Sensors, Signals, Serial ports, and Sockets

Or, How to Talk to your Robot Without Going Crazy

Topics

- Basic electricity (Ohm's Law)
- Signals from sensors: key terms
  - UART
  - Serial port
  - TTL
  - RS232
  - USB
  - I^2C
  - SPI, PWM
- Popular sensors
- pySerial API

Topics

- Message types:
  - Raw / binary / hex
  - Custom text
  - NMEA
- Sockets: client / server

Configuring an ad-hoc wireless network

- DHCP
- ifconfig / iwconfig

Understanding Signals: Ohm's Law

-
  - Voltage
  - Constriction: Resistance
  - Flow: Current
  - Flows toward the Ground
- I = V / R : Current =

  Voltage / Resistance
- Voltage: Volts; Current: Amp(ere)s;

  Resistance: Ohms (Ω)

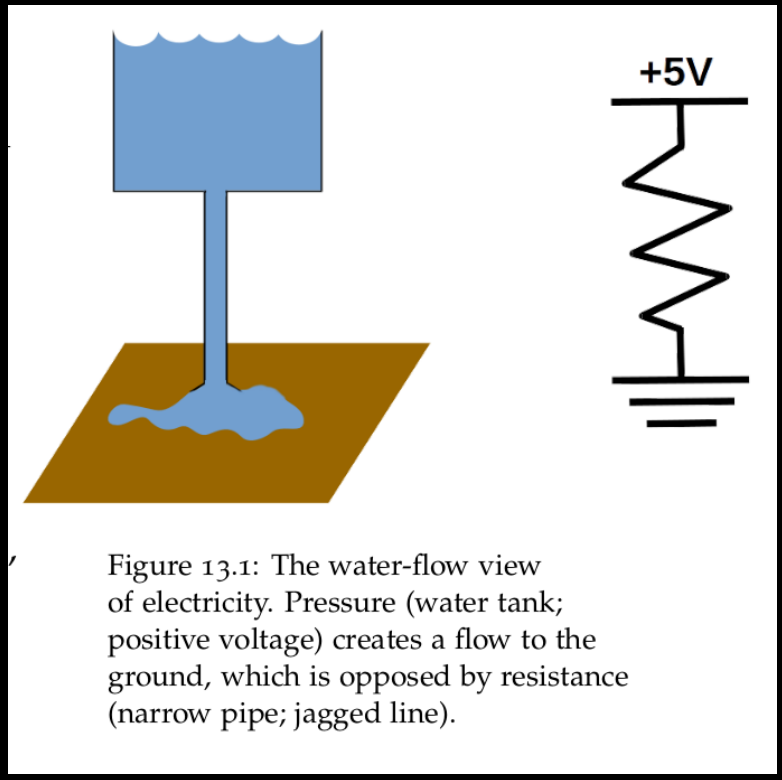


Figure 13.1: The water-flow view of electricity. Pressure (water tank; positive voltage) creates a flow to the ground, which is opposed by resistance (narrow pipe; jagged line).

Ohm's Law Example

Question: A 120 Volt lamp is powering a lightbulb that has a resistance of 240 Ohms. How much current is passing through the lightbulb?

Answer: $I = V / R$

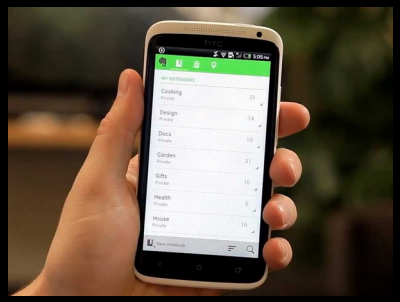
$= 120\ V / 240\ \Omega$

$= 0.5\ A$

Ohm's Law

- Very high voltage, very low current

  (10000 V * .0001 A = 1 Watt)



- Low voltage, low current

  (3.7V * 1A = 3.7 Watts)

D1YYBK Alamy Images

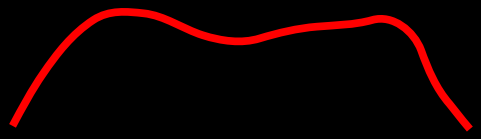
- High voltage, high current

(1000V * 1000A =

1000000 Watts!!!)

Analog vs. Digital

Analog	Digital
Continuous	Discrete
Measuring	Counting
Ruler	Abacus
Real numbers	Integers
Smooth	Step-like
“Natural”	Arbitrary



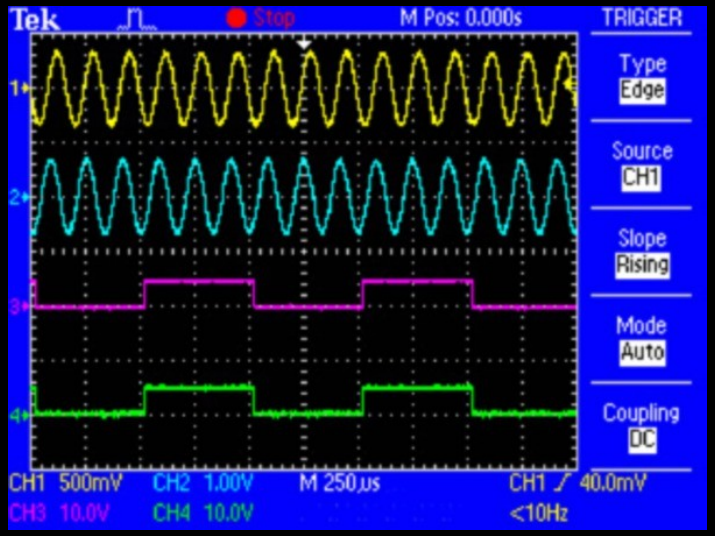
House Mountain

0,2,4,3,3,3,4,1,0

House Mountain

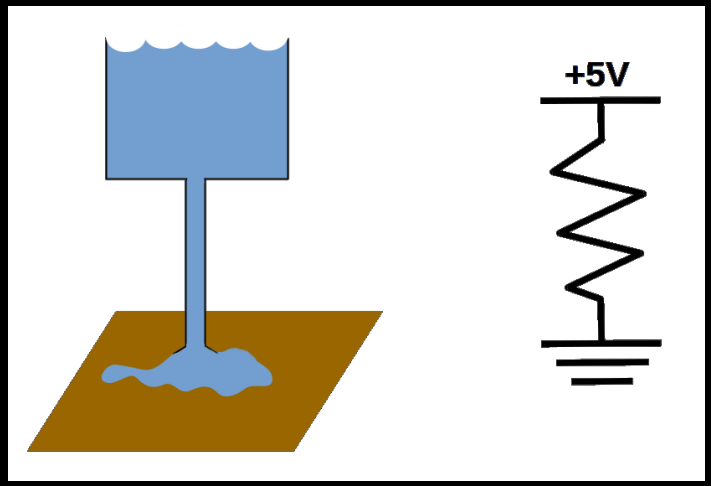
Analog vs. Digital Circuits

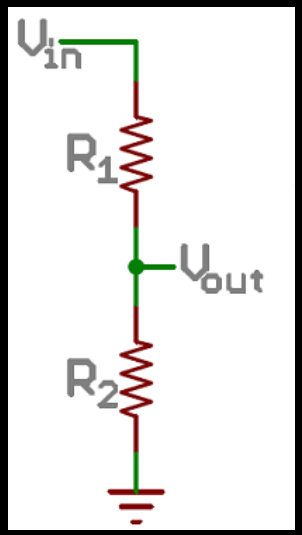
- Electricity is inherently analog: we *measure* voltage, current, resistance; we don't "count" it.
- Electronic components can be assembled to make *circuits*.
- When engineers say "digital circuit", they mean "electronic circuit that approximates discrete voltages"
- Both digital and analog circuits involve three major components
- Let's look at each in turn ….



Resistor: Reduce current or voltage

- Recall Ohm's Law: I = V / R
- So more resistance means less current
- We can also use two resistors to build a *voltage divider*:

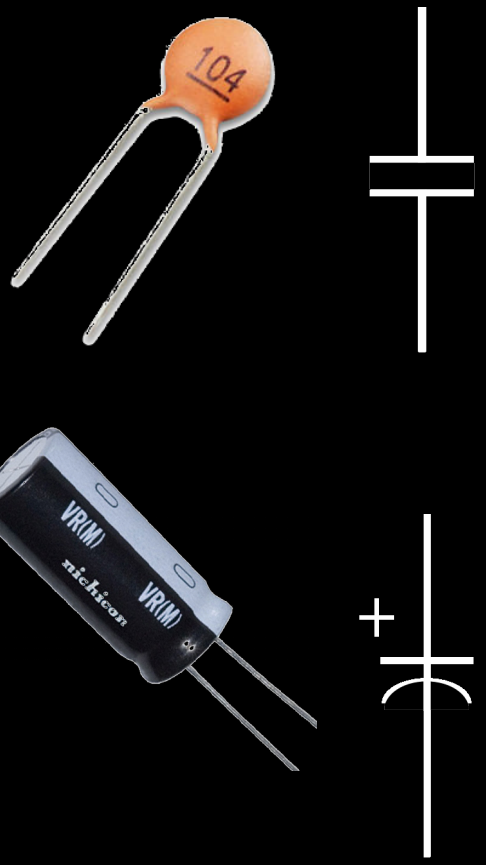




$$V_{out} = V_{in} \frac{R_2}{R_1 + R_2}$$

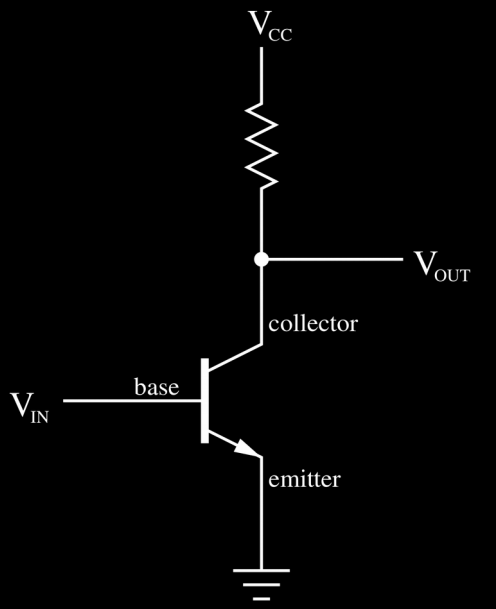
Capacitor: Slow down fluctuations

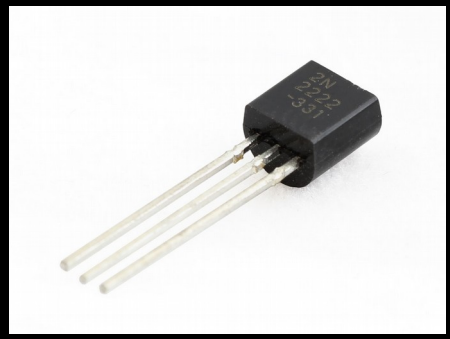
- Like a fast-charging/draining battery
- Can be used to “damp down” rapid changes in voltage (noise reduction):

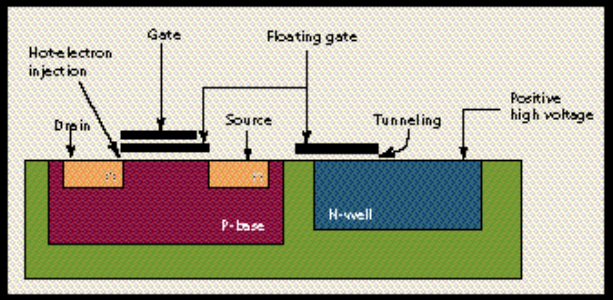


Transistor: Use one signal to control another

- Analog usage: amplifier
- Digital usage: on/off switch for Boolean logic gates
- The basis of all modern digital computers
- Integrated circuit shrinks billions of these down to a tiny die (chip) you can conceal under your finger

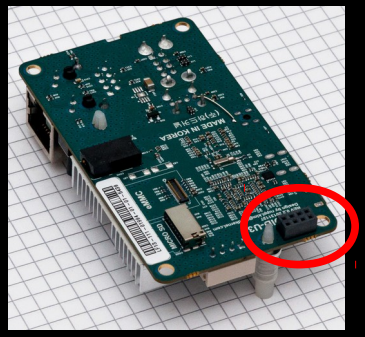


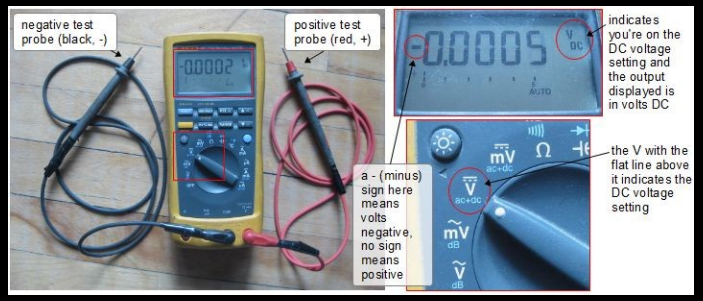




Typical Voltages

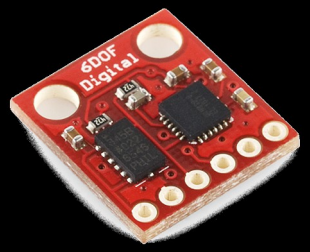
- 1.8V: UART on ODROID U-3
- 3.3V: Many sensors; Arduino Due; UART on Raspberry Pi and ODROID C-1
- 5V: Many sensors; most Arduinos
- 7 – 12 V: Vin (voltage input) for Arduino

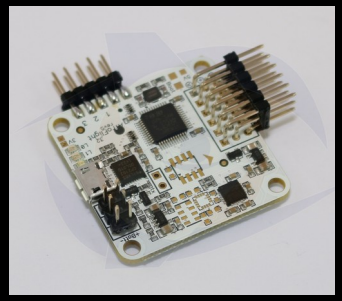




Multimeter

Popular Sensors: Inertial Measurement Unit (Accelerometer + Gyroscope)



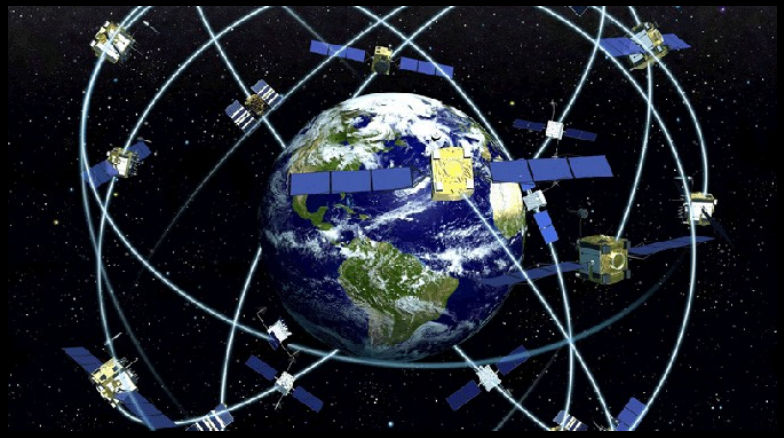


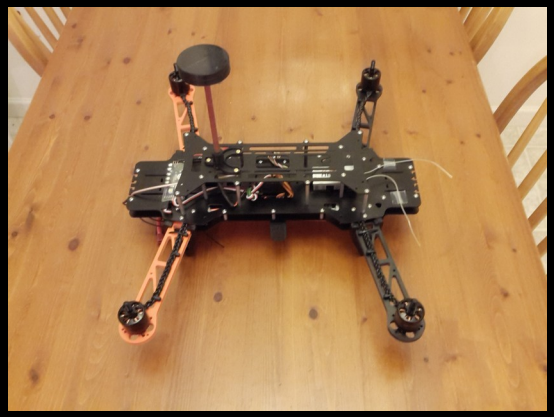
Acro Naze32

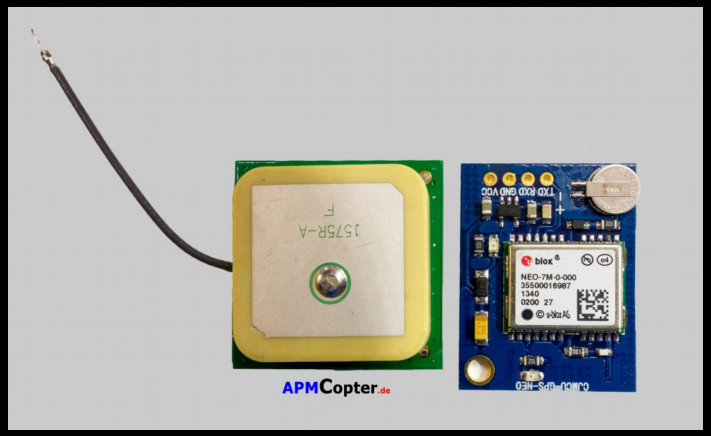
OpenPilot CC3D

- Gyro is a Micro-Electro-Mechanical Sensor
- IMU = minimal requirement for Flight Management Unit
- Typical signal type is I^2C

Popular Sensors: GPS

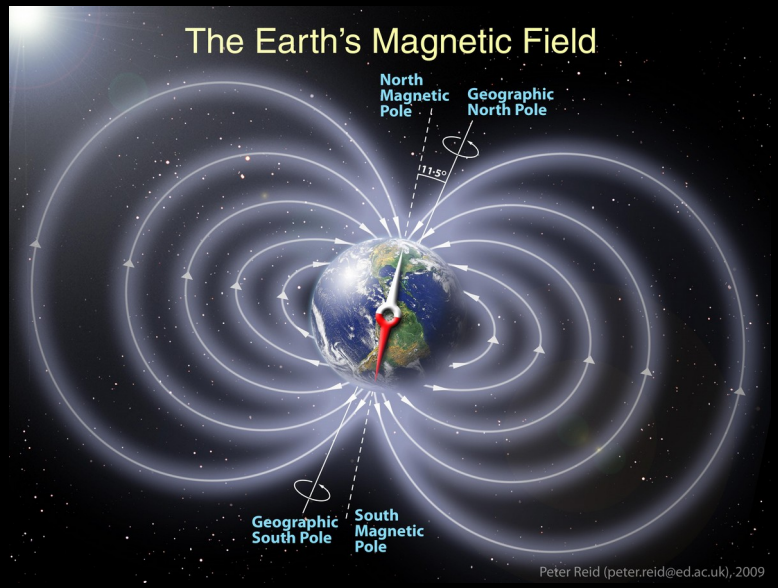






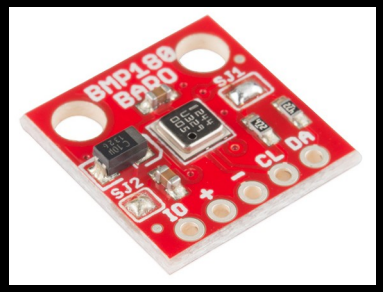
- Typical signal type is TTL
- Traditional data format = NMEA
- UBX format from UBlox is gaining

Popular Sensors: Magnetometer (Compass)

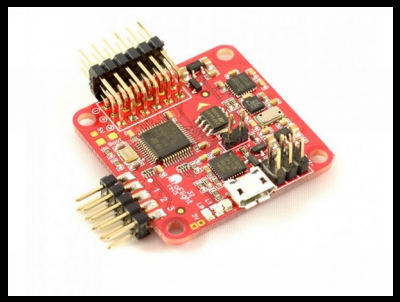


- Typical signal is I^2C

Popular Sensors: Barometer (Altimeter)



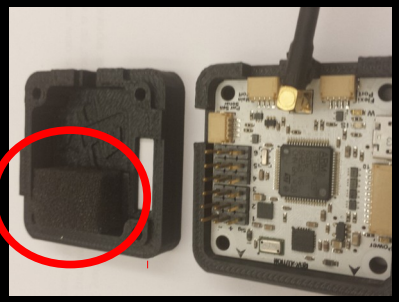
OpenPilot Revolution



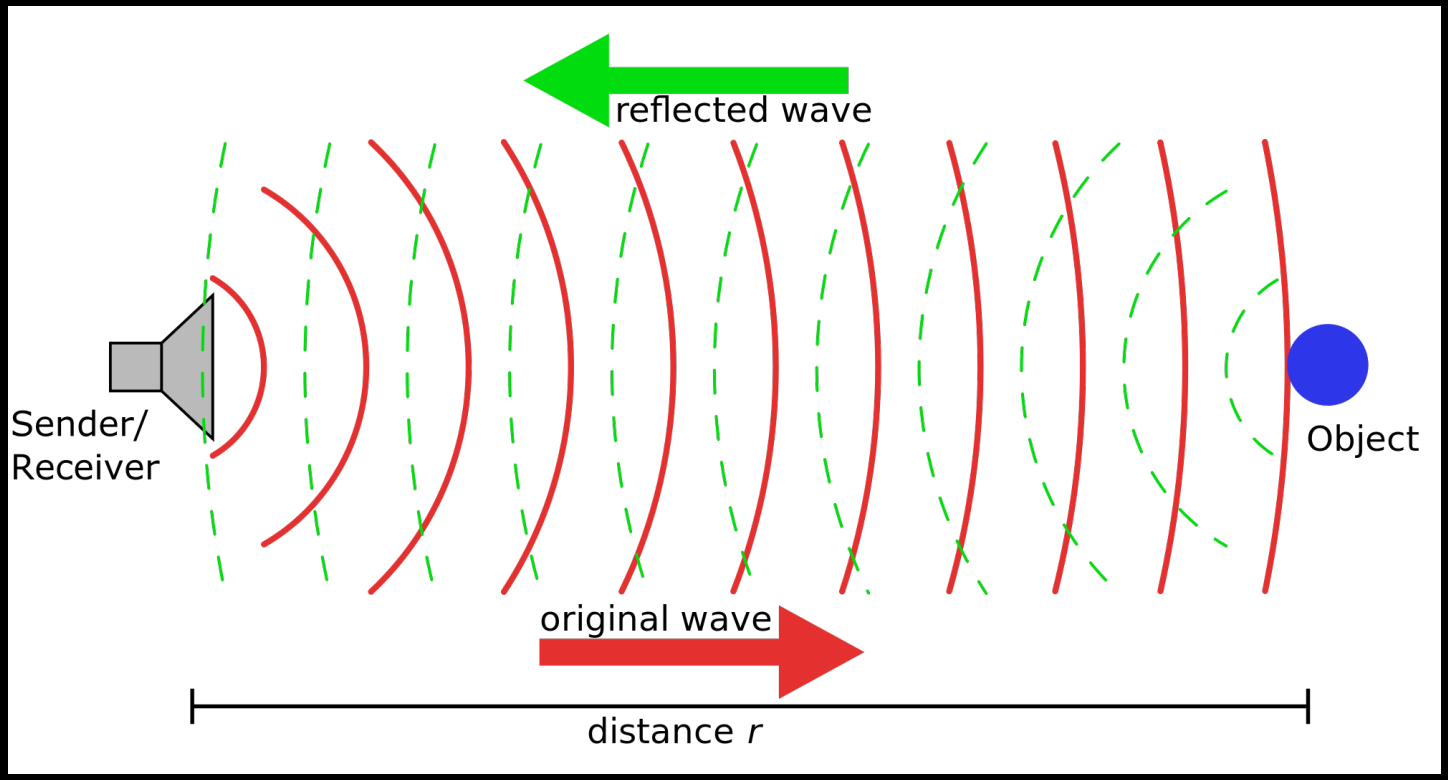
Naze32

IMU + Compass + Baro

- Typical signal is I^2C
- Cover with open-cell foam (sponge) to avoid airflow noise

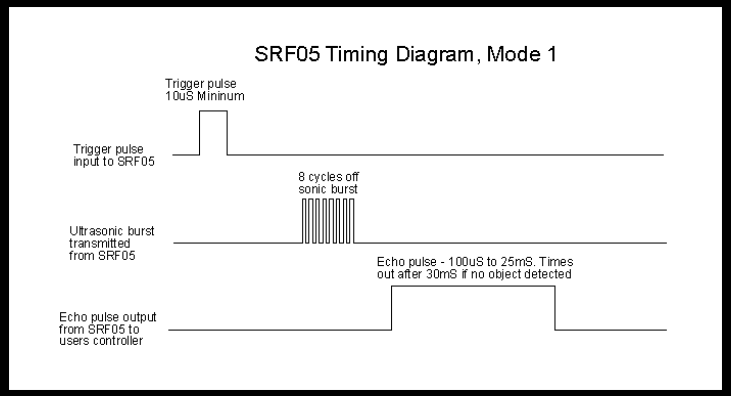


Popular Sensors: Ultrasonic (Sonar)



http://upload.wikimedia.org/wikipedia/commons/thumb/0/07/Sonar_Principle_EN.svg/2000px-Sonar_Principle_EN.svg.png

Popular Sensors: Ultrasonic (Sonar)



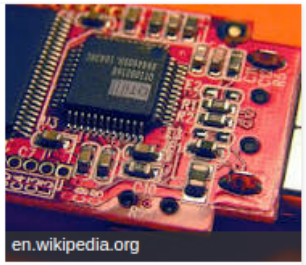
MaxBotix EZ-Sensor (USB)

Devantech SRF05

MaxBotix MB1240 (I^2C, TTL, raw voltage)

A Few More Useful Terms

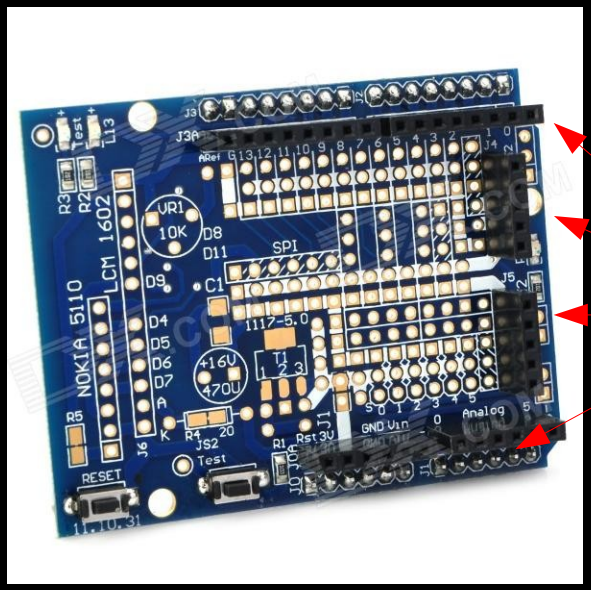
Surface-mount technology (SMT) is a method for producing electronic circuits in which the components are **mounted** or placed directly onto the **surface** of printed circuit boards (PCBs). An electronic device so made is called a **surface-mount** device (SMD).



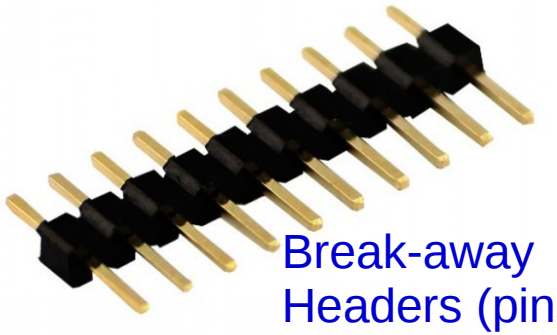
Surface-mount technology - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/**Surface-mount**_technology Wikipedia

Printed circuit board

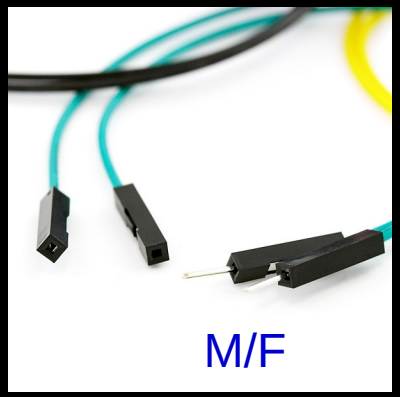
From Wikipedia, the free encyclopedia
(Redirected from Breakout board)



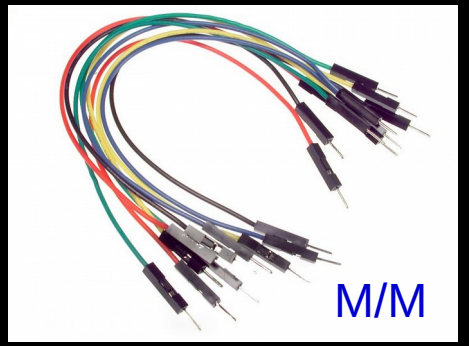
Expansion board a.k.a. shield



Break-away Headers (pins)



M/F



M/M

Jumper wires

Signals from Sensors

- Sensors output data serially (one bit at a time)
- Computers process data in parallel (one byte or word or more at a time)
- UART: Universal Asynchronous Receiver / Transmitter : device on your computer that converts serial i/o to parallel format usable internally.

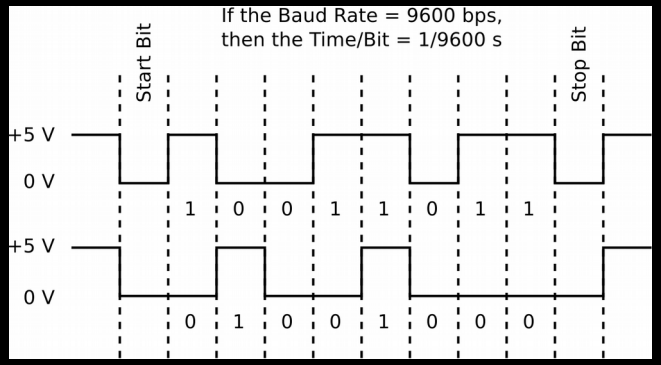
Serial Issues

- BAUD rate: symbols/pulses per second
- Typically treated as bits per second (bps), but not necessarily the same
- Usually have to specify for each device (sensor)
- Standard / common values are

  4800, 9600,

  19200, 38400,

  57600, 115200



http://www.fiz-ix.com/wp-content/uploads/2013/02/SerialCommunication.png

Émile Baudot
(1845-1903)

Serial Issues

Bit number	1	2	3	4	5	6	7	8	9	10	11
	Start bit	5–8 data bits								Stop bit(s)	
	Start	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Stop	

http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter

- Byte size (almost always eight)
- Parity bit: Final data bit can be optionally used as an error check
- Stop bit(s): signals end-of-byte: typically, one bit

7 bits of data	(count of 1 bits)	8 bits including parity	
		even	odd
0000000	0	00000000	00000001
1010001	3	10100011	10100010
1101001	4	11010010	11010011
1111111	7	11111111	11111110

http://en.wikipedia.org/wiki/Parity_bit

Serial Port

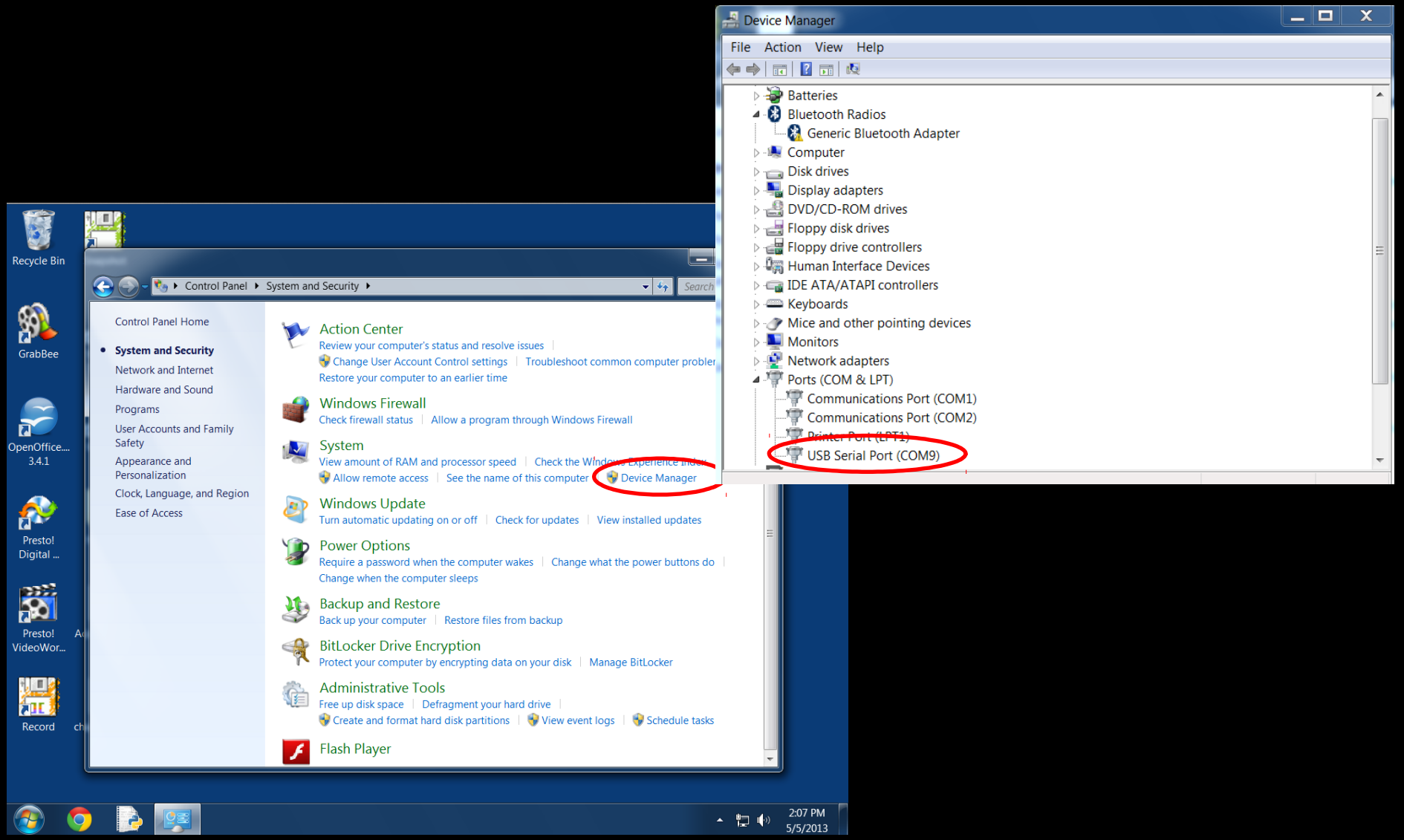
- Ambiguous usage: can refer to
  - DE-9 (DB-9) connector on the back of an older PC / laptop:

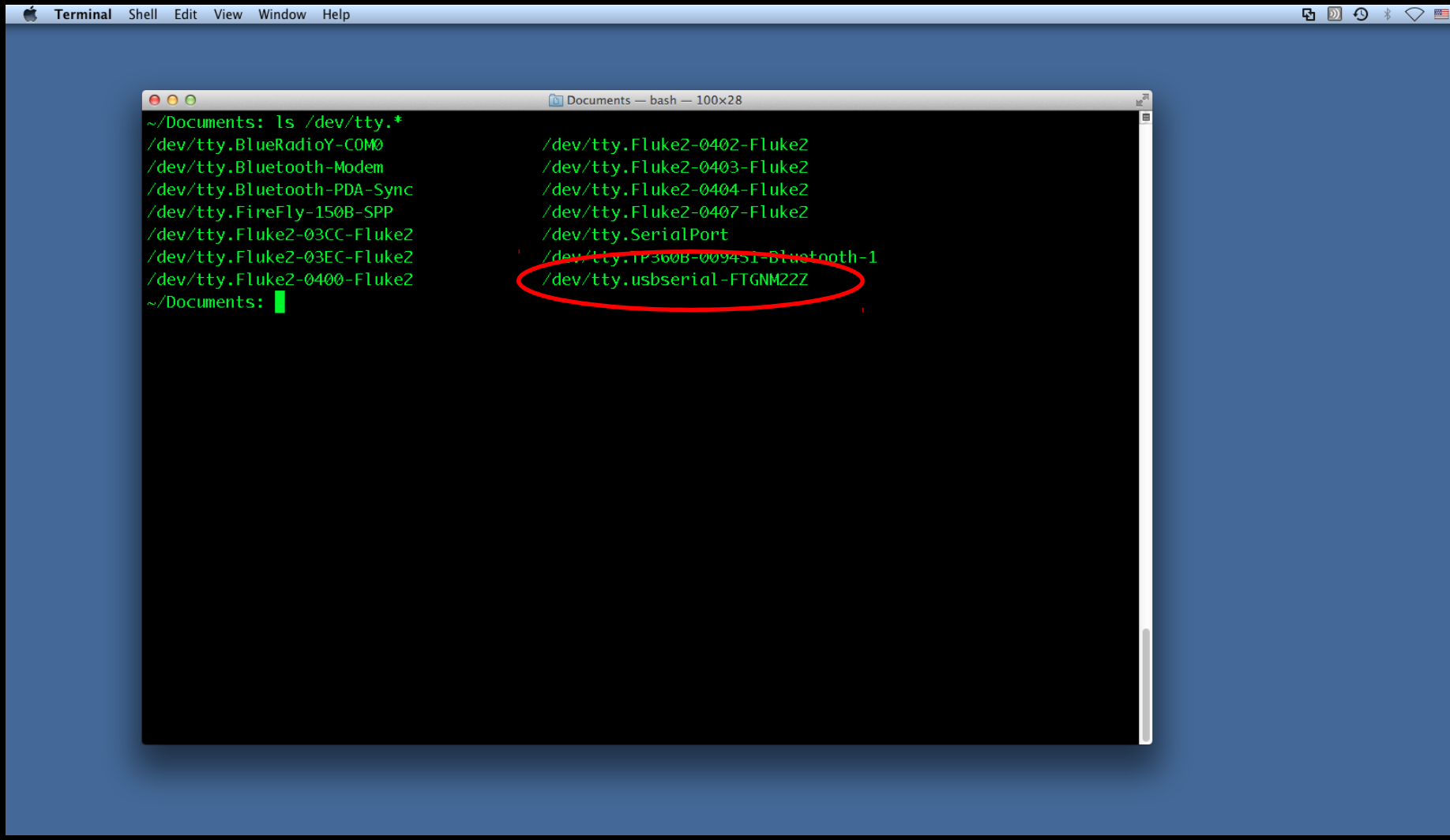
Serial Port

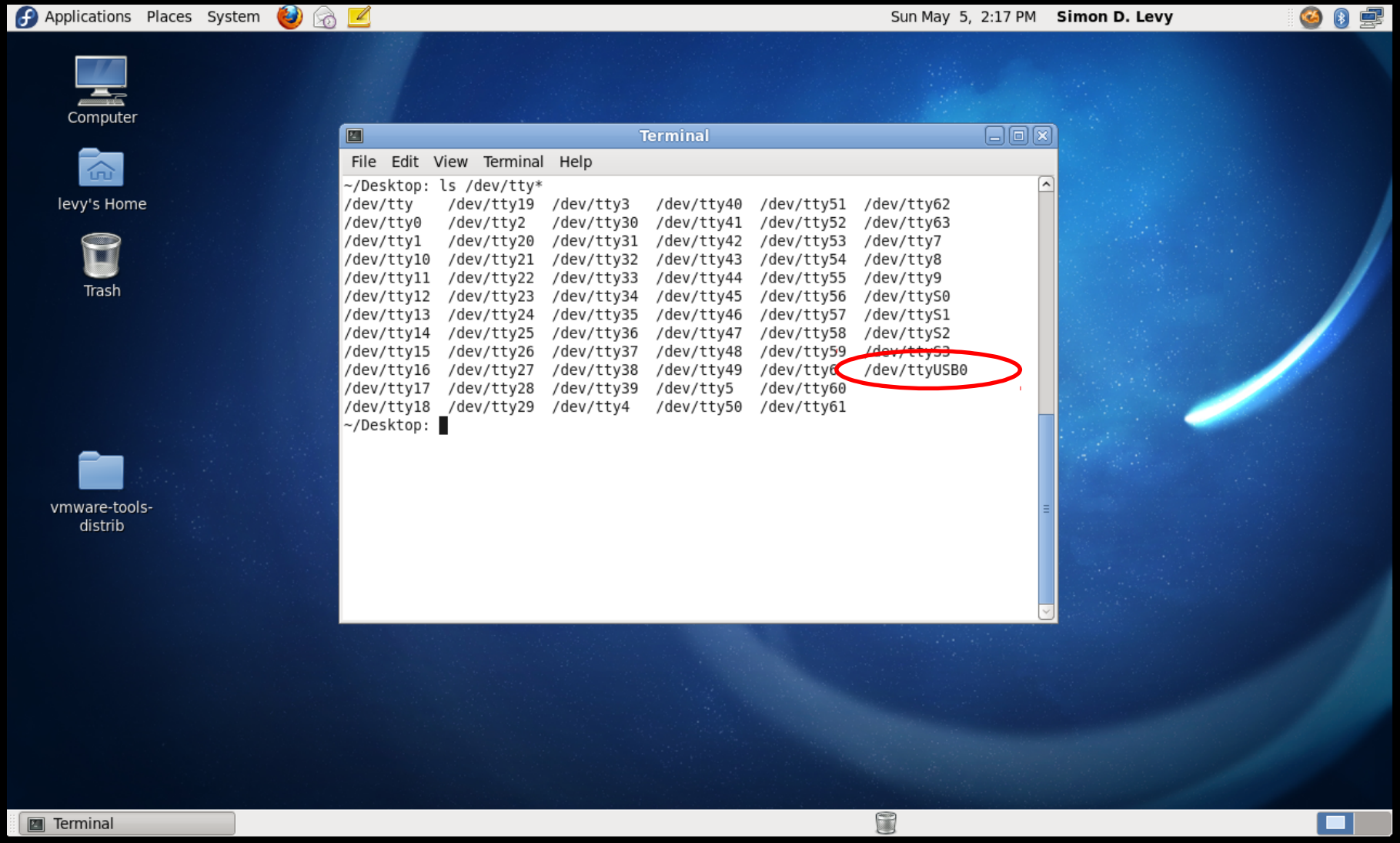
- Ambiguous usage: can refer to
  - DE-9 (DB-9) connector on the back of an older PC / laptop:

  - The representation of such a device by the operating system

Serial Ports on Popular OSs

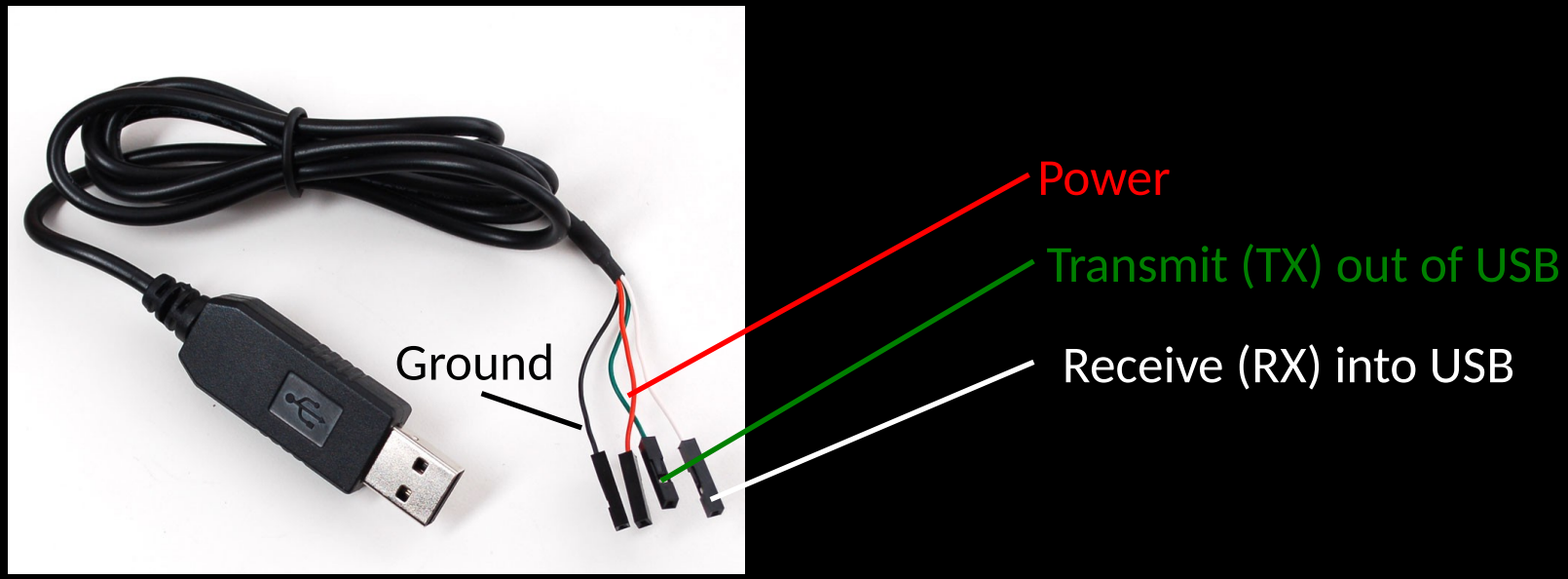



Terminal Shell Edit View Window Help
Documents — bash — 100×28
~/Documents: ls /dev/tty.*
/dev/tty.BlueRadioY-COM0
/dev/tty.Bluetooth-Modem
/dev/tty.Bluetooth-PDA-Sync
/dev/tty.FireFly-150B-SPP
/dev/tty.Fluke2-03CC-Fluke2
/dev/tty.Fluke2-03EC-Fluke2
/dev/tty.Fluke2-0400-Fluke2
/dev/tty.Fluke2-0402-Fluke2
/dev/tty.Fluke2-0403-Fluke2
/dev/tty.Fluke2-0404-Fluke2
/dev/tty.Fluke2-0407-Fluke2
/dev/tty.SerialPort
/dev/tty.usbserial-FTGNM22Z
~/Documents:


Applications
Places
System
Sun May 5, 2:17 PM
Simon D. Levy
Computer
levy's Home
Trash
vmware-tools-distrib
Terminal
File Edit View Terminal Help
~/Desktop: ls /dev/tty*
/dev/tty /dev/tty19 /dev/tty3 /dev/tty40 /dev/tty51 /dev/tty62
/dev/tty0 /dev/tty2 /dev/tty30 /dev/tty41 /dev/tty52 /dev/tty63
/dev/tty1 /dev/tty20 /dev/tty31 /dev/tty42 /dev/tty53 /dev/tty7
/dev/tty10 /dev/tty21 /dev/tty32 /dev/tty43 /dev/tty54 /dev/tty8
/dev/tty11 /dev/tty22 /dev/tty33 /dev/tty44 /dev/tty55 /dev/tty9
/dev/tty12 /dev/tty23 /dev/tty34 /dev/tty45 /dev/tty56 /dev/ttyS0
/dev/tty13 /dev/tty24 /dev/tty35 /dev/tty46 /dev/tty57 /dev/ttyS1
/dev/tty14 /dev/tty25 /dev/tty36 /dev/tty47 /dev/tty58 /dev/ttyS2
/dev/tty15 /dev/tty26 /dev/tty37 /dev/tty48 /dev/tty59 /dev/ttyS3
/dev/tty16 /dev/tty27 /dev/tty38 /dev/tty49 /dev/tty6 /dev/ttyUSB0
/dev/tty17 /dev/tty28 /dev/tty39 /dev/tty5 /dev/tty60
/dev/tty18 /dev/tty29 /dev/tty4 /dev/tty50 /dev/tty61
~/Desktop:
Terminal

Serial Signal Types: TTL (1961)

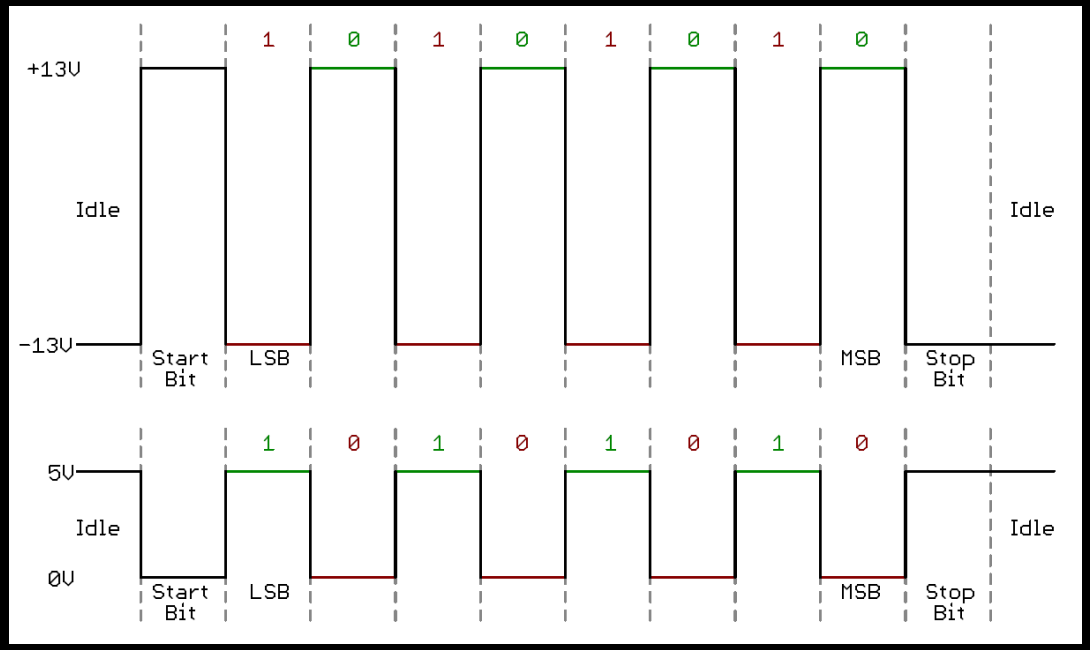
- Transistor-Transistor Logic: internal signal in computer
- Simple logic/voltage relationship: 0v = False; +3v to +5v = True
- Mostly seen in UARTs on microcontrollers & other special-purpose devices



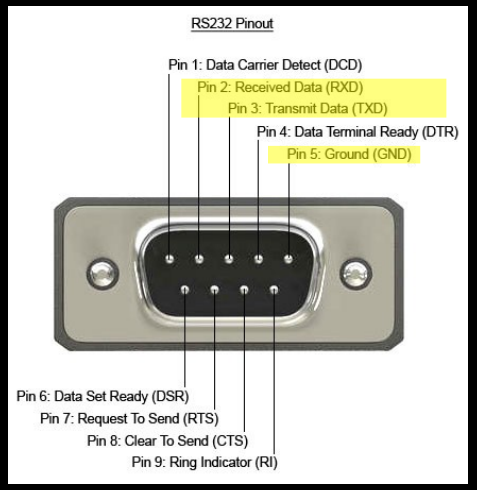
http://www.adafruit.com/products/954?gclid=CMaplr7Q_7YCFQsy4AodBEcAYQ

Serial Signal Types: RS-232 (1962)

- Most common PC UART signal for decades
- Higher voltage range (+/-13v) gives greater robustness than TTL
- Negative voltage = True; Positive = False



https://www.sparkfun.com/tutorials/215



http://www.usconverters.com/index.php?main_page=page&id=61&chapter=0

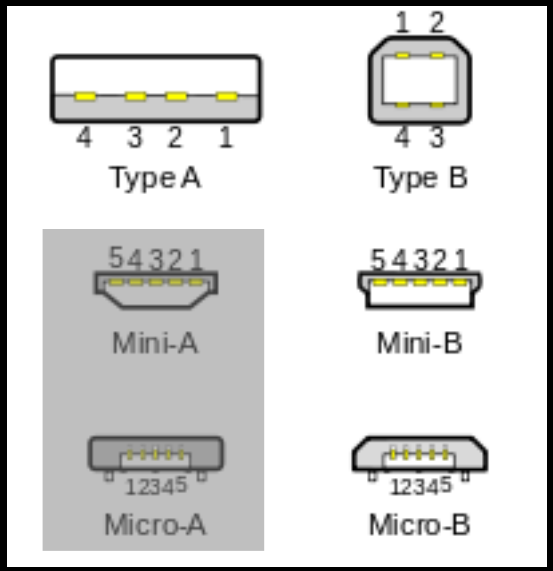
Serial Signal Types: USB (1995)

- Universal Serial Bus: vast majority of modern devices
- Variety of connector types
  - Standard / Mini / Micro
  - A (host/master) B (peripheral/slave)
  - OTG: "On The Go": Accepts A (make me a host) or B (make me a peripheral)
- Often used as power source / charger

- Some sensors are USB-ready, but for many we need an adapter.

USB 1.x/2.0 standard pinout

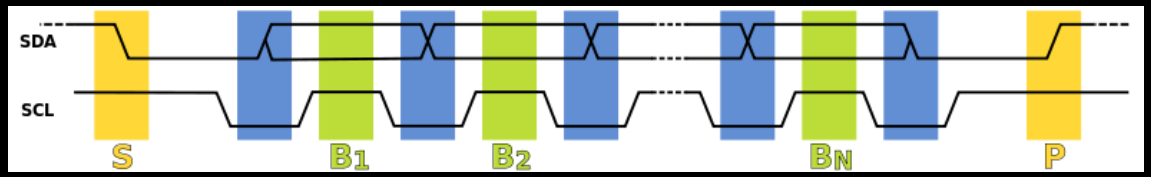
Pin	Name	Cable color	Description
1	VBUS	Red (or Orange)	+5 V
2	D–	White (or Gold)	Data –
3	D+	Green	Data +
4	GND	Black (or Blue)	Ground



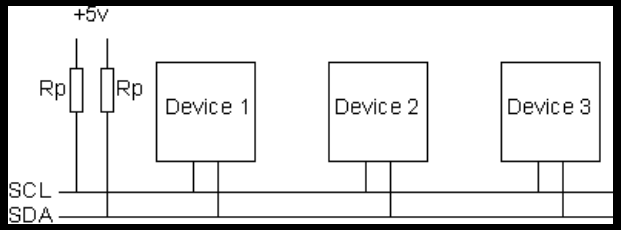
http://en.wikipedia.org/wiki/Usb

Serial Signal Types: I^2C (1982)

- Inter-Integrated Circuit
- From Philips; developed as SMBus by Intel
- Also called TWI (Two-Wire Interface) to avoid lawsuits
- Popular on robot sensors
- Allows several "slave" (sensor) devices to be accessed by same "master" (computer) via port (`/dev/i2c-0`) and addresses (`0x10, 0x12, ...`)
- Data (SDA) and clock (SCL) lines instead of TX and RX
- Active state is 0v, so need a +5v source plus "pullup resistors" to maintain idle state



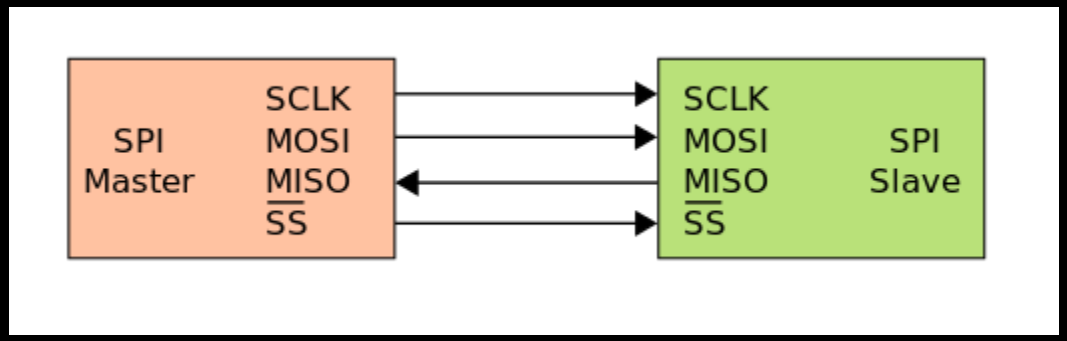
http://en.wikipedia.org/wiki/File:I2C_data_transfer.svg

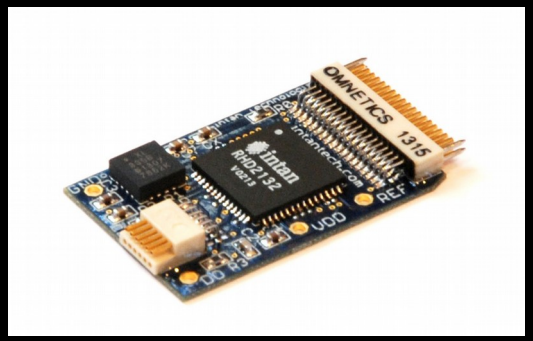


http://wiki.bozemanmakers.com/index.php/I2c

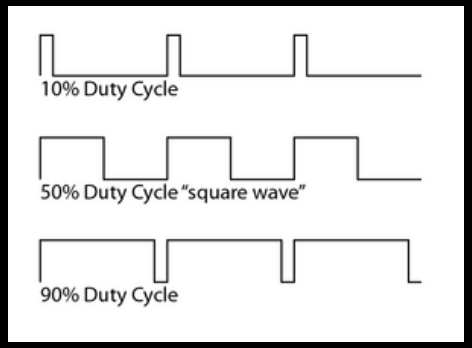
Other Signal Types: SPI, PWM

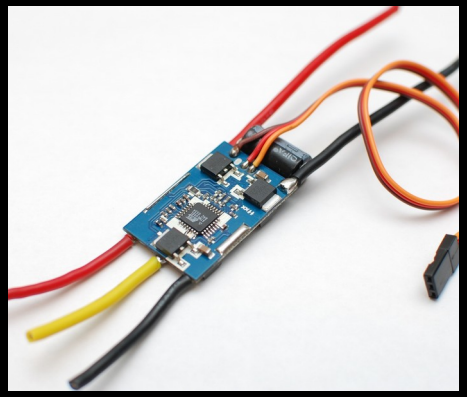
- Serial Peripheral Interface (Motorola)





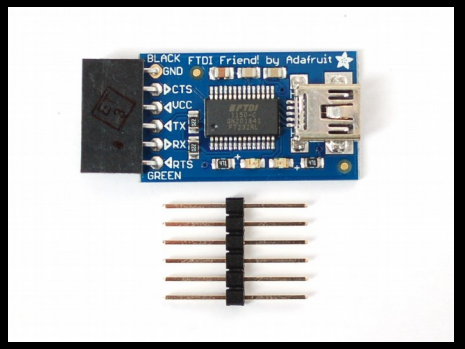
- Pulse Width Modulation





Favorite USB adapters

FTDI USB standard-A / RS232 DB-9
(DigiKey, Amazon)



"FTDI Friend" USB mini-B /
RS232 header pins
(AdaFruit)

Devantech USB standard-B to I^2C
(Acroname)

I like these adapters because the drivers for them are already installed (Linux, OS X) or are downloaded automatically when you plug in the adapter (Windows).

pySerial API

- API: Application Programming Interface: allows you to use your favorite language for a difficult task
- http://pyserial.sourceforge.net/pyserial_api.html
- Simple, powerful interface to serial port:
  - `Serial()` constructor
  - `read()`
  - `write()`
  - `close()`
- Basic steps:
  - Google the specs for your device or adapter
  - Download the device driver from the manufacturer's website (often not necessary)
  - Plug in the device and find its com-port
  - Start coding!

pySerial Example:
TruPulse 360 Laser Rangefinder

Section 8 - Serial Data Interface

The TruPulse includes a hard-wired serial (RS-232) communication port. Wireless Bluetooth communication is available as an option on the TruPulse 360B. In either case, the measurement data downloaded from the TruPulse is in ASCII Hex format, and duplicates LTI's Criterion 400 (CR400) communication protocol and download messages.

Requirements for transferring serial data using hard-wired connection:

- Serial data transfer cable to connect the TruPulse to the PC, such as:
  - 36-inch LTI 4-Pin to DB9 Download Cable (7053038)
  - 36-inch LTI 4-Pin to DB9 Download Cable with Remote Trigger (7054223)
  - 5-meter LTI 4-Pin to DB9 Download Cable (7054244)
- Data collection software installed on PC, Pocket PC, or other data collection device.

Requirements for transferring serial data using Bluetooth connection:

- See page 19.
- Data collection software installed on a Bluetooth enabled laptop PC, Pocket PC, etc.

Format Parameters

4800 baud, 8 data bits no parity, 1 stop bit

Serial Port

Figure #26 shows the pin-out assignments for TruPulse's serial port.

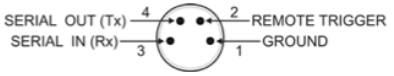


Figure #27

Download Instructions

The instructions below are provided for general information only. Specific steps may vary, depending upon your data collection program.

1. Connect the TruPulse to the PC, Pocket PC, etc.
2. Start the data collection program on the PC and adjust settings to match format parameters (4800 baud, 8 data bits no parity, 1 stop bit).
3. Power ON the TruPulse.
4. Verify/select measurement units, Measurement Mode, and Target Mode.
5. Take the desired measurement. The measurement result flashes one time indicating that it is being downloaded.

http://solutions.seilerinst.com/Portals/1/Mapping%20PDFs/TruPulse_360-B_Users_Manual_2nd_Edition_Englis

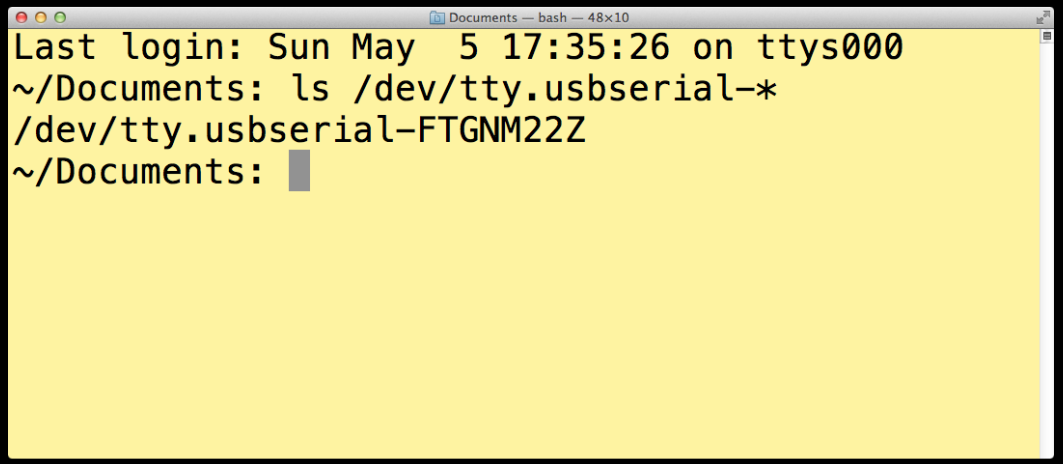
pySerial API

Classes

Native ports

class `serial.`**`Serial`** ¶

`__init__`(*port=None, baudrate=9600, bytesize=EIGHTBITS, parity=PARITY_NONE, stopbits=STOPBITS_ONE, timeout=None, xonxoff=False, rtscts=False, writeTimeout=None, dsrdtr=False, interCharTimeout=None*)




```
# trupulse.py
import serial
import sys

DEVICE = '/dev/tty.usbserial-FTGNM22Z'  # or /dev/ttyUSB0 or COM9, etc.

device = serial.Serial(DEVICE, 4800)

while True:
    c = device.read(1)  # read one byte
    sys.stdout.write(c) # print the byte as an ASCII character, no spaces

device.close()
```

```
lectures — Python — 80×22
~/Documents/csci250w2013/lectures: python trupulse.py
$PLTIT,HV,8.50,F,74.40,D,6.30,D,8.50,F*55
$PLTIT,HV,9.01,F,91.70,D,-2.70,D,9.01,F*70
$PLTIT,HV,8.50,F,68.00,D,12.90,D,8.50,F*63
$PLTIT,HV,27.50,F,90.10,D,3.10,D,27.50,F*5D
```

Download Message Format

The CR400 data format follows the guidelines of the NMEA Standard for interfacing Marine Electronic Navigational Devices, Revision 2.0. NMEA 0183 provides for both standard and proprietary data formats. Since none of the standard formats are useful for the data transferred from the TruPulse, special proprietary formats are used. Rules described in the NMEA standard governing general message structure, leading and trailing characters, numeric values, delimiting character, checksums, maximum line length, data rate, and bit format are followed exactly. As required by NMEA 0183, the CR400-format does not respond to unrecognized header formats, malformed messages, or messages with invalid checksums.

Download Message Formats

Horizontal Vector (HV) Download Messages

$PLTIT,HV,HDvalue,units,AZvalue,units,INCvalue,units,SDvalue,units,*csum**<CR><LF>**
where:

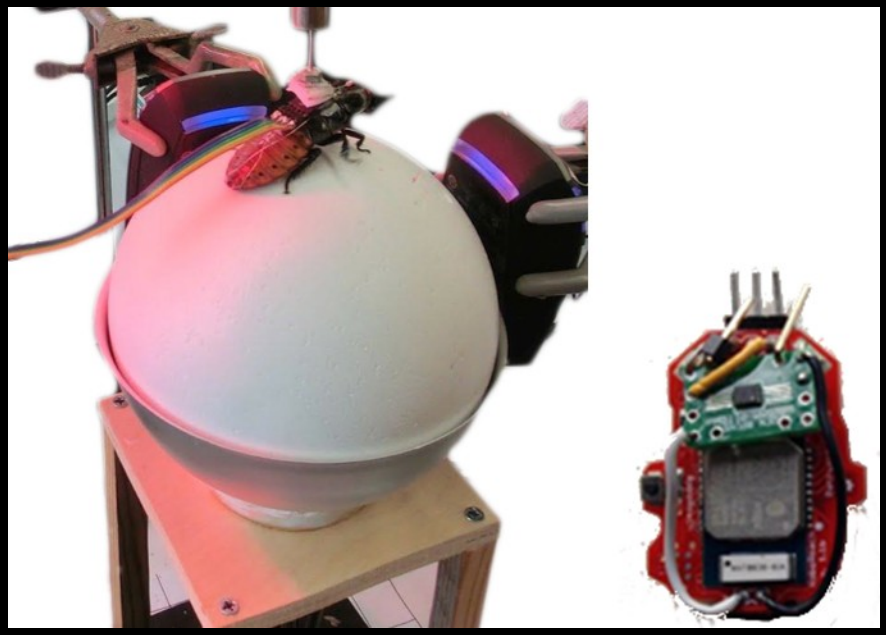
$PLTIT,		is the Criterion message identifier.
HV,		Horizontal Vector message type.
HDvalue,		Calculated Horizontal Distance. Two decimal places.
	units,	F=feet Y=yards M=meters
AZvalue,		Measured Azimuth . Two decimal places. Percent slope is not downloaded.
	units,	D=degrees
INCvalue,		Measured Inclination value. Two decimal places. May be positive or negative value.
	units,	D=degrees
SDvalue,		Measured Slope Distance Value. Two decimal places.
	units,	F=feet Y=yards M=meters
*csum		An asterisk followed by a hexadecimal checksum. The checksum is calculated by XORing all the characters between the dollar sign and the asterisk.
<CR>		Carriage return.
<LF>		Optional linefeed.

Arduino: A Microcontroller Solution

Microcontroller

- A simple computer that runs only one program at a time, called the firmware (*software* changes often; *hardware* changes infrequently, so “firmware”). You flash the firmware from your host computer to your microcontroller over a USB cable.
- Traditional microcontrollers used special firmware languages that didn’t translate to other devices.

Microcontroller



http://home.wlu.edu/~ericksonj/research.html

Arduino Microcontroller

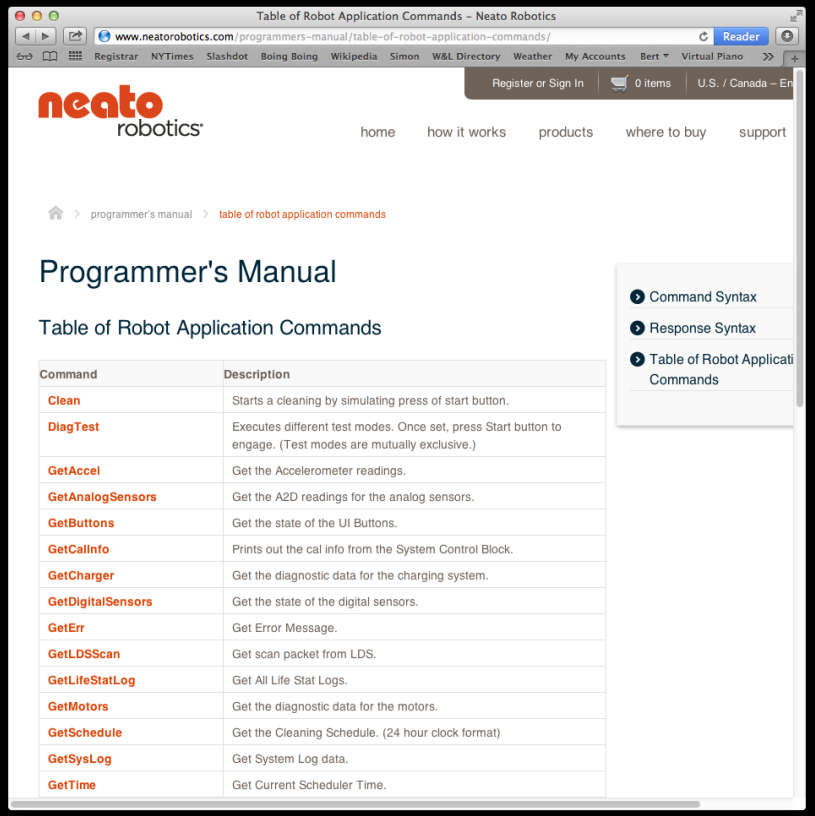
- Arduino can use:
  - Processing language: not difficult to learn, but used only for Arduino
  - C++ language: extremely popular, but difficult for beginners
- But:
  - There is a huge repository of firmware already written for many sensors
  - The Firmata firmware allows you to run the Arduino from your computer in any language (e.g., Python)

Arduino method I: Flash, Unplug, and Go

Arduino method II: Interact with host via Firmata or other firmware

Data Formats

- ASCII
  - NMEA (or other industry standard)
  - Platform-specific
- Binary
  - Images
  - Sounds
  - Numbers



NMEA Format

- Text-based (ASCII), comma-delimited
- E.g., GPS:

$GPRMC

Recommended minimum specific GPS/Transit data

```
eg1. $GPRMC,081836,A,3751.65,S,14507.36,E,000.0,360.0,130998,011.3,E*62
eg2. $GPRMC,225446,A,4916.45,N,12311.12,W,000.5,054.7,191194,020.3,E*68
```

225446	Time of fix 22:54:46 UTC
A	Navigation receiver warning A = OK, V = warning
4916.45,N	Latitude 49 deg. 16.45 min North
12311.12,W	Longitude 123 deg. 11.12 min West
000.5	Speed over ground, Knots
054.7	Course Made Good, True
191194	Date of fix 19 November 1994
020.3,E	Magnetic variation 20.3 deg East
*68	mandatory checksum

http://aprs.gids.nl/nmea/#rmc

Checksum: Like Parity, for a Whole Message

```
# compute checksum from message contents, terminated by asterisk
sum = 0
for c in msg:
    if c == '*':
        break
    sum ^= c
```

Handling Binary Data

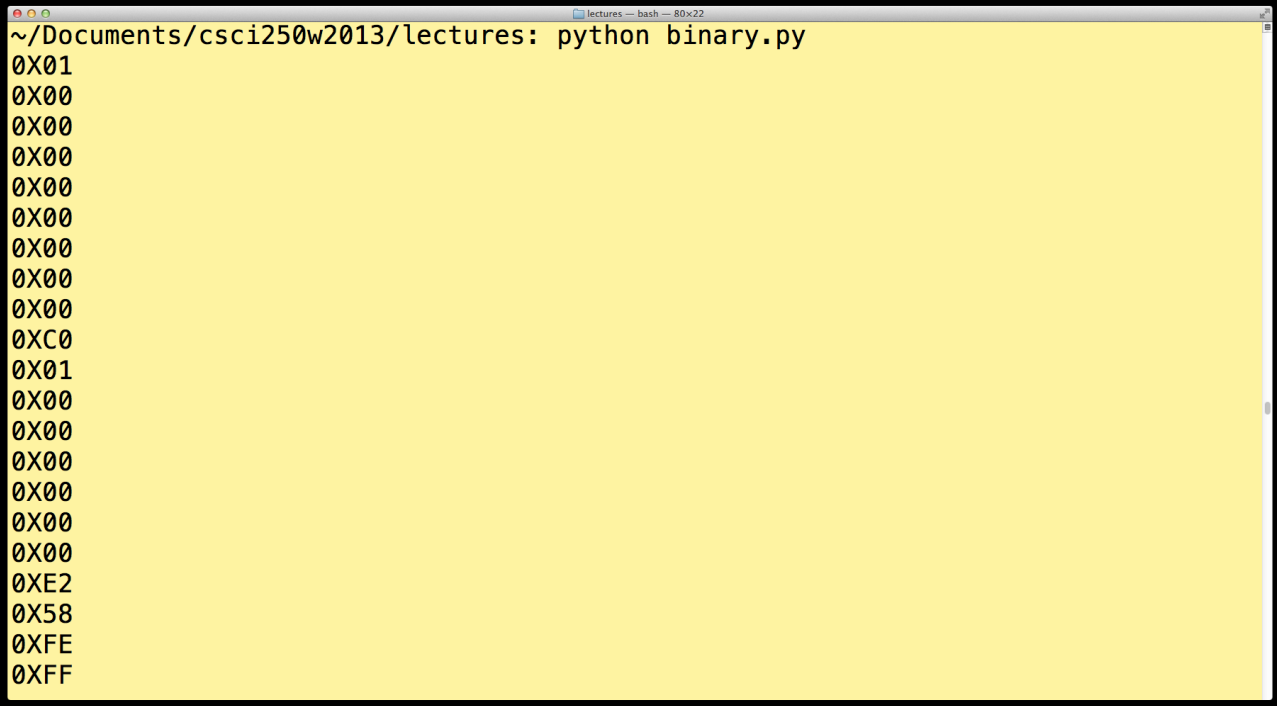
```
import serial
import sys

DEVICE = '/dev/tty.usbserial-FTGNM22Z'

device = serial.Serial(DEVICE, 38400)

while True:
    c = device.read(1)
    sys.stdout.write('0X%02X\n' % ord(c)) # Output as two-digit hexadecimal

device.close()
```



Binary Data: Byte Order

- Typically, LSB: Less (Least) Significant Byte first
- So if we're reading two-byte words:

```
0X00
0XE2
0X58
0XFE
...
```

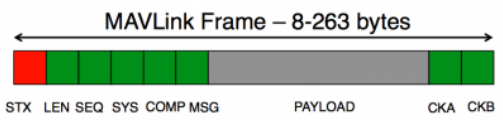
```
0XE200

0xFE58

...
```

- Then interpret words according to standard in docs (two's complement fixed-point, IEEE754 float, ...)

Binary Data Example: MAVLink for Drones



Byte Index	Content	Value	Explanation
0	Packet start sign	v1.0: 0xFE (v0.9: 0x55)	Indicates the start of a new packet.
1	Payload length	0 - 255	Indicates length of the following payload.
2	Packet sequence	0 - 255	Each component counts up his send sequence. Allows to detect packet loss
3	System ID	1 - 255	ID of the SENDING system. Allows to differentiate different MAVs on the same network.
4	Component ID	0 - 255	ID of the SENDING component. Allows to differentiate different components of the same system, e.g. the IMU and the autopilot.
5	Message ID	0 - 255	ID of the message - the id defines what the payload “means” and how it should be correctly decoded.
6 to (n+6)	Data	(0 - 255) bytes	Data of the message, depends on the message id.
(n+7) to (n+8)	Checksum (low byte, high byte)	ITU X.25/SAE AS-4 hash, **excluding packet start sign, so bytes 1..(n+6)** Note: The checksum also includes MAVLINK_CRC_EXTRA (Number computed from message fields. Protects the packet from decoding a different version of the same packet but with different variables).	

http://qgroundcontrol.org/mavlink/start

Sockets

- Socket: a mechanism allowing a process (executing program) can talk to other processes – on the same computer or across the planet
- A "low-level network interface"; i.e., what underpins the internet!
- Standard is Berkeley (BSD) Sockets
- Client/server model
  - Server listens at a specified address on a specified port
  - Client connects to the socket at that address/port

```
# server.py

import socket

HOST = '137.113.118.74'     # dijkstra.cs.wlu.edu
PORT = 20000                # Low numbers are reserved for SSH, HTTP, SQL, et al.

sock = socket.socket()

try:
    sock.bind((HOST, PORT)) # Note tuple!

except socket.error, msg:
    print('bind() failed with code ' + str(msg[0]) + ': ' + msg[1])

sock.listen(1)  # handle up to 1 back-logged connection

client, address = sock.accept()

print('Accepted connection')

while True:

    try:
        msg = raw_input('> ')
        if len(msg) < 1:
            break
        client.send(msg)

    except:
        print('Failed to transmit')
        break

client.close()
sock.close
```

```
# client.py

import socket

HOST = '137.113.118.74'
PORT = 20000

sock = socket.socket()

try:
    sock.connect((HOST, PORT)) # Note tuple!

except socket.error, msg:
   print('connect() failed with code ' + str(msg[0]) + ': ' + msg[1])

while True:

    try:
        msg = sock.recv(80) # Maximum number of bytes we expect
        if len(msg) < 1:
            break
        print(msg)

    except:
        print('Failed to receive')
        break

sock.close()
```

Creating an Ad-Hoc WiFi Network on Ubuntu (e.g., RaspberryPi)

https://github.com/simondlevy/RPiAdHocWiFi

RPiAdHocWiFi

Simple ad-hoc wireless network support for Raspberry Pi.

The RaspberryPi 3 comes with a Broadcom wifi chip onboard, making it easy to setup an ad-hoc wifi network for remote sensing, Internet of Things, and related projects. This little repository will help you do that. You should be running the Raspbian operating system on your Raspberry Pi.

NOTE: Following these instructions will clobber your existing startup script (/etc/rc.local) and DHCP configuration (/etc/udhcpd.conf). Proceed with caution.
After logging into your Raspberry Pi and cloning this repository, do the following from the command line:

```
% sudo apt install udhcpd
% sudo touch /var/lib/misc/udhcpd.leases
% cd RPiAdHocWiFi
% sudo cp rc.local /etc
% sudo cp udhcpd.conf /etc
% sudo reboot
```

Branch: **master** ▾ **RPiAdHocWiFi** / **rc.local**

 **simondlevy** Added comments

1 contributor

Executable File | 18 lines (15 sloc) | 462 Bytes

```
1  #!/bin/sh -e
2  #
3  # rc.local
4  #
5  # This script is executed at the end of each multiuser runlevel.
6  # Make sure that the script will "exit 0" on success or any other
7  # value on error.
8
9  ifconfig wlan0 down
10 iwconfig wlan0 mode Ad-Hoc
11 iwconfig wlan0 essid Raspberry-Pi-1  # Change Raspberry-Pi-1 to whatever name you like
12 sleep 1
13 ifconfig wlan0 192.168.2.2          # This should agree with the IP address in udhcpd.conf
14 ifconfig wlan0 up
15 udhcpd /etc/udhcpd.conf
16
17 exit 0
```

Branch: **master** ▾ **RPiAdHocWiFi** / **udhcpd.conf**

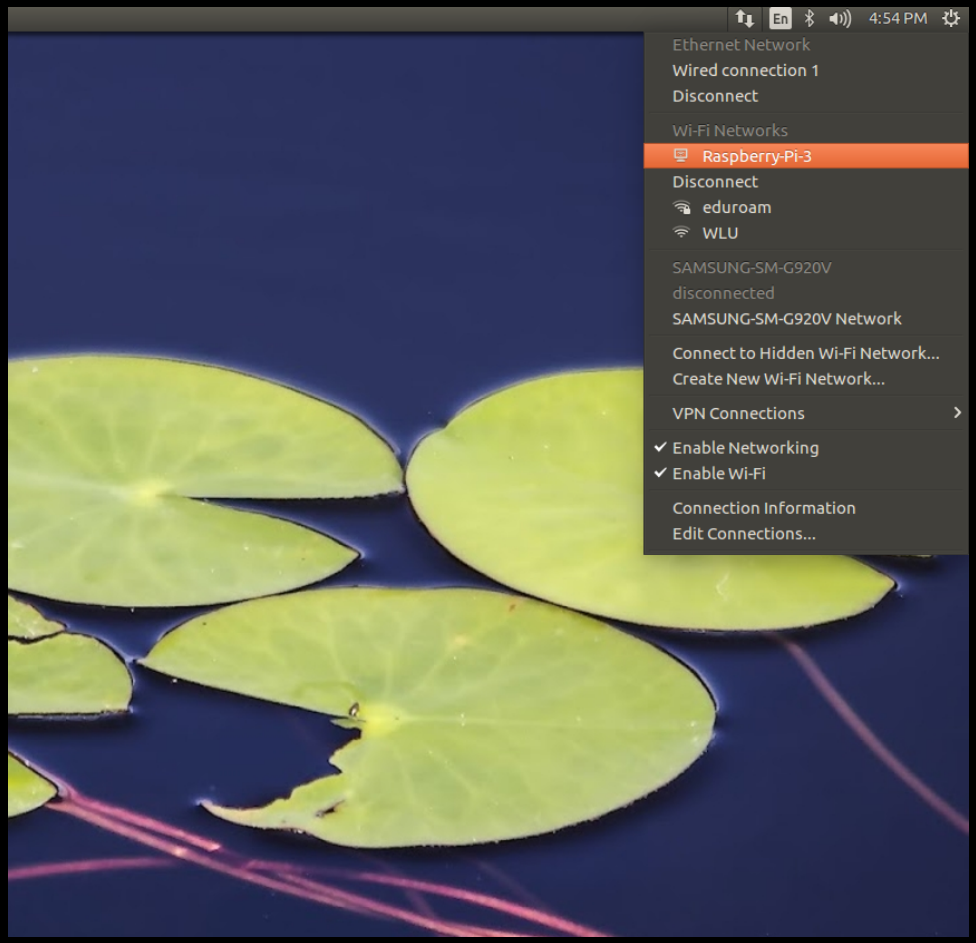
 **simondlevy** All files committed

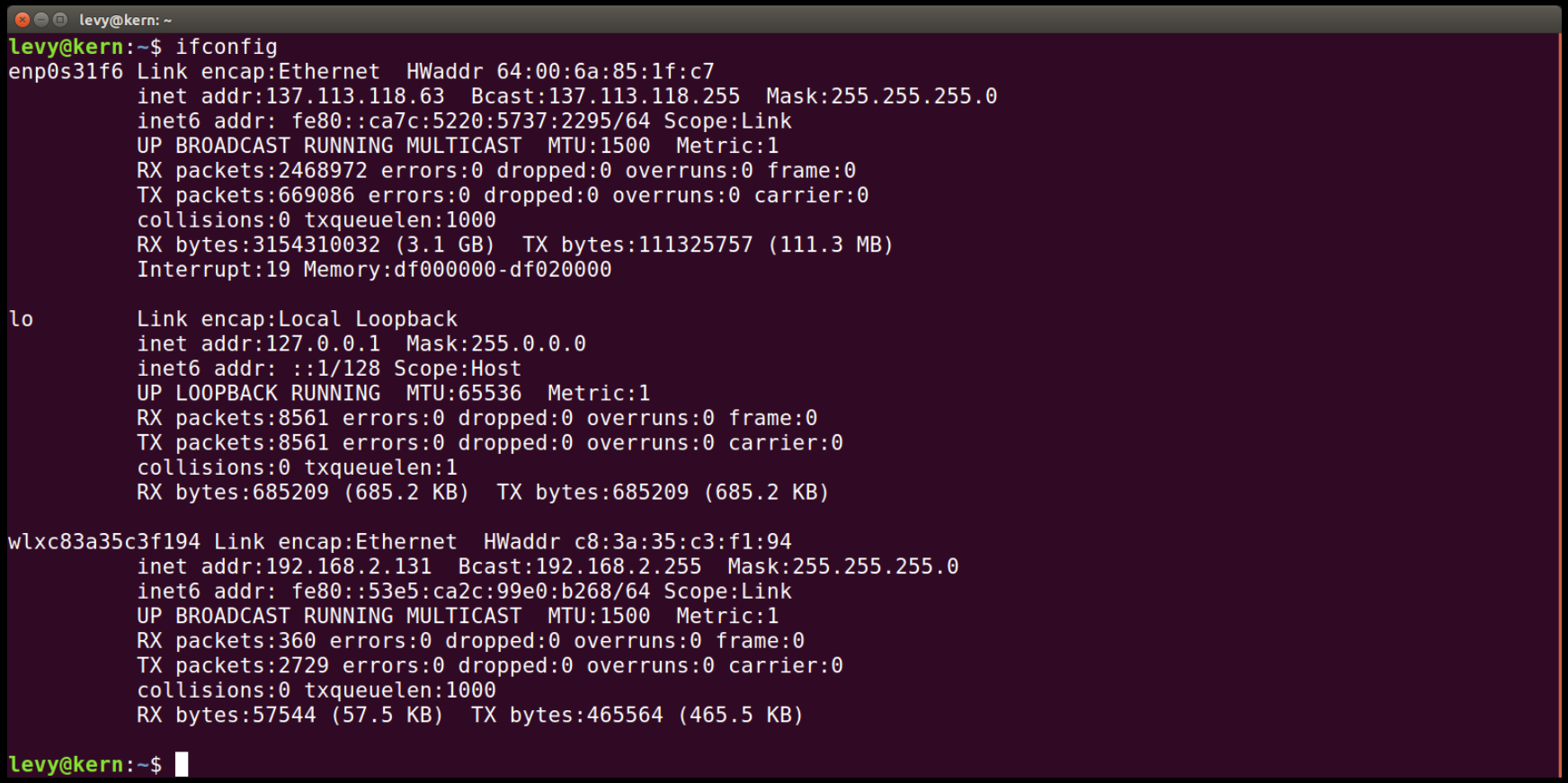
1 contributor

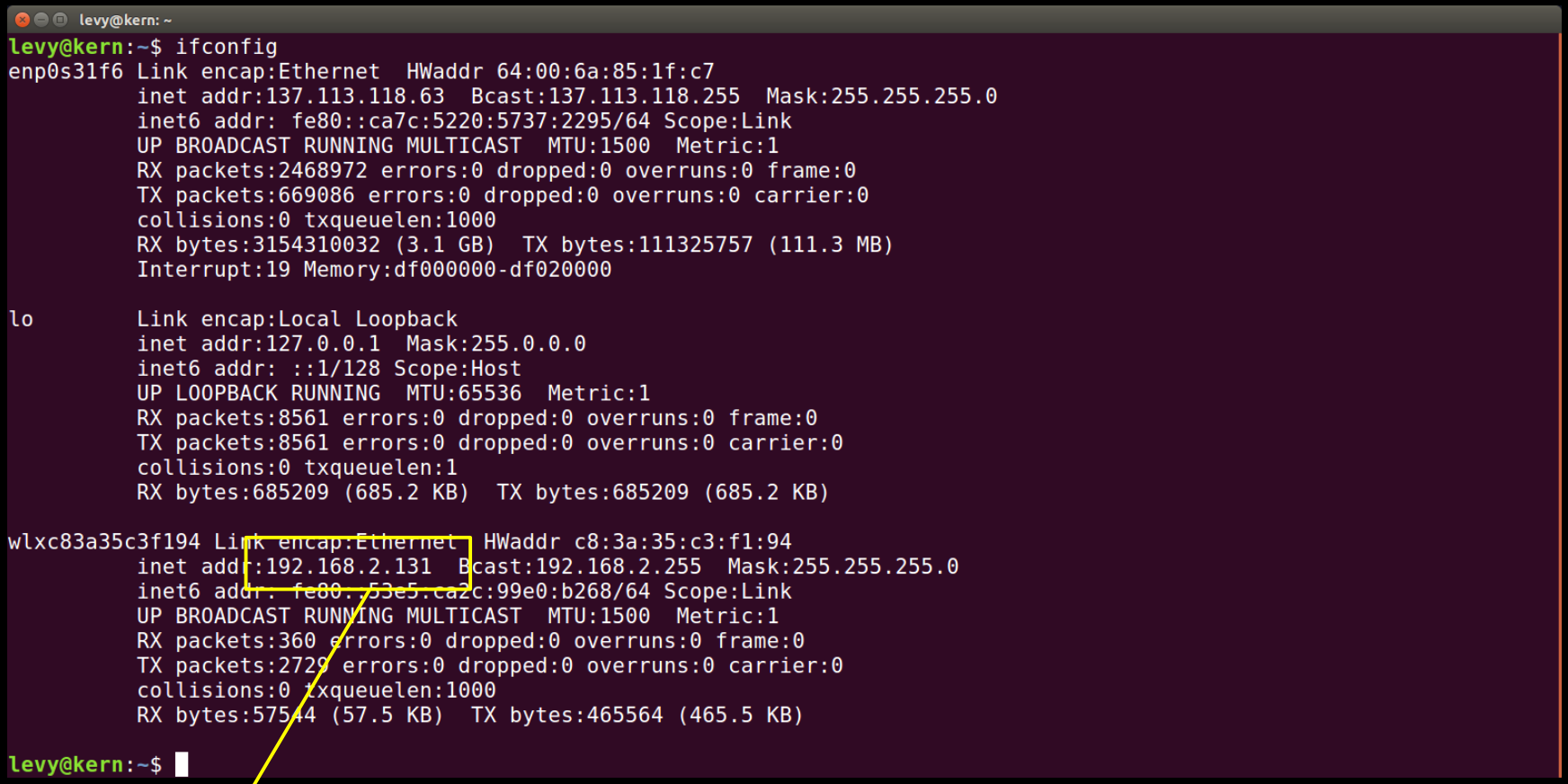
9 lines (7 sloc) | 183 Bytes

```
1  #
2  # This configuration will serve up IP addresses for an ad-hoc wireless network
3  #
4
5  start           192.168.2.3
6  end             192.168.2.254
7  interface       wlan0
8  max_leases      64
```

Connecting to the network from your computer

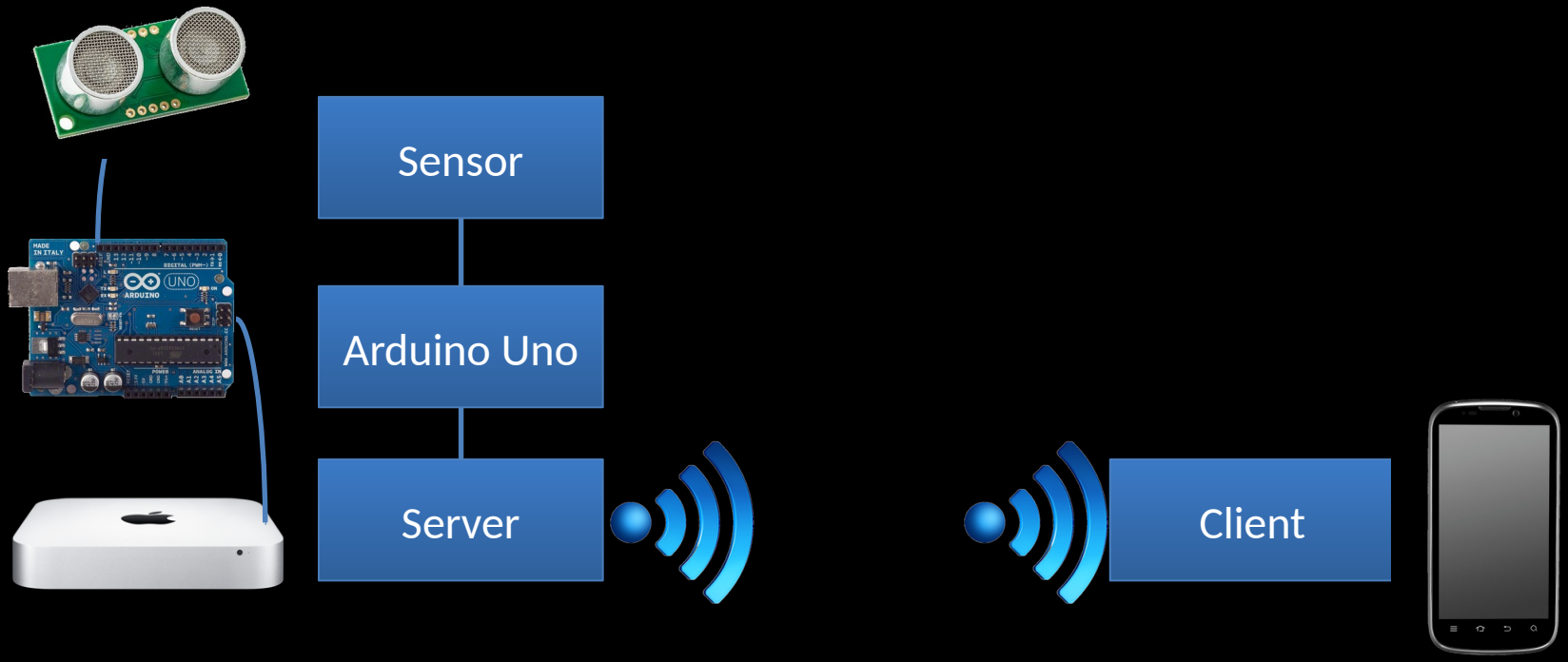



levy@kern: ~
levy@kern:~$ ifconfig
enp0s31f6 Link encap:Ethernet HWaddr 64:00:6a:85:1f:c7
inet addr:137.113.118.63 Bcast:137.113.118.255 Mask:255.255.255.0
inet6 addr: fe80::ca7c:5220:5737:2295/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:2468972 errors:0 dropped:0 overruns:0 frame:0
TX packets:669086 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:3154310032 (3.1 GB) TX bytes:111325757 (111.3 MB)
Interrupt:19 Memory:df000000-df020000
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:8561 errors:0 dropped:0 overruns:0 frame:0
TX packets:8561 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:685209 (685.2 KB) TX bytes:685209 (685.2 KB)
wlxc83a35c3f194 Link encap:Ethernet HWaddr c8:3a:35:c3:f1:94
inet addr:192.168.2.131 Bcast:192.168.2.255 Mask:255.255.255.0
inet6 addr: fe80::53e5:ca2c:99e0:b268/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:360 errors:0 dropped:0 overruns:0 frame:0
TX packets:2729 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:57544 (57.5 KB) TX bytes:465564 (465.5 KB)
levy@kern:~$



- This address is the address that the ad-hoc network on the Pi has assigned to *my* computer, via DHCP.
- Since the DHCP config file said to start such addresses at 192.168.2.3, the address of the Pi itself is 192.168.2.2.
- Now we can ssh into the Pi, use it for client/server, etc.

Putting It All Together



Postscript: Other Radios

- Bluetooth: Popular standard for short-distance applications in consumer electronics (keyboard, mouse, headset, phone syncing), but nontrivial to work with in my experience

- ZigBee: Standard for Personal-Area Networks; multiple configurations: typically, serial broadcast/receive "mesh"