Obsluha a programování platformy Wemos D1 R32 s MCU ESP32

Michal Bidlo

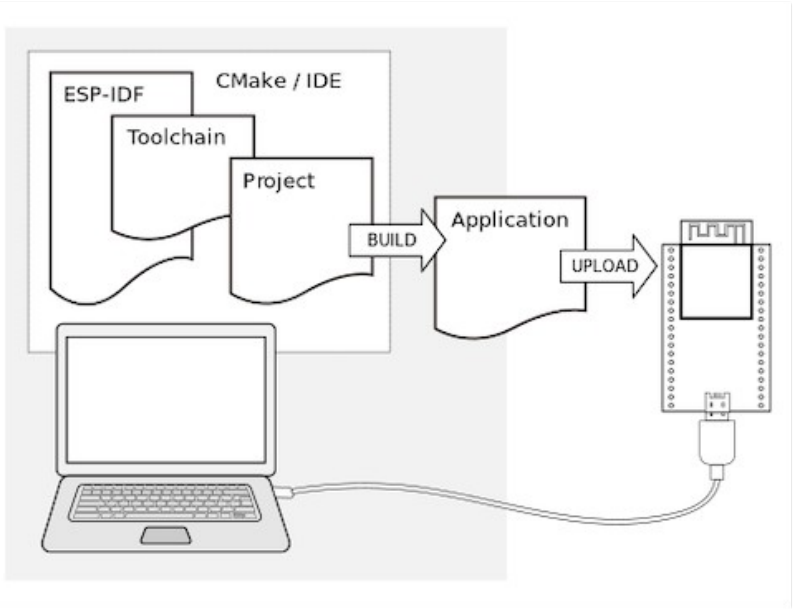
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Obsah

- Vývojové nástroje, možnosti programování
- Návod na instalaci ESP-IDF
- Architektura ESP32
- FreeRTOS
- Programování „na“ ESP IDF
- Vzorová aplikace

Vývoj aplikace pro ESP32

- Oficiální vývojové prostředí ESP-IDF firmy Espressif poskytující zejména
  - překladač gcc / g++ pro cílovou arch. ESP32 (toolchain),
  - systém pro správu projektů,
- a dále využívající
  - systém CMake + Ninja-build pro správu závislostí a sestavení aplikace,
  - vhodné IDE (volitelně).



Doporučení k instalaci ESP-IDF

- Stáhněte si Visual Studio Code (VSC) pro váš systém a nainstalujte jej (je pro Win, Lin i Mac):
  - https://code.visualstudio.com/download
- Dále spusťte VSC a nainstalujte rošíření ESP-IDF (prostředí k programnování MCU od f. Espressif) podle tohoto návodu:
  - https://github.com/espressif/vscode-esp-idf-extension/blob/master/docs/tutorial/install.md
- POZN.: V Linuxu je ještě předtím nutné doinstalovat příslušné závislosti, zejména:
  - git, cmake, ninja (ninja-build), python (python3)
- Pokud chybí něco dalšího, budete na to upozorněni.
- Po instalaci všeho potřebného restartujte VSC.

Doporučení k instalaci ESP-IDF

- Vývojový toolchain pro ESP32 je možné nainstalovat i samostatně (bez VSC) s následnou „command line“ obsluhou. Návod vizte např. zde:
  - https://medium.com/coinmonks/espressif-esp32-tutorial-programming-esp-idf-29f297837d76
- ESP32 lze programovat i v Arduino IDE, ale budete závislí pouze na prostředcích integrovaných k této platformě v podpůrných knihovnách Arduino IDE a nevyužijete plně možností ESP32. Více k tomu vizte na stránce o projektu Arduino core for the ESP32:
  - https://github.com/espressif/arduino-esp32

Další možnost

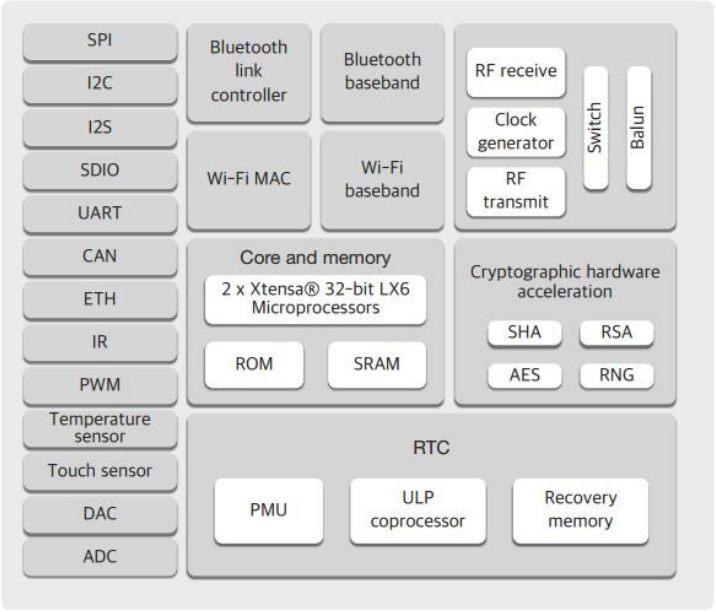
- *Platform*IO (opět jako rozšíření do VSC) – jedná se o nástavbu nad IDF pro vývoj IoT aplikací obecně, návod k instalaci od dr. Mrázka vizte zde:
  - https://www.youtube.com/watch?v=v1lCXLQuA9s

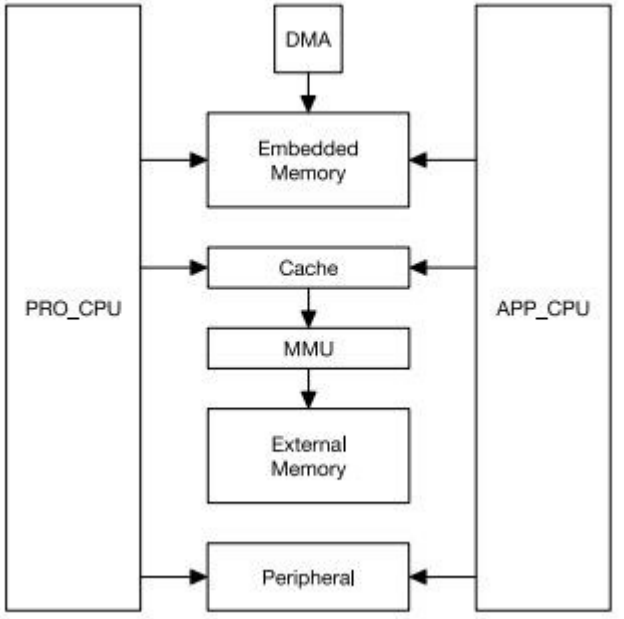
- **Pro potřeby řešení projektů na ESP32** je možné použít libovolné prostředí, které vám bude vyhovovat, pokud nejste explicitně omezeni zadáním. V případě nejistoty se informujte u vedoucího vašeho tématu.

- **V IS FIT je založeno fórum pro dotazy související s problematikou ESP32 v IMP.**

Architektura ESP32

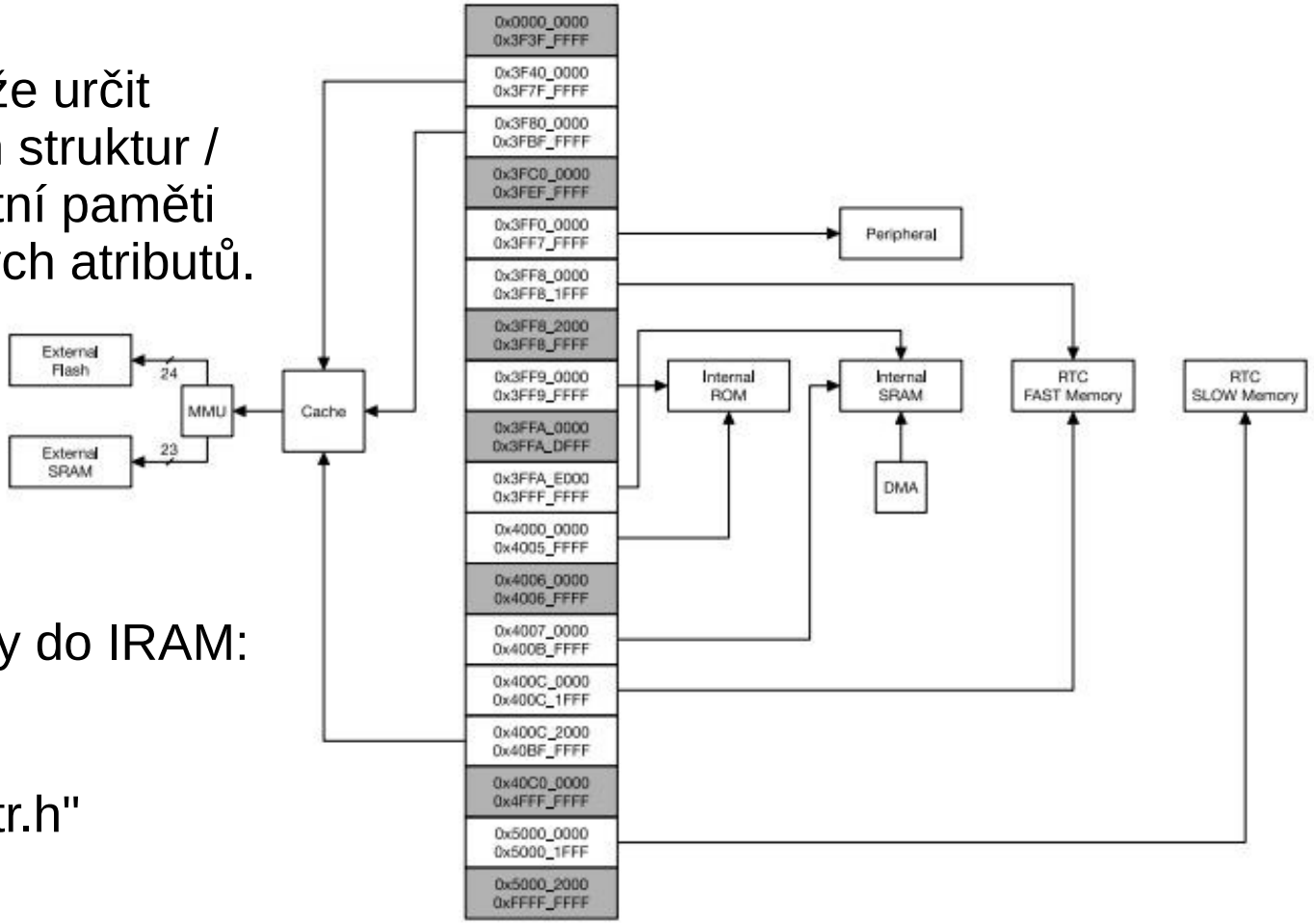
- **2x** CPU Xtensa XL6 + řada periferií (včetně např. BT a WiFi)
- 4MB **externí flash** pro uložení programu (code execute enable), externí R/W SRAM pro data, přístup prostřednictvím cache přes rozhraní QSPI
- Několik rychlých interních pamětí pro různé účely a režimy





Paměťová hierarchie ESP32

- Programátor může určit umístění různých struktur / funkcí do konkrétní paměti pomocí příslušných atributů.



Příklad: ISR vždy do IRAM:

```
#include "esp_attr.h"

void IRAM_ATTR gpio_isr_handler(void* arg)
{
    const static DRAM_ATTR uint8_t INDEX_DATA[] = { 45, 33, 12, 0 };
    const static char *MSG = DRAM_STR("I am a string stored in RAM");
}
```

Programování ESP32: microPython

- (*uPython*) ...je asi nejjednodušší způsob, jak si vyzkoušet základy práce s ESP32.
- Jedná se o interpret jazyka Python pro MCU poskytující řadu funkcí (včetně řízení periferií).
- Postup instalace:
  - V sekci Releases si stáhněte poslední verzi binárky (.bin) uPythonu pro ESP32 a postupujte podle zde uvedeného návodu: https://micropython.org/download/esp32/
  - Bude-li *esptool* hlásit, že funkce *erase_flash* není podporována, proveďte to pomocí ESP IDF:

    idf.py -p PORT erase-flash

    kde PORT je např. v linuxu /dev/ttyUSB0. **Spouštění idf.py je nutno provést z adresáře nějakého projektu (např. hello_world) zkopírovaného do místa, kde se nachází prostředí (složka) esp-idf.**
  - Pozn: *esptool* je též k dispozici jako standardní balíček v moderních distribucích linuxu.

Programování ESP32: microPython

- Ucelený tutoriál k uPythonu na ESP32 vizte např. zde: https://boneskull.com/micropython-on-esp32-part-1/

- Thonny: https://thonny.org/ - jednoduché IDE pro Python umožňující i pohodlné programování pro vestavěný yPython (v linuxu opět možno instalovat z balíčku distribuce), pro aktuální verzi vizte návod zde (Win, Linux i MAC): https://randomnerdtutorials.com/getting-started-thonny-micropython-python-ide-esp32-esp8266/

  nebo videonávod zde: https://www.youtube.com/watch?v=lvmNLuHj25o

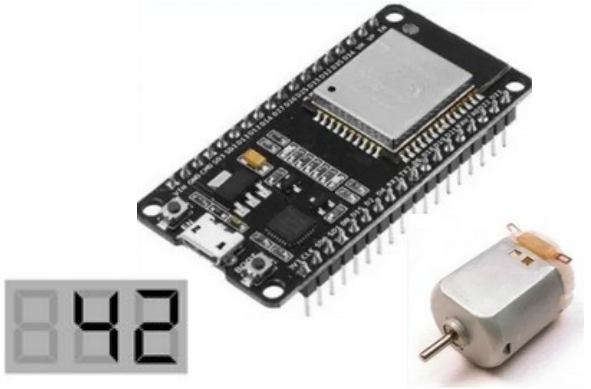
- Pro rychlé seznámení s uPythonem na ESP32: https://docs.micropython.org/en/latest/esp32/quickref.html
- ...i uPython umožňuje „bare metal“ programování – tj. low-level práce s registry MCU atd.: https://docs.micropython.org/en/latest/esp32/tutorial/peripheral_access.html

Zvyšující se složitost vestavěných systémů

- Dosud v IMP
  - Jednojádrový MCU
  - Programnování na úrovni registrů (bare metal)
  - Kombinace pooling + interrupts, příp. DMA
  - Využití knihovních funkcí (SDK, příp. třetí strany)
- Další požadavky
  - Řízení komplexních periferií (USB, WiFi…)
  - Potřeba vyššího výpočetního výkonu (např. pro potřeby zabezpečení, autentizace, šifrování...)
  - Využití výhod paralelního zpracování
    → vícejádrové procesory
  - **Toto se „ručně“ dá zvládnout už jen těžko...**

„Efektivní“ řízení vestavěné aplikace

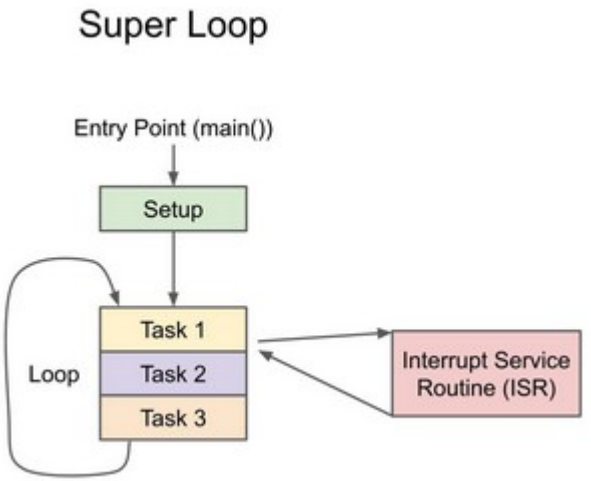
- **Motivační příklad** (hypotetický): řízení motoru s otáčkoměrem a výpisem otáček na displej
  - Řízení motoru pomocí PWM
  - Zobrazení číslic na 7-seg. displeji nutno řešit v časovém multiplexu
  - Snímání otáček vhodným senzorem
  - Aktualizace hodnoty displeje 1x za sekundu
  - Vstup od uživatele přes tlačítka



„Efektivní“ řízení vestavěné aplikace

- SW řízení pouze za pomoci MCU

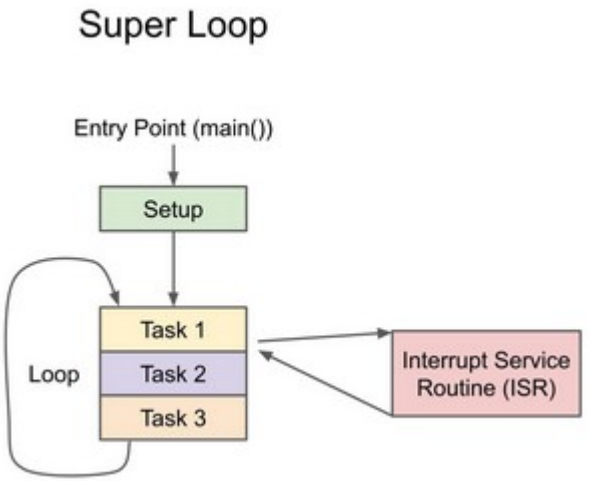
  (bare metal programming)



- Problémy:
  - Řada funkcí volaných z *main()*
  - Synchronní / asynchronní spouštění různých úloh / ISR
  - Některé funkce „běží“ nepřetržitě (ve smyčce), jiné naopak potřebujeme méně často.
  - Čtení ze senzorů není okamžité, nutnost čekání.
  - Mnohdy je nutné *garantovat* „včasnost“ odezvy.
- Jak to celé řešit efektivně?

Řízení vestavné aplikace nižší složitosti

- Jednoduché operace v *main()* (např. detekce stisku tlačítka), příp. lépe formou obsluhy přerušení
- Periodicky se opakující funkce jako ISR spouštěné časovačem



- Čekání pomocí *delay()* – možné, ale obvykle blokuje! Lépe použít neblokující variantu, v Arduinu např. millis(), ale zde je nutno sledovat podmínku…

  if (*millis() - previousTime > 1 sec) ….*

- **Reálné aplikace jsou však dnes mnohem složitější a takové řešení by bylo drahé!!**

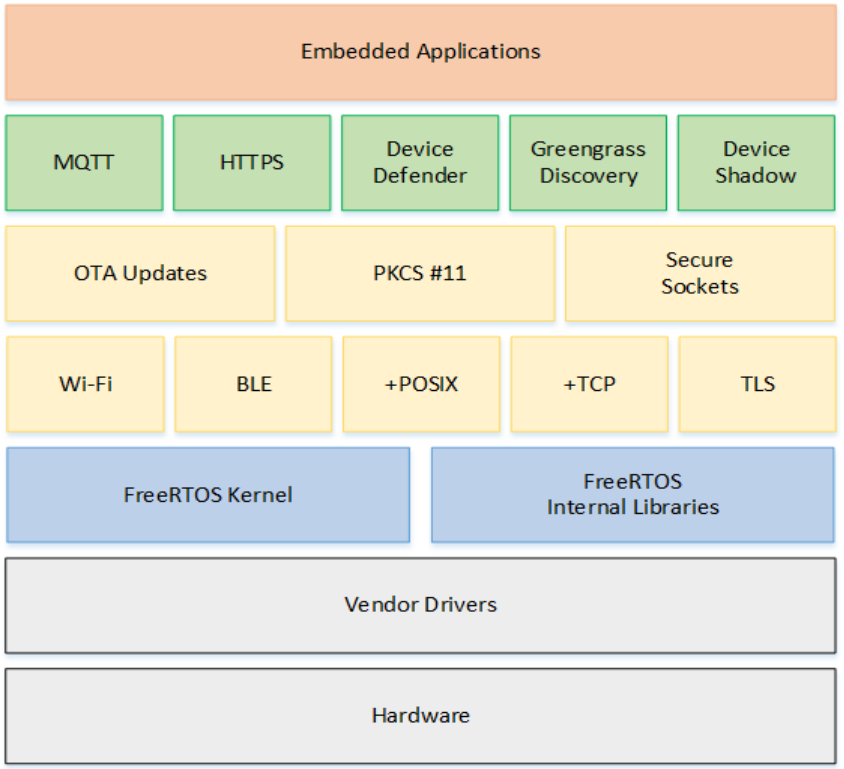
Operační systém pro MCU?

- Pro efektivní řešení složitých úloh je vhodné inspirovat se tam, kde již existuje vhodná infrastruktura (ovladače HW, správa paměti, procesů, vláken…) – **tedy v operačním systému.**
- Specifika a požadavky na OS pro vestavěné systémy (VS):
  - Přizpůsobení na omezenou paměť a výpočetní výkon.
  - ***Deterministický* plánovač úloh** jako stěžejní část OS pro VS
    - \+ garantovaná latence odezvy v daných situacích!
  - Možnost konkrétního **nastavení priorit různým úlohám** a „**Time slicing**“ – přidělování výpočetního času úlohám, příp. podpora vícejádrového CPU (SMP), je-li k dispozici.
  - Ovladače přizpůsobeny výbavě MCU, obvykle jako součást OS od konkrétního výrobce.

- Tento koncept poskytují tzv. **Real-Time OS** – název odvozen od požadavku na precizní zacházení s časem během činnosti systému.

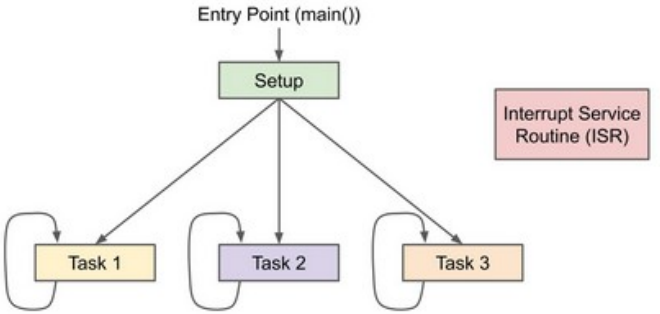
FreeRTOS – RTOS na ESP32

- Open-source implementace RTOS – **FreeRTOS je jádrem (kernel)** tohoto OS implementujícím zejména plánovač úloh, řízení priorit a komunikace mezi úlohami, to vše s ohledem na kompaktnost a nízkou spotřebu.
- Koncepce FreeRTOS pro ESP32



Aplikace s motorem a displejem pomocí OS

- Jaký bude rozdíl při použití RTOS oproti „super loop“ z pohledu programátora?



- Každá úloha (Task ~ vlákno ve FreeRTOS) bude představovat samostatný „program“ – funkci s vlastní smyčkou a zajištěním komunikace s ostatními úlohami.

```
DisplayTask()
{
    // Init ports etc.
    DisplayInit();
    // This will occur every 20 milliseconds
    while(1)
    {
        SwitchSegment();
        // Block DisplayTask function for 20 milliseconds
        // (this is a FreeRTOS function)
        vTaskDelay(MS_20);
    }
}
```

```
DisplayUpdateTask()
{
    // This will occur every 20 seconds
    while(1)
    {
        // Read from a sensor
        int rpm = ReadValue();
        UpdateDisplay(rpm)
        // Block sendSMS function for 1 second
        vTaskDelay(MS_1000);
    }
}
```

Vývojové prostředí **ESP IDF**

- **Oficiální framework dodávaný k ESP32 výrobcem**.
- **Esp**ressif **I**oT **D**evelopment **F**ramework – sada konfiguračních a překladových nástrojů, ovladačů, implementace FreeRTOS a dalších podpůrných knihoven, funkcí a příkladů pro vývoj IoT aplikací na platformě ESP32.
- Důležité odkazy:
  https://docs.espressif.com/projects/esp-idf/en/latest/esp32/
  https://github.com/espressif/esp-idf (zde jsou src, příklady,...)

Vzorová aplikace

- Vizte archiv esp32_demo_imp.zip v IS FIT.

Příklad k nástroji *menuconfig* (pro pokročilé)

- Vizte archiv esp32_demo_menuconfig.zip v IS FIT.

Reference

- ESP32 obecně (architektura, principy, programování)
  https://www.exploreembedded.com/wiki/Overview_of_ESP32_features._What_do_they_practically_mean%3F
  https://gitdemo.readthedocs.io/en/latest/general-notes.html
  https://makeabilitylab.github.io/physcomp/esp32/
- FreeRTOS
  https://www.freertos.org/Documentation/RTOS_book.html
  https://github.com/ShawnHymel/introduction-to-rtos
- Paměťová hierarchie ESP32
  https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/memory-types.html
  https://blog.espressif.com/esp32-programmers-memory-model-259444d89387
- MicroPython
  https://learn.sparkfun.com/tutorials/micropython-programming-tutorial-getting-started-with-the-esp32-thing/all