

**TECHNICKÁ UNIVERZITA V KOŠICIACH**  
**FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

**Bezpečná aktualizácia firmvéru v senzorovej sieti na báze  
ESP32**  
**Diplomová práca**

**2021**

**Martin Chlebovec, Bc.**

**TECHNICKÁ UNIVERZITA V KOŠICIACH**  
**FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

**Bezpečná aktualizácia firmvéru v senzorovej sieti na báze  
ESP32**  
**Diplomová práca**

Študijný program: Počítačové siete  
Študijný odbor: Informatika  
Školiace pracovisko: KEMT  
Školiteľ: prof. Ing. Miloš Drutarovský, CSc.

**2021 Košice**

**Martin Chlebovec, Bc.**

## **Abstrakt v SJ**

Hlavnou témou tejto diplomovej práce bezpečnej aktualizácie firmvéru je využitie mikrokontroléru ESP32 v úlohe senzorového uzla, ktorého základ bol navrhnutý už v bakalárskej práci. Popri zbere údajov z meteorologického senzora dokáže po rozšírení programovej implementácie mikrokontrolér ESP32 aktualizovať svoj firmvér (jednouúčelový spustiteľný program) zo vzdialeného servera / sieťového umiestnenia.

Pri procese vzdialených aktualizácii firmvéru, ale aj softvéru, operačných systémov všeobecne sa kladie dôraz na bezpečnosť, ktorá spočíva v garancii integrity predmetnej aktualizácie. Zaručuje tak, že aktualizácia nebola pozmenená, alebo iným spôsobom upravená a že bola vydaná dôveryhodným vydavateľom.

Diplomová práca sa zoberá priblížením možností aktualizácie mikrokontrolerovej platformy ESP32 v rôznych podporovaných programovacích jazykoch, ktoré sa líšia predovšetkým možnosťami a obmedzeniami aktualizácie, ktoré sú dostupné v lokálnej sieti, alebo cez internet. Podstatnú časť diplomovej práce tvorí opis možností a mechanizmov bezpečnosti vo frameworku ESP-IDF, ktorými je možné garantovať integritu aktualizácie firmvéru.

Pre zvýšenú úroveň bezpečnosti využíva mikrokontrolér ESP32 v mojej programovej implementácii viacero bezpečnostných mechanizmov pre samotné overenie integrity aktualizácie firmvéru, ale aj prenos aktualizácie cez bezpečný komunikačný kanál, ktorý zabezpečuje šifrovanie medzi mikrokontrolérom a serverom, ktorý aktualizáciu distribuuje.

## **Kľúčové slova v SJ**

ESP32, Espressif Systems, MCU, OTA, WiFi, mikrokontrolér, IoT, firmvér, aktualizácia

## **Abstrakt v AJ**

The main topic of this diploma thesis of safe update of firmware is the use of the ESP32 microcontroller in the role of a sensor node, the basis of which was already designed in the bachelor's thesis. In addition to collecting data from the weather sensor, the ESP32 microcontroller can update its firmware (single-purpose executable program) from a remote server / network location after extending the software implementation.

In the process of remote updates of firmware, but also software, operating systems in general, the emphasis is on security, which consists in guaranteeing the integrity of the update in question. This ensures that the update has not been altered or otherwise modified and that it has been released by a trusted publisher.

The diploma thesis deals with the possibilities of updating the microcontroller platform ESP32 in various supported programming languages, which differ mainly in the possibilities and limitations of the update, which are available in the local network or via the Internet. An essential part of the thesis is a description of the possibilities and mechanisms of security in the ESP-IDF framework, which can guarantee the integrity of the firmware update.

For an increased level of security, the ESP32 microcontroller in my software implementation uses several security mechanisms to verify the integrity of the firmware update itself, but also to transmit the update over a secure transmission channel that provides encryption between the microcontroller and the server distributing the update.

## **Klíčové slova v AJ**

ESP32, Espressif Systems, MCU, OTA, WiFi, microcontroller, IoT, firmware, update

# Zadanie práce

60848

**TECHNICKÁ UNIVERZITA V KOŠICIACH**  
FAKULTA ELEKTROTECHNIKY A INFORMATIKY  
Katedra elektroniky a multimediálnych telekomunikácií

## ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **Informatika**  
Študijný program: **Počítačové siete**

Názov práce:

**Bezpečná aktualizácia firmvéru v senzorovej sieti na báze ESP32**

Secure firmware update in sensor network based on ESP32

Študent: **Bc. Martin Chlebovec**  
Školiteľ: **prof. Ing. Miloš Drutarovský, CSc.**  
Školiace pracovisko: **Katedra elektroniky a multimediálnych telekomunikácií**  
Konzultant práce:  
Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:

Na základe dostupných informácií naštudujte mechanizmy umožňujúce realizovať bezpečnú aktualizáciu firmvéru na platforme mikropočítača ESP32. V práci opíšte hardvérové aspekty ESP32, ktoré sú pri aktualizácii využívané. Opíšte tiež využité kryptografické algoritmy a softvérové nástroje, ktoré má vývojár pre platformu ESP32 na aktualizáciu firmvéru k dispozícii. S využitím dostupných technických prostriedkov na báze ESP32 vytvorte demonštračnú aplikáciu senzorového uzla, ktorá umožní aktualizovať firmvér s využitím WiFi rozhrania a snímať údaje z vhodného senzora. Experimentálne overte funkčnosť navrhnutého riešenia a analyzujte možnosti zníženia celkovej spotreby senzorového uzla pri zachovaní možnosti bezpečnej aktualizácie firmvéru.

Jazyk, v ktorom sa práca vypracuje: slovenský  
Termín pre odovzdanie práce: 23.04.2021  
Dátum zadania diplomovej práce: 30.10.2020

.....  
prof. Ing. Liberios Vokorokos, PhD.  
dekan fakulty

## Čestné vyhlásenie

Vyhlasujem, že som celú diplomovú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Košice, 21. marca 2021

.....

vlastnoručný podpis

## **PodĎakovanie**

V prvom rade sa chcem srdečne poďakovať vedúcemu práce prof. Ing. Milošovi Drutarovskému, CSc. za cenné rady, vysvetlenie odborných termínov, konzultácie súvisiace s technickou stránkou diplomovej práce.

# Obsah

Zoznam obrázkov .....	10
Zoznam tabuliek .....	11
Zoznam symbolov a skratiek .....	12
Úvod .....	14
1. ESP32 .....	15
1.1. Proces spustenia systému ESP32 .....	17
1.2. Pamäť mikrokontroléru ESP32 .....	18
1.2.1. Embedded (vstavaná) pamäť .....	18
1.2.2. Externá pamäť .....	20
1.2.3. Partition Table .....	21
1.2.4. eFuses .....	22
1.3. ULP .....	23
1.4. Vývojové kity, samostatný čip ESP32 .....	24
1.5. ESP-IDF .....	26
1.6. Arduino Core .....	27
2. OTA .....	29
2.1. OTA v Arduino Core .....	30
2.2. OTA v ESP-IDF .....	32
2.3. Metóda digitálneho podpisu .....	35
2.3.1. Digitálny podpis – implementácia v prostredí ESP-IDF .....	35
2.4. Secure Boot v ESP-IDF .....	38
2.4.1. Secure Boot - implementácia v prostredí ESP-IDF .....	39
3. Použitý hardvér a softvér .....	42
3.1. Úpravy webového rozhrania .....	45
3.2. Vytvorenie a úpravy programu v ESP-IDF .....	46
3.3. Minimálna schéma zapojenia .....	50
Záver .....	53



Zoznam použitej literatúry .....	54
Prílohy .....	58

## Zoznam obrázkov

Obr. 1 Harvardská architektúra mikropočítača .....	16
Obr. 2 Mapa pamäte ESP32 .....	20
Obr. 3 Tabuľka partícií s tromi aplikačnými partíciami .....	22
Obr. 4 ESP32-DevKitC.....	25
Obr. 5 Upozornenie operačného systému Windows na novú aktualizáciu .....	29
Obr. 6 Sieťový OTA port v prostredí Arduino IDE.....	30
Obr. 7 OTA Web Updater .....	32
Obr. 8 Menuconfig – konfiguračné menu projektov v ESP-IDF.....	33
Obr. 9 Grafická vizualizácia eliptickej krivky NIST256p .....	36
Obr. 10 Podpísanie firmvéru súkromným kľúčom v konzolovej aplikácii ESP-IDF .....	37
Obr. 11 Sumár eFuses zobrazených cez nástroj esefuse.py.....	39
Obr. 12 Bloková schéma procesu Secure Boot po bootovaní firmvéru .....	41
Obr. 13 Bosch BME280 - senzor teploty, tlaku, vlhkosti vzduchu.....	42
Obr. 14 Vizualizácia nameraných údajov vo webovom rozhraní .....	45
Obr. 15 Grafické rozhranie webstránky pre nahranie OTA firmvéru cez HTML formulár .....	46
Obr. 16 Vlastné konfiguračné menu pre voľbu I2C parametrov a adresy, režimu BME280 .....	49
Obr. 17 Minimálna schéma senzorového uzla .....	52

## Zoznam tabuliek

Tab. 1 Prevádzkové režimy mikrokontroléru ESP32 a stav periférii .....	23
Tab. 2 Rozdiel offsetu aplikačných partícií po a pred posunutím .....	37
Tab. 3 Pripojenie vývodov ESP32 k vývodom použitých periférii .....	44
Tab. 4 Tasky senzorového uzla pre odosielanie a načítanie dát z webového rozhrania.....	47

## Zoznam symbolov a skratiek

ADC	Analógovo-digitálny prevodník
AES	Advanced Encryption Standard
ALU	Aritmeticko-logická jednotka
AP	Prístupová bod
API	Rozhranie pre programovanie aplikácii
BLE	Bluetooth Low Energy
BLK	Označenie bloku jednorázovo programovateľnej pamäte eFuse
DMA	Priamy prístup do pamäte
DPS	Doska plošných spojov
DRAM	RAM pamäť dát
eFuse	Elektronická poistka – jednorázovo programovateľná pamäť
ESP-IDF	Espressif – IoT Development Framework
FTDI	Future Technology Devices International – typ USB-UART prevodníku
FTP	File Transfer Protocol
GPIO	Vstupno-výstupný vývod
HTML	Hypertextový značkový jazyk
HTTP	Hypertextový prenosový protokol
HTTPS	Zabezpečený hypertextový prenosový protokol
I2C	Intel-Integrated-Circuit - dvojžilová obojsmerná zbernica
IEEE	Inštitút elektrických a elektronických inžinierov (organizácia, norma)
IIR	Filter s nekonečnou impulznou odozvou, vyžaduje spätnú väzbu
IO	Vstup-výstup
IoT	Internet vecí
IRAM	RAM pamäť inštrukcií

LSB	Less Significant Bit (Najmenej významný bit)
MMU	Jednotka správy pamäte
OTA	Over-The-Air – metódy popisujúce spôsoby aktualizácie firmvéru, konfigurácie
PHP	Hypertextový preprocesor (skriptovací jazyk)
PoE	Power over Ethernet
PSRAM	Pseudo Static RAM
QSPI	Štandard pre štvornásobné zrýchlenie SPI zbernice
RAM	Operačná pamäť, energeticky závislá
ROM	Pamäť len na čítanie, energeticky nezávislá
RTC	Hodiny reálneho času
SCL	Synchronizačné hodiny (hodinový signál)
SDA	Synchronizované dáta
SPI	Synchrónne sériové periférne rozhranie
SPIFFS	Súborový systém na flash disku
SRAM	Static Random Access Memory (pozostáva z preklápacích obvodov)
SSR	Solid-state relé (typ relé s triakom spínaným cez tranzistor, bez mechanickej časti)
UART	Univerzálny asynchrónny prijímač a vysielateľ
ULP	Režim nízkeho odberu elektrickej energie
USB	Univerzálna sériová zbernica
USB-UART	Prevodník USB signálov na UART rozhranie

## Úvod

Komunikačnými sieťami sa prenášajú rôzne dáta. Líšia sa veľkosťou, obsahom, dôležitosťou aj sieťou, akou sú prenášané. Pre každú kategóriu prenášaných dát sa postupom času vyvinul štandard, či norma, ktorá dokázala dáta chrániť. Vyvinuli sa osobité mechanizmy, prípadne ich kombinácia, ktorá zaručuje bezpečný prenos, ale i spracovanie údajov koncovým zariadením používateľa. Osobné údaje, bankové transakcie, či interné firemné údaje podliehajú maximálnemu stupňu zabezpečenia a kladú požiadavky na samotné vybavenie koncových zariadení, ale aj siete, ktorou sú dáta prenášané.

Vysokú úroveň bezpečnosti si vyžadujú aj aktualizácie s ktorými sa najčastejšie stretávame v operačných systémoch či už na úrovni systémových aktualizácií, alebo aktualizácii pre aplikácie, ktoré sú pod daným operačným systémom nainštalované. Osobitú kategóriu aktualizovaných zariadení tvoria mikrokontroléry a mikropočítače, ktoré môžu byť aktualizované cez sieť, internet, čo je potencionálne nebezpečné prostredie pre prenos dôležitých údajov. Pri takýchto zariadeniach je bezpečnosť maximálna priorita, nakoľko mikrokontroléry môžu obsluhovať dôležité systémy v priemysle, výrobe, inteligentnom riadení, či zbere údajov zo senzorov.

Stiahnutím a spustením podvrhnutého, či inak upraveného nebezpečného firmvéru na mikrokontroléri by došlo k plnej kontrole mikrokontroléru treťou osobou – útočníkom, ktorý daný firmvér vyhotovil, alebo upravil. Pre podobné prípady a typy útokov existujú bezpečnostné mechanizmy, ktoré sa aplikujú na úrovni mikrokontroléru (klienta), servera, ktorý danú aktualizáciu distribuuje, ale existuje aj požiadavka na sieť a zabezpečené spojenie medzi klientom a serverom.

Diplomová práca sa venuje mechanizmom pre zabezpečenie integrity firmvéru prostredníctvom využitia metódy digitálneho podpisu v kapitole 2.3 a 2.3.1. Kapitola popisuje programovú implementáciu pre mikrokontroler ESP32 v úlohe senzorového uzla a rozšírenie mechanizmu digitálneho podpisu o hardvérovú funkcionality Secure Boot v kapitole 2.4 a 2.4.1., ktorá spustí iba dôveryhodný softvérový Bootloader obsiahnutý vo flash pamäti, čím zabezpečuje bootovací proces.

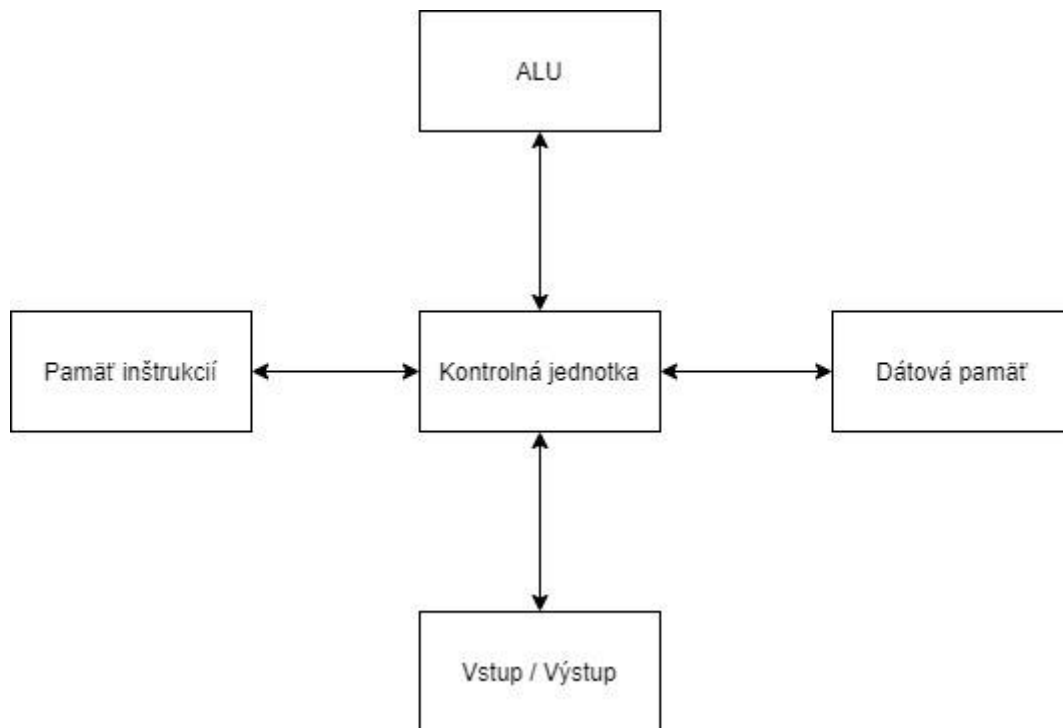
## 1. ESP32

ESP32 je mikrokontrolér [1] z produkcie čínskej firmy Espressif Systems. Je určený predovšetkým pre aplikácie internetu vecí a rôzne projekty vyžadujúce konektivitu s okolitými zariadeniami, internetom. Mikrokontrolér je vybavený WiFi (2.4 GHz) a Bluetooth konektivitou, pričom obe technológie zdieľajú spoločnú plošnú anténu na DPS (Doska plošných spojov), čím sa výrazne redukuje aj samotná veľkosť mikrokontroléru a použitie technológií je univerzálne s kompatibilným hardvérom.

WiFi modem mikrokontroléru ESP32 podporuje štandardy IEEE (Inštitút elektrických a elektronických inžinierov) 802.11 b/g/n, čím dosahuje mikrokontrolér teoretickú rýchlosť prenosu až 300 Mbit/s. Podporuje protokol ESP-NOW, ktorý umožňuje komunikáciu medzi mikrokontrolérmi z produkcie Espressif Systems. Zaujímavosťou technológie je, že nevyžaduje smerovač, ani prístupový bod, vyžaduje však párovanie obdobne ako pri technológii Bluetooth.

Použitie protokolu ESP-NOW je energeticky výhodné. Bluetooth technológia vo verzii 4.2 s podporou BLE (Bluetooth Low Energy) je vhodná pre aplikácie pre prevádzku na batériu a má spätnú kompatibilitu aj so zariadeniami so staršími verziami technológie Bluetooth. Rovnako tak novšie zariadenia s verziou Bluetooth 5.0 majú spätnú kompatibilitu a môžu komunikovať s ESP32. BLE je využívané predovšetkým pri operáciách skenovania zariadení v dosahu, pri vysielaní v cielenom / Beacon (jednosmerný maják) móde, podporuje viacnásobné spojenia.

V závislosti od verzie ESP32 [2] čipu sa mikrokontrolér líši veľkosťou osadenej flash pamäte (4 až 16 MB), podporou, prípadne dostupnou - osadenou externou RAM (Random Access Memory) pamäťou na plošnom spoji a rôznymi inými parametrami, napríklad počtom jadier procesora. Mikrokontrolér je najčastejšie vybavený dvojjadrovým, pri istých verziách čipu jednojadrovým 32-bitovým procesorom Harvardskej architektúry Tensilica Xtensa L6 s taktom 160 až 240 MHz. Architektúra má oddelenú pamäť pre dáta (údaje s ktorými mikrokontrolér pracuje) a inštrukcie (riadiaci program pre mikrokontroléru). Harvardská architektúra je vizualizovaná blokovou schémou na obrázku Obr. 1.



Obr. 1 Harvardská architektúra mikropočítača

Tento procesor môže realizovať výpočty, komunikáciu so senzormi a perifériami, ktoré sú ku mikrokontroléru pripojené cez podporované zbernice. Zároveň však riadi a obsluhuje aj WiFi / Bluetooth stack pre zabezpečenie konektivity. Za správu [3] a obsluhu WiFi / Bluetooth stacku zodpovedá jedno z jadier procesora u ktorého má WiFi / Bluetooth stack maximálnu prioritu. Jadrá sa najčastejšie označujú ako Core 0 (APP\_CPU) a Core 1 (PRO\_CPU).

Core 0 - nazývame aj aplikačný procesor spúšťa prioritne používateľskú aplikáciu, Core 1 - procesor protokolu sa stará o WiFi / Bluetooth stack, udržiava spojenie s AP (Prístupovým bodom) a zabezpečuje konektivitu. Zároveň však môže obsluhovať aj používateľskú aplikáciu (ak ju dokáže v reálnom čase obslúžiť). Výber APP a PRO procesora je voliteľný, štandardne je Core 1 vždy PRO\_CPU. Oba procesory zdieľajú spoločnú cache pamäť (64 kB) a môžu pristupovať na rovnaké miesto v pamäti. Procesor dokáže spustiť používateľské programy napísané v rôznych jazykoch s podporou kompilácie programu pre túto platformu, napríklad Arduino Core (Wiring), ESP-IDF (Espressif IoT Development Framework), Mongoose OS, MicroPython, Lua, Node.js, NodeMCU.

Espressif Systems zabezpečuje a dohliada na vývoj vlastného frameworku ESP-IDF a rovnako aj pre Arduino Core, ktoré nebolo producentom čipov ESP32 podporované pri generačne predchádzajúcej platforme ESP8266. Toho času sa vývoja Arduino Core ujali fanúšikovia a nadšenci Arduina a mikrokontrolérov.



## 1.1. Proces spustenia systému ESP32

Spustenie mikrokontroléru ESP32 pozostáva z viacerých fáz [4]:

1. Hardvérový Bootloader natiahne do RAM pamäte softvérový Bootloader, ktorá sa nachádza na offsete 0x1000 flash pamäte.
2. Softvérový Bootloader načíta tabuľku partícií a hlavnú aplikáciu z dostupnej (aplikačnej) partície s podporou bootovania. Aplikácia obsahuje segmenty v RAM pamäti a určité segmenty (read-only) mapované vo flash pamäti
3. Aplikácia sa spustí a následne aj druhé jadro procesora, RTOS plánovač

### Hardvérový Bootloader

Po reštarte mikrokontroléru ESP32 sa okamžite spustí PRO\_CPU, pričom APP\_CPU je držané v reštarte. PRO\_CPU vykonáva celú inicializáciu systému. Na základe dôvodu reštartu (Softvérový reštart, Watchdog reštart, reštart z dôvodu zobudenie hlavného čipu ESP32) dokáže PRO\_CPU prispôbiť výpis na sériový monitor a zároveň prepnúť hlavný čip do určitého režimu (pre stiahnutie programu), nahrávanie programu a iné.

Ak načítanie kódu z flash pamäte zlyhá, hardvérový Bootloader rozbalí Basic (TinyBasicPlus) interpreter (ROM Console) do RAM pamäte a spustí ho [5]. Nástroj slúži predovšetkým na debugovanie. Dokáže vykonávať základnú obsluhu GPIO (vstupno-výstupný vývod), dokáže obsluhovať pamäť.

### Softvérový Bootloader

Bootloader načíta tabuľku partícií na preddefinovanom offsete (štandardne 0x8000). Nájde aplikačné partície (s podporou bootovania) a ak sa využíva tabuľka partícií s OTA (Over-The-Air) definíciami, existuje aj partícia OTA\_DATA, kde sa nachádza príznak Bootloader pre bootovanie konkrétnej partície, kde sa nachádza spustiteľný firmvér. Ak sa tabuľka partícií využíva bez OTA definícií s jedným firmvérom, OTA\_DATA partícia neexistuje, nakoľko existuje iba jedna bootovateľná partícia. Následuje zavedenie programu aplikácie Bootloaderom do RAM pamäte (IRAM (RAM pamäť inštrukcií) a DRAM (RAM pamäť dát)).

### Spustenie aplikácie

Pri spustení aplikácie (firmvéru) sa spustí APP\_CPU (doteraz držaný v reštarte). Po inicializácii hlavných základných komponentov sa vytvorí hlavná úloha a spustí sa plánovač FreeRTOS. Hlavná úloha je funkcia app\_main(), ktorú musí obsahovať každý program. Táto úloha má návratový typ (int) a pri návrate na 0 sa ukončí. V plánovači FreeRTOS je možné spúšťať samostatné funkcie ako

úlohy – procesy (tasky). Tasky sa štandardne vytvárajú priamo v hlavnej aplikácii, ktorá ich inicializuje.

## 1.2. Pamäť mikrokontroléru ESP32

Adresný priestor [6]:

- Symetrické mapovanie pamäte
- 4 GB (32-bit) adresný priestor
- 1296 kB adresný priestor embedded pamäte
- 19704 kB adresný priestor externej pamäte
- 512 kB adresný priestor periférie
- 328 kB DMA (Priamy prístup do pamäte) adresný priestor

Embedded (vstavaná) pamäť :

- 448 kB internej ROM pamäte (Pamäť len na čítanie, energeticky nezávislá)
- 520 kB internej SRAM (Static Random Access Memory) pamäte
- 8 kB RTC FAST pamäte
- 8 kB RTC SLOW pamäte
- \* - flash pamäť môže byť aj vstavaná pri istých verziách ESP32 (napr. ESP32-PICO-D4)

Externá pamäť:

- Podpora až 16 MB SPI flash pamäte
- Podpora až 8 MB SPI SRAM pamäte

### 1.2.1. Embedded (vstavaná) pamäť

Vstavaná pamäť mikrokontroléru ESP32 má štyri segmenty - ROM, SRAM, RTC (Hodiny reálneho času) FAST a SLOW pamäť, ktoré sú vnútorne logicky rozdelené. Pamäť ROM obsahuje funkcie jadra ESP32, spodné vrstvy konektivity, obsluhu ich stacku, systémové funkcie, stub a iné... Funkcia Stub môže spustiť ROM funkciu, spustiteľný kód uložený v RTC SLOW pamäti. Využíva sa napríklad v operáciách Deep Sleep (režim hlbokého spánku) hlavného procesora ESP32 pre výpis informácií na UART (Univerzálny asynchrónny prijímač a vysielateľ) rozhranie. Funkcia pre stub využíva v definícii funkcie v zdrojovom kóde makro `RTC_IRAM_ATTR`, aby funkciu stub dokázal obslúžiť.

ROM pamäť ďalej obsahuje rutiny, ktoré sa dokážu spúšťať priamo z ROM a nezaberajú miesto v RAM pamäti. Spustenie je rýchlejšie ako z externej flash pamäte. K ROM pamäti môžu pristupovať oba jadrá procesora Xtensa.

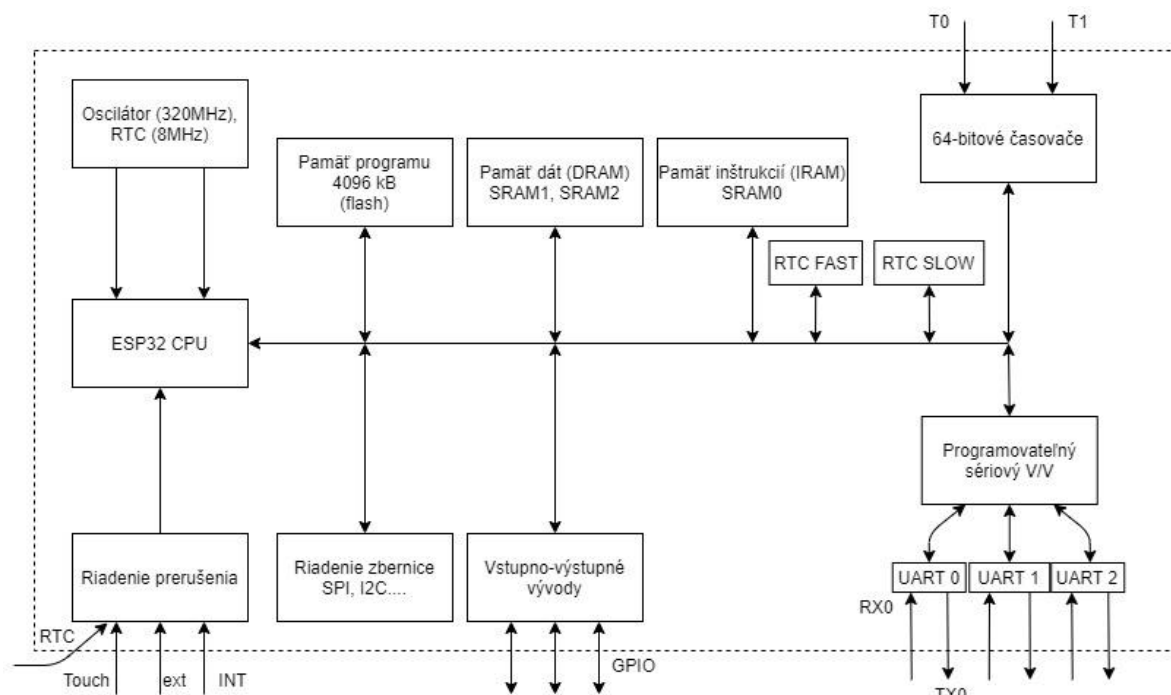
Je rozdelená na oddiely ROM 0 (384 kB) - mapovaný v pamäti inštrukcií v rozsahu 0x4000\_0000 až 0x4005\_FFFF. Zostávajúca časť - ROM 1 (64 kB) je mapovaná na dátovej zbernici pre rozsah 0x3FF9\_0000 až 0x3FF9\_FFFF (vyjadrené v hexadecimálnej hodnote). Pamäť SRAM je rozdelená na tri segmenty - SRAM0 (192 kB), SRAM1 (128 kB), SRAM2 (200 kB).

Z pamäte SRAM0 je možné vyhradiť časť 64 kB pre cache externej pamäte. Zostávajúca časť SRAM0 (128 kB) slúži na zápis a čítanie hlavným procesorom Xtensa mikrokontroléru ESP32, pristupovať a zapisovať do nej môžu obe jadrá. V prípade, že sa alokovaná časť 64 kB na cache nepoužíva, je ju možné používať ako štandardnú RAM pamäť a pristupovať nej cez akékoľvek jadro procesora, je mapovaná na zbernici inštrukcií v rozsahu 0x4007\_0000 až 0x4007\_FFFF. Pamäť SRAM0 a SRAM1 alokuje BSS, Heap segment pre program.

Zostávajúca časť SRAM0 je alokovaná tiež na zbernici inštrukcií v rozsahu 0x4008\_0000 až 0x4009\_FFFF. Pamäť SRAM1 má veľkosť 128 kB. K tejto pamäti je možné pristupovať cez inštrukčnú i dátovú zbernicu. Na dátovej má mapovanie 0x3FFE\_0000 až 0x3FFF\_FFFF a na inštrukčnej s 0x400A\_0000 až 0x400B\_FFFF. Pamäť SRAM2 má vyhradenú veľkosť 200 kB, je mapovaná na dátovej zbernici pre rozsah 0x3FFA\_E000 až 0x3FFD\_FFFF. Je k nej možné prístupíť cez akékoľvek jadro procesora. DMA používa rovnakú adresáciu ako procesor dátovej zbernice pre zápis a čítanie do SRAM1 (0x3FFE\_0000 až 0x3FFF\_FFFF) a SRAM2 (0x3FFA\_E000 až 0x3FFD\_FFFF).

DMA používa celkom trinásť zberníc: UART0, UART1, UART2, SPI1 (Synchrónne sériové periférne rozhranie), SPI2, SPI3, I2S0, I2S1, SDIO, Slave SDMMC, EMAC, Bluetooth, WiFi. RTC FAST je pamäť typu SRAM a má veľkosť 8 kB. Čítať a zapisovať do nej môže iba PRO\_CPU, procesor APP\_CPU nemá k pamäti prístup. Je adresovaná 0x3FF8\_0000 až 0x3FF8\_1FFF na dátovej zbernici, respektíve na 0x400C\_0000 až 0x400C\_1FFF na zbernici inštrukcií. RTC SLOW pamäť má rovnakú veľkosť ako RTC FAST.

Pamäť je mapovaná v rozsahu 0x5000\_0000 až 0x5000\_1FFF na dátovej i inštrukčnej zbernici zároveň. K tejto pamäti môžu však pristupovať oba procesory Xtensa. V prípade využívania ULP (Režim nízkeho odberu elektrickej energie) koprocessora je táto pamäť vyhradená pre neho a slúži na uloženie jeho programu, odkiaľ ho spúšťa. Mapa pamäte je blokovou schémou znázornená na obrázku Obr. 2.



Obr. 2 Mapa pamäte ESP32

### 1.2.2. Externá pamäť

K mikrokontroléru ESP32 je možné pripojiť externú pamäť typu flash a SRAM cez QSPI (Štandard pre štvornásobné zrýchlenie SPI zbernice) zbernicu. Procesor Xtensa mikrokontroléru ESP32 pristupuje k externej pamäti cez Cache a MMU (Jednotka správy pamäte). Na základe dostupného adresného priestoru je možné využiť maximálne 16 MB externej flash pamäte a 8 MB externej SRAM pamäte. Externá SRAM pamäť sa pripája paralelne k existujúcej externej flash pamäti. ESP32 podporuje viacero typov SRAM pamätí, avšak vývojársky framework ESP-IDF podporuje iba externá SRAM z vlastnej produkcie Espressif Systems s výrobným označením ESP-PSRAM32, ESP-PSRAM64.

Externú SRAM pamäť označujeme aj ako PSRAM (Pseudo Static RAM). Alokácia PSRAM pamäte začína od offsetu 0x3F800000 (v prípade 4 MB flash pamäte). PSRAM obsluhuje používateľská aplikácia, zodpovedá za správu pamäte, obsluhu vyrovnávacej pamäte. Externá flash pamäť je typom energeticky nezávislej pamäte. Dáta, ktoré sú do nej zapísané sú dostupné aj po odpojení a pripojení jej napájania. Pamäť typu flash je v prípade mikrokontroléru ESP32 obsiahnutá v samostatnom čipe, ktorý je s hlavným procesorom prepojený cez SPI zbernicu, ktorou sa dáta prenášajú.

Veľkosť flash pamäte, ktorá je na ESP32 osadená je najčastejšie 4 MB s možnosťou jej rozšírenia na 16 MB. Do flash pamäte je možné zapísať používateľský program (aj viacero, ak existujú dostupné partície s podporou bootovania, ich typ sa v tabuľke partícií označuje ako app), tabuľku partícií, ktorá

umožňuje logické rozdelenie flash pamäte na bootovateľné partície s firmvérom, oddiel Bootloadera (zavádzača) a SPIFFS filesystému.

Do flash pamäte je možné zapísať aj rôzne dáta a súbory (do SPIFFS filesystému), ku ktorým môže pristupovať mikrokontrolér v užívateľskej aplikácii. Najčastejšie sú to konfigurácie, obrázky, textové súbory, kaskádové štýly, súbory Javascriptu, ktoré je možné použiť vo vlastnej používateľskej aplikácii napríklad pri implementácii http (Hypertextový prenosový protokol) webservera bežiaceho na platforme ESP32, ktorá má všetky zdroje HTML (Hypertextový značkovací jazyk) stránky uložené vo vlastnej pamäti.

### 1.2.3. Partition Table

Mikrokontrolér ESP32 dokáže mať niekoľko na sebe nezávislých partícií uložených vo flash pamäti. Na preddefinovanom offsete vo flash pamäti - štandardne 0x8000 je zapísaná tabuľka partícií [7], ktorá rozdeľuje priestor vo flash pamäti na logické celky, kde je každá partícia popísaná svojím názvom, typom (rozdeľujeme systémové, bootovateľné, dátové partície a iné druhy), začiatočným offsetom a veľkosťou.. Tabuľka má pevnú veľkosť, do ktorej sa zmestí popis až 95 partícií.

Tabuľka partícií v rozhraní ESP-IDF využíva dva štandardné typy, ktoré sa najčastejšie používajú - "Single Factory App, no OTA" a "Factory App, two OTA definitions". Prvý menovaný typ je vhodný pre jeden firmvér, ktorý tvorí používateľská aplikácia. Pre metódy aktualizácie firmvéru s možnosťou jeho uloženia do inej dostupnej partície je vhodný druhý typ - "Factor App, two OTA definitions", ktorý umožňuje okrem hlavnej aplikácie firmvéru spúšťať aj firmvér uložený na dvoch bootovateľných OTA partíciách, celkovo tri firmvéry.

Každá bootovateľná (aplikačná) partícia má štandardne veľkosť 1MB pre program. Tento typ schémy partícií som využil aj v mojej diplomovej práci, nakoľko OTA aktualizáciu využívam. V prípade tohto typu s viacnásobnou bootovateľnou partíciou je do tabuľky pridaná aj partícia OTA\_DATA, ktorá popisuje príznak pre Bootloader (zavádzač), ktorý firmvér má prioritne bootovať.

Ak je OTA\_DATA prázdna, bootuje sa továrenská verzia - Factory App, čo je firmvér, ktorý je možné nahráť iba prostredníctvom USB-UART (Prevodník USB signálov na UART rozhranie) rozhrania. Okrem firmvérových partícií sú v tabuľke partícií zapísané aj iné partície dátového a konfiguračného významu. Štandardná tabuľka partícií je vizualizovaná na obrázku Obr. 3 s tromi aplikačnými partíciami.

```
# ESP-IDF Partition Table
# Name,   Type, SubType, Offset,   Size, Flags
nvs,      data, nvs,      0x9000, 0x4000,
otadata,  data, ota,      0xd000, 0x2000,
phy_init, data, phy,      0xf000, 0x1000,
factory,  app,  factory, 0x10000, 1M,
ota_0,    app,  ota_0,  0x11000, 1M,
ota_1,    app,  ota_1,  0x21000, 1M,
```

Obr. 3 Tabuľka partícií s tromi aplikačnými partíciami

Tabuľku partícií je možné plne prispôbiť pre potreby finálnej aplikácie. V prostredí ESP-IDF je možné tabuľku partícií importovať z .csv (formát dát) súboru. Každéj partícii je možné priradiť aj značku (Flag), ktorým je možné nastaviť partícii systémovú funkcionality, napríklad šifrovanie jej obsahu na danom offsete v prípade využitia funkcionality Flash Encryption. Bootovateľné partície (typ app) sú šifrované vždy bez nutnosti nastaviť značku (Flag), ak je povolený Flash Encryption

#### 1.2.4. eFuses

Mikrokontrolér ESP32 obsahuje niekoľko interných pamäťových blokov – eFuses (elektronická poistka – jednorázovo programovateľná pamäť) [8]. Tieto bloky sú jednorázovo programovateľnou pamäťou a majú veľkosť 256 bitov. Každý z blokov eFuses je rozdelený do ôsmich 32-bitových registrov. Každý register funguje aj ako 1-bitové pole, ktoré umožňuje jednorázový zápis do tejto pamäte. ESP32 má celkovo štyri eFuse bloky. Sú označené názvom Block, respektíve BLK (Označenie bloku jednorázovo programovateľnej pamäte eFuse) a indexom, ktorý ich charakterizuje 0 až 3.

Z pohľadu použitých eFuses v mikrokontroléri ESP32 ich môžeme rozdeliť na:

- Kalibračné (Kalibrácia napäťovej referencie, ADC – analógovo-digitálny prevodník)
- Konfiguračné (konfigurácia SDIO rozhrania, signálov)
- eFuse fuses („Poistkové“ eFuses – rezervácia BLK3, maska bitových polí eFuses)
- eFuses identity (MAC, Chip revision)
- Bezpečnostné (JTAG, Flash Encryption, Secure Boot, DEBUG)

Po zapísaní do systémových eFuse nie je možné jej obsah softvérovo prečítať, používa ich výhradne hardvérový podprogram, napríklad Secure Boot pri BLK2. Blok eFuse BLK0 má niekoľko polí do ktorých je možné zapísať hodnotu a majú aj systémový význam, napríklad potvrdzovacia poistka ABS\_DONE\_0 (1-bitové pole), ktorá permanentne spúšťa Secure Boot bez možnosti jeho vypnutia.. Dva bloky - eFuses BLK1 a BLK2 sú systémové poistky. Blok eFuse BLK1 obsahuje Flash Encryption šifrovací kľúč a nájde využitie pri šifrovaní niektorých partícií flash pamäte.

Pri eFuse BLK2 obsahuje tento blok Secure Boot šifrovací kľúč, ktorý zabezpečuje Bootloader a následne bootovací proces. Posledným eFuse pamäťovým blokom je BLK3, ktorým je možné nastaviť špecifickú MAC adresu mikrokontroléru ESP32, prípadne môže byť použitá aj na používateľskú aplikáciu.

### 1.3. ULP

Nízka spotreba je jednou z priorít IoT (Internet of Things) aplikácii [9]. Tieto aplikácie nazývame aj ULP (Ultra-Low Power), teda aplikácie s ultra-nízkou spotrebou elektrickej energie. Využitie týchto aplikácii je predovšetkým v implementáciách s prevádzkou na batériu. ESP32 umožňuje vypnúť hlavný procesor Xtensa, WiFi a Bluetooth modem v nečinnosti zariadenia. Dokáže tým výrazným spôsobom znížiť spotrebu elektrickej energie na minimum.

Tento jav nazývame aj režimom spánku mikrokontroléru. Pre opätovné zapnutie hlavného procesora, modemu WiFi / Bluetooth konektivity je nutné prebudiť hlavný procesor. Z pohľadu režimu spánku existuje viacero možností uspania určitých častí mikrokontroléru.

Tab. 1 Prevádzkové režimy mikrokontroléru ESP32 a stav periférií

Prevádzkový režim	Aktívny	Modem-sleep	Light-sleep	Deep-sleep
<b>Hardvér</b>	-	-	-	-
Xtensa CPU	ZAP	ZAP	PAUSED	VYP
WiFi/Bluetooth modem	ZAP	VYP	VYP	VYP
RTC pamäť a periférie	ZAP	ZAP	ZAP	ZAP
ULP koprocessor	ZAP	ZAP	ZAP	ZAP / VYP

Pre prebudenie čipu ESP32 v režime ľahkého spánku (Light Sleep) je možné využiť tieto druhy zdrojov:

- **UART** - prebudí čip ESP32, ak sa na RX (prijímači) objaví určitý počet signálov v logickej 1, ktoré indikujú príjem dát. Používa počítadlo v prerušení na vzostupnú (pozitívnu) hranu - tzv. RISING edge pri prechode z logickej 0 do logickej 1. Na základe rozhodovacieho prahu - počet signálov UART prebudí a môže prijímať dáta (existujúce dáta - signály, ktoré realizovali prebudenie nie sú uložené v buffri UART zbernice).
- **GPIO** - umožňuje využiť akýkoľvek GPIO vývod a definovať na ňom očakávanú logickú úroveň, ktorá prebudí hlavný čip Xtensa mikrokontroléru ESP32, ak sa na danom vývode objaví

Pre prebudenie čipu ESP32 v režime hlbokého spánku (Deep Sleep) je možné využiť až štyri druhy zdrojov:

- **Timer** - RTC kontroler má Timer (časovač), prostredníctvom ktorého dokáže zobudiť hlavný procesor mikrokontroleru ESP32 po určitom zadanom čase, kedy dôjde k pretečeniu časovača. Časovač je možné nastavovať prostredníctvom parametra v mikrosekundách
- **Dotyk** - využíva detekciu zmeny analógovej hodnoty na preddefinovanom vstupnom vývode s podporou TOUCH (snímania dotyku) funkcie. Dotykom sa vykoná kapacitná zmena, na ktorú dokáže mikrokontroler reagovať čítaním analógovej hodnoty vývodu a pri prekročení rozhodovacieho prahu prebudí hlavný procesor Xtensa mikrokontroleru ESP32
- **Externé prebudenie** - RTC IO (vstupno-výstupný) modul pracuje s logikou, ktorá dokáže zobudiť hlavný čip ESP32, ak je RTC GPIO nastavený na požadovanú logickú úroveň. Existuje aj rozšírenie tohto prebudenia o skupinu vývodov, kde sa skúma logická úroveň všetkých vývodov v tejto skupine a po nadobudnutí špecifickej úrovne sa hlavný čip ESP32 prebudí
- **ULP koprocesor** - spúšťa svoj program uložený v RTC SLOW pamäti v režime spánku hlavného procesora Xtensa. Dokáže obsluhovať ADC, vykonávať merania zo senzorov, dokáže pristupovať k RTC registrom a čítať / zapisovať do nich. Pri určitej udalosti - napríklad prekročení rozhodovacieho prahu ADC prevodníka dokáže hlavný čip ESP32 zobudiť

Koprocesor [10] využíva 32-bitové inštrukcie s 32-bitovou adresáciou v pamäti a má vlastné štyri 16-bitové registre do ktorých môže ukladať dáta. Majú označenie R0 až R3. Má vlastný 8-bitový čítačový register, ktorý môže byť použitý pre beh slučiek, pristupuje naň cez špeciálne inštrukcie, dokáže pristupovať aj k pamäti RTC SLOW. Programový kód, ktorý dokáže vykonávať musí byť zapísaný v jazyku Assembler a kompiluje sa spoločne s hlavnou - užívateľskou aplikáciou.

ALU (aritmeticko-logická jednotka) inštrukcie, obsluhu registrov RTC\_CNTL, RTC\_IO, SENS a inštrukcie pre prácu s pamäťou dokáže koprocesor obslúžiť v jednom cykle. Pre obsluhu externých zberníc (napríklad ADC) potrebuje 2 až 4 cykly na základe vykonávanej operácie.

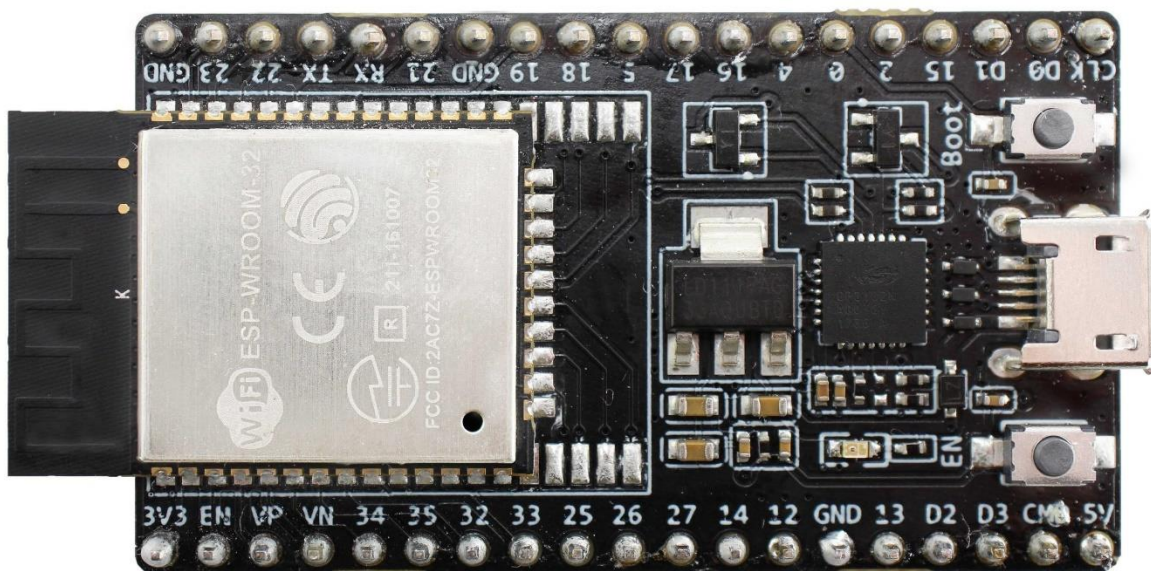
#### 1.4. Vývojové kity, samostatný čip ESP32

Pre vývoj užívateľských projektov [11] je možné využiť samostatný čip ESP32 (Standalone), ktorý je možné programovať cez externý USB-UART prevodník, napríklad FTDI (Future Technology Devices International Limited) s vyvedenými vývodmi pre pripojenie k mikrokontroleru ESP32. Taktiež existuje aj vývojový kit (nazývaný aj Devkit), ktorý je pri vývoji projektov viac obľúbený. Vývojový kit je osadený okrem čipu ESP32 aj USB-UART prevodníkom - najčastejšie CP2102 od Silicon Labs pre možnosť napájania a programovania ESP32 cez USB (Univerzálna sériová zbernica) kábel. Najpopulárnejší DevKitC od Espressifu je možné nájsť na obrázku Obr. 4.



Devkit je osadený aj indikačnými LED (luminiscenčné svetlo) diódami a má vyvedené vývody na ktoré je možné prispájkovať vývodové lišty, najčastejšie je to 30 / 38 - vývodový Devkit. To umožní lepšiu manipuláciu s mikrokontrolérom a ďalšie pripojenie periférií a aj vloženie dosky do Breadboardu (prepojovacieho poľa) pre vývoj projektu v testovacom - kompaktnom zapojení. Devkity majú vyvedené aj tlačidlá BOOT a EN. Stlačením tlačidla BOOT je možné nastaviť programovací mód čipu ESP32 (DTR signál), ktorý umožní nahrávanie programu.

Stlačením EN tlačidla je možné systém reštartovať (RTS signál). Kombináciou stlačení oboch tlačidiel súčasne je možné dosku prepnúť do módu sťahovania, kedy je možné stiahnuť celý obsah flash pamäte. Vývojové kity sú najčastejšie osadené najlacnejším ESP32 čipom - ESP32-WROOM-32 so 4 MB externou flash pamäťou bez osadenej PSRAM.



Obr. 4 ESP32-DevKitC

V kooperácii s Espressif Systems vyrábajú čipy ESP32 aj iní výrobcovia (Vendors), ktorí využívajú vlastné označenia svojich ESP32 čipov. Jedným z výrobcov s najvyššou produkciou ročne je firma AI-Thinker Čip ESP32-WROOM-32 z ich produkcie je označený ako ESP32-S [12] a je 100% ekvivalent čipu od Espressifu. Čip je doplnený o u.FL konektor, ktorý umožňuje k ESP32 pripojiť externú anténu pre zvýšenie dosahu pre Bluetooth, WiFi konektivitu.

Čipom ESP32-S sú osadené rôzne vývojové kity napríklad ESP32-S Dev Board, aj špeciálne dosky osadené kamerou - ESP-CAM. Populárnym výrobcom kitov je aj bulharská firma OLIMEX, ktorá využíva čipy ESP32 od Espressif Systems a ponúka riešenia pre oblasť IoT, riadenia výkonových spotrebičov na vlastných DPS. Osádza hotové dosky s podporou Ethernetu, PoE (Power over Ethernet), prípadne modemom mobilnej siete, ktorý je možné popri WiFi konektivite ESP32 používať.

Devkity priamo z produkcie Espressif Systems sú:

- ESP32-DevKitC
- ESP-WROVER-KIT
- ESP32-PICO-KIT
- ESP32-Ethernet-Kit
- ESP32-DevKit-S(-R)
- ESP32-PICO-KIT-1
- ESP32-PICO-DevKitM-2
- ESP32-DevKitM-1

### 1.5. ESP-IDF

Framework pre vývoj IoT aplikácií [13] založený na jazyku C pre platformu ESP32. Obsahuje API (rozhranie pre programovanie aplikácií), zahŕňa knižnice a ukázkové zdrojové kódy pre základnú obsluhu zberníc, rozhraní a komponenty pre obsluhu konkrétnej aplikácie. Obsahuje toolchain pre kompiláciu programu s build systémom CMake a Ninja build aplikácie. Cieľom je ponúknuť základné implementácie pre vývojárov IoT aplikácií s Bluetooth a WiFi konektivitou, ktoré sú bezpečné, poskytujú možnosť riadenia spotreby elektrickej energie zariadenia, ktorá je častokrát kľúčovým parametrom úspešnej (kritickej) aplikácie prevádzkovanvej na batériu pre IoT aplikácie.

Súčasťou ESP-IDF aj jednoduché konzolové rozhranie, ktoré slúži pre výber cieľového projektu, jeho kompiláciu, možnosť využitia vstavaných nástrojov pre kompiláciu programu, prácu s flash pamäťou, pamäťou eFuses ESP32 a pre kryptografické operácie (idf.py, esptool.py, espsecure.py, espfuse.py). Do frameworku ESP-IDF sú implementované aj vývojárske nástroje a komponenty vďaka ktorým je možné konfigurovať špecifické požiadavky aplikácie, napríklad - využitie kryptografického akcelérátora, vyhradenie pamäte, spätná kompatibilita s predchádzajúcimi verziami ESP-IDF a iné...

Espressif Systems vyvíja framework ESP-IDF na Githube v repozitári projektu. Projekt je vyvíjaný v niekoľkých verziách zároveň v príslušných vetvách projektu - tzv. branches. Vývojár si tak môže nainštalovať konkrétnu verziu ESP-IDF, ktorú chce využívať vrátane aktuálne vyvíjanej / release (produkčnej - stable) verzie projektu. Každá z verzii ESP-IDF má samostatnú dokumentáciu na webových stránkach frameworku: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/>. V mojej diplomovej práci som využil release verziu ESP-IDF 4.0 a neskôr 4.2 [14], čo je posledná release verzia s najvyšším číslom verzie frameworku zo 7. Decembra 2020.

V aktívnom vývoji je verzia 3.3.X, 4.1.X a verzia 4.3, ktorá je aktuálne v pre-release testovacej verzii. Každá z posledných release verzií má podporu po určitý čas pokiaľ bude dostávať aktualizácie a budú v nej vyriešené chyby vývojármi Espressif Systems a komunitou vývojárov. Dĺžka podpory pre verzie 4.X je 30 mesiacov. Verzia ESP-IDF 4.2 má podporu do júna 2023. Vývoj aktuálnych verzií frameworku ESP-IDF a nasledujúcich sa orientuje aj na podporu nových platforiem ESP32-C3, ktoré sú najnovším prírastkom do rodiny procesorov ESP32 a má novšie funkcie, ktoré nájdu využitie predovšetkým v bezpečných IoT aplikáciách.

Hlavnou výhodou frameworku ESP-IDF je možnosť programovania bližšie k fyzickej vrstve zariadenia, možnosť vytvárania komplexnejších programov, nakoľko dokáže využívať operačný systém reálneho času - FreeRTOS [15]. Základnou ideou operačného systému reálneho času je možnosť spustiť samostatné podprogramy ako úlohy - tasky, ktoré sa môžu nezávisle na sebe vykonávať na dvojjadrovom procesore Xtensa. Každý úlohe je možné priradiť veľkosť pamäte - stacku, prioritu a aj priradenie konkrétnemu jadru, ak je to potrebné. Tasky môžu medzi sebou vzájomne komunikovať.

Tento typ komunikácie sa označuje aj ako inter-task komunikácia. Jednotlivé úlohy si môžu medzi sebou vymieňať dáta a podmieňovať tak spúšťanie určitej úlohy, ktorá čaká na dáta z iného tasku, napríklad task pre odoslanie dát čaká na nameranie dát zo senzora. Existujú rôzne druhy blokovania úlohy, ktorá čaká na dáta, napríklad front (Queue), semafor (Semaphore). Blokovanie úlohy je možné používať aj v operáciách, kedy viacero taskov pristupuje k rovnakému médiu, ktoré nie je schopné obslúžiť všetky tasky súčasne - napríklad pri zápise do pamäte na konkrétny offset.

Úloha, ktorá môže pristupovať k médiu má takzvaný MUTEX (zámok), ktorý úlohu oprávňuje k prístupu k médiu. Po ukončení operácie daná úloha MUTEX ukončí a dovolí ho použiť inej úlohe, ktorá má záujem pristúpiť k médiu. Je to riadiaci mechanizmus prístupu k pamäti, médiu.

## 1.6. Arduino Core

Implementácia podpory mikrokontrolérov ESP32 do prostredia Arduino IDE. Obsahuje nástroj esptool [16] pre nahrávanie programu, kompilátor, sadu knižníc, ktoré sú rozhraním nad ESP-IDF. V paradigme jazyka Arduino (Wiring) - zjednodušený jazyk C, je možné spúšťať jednoduchými funkciami zložitejšie podprogramy na implementáciu v ESP-IDF. Vývoj Arduino Core podporuje a zastrešuje v súčasnosti Espressif Systems. Celé rozhranie Arduino Core pracuje bližšie k aplikačnej vrstve mikrokontrolérov.

Arduino Core [17] je vhodné pre používateľov a vývojárov s nižšou znalosťou programovania, umožňuje im vytvárať jednoduché programy aj pre IoT aplikácie aj bez rozsiahlejších znalostí. Pre

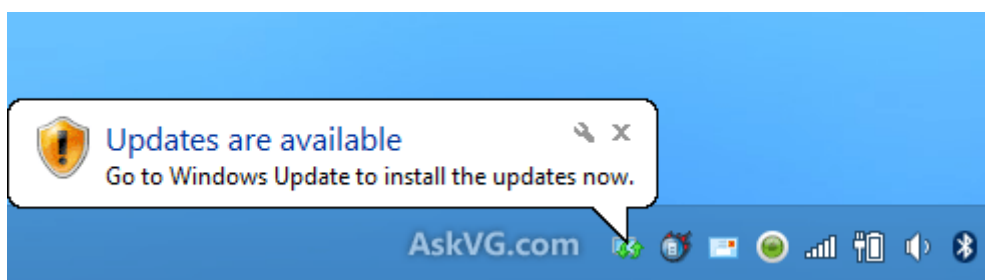
náročnejších užívateľov je možné spúšťať aj funkcie frameworku ESP-IDF a jeho komponenty. Neumožňuje však nastavenie špecifickej konfigurácie, alebo využitie vývojárskych nástrojov ako vo frameworku ESP-IDF. Jeho funkcie sú značne obmedzené.

Použitie Arduino Core sa viaže skôr na aplikačnú vrstvu vyvíjanej aplikácie, nepovoľuje pracovať bližšie k fyzickej vrstve mikrokontroléru ESP32. Umožňuje pracovať aj s FreeRTOS a využívať tasky podobne ako vo frameworku ESP-IDF. Arduino Core je stále vo vývoji a ukázkové programové implementácie (príklady) sa vytvárajú dodatočne podľa podporovaných príkladov vo frameworku ESP-IDF, ktorého API využíva.

## 2. OTA

OTA je termín popisujúci metódy distribúcie aktualizácii softvéru, konfigurácii a firmvéru pre zariadenia [18]. V praxi sa stretávame predovšetkým s takzvanými remote OTA aktualizáciami, ktoré sú distribuované prostredníctvom centrálného bodu, najčastejšie servera, ktorý je pre všetky zariadenia dosiahnuteľný cez internet. Tento typ aktualizácie poznáme z operačných systémov Windows, Android, iOS.

Operačný systém dokáže používateľa na novú aktualizáciu upozorniť vyskakovacím oknom (upozornenie na dostupnú systémovú aktualizáciu operačného systému Windows na obrázku Obr. 5), notifikáciou, prípadne dokážu aktualizáciu stiahnuť a nainštalovať bez nutnosti akcie používateľa, ktoré používateľa informujú o úspešnej aktualizácii softvéru, programom notifikáciou, informačným upozornením. Aktualizácie môžu byť systémové, programové s rôznou prioritou. Programové aktualizácie sa viažu ku konkrétnemu programu / softvéru, ktorý sa v operačnom systéme používa ako aplikácia.



Obr. 5 Upozornenie operačného systému Windows na novú aktualizáciu

Aktualizácie poskytujú ochranu pred bezpečnostnými hrozbami, ktoré sa prirodzene vyvíjajú s nárastom výpočtového výkonu a možností počítačov a zariadení. ponúkajú nové funkcionality operačných systémov a programov, opravujú chyby. Pre Remote OTA aktualizácie sú kladené isté špecifiká, nakoľko musí byť prenos dát realizovaný zabezpečeným kanálom a zariadenie musí dôverovať serveru, ktorý aktualizáciu distribuuje a aj samotnému obsahu aktualizácie - integrite dát.

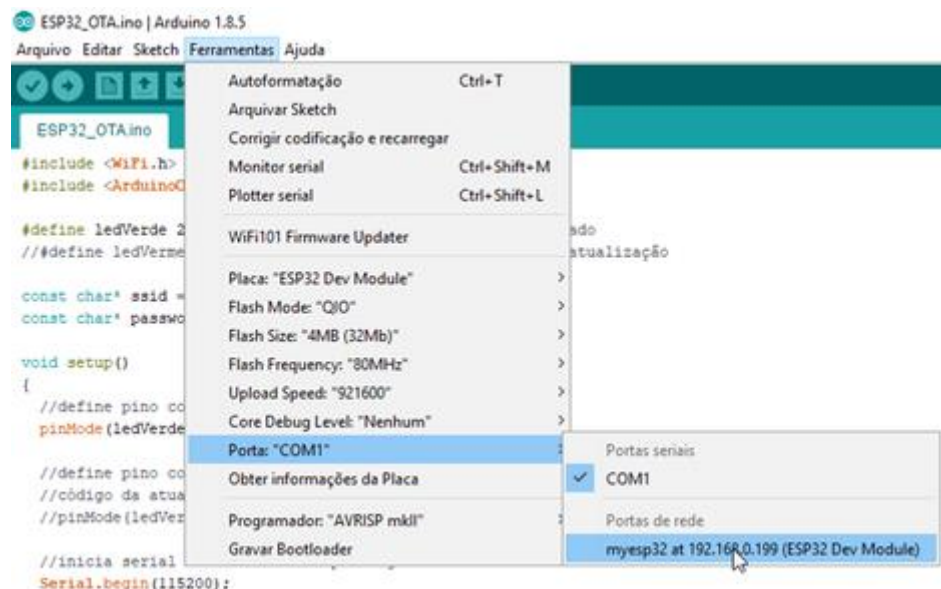
OTA aktualizácie našli uplatnenie aj pre zariadenia IoT (Internet vecí), ktoré je možné vzdialene aktualizovať cez internet, alebo z LAN siete na základe podporovanej OTA metódy bez nutnosti fyzického prístupu k zariadeniu [19]. Medzi IoT platformy podporujúce OTA aktualizácie sa radí aj mikrokontrolér ESP32, ktorý som využil vo svojej diplomovej práci. Ako som spomenul v predchádzajúcej kapitole, mikrokontrolér je možné programovať v rôznych programovacích jazykoch.

Líšia sa v programových implementáciách pre OTA aktualizácie a spôsobe samotnej aktualizácie rôznymi metódami. Pre overenie a otestovanie možností OTA aktualizácii v programových implementáciách som využil Arduino Core (Wiring) v prostredí Arduino IDE a ESP-IDF framework. Oba programovacie jazyky používajú špecifickú tabuľku partícií, ktorá definuje logické rozdelenie flash pamäte na viacero bootovateľných partícií.

## 2.1. OTA v Arduino Core

Príklady implementácii v jazyku Wiring obsahujú možnosť aktualizovať firmvér cez LAN sieť. V jednom z prípadov je možné využiť OTA sieťový port, ktorý je vysielaný v sieti a umožňuje cez neho nahráť firmvér. Programová implementácia pre tento typ aktualizácie - Basic OTA [20] vyžaduje, aby používateľ, ktorý firmvér do zariadenia aktualizuje bol v rovnakej sieti, ako ESP32. Údaje o WiFi sieti (SSID, heslo) sú nastavené v zdrojovom kóde a sú súčasťou príkladu Basic OTA.

Aby bolo možné vôbec aktualizácie vykonávať, prvýkrát musí byť OTA program nahratý do ESP32 fyzicky prostredníctvom USB-UART rozhrania cez USB kábel, alebo cez programátor, ak využívame samostatný čip ESP32 (nie Devkit) bez USB-UART prevodníka CP2102. Aby bolo možné následne nahráť aktualizovaný program do ESP32 cez OTA metódu, vyžaduje sa, aby mal používateľ nainštalovaný jazyk Python minimálne vo verzii 3, ktorý riadi celý proces aktualizácie (nahrávania nového programu cez sieť) do ESP32 automatizovane. Na obrázku Obr. 6 v prostredí Arduino IDE je dostupný sieťový OTA port medzi fyzickými COM portami.



Obr. 6 Sieťový OTA port v prostredí Arduino IDE

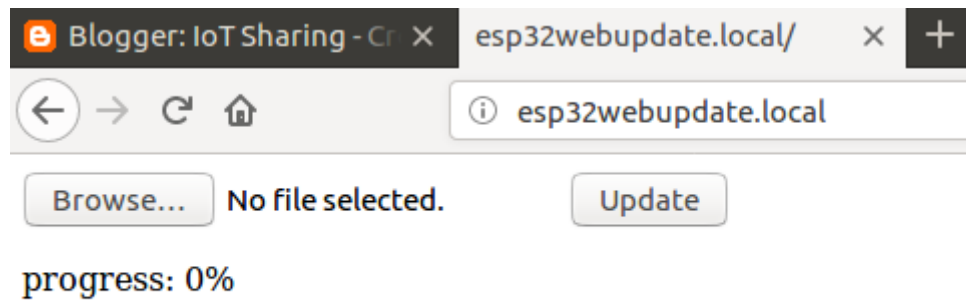
Po prvotnom nahratí OTA príkladu sa v Arduino IDE v časti Nástroje → Porty zobrazí okrem fyzických COM portov aj sieťový OTA port, ktorý je možné vybrať a nahrávať prostredníctvom neho nový

firmvér do mikrokontrolér ESP32. Sieťový port má identifikátor v tvare “esp32-MAC\_ADRESA at IP\_ADRESA” v LAN sieti, prípadne je možné definovať Hostname prostredníctvom mDNS (Multicast DNS) služby, ktorú je možné taktiež na ESP32 spustiť a sieťový port sa zobrazí v tvare: “HOSTNAME at IP\_ADRESA”. Nový program je možné po úspešnej aktualizácii nahráť priamo z vývojového prostredia Arduino IDE. Je nutné dodať, že nový program musí taktiež obsahovať OTA časť predchádzajúceho programu, ak chce používateľ do budúcnosti program opäť aktualizovať.

V prípade nahratia nového firmvéru cez OTA bez OTA implementácie sa nový firmvér nahraje, spustí bez možnosti ďalšej aktualizácie, vysielanie sieťového OTA portu sa ukončí. Výhodou je ľahká aktualizácia firmvéru priamo z prostredia Arduino IDE s možnosťou úprav zdrojového kódu a automatizovaný zápis aktualizácie do mikrokontroléru. Nevýhodou tejto možnosti OTA aktualizácie je, že ESP32 vždy prijme nový program prostredníctvom OTA aktualizácie aj v prípade nahratia duplicitného (rovnakého) programu.

Programová implementácia Basic OTA umožňuje spustiť aj primitívne funkcionality pre overenie integrity firmvéru a zvolenie špecifického OTA portu pre nahrávanie firmvéru. Integritu firmvéru je možné garantovať prostredníctvom známeho hesla v plaintexte, alebo jeho hashu v md5 formáte, ktoré je obsahom firmvéru nastavené v zdrojovom kóde. Hashovacia funkcia md5 je v súčasnej dobe zastaralá a neposkytuje takmer žiadnu úroveň ochrany, nakoľko má hashovacia funkcia md5 vysokú mieru kolízií, respektíve sú známe databázy hesiel uložených v md5 formáte, čo by znamenalo prelomenie hesla v rádoch milisekúnd.

Druhým spôsobom možnosti aktualizovať softvér v implementáciách OTA pre Arduino Core je využitie OTA Web Updater príkladu [21]. Tento typ OTA aktualizácie využíva ESP32 spustené v režime HTTP webservera, na ktorý je možné prísť prostredníctvom prehliadača. Pristupovať je možné na IP adresu ESP32 v LAN sieti, alebo cez doménové meno, ktoré je možné nastaviť cez mDNS službu. Tu však platí, že v prípade HTTP portu je adresa webstránky v tvare doménové\_meno.local (post-fix .local je nutný pre použitie doménového mena v LAN sieti) pre jej správne načítanie v prehliadači. Webové rozhranie OTA Web Updater na obrázku Obr. 7 poukazuje aj na možnosťou využitia mDNS záznamu s doménovým menom espwebupdate s postfixom local.



Obr. 7 OTA Web Updater

Súčasťou webového rozhrania je HTML formulár, prostredníctvom ktorého je možné nahráť binárny (skompilovaný) súbor nového programu a tým aktualizovať softvér. Binárny súbor je možné vygenerovať v Arduino IDE a uložiť ho pre jeho neskoršie nahranie prostredníctvom HTML formuláru. Táto forma aktualizácie je však viac zdĺhavá v porovnaní s Basic OTA. Aj tu platí, že pre budúcu aktualizáciu musí nový súbor okrem novej časti programu obsahovať aj OTA Web Updater.

Prvý krát sa program nahráva prostredníctvom USB kábla / programátora. Formulár prijíma akýkoľvek binárny súbor, aj duplicitný, žiadnym spôsobom neoveruje integritu firmvéru, ani jeho pôvod. Pre vyššiu bezpečnosť je možné využiť login formulár nad pôvodným nahrávacím formulárom, alebo možnosť využitia HTTP autentizácie menom, heslom.

V tomto prípade je možné firmvér aktualizovať aj z inej siete ak sa ESP32 nachádza a je dosiahnuteľná na verejnej IP adrese, avšak nezabezpečený HTTP protokol nie je vhodný pre aktualizáciu firmvéru. Firmvér môže byť kýmkoľvek podvrhnutý, pozmenený. Mikrokontrolér po prevzatí firmvéru neoveruje jeho integritu. Potenciálne nebezpečnú aktualizáciu - firmvér tak spustí.

## 2.2. OTA v ESP-IDF

Framework ESP-IDF obsahuje pokročilejšie implementácie OTA aktualizácií. Nájdeme tu iba remote OTA aktualizácie, ktoré sa realizujú spôsobom prebratia aktualizácie z externého zdroja - cloudu, webservera, ktorý firmvér distribuuje na preddefinovanom sieťovom umiestnení. Pripojenie je možné realizovať cez technológiu WiFi, alebo PHY Ethernet (radiaci čip LAN7820 / IP101GRI), ktorý je možné pripojiť k ESP32 cez RMII rozhranie.

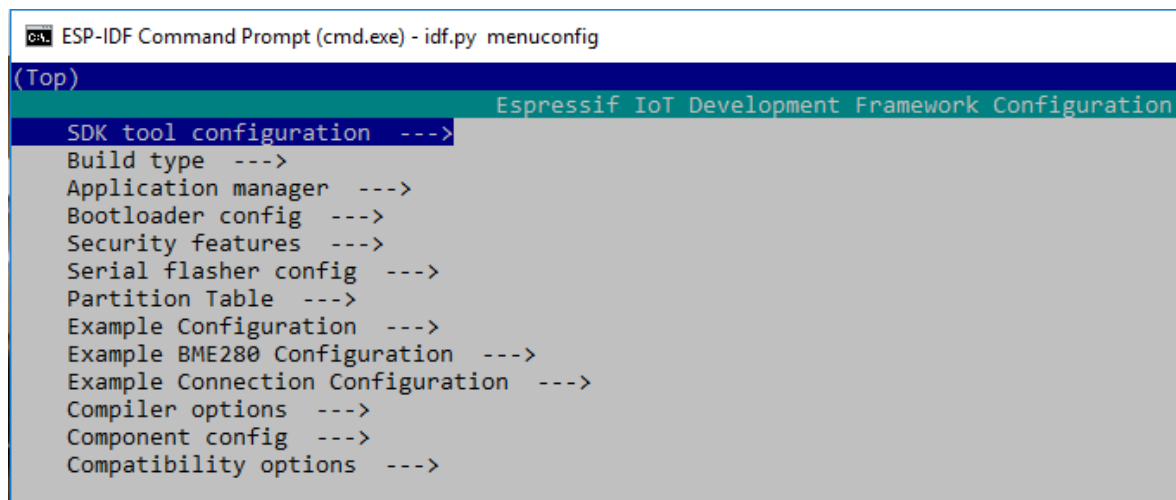
Pre overenie možností aktualizácií som využil technológiu WiFi a príklady Simple OTA a Native OTA, ktoré sú v ESP-IDF dostupné. Oba príklady využívajú zabezpečené HTTPS (Zabezpečený hypertextový prenosový protokol) spojenie na vzdialený webserver, na ktorý vykonávajú GET požiadavku pre stiahnutie firmvéru. ESP32 má kryptografický hardvérový akcelerátor AES



(Advanced Encryption Standard), ktorý využíva pri realizácii zabezpečeného spojenia, čím výrazne redukuje potrebný čas pre vytvorenie zabezpečeného spojenia. Pre realizáciu zabezpečeného spojenia cez HTTPS sa vyžaduje certifikát certifikačnej autority, ktorá vydala certifikát pre webserver.

Na rovnakom princípe dnes fungujú aj webové prehliadače, ktoré dôverujú vydavateľovi – certifikačnej autorite, ktorá vydala certifikát pre webserver (doménu) a podpísala ho svojim súkromným kľúčom. Pri realizácii mojej bakalárskej práce pred dvoma rokmi som túto problematiku riešil a certifikát pre webserver som generoval vlastnou certifikačnou autoritou. Certifikačná autorita je stále platná a rovnako tak i certifikát webservera, nakoľko bol generovaný na obdobie piatich rokov - do roku 2023. Certifikát certifikačnej autority v .pem formáte som do oboch príkladov vložil do predpripraveného - prázdneho súboru ca\_cert.pem v priečinku server\_certs.

Certifikát je vložený do aplikácie v procese kompilácie. Konfigurácia oboch základných príkladov sa realizuje cez konfiguračné menu - takzvaný Menuconfig, ktorý je možné vyvolať priamo z konzolovej aplikácie ESP-IDF. Na obrázku Obr. 8 je Menuconfig otvorený v projekte Native OTA frameworku ESP-IDF.



Obr. 8 Menuconfig – konfiguračné menu projektov v ESP-IDF

Menuconfig ponúka štandardné menu, ktoré slúžia pre nastavenie mikrokontroléru, UART rozhrania, jeho rýchlosti, nastavenie módov, zapnutie doplnkového hardvéru (akcelerátory, podpora so staršími verziami ESP-IDF a iné). Položky menu je možné rozšíriť definovaním vlastného menu, ktoré môže slúžiť pre konfiguráciu daného príkladu. Ak ide o príklad v prostredí ESP-IDF,

ktorý využíva WiFi konektivitu, má štandardne samostatné menu pre zadanie mena a hesla WiFi siete na ktorú sa dokáže pripojiť. Tieto údaje sa vložia do programu v procese kompilácie.

Príklady Simple aj Native OTA ponúkajú vytvorené menu vo svojich zložkách projektov s názvom "Example Connection Configuration" a "Example Configuration" zadať SSID a heslo WiFi siete štandardu WPA / WPA2 - PSK a taktiež aj adresu domény (prípadne IP adresu) webservera a absolútnu cestu k firmvéru odkiaľ jeho možné prevziať. Následne je možné program v konzolovej aplikácii frameworku ESP-IDF skompilovať a nahráť do mikrokontroléru. Identickú kópiu skompilovaného firmvéru som umiestnil cez FTP (File Transfer Protocol) klienta na vlastnú doménu a umiestnenie <https://esp32.sk/firmware.bin>, kde je firmvér zapísaný a odkiaľ je ho schopný mikrokontrolér ESP32 prebrať. Danú cestu som mu nastavil aj v konfiguračnom menu "Example Configuration".

Príklady Simple OTA i Native OTA fungujú po stránke pripojenia a stiahnutia firmvéru identicky. V oboch prípadoch sa vykoná HTTPS GET požiadavka na webserver a jeho sieťové umiestnenie <https://esp32.sk/firmware.bin>. Firmvér je stiahnutý a uložený do partície OTA\_1 alebo OTA\_2 (do tej, ktorej firmvér aktuálne nie je bežiaci a používaný). Príklad Simple OTA [22] po zápise do partície a zmene OTA\_DATA príznaku pre bootovanie čaká v nekonečnej slučke. V prípade reštartu mikrokontroléru ESP32 cez tlačidlo EN (RESET), bootuje nový - stiahnutý firmvér a opätovne stiahne aj identický firmvér.

Programová implementácia príkladu Native OTA [23] je obsahovo robustnejšia a obsahuje rôzne mechanizmy pracujúce s aktualizáciou, ktoré rozhodujú o jej bootovaní a reštarte mikrokontroléru. Po zápise firmvéru do partície OTA\_1 alebo OTA\_2 sa overuje verzia aktuálne bežiaceho a stiahnutého firmvéru. V prípade, že sú verzie rôzne, vykoná sa zápis do OTA\_DATA partície, ktorá definuje príznak pre Bootloader, ktorý volí firmvér, ktorý je bootovaný. Nastáva reštart a ESP32 nabootuje nový firmvér.

Vždy po reštarte mikrokontroléru je stiahnutý nový firmvér z preddefinovaného sieťového umiestnenia, pričom sa porovnáva jeho verzia s aktuálne bežiacim firmvérom. Verzia je definovaná v textovom súbore príkladu Native OTA s názvom version.txt, kde je možné číselne aktuálnu verziu vyjadriť, tá je vložená do firmvéru v procese kompilácie. V prípade, že sú verzie identické, mikrokontrolér čaká v nekonečnej slučke s inkrementovacím počítadlom a výzvou na reštart mikrokontroléru cez EN (RESET) tlačidlo každú sekundu.

Tento typ príkladu je šetrnejší k flash pamäti a obsahuje základné mechanizmy pre overenie verzie firmvéru. Nakoľko však ani jeden z príkladov nerieši integritu firmvéru a overenie vydavateľa firmvéru, musia byť tieto funkcionality implementované dodatočne. Framework ESP-IDF má

vstavané rôzne vývojárske nástroje, ktorými je možné spustiť zabezpečovacie mechanizmy firmvéru pre OTA aktualizácie, i pre firmvér nahrať do zariadenia prostredníctvom USB-UART rozhrania.

Hlavnou výhodou nástrojov je, že ich nie je potrebné do programu vkladať prostredníctvom fragmentov zdrojových kódov, ale sú dynamicky kompilované do výstupného firmvéru bez nutnosti akýchkoľvek zmien v používateľskej aplikácii napísanej v jazyku C. Integritu dát (firmvéru) je možné zabezpečiť digitálnym podpísaním firmvéru s možnosťou overenia na úrovni Bootloadera pri bootovaní, alebo pri samotnom prebratí aktualizácie. Zabezpečiť je možné aj samotný bootovací proces s overením Bootloadera.

### 2.3. Metóda digitálneho podpisu

Digitálny podpis je metóda používaná pre overenie integrity dát (firmvéru), čím garantuje, že neboli pozmenené treťou osobou (útočníkom). Overený firmvér je bezpečný pre spustenie na vývojovej platforme ESP32 [24]. Digitálny podpis vychádza z využitia asymetrickej kryptografie, ktorá používa dvojicu kľúčov - súkromný a verejný. Súkromný kľúč je použitý pre podpísanie firmvéru, ktorý je distribuovaný cez OTA službu - server. Verejný kľúč je použitý pre overenie digitálneho podpisu, vkladá sa do softvérového Bootloadera, ktorý realizuje toto overenie pri bootovaní firmvéru, alebo už v procese samotnej aktualizácie firmvéru a jeho zápis na dostupnú OTA partíciu.

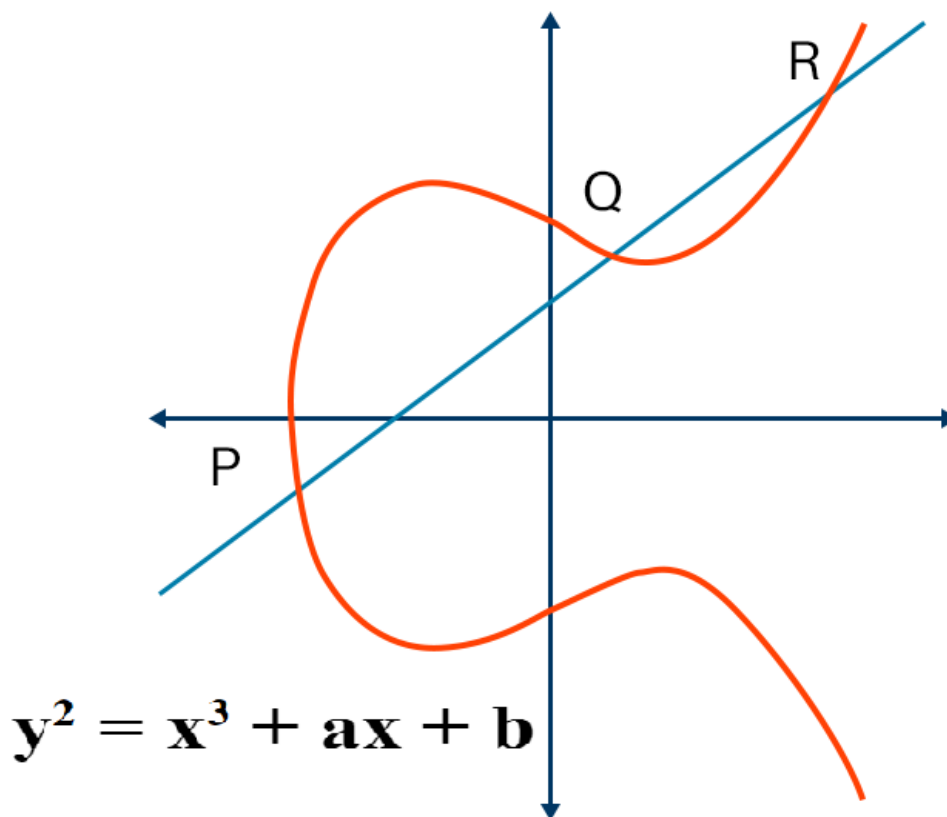
Najčastejšie sa pre generovanie dvojice kľúčov využíva algoritmus RSA (Rivest–Shamir–Adleman), alebo pre vstavané - embedded systémy s obmedzeným výpočtovým a pamäťovým vybavením sa využíva kryptografia na báze ECC (eliptických kriviek), ktorá umožňuje využiť porovnateľne kratší kľúč s rovnakou, alebo vyššou kryptografickou bezpečnosťou. Spôsob generovania súkromného kľúča v ECC je v porovnaní s RSA, ktoré využíva násobenie veľkých prvočísel iný. Metóda ECC je založená na zložitosti hľadania logaritmu pre náhodný bod eliptickej krivky nad konečným poľom. Tento typ kryptografie využíva pre generovanie dvojice kľúčov aj framework ESP-IDF.

#### 2.3.1. Digitálny podpis – implementácia v prostredí ESP-IDF

Pre kryptografické operácie je možné využiť nástroj `espsecure.py` [25], ktorý je vstavaný do prostredia ESP-IDF. Je ho možné obsluhovať cez konzolovú aplikáciu frameworku. Príkazom `espsecure.py generate_signing_key private.pem` je možné vygenerovať súkromný kľúč eliptickou krivkou NIST256p (ekvivalentné označenie `prime256v1` v nástroji OpenSSL). Nástroj OpenSSL je možné taktiež využiť v procese generovania súkromného kľúča, ktorého dĺžka je 256 bitov.

Eliptická krivka NIST256p je definovaná (popísaná) nad konečným (Galoisovým) poľom  $GF(q) \approx GF(p^n)$ , kde  $p$  je prvočíslo a  $n \in \mathbb{N}$  (na obrázku Obr. 9 s grafickým znázornením) vzťahom:

$$E(G_p): y^2 = x^3 + ax + b \pmod{p}$$



Obr. 9 Grafická vizualizácia eliptickej krivky NIST256p

Verejný kľúč je možné kryptografickým nástrojom `espsecure.py` extrahovať zo súkromného kľúča príkazom `espsecure.py extract_public_key --keyfile private.pem public.bin`. Aby bolo možné metódu digitálneho podpisu pre aplikáciu založenú na ESP32 použiť, je nutné v prostredí ESP-IDF v konfiguračnom menu - Menuconfig v časti "Security Features" zapnúť možnosti "Require signed apps", "Verify via Bootloader on startup" sa zvolia zakliknutím, pričom je možné digitálny podpis overiť aj priamo v procese stiahnutia firmvéru na takzvaný event "ON\_UPDATE", ktorý je v predmetnom menu tiež dostupný a odporúčaný pre použitie.

Ďalším parametrom, ktorý je nutné nastaviť je relatívna cesta (vzhľadom na root priečinok príkladu) k verejnému kľúču - `public.bin`, ktorý sa v procese kompilácie automatizovane vloží do softvérového Bootloadera. Nakoľko sa však pridaním verejného kľúča zväčší veľkosť Bootloadera a ten zasahuje veľkosťou do začiatočného offsetu tabuľky partícií, je ju nutné cez Menuconfig v časti "Partition Table" zmeniť parameter - začiatočný offset z `0x8000` na `0x10000`. Po posunutí začiatočného offsetu

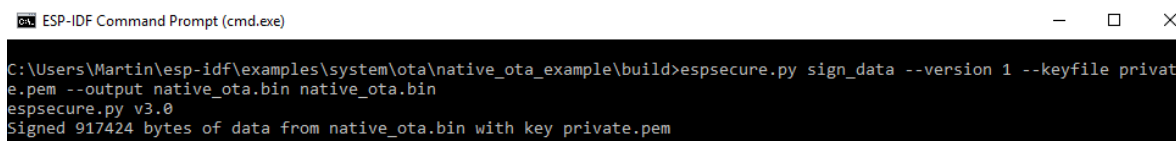
tabuľky partícií sa posunú aj všetky ostatné partície. V tabuľke Tab.2 je možné vidieť posun offsetu u všetkých aplikáčnych partícií.

Podpísanie firmvéru je nutné realizovať manuálne po kompilácii firmvéru pred nahrávaním do dosky cez USB-UART rozhranie, alebo pred vloženíu firmvéru na OTA webserver. Pre podpísanie firmvéru [26] je možné opätovne využiť vstavaný kryptografický nástroj `espsecure.py` a príkazom `espsecure.py sign_data --version 1 --keyfile private.pem --output native_ota.bin native_ota.bin` sa vykoná podpísanie firmvéru `native_ota.bin` a uloží ho do rovnakého súboru, pričom je možné využiť aj zápis podpísanej aplikácie do samostatného `.bin` súboru so zachovaním pôvodného - nepodpísaného firmvéru.

Tab. 2 Rozdiel offsetu aplikačných partícií po a pred posunutím

Názov partície	Typ	Podtyp	Offset	Pôvodný offset	Veľkosť
FACTORY_APP	APP	FACTORY	0x20000	0x10000	1MB
OTA_1	APP	OTA_1	0x120000	0x110000	1MB
OTA_2	APP	OTA_2	0x220000	0x210000	1MB

V procese podpisovania firmvéru sa z pôvodného súboru (`native_ota.bin`) vykoná odtlačok s využitím jednocestnej hashovacej funkcie SHA256. Odtlačok je zašifrovaný súkromným kľúčom a jeho výsledkom je digitálny podpis, ktorý je vložený do firmvéru za jeho pôvodný obsah. Podpis má dĺžku 68 B, pričom prvé 4 B sú slovo tvorené nulami a nasledujúcich 64 B je samotný digitálny podpis. Na Obr. 10 je dostupná vizualizácia podpisania firmvéru súkromným kľúčom v konzolovej aplikácii frameworku ESP-IDF s kryptografickým nástrojom `espsecure.py`.



```

ESP-IDF Command Prompt (cmd.exe)
C:\Users\Martin\esp-idf\examples\system\ota\native_ota_example\build>espsecure.py sign_data --version 1 --keyfile private.pem --output native_ota.bin native_ota.bin
espsecure.py v3.0
Signed 917424 bytes of data from native_ota.bin with key private.pem

```

Obr. 10 Podpísanie firmvéru súkromným kľúčom v konzolovej aplikácii ESP-IDF

Tento firmvér je možné nahráť do mikrokontroléru cez USB-UART rozhranie, alebo ho umiestniť na OTA webserver, ktorý bude túto aktualizáciu distribuovať klientom. Pri nahratí firmvéru cez USB-UART rozhranie je firmvér zapísaný do partície `FACTORY_APP` a je z nej aj štandardne bootovaný, ak nie je partíciou `OTA_DATA` s príznakom pre Bootloaderom nastavená iná partícia.

Firmvér je prenesený zabezpečeným prenosným kanálom ku prijímateľovi - využíva sa zabezpečený HTTPS protokol s end-to-end šifrovaním medzi serverom a klientom. Prijímateľ (Klient) po prevzatí firmvéru rozdelí stiahnutý súbor na časť firmvéru a digitálny podpis. Svojím súkromným kľúčom

dešifruje digitálny podpis a získa pôvodný hash súboru. Hashovacou funkciou SHA256 vytvorí odtlačok firmvéru a porovná oba hashe. V prípade, že sa zhodujú, firmvér je validný a nebol pozmenený treťou osobou.

Firmvér je dôveryhodný a je ho možné spustiť bezpečne na mikrokontroléri ESP32. Úspešnosť samotného overenia spustí následne funkciu z programu Native OTA, ktorá overuje, či je verzia bežiacieho firmvéru iná, ako verzia stiahnutého. V prípade, že je digitálny podpis pri stiahnutom firmvéri overený, API implementácie Native OTA pozmení OTA\_DATA príznak, ktorý definuje pre Bootloader miesto, odkiaľ chceme bootovať nový firmvér a vykoná softvérový reštart mikrokontroléru ESP32 pre spustenie procesu bootovania nového firmvéru.

Ak firmvér neobsahuje digitálny podpis, alebo nie je platný, alebo hash firmvéru (z dôvodu jeho pozmenenia treťou osobou) nie je zhodný z dešifrovaným hashom, Native OTA sa k overeniu verzie firmvéru nedostane a aktualizácia je ignorovaná. Tento typ overenia sa realizuje pri každom bootovaní firmvéru, ale aj pri prevzatí a uložení firmvéru do dostupnej OTA partície. Ak v procese bootovania nie je digitálny podpis firmvéru overený, Bootloader skúša spustiť firmvér z inej dostupnej partície - OTA / FACTORY\_APP. Ak sa nepodarí spustiť firmvér ani jednej z dostupných partícií, bootovací proces sa ukončí a mikrokontrolér nenabootuje firmvér.

## 2.4. Secure Boot v ESP-IDF

Metóda zabezpečenia Bootloadera a bootovacieho procesu [27]. Secure Boot kombinuje metódu digitálneho podpisu firmvéru a jeho overenia Bootloaderom pri bootovaní a pre event ON\_UPDATE. Zároveň je Secure Boot rozšírený o funkcionality overenia softvérového Bootloadera, ktorú obsluhuje hardvérový Bootloader (ROM Bootloader). V princípe ide o výpočet odtlačku (digestu) softvérového Bootloadera (zapísaného vo Flash pamäti) s využitím Secure Boot šifrovacieho kľúča, ktorý je uložený v jednorázovo programovateľnej pamäti eFuse BLK2.

Cieľom výpočtu odtlačku a následného overenia je kontrola, že je softvérový Bootloader uložený vo flash pamäti validný a zhoduje sa s odtlačkom, ktorý je ako referenčný uložený vo flash pamäti na preddefinovanom offsete 0x0. Zaručuje tak, že Bootloader uložený vo flash pamäti nebol pozmenený, čo by mohlo mať za následok spúšťanie neovereného firmvéru. Secure Boot je možné zapnúť cez Menuconfig v submenu "Security Features". Z pohľadu prevádzky Secure Bootu je možné zvoliť si jednu z jeho dvoch implementácií - "Re-flashable", ktorá je vhodná pre vývoj aplikácií a umožňuje nahrávať Secure Boot Signing key viackrát do určitej časti flash pamäte, tento režim nie je vhodný pre produkčné - finálne aplikácie.

Druhou implementáciou Secure Bootu je “One-Time Flash” a tento typ je vhodný pre produkčné aplikácie s jednorazovým zápisom Secure Boot šifrovacieho kľúča do eFuse BLK2 a permanentným zapnutím hardvérovej funkcie Secure Boot bez možnosti vypnutia. Tento typ som použil v mojej diplomovej práci pre implementáciu hardvérovej funkcionality Secure Boot. Zapnutím tejto funkcionality cez Menuconfig v časti “Security Features” však ešte nie je Secure Boot plne aktívny, nakoľko sa musí zapnúť zápisom šifrovacieho kľúča do eFuse BLK2 a zároveň je nutné zapísať potvrdzovací bit do 1-bitovej eFuse ABS\_DONE\_0.

#### 2.4.1. Secure Boot - implementácia v prostredí ESP-IDF

Šifrovací klíč pre Secure boot je nutné vygenerovať. Pre generovanie kľúča je možné využiť viackrát používaný kryptografický nástroj `espsecure.py` a príkazom `espsecure.py generate_signing_key secure-bootloader-key-256.bin` vygeneruje 256-bitový kľúč v binárnom formáte eliptickou krivkou NIST256p (prime256v1). Kľúč je potrebné zapísať do jednorázovo programovateľnej pamäte eFuse BLK2.

Pre prácu s eFuses existuje v ESP-IDF nástroj `espefuse.py` [28], ktorý sa označuje ako aj eFuse manager. Umožňuje prácu so všetkými eFuses, ktoré sú v ESP32 dostupné. Spomenutá jednorázovo programovateľná eFuse BLK2 je špeciálna systémová eFuse s veľkosťou 256-bitov do ktorej sa zapisuje šifrovací kľúč Secure Bootu. Na obrázku (Obr. 11) je sumár eFuses mikrokontroléru ESP32. V bloku BLK1 a BLK2 je zapísaný kľúč, ktorý nie je možné softvérovo prepísať / prečítať (nie je možnosť R/W pre dané bloky).

```
Security fuses:
FLASH_CRYPT_CNT (BLOCK0):      Flash encryption mode counter          = 0 R/W (0b0000000)
UART_DOWNLOAD_DIS (BLOCK0):    Disable UART download mode (ESP32 rev3 only) = False R/W (0b0)
FLASH_CRYPT_CONFIG (BLOCK0):   Flash encryption config (key tweak bits)     = 0 R/W (0x0)
CONSOLE_DEBUG_DISABLE (BLOCK0): Disable ROM BASIC interpreter fallback       = True R/W (0b1)
ABS_DONE_0 (BLOCK0):           Secure boot V1 is enabled for bootloader image = True R/W (0b1)
ABS_DONE_1 (BLOCK0):           Secure boot V2 is enabled for bootloader image = False R/W (0b0)
JTAG_DISABLE (BLOCK0):         Disable JTAG                                  = False R/W (0b0)
DISABLE_DL_ENCRYPT (BLOCK0):    Disable flash encryption in UART bootloader = False R/W (0b0)
DISABLE_DL_DECRYPT (BLOCK0):    Disable flash decryption in UART bootloader = False R/W (0b0)
DISABLE_DL_CACHE (BLOCK0):     Disable flash cache in UART bootloader    = False R/W (0b0)
BLOCK1 (BLOCK1):               Flash encryption key
= ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? -/-
BLOCK2 (BLOCK2):               Secure boot key
= ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? -/-
BLOCK3 (BLOCK3):               Variable Block 3
= 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 R/W

Flash voltage (VDD_SDIO) determined by GPIO12 on reset (High for 1.8V, Low/NC for 3.3V).

C:\Users\Martin\esp-idf\examples\system\ota\native ota example>
```

Obr. 11 Sumár eFuses zobrazených cez nástroj espefuse.py

Nástrojom `espefuse.py` som kľúč zapísal s využitím príkazu `espefuse.py burn_key secure_boot secure-bootloader-key-256.bin`. Pre zapnutie Secure Bootu, ktoré je permanentné je nutné zapísať aj potvrdzovací bit do eFuse `ABS_DONE_0`. Zápis som realizoval príkazom `espefuse.py burn_efuse ABS_DONE_0`. Od tohto momentu je eFuse `BLK2` chránená pred softvérovým zápisom a čítaním.

Potencionálny útočník tak nedokáže získať obsah - kľúč uložený v tejto eFuse, nakoľko k nej nemá prístup. K predmetnej eFuse BLK2 môže pristupovať už iba hardvér, respektíve hardvérová funkcionálna Secure Boot prostredníctvom hardvérového Bootloadera (First-stage Bootloader) uloženého v ROM, ktorá zabezpečuje overenie softvérového Bootloadera (Second-stage Bootloader) mikrokontroléru ESP32 zapísaného vo flash pamäti na preddefinovanom offsete 0x1000.

Aby bolo možné overiť softvérový Bootloader a zabezpečiť proces bootovania, je nutné vygenerovať odtlačok (digest), ktorý je výstupom algoritmu SBDA (Secure Bootloader Digest Algorithm) a zapísať ho do flash pamäte na offset 0x0, kde ho očakáva Secure Boot [29]. Hardvérová funkcionálna Secure Boot využíva SBDA k overeniu integrity softvérového Bootloadera s porovnaním odtlačku zapísaného na offsete 0x0. SBDA algoritmus načíta kľúč uložený v eFuse BLK2 - Secure Boot key v reverznej bitovej reprezentácii. Pred image bootloadera dosadí 128 bajtový vygenerovaný inicializačný vektor.

Algoritmus vykoná zarovnanie imageu Bootloadera modulo 128 a doplní 0xFF do reprezentácie reťazca modulo 128. Na každých 16 bajtov plaintextu imageu Bootloadera zašifruje s použitím šifrovacej funkcie AES256 v ECB móde. Výsledný šifrovaný text má reverznú bitovú reprezentáciu. Algoritmus vymení bajt každého 4-bajtového slova šifrovaného textu, vypočíta SHA-512 hash šifrovaného textu. Výstupom je 192 bajtový reťazec, ktorý je tvorený 128 bajtovým inicializačným vektorom a 64 bajtovým hashom funkcie SHA-512 zo šifrovaného textu.

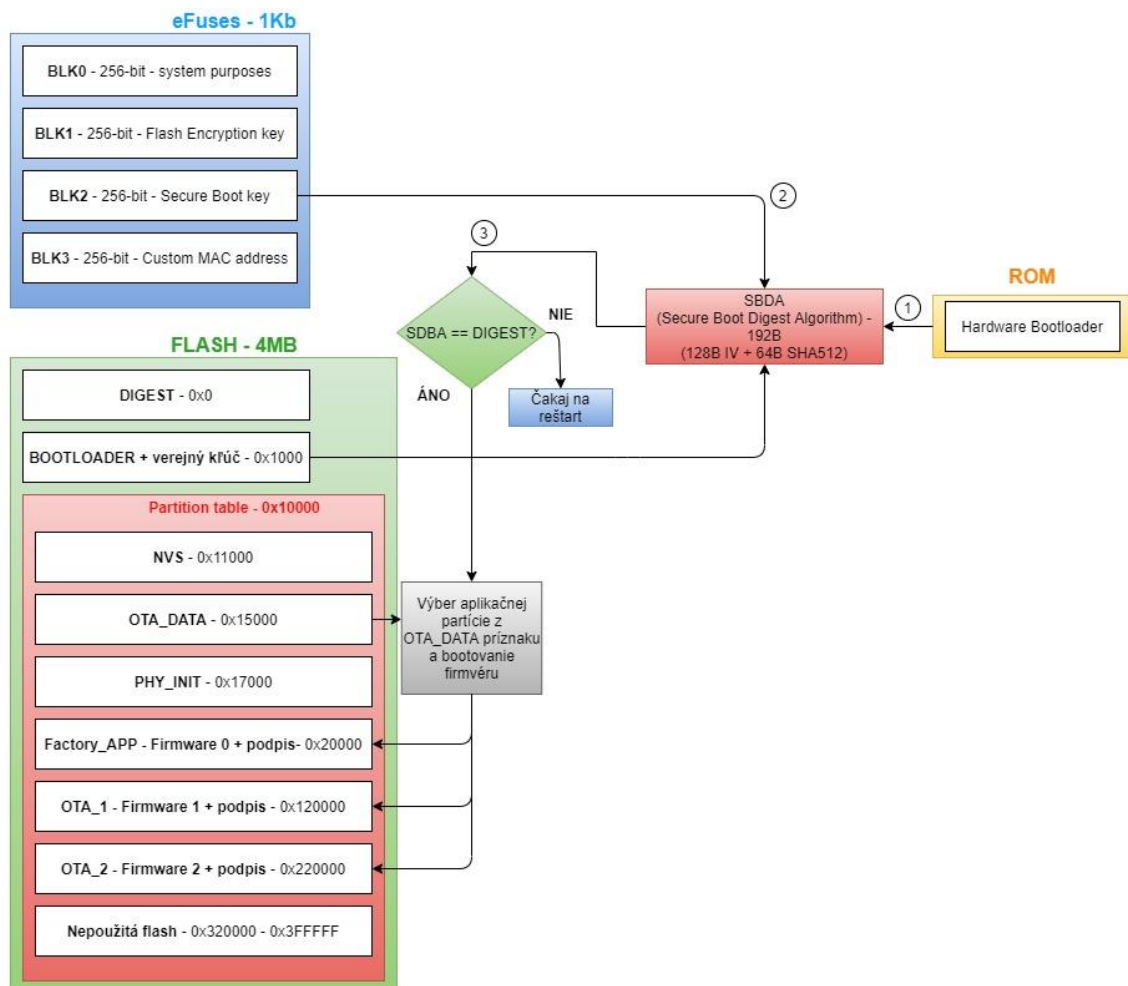
Odtlačok je možné algoritmom SBDA vygenerovať aj lokálne použitím nástroja `espsecure.py`. Ako šifrovací kľúč využijeme `secure-bootloader-key-256.bin`, ktorý sme do eFuse v predchádzajúcom kroku zapísali a máme ho stále k dispozícii. Príkazom `espsecure.py digest_secure_bootloader --keyfile secure-bootloader-key-256.bin --output ./bootloader-digest.bin build/bootloader/bootloader.bin` sa vygeneruje odtlačok, ktorý je možné zapísať do flash pamäte.

Príkaz má vstup - šifrovací Secure Boot kľúč v binárnom formáte (rovnaký bol zapísaný do eFuse BLK2) a image Bootloadera. Pri zápise súboru do flash pamäte je nutné zvoliť cieľový offset pre začiatok zápisu digestu na 0x0. Zápis som realizoval použitím príkazu: `esptool.py write_flash 0x0 bootloader-digest.bin`. Týmto u splnené všetky požiadavky pre schopnosť hardvérového Bootloadera overiť softvérový. Pri spustení mikrokontroléru sa vykoná hardvérové vypočítanie digestu, kedy hardvér pristúpi k vyčítaniu obsahu eFuse BLK2 a zo známeho offsetu, kde je zapísaný softvérový Bootloader sa prevezme jeho image.

Vykoná sa SBDA a výsledný digest porovná hardvérový Bootloader s digestom uloženým vo flash pamäti na offsete 0x0. V prípade, že sú oba digesty identické, hardvérový Bootloader umožní spustiť



softvérový Bootloader a ten pristúpi k bootovaniu firmvéru (s overením jeho digitálneho popisu...). Ak sú digesty rozdielne, bootovanie je zakázané hardvérovou funkcionalitou Secure Boot. Tento spôsob ochrany je efektívny v prípade, ak by útočník získal fyzický prístup ku mikrokontroléru a pokúsil by sa spustiť na mikrokontroléri svoj program, prepíše aj Bootloader a vypočítaný digest sa nebude zhodovať s digestom zapísaným na offsete 0x0 (za predpokladu, že nebude prepísaný) pri nahrávaní firmvéru. Na obrázku Obr. 12 je v blokovej schéme zakreslený priebeh Secure Bootu, ktorý nastáva ešte pred behom programu (bootovaním firmvéru z dostupnej partície).

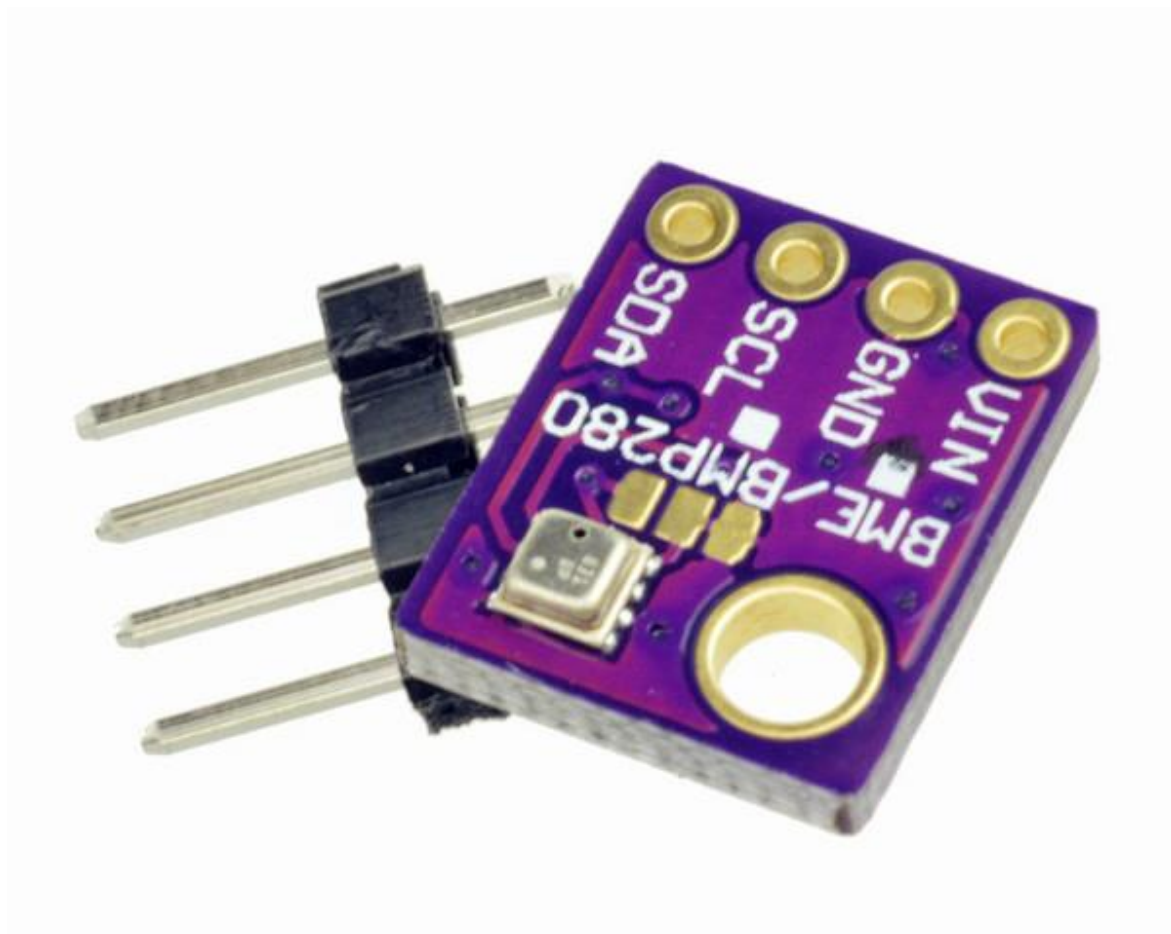


Obr. 12 Bloková schéma procesu Secure Boot po bootovanie firmvéru

### 3. Použitý hardvér a softvér

Pre implementáciu senzorového uzla som využil vývojovú dosku ESP32 Devkit V1, ktorá je osadená čipom ESP32-WROOM-32 a ponúka WiFi konektivitu. Doska má elektroniku pre zníženie napäťových úrovní - z napájacieho napätie 5V na 3,3V, čo je jeho operačná logika. Taktiež je vybavená USB-UART prevodníkom CP2102 [30], prostredníctvom ktorého je možné do flash pamäte nahrávať program a komunikovať so zariadením.

Pre návrh a realizáciu finálneho programu som využil framework ESP-IDF nakoľko je vhodnejší pre cieľovú aplikáciu a podporuje rôzne možnosti zabezpečenia firmvéru aj procesu samotnej aktualizácie. Pre ovládanie periférie - meteorologického senzora som využil implementáciu ovládača (Drivera) v jazyku C od Bosch Sensortec [31] pre BME280 [32] senzor (Obr. 13).



Obr. 13 Bosch BME280 - senzor teploty, tlaku, vlhkosti vzduchu

Pre senzorové merania som použil meteorologický senzor Bosch BME280 vo verzii s 3,3V operačnou logikou (existuje aj verzia senzora BME280 s 5V operačnou logikou) pre plnú kompatibilitu s logickými úrovňami mikrokontroléru ESP32. Senzor Bosch BME280 dokáže merať údaje o tlaku, teplote a vlhkosti vzduchu, komunikuje s mikrokontrolérom po I2C (Intel-Integrated-Circuit)

zbernici [33] so štandardnou rýchlosťou I2C zbernice s hodinovým signálom 100 kHz. Na základe známeho Bosch vzorca z dokumentácie BME a BMP (senzor tlaku a teploty vzduchu) senzorov je možné prepočítať absolútny tlak vzduchu na relatívny (na hladinu mora), pričom musíme do vzorca dosadiť aktuálnu nadmorskú výšku, prípadne ju môžeme vypočítať odhadom s priemernou hodnotou tlaku vzduchu na hladine mora (t.j. 1013.25 hPa), prípadne pre presnejší odhad nadmorskej výšky použijeme absolútny tlak vzduchu lokálnej meteorologickej stanice. Fragment zdrojového kódu v jazyku C - vzorec pre prepočet absolútneho tlaku vzduchu na relatívny:

```
pressure_sea = (bme280_compensate_pressure_double(v_uncomp_pressure_s32)/100) / pow(1 -  
((0.0065 * altitude) / (bme280_compensate_temperature_double(v_uncomp_temperature_s32) +  
(0.0065 * altitude) + 273.15)), 5.257);
```

Prostredníctvom spomenutého ovládača od Bosch Sensortec bolo nutné vykonať úpravu existujúcej programovej implementácie pre platformu ESP32 vo frameworku ESP-IDF. Samotný ovládač od Bosch Sensortec obsahuje inicializáciu komunikácie medzi mikrokontrolérom ESP32 a senzorom Bosch BME280 na I2C zbernici, umožňuje nastaviť dátové GPIO vývody pre signály SCL (Synchronizačné hodiny - hodinový signál) a SDA (Synchronizované dáta), ktoré komunikácia vyžaduje, definuje rýchlosť zbernice - hodinový signál v Hz. Ovládač od Bosch Sensortec implementuje dva prevádzkové režimy senzora Bosch BME280, ktoré je možné použiť v programovej implementácii:

- Normálny
- Vynútený (FORCED)

Normálny prevádzkový režim umožňuje využiť vyššie vzorkovanie nameraných údajov pre dosiahnutie presných meraní spoločne s nastavením koeficienta IIR (filter s nekonečnou impulznou odozvou) filtra, ktorý kompenzuje krátkodobé výkyvy tlaku vzduchu a teploty. Vzorkovanie je možné nastaviť v rozsahu 1 až 16 krát [34]. Koeficient IIR filtra je možné nastaviť v rozsahu 0 až 16. Medzi jednotlivými vzorkami je možné nastavovať pauzu - tzv. StandBy čas, ktorý definuje pauzu medzi meraniami.

Hodnota sa nastavuje makrami, ktoré sú preddefinované v ovládači v rozsahu 1 až 4000 ms (možno nastaviť aj hodnoty 63, 125, 250, 500, 1000, 2000). Energeticky úspornejší je prevádzkový režim FORCED (tzv. vynútený). Nevyužíva koeficient pre IIR filter, nekompenzuje žiadnym spôsobom krátkodobé výkyvy tlaku a teploty vzduchu. Využíva vzorkovanie 1 krát pre každú meranú hodnotu, teda prvá nameraná hodnota je výsledkom danej meteorologickej veličiny.

Pri výpise a porovnaní oboch režimov je normálny prevádzkový mód stabilný a ponúka konzistentné merania pri kontrole výpisu na tri desatinné miesta (na tisíciny), FORCED režim nebol tak stabilný a neponúkal konzistentné merania, všetky merateľné údaje sa zmenili skokovito pri každom meracom cykle nad i pod strednú hodnotu (známu z merania normálnym prevádzkovým módom). Senzorový uzol vychádza z realizácie mojej bakalárskej práce - Inteligentné relé v sieti eduroam, kde je snaha o zachovanie pôvodných funkcií systému a implementovaných možností do webového rozhrania.

Mikrokontroler ESP32 využíva z pôvodného konceptu bakalárskej práce aj ovládanie digitálneho výstupu GPIO 23, ktorý riadi SSR (Solid-state relay) relé [35]. Tento typ relé neobsahuje v porovnaní s elektromagnetickým relé cievku, ani mechanickú časť. Z pohľadu opotrebenia jeho častí má tak prakticky neobmedzenú životnosť prevádzky, nakoľko je výkonovým prvkom polovodičová súčiastka – triak, ktorý je ovládaný tranzistorom. Bližší opis SSR relé OMRON G3MB-202P [36] je dostupný v prílohe A na CD nosiči.

Tab. 3 Pripojenie vývodov ESP32 k vývodom použitých periférií

<b>ESP32</b>	<b>Bosch BME280 (senzor teploty, tlaku a vlhkosti vzduchu)</b>
3,3V	Vcc
GND	GND
GPIO22 (HW SCL)	SCL
GPIO21 (HW SDA)	SDA
<b>ESP32</b>	<b>SSR relé OMRON G3MB-202P - externé 5V napájanie</b>
GPIO23	CH1
GND	GND

Súčasťou systému je webové rozhranie, ktoré umožňuje zber nameraných údajov zo senzorového uzla a ovládanie jeho výstupu v automatickom, alebo manuálnom režime na spôsob izbového termostatu s konfiguráciou cieľovej (referenčnej) teploty a hysterézy [37]. Základné rozhranie spoločne s vygenerovaním a nakonfigurovaním certifikátov certifikačnej autority a webservera pre realizáciu zabezpečeného spojenia bolo vytvorené v mojej bakalárskej práci.

Návod na inštaláciu webového rozhrania, generovanie jednotlivých certifikátov kryptografickým nástrojom OpenSSL je dostupný v prílohe A na CD nosiči v časti Návod. Návod je univerzálny pre operačný systém webservera Cent OS a pre serverovú službu HTTPD [38]. Súčasťou príloh je aj opis funkcií webového rozhrania, ktoré ponúkala základná verzia z bakalárskej práce.

### 3.1. Úpravy webového rozhrania

Realizácia diplomovej práce na strane webservera spočívala v úpravách existujúceho webového rozhrania po grafickej stránke, počtu a následnej vizualizácii zaznamenaných údajov zo senzorového uzla. Zaznamenané údaje vo webovom rozhraní sú vizualizované na hlavnej stránke – ukážka zaznamenaných dát na obrázku Obr. 14.

Entita	Hodnota
Teplota (BME280)	21.16 °C
Vlhkosť (BME280)	54.99 % RH
Nadmorská výška	831.66 m.n.m.
Tlak (BME280) - RAW	917.27 hPa
Tlak (BME280) - prepočítaný	1009.37 hPa
Teplota - CPU	66 °C
Hall - CPU	65451 (Analog)

21.03.2021 00:47:43

Obr. 14 Vizualizácia nameraných údajov vo webovom rozhraní

Úpravy backendu (funkčnosti na serverovej strane) sa týkali PHP [39] súboru *zapisdata.php*, ktorý dokáže prevziať dáta, ktoré odosiela mikrokontrolér ESP32 a na základe ktorých vie vykonať logiku termostatu. V predmetnom PHP súbore som zmenil HTTP metódu pre príjem dát, pôvodná metóda HTTP GET bola nahradená za HTTP POST, ktorá je pre prenos údajov zo senzorového uzla bezpečnejšia, nakoľko sú dáta obsiahnuté v tele správy a nie v URL adrese ako parametre kľúčov s hodnotami.

Do webového rozhrania bola implementovaná samostatná - nová PHP podstránka, ktorá umožňuje nahráť firmvér pre mikrokontrolér ESP32 v binárnom formáte cez HTML formulár (Grafické rozhranie ukázané na obrázku Obr. 15).

Firmvér firmware.bin pre OTA aktualizáciu bol úspešne nahratý do webového rozhrania!

Nie je vybra...žiadny súbor

**OTA firmvér dostupný od:** 21. Mar 2021 00:36:52  
**Checksum firmvéru (SHA-256):** 52752fc58f0927d8d8b2aecad3e7c2d3aa0cc39a7bf5b9e5dcca841965a27f0b

Obr. 15 Grafické rozhranie webstránky pre nahratie OTA firmvéru cez HTML formulár

Serverový jazyk PHP pri spracovaní overí formát súboru (očakáva .bin), následne ho premenuje na firmware.bin (uložený firmvér na disku počítača po kompilácii príkladu Native OTA je native\_ota.bin) a uloží ho do preddefinovaného – koreňového umiestnenia priečinku webaplikácie, kde ho očakáva mikrokontrolér, nakoľko na toto cieľové umiestnenie a meno firmvéru firmware.bin vykonáva GET požiadavku pre stiahnutie jeho obsahu.

Používateľ má možnosť po nahratí firmvéru vidieť jeho kontrolný súčet – checksum (SHA-256) a tiež dátum a čas jeho publikácie. Obdobne je možné pre nahratie firmvéru do webového rozhrania použiť aj FTP klienta (napríklad WinSCP, Total Commander, Filezilla). Toto riešenie som využíval v čase, keď som ešte formulár pre nahratie firmvéru nemal vytvorený a implementovaný do webového rozhrania.

Nakoľko jazyk PHP nemá implementáciu podpory pre kontrolu podpísaného firmvéru (resp. súboru), nepodarilo sa mi do serverovej časti doplniť kontrolu pre akceptáciu iba podpísaného firmvéru, prípadne ho overiť dodatočne po zápise do zložky.

### 3.2. Vytvorenie a úpravy programu v ESP-IDF

Hlavnú aplikáciu mikrokontroleru ESP32 pre senzorový uzol som navrhol v prostredí ESP-IDF. Využil som základný program Native OTA (kap. 2.2.), ktorý mi umožnil získať WiFi konektivitu a prevziať firmvér z webservera zabezpečeným prenosovým kanálom – HTTPS protokol. Po zapnutí hardvérovej funkcionality Secure Boot (kap. 2.4.) som dokázal overiť integritu stiahnutého firmvéru z webservera na internete, ktorý je digitálne podpísaný a taktiež som zabezpečil bootovací proces.

Súčasťou aplikácie je aj prenos údajov zo senzorového uzla na rovnaké webové rozhranie, ktoré poskytuje OTA aktualizáciu firmvéru. Následne som použil príklad HTTPS\_request, ktorý je dostupný v ESP-IDF a skombinoval som ho s príkladom Native OTA. Vykonával som niekoľko úprav v konfiguračných súboroch aj v samotnom programe, nakoľko pôvodne príklad HTTPS\_request

využíva samostatný súbor pre certifikát certifikačnej autority s iným názvom ako Native OTA a v inom umiestnení (hlbka priečinka).

Úpravou som dosiahol, že OTA aj HTTPS request využíva rovnaký súbor s certifikátom certifikačnej autority. Programová implementácia oboch príkladov (Native OTA a HTTPS\_request) bola jednoduchá, nakoľko obe implementácie využívajú tasky a tak je ich možné spúšťať nezávisle na sebe. V prípade chyby v programe a pretečení buffra sa ukončí iba konkrétny task a ostatné tasky pokračujú v behu. Celkovo som využil tri HTTPS tasky pre tri rôzne druhy requestov, ktoré sa cyklicky vykonávajú.

Základná implementácia príkladu HTTPS\_request využíva GET request. Nakoľko pre prenos údajov mikrokontroler vykonáva POST request, musel som do programovej implementácie vykonať úprau HTTP metódy, vypočítať dĺžku payloadu (dát) a nastaviť encoding (application/x-www-form-urlencoded) payloadu a následne odoslať payload (na základe normy RFC 1867, ktorá ten typ prenosu a formátu dát definuje [40]).

Doba opakovania tasku sa realizuje nastavením parametra funkcie *vTaskDelay()* v programovej implementácii s časom čakacej slučky v milisekundách. Pre lepšie rozlíšenie hlášok na UART rozhraní od jednotlivých taskov som využil systém tagovania, ktorý vypíše informáciu o názve tasku, ktorý dané hlásenie na UART rozhranie posiela. Táto metóda bola efektívna aj pri debugovaní programu.

Na základe dôležitosti správy je možné využiť rôzne typy príznakov, ktoré dokážu výpis na UART rozhranie sformátovať do bielej, zelenej, červenej, žltej farby prostredníctvom funkcie *ESP\_LOGI* (informácia), *ESP\_LOGW* (varovanie), *ESP\_LOGE* (chyba) [41]...

Tab. 4 Tasky senzorového uzla pre odosielanie a načítanie dát z webového rozhrania

Názov tasku	Funkcia
https_get_task	Vykonáva pravidelný HTTPS POST request s dátami obsiahnutými v tele správy. Pri spustení tasku sa vyskladá request z nameraných údajov, ktoré sú obsiahnuté v globálnych premenných. Spustením .php scriptu sa vykoná aj logika termostatu. Request sa opakuje každých 15 sekúnd.
https_get_task2	Vykonáva pravidelný HTTPS GET request na textový súbor, načítava stav ZAP / VYP pre výstup - relé, aplikuje na GPIO23 (D23). Request sa opakuje každých 15 sekúnd.
https_get_task3	Vykonáva pravidelný HTTPS GET request na textový súbor, načíta stav OK / RST. Ak načíta RST, vykoná softvérový reštart ESP, predtým opätovným HTTP

	requestom potvrdí reštart, zmení obsah súboru na OK. Request sa opakuje každých 15 sekúnd.
--	--

Súčasťou práce s taskom `https_get_task2` bolo aj využitie digitálneho GPIO vývodu 23, ktorý je využitý pre riadenie SSR relé. Inicializácia vývodu, nastavenie vývodu ako výstupu sa realizovala vo funkcii `main` ešte pred spustením jednotlivých taskov. Tasky `https_get_task` až `https_get_task3` implementovali funkcionality, ktorá existovala už z bakalárskej práce v prostredí Arduino Core.

Programová implementácia pre meteorologický senzor Bosch BME280 využíva Driver od Bosch. `Sensortec` je navrhnutá pre jazyk C. Niekoľkými úpravami v spolupráci s prof. Ing. Milošom Drutarovským, CSc. sa nám podarilo implementovať pôvodnú programovú implementáciu do frameworku ESP-IDF. Ako som v kapitole 3 naznačil, senzor Bosch BME280 dokáže fungovať v dvoch hlavných operačných režimoch. Pre každý z režimov som vytvoril samostatný task, ktorý vykonáva inicializáciu a komunikáciu so senzorom v konkrétnom režime.

Aby bol iba jeden z taskov podmienene zapnutý, vytvoril som v konfiguračnom súbore `Kconfig.projbuild` [42], ktorý je dostupný v priečinku projektu samostatné druhé menu (prvé menu je existujúca konfigurácia Native OTA príkladu). Menu je rozvetvené na dve podmenu ("I2C Master" a "BME280 Sensor"). I2C Master ponúka zvolenie hodinového signálu v Hz a čísla GPIO pre SCL a SDA signál, ktoré sú štandardne nastavené na hardvérové I2C vývody.

Druhé menu – "BME280 Sensor" umožňuje výber komunikačnej I2C adresy senzora Bosch BME280 (Obr. 16 popisuje hlavné podmenu a jednotlivé vetvenia menu pre "I2C Master", "BME280 Sensor"). Na základe logickej úrovne vývodu SD0 je jeho komunikačná adresa na I2C zbernici 0x76 (PULLDOWN), alebo 0x77 (PULLUP), obe adresy sú v hexadecimálnom tvare.



```

C:\> ESP-IDF Command Prompt (cmd.exe) - idf.py menuconfig
(Top) → Example BME280 Configuration
Espressif IoT Development Framework Configuration
I2C Master --->
BME280 Sensor --->

C:\> ESP-IDF Command Prompt (cmd.exe) - idf.py menuconfig
(Top) → Example BME280 Configuration → I2C Master
Espressif IoT Development Framework Configuration
(22) SCL GPIO Num
(21) SDA GPIO Num
(100000) Master Frequency

C:\> ESP-IDF Command Prompt (cmd.exe) - idf.py menuconfig
(Top) → Example BME280 Configuration → BME280 Sensor
Espressif IoT Development Framework Configuration
BME280 I2C Address (BME280 (0x76) - 4 PIN - ORIG. ADDR) --->
BME280 Operation Mode (Normal Mode) --->

C:\> ESP-IDF Command Prompt (cmd.exe) - idf.py menuconfig
(Top) → Example BME280 Configuration → BME280 Sensor → BME280 I2C Address
Espressif IoT Development Framework Configuration
( ) BME280 (0x77) - 6 PIN + GND SDO
(X) BME280 (0x76) - 4 PIN - ORIG. ADDR

```

Obr. 16 Vlastné konfiguračné menu pre voľbu I2C parametrov a adresy, režimu BME280

Logická úroveň tohto vývodu mení LSB (Najmenej významný bit) adresy. Taktiež je možné vybrať operačný mód senzora Bosch BME280 NORMAL / FORCED. Premenné obsiahnuté v menu sú uložené do makier, ktoré môžu nadobúdať príznak y (yes), n (no), označujú, či je makro definované, prípadne nadobúdajú číselnú, textovú hodnotu. Všetky konfiguračné makrá majú predponu CONFIG sú uložené do súboru sdkconfig po úpravách v Menuconfigu, kde sú jednotlivé menu vyobrazené.

Pre prístup aplikácie napísanej v jazyku C k sdkconfigu je nutné v programe definovať makro COMBINED\_INIT\_CODE. Následne už môže aplikácia používať jednotlivé makrá z sdkconfigu. Makro podporované v sdkconfig môže mať dátový typ boolean (binárny yes/no), integer (celé číslo), string (plaintext). Na základe dátového typu, hodnoty je možné v programe vykonávať prostredníctvom direktív podmienenú kompiláciu. V mojom prípade som overoval, či je definované makro CONFIG\_BME280\_OPMODE a akú hodnotu nadobúda. Hodnota 0x01 je pre FORCED operačný režim, alebo 0x03 pre NORMAL operačný režim.

Daná hodnota je hodnotou registra, ktorým sa daný operačný režim senzora Bosch BME280 definuje. Táto hodnota sa posiela do senzora pri inicializácii spojenia mikrokontrolérom. Konfiguračné menu vždy zadefinuje iba jeden režim, neumožňuje zvoliť obe možnosti súčasne, tým

je zaručené, že sa vždy po kompilácii spúšťa iba konkrétny task na základe zvoleného operačného režimu.

Každý z BME taskov zapisuje namerané údaje do globálnych premenných, ktoré používa HTTPS task pre prenos údajov do webového rozhrania. HTTPS task je univerzálny a funguje nezávisle na zvolenom režime senzora Bosch BME280. Celkovo tak v aktuálnej verzii aplikácie funguje v reálnom čase nezávisle na sebe päť taskov (OTA task + 3x HTTPS task + BME280 task).

### 3.3. Minimálna schéma zapojenia

Súčasťou diplomovej práce je aj návrh minimálnej schémy zapojenia, ktorá je navrhnutá pre samostatný čip ESP32-WROOM-32 a je plným ekvivalentom súčasného zapojenia s vývojovou doskou DevKit. Schéma obsahuje aj návrh napájacej časti s regulátormi napätových úrovní pre možnosť napájania externým zdrojom napájania v rozsahu 6V až 18V.

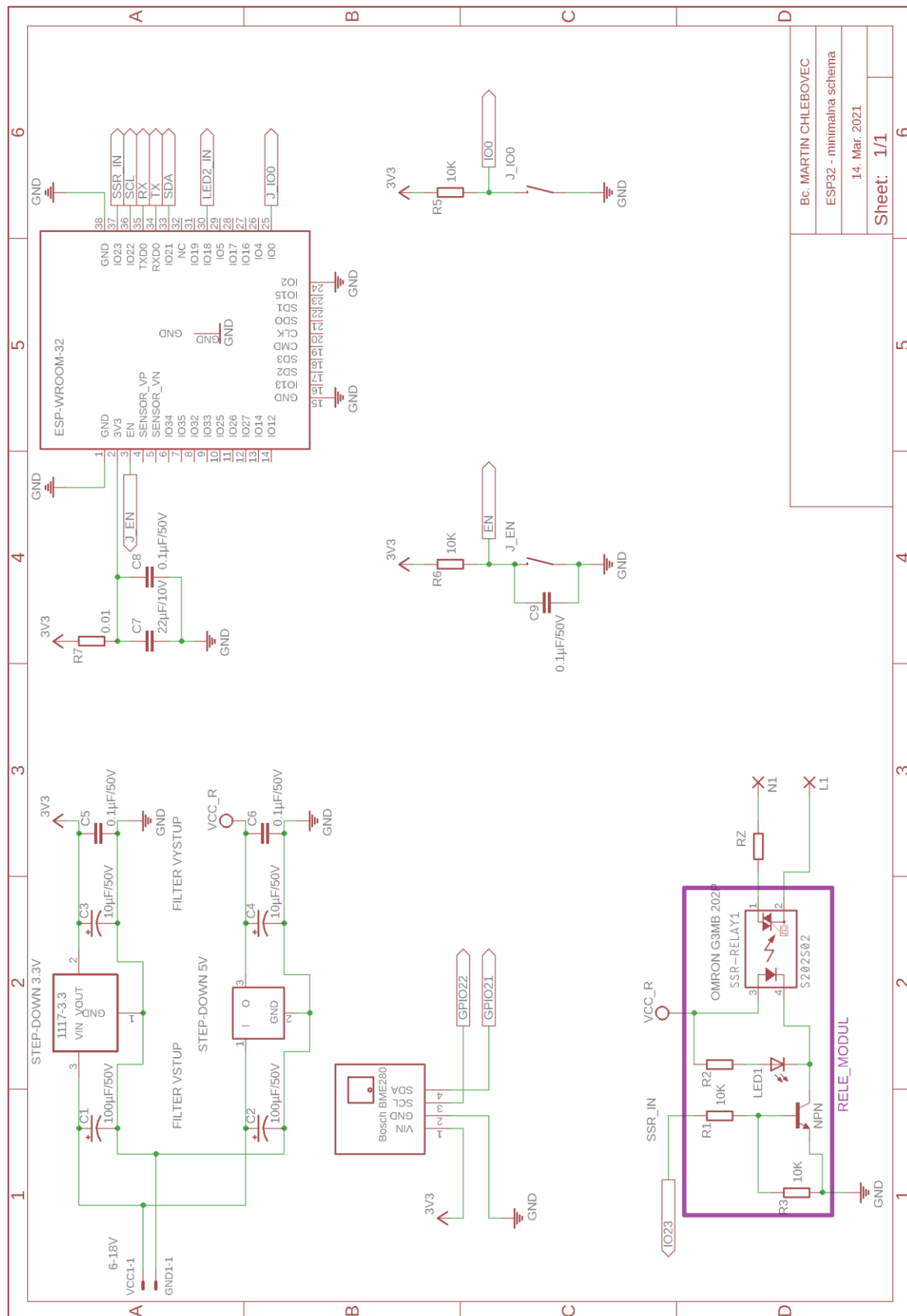
V prípade použitia samostatného čipu sú do schémy zakomponované tlačidlá pre signály EN a BOOT, ktorými je možné nastavovať špecifický režim čipu ESP32 na základe kombinácie ich stlačenia. Špecifickou sekvenciou stlačení je možné nastaviť hlavný čip ESP32 do režimu nahrávania programu (tzv. programovací režim), prípadne režim pre jeho stiahnutie, prípadne je možné vykonať reštart a spustenie bootovacieho procesu, aplikácie.

Navrhnutá časť signálov pre EN a BOOT vývod mikrokontroléru ESP32 prostredníctvom tlačidiel je navrhnutá na základe známeho problému DevKitov od Espressifu, ktorý má pre oba signály použitý kondenzátor s totožnou kapacitou, čo má za následok rovnaký napätový nábeh signálov v čase. Nakoľko sú oba signály pri spúšťaní mikrokontroléru v logickej 0, doska sa prepne do režimu sťahovania programu, kde čaká v nekonečnej slučke a pre beh programu sa vyžaduje fyzický reštart tlačidlom.

Tento problém je nebezpečný najmä v prípade, ak ESP32 riadi kritickú aplikáciu na ktorú pri power-on cykle (odpojený a pripojený napájania) nedokáže reagovať až do fyzického reštartu tlačidlom, kedy sa spustí bootovateľný program. Tento typ hardvérového problému je ľahko riešiteľný, nakoľko stačí použiť jeden z kondenzátorov s inou kapacitou, alebo jeden nepoužiť ako v prípade mojej minimálnej schémy zapojenia, aby bola na jednom zo signálov logická úroveň 1 okamžite.

V schéme zapojenia (na Obr. 17) je zakreslený relé modul OMRON G3MB-202P ku ktorému však neexistuje dostupná elektrotechnická schéma (existuje iba pre relé samostatne, nie pre relé modul), jeho zapojenie je tak iba približné. Schéma zapojenia periférie s mikrokontrolérom ESP32 je plne ekvivalentná s tabuľkovým zapojením vývodov z tabuľky (Tab. 3) v kapitole 3.

Na výstup relé je možné pripojiť záťaž (v minimálnej schéme zapojenia na Obr. 17 označené ako RZ) – pre demonštratívne účely som využil LED žiarovku na 230V s výkonom 5W. Záťaž môže mať maximálne prúdové zaťaženie 2A pri 230V vzhľadom na charakteristiku odporúčanú v katalógovom liste predmetného SSR relé OMRON G3MB-202P.



Obr. 17 Minimálna schéma senzorového uzla

## Záver

Diplomová práca opisuje možnosti IoT mikrokontrolérovej platformy ESP32 v rôznych aplikáciách s možnosťou aktualizácie firmvéru lokálnymi, alebo externými OTA metódami. Zameriava sa predovšetkým na bezpečnosť, ktorú je možné implementovať vývojárskymi nástrojmi v prostredí frameworku ESP-IDF. Zachováva pôvodnú funkčnosť senzorového uzla z bakalárskej práce a implementuje jeho funkcionality do prostredia ESP-IDF.

Metóda digitálneho podpisu má široké využitie nielen v mikrokontrolerovej technike. Digitálny podpis je spoľahlivá metóda pre overenie integrity akéhokoľvek súboru (dokumenty, spustiteľné súbory, firmvér). Pre túto metódu moja implementácia využíva súkromný kľúč generovaný eliptickou krivkou NIST256p s dĺžkou kľúča 256 bitov, ktorá má porovnateľnú kryptografickú bezpečnosť s kľúčom generovaným cez algoritmus RSA s dĺžkou kľúča 3072 bitov.

Použitie kryptografie na báze ECC sa výrazne redukuje miesto a pamäťová náročnosť na uloženie kľúča. Násobne sa skrúti aj čas pre dešifrovanie v porovnaní s RSA. Pôvodná implementácia Native OTA po overení integrity firmvéru dokáže na základe overenia verzie vykonať zmenu príznaku v OTA\_DATA partícii a vykonať softvérový reštart mikrokontroléru pre bootovanie nového firmvéru.

Pre zabezpečenie samotného bootovacieho procesu som implementoval hardvérovú metódu Secure Boot prostredníctvom zápisu šifrovacieho kľúča do jednorázovo programovateľnej pamäte eFuse BLK2 a následne som funkcionality permanentne zapísal zápisom bitu do 1-bitovej eFuse ABS\_DONE\_0. Táto implementácia zabezpečuje, že systém nabootuje, iba ak bol do mikrokontroléru nahratý dôveryhodný bootloader a správny odtlačok zo SBDA algoritmu.

Nakoľko eFuse nie je softvérovo druhý krát zapisovateľná, alebo čitateľná, nedokáže potenciálny útočník získať pôvodný šifrovací kľúč, respektíve ho nedokáže získať metódou totálnych skúšok (brute force) s aktuálnymi výpočtovými prostriedkami v prijateľnom čase (čas útoku prevyšuje plánované použitie aplikácie).

Súčasťou príloh je aj návod pre vygenerovanie certifikátov pre webserver, certifikačnú autoritu, kompletne nastavenie webservera. Pre mikrokontroler ESP32 je vytvorený návod pre kompletnú implementáciu digitálneho podpisu, Secure Boot funkcionality

## Zoznam použitej literatúry

- [1]. ESP32 [online]. Wikipedia [cit. 2021-02-18]. Dostupné z: <https://en.wikipedia.org/wiki/ESP32>
- [2]. ESP32 Modules and Boards [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z: <https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/hw-reference/modules-and-boards.html>
- [3]. ESP32 Technical Reference Manual [online]. 2.1 Introduction [cit. 2020-01-01]. Dostupné z: [https://www.espressif.com/sites/default/files/documentation/esp32\\_technical\\_reference\\_manual\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf)
- [4]. Application startup flow [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/general-notes.html#application-startup-flow>
- [5]. ESP32 ROM Console [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/romconsole.html>
- [6]. ESP32 Technical Reference Manual [online]. 2.2 Features [cit. 2020-01-01]. Dostupné z: [https://www.espressif.com/sites/default/files/documentation/esp32\\_technical\\_reference\\_manual\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf)
- [7]. Partition Tables [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z: <https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/api-guides/partition-tables.html>
- [8]. eFuse Manager [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z: <https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/api-reference/system/efuse.html>
- [9]. Sleep Modes [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z: [https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/api-reference/system/sleep\\_modes.html](https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/api-reference/system/sleep_modes.html)
- [10]. ESP32 ULP coprocessor instruction set [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z: [https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/api-guides/ulp\\_instruction\\_set.html](https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/api-guides/ulp_instruction_set.html)
- [11]. ESP32 Modules and Boards [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z: <https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/hw-reference/modules-and-boards.html>

- 
- [12]. Katalógový list ESP32-S [online]. Shenzhen Anxinke Technology [cit. 2016-10-03].  
<https://www.es.co.th/Schemetic/PDF/ESP32.PDF>
- [13]. Get Started [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z:  
<https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/get-started/index.html>
- [14]. ESP-IDF Release v4.2 [online]. Github [cit. 2020-12-07]. Dostupné z:  
<https://github.com/espressif/esp-idf/releases/tag/v4.2>
- [15]. The FreeRTOS™ Reference Manual [online]. Amazon.com, Inc. [cit. 2018-07-01].  
Dostupné z: [https://www.freertos.org/fr-content-src/uploads/2018/07/FreeRTOS\\_Reference\\_Manual\\_V10.0.0.pdf](https://www.freertos.org/fr-content-src/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.pdf)
- [16]. ESPTOOL [online]. Github [cit. 2021-02-22]. Dostupné z:  
<https://github.com/espressif/esptool/blob/master/esptool.py>
- [17]. Arduino Core for the ESP32[online]. Github [cit. 2021-02-24]. Dostupné z:  
<https://github.com/espressif/arduino-esp32>
- [18]. Over-the-air programming [online]. Wikipedia [cit 2021-02-04]. Dostupné z:  
[https://en.wikipedia.org/wiki/Over-the-air\\_programming](https://en.wikipedia.org/wiki/Over-the-air_programming)
- [19]. How to Approach OTA Updates for IoT [online]. DZone IoT [cit 2018-01-19].  
Dostupné z: <https://dzone.com/articles/how-to-approach-ota-updates-for-iot>
- [20]. Basic OTA [online]. Github [cit. 2018-03-04]. Dostupné z:  
<https://github.com/espressif/arduino-esp32/blob/master/libraries/ArduinoOTA/examples/BasicOTA/BasicOTA.ino>
- [21]. OTA Web Updater [online]. Github [cit. 2020-11-02]. Dostupné z:  
<https://github.com/espressif/arduino-esp32/blob/master/libraries/ArduinoOTA/examples/OTAWebUpdater/OTAWebUpdater.ino>
- [22]. Simple OTA [online]. Github [cit. 2021-01-15]. Dostupné z:  
[https://github.com/espressif/esp-idf/blob/master/examples/system/ota/simple\\_ota\\_example/main/simple\\_ota\\_example.c](https://github.com/espressif/esp-idf/blob/master/examples/system/ota/simple_ota_example/main/simple_ota_example.c)
- [23]. Native OTA [online]. Github [cit. 2021-01-15]. Dostupné z:  
[https://github.com/espressif/esp-idf/blob/master/examples/system/ota/native\\_ota\\_example/main/native\\_ota\\_example.c](https://github.com/espressif/esp-idf/blob/master/examples/system/ota/native_ota_example/main/native_ota_example.c)
- [24]. Signed App Verification Without Hardware Secure Boot [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z:  
<https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/security/secure-boot-v1.html#signed-app-verification-without-hardware-secure-boot>
-

- 
- [25]. ESPSECURE.py [online]. Github [cit. 2021-01-19]. Dostupné z: <https://github.com/espressif/esptool/blob/master/espsecure.py>
- [26]. Remote Signing of Images [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z: <https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/security/secure-boot-v1.html#remote-signing-of-images>
- [27]. Secure Boot [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z: <https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/security/secure-boot-v1.html#secure-boot>
- [28]. ESPEFUSE.py [online]. Github [cit. 2020-12-15]. Dostupné z: <https://github.com/espressif/esptool/blob/master/espefuse.py>
- [29]. Secure Boot Digest Algorithm [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z: <https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/security/secure-boot-v1.html#secure-bootloader-digest-algorithm>
- [30]. Katalógový list CP2102 [online]. Silicon Labs [cit. 2017-01-20]. <https://www.silabs.com/documents/public/data-sheets/CP2102-9.pdf>
- [31]. BME280 Driver - Bosch Sensortec [online]. Github [cit. 2020-07-06]. Dostupné z: [https://github.com/BoschSensortec/BME280\\_driver](https://github.com/BoschSensortec/BME280_driver)
- [32]. Katalógový list Bosch BME 280 [online]. Bosch Sensortec [cit. 2018-11-04]. <https://readthedocs.org/projects/bme280/downloads/pdf/latest/>
- [33]. I<sup>2</sup>C [online]. Wikipedia [cit. 2021-02-07]. Dostupné z: <https://en.wikipedia.org/wiki/I%C2%B2C>
- [34]. Dokumentácia BME280 Bosch Sensortec Driver [online]. Texas Instruments [cit. 2018-01-01]. [http://software-dl.ti.com/simplelink/esd/simplelink\\_cc13x2\\_sdk/2.20.00.71/exports/docs/thread/doxygen/thread/html/bme280\\_8h.html](http://software-dl.ti.com/simplelink/esd/simplelink_cc13x2_sdk/2.20.00.71/exports/docs/thread/doxygen/thread/html/bme280_8h.html)
- [35]. Solid-state relay [online]. Wikipedia [cit. 2021-03-02]. Dostupné z: [https://en.wikipedia.org/wiki/Solid-state\\_relay](https://en.wikipedia.org/wiki/Solid-state_relay)
- [36]. Katalógový list OMRON G3MB-202P [online]. Omron Electronics [cit. 2021-02-27]. <http://www.omron-russia.com/doc/relay/ssr/g3mb.pdf>
- [37]. Termostat [online]. Wikipédia [cit. 2019-10-06]. Dostupné z: <https://sk.wikipedia.org/wiki/Termostat#Hyster%C3%A9za>
- [38]. HTTPD – Dokumentácia v2.4 [online]. Apache server HTTP project [cit. 2021-03-13]. Dostupné z: <http://httpd.apache.org/docs/2.4/>
- [39]. PHP – Dokumentácia [online]. PHP Group [cit. 2021-03-01]. Dostupné z: <https://www.php.net/docs.php>
-



- [40]. RFC 1867 [online]. Internet Engineering Task Force [cit. 1995-11-01]. Dostupné z:  
<https://tools.ietf.org/html/rfc1867>
- [41]. Logging library [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z:  
<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/log.html>
- [42]. Project Configuration [online]. ESP-IDF Programming Guide [cit. 2021-02-27].  
Dostupné z: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/kconfig.html>

## Prílohy

Príloha A: CD médium – diplomová práca v elektronickej podobe, prílohy v elektronickej podobe

Príloha B: Používateľská príručka

Príloha C: Systémová príručka