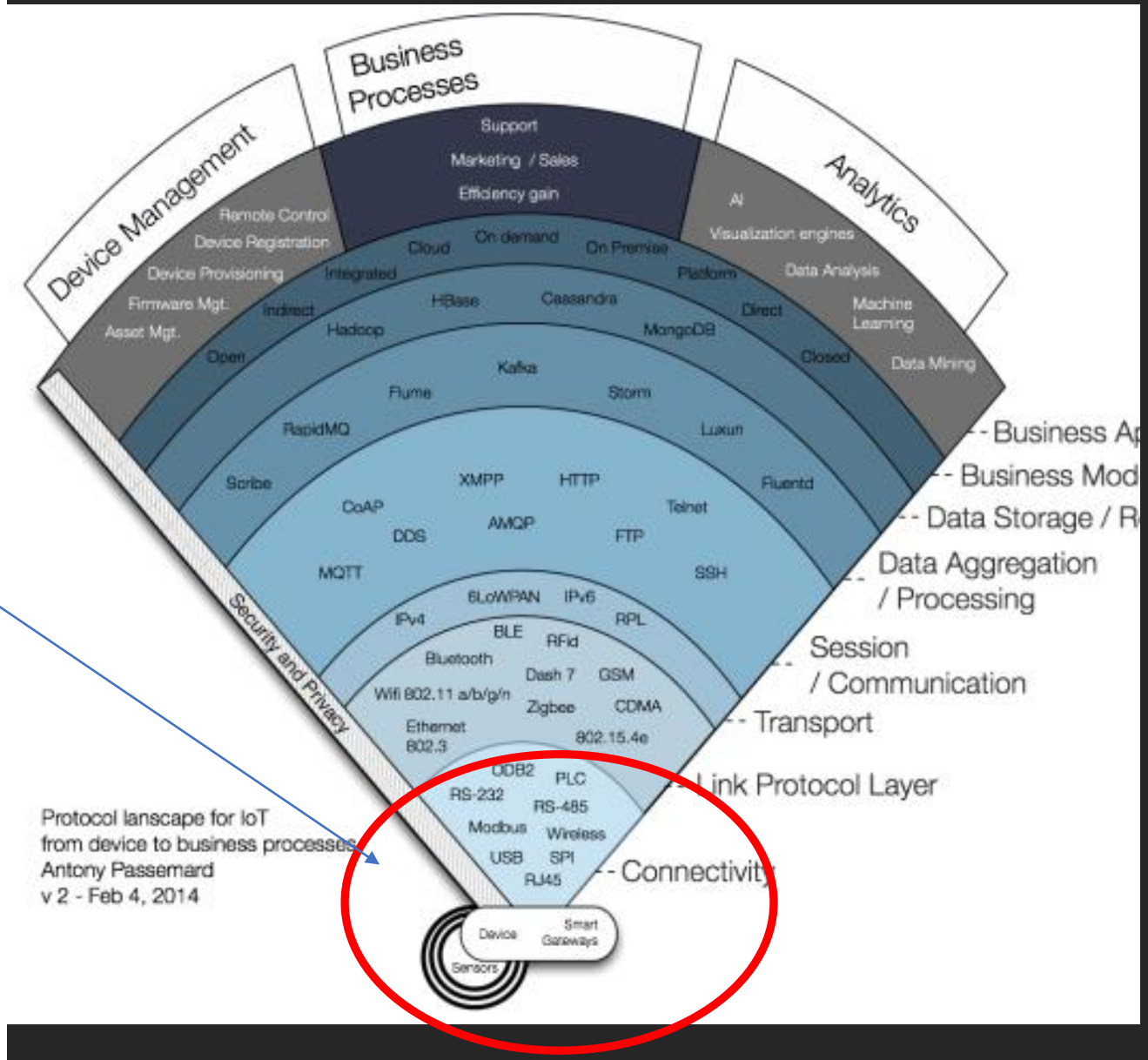
Overview of Serial Protocols

Dr. Frank Walsh

IoT Protocols

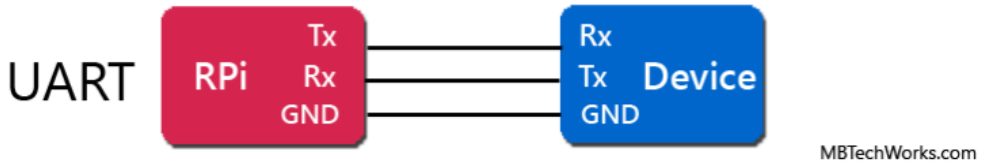
WIT

We're focussing here…



Communication Protocols for Circuits: Introduction

- Embedded electronics requires interlinking circuits (processors or other integrated circuits) to swap their information.
- Required to have a common communication protocol
- Lots of communication protocols developed
- Generally classified into two categories:
  - Serial
  - Parallel
- Serial comms often referred to as serial bus



Parallel & Serial

- A communication system that transfers data between components inside a computer or between computers
- Generally **2** types
  - Parallel
  - Serial
- Parallel is simplest to implement but takes up a lot of hardware ‘real estate’
- Serial requires fewer lines to transmit data but this adds complexity.
  - Synchronous – uses clock
  - Asynchronous – no clock but speed (baud rate) agreed before transmission.
- We’re going to look at serial communication protocols.

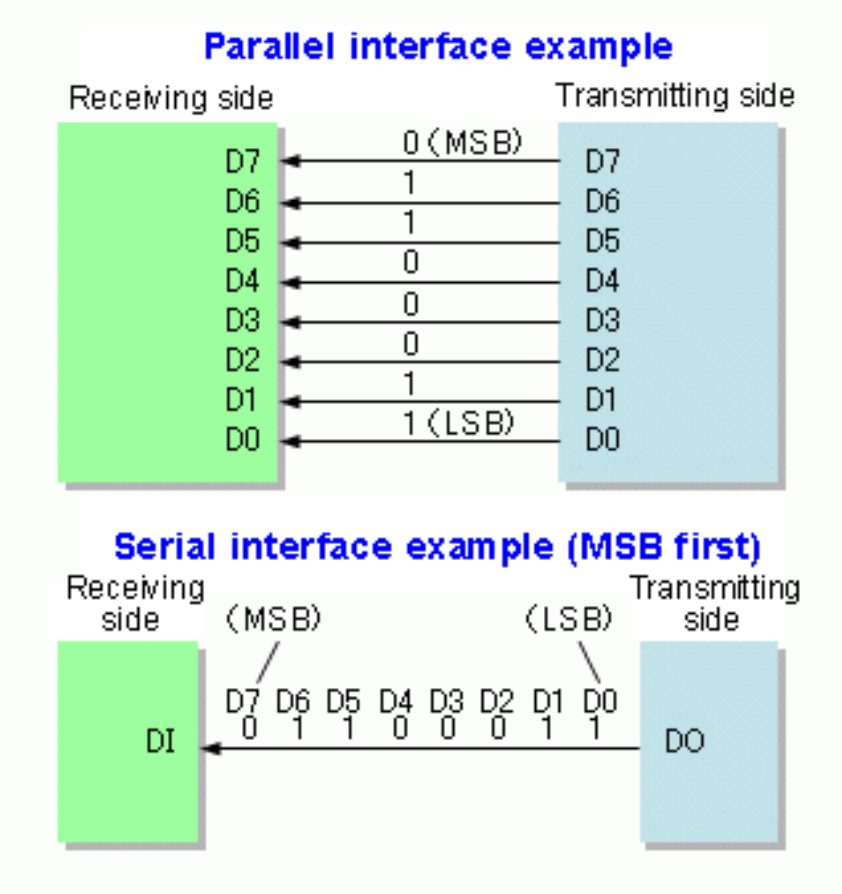
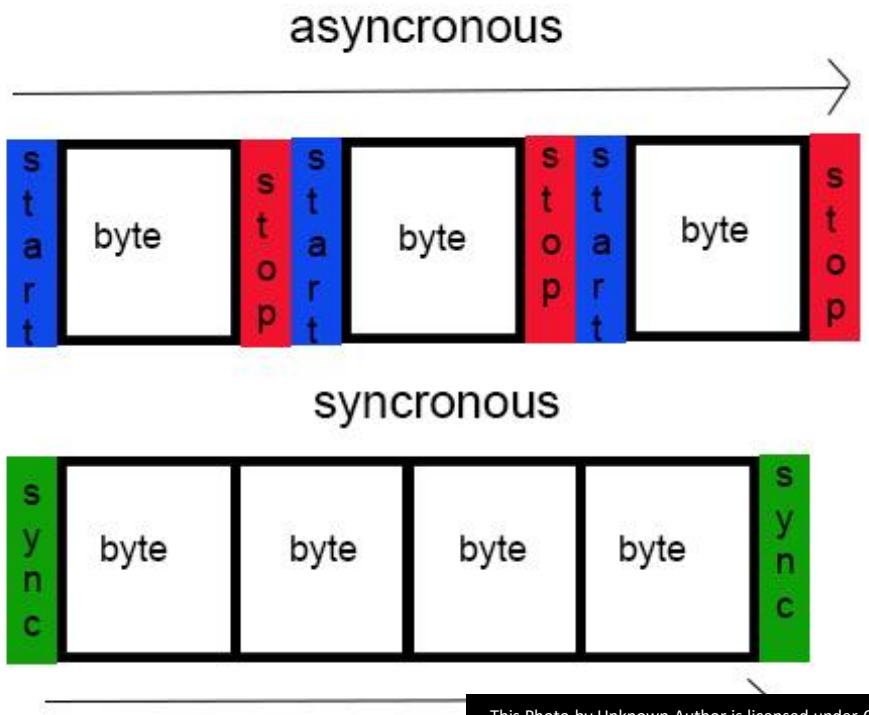


image: https://en.wikipedia.org/wiki/Parallel_communication#/media/File:Parallel_and_Serial_Transmission.gif

Synchronous and Asynchronous Serial

- Known serial interface in embedded systems
  - Universal Serial Bus (USB)
  - Ethernet
  - UART
  - SPI
  - I^2C
- Asynchronous Serial
  - Data is transferred without support from an external clock signal
  - Ideal for minimizing the required wires and input-output pins.
  - Examples: UART
- Synchronous Serial
  - Pairs its data line with a clock signal, and all devices on serial bus share a common clock
  - Faster serial transfer, but requires at least one extra wire between communicating devices
  - Example: SPI, I^2C

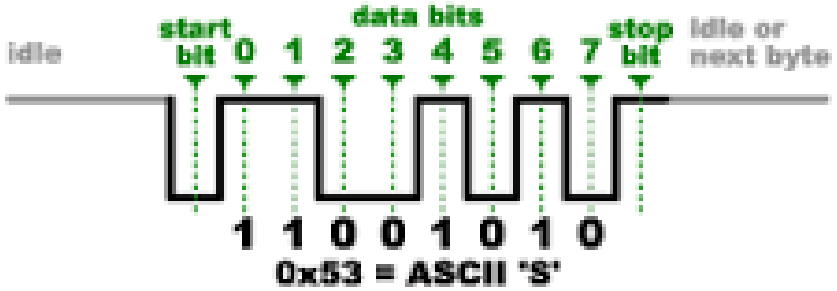


This Photo by Unknown Author is licensed under CC BY-SA

Rules of Asynchronous Serial, aka Serial

- Number of built-in rules help ensure robust and error free data transfer
  - Baud rate
  - Data bits
  - Synchronization bits
  - Parity bits

- Baud rate
  - Specifies how fast data is sent over a serial line.
  - Usually expressed in units of bits-per-second (bps)
  - Determines how long the transmitter holds a serial line high/low and period the receiving device samples its line
  - Can be of any value, but both transmitter and receiver should have the same baud rate
  - Standard baud rates are 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200



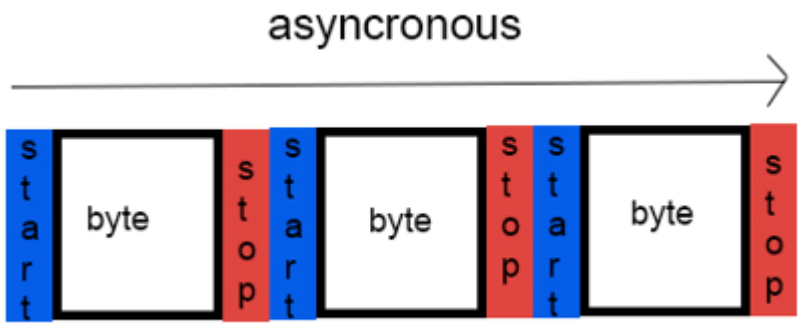
This Photo by Unknown Author is licensed under CC BY-SA

Data Packets in Serial transfer

- Each block of data transmitted is actually sent in a frame of bits.

Source: Sparkfun

- Data bits
  - Amount of data in each frame can be from 5 to 9 bits
  - It is important to agree on **endianness** (is data sent Most Significant Bit to Last Significant Bit or vice-versa?)
  - If order not stated, default is LSB sent first



- Synchronization bits
  - Two or three special bits are transferred along with data bits
  - **Start bit** marks the beginning of a data packet. Usually 1 bit
    - Indicated by an idle data line going from 1 to 0
  - **Stop bit(s)** marks the end of a data packet. Configurable to 1 or 2 bit, usually 1 bit.
    - Indicated by stop bit(s) transition back to the idle state by holding the line at 1.

Data Packets in Serial transfer

- Parity bits *(Optional)*
  - A simple and low-level error checking method
  - Options: Odd or Even parity
  - Odd parity bit = 1 if the number of 1's in data is odd
  - Even parity bit = 1 if the number of 1's in data is even

Source: Sparkfun

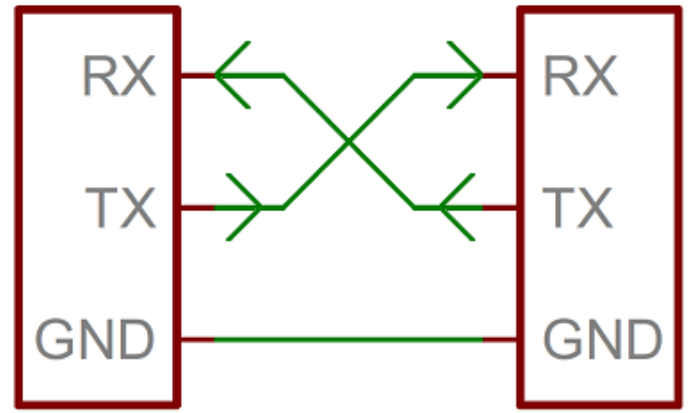
- Example: 9600 8N1 OK
  - Baud rate = 9600
  - Data bits = 8
  - Parity bits = No
  - Stop bits = 1
  - Data = OK

Source: Sparkfun

  - O – ASCII Value = 79 = 01001111
  - K – ASCII Value = 75 = 01001011

Serial – Wiring and Hardware

- Serial bus consists of two wires
  - TX – sends data
  - RX – receives data



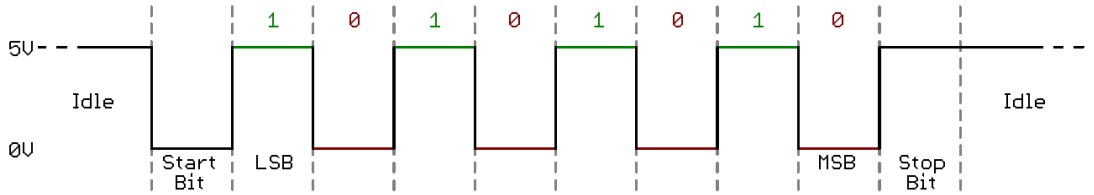
- Interface can operate as **full-duplex** or **half-duplex**
  - Full duplex – both devices can send and receive simultaneously
  - Half duplex – devices must take turns sending and receiving

- Some devices need only one way communication(simplex).
  - Accordingly, only wire is visible

Source: Sparkfun

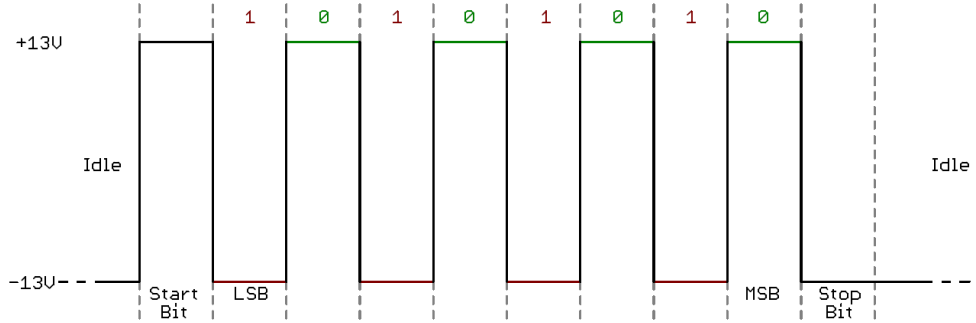
Serial – Hardware Implementation

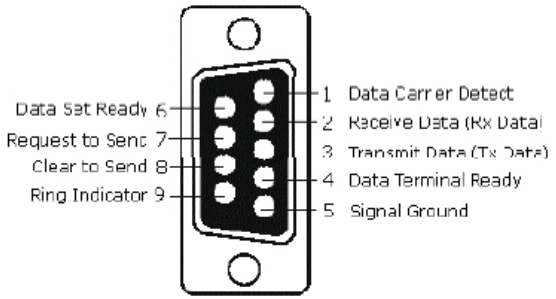
- Transistor-Transistor Logic (TTL)
  - TTL serial signals exist between a microcontrollers' voltage supply range (usually 0V to 3.3V or 5V)
  - A signal at the VCC level indicates either an idle line, a bit of value 1, or a stop bit.
  - A 0V (GND) signal represents either a start bit or a data bit of value 0.
  - Easier to implement in embedded systems, but susceptible to losses across long transmission lines.



Source: Sparkfun

- RS-232
  - Usually found in old computers and legacy peripherals
  - Usually range between -5V to +5V, though specs allow for anything from +/- 3V to +/- 25V.
  - A low voltage indicates either the idle line, a stop bit, or a data bit of value 1.
  - A high voltage indicates either a start bit, or a 0-value data bit.
  - RS-485 is better suited to long range serial transmissions.



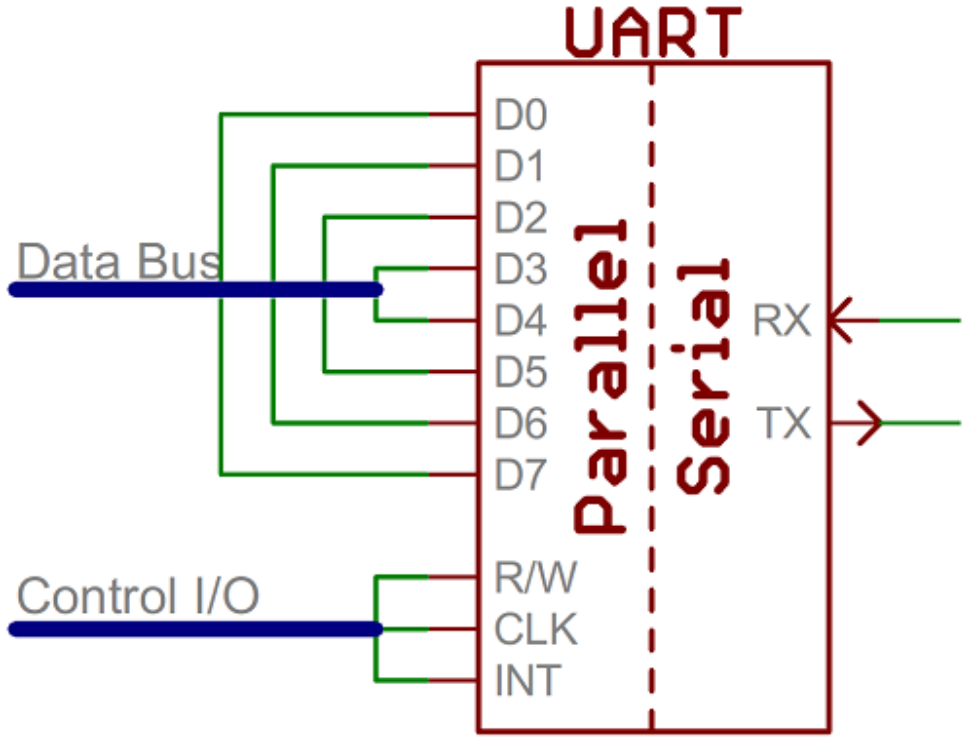


Asynchronous Serial (RS232)

- Used for 1-to-1 communication
- Many variants, simplest just uses **2** lines.
- Often used for console access to configure Network Routers

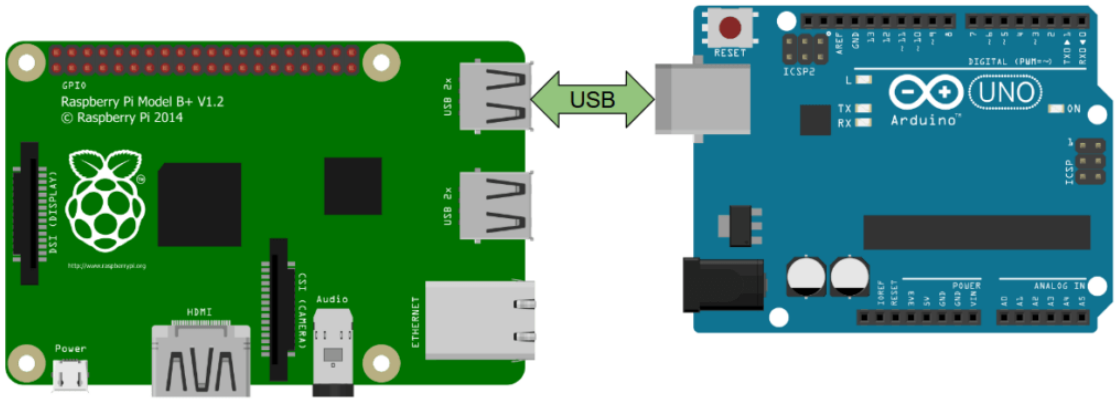
Universal Asynchronous Receiver/Transmitter - UART

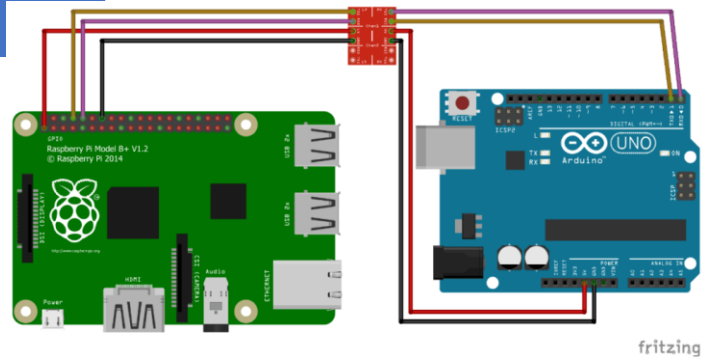
- UART is a block of circuitry responsible for implementing serial communication.
- UART acts as an intermediary between parallel and serial interfaces.
- Available as standalone ICs, but generally integrated inside a microcontroller.
- On the transmit side, a UART must create the data packet and send it out with precise timing.
- On the receive side, a UART has to sample RX lines, and extract the data from the packet received

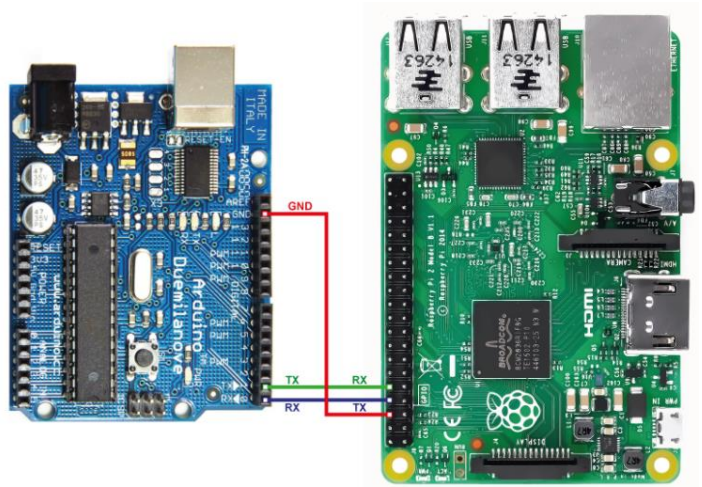


Source: Sparkfun

Raspberry Pi Serial Communication Examples

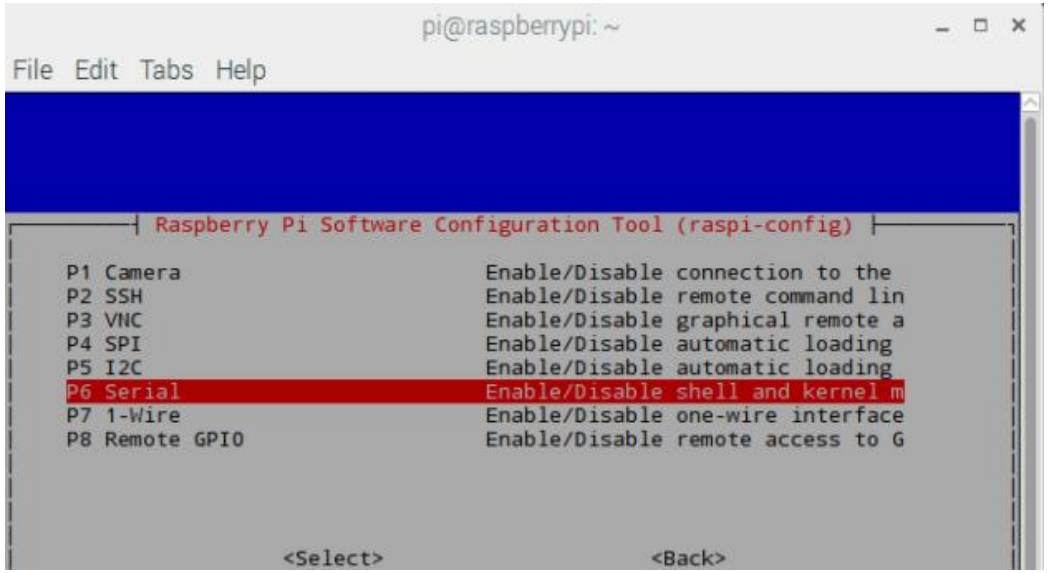






Demo: PySerial

- Serial Communication uses a hardware interface “port” in programming
  - May need some configuration in OS
- Libraries to access, configure and use ports for serial comms
  - handle all the low layers for you!

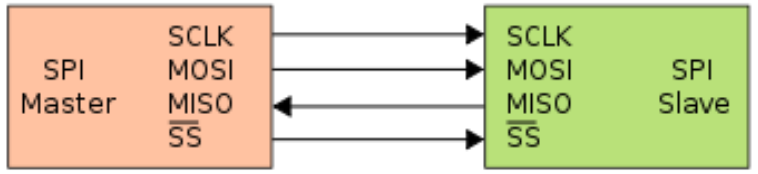


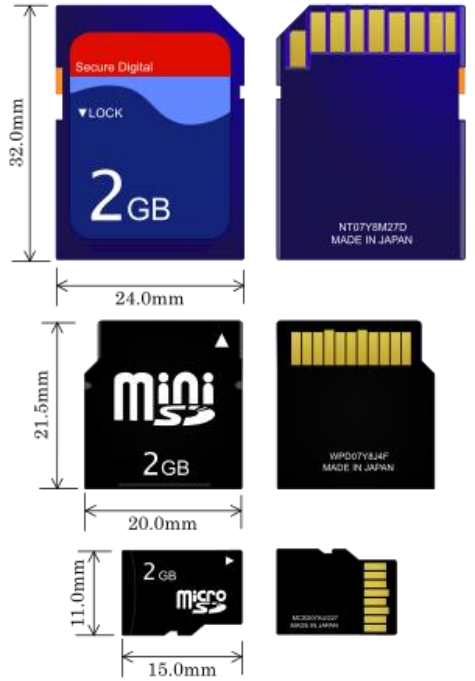

```
import serial
from time import sleep

ser = serial.Serial ("/dev/ttyS0", 9600)    #Open port with baud rate
while True:
    received_data = ser.read()              #read serial port
    sleep(0.03)
    data_left = ser.inWaiting()             #check for remaining byte
    received_data += ser.read(data_left)
    print (received_data)                   #print received data
    ser.write(received_data)                #transmit data serially
```

Synchronous Communication

Serial Peripheral Interface (SPI)

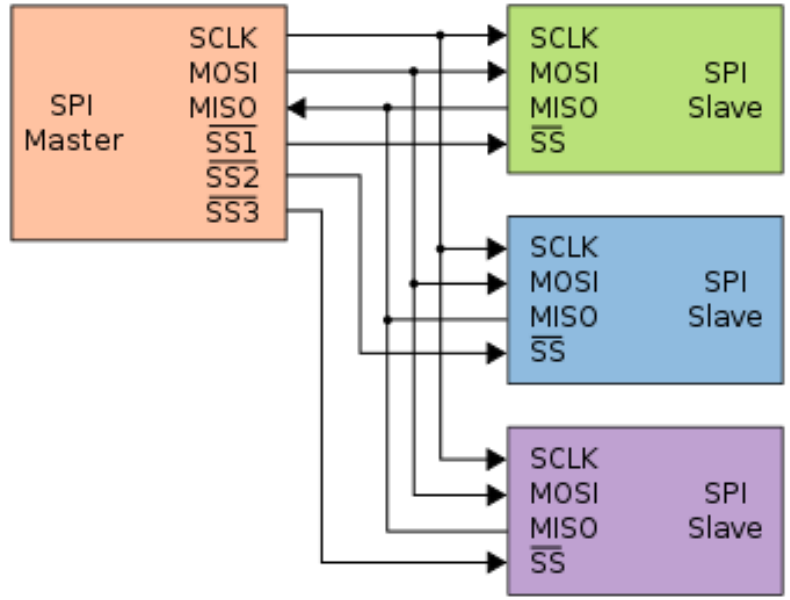
- Master and Slave Devices
  - One master and multiple slaves
- Used in liquid crystal displays, SD Cards, MPC3006 ADC
- Master sets the speed
- Signals
  - SCLK: Serial Clock (output from master).
  - MOSI: Master Output Slave Input, or Master Out Slave In (data output from master).
  - MISO: Master Input Slave Output, or Master In Slave Out (data output from slave).
  - SS: Slave Select (pulling line low selects slave, output from master).





Serial Peripheral Interface (SPI)

- Slave select line goes low to select slave
- Full duplex data transmission occurs.
  - The master sends a bit on the MOSI line and the slave reads it
  - The slave sends a bit on the MISO line and the master reads it.
- MCP3008 Analog to Digital converter uses it
- https://www.microchip.com/wwwproducts/en/MCP3008



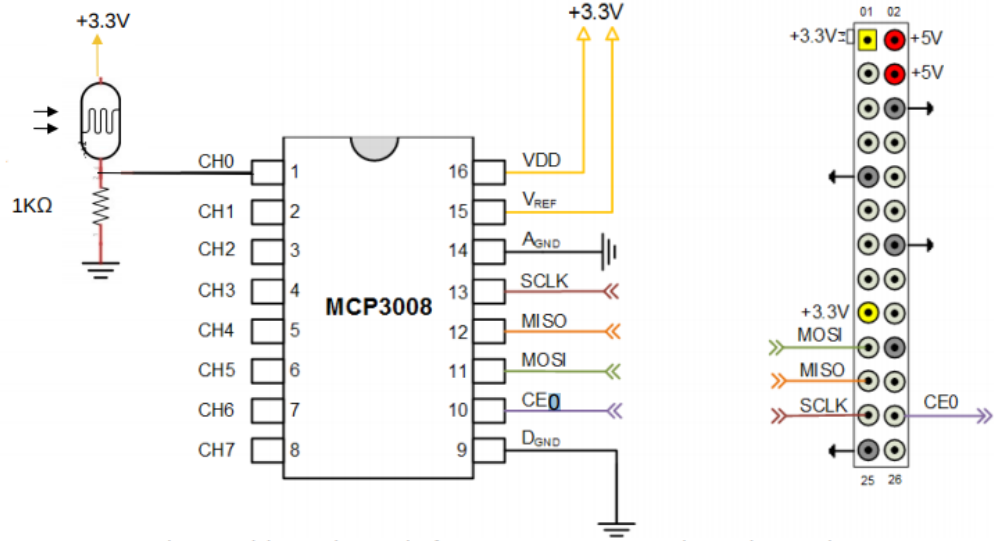
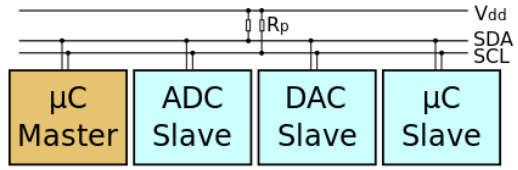


Fig. 5. Wiring Schematic for LDR, MCP3008, and Raspberry Pi

I²C (Inter-Integrated Circuit)

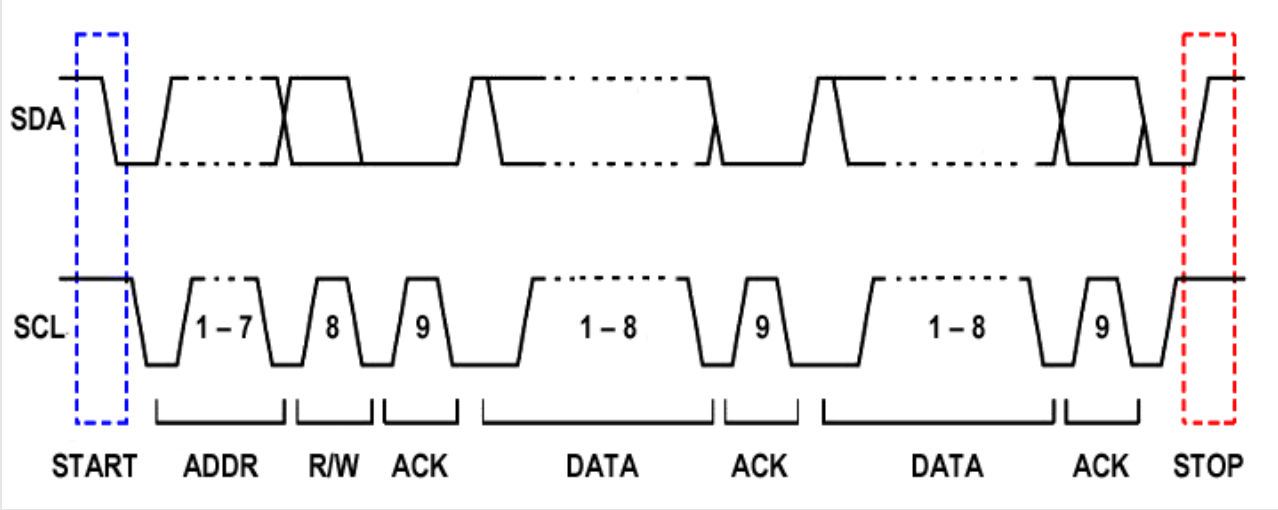
- Also referred to as 2-wire bus.
  - Clock(SCL) and Data(SDA)
- Used for connecting lower-speed devices to processors and microcontrollers
- Master-slave approach.
- Unlike SPI, uses addressing instead of physical Slave Select lines (hence only **2** wires).
- Speeds: 100kbs, 400kbs, 1Mbs and 3.4Mbs



I²C (Inter-Integrated Circuit)

1. SDA,SLC start high
2. Master: SDA to low to signal start
3. Master: Send SCL with 7 bit address followed by 0 (for write)
4. Slave: pulls SDA to low for Acknowledgement
5. Master: sends **8** bit data on SDA
6. Slave: Acknowledgement
7. All: allow SDA, when SCL is high to Stop

https://www.youtube.com/watch?v=qeJN_80CiMU

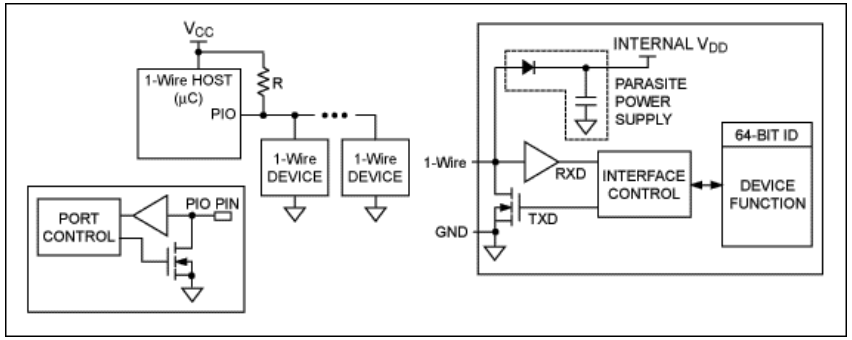


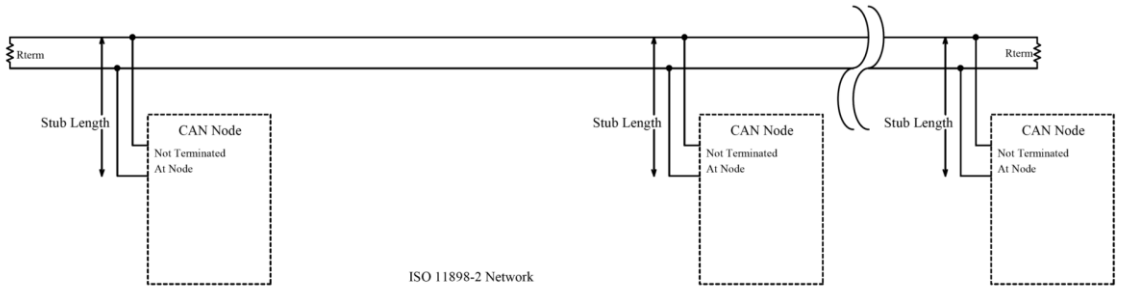
I²C (Inter-Integrated Circuit)

- You can transfer multiple bytes consecutively

1-Wire

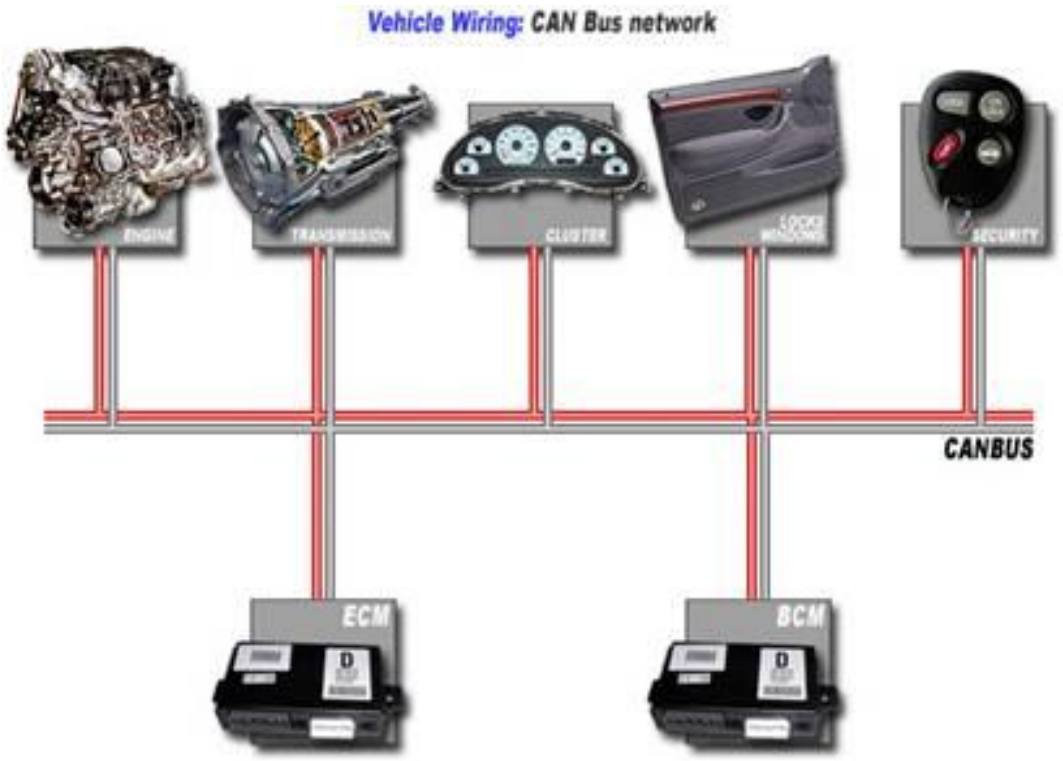
- provides low-speed data, signaling, and power over a single wire.
- Similar to I^2C, but lower data rates and longer range.
- Despite the name, you need **2** wires:
  - Data and ground.
- Because there's no power(Vcc line), 1-Wire devices use capacitors to store power
  - Parasitic device – takes power from bus.
  - power the device when the data line is active





CAN (Controller Area Network)

- A Controller Area Network (**CAN bus**) allows microcontrollers and devices to communicate with each other.
- Predominantly used in Automotive
  - Also in aviation/industrial
- No Master-Slave, it’s mutli-master
  - Any node can initiate comms.
- All nodes are connected to each other through a two wire bus



CAN (Controller Area Network)

- CAN reduces wiring requirements
- Robust protocol with built in fault tolerance
- Reliable
  - That's why it's the defacto protocol in automotive
- Relatively straight forward protocol to understand….

Universal Serial Bus (USB)

- USB resulted from mixture of connection methods used on PCs
  - Serial ports (modems)
  - Parallel ports (printers)
  - PCI (keyboards and Mice)
- Now it’s “defacto”
- Low speed
  - Mice, keyboards
- Full speed
  - Other devices
- High speed
  - USB 2.0, media devices
- USB **3.0…**
- Used for all sorts of stuff!!!

USB 1.0	January 1996	Low Speed (1.5 Mbit/s)
USB 1.1	August 1998	Full Speed (12 Mbit/s)[24]
USB 2.0	April 2000	High Speed (480 Mbit/s)
USB 3.0	November 2008	SuperSpeed (5 Gbit/s)
USB 3.1	July 2013	SuperSpeed+ (10 Gbit/s)
USB 3.2	?-September 2017	SuperSpeed++ (20 Gbit/s)

Standard	Tx Type	# Signal Wires	Data Rate & Distance	Hardware $	Scalability	Application Example
UART	Asynchronous	2	20kbps @ 15m	Medium (transceiver)	Low (point-to-point)	Diagnostic display
LIN	Asynchronous	2	20kbps @ 40m	Medium (transceiver)	High (identifier)	Washing machine subsystem network
SPI	Synchronous	4+	25Mbps @ 0.1m	Low	Medium (chip selects)	High speed chip to chip link
I2C	Synchronous	2	1Mbps @ 0.5m	Low (resistors)	High (identifier)	System sensor network