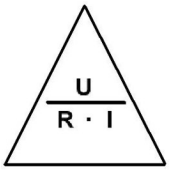
Elektronika

Napětí, Proud. Ohmův zákon

Napětí a proud jsou základní fyzikální veličiny a běžně se značí písmeny U a I a jejich jednotky jsou V (volt) a A (ampér).

Obě jsou na sobě závislé. Proud nemůže existovat bez napětí. (analogie voda: napětí-rozdíl hladin,proud-průtok vody). K vyjádření vztahu mezi napětím a proudem používáme spojení elektrický odpor, značíme jej písmenem R a jeho fyzikální jednotka je Ω

Vztah mezi proudem, napětím a elektrickým odporem lze zapsat následujícím způsobem:



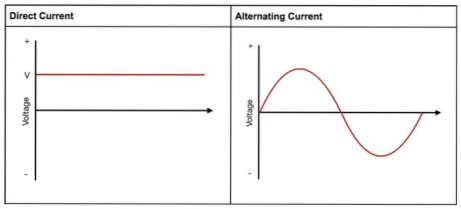
$U=R \cdot I$ [V,Ω,A]

$R = U / I$

$I = U / R$

Výkon lze zapsat jako P = U.I [W], Práce pak W = U.I.t [J,Ws] ([kWh])

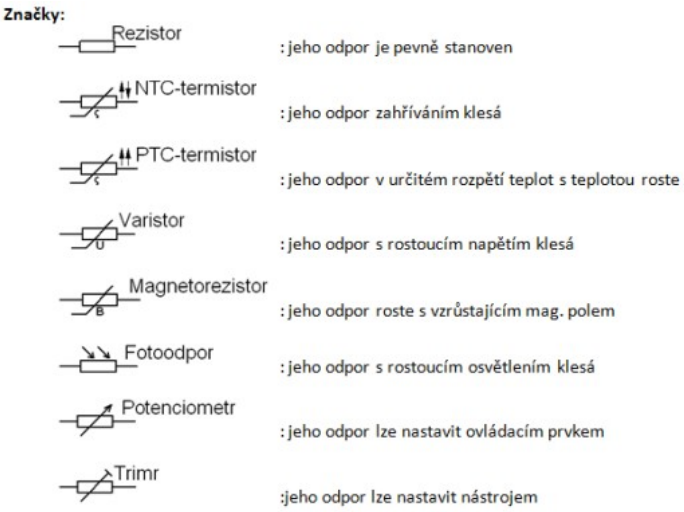
Stejnosměrné – střídavé napětí.



Elektronické součástky

Rezistor (Odpor)

Rezistor je pasivní (dvouvývodová) elektrotechnická součástka projevující se v elektrickém obvodu v ideálním případě jedinou vlastností – elektrickým odporem (jednotka Ohm, značka Ω). Důvodem pro zařazení rezistoru do obvodu je obvykle snížení velikosti elektrického proudu nebo získání určitého úbytku napětí. Pokud rezistorem prochází vyšší proudy pak je třeba vzít do úvahy i jeho dovolený ztátový výkon který bývá úměrný velikosti.



Rezistory se rozlišují podle konstrukce, podle velikosti odporu a dovoleného zatížení. Podle odporového materiálu lze rozlišit např. drátové rezistory, vrstvové rezistory a uhlíkové rezistory. (vzájemně se pak liší např. stabilitou (např. teplotní), parazitními vlastnostmi (např. indukčnost)) Rezistory, jejichž odpor lze měnit, se nazývají reostaty, potenciometry nebo trimry.

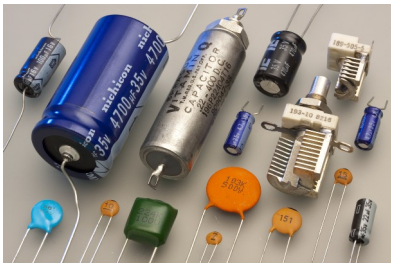
Typický rozsah odporu u bežných malých rezistorů je 1Ohm-10MOhm a ztrátový výkon 0.1-1W.

Vyrábí se v přesnostních řadách tak aby logaritmicky pokrývali celý rozsah (řada E6,E12,E24,…)

Kondenzátor

Kondenzátor je elektrický prvek, který slouží k ukládání elektrického náboje a energie v elektrickém poli. Skládá se ze dvou vodičů (elektrod) oddělených dielektrikem. V ideálním případě má jedinou vlastnost – Kapacitu [F]. Pro běžné typy je 1F příliš velká jednotka a bežný rozsah kapacit je 1pF-1000uF . Konstrukce kondezátoru obvykle nutí vzít do úvahy další parametry jako je maximální dovolené napětí (typicky 5-50V), případně ERS (ekvivalentní sériový odpor) a závislost kapacity (na teplotě, napájecím napětí atp).

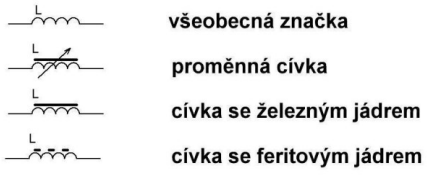
Pro menší kapacity (do řádu uF) jsou kondenzáory obvykle nepolarizované, pro vyšší kapacity se používají kondenzátory elektrolytické kde izolantem je speciální roztok – elektrolyt. Tyto kondenzátory mají danou polaritu kterou je třeba dodržet.



[(130) Capacitor and SMD Capacitor Codes Explained with Examples - YouTube]

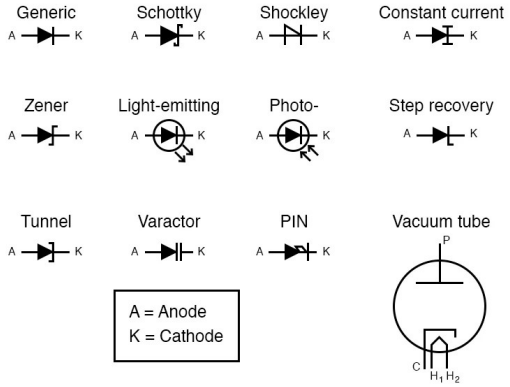
Cívka

Cívka je pasivní elektronický součástkový prvek, který se používá k ukládání energie v magnetickém poli. Cívka se skládá z drátu nebo vodiče navinutého na dutou trubku nebo jiný materiál. Když se přivádí elektrický proud do cívky, vytváří se magnetické pole kolem ní. Cívky se používají v mnoha elektronických zařízeních jako např. zdroje, reproduktory, elektromotory, transformátory a mnoho dalších.



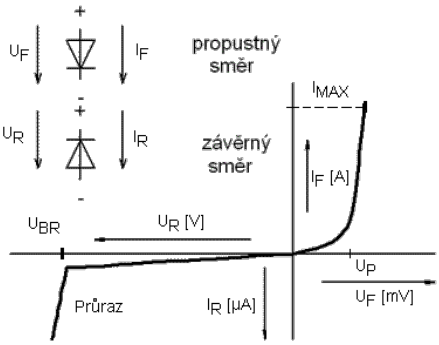
Dioda

Dioda je aktivní polovodičová součástka (přechod PN) se dvěma elektrodami, označovanými jako anoda a katoda. Dioda se vyznačuje velmi odlišným tvarem voltampérové charakteristiky v závislosti na polaritě přiloženého napětí. Pokud je na diodu přiloženo závěrné napětí (napětí v opačném směru), tak dioda prakticky nevodí proud. Pokud je na diodu přiloženo kladné napětí (napětí v propustném směru), tak dioda vodí proud.



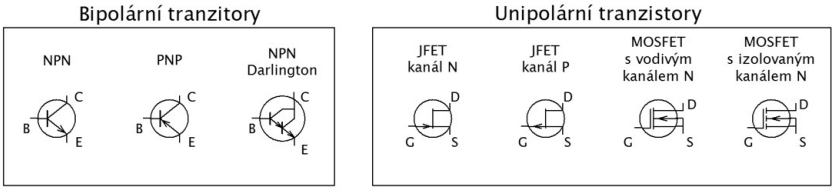
Tranzistor

Tranzistor je polovodičová součástka, která se skládá ze tří vrstev polovodiče (tří elektrod) – kde lze

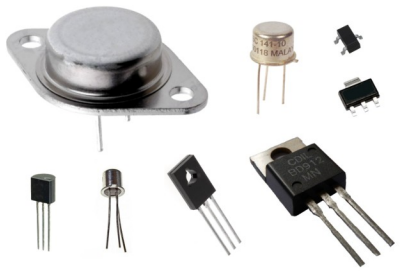


průtok mezi krajními elekrodami řídit prostřední elektrodou. Tranzistory se dělí na bipolární a unipolární podle toho, zda se na zesilování podílí oba typy nosičů nábojů (vodivostní elektrony a díry), nebo jen jeden typ nosičů. Bipolární tranzistor se skládá ze dvou PN přechodů - Tyto vrstvy jsou označovány jako Emitor, Báze a Kolektor – a podle jejich pořadí rozličujeme tranzistotru NPN nebo PNP.

Unipolární tranzistor se skládá z jednoho polovodičového materiálu, je řízen napětím a elektrody se označájí jako Source,Gate,Drain. Podle typu polovodiče pak rozlišujeme tranzistory s N-kanálem a P-kanálem.



Tranzistor lze použít jako zesilovač signálu a nebo ve spínacím režimu (stav zavřeno/otevřeno) který se používá v logických obvodech.



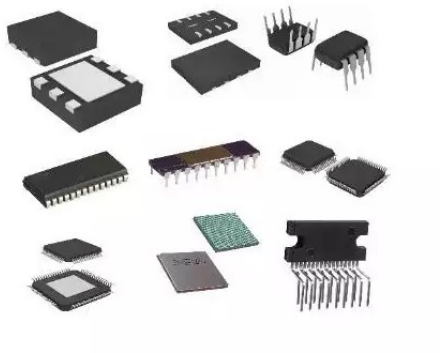
Integrované obvody

Oboby vyssí integrace, obsahující složitější struktury ze základních součástek.

Lze rozdělit na číslicové(digitální) a analogové, případně hybridní.

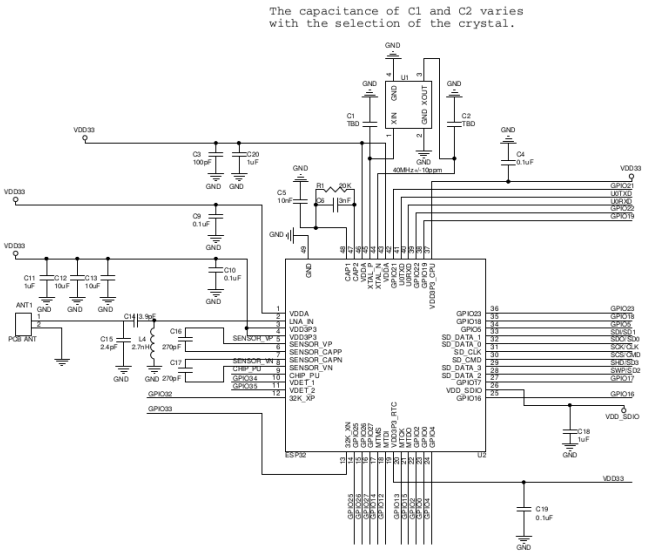
Příkladem číslicových součástek jsou procesory, programovatelné obvody, logická hradla, atd.

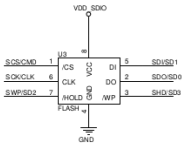
Příkladem analogových integrovaných obvodů jsou operační zesilovače, obvody spínaných zdrojů, budiče sběrnic atd.

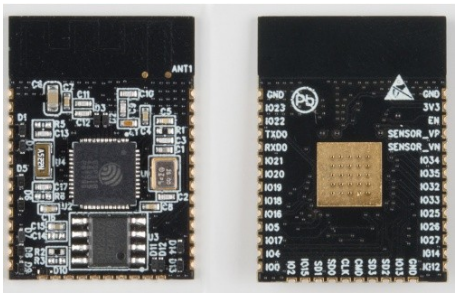


Elektrické obvody

Popisujeme schématem, při výrobě typicky navrhujeme pcb (printed circuits board - deska plošných spojů), osazení ručně (pájkou) nebo automaticky (osazovací automat)







Měřící přístroje

Multimetr

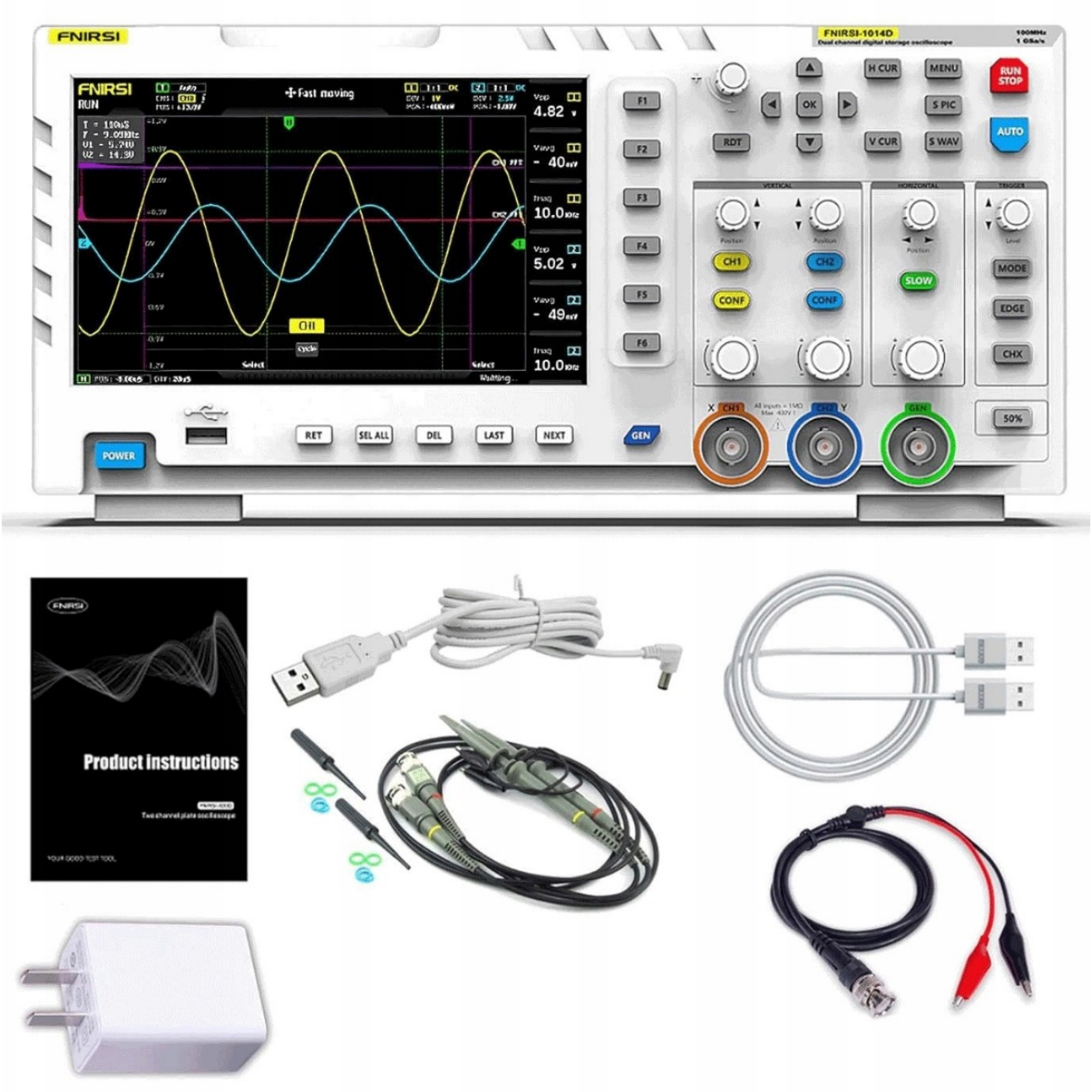
Multimetr je (typicky menší přenosný) elektronický měřicí přístroj, který v sobě kombinuje několik funkcí jako například ampérmetr, voltmetr a ohmmetr. Jednotlivé měřicí funkce jsou na multimetru

zpravidla rozděleny do několika měřicích rozsahů proto, aby bylo dosaženo relativně širokého měřicího rozsahu, vysokého rozlišení se zachováním vysoké přesnosti měření.

Problematika/limity měření dc, ac (true RMC) a obecných signálů.

Osciloskop / logický analyzátor

Osciloskop je elektronický měřicí přístroj s obrazovkou vykreslující časový průběh měřeného (typicky napěťového) signálu.



Časový průběh je obvykle vykreslován od nějakého spouštěcího bodu (trigger) který je uživatelsky definovatelný (úroveň napětí, směr/hrana signálu, doba trvání atp) a umožnuje tak získat pohled na signál v bodě zájmu.

Napájení Zařízení

Napětí / proud (vnitřní odpor , „kapacita”)

Typicky používáme zdroje s konstantním napětím – a je limitováno kolik jsou schopny dodat proudu (tj. Výkon). Nejpoužívanější zdroje:

Síť nn (power line/grid) – 230V st (střídavých) (ac = alternating current)

Baterie (primární články) 9V, 1.5V (AA,AAA), 3V CR2032 ss (stejnosměrných) (dc = direct current)

Akumulátory – NiCd,NiMh (1.2V), Li-ion,Li-pol (cca 4.2V) ss

Pro získání jiného napětí používáme transformátory (pro ac) nebo měniče/stabilizátory (pro dc)

Transformátor - přenáší střídavou el.energie pomocí el.mag. Indukce – napětí je při převodu možné případně zvyšovat i snižovat.

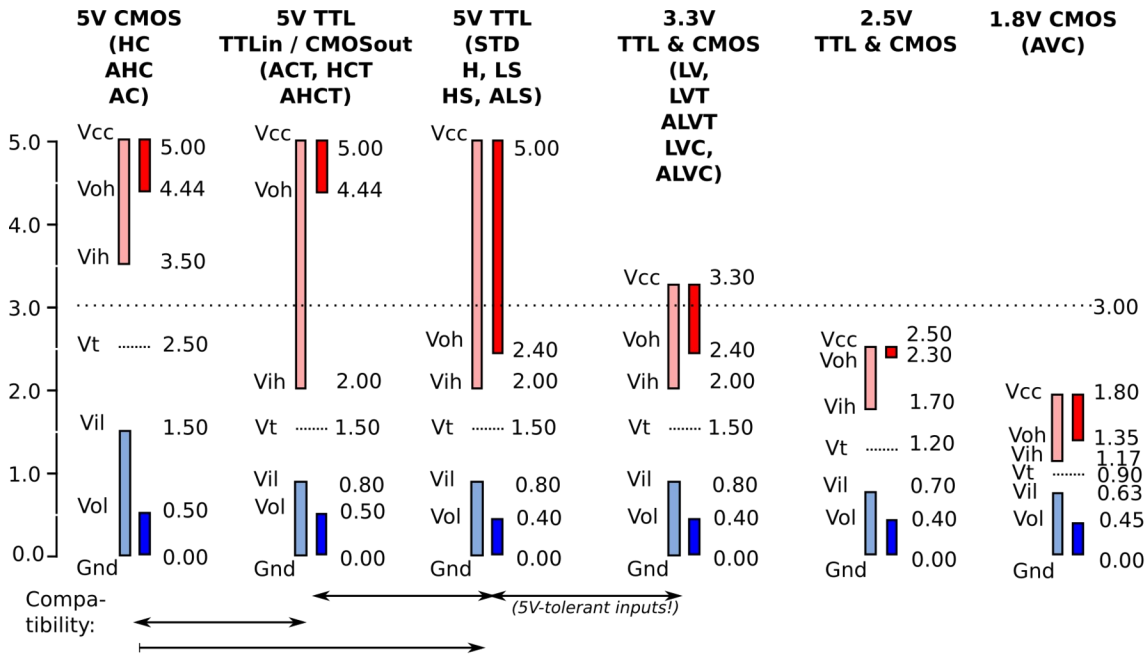
Stabilizátor – lineární napěťový regulátor. z vyššího (stejnosměrého) napětí dělá nižší, napětí je snižováno „pálením“ energie (nízká účinnosti pro větší rozdíl napětí), nízký šum

Měnič (dcdc měnič) – je elektronické zařízení určené pro změnu velikosti stejnosměrného napětí nebo proudu - pro spínání se používají tranzistory a diody, jako zásobníky energie při převodu se používají cívky a kondenzátory. Dle konstrukce je možné napětí zvyšovat či snižovat:
Buck měnič: Snižuje vstupní napětí na nižší výstupní napětí.
Boost měnič: Zvyšuje vstupní napětí na vyšší výstupní napětí.
Buck-Boost měnič: Může buď zvyšovat, nebo snižovat vstupní napětí v závislosti na potřebě.

Konverze ac → dc – usměrňovač. (jednoduchy = dioda)
Konverze dc → ac – střídač (aktivní el.obvod, používaný v dcdc měničích)

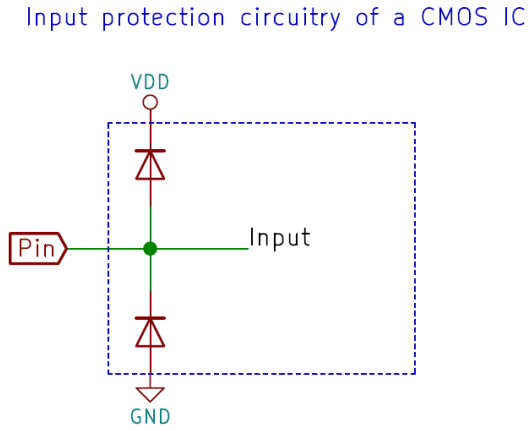
Logické (datové) signály

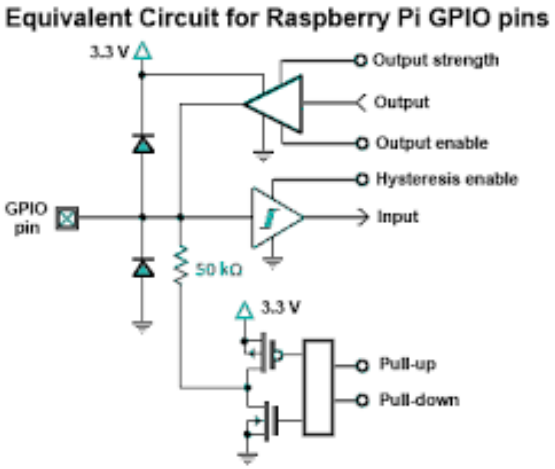
Logické úrovně 0/1 - odpovídající napěťové úrovně – pro digitální vstup/výstup.
Většina obvodů je nyní technologie CMOS
Na výstupu je nižší povolený rozsah než na vstupu (aby bylo zajištěno spolehlivé propojení – rušení, ztráty na vedení)



Data source: EETimes, A brief recap of popular logic standards (Mark Pearson, Maxim).

Ochrany I/O (vstupů/výstupů)





→ napětí mimo rozsah je odváděno diodami (ale diody mají omezenou zatížitelnost!)

Co se stane nedodržíte-li stanovená napětí/proudy:

(Magic smoke - Wikipedia ;-)

Electronics massively overloaded - YouTube

Kdy se to (nejčastěji) stane: Napájení vyšším napětím, připojení vyššího napětí na vstup, spojení výstup-výstup, zapojení výstupu do zkratu

ESP32 (Xtensa® 32-bit 240Mhz LX6)

ESP32 je 32bitový procesor s frekvencí až 240MHz, 2 jádra, FPU (single+double).
Interní ROM 448kB, SRAM 520kB, externí qSPI paměť FLASH+volitelně PSRAM.
Interní periférie: memory protection unit (MPU), and memory management unit (MMU), DMA controler, Cache, ULP koprocesor (ultra low power coprocesor s přístupem k GPIO a ADC)
Wifi + bluetooth (+BLE) interface

GPIO periférie:

ADC (Analog to Digital Converter): 18 channels of 12-bit SAR ADC
DAC (Digital to Analog Converter): 2 channels of 8-bit DAC
Touch Sensors: 10 capacitive touch GPIOs
SPI (Serial Peripheral Interface): 4 SPI interfaces
I²S (Inter-IC Sound): 2 I²S interfaces for audio applications
I²C (Inter-Integrated Circuit): 2 I²C interfaces for communication with sensors and other devices
UART (Universal Asynchronous Receiver/Transmitter): 3 UART interfaces for serial communication
SD/SDIO/MMC/eMMC Host Controller: For interfacing with SD cards and other storage devices
Ethernet MAC Interface: With dedicated DMA and planned IEEE 1588 Precision Time Protocol support
CAN Bus: CAN 2.0 interface for automotive and industrial applications
Infrared Remote Controller: Up to 8 channels for IR communication
Pulse Counter: Capable of full quadrature decoding
PWM (Pulse Width Modulation): Motor PWM and LED PWM (up to 16 channels)
Ultra Low Power Analog Pre-Amplifier: For low-power applications12.

Napájení 3.3V, spotřeba >100mA, řádově uA,10uA v low power mode.

M5stack core2

M5Stack Core2 ESP32 IoT Development Kit | m5stack-store

M5core2 je vývojové zařízení určení pro IoT aplikace. Jádrem je procesor ESP32 (viz výše) a obsahuje 16MB flash a 8MB RAM. USB-C interface pro komunikaci a nabíjení, vestavěnou li-ion baterii 390mAh, 2“ display. vibrační motor, RTC (real-time clock), LED indikace, micro SD karta, audio (I2S interface), 3 kapacitní tlačítka, a s rozšířením 6-osý IMU (intertial measurement unit) (zrychlení + otáčení) a mikrofon.

SW podpora: Arduino (C/C++), UIFlow (Blockly, MicroPython), .NET nanoFramework

(Micro)Python

MicroPython je implementace programovacího jazyka Python, která je optimalizována pro běh na mikrokontrolérech a vestavěných systémech. Je navržena tak, aby byla co nejmenší a nejefektivnější, což umožňuje běh Pythonu na zařízeních s omezenými zdroji, jako jsou malé paměti a „nízký“ výkon procesoru.

Rozdíly mezi MicroPythonem a Pythonem (CPython):

Velikost a výkon: MicroPython je optimalizovaný pro malé paměťové prostory a nízký výkon. Obsahuje pouze základní funkce a knihovny.

Knihovny: MicroPython obsahuje omezenou sadu modulů a funkcí, které jsou optimalizovány pro vestavěné systémy. Některé moduly jsou zjednodušené nebo úplně chybí.

MicroPython je navržen pro efektivní správu paměti a zdrojů, což je klíčové pro běh na mikrokontrolérech s omezenou pamětí.

MicroPython poskytuje přímý přístup k hardwarovým periferiím, jako jsou GPIO piny, I2C, SPI, UART, ADC, atd. To je klíčové pro vestavěné aplikace, Cpython obvykle nemá přímý přístup k hardwaru, ale může používat knihovny třetích stran pro interakci s hardwarem.

Pozor – obecně každá deska má specifický postup pro instalaci MicroPythonu a příkazy pro přístup k HW periférium, které jsou závislé na specifikách daného HW.

MicroPython má typicky interaktivní REPL (Read-Eval-Print Loop), který vám umožňuje psát a testovat kód v reálném čase. Můžete začít psát jednoduché příkazy přímo v REPL nebo vytvořit skripty a nahrát je na desku.

Specifické programové konstrukce pro M5stack core2: m5-docs (m5stack.com)

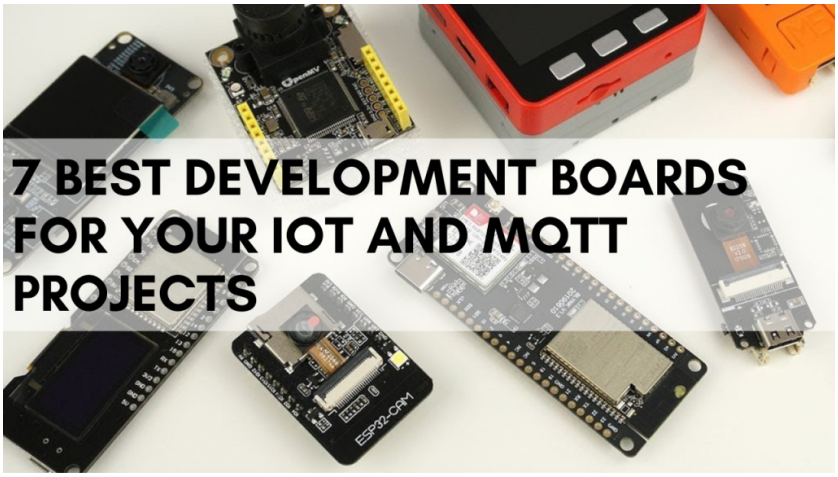
IoT

Internet of Things (Internet věcí) – (vzdálené, bezdrátové) propojení modulů a dat pro dosažení společné funkce.

Příklady aplikací
Chytrá města (osvětlení, doprava, správa odpadu)
Chytré domácnosti (řízení prostředí (teplota, vlhkost, rekuperace), efektivní využívání energií (fve), skladování (chytrá lednice)
Zdravotnictví/Healthcare – sledování zdravotního stavu (monitoring, diagnostic. Help)
Zemědělství – monitoring, precizní zemědělsví

Internet of Things; Why? Because we can… | thethings.iO Blog (BigBangTheory, S01e09) ;-)

S čím jiným například lze dále pracovat: (ceny modulů v řádech desítek korun az vyšších stokorun)



7 best development boards for IoT and MQTT projects

První krůčky s M5stack core2

Power-on – stisk levé tlačítko, Power off – dlouhý stisk levého tlačítka
Reset – stisk tlačítka dole (pozor, má zpoždění)

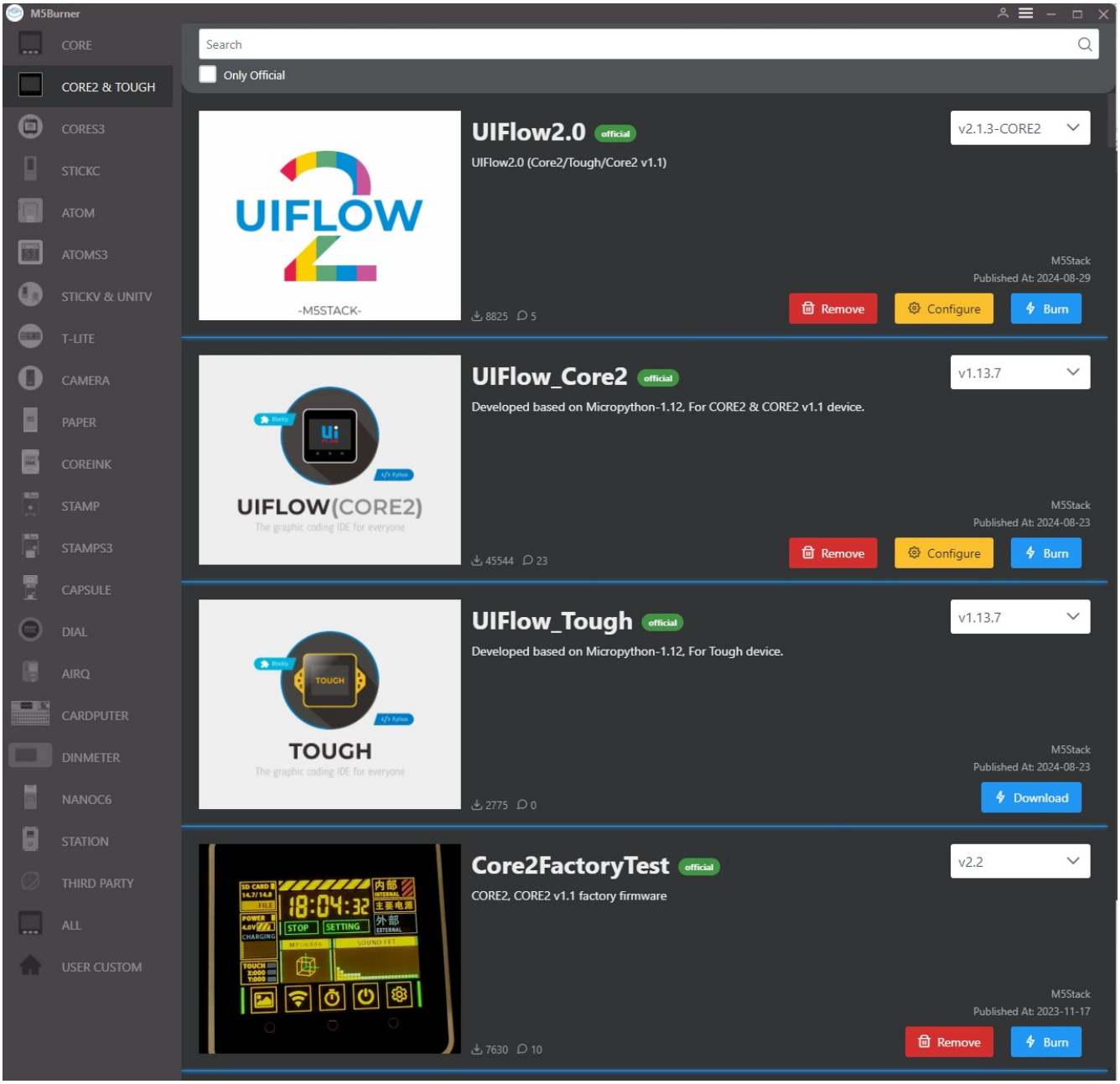
M5stack core2 instalační soubory: m5-docs (m5stack.com)

Nainstalovat driver CH9102 a CP210x pro USB připojení. kontrola seriového portu we win11: win+X, správce zařízení, porty (COM a LPT) - zjistit číslo portu

UIFLOW FIRMWARE BURNING TOOL – nahrání systému - vlevo vybrat vaše zařízení (core2 & touch), napravo pak firmware (UIflow core2) - nebo případně dalsích spec.programů např.Hwtest

M5Burner.exe (UIFLOW FIRMWARE BURNING TOOL)

„Burn“ pro nahrání firmware (je třeba zvolit port. kontrola seriového portu: win+X, správce zařízení, porty (COM a LPT))



Core2FactoryTest – test základních sensorů, displaye atp.

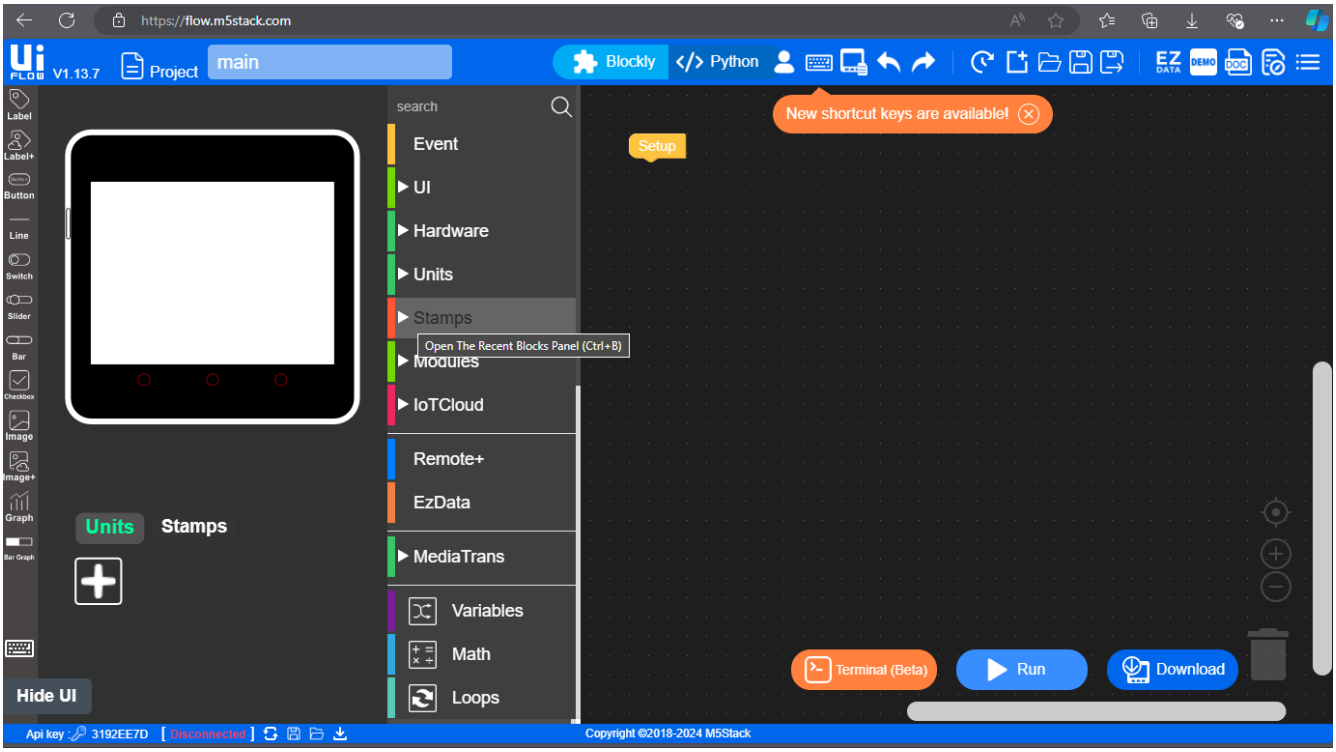
Uiflow(_core2) – základní firmware pro Blockly, MicroPython - nainstalovat pro vývoj/testování

(po instalaci/spuštění na m5core2 nastavit přístupové údaje k síti)

Přístup přes webový prohlížeč (vývojové prostředí UIflow): https://flow.m5stack.com/

Zde vybrat vývojové prostředí UIflow (nebo UIflow2 pro UIflow2.0)

(web vyžaduje přihlášení. Pro vzdálený přístup k m5core nutnost zadat APIKEY z display zařízení)



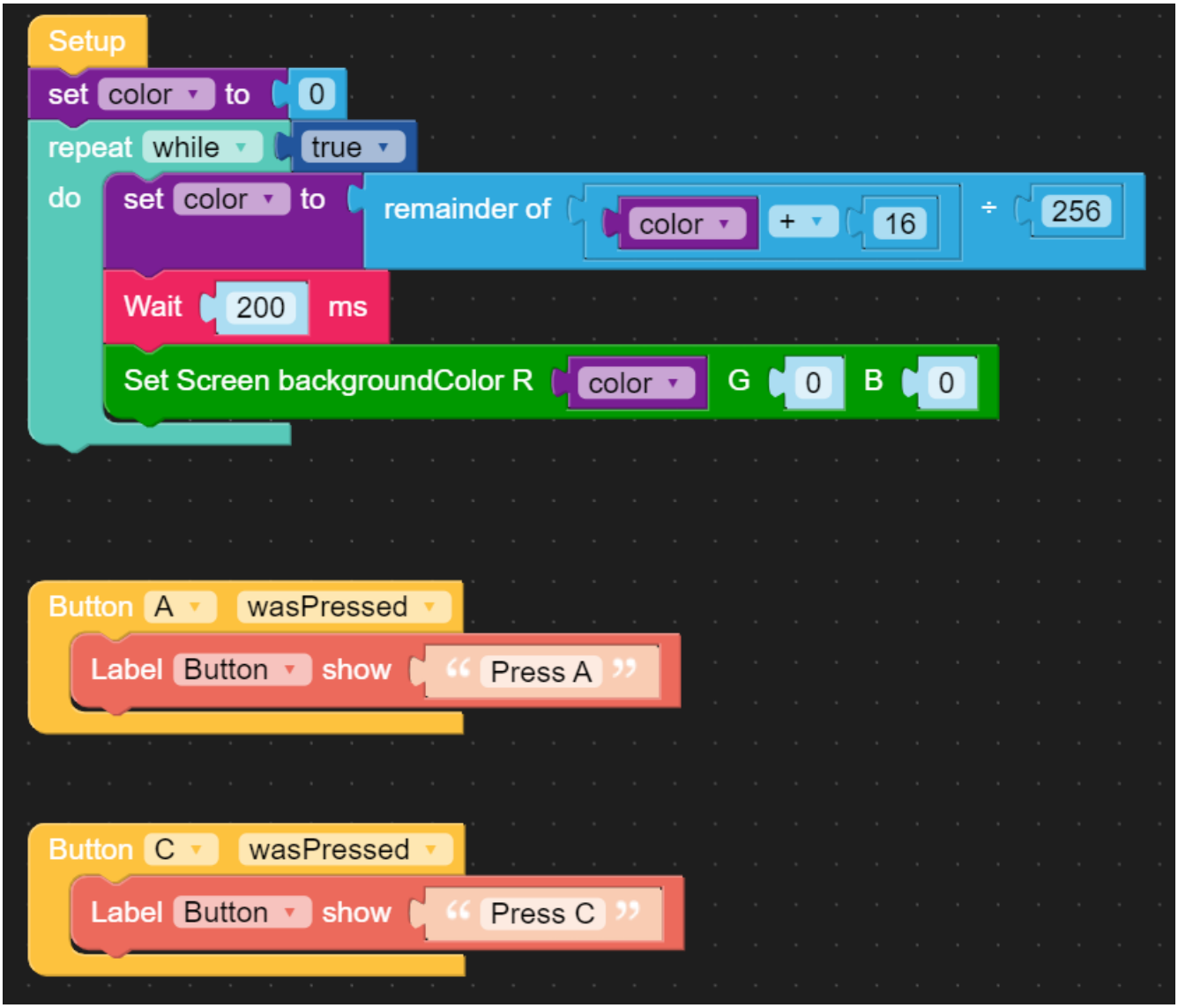
Vpravo nahoře –  - settings – zadává se API KEY (které se zobrazí na M5stack)

Run – spusteni z vývojového prosředí
Download – nahrání do zařízení (možno po té spouštět přímo na zařízení výběrem ze seznamu uložených)

pozn: Je-li v m5core nastaveny údaje pro wifi síťové připojení pak se připojí samo. V opačném případě vytvoří vlastní AccessPoint ke kterému je třeba se připojit (mobil, PC) – do webového prohlížeče zadat lokální adresu z displaye (typicky 192.168.4.1) a přes zobrazenou webocou stránku vybrat místní wifi síť a zadat heslo (ZPIKIV / Int3rn3t_VECI, zpikiv / 78086975)

První program

Dokumentace UIFlow: https://docs.m5stack.com/en/uiflow/uiflow_web



Tři „procesy“ – hlavní, setup (základní nastavení HW), proměnná „color“, nekonečná smyčka “while“ a uvnitř smyčky v každém kroku zvyšování proměnné color o 16 v rozsahu do 256, po té pauza a nastavení barvy pozadí na odstín červené dle hodnoty color.

Dalčá dva „procesy“ (lépe eventy či interrupty) jsou spuštěny po stisku příslušného tlačítka a změní text na tlačítka na obrazovce.

(pozn. z blockly prostředí lze někde lépe dovodit způsob použití python funkcí než z dokumentace)

```
from m5stack import *
from m5stack_ui import *
from uiflow import *
import time

screen = M5Screen()
screen.clean_screen()
screen.set_screen_bg_color(0xb2e310)

color = None
Button = M5Label('label0', x=54, y=36, color=0x000, font=FONT_MONT_30, parent=None)

def buttonA_wasPressed():
  global color
  Button.set_text('Press A')
  pass
btnA.wasPressed(buttonA_wasPressed)

def buttonC_wasPressed():
  global color
  Button.set_text('Press C')
  pass
btnC.wasPressed(buttonC_wasPressed)

color = 0
while True:
  color = (color + 16) % 256
  wait_ms(200)
  screen.set_screen_bg_color((color << 16) | (0 << 8) | 0)

```

Programování v C

Lze např. v Arduino IDE které má podporu pro M5core2, postup instalace např.

M5Stack Arduino IDE Setup in 5 minutes (youtube.com)

příklady projektů (pro M5core (core bez 2))

Use-Cases - Makerfactory Documentation

Python

Python - pravděpodobně jej máte na svém počítači, nebo lze doinstalovat.
Bez vývojového prostředí lze python scripty (soubory s pyhton programem, s příponou*.py) spouštět z příkazového řádku pomocí: python muj_kod.py , případně python3 muj_kod.py

Existuje řada online nástrojů pro testování,
např. Online Python Compiler - online editor (onlinegdb.com)
a stránek s příklady,
např. Learn Python - Free Interactive Python Tutorial

Hallo world

```
print("Hallo world")
```

```
Hallo world
```

Proměnné, podmínka a odsazení

jména proměnných jsou case-sensitive (tj. X není to samé jako x). odsazení bloků je vyžadováno.

```
x = 2
if (x == 1):
    # mezery na zacatku musí byt
    print("x is 1.")
else:
    print("x is not 1")
```

```
x is not 1
```

Typy proměnných, print

```
# Celé číslo
cislo = 10
print("Celé číslo:", cislo)

# Desetinné číslo
desetinne_cislo = 3.14
print("Desetinné číslo:",
desetinne_cislo)

# Textový řetězec
text = "Ahoj, světe!"
print("Textový řetězec:", text)

# Logická hodnota
pravda = True
print("Logická hodnota:", pravda)

# Seznam
seznam = [1, 2, 3, 4, 5]
print("Seznam:", seznam)
```

```
Celé číslo: 10

Desetinné číslo: 3.14

Textový řetězec: Ahoj, světe!

Logická hodnota: True

Seznam: [1, 2, 3, 4, 5]
```

```
# Slovník
slovnik = {"jmeno": "Jan", "vek": 30}
print("Slovník:", slovnik)
```

```
Slovník: {'jmeno': 'Jan', 'vek': 30}
```

```
# Celé číslo
cislo = 10
print("Celé číslo: %d" % cislo)
```

```
Celé číslo: 10
```

```
# Desetinné číslo
desetinne_cislo = 3.14
print("Desetinné číslo: %.2f" %
desetinne_cislo)
```

```
Desetinné číslo: 3.14
```

```
# Textový řetězec
text = "Ahoj, světe!"
print("Textový řetězec: %s" % text)
```

```
Textový řetězec: Ahoj, světe!
```

```
# Logická hodnota
pravda = True
print("Logická hodnota: %s" % pravda)
```

```
Logická hodnota: True
```

```
# Seznam/ (převedený na řetězec)
seznam = [1, 2, 3, 4, 5]
print("Seznam: %s" % seznam)
```

```
Seznam: [1, 2, 3, 4, 5]
```

```
# Slovník (převedený na řetězec)
slovnik = {"jmeno": "Jan", "vek": 30}
print("Slovník: %s" % slovnik)
```

```
Slovník: {'jmeno': 'Jan', 'vek': 30}
```

```
print("\nmulti\n%d*%s" % (cislo,text))
```

```
multi
10*Ahoj, světe!
```

Základní operátory

```
Number = 1 + 2 * ( 1 + 1 ) / 4.0
print(number)          #bezne operace
```

```
2.0
```

```
remainder = 11 % 3     #zbytek po deleni
print(remainder)
```

```
2
```

```
squared = 7 ** 2       #mocnina
print(squared)
```

```
49
```

```
rounded=round(12.43)   #zaokrouhleni
print(rounded)
```

```
12
```

```
root=math.sqrt(16)     #odmocnina
print(root)
```

```
4
```

```
rnd=random.randint(0,255) #nahodne cislo
print(rnd)                #0 az 255
```

```
192
```

```
cele=int(12.45)        #zmena typu
desetinne=float(12)    #zmena typu
```

```
helloworld = "hello" + " " + "world"
print(helloworld)   #scitani retezcu
```
```
hello world
```

```
lotsofhellos = "hello" * 4
print(lotsofhellos) #mocnina retezcu
```
```
hellohellohellohello
```

```
evens = [2,4,6,8]
odds = [1,3,5,7]
all_numbers = odds + evens
print(all_numbers)  #scitani seznamu
```
```
[1, 3, 5, 7, 2, 4, 6, 8]
```

Operace s řetezci/stringy

```
str = "Hello world!"
print(str.index("l")) #kde je l?
print(str.count("l")) #kolikrat l?
print(str[1:5])   #znaky 1..5
print(str[-3:-1]) #od konce
print(str[0:5:2]) #..po 2
print(str[::-1])  #vsechny pozpatku
print(str.upper()) #na velka
print(str.lower()) #na mala
print(str.startswith("Hel")) #začína?
strsplit=str.split("l")  #rozděl
print(strsplit)
```
```
2
3
ello
ld
Hlo
!dlrow olleH
HELLO WORLD!
hello world!
True

['He', '', 'o wor', 'd!']
```

Podmínky/Conditions

```
p1 = True; x=7;
p2 = x==6
print(p2);
if (x==7) and (p1): print("x7 a p1");
if (p1) or (p2): print("p1 nebo p2");

if p2:             #jestlize
  print("p2")
elif x==7:         #anebo
  print("ne p2, x7")
else:              #jinak
  print("ne p2, ne x7")

if not(p2): print("not2")   #not, negace

if (p1):
  if (p2):  #vnorena podminka, odsazeni
    print("p1 a p2")
  else:
    print("p1 a ne p2")
else:
  print("ne p1")
```
```
False
x7 a p1
p1 nebo p2

ne p2, x7

not2

p1 a ne p2
```

Smyčky/Loops - for

```
for x in range(5):  #x postupne 0 az 4 (5-1)
  print(x, end="*") #end=zakonceni
```
```
0*1*2*3*4*
```
pozn. bez end=... co číslo to řádka

```
for x in range(3, 6):  #3 az 5
  print(x, end=" ")
```
```
3 4 5
```

```
for x in range(3, 8, 2): #3 az 7 po 2
  print(x, end=" ")
```
```
3 5 7
```

```
list = [2, 3, 5, "txt"]  #i ruzne typy
for item in list:  #projdi list
  print(item)
```
```
2
3
5
txt
```

```
for x in range(10): #x posupne 0 az 9
  if x % 2 == 0:
    continue     #pokracuje dalsi smyckou
  print(x, end="")
  if x >= 8:
    break;       #ukonci smycku
```
```
13579
```

Smyčky/Loops - while

```
Count=0   #startovni hodnota count
while(count<5):   #dokud count<5
  print(count,end="")
  count +=1

print("\n")        #tisk odradkovani
while True:        #nekonecna smycka
  print(count,end="")
  count += 1
  if count >= 10:  #podminka ukonceni
    break
```
```
01234
```
```
56789
```

Seznamy/Lists

```
Seznam = []             #prazdny seznam
seznam = [6, 5, 4, 3]   #seznam s prvky
print(seznam)     # cely seznam
print(seznam[0])  # První prvek (1)
print(seznam[-2])  #Predposledni prvek
seznam.append(2)   #pridani prvku nakonec
print(seznam)
seznam.remove(5) #odebrani prvku 5 (prvniho)
print(seznam)
seznam.pop(1)    #odebrani prvku na indexu 1
print(seznam)
seznam.sort()    #setrideni seznamu
print(seznam)
seznam.reverse() #obraceni seznamu
print(seznam)
```
```
[6, 5, 4, 3]
6
4
[6, 5, 4, 3, 2]
[6, 4, 3, 2]
[6, 3, 2]
[2, 3, 6]
[6, 3, 2]
```

pozn. v seznamech (na rozdíl od polí) mohou být i proměnné různých typů.

Slovníky/Dictionaries

```
Weight = {}        #prazdny slovnik
weight["John"] = 80
weight["Jack"] = 70
weight["Jill"] = 60

print(weight)                                   {'John': 80, 'Jack': 70, 'Jill': 60}
print(weight.keys())                            dict_keys(['John', 'Jack', 'Jill'])
print(weight.values())                          dict_values([80, 70, 60])

weight = {
 "John" : 80,
 "Jack" : 70,
 "Jill" : 60
}
print(weight["Jack"])                           70
print(weight.get("Jack", "Neznam"))             70
print(weight.get("Lili", "Neznam"))             Neznam

for jmeno,vaha in weight.items():               John vazi 80kg
  print("%s vazi %skg" % (jmeno,vaha))          Jack vazi 70kg
                                                Jill vazi 60kg
if "Jake" in weight:
    print("Jake je ve slovniku")
if "Blabla" not in weight:                      Blabla neni ve slovniku
    print("Blabla neni ve slovniku")
```

Funkce

Funkce jsou (pojmenované) bloky kódu, které provádějí specifické úkoly a mohou být znovu použity v různých částech programu. Funkce může přijímat vstupy (parametry), provádět určité operace a vracet výstup (výsledek). Funkce se používají k modularizaci kódu, což zlepšuje jeho čitelnost a údržbu.

```
def myfunc():  #funckce tisku
  print("Hello",end="_")
  print("World")

myfunc()                                        Hello_World
myfunc()                                        Hello_World
#---
def max(a,b):  #funkce vypoctu maxima dvou cisel
  if (a>b):
    return(a)
  else:
    return(b)

c=max(6,9)
print(c)                                        9

d=max(4,max(5,6))
```

```
print(d)
#---
def text(jmeno,udalost)   #funkce tisku s parametry
  return("Ahoj %s,\nstastne %s\n"%(jmeno,udalost))

print(text("Adame","velikonoce"))


print(text("Evo","vanoce"))


#---
x=41         #globalni promenna
def zvys_x_spatna():
  x=x+1             #zde nastane chyba, pristup
  print(x)          #  ke globalni promenne

def zvys_x_spravna():
  global x          #toto je treba pridat
  x=x+1
  print(x)
```

```
6


Ahoj Adame,
stastne velikonoce

Ahoj Evo,
stastne vanoce


42
```

Moduly a Balíčky (Modules and Packages)
Modul je (externí) soubor s příponou .py, který obsahuje kód Pythonu. Tento kód může zahrnovat funkce, třídy, proměnné a další objekty, které lze importovat a používat v jiných Python skriptech.
Balíček je pak kolekce jednoho nebo více modulů, které jsou organizovány v adresáři. Tento adresář musí obsahovat speciální soubor __init__.py, který označuje, že adresář je balíček. Balíčky mohou také obsahovat dílčí balíčky, což umožňuje hierarchickou strukturu kódu.
Např. Modul: math.py obsahuje funkce jako sqrt(), sin(), atd
Balíček: Adresář math_tools/ obsahuje soubory __init__.py, operations.py, statistics.py, atd.

```
import math                     Importuj(umožni používat) modul math
y=math.sin(0)                   na jeho funkce je třeba se odkazovat se jménem

import math as m                Importuj(umožni používat) modul math, ale pod jiným
y=m.sin(0)                      jménem.

from math import *              Použij všechny prvky z modulu math
y=sin(0)                        takto je možné používat bez odkazu na jméno modulu

from math import sin as sinus   Z math importuj sin ale pod jinym jmenem
y=sinus(0)
```

M5core2 UI (User Interface)

M5core je vybaveno 3 tlačítky, dotykovou obrazovkou s 320*240 body, a zvukovým vstupem/výstupem

Textové a Grafické prvky

(v UIFlow/blockly jsou grafické prvky nabízené v levém svislém menu a příslušné blokové programové struktury pak ve středním sloupci, záložce UI)

Základní inicializace obrazovky

```
from m5stack import *                       Import knihoven
from m5stack_ui import *
from uiflow import *
from m5ui import *

screen = M5Screen()
screen.clean_screen()
screen.set_screen_bg_color(0x00ff00)        color: 0xRRGGBB hex číslo

lcd.print("textovy vypis")                  #textovy vypis na display (pro ladeni)
```

Barva je kódována pomocí 3bytového hexadecimálního čísla 0xRRGGBB kde RR,GG a BB jsou jednotlivé barevné složky červená,zelená a modrá (Red,Green a Blue) které mohou nabývat hodnot 00 až ff (tj. Dekadicky 0-255), 00 je nejnižší a ff nejvyšší. Tedy např. 000000=černá, 00ff00=zelená, 00ffff=modrozelená(cyan),ffffff=bílá.

Textové pole / Label

pro zoabrazení fixního případně uřivatelského textu

```
#Text 'text' na pozici (1,1), černou barvou, fontem velikosti 30:
label0 = M5Label('text', x=1, y=1, color=0x000000, font=FONT_MONT_30)

label0.set_text('novy text')     #zmena textu
label0.set_text_color(0xffffff)  #zmena barvy textu
label0.set_hidden(True)          #skrytí textu
label0.set_hidden(False)         #opětovné zobrazení
label0.set_pos(0, 0)             #presun pozice
```

Grafická Tlačítka / Button

```
#Tlačítko 'text' na pozici (0,50), se sirkou 70 a vyskou 30bodu
#bílé tlacitko s černým textem, fontem velikosti 14
button0 = M5Btn(text='text', x=0, y=50, w=70, h=30, bg_c=0xFFFFFF,
text_c=0x000000, font=FONT_MONT_14)
```

```
button0.pressed(button0_pressed)  #definovani eventu/funkce pro stisk

def button0_pressed():     #event/funkce je li tlačítko stisknuto...
  button0.set_bg_color(0xff0000)   #nastav barvu pozadí na červenou
  button0.set_btn_text('novy text')   #zmen text na tlacitku
```

Fyzická Tlačítka / Button

Pod displayem jsou tři tlačítka, v knihovnách jsou již předdefinovaná jako btnA,btnB a btnC

```
btnA.wasPressed(buttonA_wasPressed) #definovani eventu/funkce pro stisk tl.A

def buttonA_wasPressed():           #event/funkce je li tlačítko stisknuto...
  #akce při stisku tlačítka         #pokud zde neni kod, misto nej pouzijeme
  pass                              #pass = příkaz co nedělá nic
```

Přepínač / Switch

```
switch0 = M5Switch(x=1, y=80, w=70, h=30, bg_c=0xCCCCCC, color=0x000000,
parent=None)

switch0.on(switch0_on)              #definovani eventu/funkce pro zapnuti
switch0.off(switch0_off)            #definovani eventu/funkce pro vypnuti

def switch0_off():                  #event/funkce je li switch vypnut
  pass
def switch0_on():                   #event/funkce je li switch zapnut
 pass

switch0.set_on()                    #programove zapnuti swiche
switch0.set_off()                   #programove vypnuti switche
```

Posuvnítko / Slider

```
slider0 = M5Slider(x=25, y=134, w=70, h=12, min=0, max=100, bg_c=0xa0a0a0,
color=0x08A2B0, parent=None)

slider0.set_range(0, 100)        #virtualni rozsah slideru, 0 az 100 je default
slider0.set_value(50)            #programovy posuv posuvnitka
v=slider0.get_value()            #cteni pozice slideru

slider0.changed(slider0_changed) #definovani eventu/funkce pri zmene

def slider0_changed(value):      #event, je li posuvnitko presunuto na <value>
  pass
```

Zaškrtávátko / Checkbox

```
checkbox0 = M5Checkbox(text='checkbox0', x=144, y=87, text_c=0x000,
check_c=0x000, font=FONT_MONT_18)
```

```
checkbox0.checked(checkbox0_checked)     #definovani eventu/funkce pro zapnuti
checkbox0.unchecked(checkbox0_unchecked) #definovani eventu/funkce pro vypnuti

def checkbox0_checked():                 #event/funkce je li checkbox zapnut
 pass

def checkbox0_unchecked():               #event/funkce je li checkbox vypnut
 pass

checkbox0.set_checked()                  #nastavit zaskrtnuti
```

Grafická primitiva

```
#cerna usecka (z bodu (0,0) do bodu (319,239)
line0 = M5Line(x1=0, y1=0, x2=319, y2=239, color=0x000000, width=1)

#cerny vyplneny obdelnik (na pozici 50,25 s delkou 100 a sirkou 50)
#cerny vyplneny obdelnik s bilym okrajem
rect = M5Rect(50, 25, 100, 50, 0x000000);
rect = M5Rect(50, 25, 100, 50, 0x000000, 0xffffff);
rect.setSize(40, 15, 120, 70)   #zmena rozmeru

#cerny vyplneny kruh (stred 50,50, polomer 25), pripadne s bilym okrajem
circ=M5Circle(50,50, 25, 0x000000);
circ=M5Circle(50,50, 25, 0x000000,0xffffff);

circ.setPosition(80,50)   #presun kruhu na jinou pozici
```

..a další

Výčet zde není vyčerpávající, v uvedených knihovnách existují i další grafické objekty např. graf, možnost vložení obrázku, atd… viz dokumentace.

Touchscreen

```
(x,y) = touch.read()  #souradnice dotyku
x=touch.read()[0]     #x-souradnice dotyku
y=touch.read()[1]     #x-souradnice dotyku
if touch.status():    #dotyk ne/ano - False/True
  #do somenthing
```

Vibromotor

```
power.setVibrationIntensity(50)    #intenzita vibraci (1-100)
power.setVibrationEnable(True)     #zapnuti vibrovani
wait_ms(200);                      #pauza 200ms
power.setVibrationEnable(False)    #vypnuti vibrovani
```

Reproduktor

```
speaker.playTone(440, 2, volume=4)    #play – frekvence, delka tony, hlasitost
```

Příklad aplikace – User Interface

```
#program pro padajci kulicku. Lze nastavit smer pohybu kulicky,vibrace, mozno ji posunout dotykem
from m5stack import *
from m5stack_ui import *
from uiflow import *
from m5ui import *

pos=120;                            #globalni priznak polohy kulicky (y-souradnice)
smer=False;                         #globalni priznak smeru  kulicky dolu/nahoru
vibr=False;                         #globalni priznak vibraci ne/ano
screen = M5Screen()                 #inicializace/smazani obrazovky
screen.clean_screen()
screen.set_screen_bg_color(0x000000)   #cerna obrazovka

circ = M5Circle(240,pos, 20, 0x0000ff);  #modre kulicka (kolecko o polomeru 20bodu)

def slider0_changed(value):          #event pro slider/posuvnik - zmena hodnoty
  col=0x010101*round(255*value/100)  #nova barva - odstin sede (prevod 0-100 na 0x000000 az 0xffffff)
  screen.set_screen_bg_color(col)    #nastaveni barvy

def switch0_off():                   #event/funkce je li switch vypnut
  global smer
  smer=False;                        #nastav smer dolu
  label0.set_text('DOWN')

def switch0_on():                    #event/funkce je li switch zapnut
  global smer
  smer=True;                         #nastav smer nahoru
  label0.set_text('UP')

def checkbox0_checked():             #event/funkce je li checkbox zapnut
  global vibr
  vibr=True;                         #povol vibrace

def checkbox0_unchecked():           #event/funkce je li checkbox vypnut
  global vibr
  vibr=False;                        #zakaz vibrace

def buttonA_wasPressed():            #event/funkce je li tlačítko stisknuto...
  global pos
  pos=120                            #nastav pocici kulicky do stredu obrazovky
  circ.setPosition(240,pos);

slider0 = M5Slider(x=30, y=120, w=70, h=12, min=0, max=100, bg_c=0xa0a0a0, color=0x08A2B0)   #definice posuvniku
slider0.changed(slider0_changed)  #a jeho obsluzneho eventu   - zmena barvy pozadi

switch0 = M5Switch(x=30, y=60, w=70, h=30, bg_c=0xCCCC00, color=0x000040, parent=None)        #definice switche
switch0.on(switch0_on)              #definovani eventu/funkce pro zapnuti switche - smer pohybu
switch0.off(switch0_off)            #definovani eventu/funkce pro vypnuti switche - smer pohybu

label0 = M5Label('DOWN', x=30, y=10, color=0xff0000, font=FONT_MONT_30)  #definice textoveho pole - info o smeru

checkbox0 = M5Checkbox(text='VIB', x=30, y=170, text_c=0xff0000, check_c=0x00ffff, font=FONT_MONT_30)
checkbox0.checked(checkbox0_checked)     #definovani eventu/funkce pro zapnuti - povoleni vibraci
checkbox0.unchecked(checkbox0_unchecked) #definovani eventu/funkce pro vypnuti - zakazani vibraci

btnA.wasPressed(buttonA_wasPressed) #definovani eventu/funkce pro stisk tl.A  - nastaveni stredni pozice kulucky

while(True):
  wait_ms(10);
  (x,y) = touch.read()  #souradnice dotyku
  if (touch.status()and(x>160)):  #je li dotyk v prave polovine obrazovky
    pos=y;   #uprav y-pozici kulicky na pozici dotyku

  move=False;     #priznak, pohybuje li se kulicka
  if smer:        #pada li nahoru
    if (pos>20):  #a jeste nedopadla
      pos=pos-1   #posun nahoru
      move=True;
  else:           #pada li dolu
    if (pos<220): #a jeste nedopadla
      pos=pos+1   #posun dolu
      move=True;
  circ.setPosition(240,pos); #vykresli (zmen pozici na novou) kulicku

  if (move)and(vibr):  #pokud se kulicka pohubuje a jsou povoleny vibrace
    power.setVibrationEnable(True)  #vibruj
  else:
    power.setVibrationEnable(False) #jinak nevibruj
```

M5corore2 – interní sensory

Mikrofon

```
import MicrophonePDM as MIC
MIC.begin(pin_ws=0,pin_data=34,sample_rate_hz=16000,buffer_length_ms=1000,
block_length_ms=100)    #inicalizace snimani
rms = MIC.getRMS()      #sila signalu z mikrofonu
```

IMU (akcelerometr, gyroskop)

```
import imu
imu0 = imu.IMU()    #inicializace IMU
(t,a,b) = imu0.ypr    #naklon ve dvou osach (a,b)
(x,y,z) = imu0.acceleration #zrychleni x,y,z[G] (x=nahoru/dolu,y=levo/pravo,z=zem)
(q,w,e) = imu0.gyro         #otaceni [°/s]
```

Power info

```
c = power.getChargeState()    #indikace nabijeni ne/ano
u = power.getBatVoltage()     #napeti baterie
p = power.getBatPercent()     #procenta nabiti baterie
i = power.getBatCurrent()     #spotreba z baterie
t = power.getPmuInTemp()      #interni teplota systemu
```

M5corore2 – System

RTC

```
import time
rtc.settime('ntp', host='cn.pool.ntp.org', tzone=8)  #synchronizace z NTP serverem

(y,m,d,w,h,n,s) = rtc.datetime()   #cteni casu rok,mesic,den,denvtydnu,hod,min,sec
```

Reset

```
import machine
machine.reset()           #reset celeho zarizeni
```

Watchdog

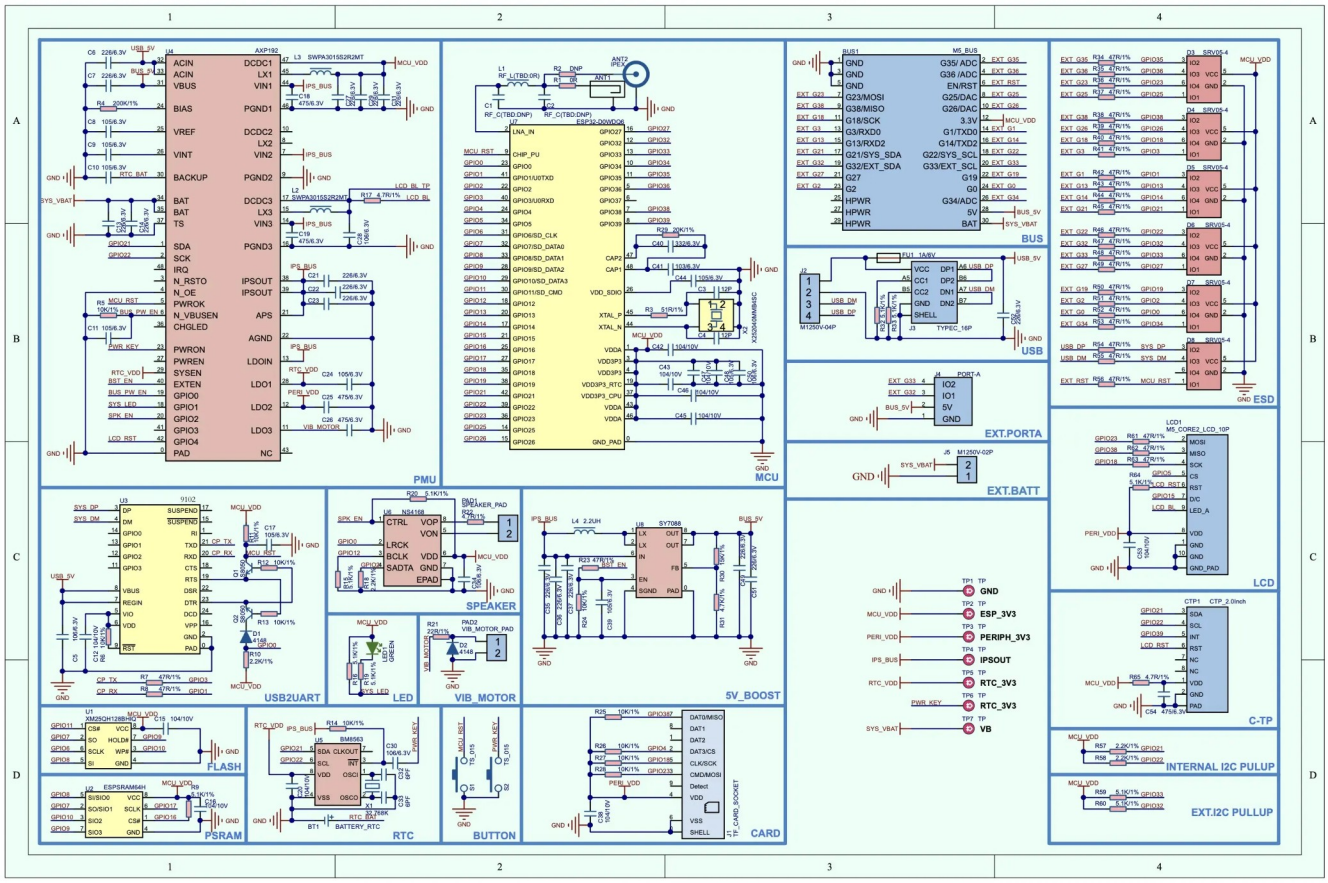
Watchdog v embedded systémech je bezpečnostní mechanismus, který monitoruje správnou funkci systému a v případě zjištění chyby nebo zablokování systému provede reset. Tento mechanismus je často implementován jako samostatný hardware nebo jako součást mikrokontroléru.
Watchdog funguje na principu časovače, který musí být pravidelně resetován softwarem. Pokud software selže a neprovede reset včas, watchdog vyvolá reset systému, čímž se obnoví jeho funkčnost

```
from machine import WDT
wdt = WDT(timeout=2000)     #nastaveni timeoutu watchdogu (cas do resetu [ms])
#…
wdt.feed()                  #reset casovace watchogu (start citani casu)
```

M5core2 pinout

[m5-docs (switch-science.com) https://docs.m5stack.switch-science.com/en/core/core2](https://docs.m5stack.switch-science.com/en/core/core2)

GND	ADC	G35	
GND	ADC	G36	
GND	RST	EN	
G23	MOSI	DAC	G25
G38	MISO	DAC	G26
G18	SCK	3.3V	
G3	RXD0	TXD0	G1
G13	RXD2	TXD2	G14
G21	intSDA	intSCL	G22
G32	PA_SDA	PA_SCL	G33
G27	GPIO	GPIO	G19
G2	I2S_DOUT	I2S_LRCK PDM_CLK	G0
NC		PDM_DAT	G34
NC		5V	
NC		BAT	



M5Stack	
LCD	
GPIO23	MOSI
GPIO18	SCK
GPIO14	CS
GPIO27	R/S
GPIO33	RST#
GPIO32	Backlight
Button	
GPIO37	Btn C
GPIO38	Btn B
GPIO39	Btn A
SDCard	
GPIO23	SDDI
GPIO19	SDDO
GPIO18	SDCLK
GPIO4	SDCS
Speaker	
GPIO25	INP

M5Core2 Bus			M5Stack Bus						
GND			GND	1	2	ADC1	GPIO35	GPIO35	ADC
GND			GND	3	4	ADC2	GPIO36	GPIO36	ADC
GND			GND	5	6	RESET	EN	EN	RST
MOSI	GPIO23	GPIO23	MOSI	7	8	AUDIO_L	GPIO25	GPIO25	DAC
MISO	GPIO38	GPIO19	MISO	9	10	AUDO_R	GPIO26	GPIO26	DAC
SCK	GPIO18	GPIO18	SCK	11	12	3.3V			3.3V
RXD0	GPIO3	GPIO3	RXD1	13	14	TXD1	GPIO1	GPIO1	TXD0
RXD2	GPIO13	GPIO16	RXD2	15	16	TXD2	GPIO17	GPIO14	TXD2
intSDA	GPIO21	GPIO21	SDA	17	18	SCL	GPIO22	GPIO22	intSCL
PA_SDA	GPIO32	GPIO2	IO6	19	20	IO7	GPIO5	GPIO33	PA_SCL
GPIO	GPIO27	GPIO12	IIS_SCLK	21	22	IIS_WS	GPIO13	GPIO19	GPIO
I2S_OUT	GPIO2	GPIO15	IIS_OUT	23	24	IIS_MCLK	GPIO0	GPIO0	2S_LRCLK
NC			HPWR	25	26	IIS_EN	GPIO34	GPIO34	PDM_DAT
NC			HPWR	27	28	5V			5V
NC			HPWR	29	30	BATTERY			BAT

GPIOXX same
GPIOXX M5Stack only
GPIOXX M5Core2 only
GPIOXX moved

Flash 1)	
GPIO6	SCLK
GPIO7	SO
GPIO8	SI
GPIO9	HOLD#
GPIO10	WP#
GPIO11	CS#

1) M5Stack and M5Core2

PSRAM 2)	
GPIO17	SCLK
GPIO7	SO/SIO1
GPIO8	SI/SIO0
GPIO9	SIO3
GPIO10	SIO2
GPIO16	CS#

2) Also present in M5Stack Fire

M5Core2	
LCD	
GPIO23	MOSI
GPIO38	MISO
GPIO18	SCK
GPIO5	CS
GPIO15	D/C
AXP_IO4	Reset
AXP_DCDC3	Backlight
Touch	
GPIO21	SDA
GPIO22	SCK
GPIO39	INT
SDCard	
GPIO23	CMD/MOSI
GPIO38	DAT0/MISO
GPIO18	CLK/SCK
GPIO4	DAT3/CS
Speaker	
GPIO0	LRCK
GPIO12	BCLK
GPIO2	SADTA
AXP_IO2	Enable
RTC/AXP	
GPIO21	SDA
GPIO22	SCK
SYS_LED	
AXP_IO1	on

Periferie v mikrokontrolerech pro připojení senzorů

Tří základní způsoby obsluhy periférií

- Pooling – dotazující. blokující čas procesoru

- Interrupt – Periférie pro požadavku na pozornost (např. příjem dat, chyba, stanovený vzor, úroveň atp) vyvolá přerušení. Alternativně použijeme přerušení od časovače ve kterém obsluhujeme periférii.

- DMA – periférie předává data přes DMA řadič kterým jsou data rovnou ukládány do paměti. Volitelně interrupt po dokončení/v polovině atp. Při požadavku na zpracování dat.

Příklad jednotlivých přístupů (pseudokód) prezentovaný na úloze výstupu signálu o frekvenci 10kHz.

Pooling

```
p0 = Pin(0, Pin.OUT);
While(True):
  p0.on()  # set pin to 1
  delayus(50) #50us pause
  p0.off() # set pin to 0
  delayus(50)
```

Jednoduchost, bez vazby na periférie procesoru

Plně vytěžuje procesor, nemusí být zajištěno přesné časování, obtížná modifikace kódu

Interrupt

```
po=0;
p0=Pin(0,Pin.OUT,value=po)
timer1_set(50)

def timer1():
  p0.value(po)
  po=1-po  #change 0<->1
```

Kód je spuštěn jen při požadavku na akci

Dochází k přerušení hlavního kódu, je třeba řešit přístup k datům (synchronizaci)

DMA

```
p0 = Pin(0, Pin.OUT);
po=[0,1]
dma_settimer(50);
dma_pinout(pout,2,restart);
```

Po spuštění pracuje sám buď bez potřeby procesorového času nebo s blokovým zpracováním dat

Nehodí se pro komplexní úlohy, např. kde neznáme délku dat

Periférie / MicroPython (python for m5core2)

(Vývojové prostředí UIflow: https://flow.m5stack.com/)

m5-docs (m5stack.com) https://docs.m5stack.com/en/mpy/official/machine

GPIO

GPIO = General purpose input-output.

Vstup: Floating, With pull-up/down. Napěťové úrovně. Hystereze (ST = Schmith trigger).

Výstup: Dvoustavový. Push-pull (PP) / OpenCollector OpenDrain (OC). Napěťové úrovně. Maximální proud. Rychlost.

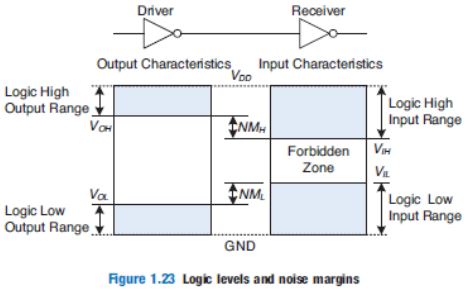
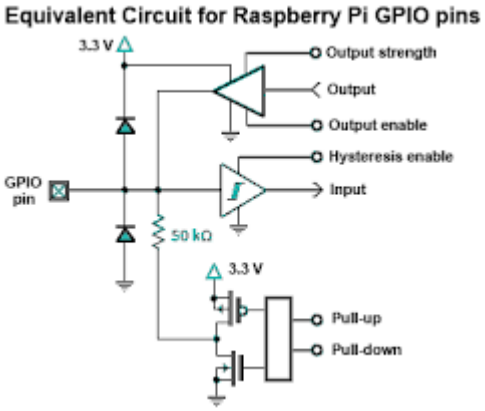


Figure 1.23 Logic levels and noise margins




```
from machine import Pin

p0 = Pin(32, Pin.OUT)    # create output pin on GPIO32
p0.on()                  # set pin to "on" (high) level
p0.off()                 # set pin to "off" (low) level
p0.value(0)              # set pin to off/low/log.0
p0.value(1)              # set pin to on/high/log.1

p2 = Pin(32, Pin.IN)     # create input pin on GPI32
print(p2.value())        # get value, 0 or 1 – read input

p4 = Pin(32, Pin.IN, Pin.PULL_UP) # enable internal pull-up resistor
p5 = Pin(32, Pin.OUT, value=1) # set pin high on creation
```

Pozor – číslování pinů může být v různých zařízeních/vývojových prostředích (i založených na stejném procesoru) zcela jiné a nemusí mít vazbu na číslování pinů procesoru.

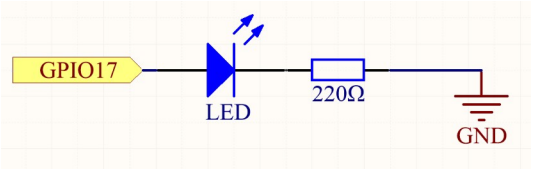
GPIO jako výstup – připojení periférií

Log.úroveň 0/1 → odpovídající napětí, ale omezený proud (typicky třeba max. 4,10,20mA)

Co lze například přípojit? (kromě vstupů dalsích logických obvodů)

LED dioda

LED diody využívají PN přechod k emitování světla (elektroluminiscence). Řízení LED viz její VA charakteritistika. Svítivost je (přibližně) úměrná příkonu(proudu).



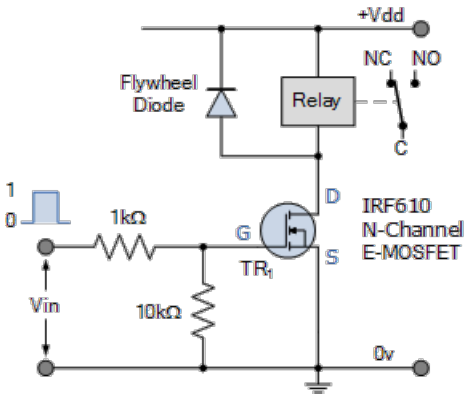
Pozor - nepřipojovat bez omezujícího odporu.

R = (Un-Ur) / I

(U=výstupní napětí gpio, Ur=napětí diody, I=požadovaný proud/svítivost, typicky 2 až 20mA dle typu diody.

Pozor, pro 3.3V už některé diody nemusí mít dostatečně nízké Ur (modré, bílé) aby byli přímo připojitelné na gpio s dostatečným jasem. Pozor na maximální výstupní proud pinu (nepřekročit).

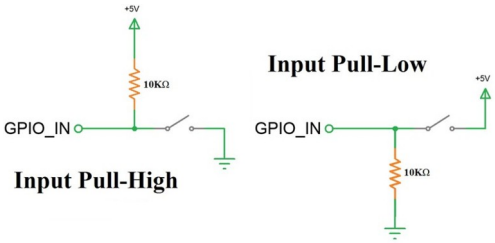
Relé



Mechanický spínací prvek, cívka (elekromagnet) –> magneticky ovládaný spínač/přepínač – poskytuje galvanické oddělení, může spínat výšší proudy/napětí.

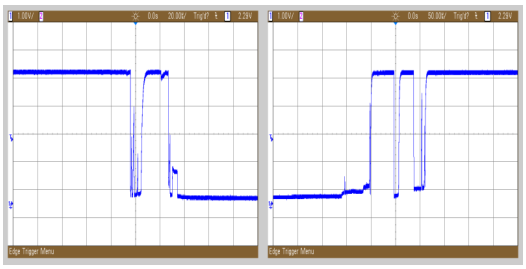
Cívka relé potřebuje k sepnutí obvykle vyšší proud a napětí než poskytne GPIO, takže je nutné spínat relé přes pomocný prvek, typicky tranzistor.

GPIO jako vstup – připojení periférií



Detekce vstupního napětí v daných intervalech (log. 0/1). může sloužit jako vstup z dalsích logických sytémů, jako dvou stavový vstup (např. vstup z PIR detektoru (detekce pohybu osob))

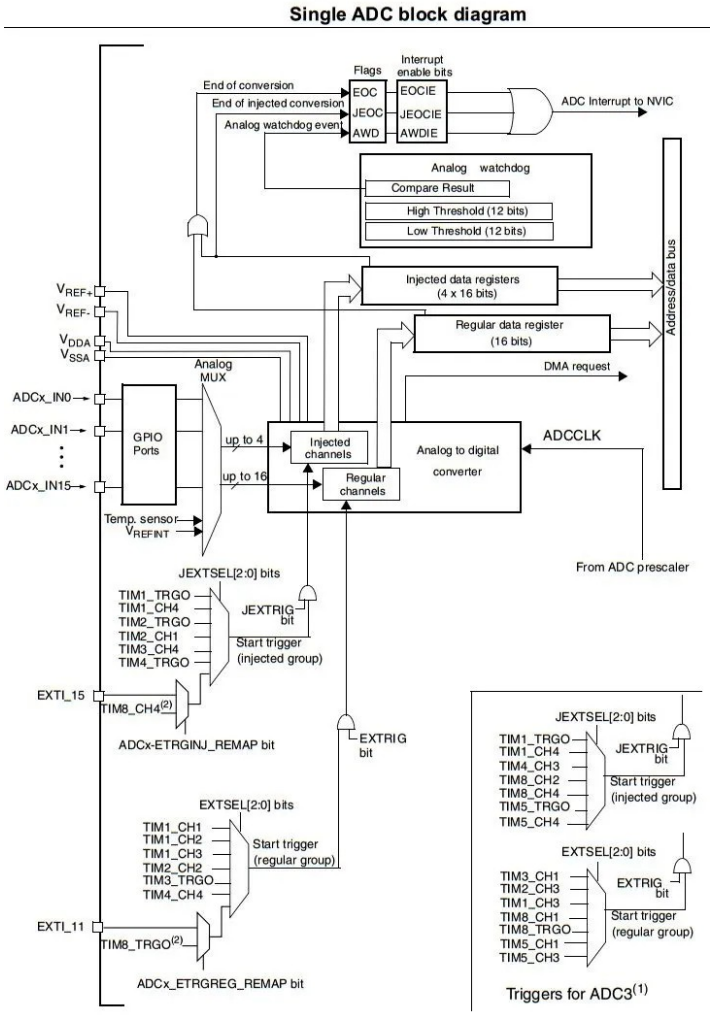
Použití jako vstupu z **tlačítek/spínačů** – viz obrázek – jedna úroven signálu je definována externím (nebo vnitřním pullup/down) odporem, druhá úroveň vznikne stisknutím tlačítka.



Pozor při stisku tlačítka (mechanický prvek) dochází k okamžiku stisku/puštění/přepnutí k zákmitu signálu – viz obr. To je třeba programově ošetřit. (tyto jevy jsou typicky <1ms)

ADC

ADC (analog-digital converter) – je obvod převádějící vstupní analogovou hodnotu na na digitání – číselnou které je uměrná vstupní analogové hodnotě - typickyvstupnímu napětí. Vstupy do adc jsou typicky připojeny na vstuoní piny, ale AD převodník může dále mít speciální interní vstupy jako je teplota, napětí záložní baterie, referenční napětí, vstup z interního operačního zesilovače atp.



ADC obecně: ADC vstupy – mají definovaný napěťový rozsah, mohou být jednoduché (Single edge) nebo diferenciální (Diferential) (tj. vyhodnocuje/převádí se rozdíl napětí mezi dvěmi vstupy). Mohou mít na vstupu zesilovač (OpAmp, někdy programovatelný). Obvykle bývá na vstupu Multiplexer - tj. analogové vstupy se postupně připojují na převodník, nečtou se najednou. To může být někdy na závadu, takže některé microcontrolery mají více převodníků a pak je možné číst více vstupů najednou / synchronně).

Z elektrického hlediska se referenční napětí pro převodník může odvozovat od napájení, od interní reference nebo od externích vstupů Vref+/Vref- (VDDa,VSSa). Na vstupu adc je typicky Sample&Hold obvod – u připojeného vstupu je třeba dodržet minimální vstupní impedanci (input source resistance).

Časování převodu adc je typicky odvozeno od interních hodin procesoru, nebo od speciálního interního volně běžícího časovače (nižší spotřeba, nižší přesnost – možnost běhu při sleep režimu). Konverzi je možné zahájit SW ale také pomocí externího vstupu (triggeru).

Konverze může být jednoduchá/jednokanálová (single conversion) nebo vícekanálová (N-kanálů postupně, scan) a nebo průběžná (continuous) kdy po dokončení jedná sekvence převodu se automaticky opět nastartuje.

Konec konverze je možné programově testovat nebo využít přerušení a nebo použít k uložení výsledků DMA)

Použití ADC pro m5core2:

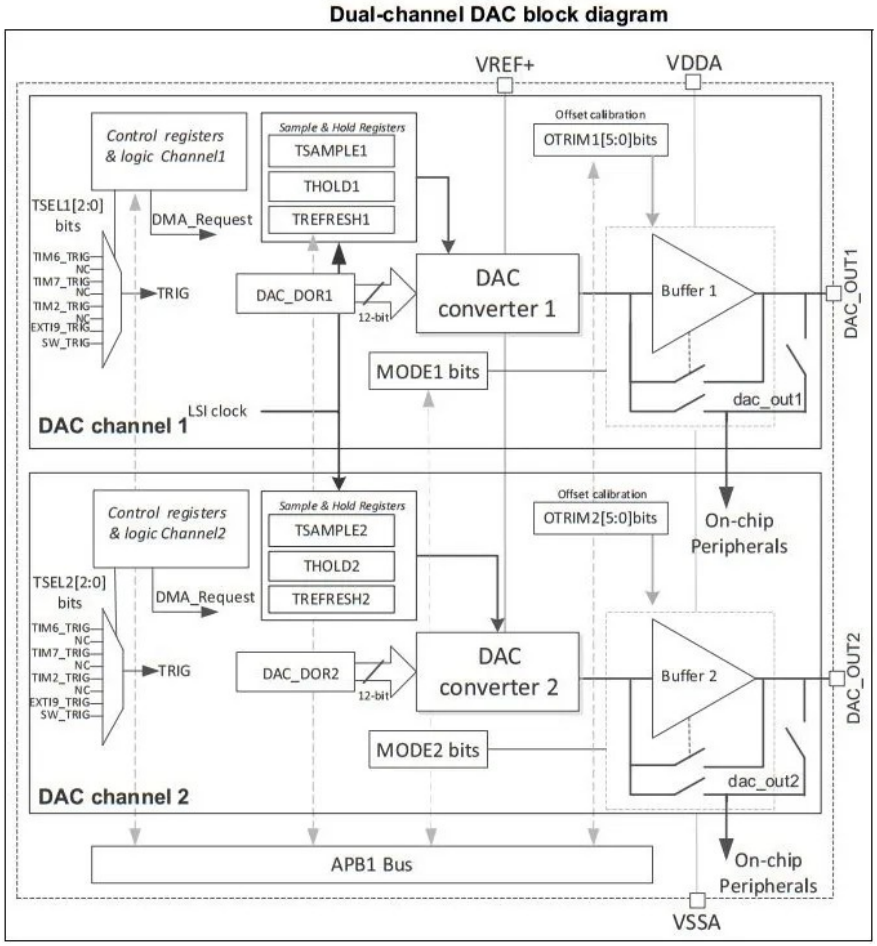
```
from machine import ADC

adc = ADC(Pin(32))          # create ADC object on ADC pin
adc.read()                  # read value, 0-4095 range 0.0v - 1.0v

adc.atten(ADC.ATTN_11DB)    # set 11dB attenuation (range 0.0v – 3.6v)
                            # options: 0DB,2_5DB,6DB,11DB
adc.width(ADC.WIDTH_12BIT)  # set 12 bit return values (default)
                            # options: 9BIT,10BIT,11BIT,12BIT
v=adc.read()                # read value using the new configuration

printf(v*3.6/4096);         #print measured voltage
```

DAC

DAC (digital-analog converter) – Ovod konvertující číselnou hodnotu v interním registru na analogový, typicky napěťový signál. Výstup DAC je možné typicky připojit na výstupní pin. Někdy je ho též možné interně propojit s interním komparátorem nebo Operačním zesilovačem (OpAmp) na jehož druhý vstup je přivedena Analogový hodnota ze vstupního pinu.

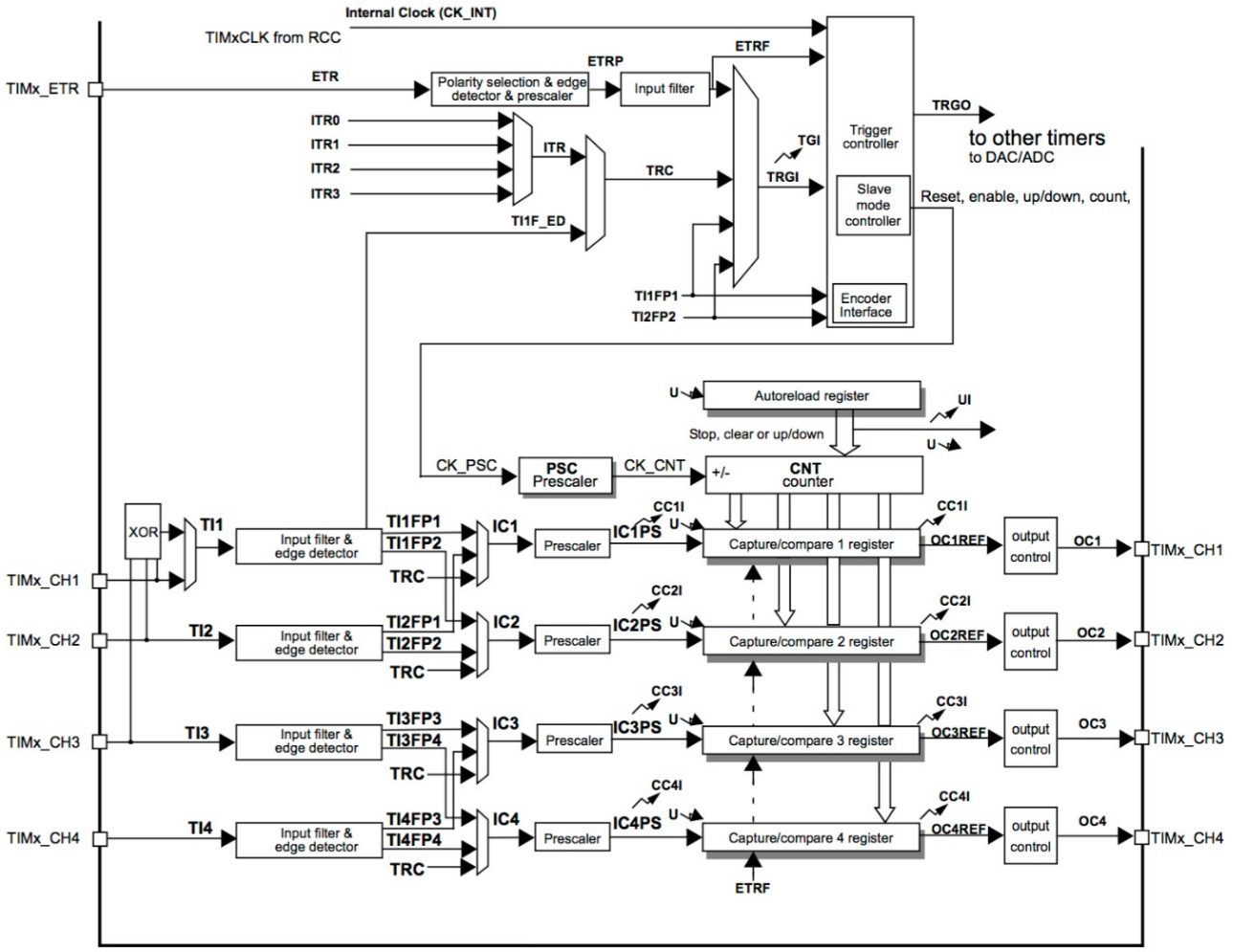
Použití DAC pro m5core2:

```
import machine

dac0 = machine.DAC(25)      #DAC na pinu 25  (dalsi mozny 26,jako dac1)
dac0.write(128)             #zapis hodnotu na vystup (0-255 → 0-3.3V)
dac0.freq(1800)             #gneruj vystupni signal o frekvenci 1800
dac0.stopwave()             #zastav generovani
dac0.waveform(1800, dac0.SINE, duration=200, scale=0, offset=0, invert=0)
                             #generuj komplikonvanejsi vyst.signal
```



Timers (Counters, PWM)

Časovače/Čítače mohou být realizovány SW nebo HW. Slouží pro přesné časování událostí, odměřování času, pro čítání událostí, nebo pro generování (přesně časovaného) signálu.



Timer mode

Čítač slouží pro měření času nebo pro generování přerušení – buď po daném čase nebo periodicky se opakujícího.

Použití timeru pro m5core2:
(pozn: M5core ma hardwarové i softwarové timery – harwarové jsou přesnější, navázané na timery ale tedy limitované dostupným hardwarem (omezený počet) a jejich použití může kolidovat s PWM. Příklady níže jsou sofwarové alarmy, kterých je možné definovat libovolný počet)

```
import machine

@timerSch.event('timer1')   #event/interrupt/udalost pro timer1*
def ttimer1():
 #…                        #zde kod pro timer1
 pass

#spusteni timer1*, periodicky, perioda=100ms – tj. Po 100ms volan event
```

```
#(*timer1 muze byt nahrazovno jinym jmenem, napr. kuku)
timerSch.run('timer1', 100, 0)
#zastaveni timeru1
timerSch.stop('timer1')

#spusteni jednorazove udalosti za 100ms (jednorazove - posl.parametr je 1)
timerSch.run('timer1', 100, 1)
```

Input mode, Catch mode

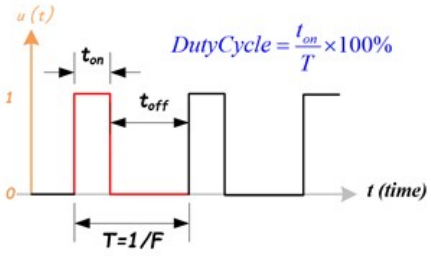
Vstup časovače je spojen s externím vstupem – umožňuje buď přímé (po)čítání vnějšího signálu a nebo odměřování času na jeho základě.

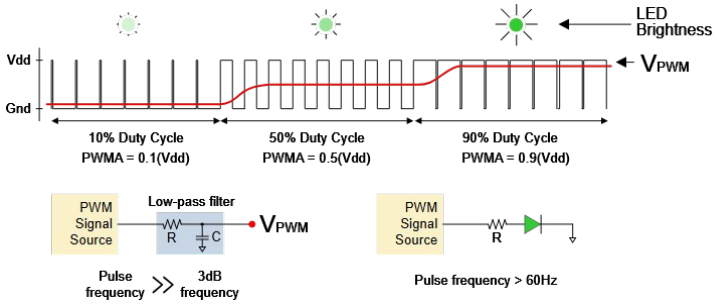
PWM mode

Čítač je nastaven do módu generování pwm signálu – perioda(frekvence) výstupního signálu je typicky fixní a měníme střídu signálu (tj. místo překlopení výstupu uvnitř periody pomocí změny komparačního registru).

Klíčové parametry, které definují PWM signál, jsou
– perioda [s] (frekvence [Hz] = 1/perioda)
- střída (duty cycle) což je poměr doby, kdy je signál v zapnutém stavu, k celkové době jednoho cyklu. Vyjadřuje se obvykle v procentech, tj. například 25% střída znamená, že signál je zapnutý čtvrtinu času a vypnutý tři čtvrtiny
- amplituda - hodnota napětí nebo proudu, kterou signál dosahuje v zapnutém stavu

Pomocí PWM lze řídit výkon dodávaný do elektrické zátěže pomocí digitálního signálu, což může sloužit například pro regulaci jasu LED nebo rychlosti motoru, případně jako náhrada DAC.



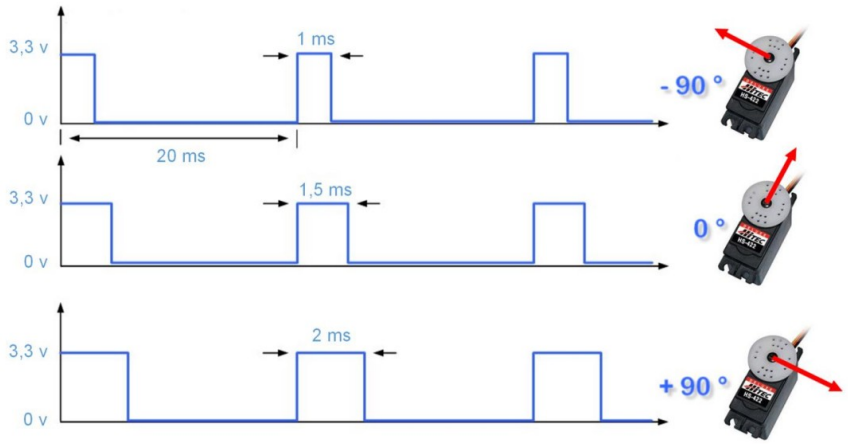


Použití PWM výstupu pro m5core2:

```
import machine

#inicializace pwm na pinu 26
#s frekvenci 10kHz, stridou 50% na citaci 0 (lze alternativne 3)
PWM0 = machine.PWM(26, freq=10000, duty=50, timer=0)

PWM0.freq(50)    #(pře)nastaveni frekvence 50Hz
PWM0.duty(0)     #(pře)nastaveni stridy
PWM0.pause()     #pozastaveni pwm
```

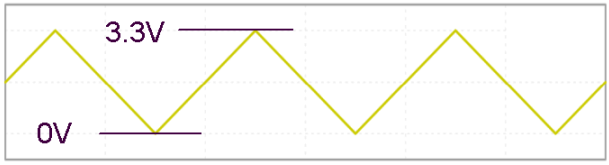
```
PWM0.resume()    #znovuspusteni pwm
```

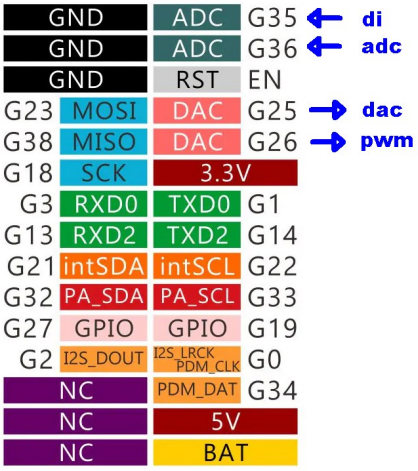
Výhoda PWM oproti DAC může být ve vyšší odolnosti proti rušení (digitální vs analogový signál). Existuje řada zařízení jejichž funkci lze externě ovládat pomocí PWM signálu. Jedním příkladem jsou serva, kdy bežná serva lze ovládat PWM signálem o frekvenci 50Hz (tj. Perioda 20ms) a střídou v rozsahu 5-10% (tj. Aktivní signál 1 až 2ms = poloha -90° až 90°).



Příklad užití GPIO - program

Na DAC (a PWM) budeme generovat trojuhelníkový signál postupně v rozsahu 0-3.3V (napájecí napětí). Výstup DAC můžeme připojit ke vstupu ADC, kde budeme číst hodnotu analogového signálu a digitálnímu vstupu di, kde budeme sledovat kdy dochází ke změně log0 a log1 a automaticky určíme rozhodovací úroveň logického vstupu (cca 1.7V) .





```
#GPIO testovaci program - pilovy signal na DAC a PWM,  cteni ADC a DI
#automaticke vyhodnoceni napetive urovne pro log.0 a 1
from m5stack import *
from m5stack_ui import *
from uiflow import *
from machine import Pin
from machine import ADC
import machine

screen = M5Screen()
screen.clean_screen()
screen.set_screen_bg_color(0xFFFFFF)

#Fix Texts on screen
labeltdo = M5Label('Dout', x=10, y=10, color=0x000000, font=FONT_MONT_30)
labeltdi = M5Label('Din',  x=10, y=40, color=0x000000, font=FONT_MONT_30)
labeltai = M5Label('Ain[V]',  x=10, y=70, color=0x000000, font=FONT_MONT_30)
labeltao = M5Label('Aout[V]', x=10, y=100, color=0x000000, font=FONT_MONT_30)
labeltl0 = M5Label('log0[V]', x=10, y=130, color=0x000000, font=FONT_MONT_30)
labeltl1 = M5Label('log1[V]', x=10, y=160, color=0x000000, font=FONT_MONT_30)
#variable (non-fix) text on screen, measured values
labeldo = M5Label('---', x=160, y=10, color=0x000000, font=FONT_MONT_30)
labeldi = M5Label('---', x=160, y=40, color=0x000000, font=FONT_MONT_30)
labelai = M5Label('---', x=160, y=70, color=0x000000, font=FONT_MONT_30)
labelao = M5Label('---', x=160, y=100, color=0x000000, font=FONT_MONT_30)
labell0 = M5Label('---', x=160, y=130, color=0x000000, font=FONT_MONT_30)
labell1 = M5Label('---', x=160, y=160, color=0x000000, font=FONT_MONT_30)

#variable initialisation
adir=1   #direction -1 down, 1 up
ao=0     #analog out 0..255
wo=0;    #pwm out 0-100
do=0;    #digital out 0/1
di=0;di0=0; #digital in actual,last 0/1
ai=0;    #analog in 0-4095

#gpio initialisation
pi = Pin(35, Pin.IN)       #create input pin on GPIO35
adc = ADC(Pin(36))         # create ADC object on ADC pin
adc.atten(ADC.ATTN_11DB)   #range 0-3.6V
dac = machine.DAC(25)      #dac on gpio25
pwm = machine.PWM(26, freq=50, duty=0, timer=0)  #init pwm 50Hz on gpio26
po = Pin(33, Pin.OUT)      #create output pin on GPIO33


@timerSch.event('timer')  #event/interrupt/udalost pro timer
def ttimer1():
  global do
  do=1-do;                #0->1, 1->0   (change state)
  po.value(do)            #set digital output  - change output state

timerSch.run('timer', 1000, 0)  #1[s] timer (precision is 1s+(5-25)ms)

#main loop
while(True):
  ao+=adir;               #update Analog output value (change in direction)
  wo=ao*100/255;          #transf. analog range to pwm range (0-255 to 0-100)
  if (ao>=255): adir=-1   #change direction down (max is reached)
  if (ao<=0): adir=1      #change direction up (min is reached)
  screen.set_screen_bg_color(0x808000+0x010101*int(ao/2))
  dac.write(ao)           #analog output pin set
  pwm.duty(wo)            #pwm pin set duty cycle
  wait_ms(10)             #wait
  di0=di;                 #store previous state of di
  di=pi.value()       #read digital input state (0/1)
  ai=adc.read();      #read analog intput
  aov=ao/256*3.3;     #analog out voltage (conversion from 0-255 to 0-3.3V)
  aiv=ai/4096*3.3;    #analog input voltage (conversion from 0-4095 tp 0-3.3V)
  if (di0==0)and(di==1): labell1.set_text("%0.2f"%aov) #0->1 transition, log.1
  if (di0==1)and(di==0): labell0.set_text("%0.2f"%aov) #1->0 transition, log.0
  labeldo.set_text(str(do))     #zmena textu
  labeldi.set_text(str(di))     #zmena textu
  labelai.set_text("%0.2f"%aiv) #zmena textu
  labelao.set_text("%0.2f"%aov) #analog out in [V], formated 2dec.point
```

Zpracování dat ze sensorů

Již částečne řešeno (deboucing u tlačítek).

Jednoduchá filtrace dat

Průměr - průměr z N-vzorků

Klouzavý průměr (Simple Moving Average) – Průměr z N posledních vzorků

Lineární Filtr, základní - Dout = Din*(1-f) + D(out-1)*f (f je v rozsahu <0;1), 0=nefiltruje, 0.999.. maximální filtr) – pro velký filtr pomalá reakce na skokové změny vs malý filtr přílišná reakce na špičky

Nelineární filtry – **medián** - odfiltruje odlehlé hodnoty, obvykle se používá filtrace jen na několik hodnot (3 či 5). Pro více hodnot případně nezapomenout na efektivitu. Quicksort O(n.log(n)) → Quickselect O(2n). (pozn. median = setřídění vzorků, vezme se prostřední)

Převod na dvoustavovou hodnotu

Převod spojitého/vícestavového signálu na dvoustavovou hodnotu (ne/ano).

Prahování

Prostá rozhodovací úroveň

If (signal>treshold) then result=log1 else result=log0

Problém – hodnoty blízko rozhodovací úrovně – zákmity

Prahování s hyterezí (Schmitt trigger)

Dvě rozhodovací úrovně – jistá log1 – treshold1 a jistá log.0 – treshold0.

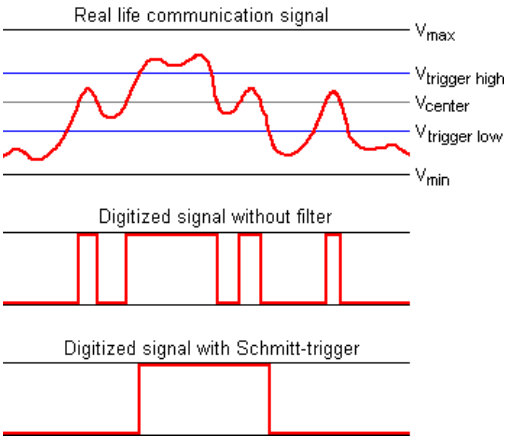
Hystereze = rozdíl treshold1-treshold0 (platí že treshold1>=treshold0)

pseudoprogram (algoritmus):

if signal>=treshold1 then result=log1

else if signal<treshold0 then result=log0

(else result=result)



3D sensory - akcelerometr

sensory poskytující 3D (3 osovou) informaci – např. akcelerometr.

Platí (pro ustálený statický stav na povrchu země) že velikost vektoru je konstanta tj. sqrt(X^2+Y^2+Z^2) = 1 a tento vektor směřuje směrem do středu země.

Typicky jsou hodnoty kalibrovány v [G] (1G=zemská přitažlivost na potvrchu země, cca 9.81m/s^2)

Pozor – hodnoty vykazují mírný offset/posun, který se typicky může měnit s teplotou. (tj. reálných 0G bude třeba -0.01G atp)

3D sensory – gyroskop

Detekuje otáčení kolem 3 os. Bez pohybu je hodnota ideálně nulová – opět pozor hodnoty vykazují mírný offset/posun, který se typicky může měnit s teplotou.

Příklad – filtrace dat z IMU

Program měří a zobrazuje data z IMU (zrychlení, otáčení, náklon) – měřené (nahoře) a filtrovnané (dole). Pomocí posuvnítka je možné nastavit „sílu“ lineárního filtru F (od 0=bez filtrace, 0.5=průměr staré a nové hodnoty, do 0.999 = největší filtrace). Pro představu, postupně každá další nastavitelná hodnota znamená filtrace cca dvojnásobnou filtraci oproti předchozí)

```
from m5stack import *
from m5stack_ui import *
from uiflow import *
import imu

screen = M5Screen()
screen.clean_screen()
screen.set_screen_bg_color(0xFFFFFF)

labela = M5Label('---', x=10, y=10, color=0x000000, font=FONT_MONT_30)
labelg = M5Label('---', x=10, y=40, color=0x000000, font=FONT_MONT_30)
labeli = M5Label('---', x=10, y=70, color=0x000000, font=FONT_MONT_30)

slider0 = M5Slider(x=150, y=110, w=150, h=12, min=0, max=10, bg_c=0xa0a0a0, color=0x08A2B0,
parent=None)
labelf = M5Label('---', x=10, y=100, color=0x000000, font=FONT_MONT_30)

labelaf = M5Label('---', x=10, y=130, color=0x000000, font=FONT_MONT_30)
labelgf = M5Label('---', x=10, y=160, color=0x000000, font=FONT_MONT_30)
labelif = M5Label('---', x=10, y=190, color=0x000000, font=FONT_MONT_30)

#filter: f=konstanta filtru (0-1), old=filtrovana(predchozi_ hodnota, new=nove zmerena hodnota
# vystup: filtrovana hodnota (nova)
def filter(f,old,new):
  return(old*f+new*(1-f))

imu0 = imu.IMU()    #inicializace IMU

#nacteni startovacich hodnot
(t,af,bf) = imu0.ypr    #naklon ve dvou osach
(xf,yf,zf) = imu0.acceleration #zrychleni x,y,z[G] (x=nahoru/dolu,y=levo/pravo,z=zem)
(qf,wf,ef) = imu0.gyro         #otaceni [°/s]

while(True):
  #reading IMU
  (t,a,b) = imu0.ypr    #naklon ve dvou osach a,b
  (x,y,z) = imu0.acceleration #zrychleni x,y,z[G] (x=nahoru/dolu,y=levo/pravo,z=zem)
  (q,w,e) = imu0.gyro         #otaceni [°/s]
  #show measured values (unfiltered)
  labela.set_text("A :%+0.2f %+0.2f %+0.2f"%(x,y,z)) #zrychleni
  labelg.set_text("G :%+0.2f %+0.2f %+0.2f"%(q,w,e)) #otaceni
  labeli.set_text("I :%+0.2f %+0.2f"%(a,b))         #naklon
  #slider -> filter constant
  f=slider0.get_value() #slider value
  f=1-1/2**f; #conver slider to filter value (non linear)
  labelf.set_text("F:%0.3f"%(f)) #show filter value
  #filtering
  xf=filter(f,xf,x);yf=filter(f,yf,y);zf=filter(f,zf,z)
  qf=filter(f,qf,q);wf=filter(f,wf,w);ef=filter(f,ef,e)
  af=filter(f,af,a);bf=filter(f,bf,b);
  #show filtered values
  labelaf.set_text("Af:%+0.2f %+0.2f %+0.2f"%(xf,yf,zf)) #zrychleni filtrovane
  labelgf.set_text("Gf:%+0.2f %+0.2f %+0.2f"%(qf,wf,ef)) #otaceni filtrovane
  labelif.set_text("If:%+0.2f %+0.2f"%(af,bf))           #naklon filtrovany
  wait_ms(5)         #wait
```

Komunikační sběrnice sériové

	RS232	RS485	CANbus	I2C	SPI	1wire	USB	
Norma	EIA-232	EIA-485	ISO 11898	Philips		maxim.com	usb.org	
Typ komunikace	p2p	m-point s/m-master	m-point	m-point s/m-master	m-point S-master	M-point s-master	P2p (m-point) s-master	p=point s=single m=multi
Duplex?	duplex	half	half	half	duplex	half	half	
Datových vodičů typ.značení	2 (++) RX/TX	2 A/B	2 H/L	2 SCL/SDA	3 (+1n) SCK/ SDO/SDI (+CE)	1	2 D+/D-	
Napětí	+/- 5-15V	+/-2V diff	2.5V diff	*L, OC (1.8-5V)	*L (1.8-5V)	*L, OC (3-5V)	3.3V; 400mV diff	
Max.zařízení	2	32 (256)	*C	*C	*C	*C	127	
Kom.rychlost vzdálenost	kb-100kb m-10m	Kb-Mb m-1000m	Kb-Mb m-1000m	100kb(std)-3.4Mb(hs) [<m]	10Mb [<m]	16k (std) (160k od) m-100m	1.5,12; (LS/FS) 480Mb (HS) 5m	
Přenos	async	async	async	sync	sync	async	async	
Jednotka přenosu	Byte	byte	Paket: adresa a 0-8B	byte	Byte/word	byte	Paket 8-256B frame 1ms/125us	
adresace	-	*P	11b,29b	7,10bitů	Vodičem CE	64bitů	-	
zabezpečení	(parita)	(parita)	CRC16	-	-	CRC8	CRC16	
Fyz.odolnost	Zkrat, (napětí) rušení	Zkrat, napětí rušení	Zkrat, napětí rušení	Zkrat	není	zkrat	Není rušení	

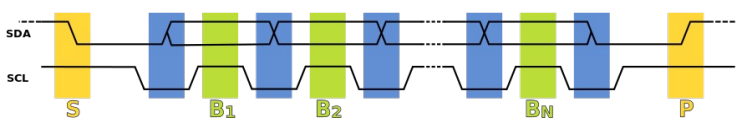
*C – limitováno kapacitou sběrnice a bit.rychlostí

*L – Úroveň dána připojenými log.obody, typ. 1.8,2.5,3.3,5V. OC=otevřený kolektor

*P – závislé na protokolu vyšší vrstvy

I²C (IIC)

I2C (Inter-Integrated Circuit) je synchronní sběrnice typu multimaster, původně vyvinutá firmou philips pro připojení nízkorychlostních periférií na desce. (z licenčních důvodů ji někteří výrobci označují i jinak, např. TWI)



Komunikace probíhá pomocí dvou vodičů - hodinový signál SCL (Synchronous Clock) a datový vodič SDA (Synchronous Data). Sběnije je typu OC (Open Colector), tj. Log.1 zajišťuje pull-up rezistor, log.0 je pak řízena aktivně obvody.

Každá stanice připojená na I^2C má přidělenou 7 bitovou adresu. Po zachycení podmínky START porovnávají všechny obvody svou adresu s adresou, která je vysílána na sběrnici. Zjistí-li některý z obvodů shodu, je vysílání určeno právě jemu a musí přijetí adresy potvrdit bitem ACK. Potom přijímá nebo vysílá další data. Několik adres je na I^2C vyhrazeno pro speciální účely. Například adresa 0000000 je určena pro vysílání broadcast, adresa 11110XX pro 10 bitovou adresaci.

Standardní rychlost je 100kbps, dalšími podporovanými je 400kb,1Mb a 3.4Mbps.

Reálné použití:
- pozor na počet zařízení, délku sběrnice/rychlost, pull-up odpory
- více stejných sensorů –> konflikt adres
- existují i2c oddělovače

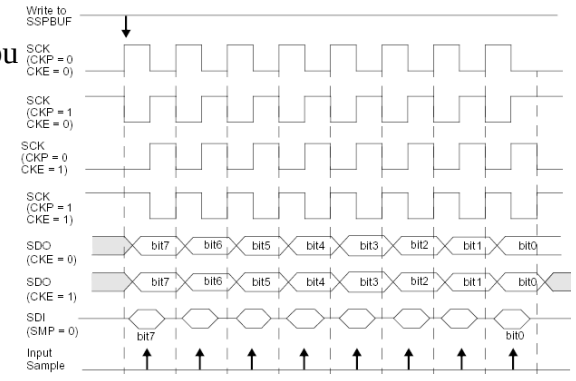
Příkladem I2C zařízeních jsou nejrůznější sensory – teplot, tlak, vlhkost, vzdálenost, osvětlení, zrychlení, magnetické pole, ...

Použití I2C pro M5core (M5core je master):

```
from machine import I2C, Pin

#definice/inicializace I2C – komunikacni frekv. 400kbps
i2c = I2C(1, scl=Pin(33), sda=Pin(32), freq=400000)

#adresa se kterou chceme komunikovat (slave address)
address = 0x44

i2c.writeto(address, b'\x2C\x06')       #zapis po i2c 0x2c,0x06
i2c.writeto(address, bytes([0x2c,0x06]) #zapis po i2c 0x2c,0x06
#pozn. komunikacni protokol je často ve tvaru: č.registru+data
#zde tedy do reg.0x2c zapisuji hodnotu 0x06

data=i2c.readfrom(address,6)   #cteni 6 bytes po i2c
cislo1B = data[0]              #zpracovani/prevod nactenich dat (priklad)
cislo2B = data[0]*256 + data[1]#zpracovani/prevod nactenich dat (priklad)

#proscanuj která zařízení jsou na sběrnici, vrať/vytiskni seznam adres
print(i2c.scan())   #scan vraci list/seznam i2c address
```

SPI

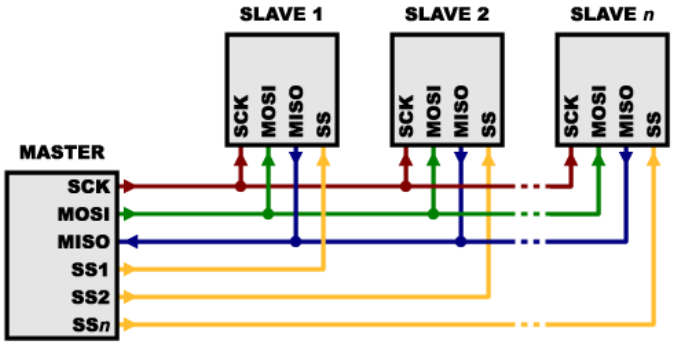
SPI (*Serial Peripheral Interface*) je sériové periferní rozhraní typu master-slave. Pro komunikaci slouží tři vodiče SCK (hodinový signál), SDO (data output, někdy též MOSI – master out, slave in), SDI (data input, též MISO – master in, slave out). K „adresaci“/výběru zařízení se kterým bude master komunikovat slouží dedikovaný signál SCE (pro každé zažízení – tímto signálem se také defnuje počátek a konec komunikace.

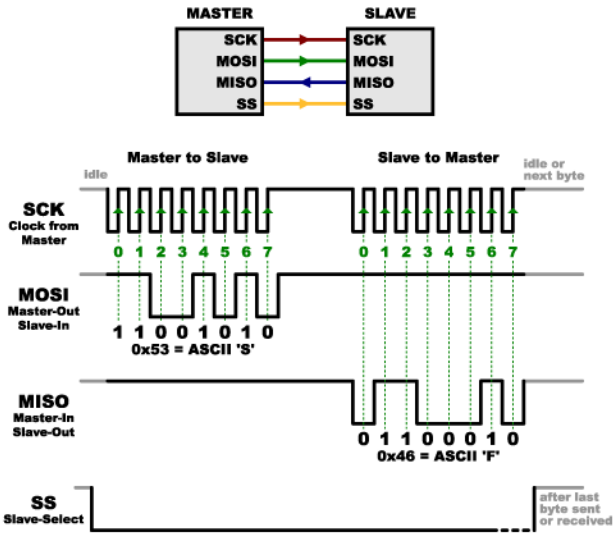


Master a slave musí mít stejnou klidovou uroven signalu SCK a reakci na kterou hranu hodin jsou data validní – nazývá se to obvykle polarita a fáze SPI signálu (4 kombinace)

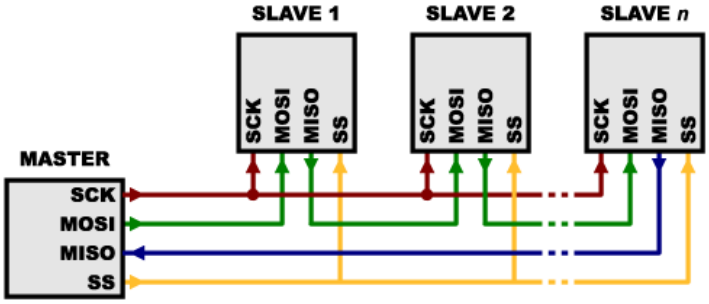
Implementačne je sběrnice SPI velmi jednoduchá.

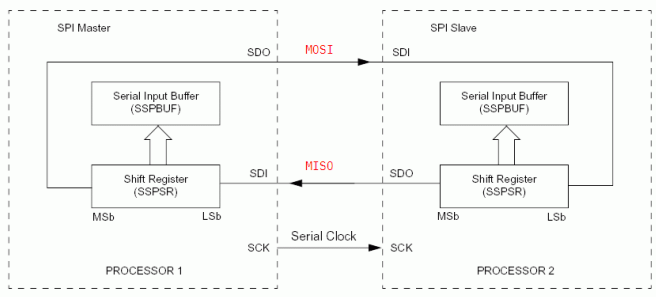
Pro více slave jednotek se použžívá typicky toto zapojení:





Ale ve spec. případech (nutná i podpora na straně slavů) lze použít zřetězení



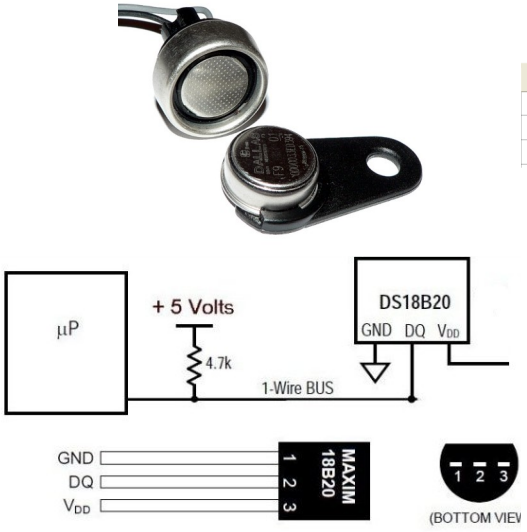


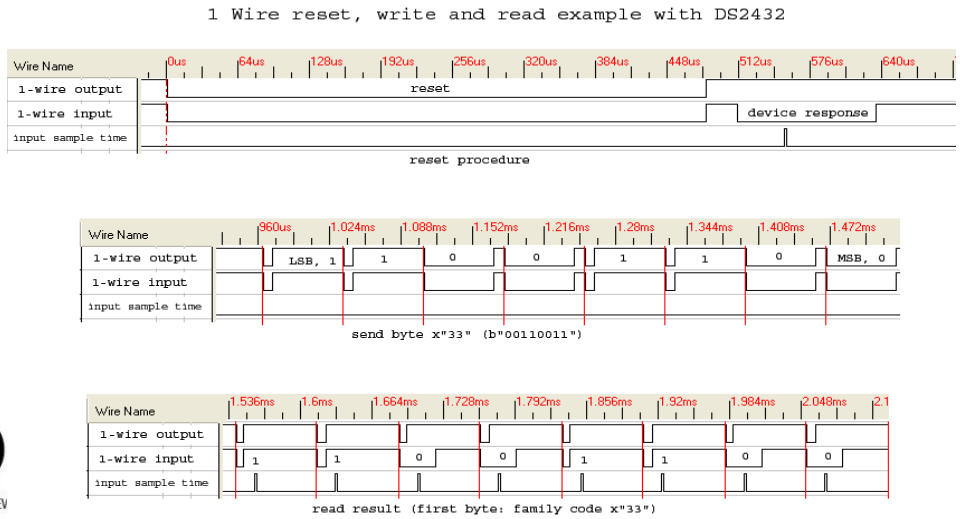
Použití SPI pro M5core (M5core je master):

```
#definice/inicializace SPI – komunikacni frekv. 500kbps
spi2 = machine.SPI(1, 500000, sck=machine.Pin(18), miso=machine.Pin(38),
mosi=machine.Pin(23), firstbit=machine.SPI.MSB, polarity=0, phase=0)

spi2.deinit()   #deinicialiace/vypnuti SPI

cs = Pin(4, mode=Pin.OUT, value=1)      #Create chip-select on pin 3

txdata = b"12345678"            #bytearray data – z ascii znaků
txdata = b"\x12\x34\x56\x78"    #bytearray data – z hex cisel 0x12,..
txd=[1,2,3,4,5,6,7,8]           #list/seznam ciselnych dat
txdata = bytearray(txd)         #bytearray z dat z listu/seznamu
rxdata = bytearray(len(txdata)) #misto pro prijmuta data

spi2.write_readinto(txdata,rxdata)  #write/read SPI

rdx=list(rxdata)                #list z bytearray
```

1wire

1-Wire je sběrnice navržená firmou Dallas pro komunikaci zařízení nízkou datovou rychlostí po jednom vodiči (+zem) (někdy lze i +vcc). Komunikace je master-slave, sběrnice je typu otevřený kolektor. Každé vyrobené 1-Wire zařízení má unikátní 48bitový ID kód (+8b typ zařízení +8b CRC = 64b) úpomocí kterého jej lze na sběrnici adresovat





Základní sekvence je reset puls následovaný 8 bity příkazu a potom odesílaná, nebo přijímaná data ve skupinách 8 bitů. Log.0/1 je k=odov8na délkou pulsu do země.

Je li na sběrnici více zařízení je možno zjistit jejich adresy (prohledávání pomocí píkazy search – a detekcí kolizí)

Reálné použití

- obvykle není v procesoru specializované periférie. Pro komunikaci lze vhodně „zneužít“ uart i spi.

RS232 (uart)

RS232 je rozhraní pro přenos informací - přenos probíhá asynchronně, duplexně, pomocí pevně nastavené přenosové rychlosti a synchronizace sestupnou hranou startovacího bitu, na napěťové úrovních+/- 5-15V

RS 232 používá dvě napěťové úrovně. Log. 1 je indikována zápornou úrovní, zatímco logická 0 je přenášena kladnou úrovní výstupních vodičů. Signály jsou odolné proti zkratu.

Kromě dat. Vodič RX/TX norma zahrnuje I “handshakové“ signály k řízení přenosu (RTS/CTS, ...)

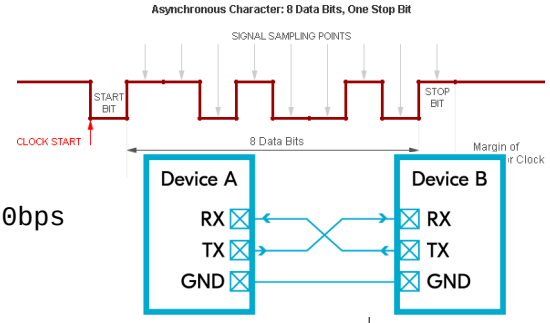
Úroveň	Vysílač	Přijímač
Log. L	+5 V to +15 V	+3 V to +25 V
Log. H	-5 V to -15 V	-3 V to -25 V
Nedefinovaný	-3 V to +3 V	

RS232

Pin 1	DCD
Pin 2	RXD
Pin 3	TXD
Pin 4	DTR
Pin 5	GND
Pin 6	DSR
Pin 7	RTS
Pin 8	CTS
Pin 9	RI

Přenos probíhá asynchronně: 1 start bit, N-dat.bitů (typicky 8), volitelně parita, a 1-2 stop bity.

Periférie v procesorech implementují tento standard do univerzální perifŕie uart (universal asynchronous seriál), případně usart (přidává synchronní variantu) – s bežnými napěťovými úrovni na výstupu – kde pro připojení ke sběrnici dle standardu RS232 (či RS422/RS485 – viz dále) je třeba ještě připojit driver (integrovaný obvod který mimo jiné převádí napěťové úrovně signálu)

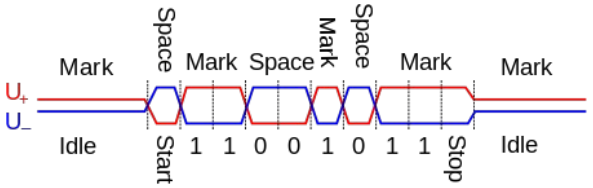


Použití UART pro M5core:

```
#definice/inicializace UART – komunikacni frekv. 115200bps
#8bitu data, bez parity, 1 stop bit. Piny 1 a 3.
uart0 = machine.UART(1, tx=1, rx=3)
uart0.init(115200, bits=8, parity=None, stop=1)

uart0.write('text to transmit\r')  #text posilany uartem

while uart0.any():                 #vyprazdneni vstupni fronty
  if uart0.read(1): pass           #ctene znaky jsou zahazovany/nepouzity

if uart0.any():                    #je neco ve vstupni fronte
  buf = str((uart1.readline()))    #nacti radku textu
  if buf.count('XXX') == 1: #...   #jestli radka obsahuje XXX pak ...
```

RS485 (RS422, RS232)

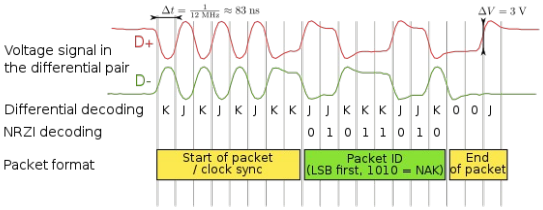
RS485 je standard sériové komunikace používajíc se především v průmyslovém prostředí – jedná se o dvouvodičový poloduplexního vícebodový sériový spoj. Norma definuje až 32zařízení (při použití spec.budících obvodů lze více) a max.vzdálenost 1200m. Rychlost až 10Mbs na krátké vzdálenosti (do 10m). Vedené má definované zakončovací odpory 110R (zabránění odrazům).
Logické úrovně (nebo stavy) jsou reprezentovány rozdílovým napětím mezi oběma vodiči (differenciální signál) . Vodiče se označují A/B (někdy -/+) - v klid.stavu by na A mělo být menší napětí nez na B. Vysílač by měl na výstupu při logické 1 (klidový stav linky) generovat na vodiči A napětí −2 V, na vodiči B +2 V, při logické 0 by měl na vodiči A generovat +2 V, na vodiči B −2 V. Přijímač vyhodnocuje rozdíl A-B od 200mV.



Obdobně jako u RS232 přenos probíhá asynchronně: 1 start bit, Ndat.bitů (typ.8), volitelně parita, 1-2 stop bity.
RS485 – poloduplexní přenos
RS422 – plně-duplexní přenost
Nejrozšířenejší protokol postavený nad RS485 je protokol **ModBus**.

USB

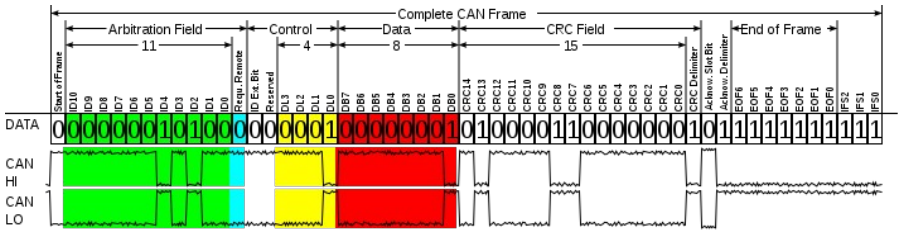
Universal Serial Bus (USB) je univerzální sériová sběrnice snačící se jednotně nahradit různé dříve existujíc standardy přípojení (RS232,par.port,PS/2, Gameport) pro bežné druhy periférií (tiskárny, klávesnice, myši, modemy) ale i připojit dalsí periférie (kamety, paměti flash, HD, optické mechaniky



atd). USB je point-to-point spojení, master-slave (host-device). Více zařízení je možno připojit přes HUB (rozbočovač) – stomové větvení (až 5 úrovní). USB je značně komplexní protokol.
Data jsou přenášena polodulexně, differenciálním párem D+/D- s definovanou impedancí bez zakončení. pro LS/FS (low/full speed – 1.5,12Mb) je úroveň signállu cca 3.3V, pro HS (hi speed – 480Mb) cca 400mV
Komunikační protokol pracuje na principu paketového přenosu – kdy s pomocí malých bloků údajů je schopen současně řešit více požadavků a přenosů.

CAN bus

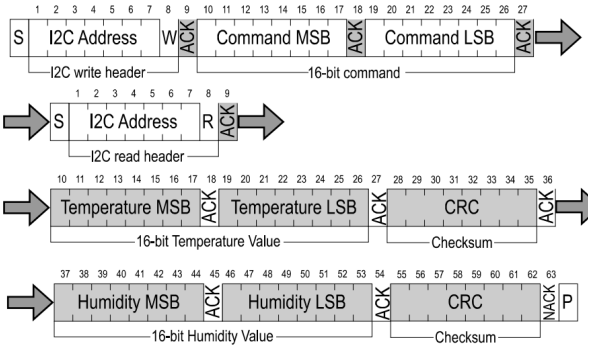
CAN (Controller Area Network) je diferenciální, multimaster sběrnice vyvinuté pro použití v automotive průmyslu. Data se odesílají v rámcích, každý rámec může obsahovat 0-8 datových byte, adresu/identifikátor který definuje obsah přenášené zprávy a zároveň i prioritu zprávy. zpráva je doplněna zabezpečením CRC. Maximální teoretická rychlost přenosu na sběrnici je 1 Mbs/40m.



Příklad I2C komunikace – program

Program který bude číst hodnot z čidla SHT30 – teplotu a vlhkost (Dokumentace k čidlu viz Datasheetx k SHT30).

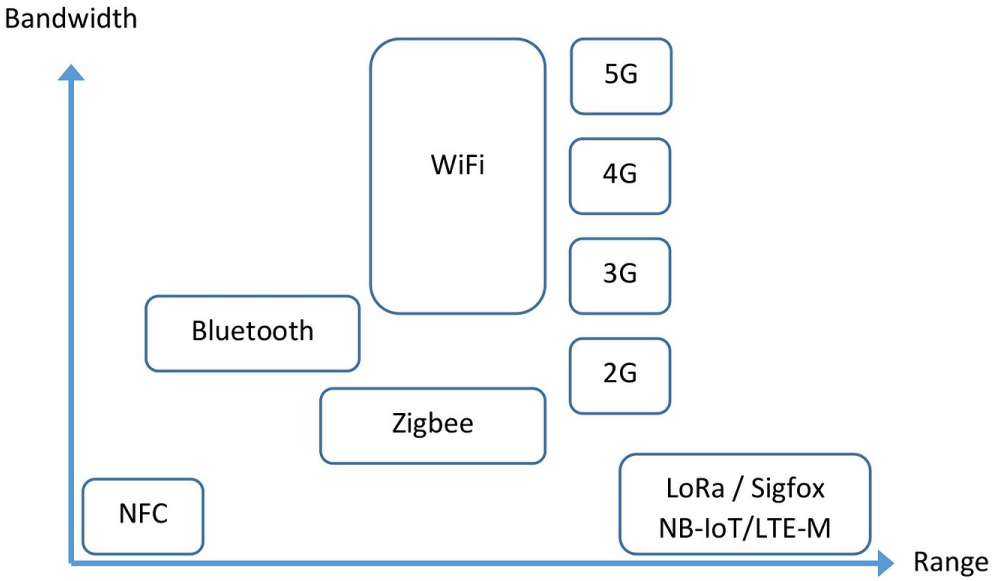
(pozn. SHT30 je součástí čidla ENV.II SENSOR)

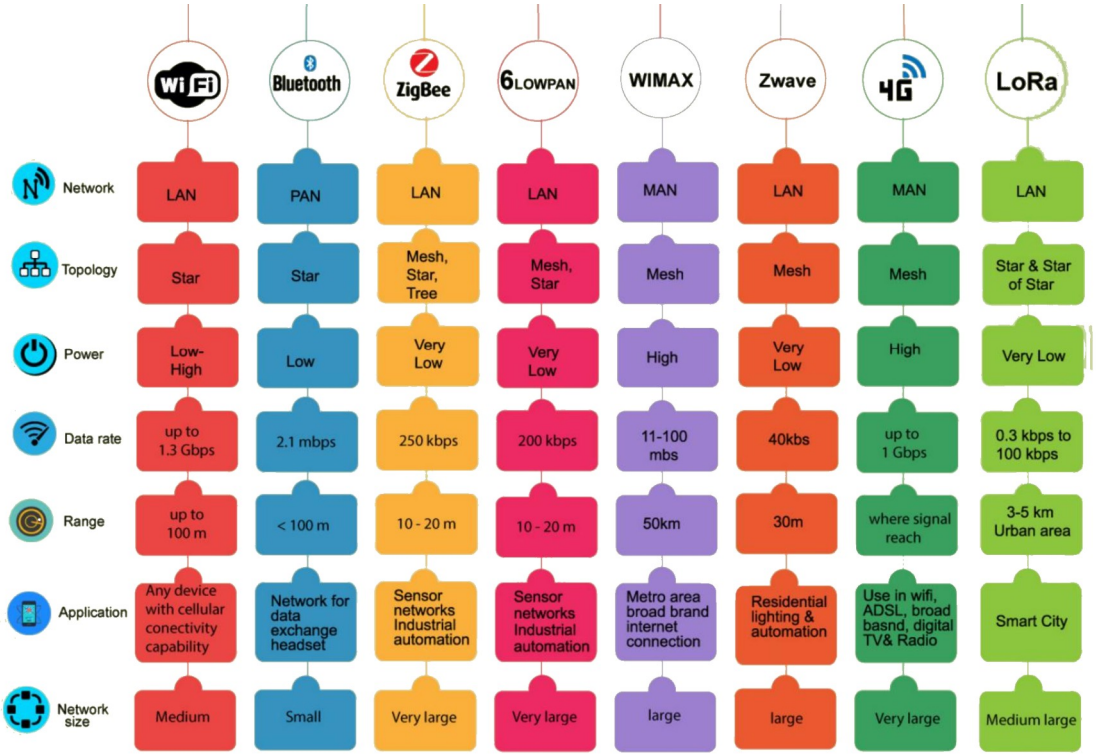


```
#I2C testovaci program - ocekava cidlo SHT30 na I2c (piny 32,33)
#cte hodnoty z cidla (teplotu, vlhkost) a vypisuje na obrazovku
from m5stack import *
from m5stack_ui import *
from uiflow import *
import machine
from machine import I2C, Pin

screen = M5Screen()
screen.clean_screen()
screen.set_screen_bg_color(0xFFFFFF)

BMP280_ADDRESS = 0x76   #i2c addresa 0x76=118
SHT30_ADDRESS = 0x44    #i2c addresa 0x44=68

#Fix Texts on screen
labeltd = M5Label('Devs:',    x=10, y=10, color=0x000000, font=FONT_MONT_30)
labelth = M5Label('Hum[%]:',   x=10, y=70, color=0x000000, font=FONT_MONT_30)
labeltt = M5Label('Tmp[°C]:',  x=10, y=100, color=0x000000, font=FONT_MONT_30)
#variable (non-fix) text on screen, measured values
labeld = M5Label('---', x=160, y=10, color=0x000000, font=FONT_MONT_30)
labelh = M5Label('---', x=160, y=70, color=0x000000, font=FONT_MONT_30)
labelt = M5Label('---', x=160, y=100, color=0x000000, font=FONT_MONT_30)

#variable initialisation
hum=0;     #humidity
temp=0;    #temperature1


#definice/inicializace I2C – komunikacni frekv. 400kbps
i2c = I2C(1, scl=Pin(33), sda=Pin(32), freq=400000)

#proscanuj která zařízení jsou na sběrnici, vrať seznam adres (ukaz na obrazovce)
labeld.set_text(str(i2c.scan()))


#main loop
while(True):
  # Odeslání příkazu pro měření
  i2c.writeto(SHT30_ADDRESS, bytes([0x2c,0x06]))

  wait_ms(500)   #wait for data measure
  #read 6 bytes - format: Temper.MSB,Temper.LSB,Temper.CRC,Hum.MSB,Hum.LSB,Hum.CRC
  data=i2c.readfrom(SHT30_ADDRESS, 6)
  # Zpracování teploty
  temp_raw = data[0] << 8 | data[1]
  temp = -45 + (175 * temp_raw / 65535.0)
  # Zpracování vlhkosti
  hum_raw = data[3] << 8 | data[4]
  hum = 100 * hum_raw / 65535.0
  #measured values to screen
  labelh.set_text("%0.2f"%hum) #zmena textu
  labelt.set_text("%0.2f"%temp) #zmena textu
```

Bezdrátová Komunikace

Srovnání/použitelnost technologií





pozn. PAN,LAN,MAN = persponal,local,metropolite area network

Bluetooth

Bluetooth (BT) je bezdrátová technologie, která umožňuje propojení různých elektronických zařízení na krátkou vzdálenost. Je standardizováno pod IEEE 802.15.1 a spadá do kategorie osobních počítačových sítí (PAN).

Bluetooth Low Energy (BLE) je specifická verze Bluetooth technologie, která je optimalizována pro nízkou spotřebu energie a je ideální pro aplikace, kde je důležitá dlouhá životnost baterie

Využívá rádiové vlny v pásmu 2,4 GHz, které je rozděleno na 79 kanálů, každý o šířce 1 MHz. Používá různé modulační techniky, jako je GFSK (Gaussian Frequency Shift Keying) pro základní rychlosti a DQPSK pro vyšší rychlosti

Typický dosah Bluetooth je od 1 do 100 metrů, v závislosti na třídě zařízení. Bluetooth je navrženo tak, aby mělo nízkou spotřebu energie, což je ideální pro mobilní zařízení. Rychlost přenosu dat se liší podle verze Bluetooth. Například verze 5.0 umožňuje rychlost až 2 Mbps.

Bluetooth je zpětně kompatibilní, což znamená, že novější verze mohou komunikovat se staršími verzemi (nekompatibilita BT – BLE, zařízení nemusí podporovat obě).

BLE podporuje několik topologií, které umožňují flexibilní a efektivní komunikaci mezi zařízeními:
-Point-to-Point: Nejčastější topologie, kde jedno centrální zařízení (central / **master**) komunikuje s jedním nebo více periferními zařízeními (periphery / **slave**), počet slave bývá omezen.
-Broadcasting: Zařízení vysílá data, která mohou být přijímána více zařízeními, aniž by bylo navázáno spojení.
-Mesh Networking (nové, od V5, BLE): Umožňuje vytvoření sítě, kde každé zařízení může komunikovat s více dalšími zařízeními, což je ideální pro aplikace jako chytré domácnosti a průmyslové automatizace.

Zigbee

Zigbee je ideální pro aplikace jako chytré domácnosti, průmyslové automatizace a zdravotnické zařízení, kde je klíčová nízká spotřeba energie a spolehlivá komunikace na kratší vzdálenosti.

Zigbee pracuje v nelicencovaném pásmu 2,4 GHz, ale také v pásmech 868 MHz (Evropa) a 915 MHz (Severní Amerika). Používá modulační techniky jako je DSSS (Direct Sequence Spread Spectrum) pro zajištění spolehlivého přenosu dat. Zigbee je založeno na standardu IEEE 802.15.4, který definuje fyzickou vrstvu a vrstvu řízení přístupu k médiu (MAC) pro nízkorychlostní bezdrátové osobní sítě (WPAN).
Zigbee zahrnuje bezpečnostní mechanismy jako je 128bitové šifrování AES pro zajištění bezpečnosti přenášených dat.

Zigbee je navrženo pro aplikace s nízkou spotřebou energie, což umožňuje dlouhou životnost baterie. Typický dosah Zigbee je od 10 do 100 metrů, v závislosti na prostředí a výkonu zařízení. Zigbee podporuje rychlosti přenosu dat až 250 kbps.
Zigbee podporuje mesh topologii, což znamená, že zařízení mohou přenášet data přes další zařízení, čímž se zvyšuje dosah a spolehlivost sít

Ve srovnání s BLE - Obě technologie jsou navrženy pro nízkou spotřebu energie, ale Zigbee je často preferováno pro aplikace s velmi dlouhou životností baterie. Podpora mesh topologii je zde od počátku (k BLE přidáváno), což obecně umožňuje větší pokryutí a spolehlivost sítě ve srovnání s BLE.

Typy zařízení u Zigbee:
- Koordinátor (**Coordinator**) - centrální prvek Zigbee sítě. Je zodpovědný za inicializaci a správu celé sítě. Koordinátor vytváří síť, přiděluje adresy novým zařízením a zajišťuje bezpečnostní klíče pro šifrování komunikace. Koordinátor může komunikovat se všemi zařízeními v síti a často funguje jako brána mezi Zigbee sítí a dalšími sítěmi, například internetem.
- Koncové zařízení (**End Device**) - je obvykle jednoduché zařízení s nízkou spotřebou energie, které komunikuje pouze s jedním routerem nebo koordinátorem. Koncová zařízení mohou být např. senzory, spínače nebo jiné jednoduché zařízení, které sbírají data nebo vykonávají jednoduché úkoly. Koncová zařízení neprovádějí směrování dat pro jiná zařízení, což znamená, že nemohou přenášet data mezi jinými zařízeními v síti.
- **Router** - pomáhají rozšiřovat dosah sítě a směrovat data mezi zařízeními. (pozn. jedno zařízení může mít více funkcí, tedy např. end device+router, router+koordintátor.

Wifi

Wi-Fi je bezdrátová síťová technologie, která umožňuje zařízenímkomunikovat přes bezdrátový signál – wifi (dle verze) pracuje v několika frekvenčních pásmech, nejčastěji 2,4 GHz a 5 GHz (Wifi 5), nebo 6 GHz (Wi-Fi 6E). Wi-Fi je založeno na standardech IEEE 802.11, které definují různé verze, jako jsou 802.11a/b/g/n/ac/ax3. Rychlost přenosu dat se liší podle verze standardu, například Wi-Fi 6 (802.11ax) může dosáhnout rychlosti až 9,6 Gbps. Dosah Wi-Fi signálu závisí na prostředí a výkonu zařízení, obvykle se pohybuje od 20 do 100 metrů. Wi-Fi používá různé modulační techniky, jako je OFDM (Orthogonal Frequency-Division Multiplexing) a DSSS (Direct Sequence Spread). Frekvenční pásma jsou rozdělena na kanály, například pásmo 2,4 GHz má 14 kanálů, každý o šířce 22 MHz.

Wi-Fi zahrnuje různé bezpečnostní protokoly, jako je WPA2 a WPA3, které zajišťují šifrování a ochranu dat před neoprávněným přístupem.

Wi-Fi sítě mohou být konfigurovány v různých topologiích, jako je infrastruktura (přístupový bod (**access point**) a klienti (**client**) – nečastější řešení) nebo ad-hoc (přímá komunikace mezi zařízeními).

Wi-Fi je široce používáno pro domácí i komerční účely, umožňuje snadné připojení k internetu a sdílení dat mezi zařízeními bez potřeby kabelů.

LoRa

LoRa (Long Range) je bezdrátová komunikační technologie navržená pro dlouhý dosah a nízkou spotřebu energie. LoRa umožňuje komunikaci na vzdálenosti až několik kilometrů, v závislosti na prostředí a podmínkách. Technologie je optimalizována pro nízkou spotřebu energie, což umožňuje dlouhou životnost baterií v zařízeních. LoRa využívá modulaci s rozprostřeným spektrem (Chirp Spread Spectrum, CSS), která je velmi odolná proti rušení a umožňuje spolehlivý přenos dat i v

rušeném prostředí. LoRa používá CSS, kde se frekvence signálu lineárně mění v čase (chirp-up a chirp-down). Tato modulace je odolná vůči Dopplerovu jevu a umožňuje dekódování signálu i při nízkém poměru signálu k šumu (SNR). LoRa pracuje v nelicencovaných pásmech, jako jsou 433 MHz, 868 MHz (Evropa) a 915 MHz (Severní Amerika). Typické šířky pásma jsou 125 kHz, 250 kHz a 500 kHz, což umožňuje flexibilní nastavení podle požadavků aplikace. LoRa umožňuje nastavení různých hodnot SF (spread factor), které ovlivňují dosah a rychlost přenosu dat. Vyšší SF zvyšuje dosah, ale snižuje rychlost přenosu. LoRa používá dopřednou korekci chyb (Forward Error Correction, FEC), která zvyšuje spolehlivost přenosu dat. Rychlosti přenosu jsou (dle SF a šířky pásma) v rozsahu 0.3 až 50 kbps (kbit per sec – kilobitů za sekundu) – tedy relativně pomalé.

LoRa technologie má stanovená omezení na poměr vysílání a ticha, což je důležité pro dodržování regulačních požadavků a zajištění efektivního využití spektra. Duty Cycle – je procento času, během kterého zařízení může vysílat v daném časovém intervalu. V Evropě, kde LoRa pracuje v pásmu 868 MHz, je duty cycle omezen na 1 % nebo 0,1 % v závislosti na konkrétním kanálu. Toto omezení znamená, že zařízení může vysílat pouze po omezenou dobu a musí zůstat tiché po zbytek času. Například při 1% duty cycle může zařízení vysílat maximálně 36 sekund za hodinu.
Listen Before Talk (LBT) - Některé regiony vyžadují, aby zařízení před vysíláním zkontrolovalo, zda je kanál volný. Pokud je kanál obsazen, zařízení musí počkat, než začne vysílat. Tento mechanismus pomáhá minimalizovat rušení mezi zařízeními a zajišťuje spravedlivé využití spektra.

LoRa/LoRaWAN jsou ideální pro aplikace, kde je potřeba dlouhý dosah, nízká spotřeba energie a spolehlivý přenos dat, jako jsou chytré města, průmyslové monitorování a zemědělství.

Typickými příklady použití jsou senzory, které odesílají data pouze občas (např. jednou za hodinu), mohou snadno dodržovat omezení duty cycle, nebo zařízení pro monitorování prostředí nebo infrastruktury, která potřebují pravidelně odesílat malé množství dat.

Sigfox

Sigfox (podobně jako LoRa) spadá do kategorie nízkoenergetických širokopásmových sítí (LPWAN) a jsou navrženy pro podobné aplikace v oblasti IoT.

Sigfox: Používá ultra-úzkopásmovou (UNB) modulaci, která umožňuje velmi úzké kanály (200 Hz), což zvyšuje odolnost proti rušení a umožňuje dlouhý dosah. Pracuje v nelicencovaných pásmech, jako jsou 868 MHz (Evropa) a 902 MHz (Severní Amerika). Podporuje rychlosti přenosu dat až 100 bps (tedy velmi pomalé). Používá hvězdicovou topologii, kde zařízení komunikují přímo s centrálními základnovými stanicemi. Omezuje počet zpráv na 140 denně, každá zpráva může mít až 12 bajtů

GSM

GSM (Global System for Mobile communications) je standard pro mobilní komunikaci, který byl původně vyvinut pro druhou generaci (2G) mobilních sítí. GSM používá digitální přenos hlasu a dat, což zvyšuje kvalitu a bezpečnost komunikace.

GSM pracuje v různých frekvenčních pásmech (v různých sátech se mírně liší) – např. 900/1800MHz v Evropě a 850/1900 MHz v Severní Americe. GSM používá kombinaci frekvenčního dělení (FDMA) a časového dělení (TDMA) pro efektivní využití spektra.

Původní GSM podporuje rychlosti přenosu dat až 9.6 kbps pomocí CSD (circuits switched data). CSD je vhodné pro aplikace, které vyžadují stabilní a nepřerušovaný přenos dat, jako je fax nebo dial-up internet.
High-Speed Circuit-Switched Data (HSCSD) - zvyšuje rychlost přenosu dat až na 57.6 kbps kombinací více časových slotů
GPRS (General Packet Radio Service) – pak umožňuje rychlosti přenosu dat od 56 kbps do 114 kbps. Používá paketový přenos dat, což umožňuje efektivnější využití spektra. GPRS je někdy označováno jako 2.5G technologie, protože poskytuje vyšší rychlosti než původní GSM.

EDGE - EDGE zvyšuje rychlosti přenosu dat až na 384 kbps2. EDGE je vylepšení GPRS, které používá sofistikovanější metody modulace pro zvýšení rychlosti přenosu dat. EDGE je považováno za 2.75G technologii a poskytuje téměř 3G rychlosti pro aplikace, které vyžadují vyšší přenosové rychlosti.

3G (Třetí generace) - přinesla významné zlepšení oproti 2G a 2.5G technologiím, jako jsou GPRS a EDGE. 3G sítě podporují rychlosti přenosu dat od 144 kbps až po několik Mbps, v závislosti na konkrétní technologii a podmínkách. 3G zahrnuje různé technologie, jako jsou UMTS (Universal Mobile Telecommunications System) a CDMA2000. UMTS používá W-CDMA (Wideband Code Division Multiple Access) pro přenos dat.

4G (Čtvrtá generace) - 4G sítě mohou dosahovat rychlostí až 100 Mbps pro mobilní uživatele a až 1 Gbps pro stacionární uživatele. 4G zahrnuje technologie jako LTE (Long Term Evolution) a WiMAX. LTE je nejrozšířenější a používá OFDM (Orthogonal Frequency Division Multiplexing) pro efektivní přenos dat. 4G umožňuje vysokorychlostní internet, HD videohovory, online hraní a další náročné aplikace.

5G (Pátá generace) - přináší další zlepšení v rychlosti, latenci a kapacitě. 5G sítě mohou dosahovat teoretických rychlostí až 20 Gbps, což je výrazně více než u 4G. 5G využívá nové technologie, jako je mmWave (millimeter wave) pro vyšší frekvence, MIMO (Multiple Input Multiple Output) pro zvýšení kapacity a beamforming pro lepší pokrytí.

Architektura GSM sítě:
- Mobilní stanice (MS) - Mobilní telefony a další zařízení, která komunikují s GSM sítí.
- Bázové stanice (BTS) - Zařízení, která zajišťují bezdrátovou komunikaci mezi mobilními stanicemi a sítí1
- Řídicí stanice bázových stanic (BSC) - Spravuje více BTS a zajišťuje přepínání a řízení zdrojů.
- Mobilní přepínací centrum (MSC) - Hlavní prvek sítě, který zajišťuje přepínání hovorů a správu připojení.

Uživatelská datová komunikace v GSM sítích je založena na IP protokolu. K připojení jsou používány GSM modemy – které se (často) ovládají AT příkazy.

Komunikační protokoly

S výjimkou ESPnow je možné níže uvedené protokoly realizovat jak bezdrátově (typicky WiFi) tak s pevným připojením (typicky Ethernet).

UDP/IP, TCP/IP

Základní komunikační protokoly dnešního internetu. Nad nimy jsou postavenu ostatní služby (např. dále zmínené HTTP, MQTT).

IP (Internet Protocol) – je na síťové vrstvě (Network Layer) – je zodpovědný za adresování a směrování paketů mezi zařízeními v síti. Používá IP adresy k identifikaci zařízení a zajišťuje, že data mohou být směrována a doručena na správné místo. IP paket může mít maximální velikost 65,535 bajtů, včetně hlavičky. Dvě verze protokolu: IPv4 kde adresy jsou 32bitové a jsou obvykle zapisovány ve formátu čtyř desítkových čísel oddělených tečkami (např. 192.168.1.1) a IPv6 kde adresy jsou 128bitové a jsou zapisovány ve formátu osmi skupin čtyř hexadecimálních číslic oddělených dvojtečkami (např. 2001:0db8:85a3:0000:0000:8a2e:0370:7334).

UDP (User Datagram Protocol) – postaven na IP protokolu na transportní vrstvě (Transport Layer) – je to jednoduchý, bezspojový (bezestavový) protokol, který umožňuje rychlý přenos dat bez zajištění doručení nebo pořadí paketů. Je vhodný pro aplikace, kde je rychlost a jednoduchost důležitější než spolehlivost, jako jsou streamování videa nebo online hry nebo jednoduchá komunikace.

TCP (Transmission Control Protocol) - postaven na IP protokolu na transportní vrstvě (Transport Layer) – je to spojovaný (stavový) protokol (je vytvářeno point-to-point spojení), který zajišťuje spolehlivý přenos dat mezi zařízeními. Poskytuje kontrolu toku, detekci chyb a zajišťuje, že data dorazí ve správném pořadí a bez ztráty (tj. detekce chyb, timeoutů, pořadí dat, zajištění případného opakování přenosu atd.)

Protokoly UDP a TCP používají porty k identifikaci specifických aplikací nebo služeb na zařízení. Porty jsou 16bitové čísla, což znamená, že mohou mít hodnoty od 0 do 65535. Některé porty jsou rezervovány pro specifické služby (např. port 80 pro HTTP, port 443 pro HTTPS, 1883/8883 pro MQTT atp).

ESPnow

ESP-NOW (https://www.espressif.com/en/solutions/low-power-solutions/esp-now) je bezdrátový komunikační protokol vyvinutý společností Espressif, který umožňuje rychlou a nízkoenergetickou komunikaci mezi zařízeními bez potřeby routeru (centrálního prvku).
ESP-NOW je kompatibilní s čipy ESP8266 a ESP32 (ESP32, ESP32-S, ESP32-C)

Hlavní vlastnosti ESP-NOW

ESP-NOW pracuje ve stejném frekvenčním pásmu jako Wi-Fi, tedy 2.4 GHz, ale nepoužívá wifi infrastrukuru (access point) - využívá technologii podobnou bezdrátovým klávesnicím a myším, což umožňuje rychlý přenos dat. Jedná se o protokol na linkové vrstvě OSI modelu (a tedy není zde spojitost s IP protokolem). Pro případné připojeneí k internetu je třeba brána.

- Rychlá odezva: Díky minimalizaci vrstev OSI modelu je komunikace velmi rychlá
- Nízká spotřeba energie: Protokol je navržen tak, aby spotřebovával co nejméně energie, což je ideální pro bateriově napájená zařízení
- Podpora více zařízení: ESP-NOW umožňuje komunikaci mezi více zařízeními, a to jak v režimu one-to-many, tak many-to-many a one-to-one (zde případně i potvrzování doručení zprávy)
- Bezpečnost: Podporuje i šifrovanou komunikaci pomocí algoritmů ECDH a AES1
- V jednom paketu je možné přenést až 250byte
- Pozor na souběh s WiFi – na ESP je možné používat zároveň s wifi ale kanál wifi a kanál pro ESPnow musí být totožný

```
#ESPnow broadcast example - send messages (to all, who listening), receive messages
#show: local MAC adress, last received data, last received MAC, last transmited data
#action: button A - send next counter, button B - send X
from m5stack import *
from m5stack_ui import *
from uiflow import *
from m5ui import *
import espnow, wifiCfg
import machine

setScreenColor(0x000000)

#Wifi and ESPnow init
wifiCfg.wlan_ap.active(True)
wifiCfg.wlan_sta.active(True)
espnow.init(0)                #ESPnow – activate on channel 0 (can be 0-13)
addr = espnow.get_mac_addr()  #get own/local MAC address

#espnow.add_peer('30aea4587df1', id=1) #set MAC address of peer 1

title = M5Title(title='MAC: '+addr, x=3, fgcolor=0xFFFFFF, bgcolor=0x0000FF)        #own MAC
labelr = M5TextBox(5, 50,  "received data", lcd.FONT_DejaVu40, 0xFFFFFF, rotate=0)  #receive data
labela = M5TextBox(5, 100, "received mac", lcd.FONT_DejaVu40, 0xFFFFFF, rotate=0)   #receive MAC
labelt = M5TextBox(5, 150, "last transmit", lcd.FONT_DejaVu40, 0xFFFFFF, rotate=0)  #transmit data
cnt=0

#event - for received ESPnow packet
def recv_cb(_):
  addr, _, data = espnow.recv_data(encoder='str')    #get received packet (address and data)
  labelr.setText(data)  #show received data string
  labela.setText(addr)  #show address (MAC) of device who send the message
  pass

espnow.recv_cb(recv_cb)

#event - buttonA pressed -> send counter <cnt>
def buttonA_wasPressed():
  global cnt
  message=str(cnt)
  #espnow.send(id=1, data=message)  #send to peer 1 only
  espnow.broadcast(data=message)
  labelt.setText('...'+message)
  cnt+=1;
  pass

#event - buttonB pressed -> send address
def buttonB_wasPressed():
  global addr
  labelt.setText('...X')
  espnow.broadcast(data='X')
  pass

btnA.wasPressed(buttonA_wasPressed)  #event function for button A
btnB.wasPressed(buttonB_wasPressed)  #event function for button B

espnow.broadcast(data="Hallo") #First mesage for all

while(True):
  addr, _, data = espnow.recv_data(encoder='str') #hack! for flushing receive buffer
  wait_ms(1000)
```

(pozn. pro hack – 1/ takto je možné číst zprávy nepoužíváte-li event. 2/ Při použití eventu je zřejmě možná desynchronizace a event pak nevrací právě došlou zprávu ale přechozí došlou. Přidáním čtení se tato z bufferu přečte.)

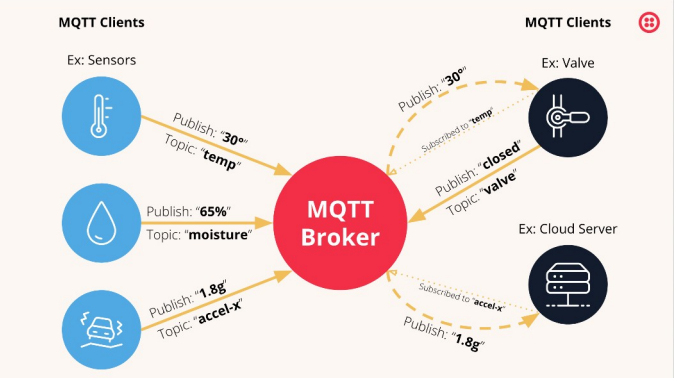
MQTT

MQTT (Message Queuing Telemetry Transport) je jednoduchý, publish-subscribe protokol určený pro komunikaci mezi zařízeními v rámci Internetu věcí (IoT). Byl navržen tak, aby byl efektivní z hlediska šířky pásma a energetické náročnosti, což je ideální pro zařízení s omezenými zdroji nebo připojeními s nízkou šířkou pásma.

Použití MQTT zahrnuje domácí automatizaci (Ovládání a monitorování chytrých zařízení v domácnosti), Průmyslovou automatizaci (Monitorování a řízení průmyslových procesů), Dopravu a logistiku (Sledování vozidel a zásilek v reálném čase) atp.

Hlavní vlastnosti MQTT:
Jednoduchost - MQTT má malé hlavičky zpráv, což minimalizuje nároky na šířku pásma.
Publish-Subscribe Architektura - Umožňuje zařízení publikovat zprávy na určité téma a jiná zařízení se mohou přihlásit k odběru těchto témat.
Spolehlivost - MQTT podporuje různé úrovně kvality služeb (QoS), což zajišťuje spolehlivé doručení zpráv.
Škálovatelnost - Může být použit pro jedno i tisíce zařízení.
Podpora pro nestabilní sítě - MQTT je navrženo tak, aby dobře fungovalo i na nestabilních sítích, jako jsou mobilní nebo bezdrátové sítě.

MQTT broker je klíčová součást architektury MQTT. Funguje jako centrální uzel v systému publish/subscribe, kde přijímá zprávy od vydavatelů (publishers) a předává je odběratelům (subscribers) na základě jejich předplatných témat.



Hlavní funkce MQTT brokera:
- Přijímání zpráv: Broker přijímá zprávy od různých zařízení nebo aplikací, které je publikují na specifická témata.
- Filtrování a distribuce: Broker filtruje zprávy podle témat a distribuuje je pouze těm klientům, kteří se přihlásili k odběru daného tématu.
- Správa připojení: Broker spravuje připojení klientů, včetně navazování a ukončování spojení, a zajišťuje, že zprávy jsou doručeny i při opětovném připojení klientů.
- Bezpečnost: MQTT broker může poskytovat bezpečnostní opatření, jako je autentizace a šifrování, aby zajistil bezpečný přenos dat mezi zařízeními a aplikacemi.

Existuje řada (i free) MQTT brokerů k instalaci na vlastní HW + řada veřejných (i free) i neveřejných serverů, např. https://mntolia.com/10-free-public-private-mqtt-brokers-for-testing-prototyping/

MQTT klient je jakékoli zařízení nebo aplikace, které používají knihovnu MQTT a komunikují s MQTT brokerem přes síť. Klient může fungovat (i zároveň) ve dvou rolích:

- Publisher (vydavatel): Klient, který odesílá zprávy na určitá témata.
- Subscriber (odběratel): Klient, který přijímá zprávy z témat, na která se přihlásil.

Hlavní funkce MQTT klienta:
- Připojení k brokeru: Klient se připojuje k MQTT brokeru pomocí TCP/IP nebo jiného transportního protokolu.
- Publikování zpráv: Klient může odesílat (public)ú zprávy na specifická témata, která jsou poté distribuována brokerem.
- Odběr zpráv: Klient se může přihlásit k odběru (substribe) zpráv na určitá témata a přijímat je, když jsou publikovány.
- Udržování spojení: Klient udržuje aktivní spojení s brokerem a může znovu navázat spojení v případě výpadku.
Pro testování existuje řada MQTT klientů, např. https://mntolia.com/5-best-mqtt-tools-for-desktop-mobile/ . Pro platofrmu PC dále třeba: MQTTX, MQTT Explorer nebo HiveMQ WebSocket Client který běží přímo v prohlížeči https://www.hivemq.com/demos/websocket-client/

MQTT podporuje tři úrovně kvality služeb (**QoS**) při přenosu zpráv.
- QoS 0 (At most once) - Na této úrovni publisher posílá brokeru zprávu jenom jednou a nečeká na žádnou odpověď, tj. nepotvrzováno, odešle a zapomene.
- QoS 1 (At least once) - Tato úroveň zajišťuje, že zpráva bude doručena brokeru, ale existuje možnost duplicity zpráv
- QoS 2 (Exactly once) - Na této úrovni je zaručeno doručení zpráv klientovi a je vyloučeno možné duplikování zpráv

Online MQTT brokers: https://github.com/emqx/blog/blob/main/en/202111/popular-online-public-mqtt-brokers.md , https://mqtt.org/software/

V MQTT se používají témata(**topic**) k identifikaci a směrování zpráv mezi publikátory a odběrateli. Struktura témat je typicky hierarchická a jednotlivé úrovně jsou odděleny lomítkem (/), podobně jako cesty URL. Například: (dle lokace)

domov/přízemí/obývací_pokoj/teplota

auto/1P23456/poloha/zemepisna_sirka

Alternativně (dle měřené veličiny):

teplota/domov/přízemí/obývací_pokoj

poloha/auto/1P23456/

Základní pravidla pro strukturu témat:

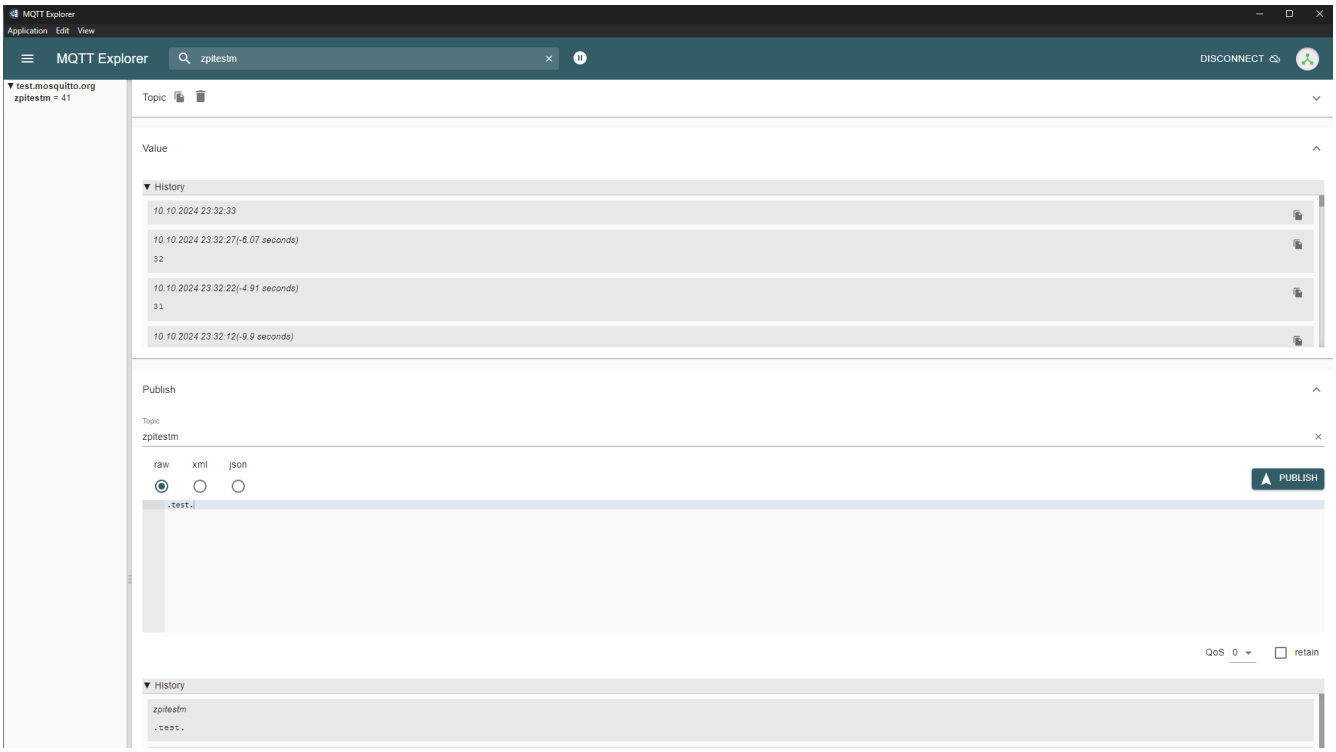
Hierarchická struktura: Témata jsou organizována do úrovní, které jsou odděleny lomítkem. Například senzor/1/teplota označuje teplotní senzor číslo 1 měřící teplotu.

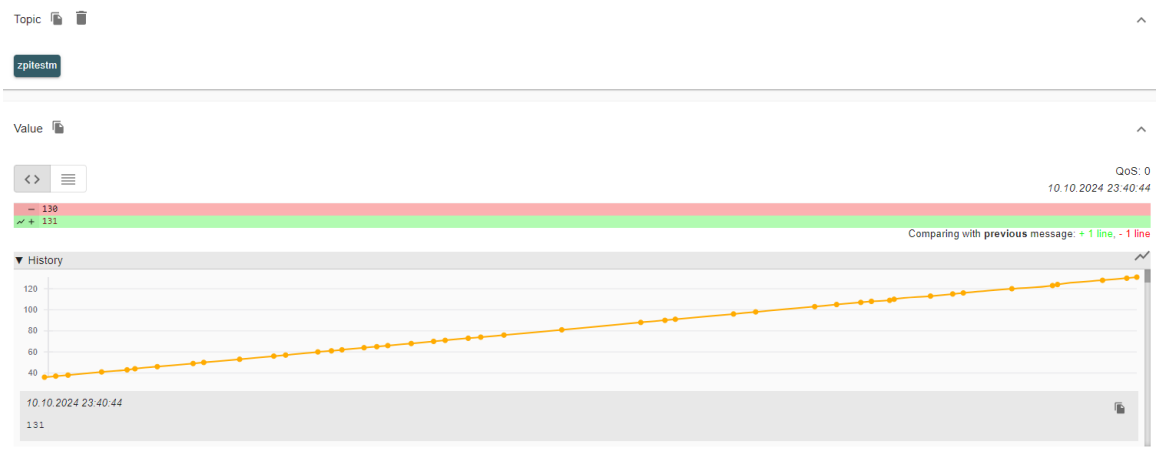
Témata jsou citlivá na velikost písmen, proto je doporučeno používat pouze malá písmena. Nepoužívejte mezery v názvech témat, místo toho používejte podtržítka nebo pomlčky. Použití znaků národních abeced není zakázáno, ale kvůli kompatibilitě s různými MQTT klienty a servery (které mohou mít různé úrovně podpory pro národní znaky) je často doporučeno používat pouze ASCII znaky. Vyhněte se používání speciálních znaků, jako jsou # a +, které mají v MQTT specifický význam.

Zástupné znaky (Wildcards) v MQTT:
Jednoúrovňový wildcard (+): Nahrazuje jednu úroveň v tématu. Například senzor/+/teplota bude odpovídat senzor/1/teplota, senzor/2/teplota atd. (+ lze použít vícekrát na různých úrovních)
Víceúrovňový wildcard (#): Nahrazuje všechny následující úrovně v tématu. Například senzor/# bude odpovídat senzor/1/teplota, senzor/2/vlhkost atd. (# lze použít jen jednou, na konci)

```
#MQTT communication example -public and subscribe
#sends every 5s topic <toppub> (counter)
#all received topic <topsub> shows on screen

from m5stack import *
from m5stack_ui import *
from uiflow import *
import wifiCfg
from m5mqtt import M5mqtt

screen = M5Screen()
screen.clean_screen()
screen.set_screen_bg_color(0x222222)

toppub = 'zpitestm'  #topic for publishing
topsub = 'zpitestm'  #topic for subscibing
topend = 'zpitestm'  #topic for lastwill

#wifiCfg.doConnect('zpikiv', '78086975')   #wifi connection

def sub_cb(topic_data):
  lcd.print(topic_data)
  pass

#mqtt – device name, MQTTserver, port, username, password, time-to-live
m5mqtt = M5mqtt('zpidev', 'test.mosquitto.org', 1883, '', '', 20)

m5mqtt.subscribe(str(topsub), sub_cb)  #substribe registration

m5mqtt.set_last_will(topend, '.Konec.')    #last will
m5mqtt.start()                             #start MQTT

m5mqtt.publish(str(toppub), str('.Start.'),  )  #first mesage
lcd.print('.Start.')


cnt=0
while(True):
  wait(5)
  m5mqtt.publish(str(toppub), str(cnt), )   #regular message
  cnt+=1

m5mqtt.deinit()
lcd.print('.Konec.')
```

MQTT client (pro PC) – MQTT explorer : https://mqtt-explorer.com/





MQTT server: test.mosquitto.org (pro neautentizované, nešifrované připojení – port 1883)

HTTP (HTTPS)

HTTP (hyper text transport protocol) je základní protokol pro webové služby na internetu - umožňuje přenos hypertextových dokumentů, jako jsou webové stránky, mezi servery a klienty (např. webové prohlížeče).

Http protokol běží na aplikační vrstvě (application Layer) v modelu OSI, Standardně používá port TCP/80. Je to bezestavový protokol, což znamená, že každý požadavek od klienta k serveru je

nezávislý a server neuchovává žádné informace o předchozích požadavcích. HTTP podporuje různé metody požadavků, jako jsou GET, POST, PUT, DELETE, HEAD, OPTIONS, TRACE a CONNECT.

Pro zabezpečení komunikace se používá HTTPS (HTTP Secure), které kombinuje HTTP s protokolem TLS (Transport Layer Security) nebo SSL (Secure Sockets Layer) pro šifrování dat.

Příklad využití webové služby pro zjištění aktuální teploty v Praze:
(zakomentovaná řádka je testovací http pro M5stack vracející „Hello M5 User“)

```
from m5stack import *
from m5ui import *
from uiflow import *
import urequests

setScreenColor(0x222222)

label0 = M5TextBox(69, 67, "Text", lcd.FONT_DejaVu24, 0xFFFFFF, rotate=0)

label0.setText('send request')
try:
  #req = urequests.request(method='GET', url='http://api.m5stack.com/v1', headers={'Content-Type':'text/html'})
  req = urequests.request(method='GET', url='https://wttr.in/Prague?format=%t', headers={'Content-Type':'text/html'})
  label0.setText('succeeded')  #http ok
  lcd.print((req.text), 0, 0, 0xffffff, rotate=0)  #req.text = http response, např. +14C
  print(req.text)
  gc.collect()
  req.close()
except:
  label0.setText('failed')  #not success
```

Automatizace

NodeRed

Node-red (Node-RED (nodered.org)) je nástroj pro vizuální programování, který umožňuje propojení (hardwarových) zařízení, API a online služeb pomocí jednoduchého drag-and-drop rozhraní. Je postaven na platformě Node.js. Editor je založený na prohlížeči a umožňuje snadné vytváření a úpravu flow pomocí široké škály uzlů. Díky podpoře JavaScriptu lze psát vlastní funkce přímo v editoru. Může běžet na různých zařízeních, od Raspberry Pi až po cloudové servery.

Existuje několik možností, jak **vyzkoušet Node-RED**:

IBM Cloud: IBM nabízí bezplatný hosting pro Node-RED. Stačí se zaregistrovat na IBM Cloud a vytvořit Node-RED instanci pomocí Cloud Foundry1. IoT Project - Installation of NodeRed on IBM Cloud (youtube.com)

Node-red Glitch - web/online node-red: https://glitch.com/~node-red-glitch

Oracle Cloud: Oracle poskytuje možnost vytvořit si bezplatnou virtuální mašinu, na které můžete Node-RED nainstalovat a provozovat2.

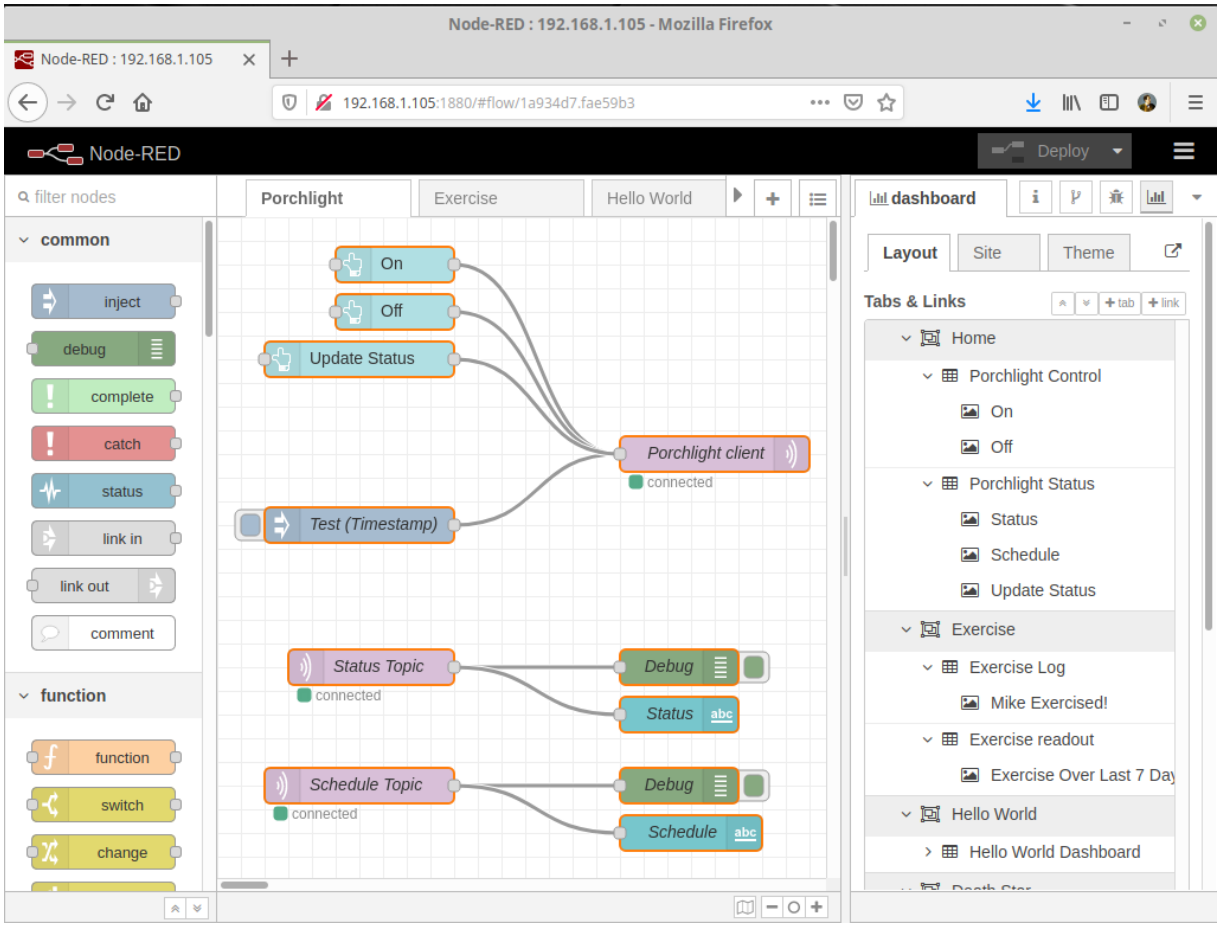
FlowFuse: Další možností je využít FlowFuse, což je platforma pro provozování Node-RED v cloudu. (nově již není zdarma)

Úvod do node-red (video, shlédnout): [Node-RED 101 (Štěpán Bechynský) (youtube.com)]

Node red (text): [Node-RED Programming Guide - Node RED Programming Guide (noderedguide.com)]

Node / uzel - základní stavební blok používaný k vytváření flow/toku. Každý uzel představuje konkrétní funkci nebo akci, kterou lze ve flow provést. Flow může obsahovat více propojených uzlů.

Message (Zpráva) - je objekt (JavaScriptu), který přenáší data mezi uzly ve flow. Obsahuje hlavní datový payload a další metadata. Payload je hlavní vlastnost zprávy, která obsahuje data, se kterými uzly ve flow pracují. Existují tři typy contextu (uložení dat – lišící se dosahem/platností): Node (uzel), Flow (proud) a Global (globální).



Příkladem vstupního uzlu může být inject (časové spouštění), mqtt (příchod subcribovaného mqtt topicu), http (http požadavek).

Příkladem funkčního uzlu jsou různé funkce na daty (function), doplňující http dotazy na externí zdroje (http request), rozvětvení flow (switch)

Příkladem výstupního uzlu může být mqtt (public zprávy), ladící výpisy (debug).

Dále je možné nainstalvat řadu rozšiřijících ulů a doplňků – např pro načítání údajů z různých webů, nebo generování webového rozhraní pro přístup/zobrazení dat, přístupy k databázím/cloudům a jiné.

Příklad automatizace osvětlení pomocí Node-RED

Cíl: Automatizovat osvětlení v domě tak, aby se světla zapínala při západu slunce a vypínala při východu slunce.

Krok 1: Instalace Node-RED (např. na váš počítač nebo Raspberry Pi)

Krok 2: Přidání uzlů
Inject Node: Tento uzel bude sloužit k inicializaci toku.
Sunset/Sunrise Node: Tento uzel získá informace o čase západu a východu slunce.
Switch Node: Tento uzel rozhodne, zda je čas zapnout nebo vypnout světla.
MQTT Node: Tento uzel bude komunikovat s vašimi chytrými žárovkami přes MQTT protokol.

Krok 3: Vytvoření toku
Připojte Inject Node k Sunset/Sunrise Node.
Připojte Sunset/Sunrise Node k Switch Node.
Připojte Switch Node k dvěma MQTT Nodes – jeden pro zapnutí světel a druhý pro vypnutí.

Krok 4: Konfigurace uzlů
Inject Node: Nastavte na spuštění při startu a každých 24 hodin.
Sunset/Sunrise Node: Zadejte vaši geografickou polohu.
Switch Node: Nastavte podmínky pro západ a východ slunce.
MQTT Nodes: Konfigurujte s detaily vašich chytrých žárovek.

Krok 5: Nasazení toku
Klikněte na tlačítko “Deploy” v Node-RED editoru a váš tok bude aktivní.

OpenHab

OpenHAB (open Home Automation Bus) je open-source platforma pro domácí automatizaci, která je nezávislá na konkrétní technologii nebo výrobci. Podporuje širokou škálu zařízení a systémů od různých výrobců, což umožňuje integraci různých technologií do jednoho řešení.

Umožňuje propojení a ovládání různých chytrých zařízení a systémů, jako jsou osvětlení, vytápění, bezpečnostní systémy, senzory a další.
Umožňuje vytváření automatizačních pravidel, která mohou spouštět akce na základě různých podmínek, jako je čas, stav zařízení nebo senzory1.
Poskytuje různé uživatelské rozhraní pro ovládání a monitorování zařízení, včetně webového rozhraní, mobilních aplikací a hlasových asistentů.

openHAB je napsán v jazyce Java. Lze jej nasadit na různé operační systémy, včetně Linuxu, Windows, macOS. Používá (modulární) OSGi moduly, což umožňuje dynamické přidávání a odebírání funkcionalit. Podporuje různé perzistentní backendy pro ukládání a dotazování dat, včetně relačních a časových databází. Po instalaci může openHAB automaticky detekovat zařízení v síti a umožňuje jejich snadnou konfiguraci. Umožňuje uživatelům definovat, jak budou zařízení v domácnosti uspořádána a vizualizována (Sitemaps).

Instalace a Základy OpenHab (video, shlédnout) - Automate Your Home with openHAB – YouTube

Open hub dokumentace (text): https://www.openhab.org/docs/

MyopenHAB je cloudová služba, která slouží jako doplněk k openHAB, open-source platformě pro domácí automatizaci. Tato služba umožňuje uživatelům přístup k jejich openHAB instancím odkudkoli na světě, což je užitečné pro vzdálené ovládání a monitorování domácí automatizace. Podporuje též zasílání notifikací na mobilní zařízení nebo e-mail, což je užitečné pro upozornění na různé události v

domácnosti. Umožňuje integraci s populárními cloudovými službami, jako jsou Google Assistant, Amazon Alexa a další.

Uživatelé se musí zaregistrovat na myopenHAB.org a propojit svou openHAB instanci s cloudovou službou. Po propojení (Nainstalujte openHAB Cloud Connector: Tento doplněk umožní propojení vašeho openHAB serveru s myopenHAB) mohou uživatelé přistupovat ke své openHAB instanci přes webové rozhraní nebo mobilní aplikaci.

HomeAssistant

Home Assistant je open-source platforma pro chytrý domov, která umožňuje integraci a automatizaci různých chytrých zařízení a služeb. Home Assistant můžete provozovat na různých zařízeních, jako je Raspberry Pi, počítač nebo NAS (Network Attached Storage). Konfigurace se provádí pomocí YAML souborů nebo uživatelsky přívětivého rozhraní.

HomeAssistant dokumentace (text): Documentation - Home Assistant (home-assistant.io)

Home assistant instalace, základy (video, en): Getting Started With Home Assistant In 2024: The Ultimate Guide (youtube.com)

Home assistant základy (video, cz): (26) VYTVÁŘÍME AUTOMATIZACE DOMÁCNOSTI LOKÁLNĚ BEZ CLOUDU! - Home Assistant - YouTube

Odkazy

M5stack core2 dokumentace: m5-docs (m5stack.com)

M5stack core2 instalační soubory: m5-docs (m5stack.com)

UIFlow web / IDE : https://flow.m5stack.com/

UIFlow dokumentace: https://docs.m5stack.com/en/uiflow/uiflow_web

OpenHub pro testování: myopenHAB

Micropython, dokumentace: https://docs.micropython.org/en/latest/library/machine.html

Online python: Online Python Compiler - online editor (onlinegdb.com)

MQTT client (MQTT explorer) : https://mqtt-explorer.com/

MQTT server - test.mosquitto.org

MQTT (online) broker HiveMQ: https://www.hivemq.com/company/get-hivemq/
MQTT Tutorial: An Easy Guide to Getting Started with MQTT (hivemq.com)

MQTT client online - hivemq: https://www.hivemq.com/demos/websocket-client/

Node-red: https://nodered.org/

Node-red Glitch (web/online) node-red: https://glitch.com/~node-red-glitch

Pomocné nástroje

Online formatovani kodu (pro tvorbu dokumentace): Online Syntax highlighter (pinetools.com)

Poznámky k Instalaci SW pro M5stack

M5stack core2 instalační soubory: m5-docs (m5stack.com)

Nainstalovat driver CH9102 a CP210x - kontrola seriového portu + win+X, správce zařízení, porty (COM a LPT)

UIFLOW FIRMWARE BURNING TOOL – nahráni systému, případně dalsích spec.programů např.HWtest