# Übersicht IoT Hardware

**Raspberry PI 3B+**

Ist ein Microcomputer 4core CPU, 1GB RAM, 4 USB Ports, LAN, WLAN, Bluetooth und HDMI Anschluss

Verschieden Linux Betriebssystem zB Rasbpian (ist ein Linux, Devian-Clone) gibt’s asl Desktopn Variante oder nur mit shell. Paketmanager und Shell wie bei ubuntu. Es funktionieren eigentlich alle Pakete.

Für das Iot hat der Raspberry GPIO Ports, Ports für den I2C Bus, mit AD Wandler Karten erweiterbar  
Achtung der Raspberry ist genauso wenig echtzeitfähig wie ein Windows Rechner (Linux ist ja ein Multitasking OS). Vorteil: root hat alle Rechte und kann ungehindert auf die Hardware zugreifen (evtl muss ioperm() also input output permission aufgerufen werden, aber dann hat man Zugriff)

### Alternativen zum Raspberry:

Andere Microcomputer, ODROID, Banana etc unterscheiden sich durch CPU, mehr oder weniger RAM, aber im Prinzip gleich

**Microcontroller Arduino und clones**Vorteile: echtzeitfähig, sehr einfache IDE, AD Wandler onboard, günstig, viele Bsp Programme, viele Bauformen  
Nachteile: Erweiterungen müssen per shields dazugebaut werden, kein echtes Threading,

**Microcontroller ESP866 oder ESP32**  
Vorteile: ähnlich Arduino, zusätzlich I2C, UART, SPI, WLAN eingebaut, sehr klein und günstig (wenige €), viele Beispielprogramme.   
+++Es gibt sogar ein RealTime Operating System (freeRTOS für den ESP32 - sind nur ein paar C –Files) dann hat man Echtzeit und threads. Auch mit mikropython als Programmeirschnittstelle geht das.

# Raspberry als IoT Device

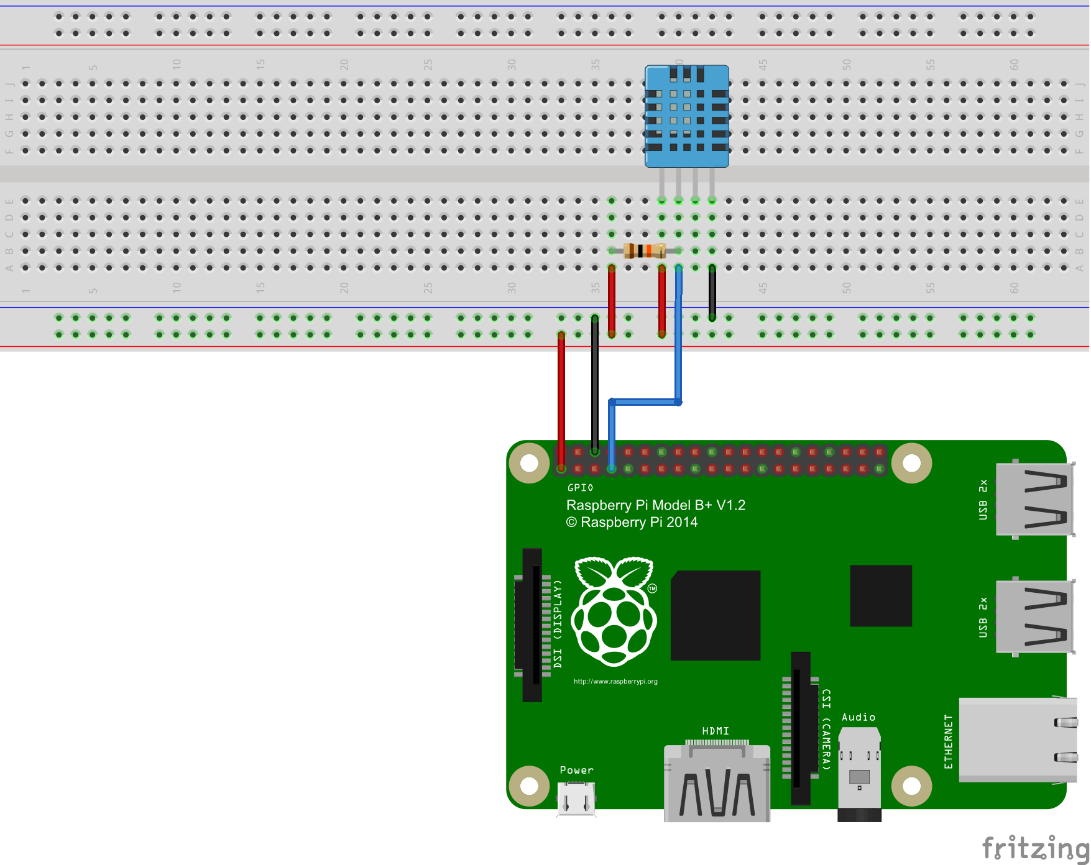
## Raspberry für die Messdatenerfassung verwenden

Der Raspberry hat eine Menge GPIO-Ports (General Purpose Input Outputs), die als Pin Anschlüsse auf der Platine verwendet werden können. An diese Pins kann man Sensoren oder Aktoren anhängen. Die GPIOs können als Eingänge oder als Ausgänge verwendet werden  
als Eingänge: wenn ein Sensor dranhängt und eine Spannung über 3V erzeugt liegt am Eingang ein true, unter 0,3V ein false.  
als Ausgang: wenn man einen Port auf true setzt wird unter 5V Spannung gesetzt, bei false wird die Spannung abgeschaltet.

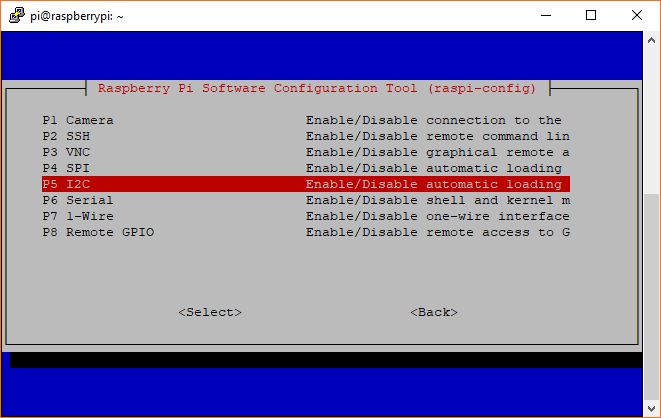
Über Pulsweitenmesung können auch andere Werte als true und false gesendet werden:

Prinzip: Sender der an einem Pin hängt will den Wert 321 senden: dann setzt er die Leitung von 0 auf 5V und hält sie 321 ms lang auf 5V, danach wieder auf 0. Der Raspberry misst die Zeit in der die Leitung auf 5V war und hat so die Info. Oder es werden eine Sekunde lang 321 Takte gesendet und der Raspberry zählt die steigenden Flanken, dh jeweil 1/642ms Sekunden lang ein high dann 1/642ms lang ein low und das 321 mal.

Beispiel Temperaturmessung:



Der blaue Sensor ist ein sogenannter DHT11, der Temperatur und Luftfeuchtigkeit messen kann (ist zwar langsam, ca 1 Wert/Sekunde, aber recht genau und sendet seine Werte per I2C Bus an einen GPIO Port)  
I2C muss aktiv sein, dass stellt man im Betriebssystem mittels sudo raspi-config ein:

[](https://camo.githubusercontent.com/e505cd3bf2ad5462be044d61d9e0706211393472/68747470733a2f2f63646e2e737061726b66756e2e636f6d2f6173736574732f6c6561726e5f7475746f7269616c732f342f342f392f6932632d6d656e75322e706e67)

IoT Programm für den Raspberry kann man mit C und C++, Java, oder python schreiben

Für C und Java braucht man die sogenannte wiringpi library (also die Pi Verkabelungsbibliothek) und für Java zusätzlich noch pi4J

Wirklich einfach gehts aber mit python (dh python mit apt-get installieren)  
Außerdem gibt es für fast alle Sensoren fertige python Bibliotheken zB von Adafruit (von denen ist auch der DHT11 und der neuere DHT22)

### Adafruit Bibliothek für die Sensoren installieren:

*sudo git clone https://github.com/adafruit/Adafruit\_Python\_DHT.git && cd A*dafruit\_Python\_DHT

sudo python setup.py install

Jetzt kann man den Sensor schon testen (hier ist es ein DHT 22 am GPIO Port 4)

./examples/AdafruitDHT.py 22 4

oder (am besten im Ordner examples) ein eigenes python script erstellen:

#!/usr/bin/python

import sys

import Adafruit\_DHT

#Sensortyp und GPIO festlegen

sensor = Adafruit\_DHT.DHT22

gpio = 4

# Daten auslesen

humidity, temperature = Adafruit\_DHT.read\_retry(sensor, gpio)

if humidity is not None and temperature is not None:

print('Temp={0:0.1f}\* Humidity={1:0.1f}%'.format(temperature, humidity))

else:

print('Failed to get reading. Try again!')

sys.exit(1)

1. Erstellen: nano DHT11.py
2. Rechte fürs ausführen ändern: sudo chmod ug+x DHT11.py
3. Ausführen mit ./DHT11.py oder mit sudo python DHT11.py (dann gehts auch ohne vorheriges chmod)

## Python und MQTT

Wirklich nett wird das Ganze weil man mit python auch sehr einfach das MQTT Broker Konzept verwenden kann. Auch hier wird wie bei den anderen Programmiersprachen die paho Bibliothek verwendet. Die Messwerte sollen an einen MQTT Broker gesendet (gepublished) werden

Man benötigt das passende mqtt modul für python

sudo apt-get install python-pip

sudo pip install paho-mqtt

Das folgende Beispielprogramm liest Temperatur und Luftfeuchte von einem DHT22 ein und sendet die Messwerte zum Broker (iot.eclipse.org ist ein freier Broker). Um zu zeigen wie GPIOs geschaltet werden, wird eine LED am Port 18 eingeschaltet, falls die Luftfeuchte über 50% liegt

#!/usr/bin/python

# -\*- coding: utf-8 -\*-

import sys

import time

import Adafruit\_DHT

import RPi.GPIO as GPIO

import paho.mqtt.client as mqtt

GPIO.setmode(GPIO.BCM)   
#wenn man GPIO.BOARD angibt muss man die Pin-Nummern nehmen

GPIO.setup(18, GPIO.OUT) #da hängt eine LED dran

publisher = mqtt.Client()

broker="iot.eclipse.org"

publisher.connect(broker)

#Sensortyp und GPIO festlegen

sensor = Adafruit\_DHT.DHT22

gpio = 4

i=20 #testhalber 20 Messwerte auslesen und zum Broker senden

while i>0:

humidity, temperature = Adafruit\_DHT.read\_retry(sensor, gpio)

if humidity is not None and temperature is not None:

print('Temp={0:0.1f}\* Humidity={1:0.1f}%'.format(temperature, humidity))

topic="test/humidity"

s="%.1f°C %.1f%%rF" % (temperature,humidity)

publisher.publish(topic, payload=s, qos=1, retain=False)

if humidity>70:

GPIO.output(18, GPIO.HIGH)

print("Luftfeuchte erhöht")

else:

GPIO.output(18, GPIO.LOW)

print("Luftfeuchte ist ok")

else:

print('Failed to get reading. Try again!')

sys.exit(1)

time.sleep(1)

i=i-1

GPIO.cleanup()

publisher.disconnect()

Das folgende Beispiel empfängt vom Broker (der am Raspberry mit der Adresse 192.168.1.205 läuft) ein „on“ oder „off“ zum Topic „Lampe“, bei on wird eine LED am Port 18 eingeschaltet bei off wird sie wieder abgeschaltet. Das Programm arbeitet immerhin mit einer callback für den LED Steuerbefehl. QoS und Persistenz der Verbindung sind nicht eingestellt, es soll nur das Prinzip gezeigt werden.

#!/usr/bin/python

# -\*- coding: utf-8 -\*-

import sys

import time

import RPi.GPIO as GPIO

import paho.mqtt.client as mqtt

GPIO.setmode(GPIO.BCM) #wenn man GPIO.BOARD angibt muss man die Pin-Nummern nehmen

#GPIO.cleanup()

GPIO.setup(18, GPIO.OUT) #da hängt die LED dran

#define callback

def on\_message(subscriber, userdata, message):

time.sleep(1)

s=str(message.payload.decode("utf-8"))

print("received message =",s)

if s=="on":

GPIO.output(18, GPIO.HIGH)

elif s=="off":

GPIO.output(18, GPIO.LOW)

subscriber = mqtt.Client()

broker="192.168.1.205"

subscriber.on\_message=on\_message

subscriber.connect(broker)

print("Verbindung zum Broker ",broker)

subscriber.loop\_start() #startet den eventhandler fürs on\_message

subscriber.subscribe("Lampe")

while 1: #nicht sehr schön, mit polling bis zum Strg-C

time.sleep(0.5)

subscriber.disconnect() #disconnect

subscriber.loop\_stop() #stop loop

GPIO.cleanup()

## Der Raspberry als Broker

Man muss nicht einen externen Broker wie eclipse.org verwenden, man kann den Brokerservice auch direkt am Raspberry installieren:

sudo apt-get install mosquitto mosquitto-clients

den server starten, am besten im verbose mode

mosquitto –v

falls man weiterarbeiten will strg-z stoppen und mit bg in den Hintergrund legen

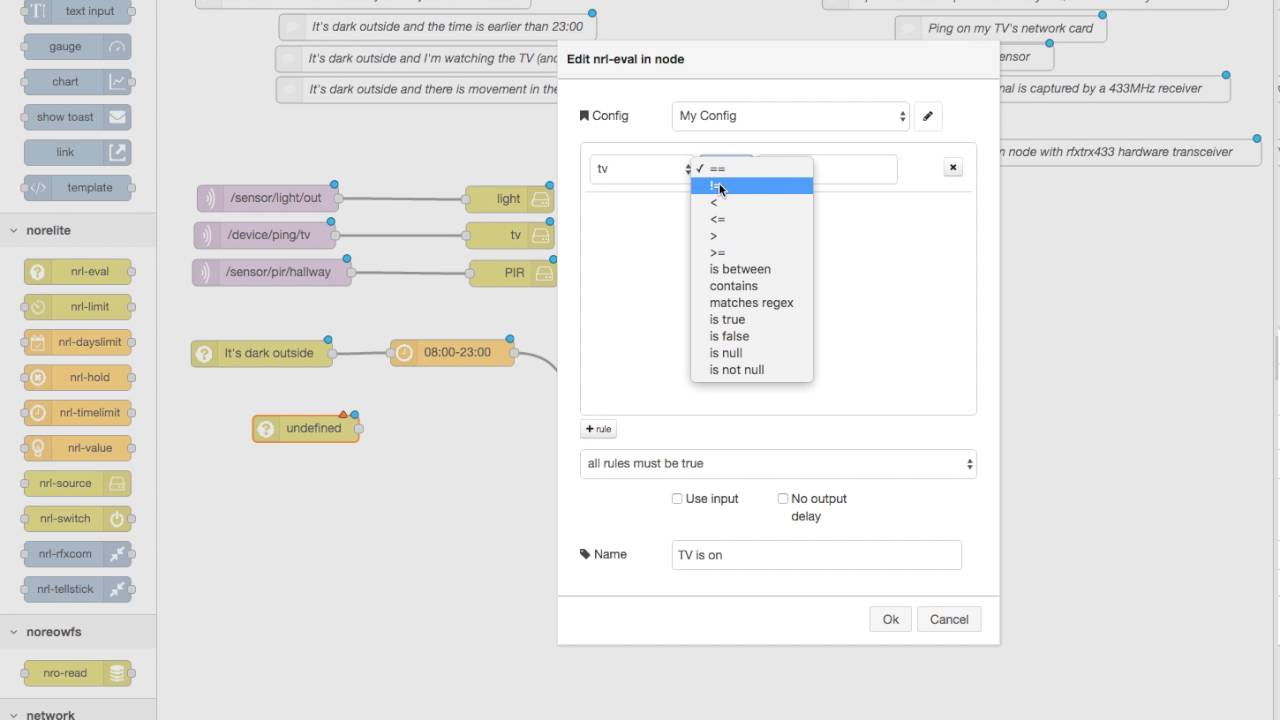
oder gleich mit & im Hintergrund starten

mosquitto –v &

den Broker kann man sogar mittels account Listen absichern und die Übertragungen per SSL verschlüsseln.

## Node-Red

Node-Red ist eine grafische Programmierumgebung für das IoT. Es unterstützt alle möglichen Protokolle wie TCP, websockets, mqtt, hat alle möglichen parser, formatter, und Zugriff zu Datenbanken zB google firestore. Mit dem Dashboard kann man sehr effiziente GUIs mit buttons, slidern, charts… bauen, auf die über die URL des Gerätes, auf dem node-red rennt, per http oder https zugegriffen werden kann.



# Fazit

Der Raspberry mit seinen Ports, einem lokalen mqtt Broker, lokalen node-red und eigener Datenbank kann eine vollständige IoT Lösung ermöglichen auf die man per App zugreifen kann.

zB könnte man damit eine komplette Heizungssteuerung basteln, in der alle Temperaturen gemessen und Ventile geschalten werden. Der User kann per HandyApp über WLAN zugreifen, Temperaturwerte einsehen und Teile aus/und einschalten. Software ist komplett frei, Security kann man einbauen. Kosten Raspberry + Gehäuse + Netzteil + SD Karte <50Euro