Developing an IoT System

Uli Raich

Two lectures on Hardware and Software for
the Internet of Things

Lecture 1: Introduction to the hardware
and development environments and interfacing the “things”
(The “T” part of IoT)

Presented at the African Internet Summit Online 2021

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
- 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

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

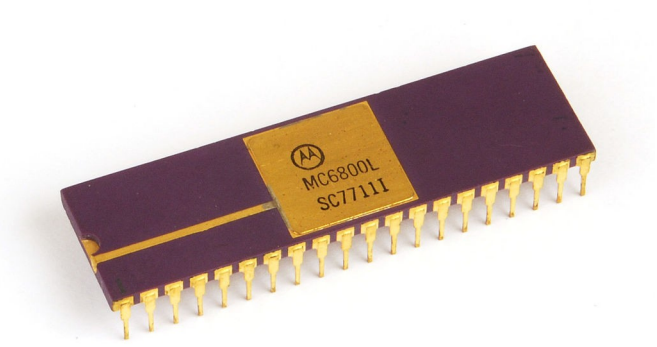
When I was a doctoral student this was the computer I worked with:

- 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
- Assembly language only
- Programmed in binary machine code entered through a keypad

… and today? for 7 Euros?

Hardware Specifications

Chipset	ESPRESSIF-WROVER-B 240MHz Xtensa® single-/dual-core 32-bit LX6 microprocessor
FLASH	QSPI flash 4MB / PSRAM 8MB
SRAM	520 kB SRAM
Button	reset
USB to TTL	CP2104
Modular interface	UART、SPI、SDIO、I2C、LED PWM、TV PWM、I2S、IRGPIO、ADC、DACLNA pre-amplifier
On-board clock	40MHz crystal oscillator

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

What is IoT

IoT stands for the **I**nternet **o**f **T**hings.

We need:

- A network interface: either Ethernet of WiFi
- A processor powerful enough to run the network protocol layers
- Interfaces to *the things:*
  - 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 do we find micro-controllers?

Answer: Everywhere!

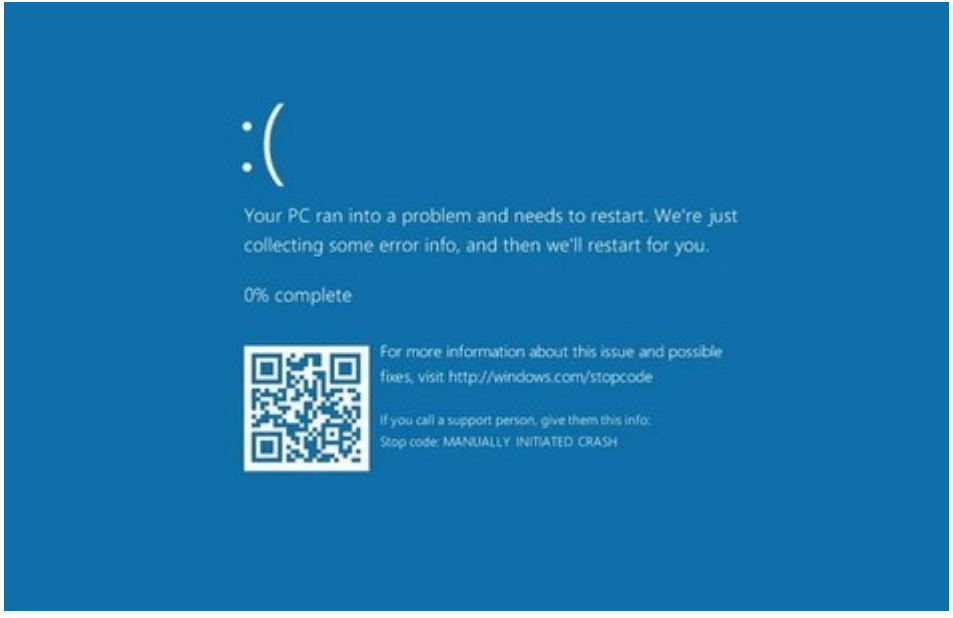
- Car (many of them!)
- Coffee machine
- TV set, radio
- Watch
- Hand phone

What used to be a joke

Two engineers standing around a coffee machine with this screen

One saying to the other:

“Those were the good old times when
coffee machines still worked without Windows”



...becomes reality

A Microsoft Coffee machine .. with Windows.

by Hilbert Hagedoorn on: 01/12/2009 11:32 AM | source: | 0 comment(s)

Among the technologies Microsoft is highlighting in its booth at CES this year is a hardware and software platform for the next generation of more useful and flexible household objects, appliances and accessories. The Windows-based platform is the result of a collaboration between Microsoft and a recently formed company called Fugoo. Two of the concept designs featured in a video at the booth are a

« QNAP new 6-bay NAS · **A Microsoft Coffee machine .. with Windows.** · DRAM module makers increase prices »

Embedded systems → IoT

The embedded systems course used a Raspberry Pi,
cost per station: ~ 120 US $

Programming language: C

I was promised that all exercises of the course would be ported to Python

Decided not only to port to Python but also change to cheapest possible hardware and extend to Internet access (IoT)

Cost per station now: < 50 US$

If you run the course for 5 years this brings you to a cost of 10 US $ per student and year. Students at UCC were allowed to take the equipment home for experimentation.

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
- 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

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

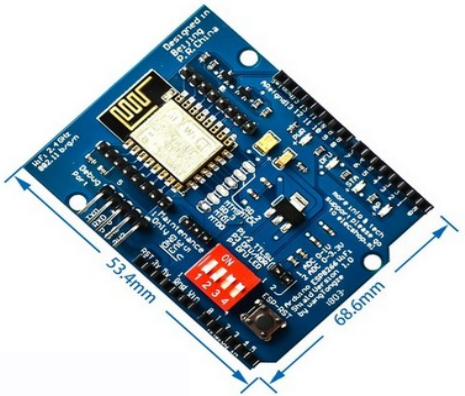
Arduino + WiFi shield

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
```



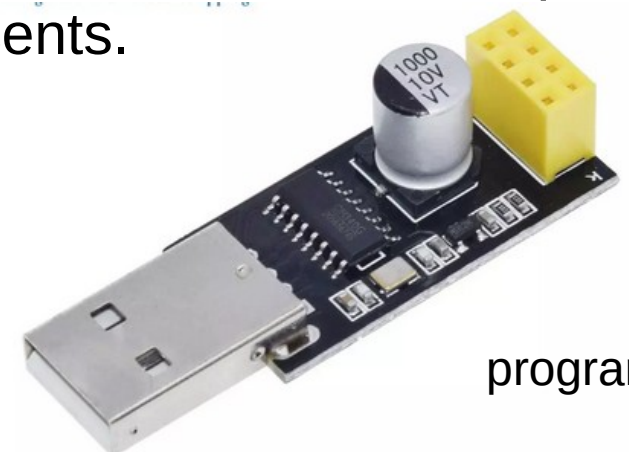
Cost: ~12-15 US $
Popular because of simple C++ IDE

The low end

If you just need an Internet connected temperature sensor, the ESP01 will do!

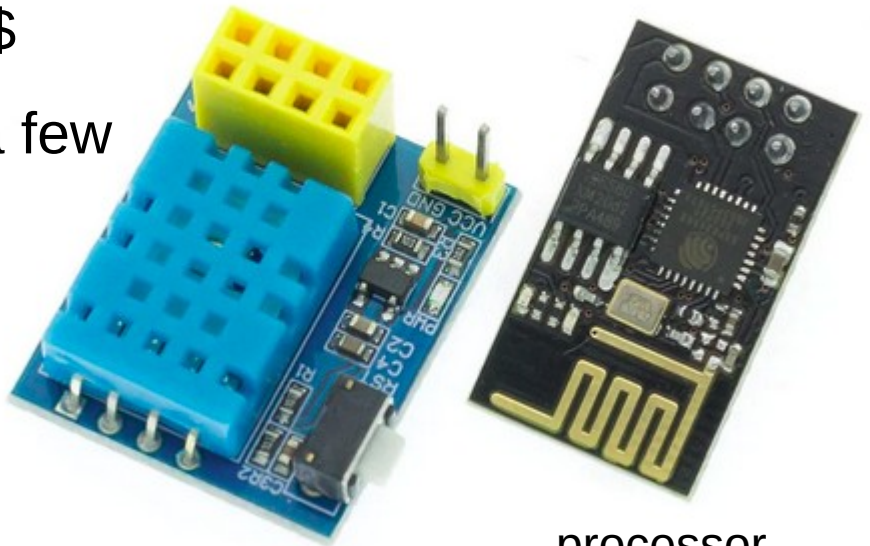
Cost of processor and sensor: < 2 US $

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.

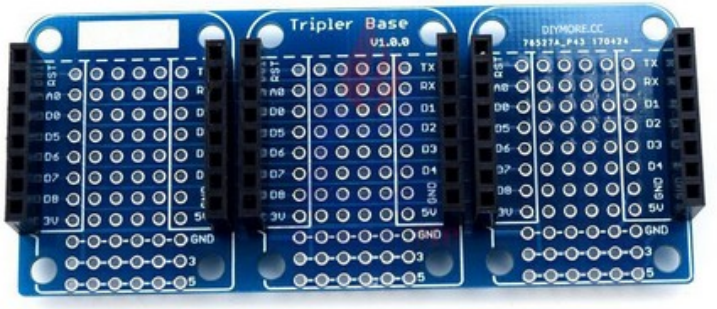
It provides a selection of CPU boards

- ESP8266 CPU
- ESP32 with or without SPIRAM

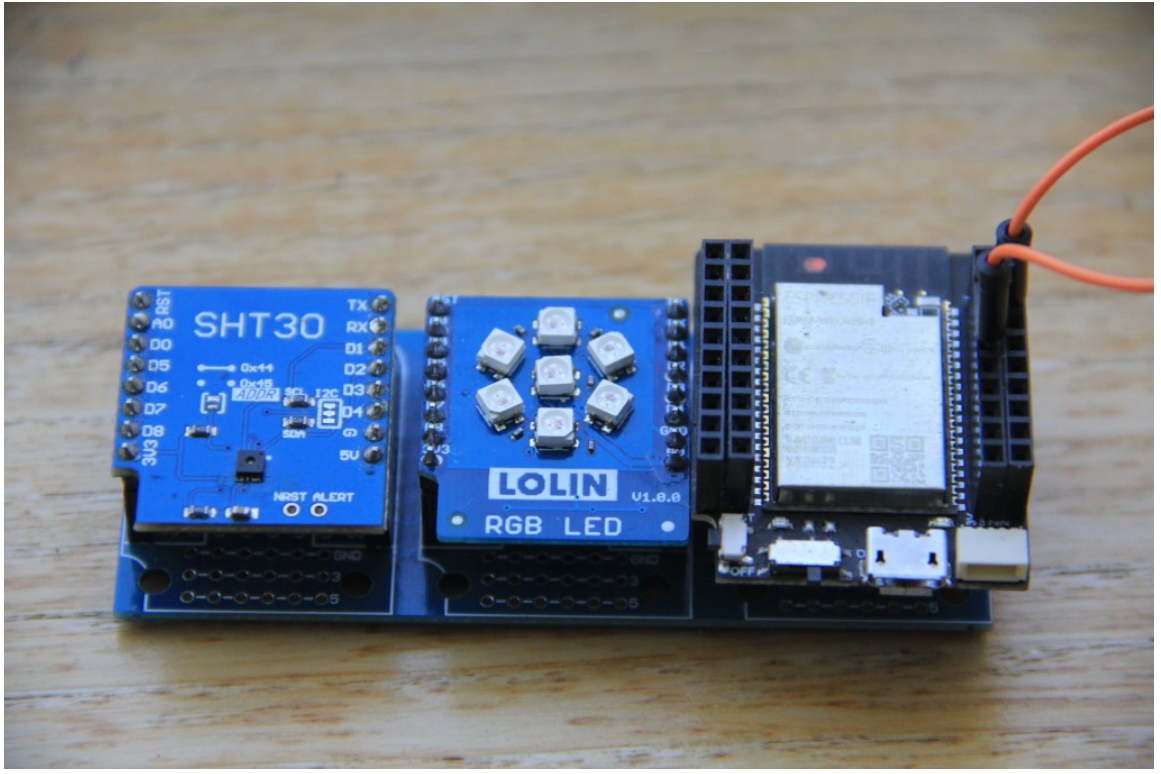
and it comes with a large selection of sensor/actuator boards

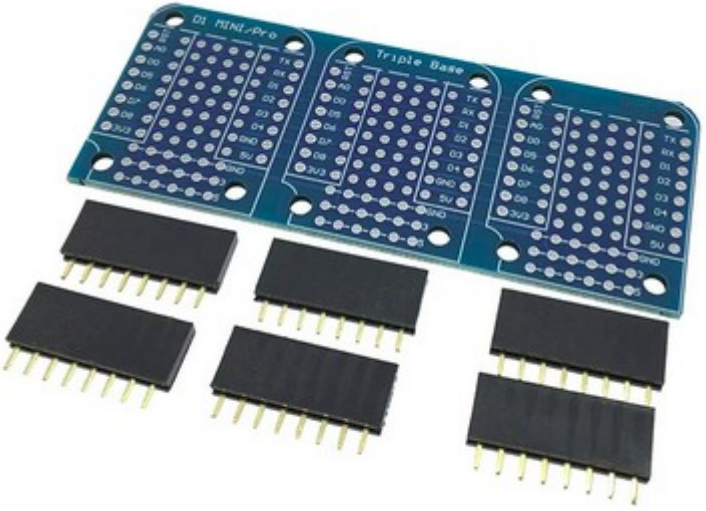
The sensor boards are connected to the

CPU through a simple plug and play system



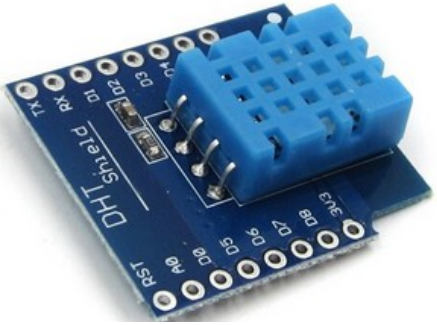
Connection between CPU and sensors

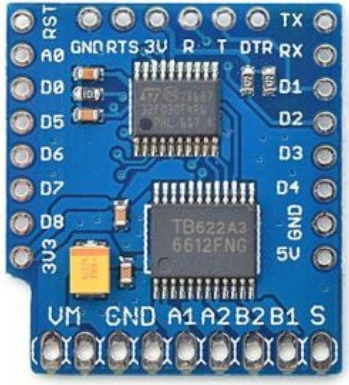




AFRICA INTERNET SUMMIT'21 ONLINE 31 MAY - 4 JUNE 2021

… 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 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 ...

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++.

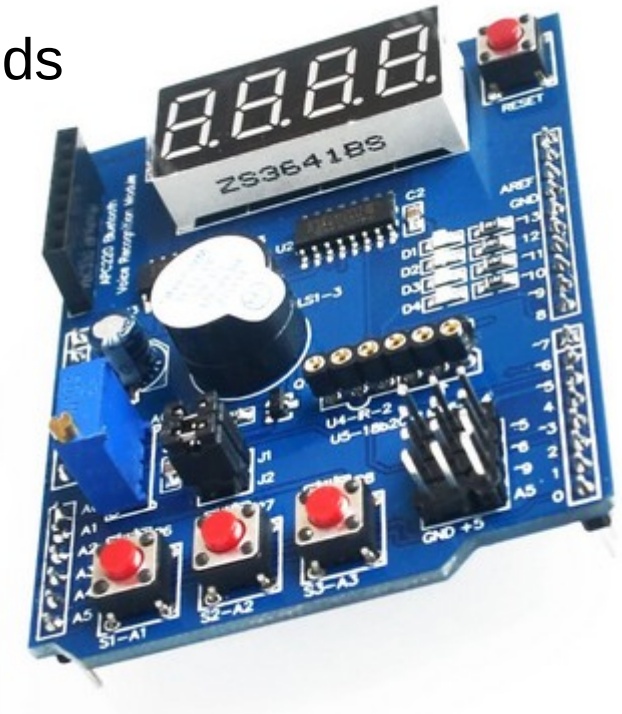
The learning curve to start your first application is very steep.

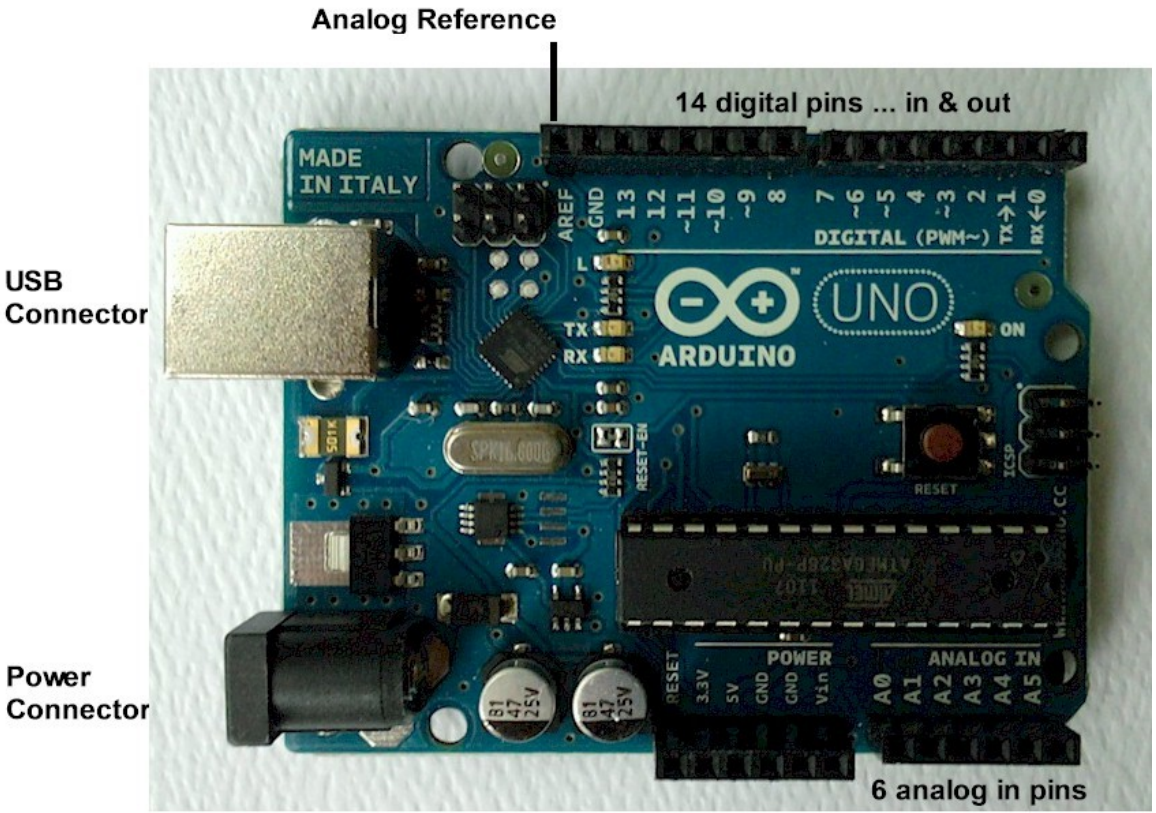
The same is true for the STM32 chips.

The Arduino

Open source hardware

Shields





Arduino IDE

Designed for the beginner and hobbyist market

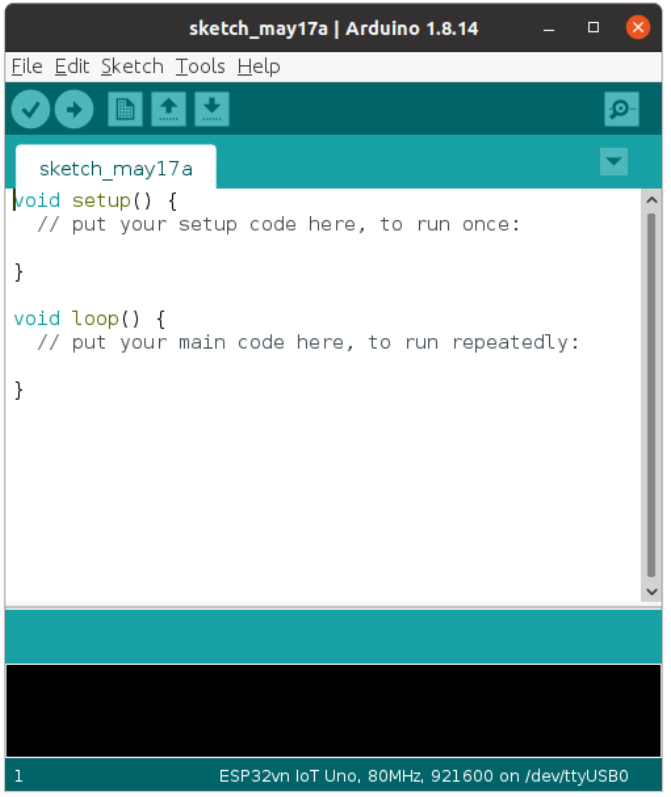
Originally provided for the AVR processors but now also available for ESP8266, ESP32 or the STM32 processors

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 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 (the **R**ead, **E**valuate,**P**rint **L**oop) 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

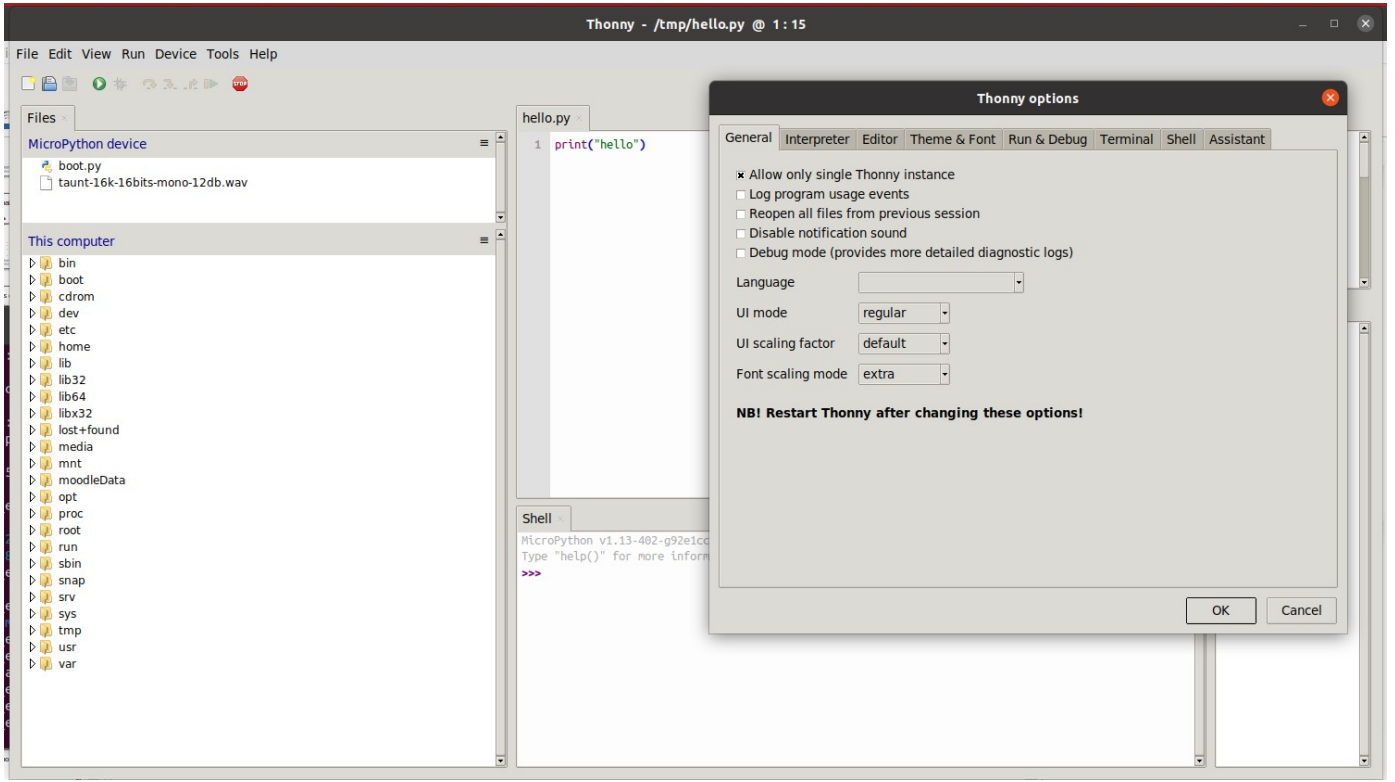
How do we talk to the ESP32?

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

thonny


Thonny - /tmp/hello.py @ 1 : 15
File Edit View Run Device Tools Help
Files
MicroPython device
boot.py
taunt-16k-16bits-mono-12db.wav
This computer
bin
boot
cdrom
dev
etc
home
lib
lib32
lib64
libx32
lost+found
media
mnt
moodleData
opt
proc
root
run
sbin
snap
srv
sys
tmp
usr
var
hello.py
1 print("hello")
Shell
MicroPython v1.13-402-g92e1cc
Type "help()" for more inform
>>>
Thonny options
General
Interpreter
Editor
Theme & Font
Run & Debug
Terminal
Shell
Assistant
Allow only single Thonny instance
Log program usage events
Reopen all files from previous session
Disable notification sound
Debug mode (provides more detailed diagnostic logs)
Language
UI mode
regular
UI scaling factor
default
Font scaling mode
extra
NB! Restart Thonny after changing these options!
OK
Cancel

Interfacing to the “things”

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

- GPIO pins
- PWM
- Capacitive touch sensor
- I2C interface
- SPI interface
- Analogue to digital converter
- Digital to analogue converter
- Timer

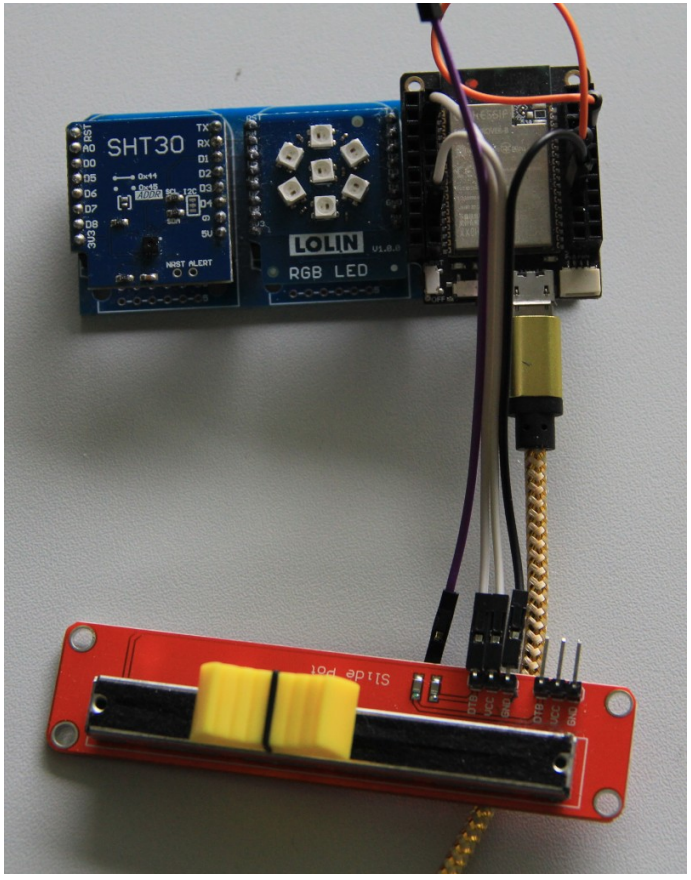
Setup for demos

The ESP32 CPU on the right with a user LED

The Lolin NeoPixel board with 7 rgb LEDs

The SHT30 temperature and humidity sensor controlled through the I2C bus

A linear potentiometer connected to the ESP32 ADC



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, analyze serial protocols

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!

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 is programming infrastructure

- Editor, compiler, linker
- Program execution

work well.

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

Pulse Width Modulation

How can we change the LED light intensity with a single digital GPIO line?

The answer is: Pulse Width Modulation (PWM)

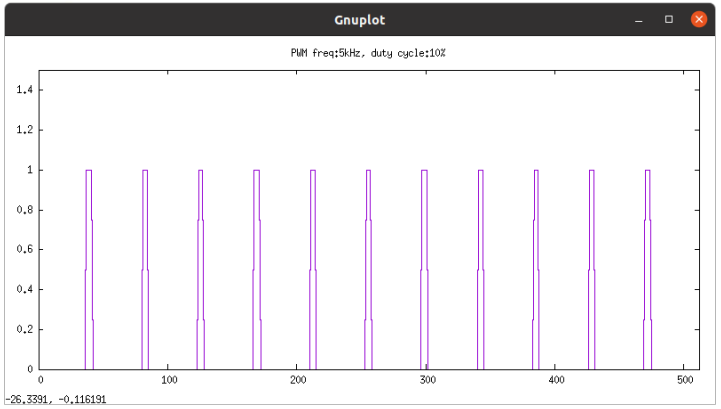
Instead of emitting a steady zero or one signal level we emit a frequency (the modulation frequency) and we change the time the signal is high during a pulse (duty cycle)

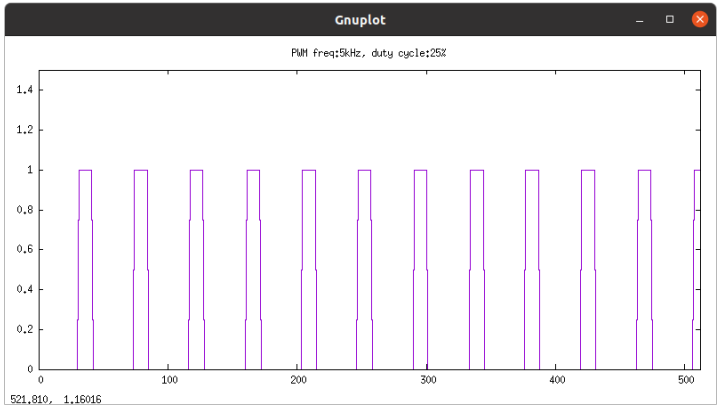
The frequency is high enough and the LED persistence long enough, such that the human eye cannot resolve the frequency.

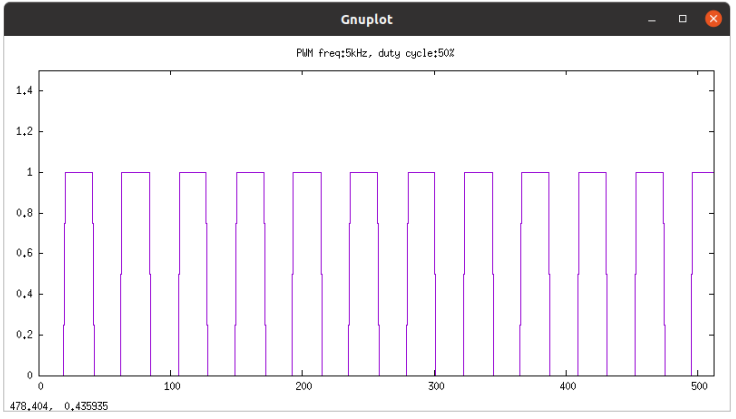
The average current through the LED is changed and such the light intensity.

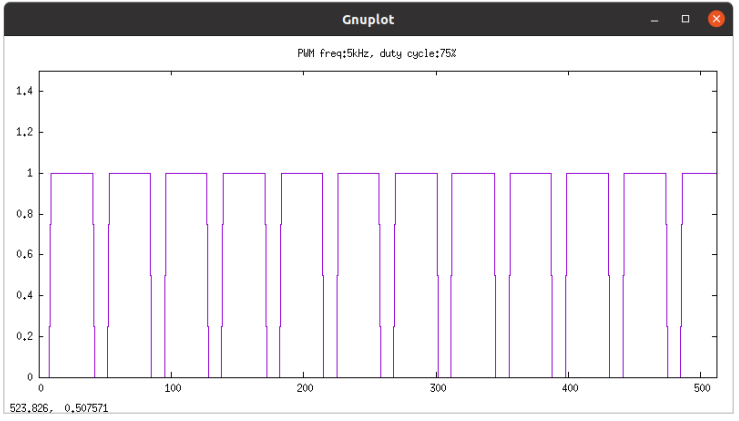
PWM is also used to control servo motors

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

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

The sensor shield uses GPIO 26 to control the LEDs

NeoPixel program

```
import machine, neopixel, time

pin = 26   # connected to GPIO 26 on esp32
n=7        # number of LEDs

def clearChain():
    for i in range(n):
        neoPixel[i] = (0, 0, 0)
    neoPixel.write()

neoPixel = neopixel.NeoPixel(machine.Pin(pin), n)

for i in range(n):
    print("LED address: ",i)
    neoPixel[i] = (0,0,0x1f)
    neoPixel.write()
    time.sleep(1)
    clearChain()

clearChain()
```

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

ADC readout program

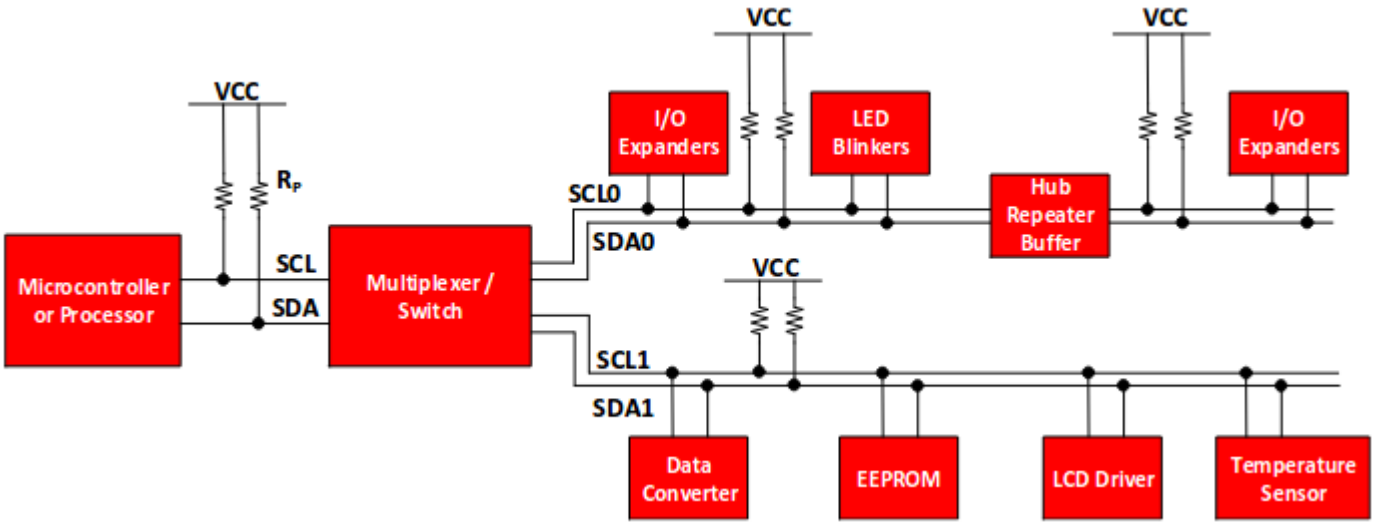
```
from machine import Pin, ADC
from time import sleep

slider = ADC(Pin(36))  # create ADC object on ADC pin 36
slider.atten(ADC.ATTN_11DB)

while True:
    print("Slider: ",slider.read())
    sleep(0.5)
```

The I2C bus

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



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.

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 humidity (in %) values.

All the detailed command handling is done within the driver

IoT course exercises

Course on Internet of Things

Exercises:

- Exercise 1: REPL and standard Python programming
- Exercise 2: LEDs and NeoPixel
- Exercise 3: Switches
- Exercise 4: The DHT11 Temperature and Humidity Sensor
- Exercise 5: The I2C Bus and the SHT30 Temperature and Humidity Sensor
- Exercise 6: GPS and interface through UART
- Exercise 7: Motors
- Exercise 8: Real Time Clock and Data Logging
- Exercise 9: ADC and DAC exercises
- Exercise 10: ST7735 TFT Display
- Exercise 11: A WEB Server
- Exercise 12: IoT via MQTT and Cayenne

Supplementary exercises:

- Exercise 13: Seven Segment Display and Keypad
- Exercise 14: BMP180 air pressure sensor
- Exercise 15: Air Quality
- Exercise 16: Infrared remote control
- Exercise 17: Audio systems