Developing an IoT System

Uli Raich (uli.raich@gmail.com)

Two lectures on Hardware and Software for
the Internet of Things

Lecture 1: Introduction to the hardware
and development environment and access to the “things”

Presented online at the African Internet Summit Johannesburg 2023

Introduce myself

- Dr. Ernst Ulrich Raich
  short: Uli
- PhD in Physics (not computer science!)
  but 4 years of IT studies without
  diploma.
- Married, 3 adult children, 3 grand children
- Staff member of CERN for 35 years
  now retired
- Teaching (short microprocessor courses) since 1980
- Guest lecturer at the University of Cape Coast 2017
  Set up a microprocessor lab and gave a full semester course on embedded systems
- Based on Raspberry Pi and the C language

About this tutorial

This is not just a talk, but the tutorial aims at demonstrating with live programs all the concepts that are explained.
Some of you may want to re-create the demos after the tutorial. Therefore, I will give you a list of all devices (cost ~ 25 Euros) needed to do so, in a later slide.
All the course material:

- The firmware running on the micro-controller
- The slides
- The driver for the temperature and humidity sensor
- The demo programs

are available on github: https://github.com/uraich/AIS-2022_IoT_Tutorial
Questions are welcome (also in French or German!)

The Microprocessor Revolution

From all technical advances during the last 50 years the development of micro-processors certainly had the biggest effect on our daily life.
When I was a student a computer looked like this:

It filled a whole room and the cost was several 100 k$

Minicomputers

When I was a doctoral student this was the computer I worked with:

- It fitted into a rack
- Cost: several 10 k$
- Typical memory size: 128 kBytes
- Hard disk: 600 Mbytes (which was huge!)
- Black and White serial terminal for programming
- On such a machine the Unix OS was developed

The first Microprocessors

… and then came the Microprocessor
The first one I played with was the Motorola MC6800, 8bit microprocessor

- Cost at introduction: ~ 500 $ US
- Clock frequency: 1 MHz
- Only external memory (my first system had 128 Bytes)
- “OS” in EPROM (ca. 2 kBytes)
- External parallel and serial interface
- Programs were stored on audio tape
- Total cost of a system: ~ 2000 $US
- Programmed in binary machine code entered through a keypad
-

… and today? for 10 Euros?

Hardware Specifications	
Chipset	ESPRESSIF-WROVER-B 240MHz Xtensa® single-/dual-core 32-bit LX6 microprocessor
FLASH	QSPI flash 4MB 16MB / PSRAM 8MB
SRAM	520 kB SRAM
Button	reset
Modular interface	UART、SPI、SDIO、I2C、LED PWM、TV PWM、I2S、IRGPIO、ADC、DACLNA pre-amplifier
On-board clock	40MHz crystal oscillator
Working voltage	2.7V-3.6V
Working current	About 30mA
Working temperature range	-40°C ~ +85°C
Size&weight	40.27mm*31.07mm*8.13mm(6.42g)

ESP32 network connection

Wi-Fi	
Standard	FCC/CE-RED/IC/TELEC/KCC/SRRC/NCC(esp32 chip)
Protocol	802.11 b/g/n(802.11n，speed up to150Mbps)A-MPDU and A-MSDU polymerization，support 0.4μS Protection interval
Frequency range	2.4GHz~2.5GHz(2400M~2483.5M)
Transmit Power	22dBm
Communication distance	300m
Bluetooth	
Protocol	meet bluetooth v4.2BR/EDR and BLE standard
Radio frequency	with -97dBm sensitivity NZIF receiver Class-1,Class-2&Class-3 emitter AFH
Audio frequency	CVSD&SBC audio frequency

Where do we find micro-controllers?

Answer: Everywhere!

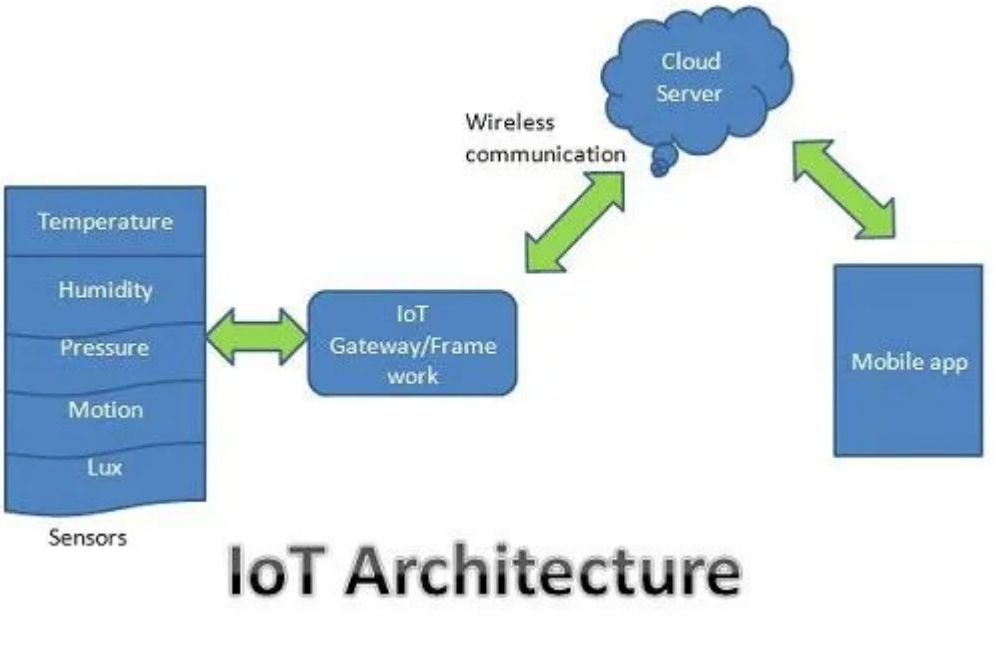
- Car (many of them!)
- Coffee machine
- TV set, radio
- Watch
- Hand phone

What is IoT?

Wikipedia:

The **Internet of Things (IoT)** describes physical objects (or groups of such objects) with sensors, processing ability, software and other technologies that connect and exchange data with other devices and systems over the internet or any other communication networks

IoT system architecture



What is an IoT system composed of?

We need:

- Sensors and actuators
- A processor powerful enough to run the network protocol layers
- A network interface: either Ethernet of WiFi
- Interfaces to the sensors and actuators*:*
  - General Purpose I/O (GPIO) lines
  - I2C, I2S, SPI, serial interfaces
- The cost for the controller should be in a good relation with the cost of *the things*

Where to buy?

Local electronics shop (if it exists!):
Advantages: advice, guaranty, quick delivery
Disadvantages: high cost (device cost in Sénégal was more than 10 times the price I paid)

Amazon or similar online shop:
Advantages: Delivery may be relatively quick, price better than local electronics shop
Disadvantages: Devices are often only available on market place with vendors from China, resulting in the same disadvantages as ordering from China directly.

Aliexpress or other online shop in China
Advantages: Very low price
Disadvantages: Long delivery time or high shipping cost. Sometimes devices are faulty and even though the problem is known, the companies continue selling them. No documentation. Most of the time you can find the docs on the Internet though.

University courses on IoT in Africa

When asking African colleagues about micro-controller courses (embedded systems or IoT) in Africa I often get the answer:

- We have no such course, the micro-processor lab would be too expensive.

I wanted to find out and I therefore prepared such a course for the University of Cape Coast, Ghana.
The cost for 1 experimental station was ~ 50 $ US. If you run the course for 5 years this gets you to a cost of 10 $ US per student and year! This is cheap enough to allow students to take the equipment home for experimentation!
The real problem is, that there are too few lecturers capable of preparing and running such a course and preparation of exercises and solutions is very labor intensive. (I took 6 months for the preparation)
Much of the following material has been extracted from the UCC IoT course.

Documentation of UCC IoT course

The IoT course was given at the University of Cape Coast, Ghana, for the first time at the beginning of 2021
Everything is documented on a TWiki server located in Accra
It consists of

- Hardware and software documentation with plenty of links to relevant WEB pages
- Explanations for a range of experiments (exercises)
- Lecture slides (of these lectures)
- Exercise sheets
- Solutions to the exercises that can be downloaded from a github repository

Efforts are under way at the Université Cheik Anta Diop, Dakar, Sénégal, to provide a similar course with all documentation written in French, which you can find on the above Twiki server.

Processors for IoT

There is a huge selection of different chips:

- STM32: ARM Cortex MCU. This is a whole family of chips with different performance and price.
- Raspberry Pi: The RPi is more like a little computer. It has a quad core micro-controller capable of running an ARM based Linux OS. All you need to make this a full computer is a keyboard, a mouse and a screen.
- ESP32 based boards: Dual core processor, SRAM and flash on chip. WiFi and BlueTooth implemented on chip.

Which one should I use?

The Raspberry Pi is a small computer powerful enough to run a full blown Linux operating system:

- A quad core 1.2 GHz Broadcom 64 bit ARM CPU
- 1 Gbytes of Ram
- Ethernet and wireless networks
- 4 USB2 ports
- Micro SD connector
- 40 pin extended GPIO connector with
  - GPIO pins
  - SPI and I2C bus interface
- Cost ~ 80-100 US $
- No ADC

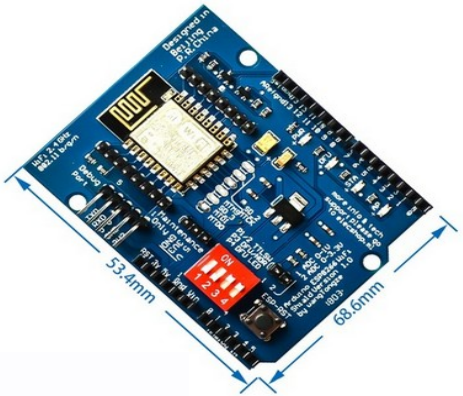
Arduino + WiFi shield

The ATmega328 chip found on the Uno has the following amounts of memory:

```
Flash  32k bytes (of which .5k is used for the bootloader)
SRAM   2k bytes
EEPROM 1k byte
```

The ATmega2560 in the Mega2560 has larger memory space :

```
Flash  256k bytes (of which 8k is used for the bootloader)
SRAM   8k bytes
EEPROM 4k byte
```



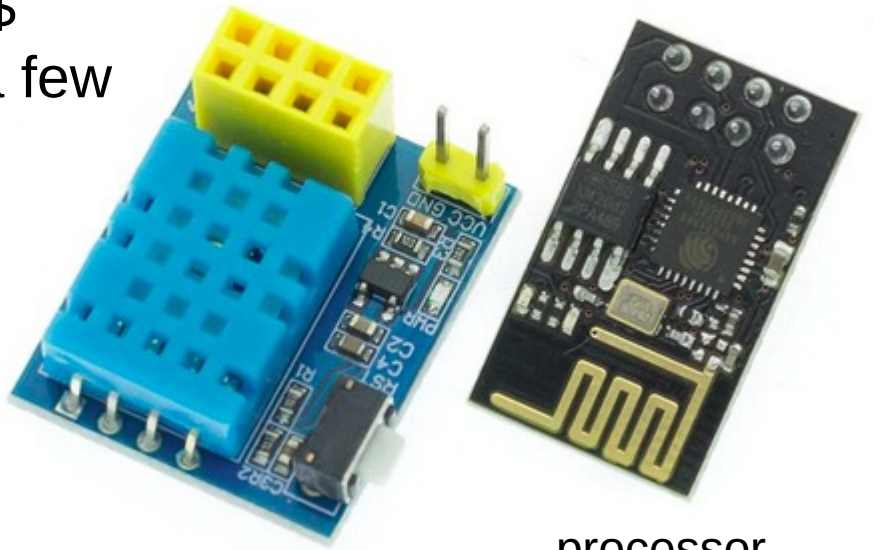
Cost: ~12-15 US $
Popular because of simple C++ IDE

The low end

If you just need an Internet connected temperature sensor, the ESP01 will do!
Cost of processor and sensor: < 2 US $
In addition ou need a programmer for a few cents.

programmer

Temperature sensor



processor

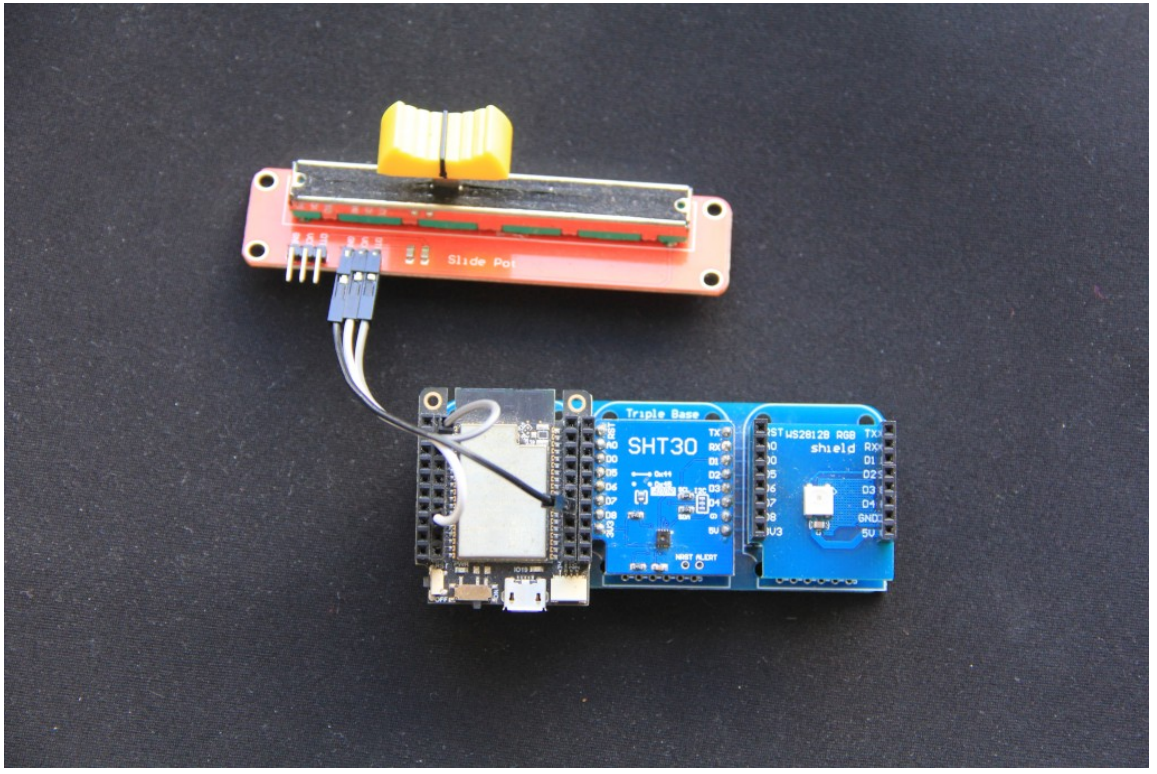
Sensors and Actuators

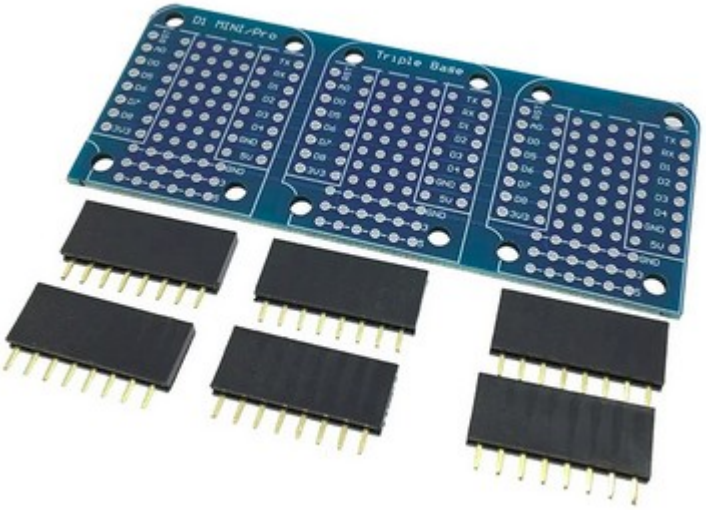
For the IoT course I selected the
WeMos D1 mini series of boards.
It provides a selection of CPU boards

- ESP8266 CPU
- ESP32 with or without SPIRAM

and it comes with a large selection of sensor/actuator boards
The sensor boards are connected to the CPU through a simple plug and play system

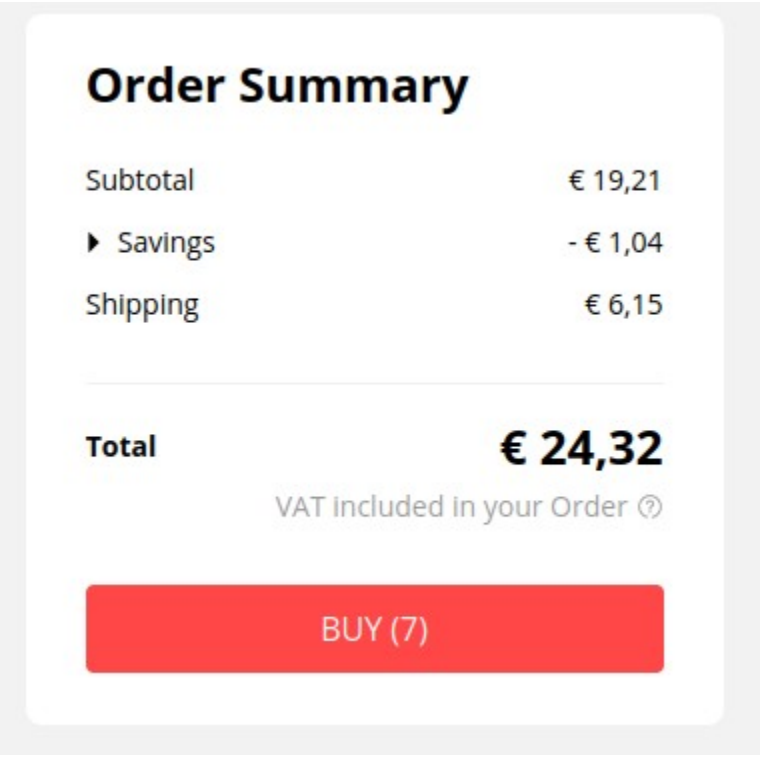
Connection between CPU and sensors



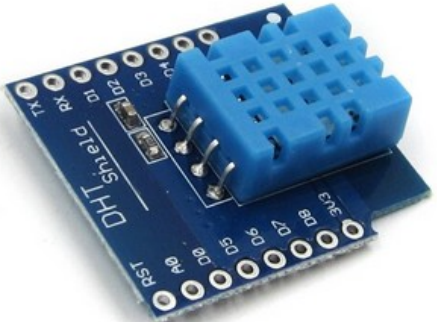


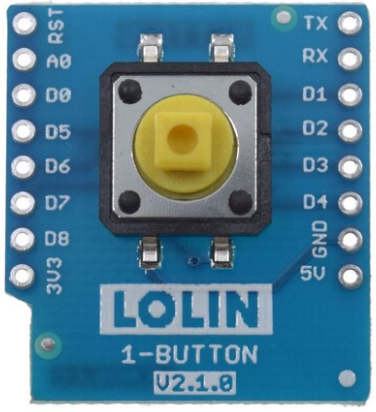
Cost of devices shown in this tutorial

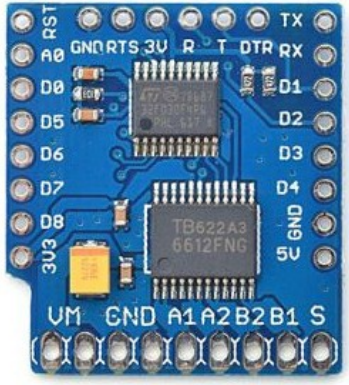
- The WeMos T7 V1.5 CPU board
- The triple base with connectors
- The SHT30 temperature and humidity sensor
- The rgb LED ring
- The micro USB cable for communication with the PC
- Linear potentiometer



… a big number of sensor shields







WeMos D1 sensors

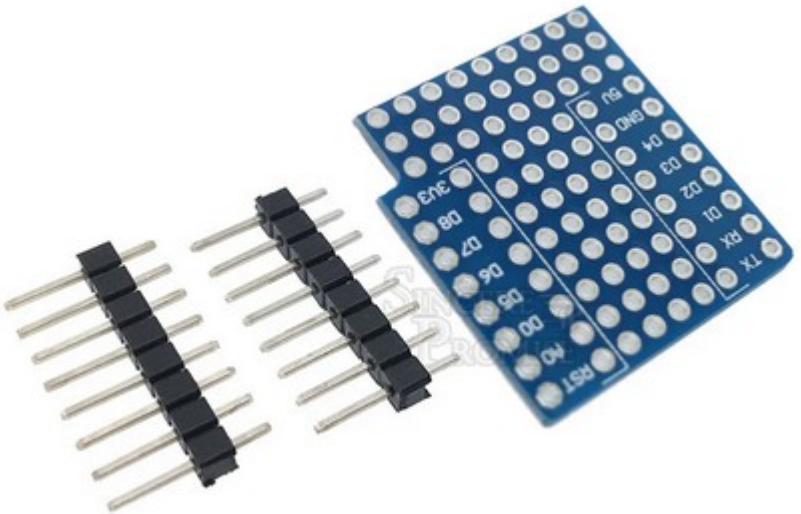
Here is an incomplete list of sensor and actuator modules:

- user LED on CPU module
- Simple mechanical push button
- WS2812 single rgb LED or LED ring with 7 LEDs
- IR sender and receiver
- Passive buzzer
- PIR (Passive Infra Red) sensor
- DS18b20 digital temperature sensor
- BMP180 barometric pressure sensor
- SHT30 I2C temperature and humidity sensor
- DHT11 temperature and humidity sensor
- Ambient light detector
- RTC and data logger with SD card interface
- Relay module
- DC motor controller
- and so on ...

Prototype Boards

If this is not enough, you can mount small sensor boards onto a WeMos D1 mini prototype board and interface it to your ESP32 CPU.

We will see an example later.



How to develop code

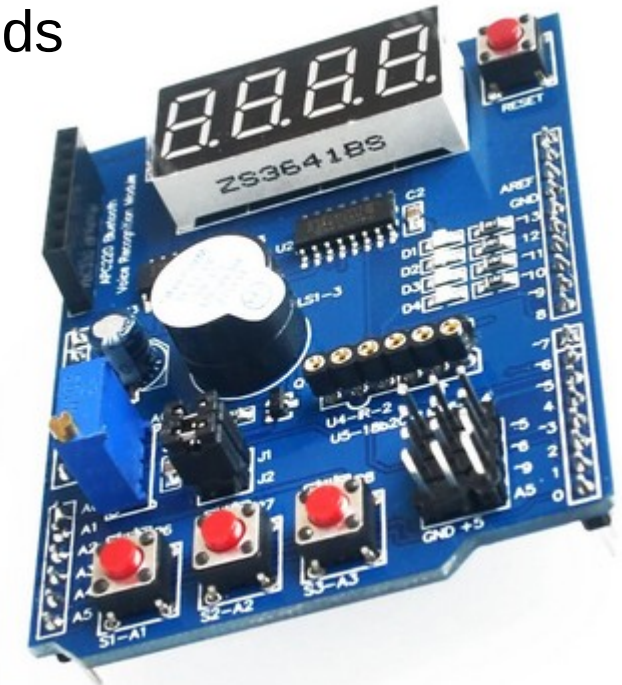
The Raspberry Pi is a bit different than the others: Here the compiler can run natively on the Pi. If you know Linux, then you know how to develop on the Pi.

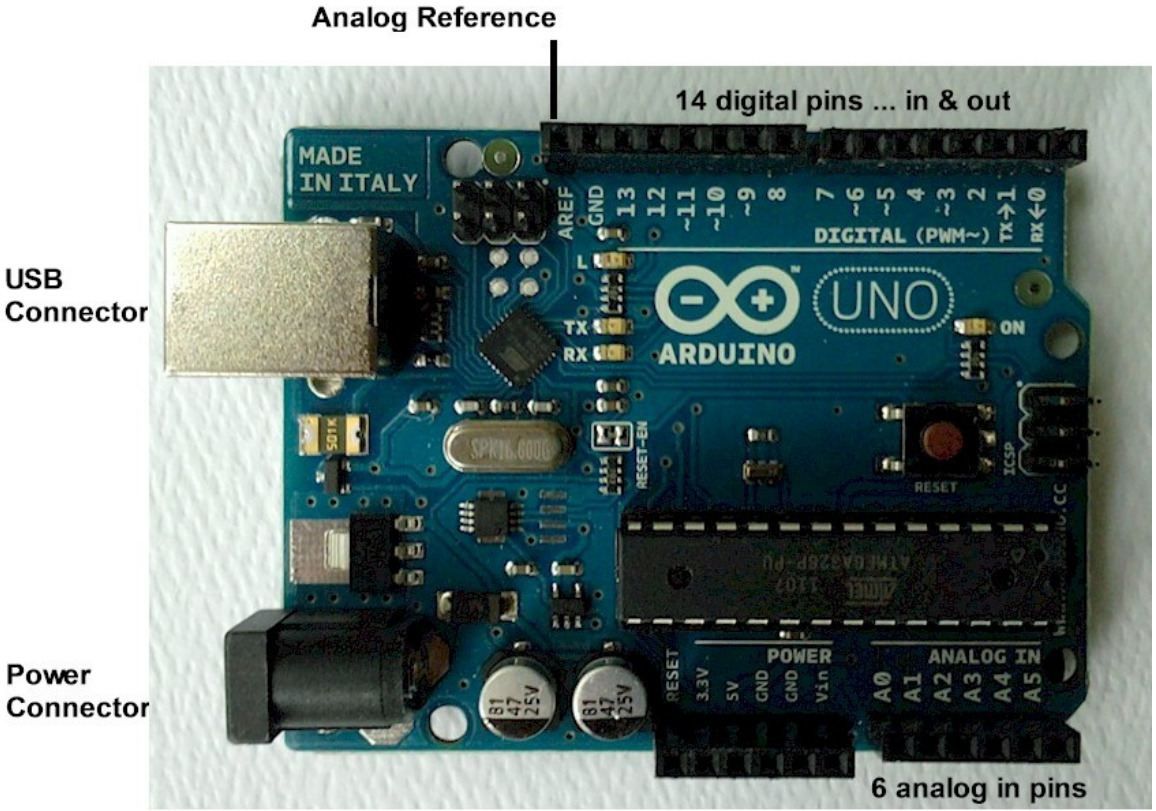
All other CPU suppliers provide a cross-development environment with their chips. The ESP32 CPU is described in a manual of more then 1000 pages! The library to access all the hardware functionality is huge.

Espressif (the company behind the ESP32) supplies *esp-idf*, a build system based on *cmake* and the hardware access library. Programming is done in C or C++.
The learning curve to start your first application is very steep.
The same is true for the STM32 chips.

The Arduino

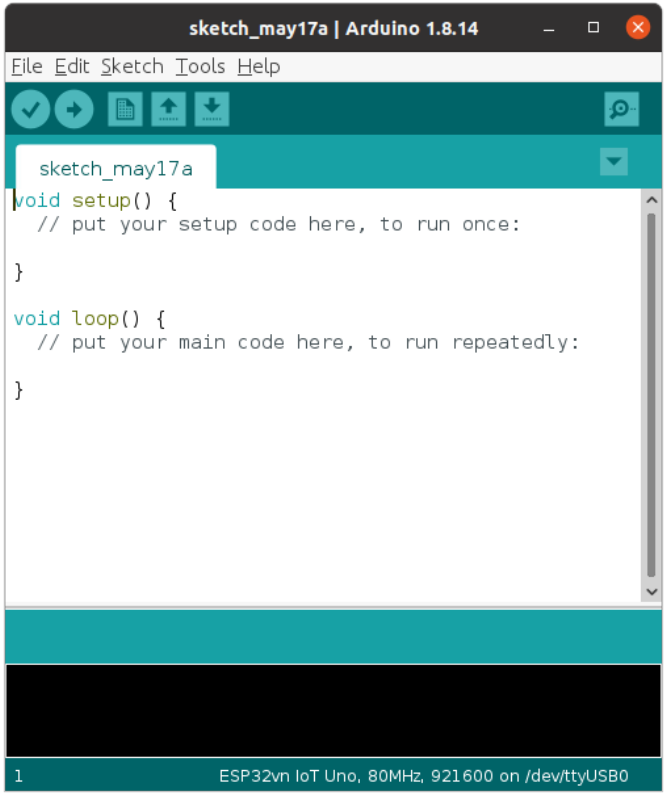
Open source hardware
Shields





Arduino IDE

Designed for the beginner and hobbyist market
Originally provided for the AVR processors but now also available for ESP8266, ESP32 or the STM32 processors
Uses a C++ like language
Simplifies program development a lot
Comes with a huge collections of examples and libraries
Upload and flash programming is integrated



MicroPython

MicroPython is stripped down Python interpreter based on Python 3.5 and dedicated to micro-controllers. It is provided in source format in form of a github repository
Excellent documentation is provided and if this is not enough you can have a look at the source code
A very active user forum helps in case of problems
However:
It needs quite a bit of infrastructure on the PC to be able to cross-compile Micropython, which depends on

- The xtensa-esp32-elf-gcc cross compiler
- The espidf libraries
- esptool to erase and program the ESP32 flash
- ampy of ftpd to transfer files to the ESP32 file system

Micropython exists for a series of micro-controllers, the ESP32 being a popular port.
The interpreter is written in C

Bringing up the system

- MicroPython is built using a Makefile
- It uses Espressif's build system based on cmake
- esptool is used to burn the firmware into the ESP32 flash (esptool is accessed by the Micropython Makefile such that
  make deploy will flash the firmware)
- MicroPython REPL (**R**ead, **E**valuate **P**rint **L**oop) can be accessed through a USB to serial adapter combined with a virtual terminal program like minicom or gtkterm
- MicroPython can also be accessed over the network
- A working custom binary of MicroPython, built for the UCC course, is available on github
- Programs are written on the PC and uploaded to the MicroPython file system before being executed
- We use the *thonny* IDE to simply the procedure

Learning Python

In the first lecture and exercise session we give an introduction to Python. Basic knowledge of Python is a major advantage. If you are totally new to the language then go through the Python tutorial first.

- We use the Python interactive shell REPL to get acquainted with Python
- We write our first simple Python scripts using *thonny*
- Since none of the example programs depends on specific hardware we can run the programs on the PC or on the ESP32
- The exercise sheet is available on the Twiki and as Libreoffice document

How do we talk to the ESP32?

The CPU board has a Micro USB connector and a USB to serial converter.
We connect it to the PC with the same micro USB cable you use on your smart phone for charging and data transfer.
We can use a serial terminal emulator to communicate with the ESP32 or the *thonny* IDE for communication and program development

Communication tools

Serial communication:

- minicom
- gtkterm
- rshell

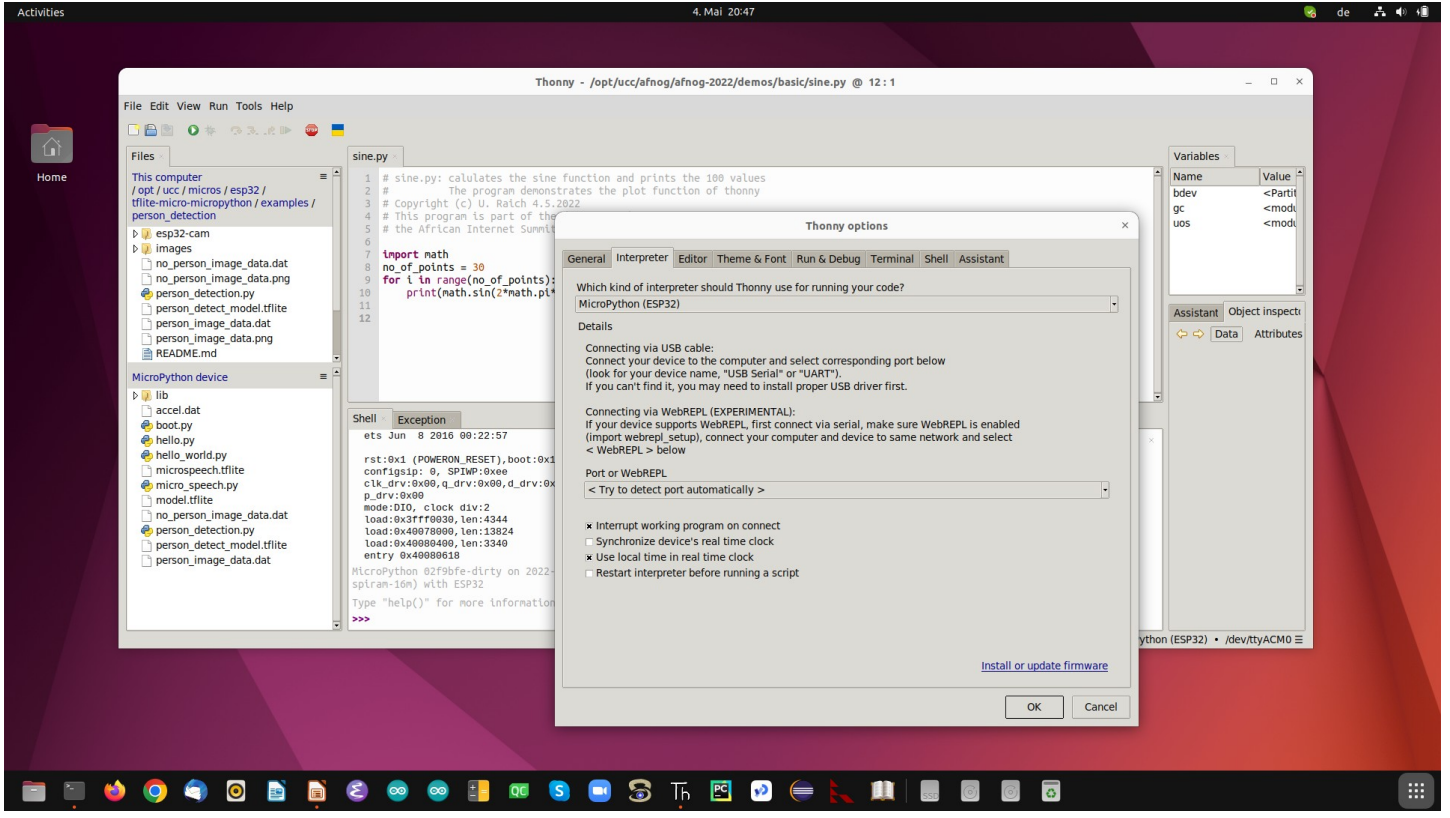
File transfers:

- ampy
- rshell
- ftp, needs the ftp server to be running on the ESP32

IDE:

- thonny
- pycharm

thonny


Activities
4. Mai 20:47
Thonny - /opt/ucc/afnog/afnog-2022/demos/basic/sine.py @ 12 : 1
File Edit View Run Tools Help
Home
Files
This computer
/ opt / ucc / micros / esp32 /
tflite-micro-micropython / examples /
person_detection
esp32-cam
images
no_person_image_data.dat
no_person_image_data.png
person_detection.py
person_detect_model.tflite
person_image_data.dat
person_image_data.png
README.md
MicroPython device
lib
accel.dat
boot.py
hello.py
hello_world.py
microspeech.tflite
micro_speech.py
model.tflite
no_person_image_data.dat
person_detection.py
person_detect_model.tflite
person_image_data.dat
sine.py
# sine.py: calulates the sine function and prints the 100 values
# The program demonstrates the plot function of thonny
# Copyright (c) U. Raich 4.5.2022
# This program is part of the
# the African Internet Summit
import math
no_of_points = 30
for i in range(no_of_points):
print(math.sin(2*math.pi*
Shell
Exception
ets Jun 8 2016 00:22:57
rst:0x1 (POWERON_RESET),boot:0x1
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x
p_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:4344
load:0x40078000,len:13824
load:0x40080400,len:3340
entry 0x40080618
MicroPython 02f9bfe-dirty on 2022-
spiram-16m) with ESP32
Type "help()" for more information
>>>
Variables
Name
Value
bdev
gc
uos
Assistant
Object inspecti
Data
Attributes
ython (ESP32) • /dev/ttyACM0
Thonny options
General
Interpreter
Editor
Theme & Font
Run & Debug
Terminal
Shell
Assistant
Which kind of interpreter should Thonny use for running your code?
MicroPython (ESP32)
Details
Connecting via USB cable:
Connect your device to the computer and select corresponding port below
(look for your device name, "USB Serial" or "UART").
If you can't find it, you may need to install proper USB driver first.
Connecting via WebREPL (EXPERIMENTAL):
If your device supports WebREPL, first connect via serial, make sure WebREPL is enabled
(import webrepl_setup), connect your computer and device to same network and select
< WebREPL > below
Port or WebREPL
< Try to detect port automatically >
Interrupt working program on connect
Synchronize device's real time clock
Use local time in real time clock
Restart interpreter before running a script
Install or update firmware
OK
Cancel

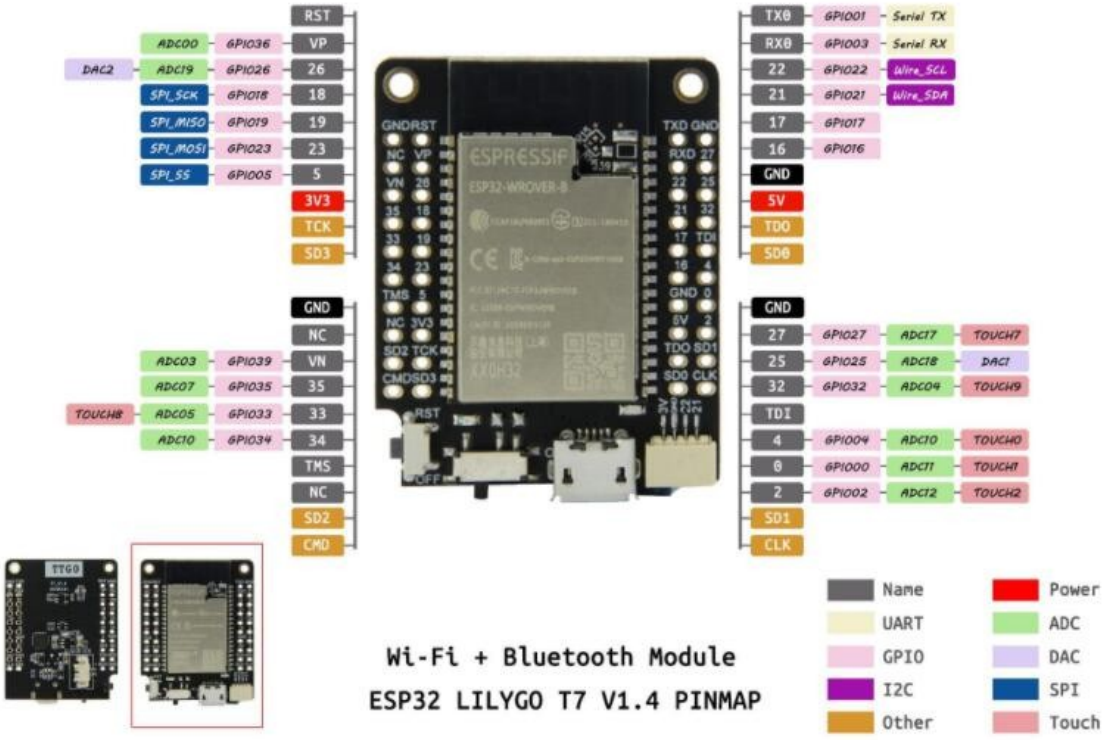
d

Interfacing to the “things”

The ESP32 has a big number of interfaces implemented on the chip:

- GPIO pins
- PWM
- Capacitive touch sensor
- I2C (**I**nter **I**ntegrated **C**ircuit) interface
- SPI (**S**erial **P**eripheral **I**nterface) interface
- Analogue to digital converter
- Digital to analogue converter
- Timer

ESP32 pinout



GPIO

A big number of **G**eneral **P**urpose **I**nput/**O**utput lines
These lines can be programmed as outputs or inputs with or without pull-up resistors
They are used to control:

- LEDs, relays, stepping motors…
- You can implement serial protocols in “bit-banging” mode

or they can be used to read

- status bits, switches

GPIO driver in MicroPython

The GPIO driver in MicroPython makes GPIO access super simple:
We use the class *Pin* in the *machine* module:

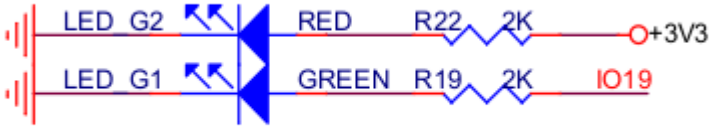
```
from machine import Pin
led = Pin(19,Pin.OUT) # GPIO 19 connects to the user LED
led.on()
```

is all that is needed to control a LED (or a relay)
We even do not need a program to accomplish this.
Let's try!

d

User LED connection

Many ESP32 CPU boards use GPIO 2 to connect to the user LED.
How do I know that the user LED is connected to GPIO 19?
The circuit diagram of the board is available at
https://github.com/LilyGO/TTGO-T7-Demo/blob/master/t7_v1.5.pdf



d

The blink program

With the C programming language the ubiquitous Hello World program has become well known.
It is the most simple program you can possibly write in C,
printing “Hello World!”

It is useful to demonstrate that the programming infrastructure

- Editor, compiler, linker
- Program execution

work well.
The equivalent in the world of embedded systems or IoT is the blinking LED.

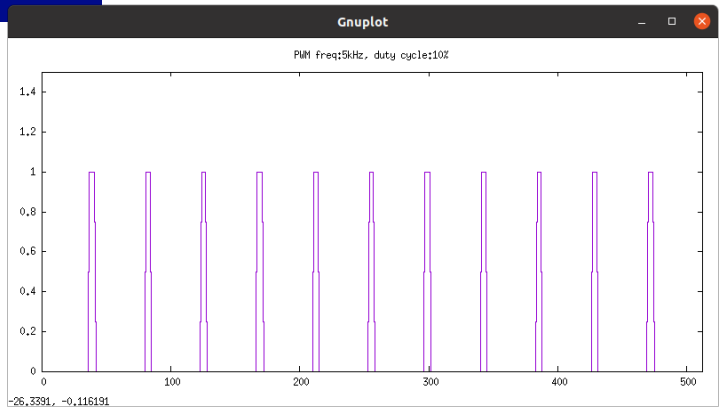
d

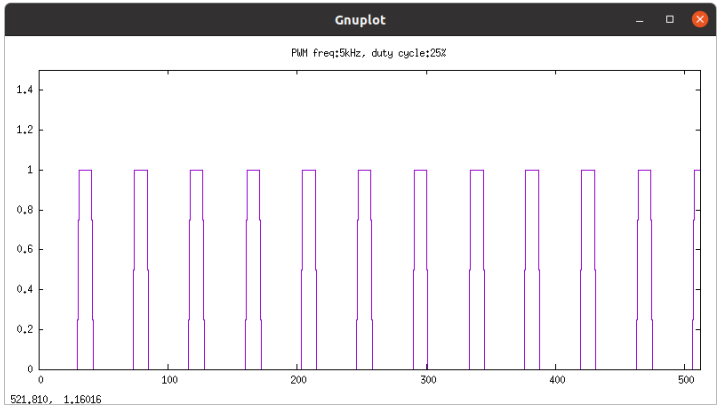
Working blink program

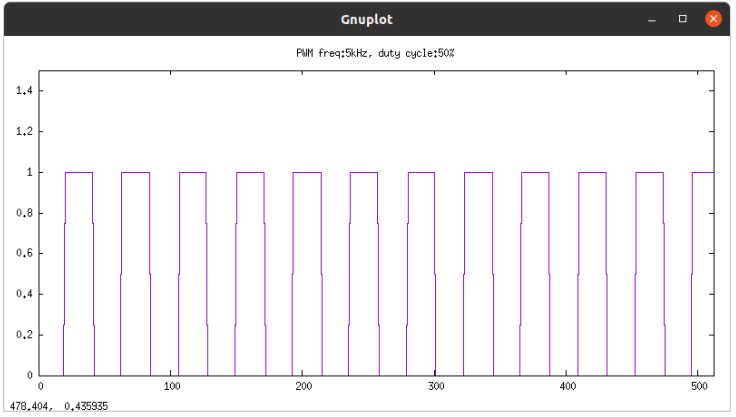
Pulse Width Modulation

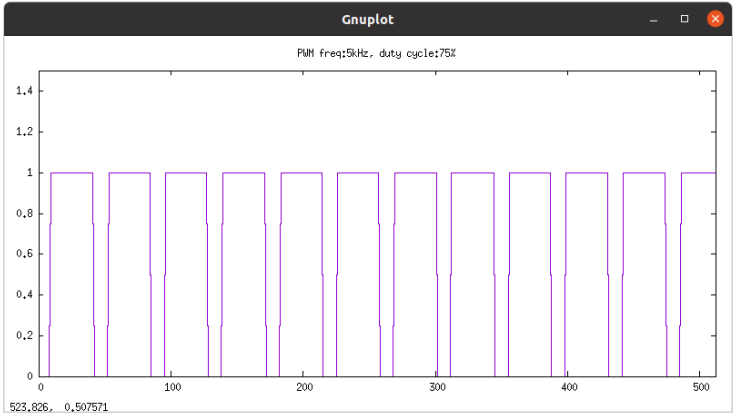
How can we change the LED light intensity with a single digital GPIO line?
The answer is: Pulse Width Modulation (PWM)
Instead of emitting a steady zero or one signal level we emit a frequency (the modulation frequency) and we change the time in which the signal is high during a pulse (duty cycle)
The frequency is high enough and the LED persistence long enough, such that the human eye cannot resolve the frequency.
The average current through the LED is changed and such the light intensity.
PWM is also used to control servo motors

PWM


Gnuplot
PWM freq:5kHz, duty cycle:10%
-26.3391, -0.116191


Gnuplot
PWM freq:5kHz, duty cycle:25%
521.810, 1.16016


Gnuplot
PWM freq:5kHz, duty cycle:50%
478.404, 0.435935

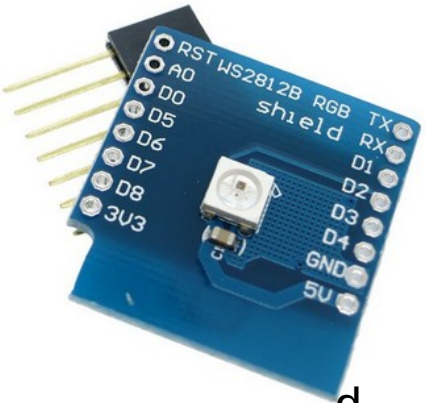

Gnuplot
PWM freq:5kHz, duty cycle:75%
523.826, 0.507571

d

The breathing LED

NeoPixels

The addressable rgb LED of type WS2812 is used in many LED chains
It uses a sophisticated timing sequence to set an individual LED in the chain
We use a simple, 7 LED chain for demonstration purposes
MicroPython provides the NeoPixel driver, which looks after the protocol and its stringent timing requirements
The sensor shield uses GPIO 26 to control the NeoPixels



Unfortunately I killed my LoLin RGB LED card just a few days before this presentation and I had to replace it with a card featuring a single WS2812 LED connected onto GPIO 21.

d

LED ring

Despite my misfortune with the LoLin rgb LED, I can demonstrate an LED ring.

This ring features 24 LEDs. It has just 3 connections:

- 5V Vcc
- GND
- Signal pin

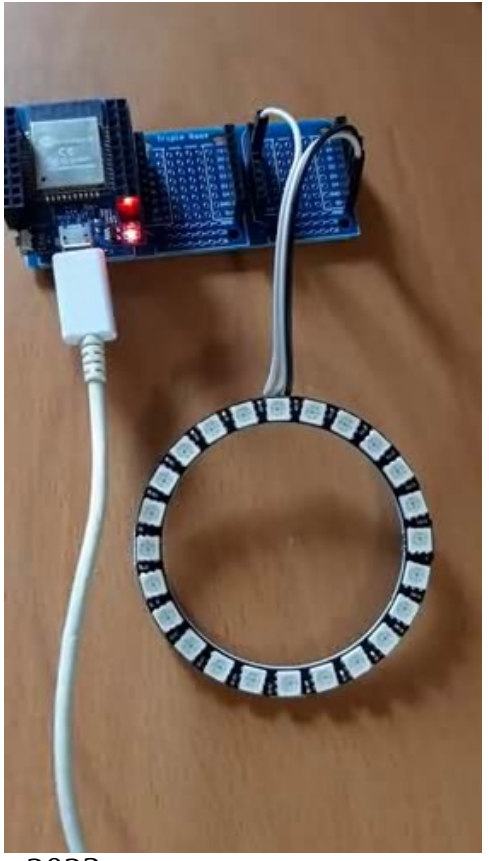
I connect the signal pin to D0 or GPIO 26

d

LED addresses in the ring

We switch on one LED after the other.
This allows us to figure out, which LED corresponds to which address

Of course the light intensity and the color can also be changed

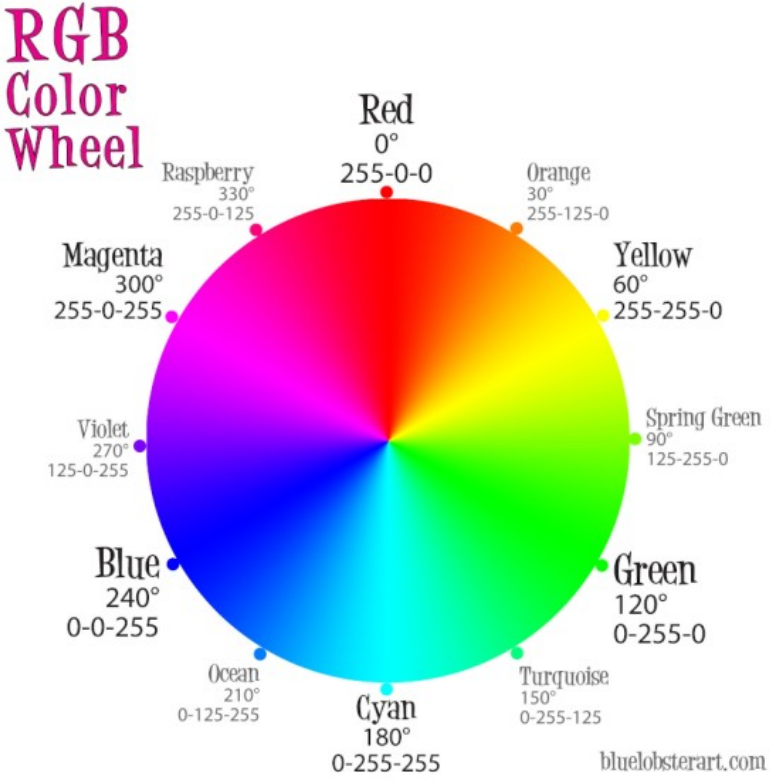


The Color Wheel

The color wheel shows all the colors of the rainbow.

At each step a single of the three color components changes its values

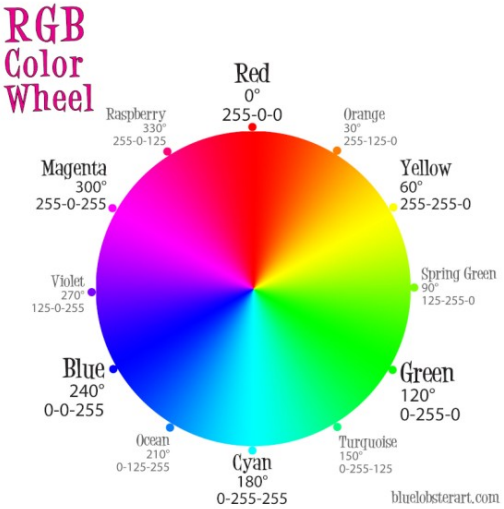
There are 6 sectors and for each sector the varying color component changes

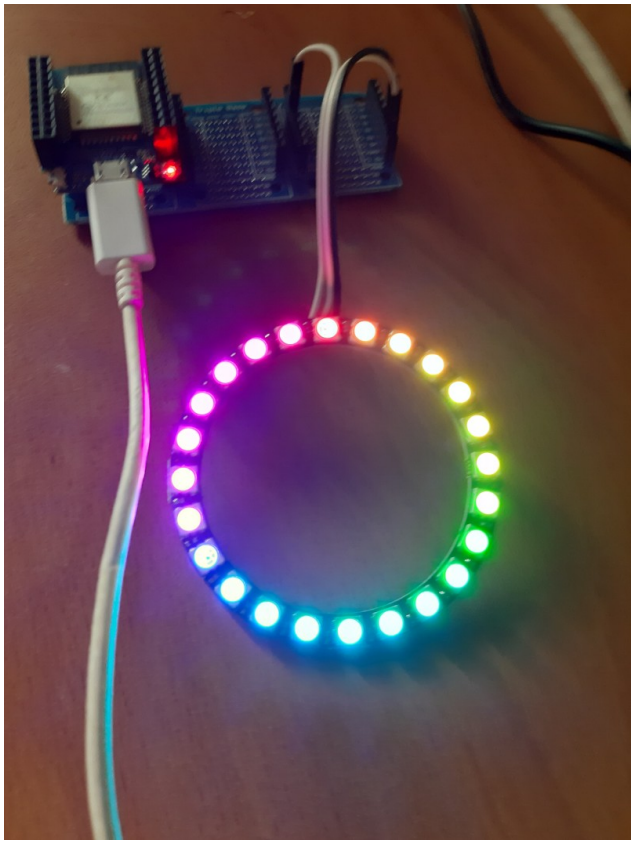


The color wheel on the LED ring

The LED ring has 24 LEDS.

We therefore proceed with a step size of 360° / 25 = 15° on the color wheel





Analogue Signals

The ESP32 has two 12bit Analogue to Digital Converters (ADCs) with a total of max 18 input channels
We will use ADC1 on GPIO 36
The ADC accepts signal levels 0..1V but there is an attenuator in front of it extending the signal range
An attenuation of 11dB provides an input range of ~ 0..3.6V fitting well with the 3.3V Vcc of the CPU board
Again the MicroPython ADC driver provides easy access to the hardware

Linear Potentiometer

To test the ADC on the ESP32 we can connect a 10 kΩ potentiometer
There are 3 pins:

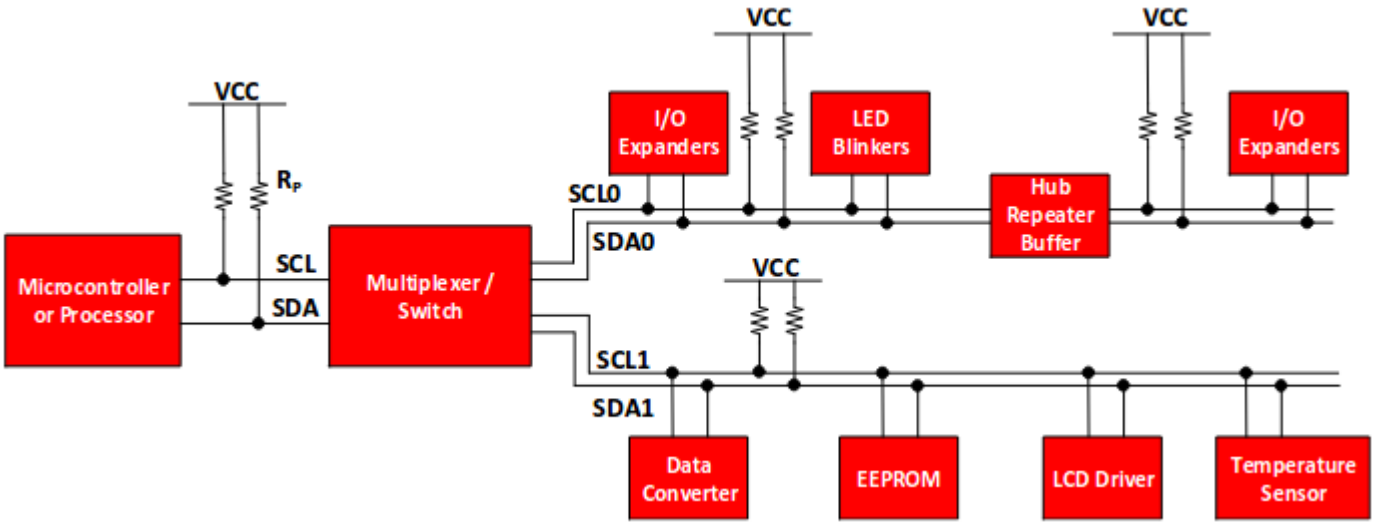
- Vcc → 3.3V
- GND → GND
- OTA(B) → ADC

When you move the slider you get voltage levels 0V on OTA(B)

d

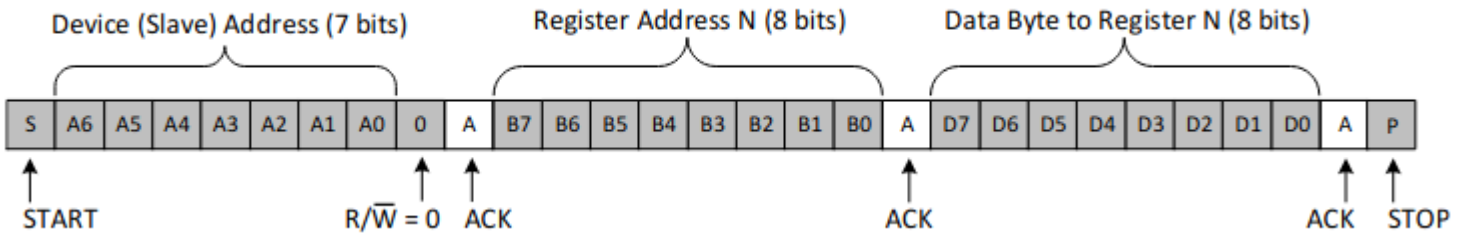
The I2C bus

The Inter-Integrated Circuit (I2C) bus was invented by Philips in the early 1980. It is used by many sensors to communicate their data to the CPU.

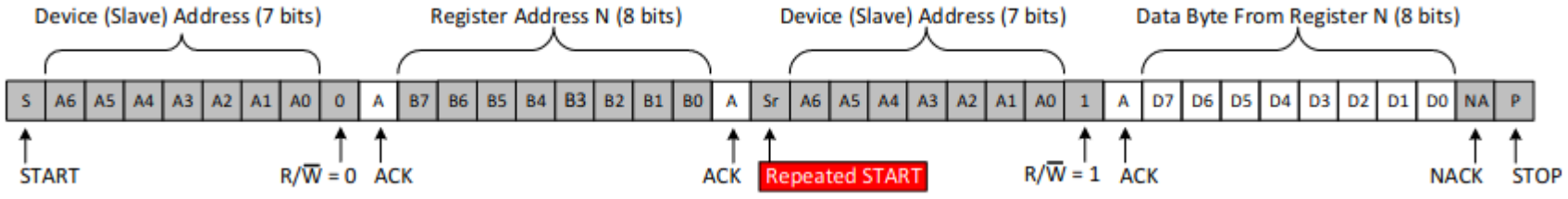


The I2C protocol

Write to the bus

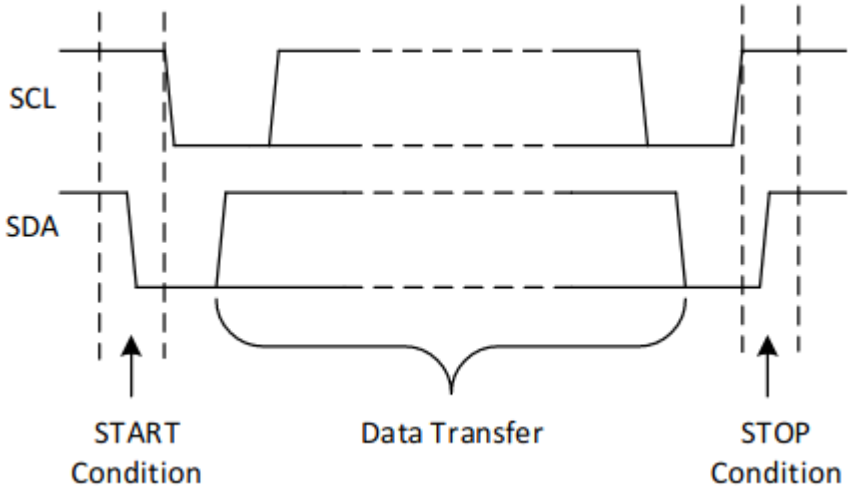


Read from the bus

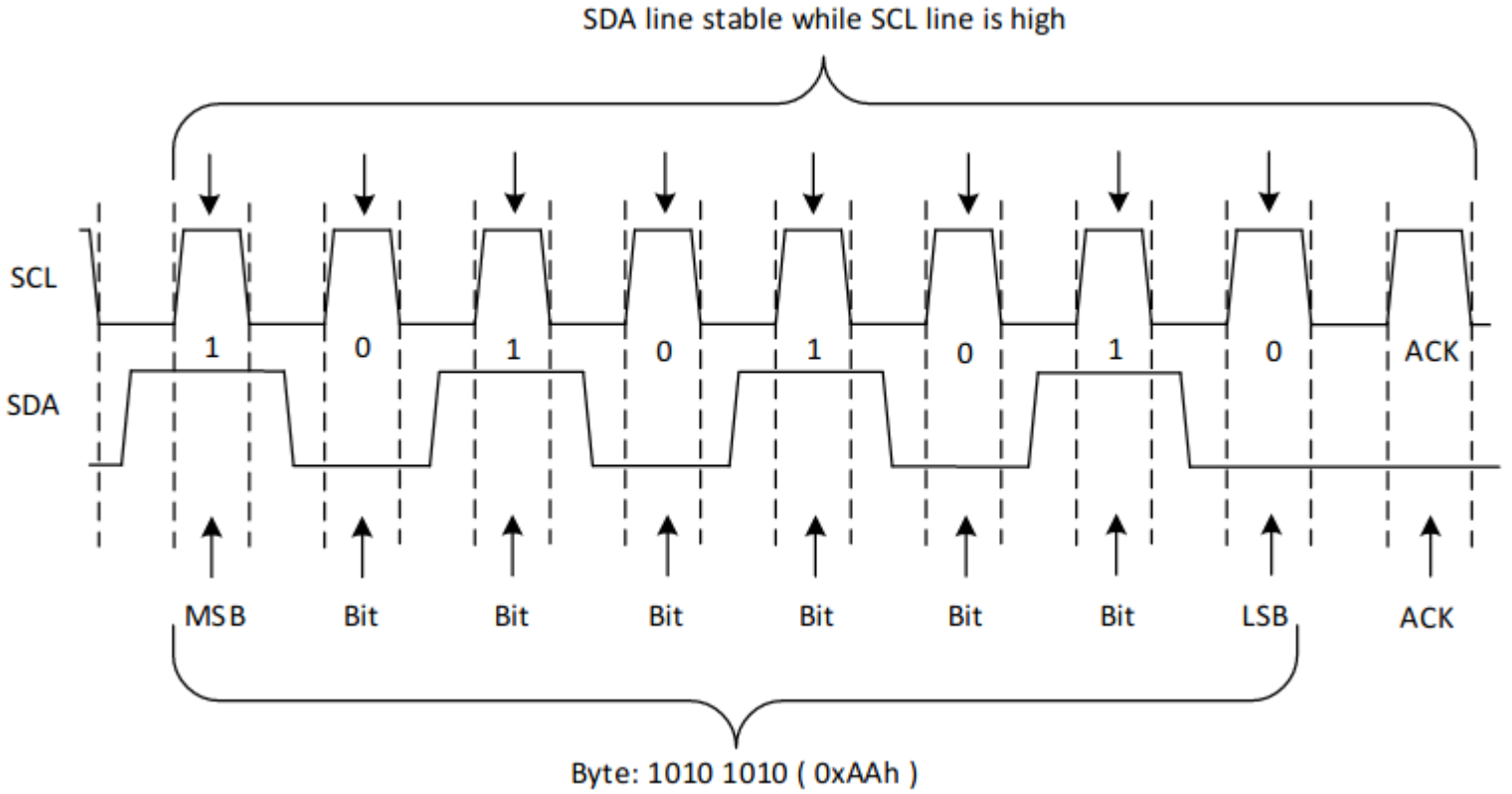


7 bit device address + R/W bit

Start and stop conditions



Data transfers



I2C addressing

The I2C bus uses the master/slave paradigm. The CPU acts as the master and the sensors as slaves.
The I2C bus uses 7 address bits (the 8th bit is the R/W line). A maximum of 128 slaves can therefore be connected to a single master.
The ESP32 has two hardware I2C buses, but MicroPython also provides a software I2C implementation using bit-banging on any GPIO line.
The pins for SCL (the I2C clock line) and SDA (the I2C data line) are

- SCL: GPIO 22
- SDA: GPIO 21

Control and readout of an I2C based sensor requires a driver accessing it using the MicroPython I2C driver calls. An example for the SHT30 temperature and humidity sensor is explained here.

MicroPython's I2C driver

From the MicroPython doc on the I2C bus for the ESP32
On the ESP32 CPU:
scl : GPIO 22
sda: GPIO 21

```
from machine import Pin, SoftI2C

i2c = SoftI2C(scl=Pin(5), sda=Pin(4), freq=100000)

i2c.scan()              # scan for devices

i2c.readfrom(0x3a, 4)   # read 4 bytes from device with address 0x3a
i2c.writeto(0x3a, '12') # write '12' to device with address 0x3a

buf = bytearray(10)     # create a buffer with 10 bytes
i2c.writeto(0x3a, buf)  # write the given buffer to the peripheral
```

```
from machine import Pin, I2C

i2c = I2C(0)
i2c = I2C(1, scl=Pin(5), sda=Pin(4), freq=400000)
```

Scanning the I2C bus

```
i2c = I2C(1,scl=scl,sda=sda)
addr = i2c.scan()
```

```
Scanning the I2C bus
Program written for the workshop on IoT at the
African Internet Summit 2019
Copyright: U.Raich
Released under the Gnu Public License
Running on ESP32
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- 45 -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

>>>
```

d

The SHT30

The SHT30 is a digital temperature and relative humidity sensor
I wrote the sht3x driver myself and integrated it into the MicroPython firmware
The user program becomes very simple:

```
from sht3x import SHT3X
sht30 = SHT3X()
tempC, humi=
sht30.getTempAndHumi(clockStretching=SHT3X.CLOCK_STRETCH,repeatability=SHT3X.REP_S_HIGH)
```

is all that is needed to get at the measured temperature (in °C) and relative humidity (in %) values.
All the detailed command handling is done within the driver

d

Drivers are classes

Drivers are often implemented as Python classes
When an instance of a class is created the __init__ method its called
This method initializes the I2C bus and performs any initialization that the device may need.
The ESP32 has 2 hardware I2C interfaces and I2C bus 1 is reserved for users.
MicroPython includes a driver for the I2C bus whose methods are used to communicate with the specific I2C device.

!!! Pause !!!

Question and Answer Session