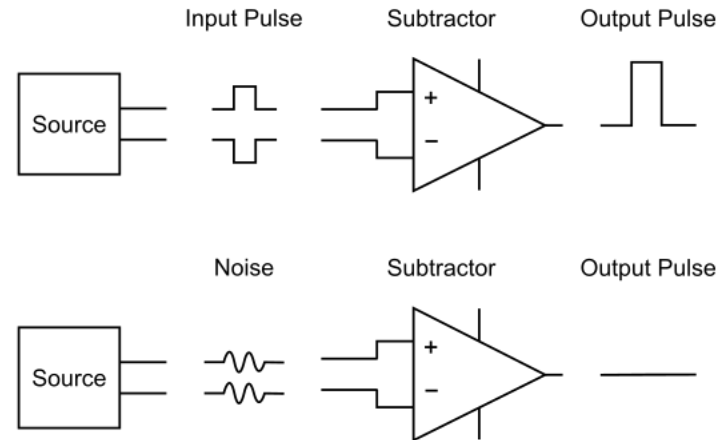
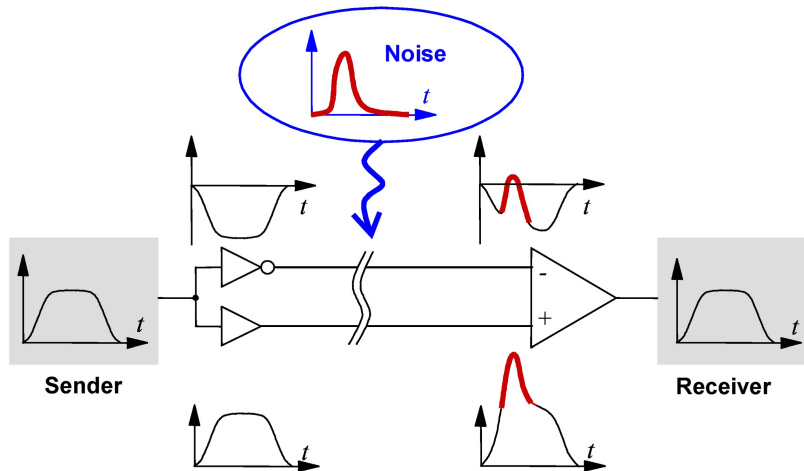
- ensure sufficient data transitions for clock recovery
- A **DC-balanced** serial data stream
 - it has almost same number of 0s and 1s for a given length of data stream.
 - DC-balance is important for certain media as it avoids a charge being built up in the media.



3b Binary (HGF)	4b Binary (fghi)
000	0100 or 1011
001	1001
010	0101
011	0011 or 1100
100	0010 or 1101
101	1010
110	0110
111	0001 or 1110 or 1000 or 0111

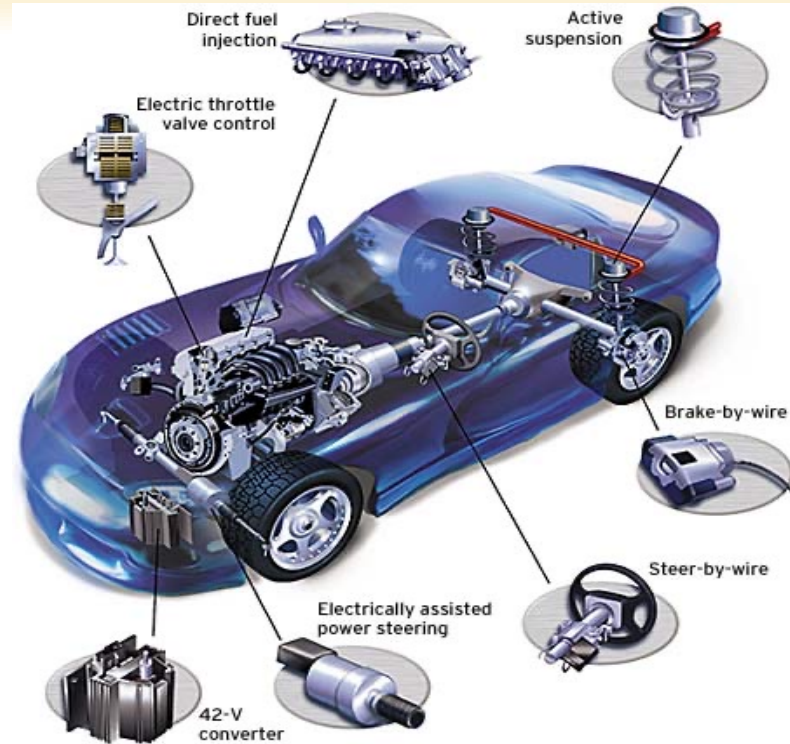
Simple Differential Signaling

- Information is transmitted using two complementary signals
- Improves reducing noise

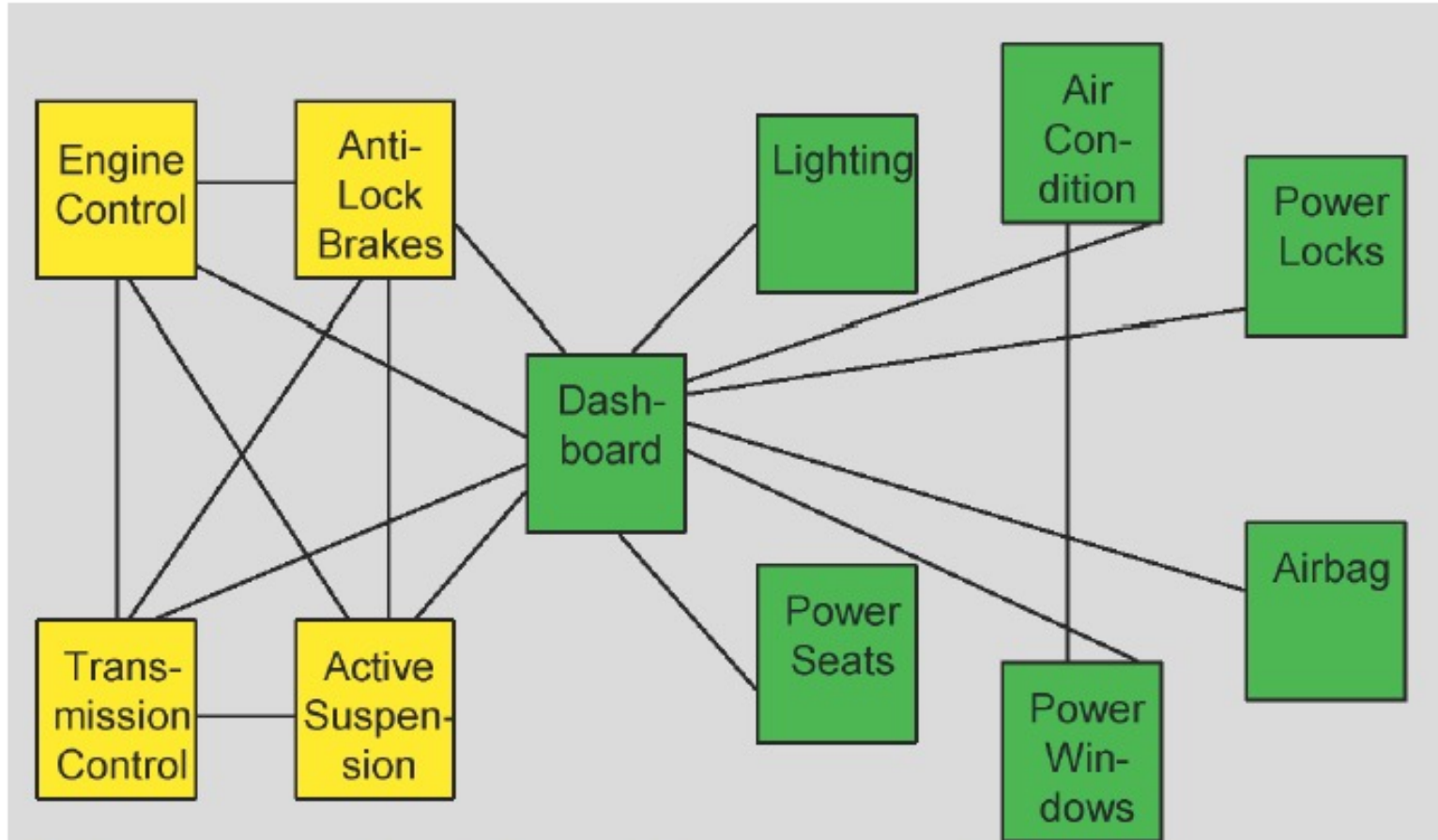


Controller Area Network (CAN) Bus

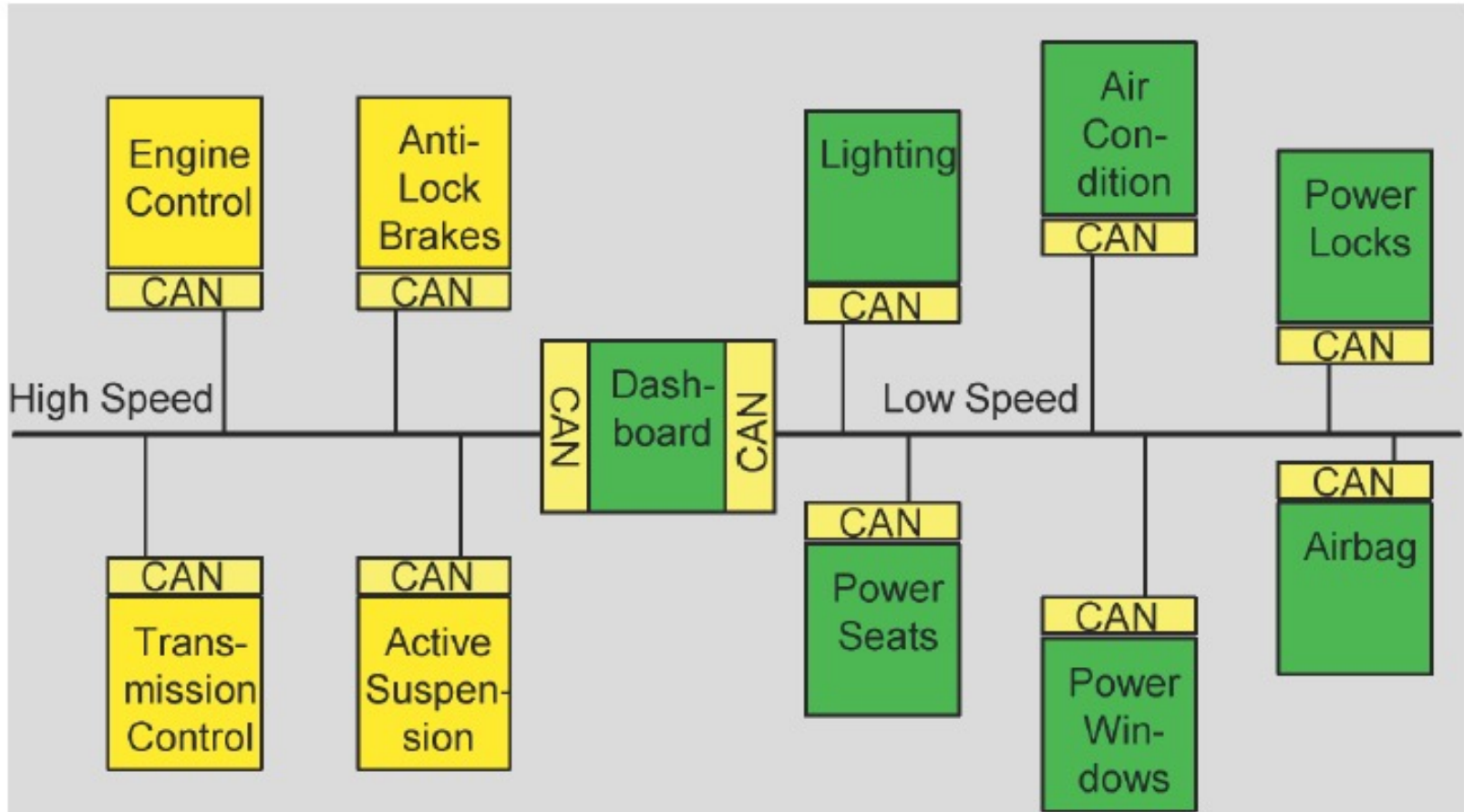
- Serial communication
- Multi-Master Protocol
- Compact
 - Twisted Pair Bus line
- 1 Megabit per second



Before CAN



After CAN



Layered Approach (CAN)

