|  |  |
| --- | --- |
| **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 diplomovej práce je využitie mikrokontroléru ESP32 v úlohe senzorového uzla, ktorý dokáže vykonať vzdialenú aktualizáciu firmvéru cez internet. Proces vzdialenej aktualizácie kladie prioritu najmä na bezpečnosť, nakoľko s dá prenášajú cez internet, čo je nebezpečné prostredie pre prenos citlivých informácii a údajov. Mikrokontrolérová platforma ESP32 umožňuje implementovať mechanizmy, ktoré dokážu zabezpečiť integritu (nepozmenenosť) prevzatého firmvéru od vytvorenia autorom zo servera dosiahnuteľného cez internet bezpečným prenosovým kanálom. Nakoľko môže byť mikrokontróler ESP32 vystavený aj rôznym pokusom o spustenie nedôveryhodného firmvéru, alebo pokusu o zmenu Bootloadera, platforma je vybavená mechanizmami, ktoré dokážu zabrániť spusteniu nedôveryhodného firmvéru. Mechanizmy dokážu zabezpečiť aj bootovací proces a bez dôveryhodného Bootloadera mikrokontróler nepristúpi k bootovaniu firmvéru. Mikrokontróler ESP32 skorších revízií neumožňuje zakázať stiahnutie obsahu flash pamäte cez fyzické rozhranie, dokáže však obsah flash pamäte pred útočníkom šifrovať a znemožniť mu spustiť zašifrovaný firmvér. Mechanizmy je možné aplikovať vo vývojovom prostredí – frameworku ESP-IDF priamo z produkcie firmy Espressif Systems, ktorá produkuje mikrokontrolérové platformy ESP32 a poskytuje im plnú hardvérovú a softvérovú podporu.

.

**Kľúčové slova v SJ**

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

**Abstrakt v AJ**

The main topic of the diploma thesis is the use of the ESP32 microcontroller in the role of a sensor node that can perform remote firmware updates via the Internet. The remote update process places particular priority on security, as it can be transmitted over the Internet, which is a dangerous environment for the transmission of sensitive information and data. The ESP32 microcontroller platform enables the implementation of mechanisms that can ensure the integrity (immutability) of the acquired firmware since its creation by the author from a server accessible via the Internet via a secure transmission channel. Because the ESP32 microcontroller can also be exposed to various attempts to run untrusted firmware, or to attempt to change the Bootloader, the platform is equipped with mechanisms that can prevent untrusted firmware from running. The mechanisms can also secure the boot process, and without a trusted bootloader, the microcontroller will not boot the firmware. The ESP32 microcontroller of earlier revisions does not allow to prevent the download of flash memory content via the physical interface, but it can encrypt the contents of flash memory in front of an attacker and prevent him from running encrypted firmware. The mechanisms can be applied in a development environment - the ESP-IDF framework directly from the production of Espressif Systems, which produces ESP32 microcontroller platforms and provides them with full hardware and software support.

**Kľúčové slova v AJ**

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

**Zadanie práce**

**Čestné vyhlásenie**

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

Košice, 18. apríla 2021 ..........................................

vlastnoručný podpis**Poďakovanie**

V prvom rade sa chcem 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](#_Toc69660951)

[Zoznam tabuliek 11](#_Toc69660952)

[Zoznam symbolov a skratiek 12](#_Toc69660953)

[Úvod 14](#_Toc69660954)

[1. ESP32 16](#_Toc69660955)

[1.1. Proces spustenia mikrokontroléru ESP32 17](#_Toc69660956)

[1.2. Vývojové kity, samostatný čip ESP32 19](#_Toc69660957)

[2. Pamäť mikrokontroléru ESP32 22](#_Toc69660958)

[2.1. Embedded (vstavaná) pamäť 23](#_Toc69660959)

[2.2. Externá pamäť 24](#_Toc69660960)

[2.3. Tabuľka partícii 25](#_Toc69660961)

[2.4. eFuses 27](#_Toc69660962)

[3. Nízkopríkonový režim mikrokontroléru ESP32 29](#_Toc69660963)

[4. Vývojársky framework ESP-IDF 31](#_Toc69660964)

[4.1. Vývojárske nástroje v ESP-IDF 31](#_Toc69660965)

[4.2. Vývoj frameworku ESP-IDF 32](#_Toc69660966)

[5. Arduino Core 35](#_Toc69660967)

[6. Vzdialená aktualizácia firmvéru 36](#_Toc69660968)

[6.1. Aktualizácia firmvéru v Arduino Core 36](#_Toc69660969)

[6.2. Aktualizácie firmvéru v ESP-IDF 40](#_Toc69660970)

[6.3. Metóda digitálneho podpisu 45](#_Toc69660971)

[6.3.1. Digitálny podpis – implementácia v prostredí ESP-IDF 46](#_Toc69660972)

[6.4. Secure Boot v ESP-IDF 51](#_Toc69660973)

[6.4.1. Secure Boot - implementácia v prostredí ESP-IDF 53](#_Toc69660974)

[6.4.2. SBDA algoritmus 56](#_Toc69660975)

[6.5. Šifrovanie flash pamäte 60](#_Toc69660976)

[6.5.1. Proces šifrovania flash pamäte 62](#_Toc69660977)

[6.5.2. Algoritmus šifrovania flash pamäte 63](#_Toc69660978)

[6.5.3. Šifrovanie flash pamäte – implementácia v prostredí ESP-IDF 64](#_Toc69660979)

[7. Realizácia programu pre aktualizáciu senzorového uzla 67](#_Toc69660980)

[7.1. Realizácia programu pre senzorový uzol 68](#_Toc69660981)

[7.2. Úpravy webového rozhrania 76](#_Toc69660982)

[7.3. Minimálna schéma zapojenia 79](#_Toc69660983)

[Záver 82](#_Toc69660984)

[Zoznam použitej literatúry 84](#_Toc69660985)

[Prílohy 89](#_Toc69660986)

Zoznam obrázkov

[Obr. 1 Proces spustenia mikrokontroléru ESP32 19](#_Toc69661021)

[Obr. 2 ESP32-DevKitC 21](#_Toc69661022)

[Obr. 3 Bloková schéma mikrokontroléru ESP32 22](#_Toc69661023)

[Obr. 4 Mapa pamäte mikrokontroléru ESP32 23](#_Toc69661024)

[Obr. 5 Bloková schéma flash pamäte s ofsetmi a rozdelením jednotlivých partícii 28](#_Toc69661025)

[Obr. 6 Sieťový OTA port v prostredí Arduino IDE 38](#_Toc69661026)

[Obr. 7 OTA Web Updater 40](#_Toc69661027)

[Obr. 8 Menuconfig – konfiguračné menu projektov v ESP-IDF 42](#_Toc69661028)

[Obr. 9 Úspešná aktualizácia firmvéru, reštart systému 44](#_Toc69661029)

[Obr. 10 Stiahnutie identického firmvéru, čakanie v slučke na reštart 44](#_Toc69661030)

[Obr. 11 ESP32 - Native OTA – proces aktualizácie 45](#_Toc69661031)

[Obr. 12 Podpísanie firmvéru dôveryhodným vydavateľom 46](#_Toc69661032)

[Obr. 13 Overenie digitálneho podpisu verejným kľúčom podpisujúceho 47](#_Toc69661033)

[Obr. 14 Nastavenie ofsetu tabuľky partícii v Menuconfig 49](#_Toc69661034)

[Obr. 15 Podpísanie firmvéru súkromným kľúčom v konzolovej aplikácii ESP-IDF 50](#_Toc69661035)

[Obr. 16 Úspešné overenie digitálneho podpisu firmvéru pri bootovaní 51](#_Toc69661036)

[Obr. 17 Neúspešné overenie digitálneho podpisu firmvéru 52](#_Toc69661037)

[Obr. 18 Secure Boot - neoverený softvérový Bootloader, zakázaná fáza bootovania 53](#_Toc69661038)

[Obr. 19 Nastavenie Secure Bootu a verejného kľúča pre Bootloader v Menuconfigu 54](#_Toc69661039)

[Obr. 20 Sumár eFuses zobrazených cez nástroj espefuse.py 55](#_Toc69661040)

[Obr. 21 Bloková schéma procesu Secure Boot po bootovanie firmvéru 61](#_Toc69661041)

[Obr. 22 Šifrovanie flash pamäte – Release režim 62](#_Toc69661042)

[Obr. 23 Konfigurácia bezpečnostných makier pre Secure Boot a Flash Encryption 66](#_Toc69661043)

[Obr. 24 Problém s načítaním obshu flash pamäte 67](#_Toc69661044)

[Obr. 25 Bosch BME280 - senzor teploty, tlaku, vlhkosti vzduchu 69](#_Toc69661045)

[Obr. 26 Konzistentnosť meraní senzora BME280 v NORMAL móde 71](#_Toc69661046)

[Obr. 27 Konzistentnosť meraní senzora BME280 vo FORCED móde 72](#_Toc69661047)

[Obr. 28 Webové rozhranie termostatu pre ovládanie relé 73](#_Toc69661048)

[Obr. 29 Vlastné konfiguračné menu pre voľbu I2C parametrov a adresy, režimu BME280 75](#_Toc69661049)

[Obr. 30 Bloková schéma riešenia senzorového uzla na báze ESP32 77](#_Toc69661050)

[Obr. 31 Vizualizácia nameraných údajov vo webovom rozhraní 77](#_Toc69661051)

[Obr. 32 Grafické rozhranie webstránky pre nahratie OTA firmvéru cez HTML formulár 78](#_Toc69661052)

[Obr. 33 HTTP Basic Auth 79](#_Toc69661053)

[Obr. 34 Minimálna schéma senzorového uzla 82](#_Toc69661054)

Zoznam tabuliek

[Tab. 1 Prevádzkové režimy mikrokontroléru ESP32 a stav periférii 30](#_Toc69661055)

[Tab. 2 Metódy zabezpečenia mikrokontroléru 46](#_Toc69661056)

[Tab. 3 Rozdiel ofsetu aplikačných partícii po a pred posunutím 49](#_Toc69661057)

[Tab. 4 Pripojenie vývodov ESP32 k vývodom použitých periférii 72](#_Toc69661058)

[Tab. 5 Tasky senzorového uzla pre odosielanie a načítanie dát z webového rozhrania 74](#_Toc69661059)

Zoznam symbolov a skratiek

ADC Analógovo-digitálny prevodník

AES Advanced Encryption Standard

ALU Arithmetic logic unit (aritmeticko-logická jednotka)

AP Access Point (prístupový bod)

API Application programming interface (rozhranie pre programovanie aplikácii)

BLE Bluetooth Low Energy

BLK Označenie bloku jednorázovo programovateľnej pamäte eFuse

COM Communication port (Sériová linka)

CSS Cascading Style Sheets (Kaskádové štýly)

DMA Direct Memory Access (Priamy prístup do pamäte)

DNS Domain Name System (Systém názvov domén)

DPS Doska plošných spojov

DRAM RAM pamäť dát

eFuse Elektronická poistka – jednorázovo programovateľná pamäť

ESP-IDF Esressif – IoT Development Framework

FTDI Future Technology Devices International – typ USB-UART prevodníku

FTP File Transfer Protocol (Protokol pre prenos súborov)

GPIO Vstupno-výstupný vývod

HTML Hypertextový značkovací jazyk

HTTP Hypertextový prenosový protokol

HTTPS Zabezpečený hypertextový prenosový protokol

I2C Intel-Integrated-Circuit - dvojvodičová obojsmerná zbernica

IEEE Inštitút elektrických a elektronických inžinierov (organizácia, norma)

IIR Filter s nekonečnou impulznou odozvou, vyžaduje spätnú väzbu

IO Vstup-výstup

IoT Internet of Things (internet vecí)

IRAM RAM pamäť inštrukcií

LSB Less Significant Bit (Najmenej významný bit)

mDNS Multicast Domain Name System

MMU Memory Management Unit (jednotka správy pamäte)

OTA Over The Air – metódy opisujúce 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

RMII Reduced media-independent interface

ROM Pamäť len na čítanie, energeticky nezávislá

RTC Real Time Clock (hodiny reálneho času)

SBDA Secure Boot Digest Algorithm (Algoritmus zabezpečeného bootovacieho procesu)

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 vysielač

ULP Ultra Low Power (Režim nízkeho odberu elektrickej energie)

USB Univerzálna sériová zbernica

USB-UART Prevodník USB signálov na UART rozhranie

XOR Exkluzívny súčet

Úvod

Senzorové siete sú rozšírené v oblastiach priemyslu, automatizácie a internetu vecí. Poskytujú zber dát na základe ktorých je možné riadiť výkonové spotrebiče, stroje a automatizovať výrobné procesy a zjednodušiť každodenný život. Mikrokontroléry našli uplatnenie v rôznych aplikáciách s nízkym prúdovým odberom, kde môžu byť prevádzkované na batériu niekoľko mesiacov až rokov. Medzi tieto mikrokontroléry môžeme zariadiť aj platformu ESP32, ktorá je použitá v tejto diplomovej práci.

Mikrokontróler má WiFi a Bluetooth konektivitu, čo ho predurčuje na použitie v bezdrôtových aplikáciách senzorového uzla s možnosťou prenosu nameraných údajov dát cez WiFi rozhranie, ktoré môže byť použité aj na prevzatia vzdialenej aktualizácie spustiteľného firmvéru - aplikácie. Kapitola 1 opisuje základné parametre mikrokontroléru ESP32 vrátane vývojových kitov, kapitola 2 opisuje rozdelenie interných a externých pamätí vrátane ich mapovaní v adresnom priestore mikrokontroléru ESP32.

Nakoľko je platforma vhodná aj pre aplikácie s nízkym prúdovým odberom, opis tohto režimu prevádzky je v kapitole 3. ESP32 je možné programovať v rôznych vývojových prostrediach, ktoré sa líšia predovšetkým možnosťami a implementovanými vývojárskymi nástrojmi. Pre výber a návrh výslednej aplikácie som využil framework ESP-IDF z produkcie Espressif Systems v ktorom som vytvoril finálnu aplikáciu pre senzorový uzol s možnosťou vzdialenej aktualizácie firmvéru bezpečným prenosovým kanálom. Framework je s vývojárskymi nástrojmi opísaný v kapitole 4.

V kapitole 5 je opísané vývojové prostredie Arduino IDE v ktorom som overoval možnosti vzdialenej aktualizácie firmvéru, nakoniec som ho pre návrh finálnej aplikácie nevyužil, nakoľko podporované metódy vzdialenej aktualizácie firmvéru boli obmedzené iba pre použitie v LAN sieti. Samotné metódy vzdialenej aktualizácie firmvéru v jednotlivých vývojových prostrediach sú opísané v kapitole 6.

Prostredie ESP-IDF umožňuje využiť vývojárske nástroje a vstavané funkcionality, ktoré umožňujú zabezpečiť integritu prevzatého firmvéru firmvéru. Opisom digitálneho podpisu pre podpísanie firmvéru a následné overenie integritu firmvéru sa zaoberá podkapitola 6.3, ktorá obsahuje aj návod pre implementáciu v prostredí ESP-IDF.

Vstavané nástroje umožňujú zabezpečiť aj bootovací proces s využitím funkcie Secure Boot, ktorá umožní spúšťať iba Bootloader a firmvér od dôveryhodného vydavateľa. Kľúč, ktorý využíva pre výpočet odtlačku je bezpečne uložený v jednorázovo programovateľnej pamäti eFuse, ktorý útočník získať nemôže. Hardvérová funkcionalita Secure Boot je opísaná v podkapitole 6.4 s návodom na implementáciu v prostredí ESP-IDF.

Pre zvýšenie bezpečnosti je Secure Boot rozšírený o šifrovanie flash pamäte a v prípade útoku na flash pamäť tak dokáže útočník získať iba šifrovaný text, ktorý je pre neho nepoužiteľný a nespustiteľný na mikrokontrolérovej platforme ESP32. Opis šifrovania flash pamäte, algoritmu, ktorý využíva je v podkapitole 6.5, kde je dostupný aj návod pre jeho implementáciu v prostredí ESP-IDF.

Kapitola 7 opisuje realizáciu finálnej aplikácie a jej výsledky pre senzorový uzol na platforme ESP32 s využitým hardvérom, opisuje úpravy webového rozhrania na ktorom je dostupný firmvér pre vzdialenú aktualizáciu a sú tam vizualizované namerané údaje senzorového uzla.

1. ESP32

ESP32 je mikrokontróler [1] z produkcie čínskej firmy Espressif Systems. Je určený predovšetkým pre aplikácie internetu vecí a rôzne projekty vyžadujúce konektivitu s okolitými zariadeniami, internetom. Mikrokontróler 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 modulu. Použitie oboch technológii je univerzálne s kompatibilným hardvérom.

WiFi modem mikrokontroléru ESP32 podporuje štandardy IEEE (Inštitút elektrických a elektronických inžinierov) 802.11 b/g/n, čím dosahuje mikrokontróler teoretickú rýchlosť prenosu až 300 Mbit/s. Podporuje protokol ESP-NOW [2], 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) [3] 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 [4] čipu sa mikrokontróler líši veľkosťou externej 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.

Mikrokontróler je najčastejšie vybavený dvojjadrovým, pri istých verziách čipu jednojadrovým 32-bitovým procesorom Harvardskej architektúry Tensilica Xtensa L6 s nastaviteľným taktom 80 / 160 až 240 MHz. Architektúra má oddelenú pamäť pre dáta (údaje s ktorými mikrokontróler pracuje) a inštrukcie (riadiaci program pre mikrokontrolér).

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 [5] 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, udržuje 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 vývojových prostrediach s podporou kompilácie programu pre túto platformu, napríklad Arduino Core (prog. Jazyk Wiring), ESP-IDF (Espressif IoT Development Framework), Mongoose OS, MicroPython [6], Lua [7], Node.js [8], 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 [9]. Toho času sa vývoja Arduino Core ujali fanúšikovia, nadšenci Arduina a mikrokontrolérov.

* 1. Proces spustenia mikrokontroléru ESP32

Spustenie mikrokontroléru [10] ESP32 nastáva pri každom štartovacom cykle v dôsledku reštartu, pripojenia napájania. Proces pozostáva z viacerých fáz, ktoré na seba naväzujú a majú jasné poradie vykonávania:

1. Hardvérový Bootloader načíta do RAM pamäte softvérový Bootloader, ktorá sa nachádza na ofsete 0x1000 flash pamäte,
2. softvérový Bootloader načíta tabuľku partícii 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.

Proces spustenia mikrokontroléru ESP32 je bližšie opísaný blokovou schémou na Obr. 1. Úspešný proces bootovania závisí v prvom rade od detekcie externého úložiska – flash pamäte, kde je uložený softvérový Bootloader (Second-Stage Bootloader) a spustiteľný firmvér, ktorý tento Bootloader dokáže zaviesť do RAM pamäte a spustiť. Samotnú detekciu realizuje Hardvérový Bootloader (First-Stage Bootloader) uložený v ROM pamäti.



Obr. 1 Proces spustenia mikrokontroléru ESP32

**Hardvérový Bootloader**

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

Ak načítanie kódu z flash pamäte zlyhá, hardvérový Bootloader rozbalí Basic (TinyBasicPlus) interpreter (ROM Console) do RAM pamäte a spustí ho [11]. Nástroj slúži predovšetkým na debugovanie. Dokáže vykonávať základnú obsluhu GPIO (vstupno-výstupný vývod), dokáže obsluhovať pamäť, zapisovať a čítať z nej.

**Softvérový Bootloader**

Bootloader načíta tabuľku partícii na preddefinovanom ofsete (štandardne 0x8000). Nájde aplikačné partície (s podporou bootovania) a ak sa využíva tabuľka partícii 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 partícii využíva bez OTA definícii s jedným firmvérom, OTA\_DATA partícia nie je vytvorená a alokovaná, nakoľko existuje iba jedna bootovateľná partícia. Následuje zavedenie programu aplikácie Bootloaderom do RAM pamäte (IRAM (RAM pamäť inštrukciíí) a DRAM (RAM pamäť dát)).

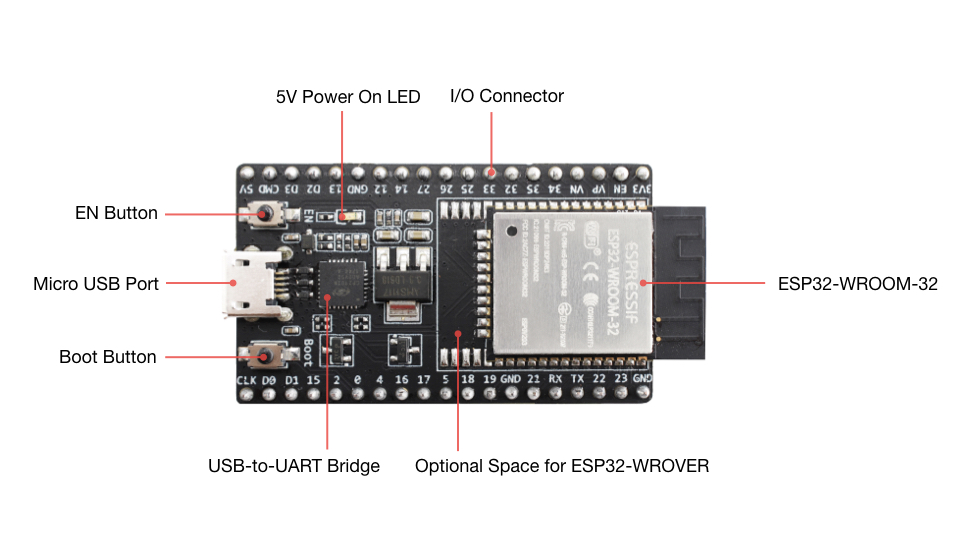
**Spustenie aplikácie**

Pri spustení aplikácie (firmvéru) sa spustí APP\_CPU (doteraz držaný v resete). 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 a následne je ukončená.

* 1. Vývojové kity, samostatný čip ESP32

Pre vývoj užívateľských projektov [12] je možné využiť samostatný čip ESP32 (Standalone), ktorý je možné programovať cez externý USB-UART (Prevodník USB signálov na UART rozhranie) prevodník, napríklad FTDI (Future Technology Devices International Limited) s vyvedenými vývodmi pre pripojenie k mikrokontroléru ESP32.

Taktiež existuje aj vývojový kit (nazývaný aj DevKit), ktorý je pri vývoji projektov viac obľúbený, nakoľko práca s ním je pohodlnejšia. Vývojový kit je osadený okrem čipu ESP32 aj USB-UART prevodníkom - najčastejšie CP2102 [13] od Silicon Labs pre možnosť napájania a programovania ESP32 cez USB (Univerzálna sériová zbernica) kábel. Najpopulárnejší DevKitC od Espressifu je možné nájsť na Obr. 2 aj s opisom základných hardvérových prvkov a komponentov, ktoré vývojový kit obsahuje.



Obr. 2 ESP32-DevKitC

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 periférii 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 určitý, alebo celý obsah flash pamäte. Vývojové kity sú najčastejšie osadené najlacnejším a najdostupnejším ESP32 čipom - ESP32-WROOM-32 so 4 MB externou flash pamäťou bez osadenej PSRAM (Pseudo Static RAM) na DevKit doske.

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 [14] 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, vyžaduje však aj zmenu fyzickej prepojky, ktorá volí plošnú anténu na DPS / externú anténu pre dosiahnutie optimálnej prevádzky a efektivity prenosu.

Čipom ESP32-S sú osadené rôzne vývojové kity od AI-Thinker, 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 (Internet of Things), 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.

Zjednodušená bloková schéma mikrokontroléru ESP32 od Espressif Systems je znázornená na Obr. 3. Schéma opisuje typy pamätí, zbernice, časovače, prerušenia, ktoré mikrokontróler ESP32 obsahuje, približuje možnosti komunikačných zberníc pre rôzne typy periférii s ktorými môže byť táto mikrokontrolérová platforma použitá.

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.



Obr. 3 Bloková schéma mikrokontroléru ESP32

1. Pamäť mikrokontroléru ESP32

Mikrokontróler ESP32 má rôzne typy pamätí, ktoré používa. Z pohľadu ich umiestnenia môžeme pamäte kategorizovať na interné a externé. Interná pamäť je súčasťou hlavného čipu Xtensa a poskytuje rýchlu komunikáciu s procesorom, externá pamäť je obsiahnutá v samostatnom čipe komunikujúcom po SPI zbernici.

Rýchlosť komunikácie a prenosu externých pamätí v porovnaní s internými je nižšia. Pamäte majú bohaté členenie aj z pohľadu ich logického rozdelenia [15] pamäťových oblastí. Na Obr. 4 je dostupná bloková schéma mapy pamäte, ktorá približuje jednotlivé typy externých a interných pamätí a ich ofsety v adresnom priestore, kde sú mapované na dátovej, alebo inštrukčnej zbernici.



Obr. 4 Mapa pamäte mikrokontroléru ESP32

Adresný priestor:

* 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,
* \* - Niektoré verzie ESP32 majú flash pamäť integrovanú (napr. ESP32-PICO-D4).

Externá pamäť:

* podpora až 16 MB SPI flash pamäte,
* podpora až 8 MB SPI SRAM pamäte.
  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, ktoré zodpovedajú za správne bootovanie systému a spustenie mikrokontroléru, obsahuje spodné vrstvy konektivity, obsluhu ich stacku, systémové funkcie, stub a iné...

Funkcia Stub môže spustiť kód uložený v ROM pamäti, ale aj spustiteľný kód uložený v RTC SLOW pamäti. Využíva sa napríklad v operáciách Deep Sleep (režim hlbokého spánku) hlavného procesora ESP32 pre výpis informácii na UART (Univerzálny asynchrónny prijímač a vysielač) rozhranie. Funkcia pre stub využíva v definícii funkcie v zdrojovom kóde makro „RTC\_IRAM\_ATTR“, aby bola funkcia dostupná v RTC SLOW pamäti a mohla byť obslúžená aj pri režime spánku hlavného procesora mikrokontroléru ESP32.

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.

Zostávajúca časť SRAM0 je alokovaná tiež na zbernici inštrukcií v rozsahu 0x4008\_0000 až 0x4009\_FFFF. Pamäť SRAM1 má veľkosť 128 kB, je k nej 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 pristúpiť 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). Priamy prístup k pamäti využíva celkom trinásť zberníc: UART0, UART1, UART2, SPI1 (Synchrónne sériové periférne rozhranie), SPI2, SPI3, I2S0, I2S1, SDIO, Slave SDMMC, EMAC, Bluetooth, WiFi. RTC FAST ja 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) koprocesora je táto pamäť vyhradená pre neho a slúži na uloženie jeho programu, odkiaľ ho spúšťa.

* 1. 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. Alokácia PSRAM pamäte začína od ofsetu 0x3F80\_0000 (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 partícii označuje ako app), tabuľku partícii, 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 mikrokontróler 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. Tabuľka partícii

Mikrokontróler ESP32 dokáže mať niekoľko na sebe nezávislých partícii uložených vo flash pamäti. Na preddefinovanom ofsete vo flash pamäte - štandardne 0x8000 je zapísaná tabuľka partícii [16], ktorá rozdeľuje priestor vo flash pamäti na logické celky, kde je každá partícia opísaná svoji názvom, typom (rozdeľujeme systémové, bootovateľné, dátové partície a iné druhy), začiatočným ofsetom a veľkosťou.. Tabuľka má pevnú veľkosť, do ktorej sa zmestí opis až 95 partícii.

Tabuľka partícii 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 partícii som využil aj v mojej diplomovej práci, nakoľko OTA aktualizáciu využívam. V prípade tohto typu s viacnásobnou bootovateľnom partíciou je do tabuľky pridaná aj partícia OTA\_DATA, ktorá opisuje príznak pre Bootloader (zavádzač), ktorý firmvér má prioritne bootovať.

Ak je OTA\_DATA prázdna, bootuje sa továrenská verzia z partície FACTORY, čo je firmvér, ktorý je možné nahrať iba prostredníctvom USB-UART rozhrania. Okrem firmvérových partícii sú v tabuľke partícii zapísané aj iné partície dátového a konfiguračného významu. Štandardná tabuľka partícii pre OTA aplikácie má celkovo tri bootovateľné partície označené ako aplikačné (APP).

**Tabuľka partícii je zapísaná v .csv formáte:**

# 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, 0x110000, 1M,   
ota\_1, app, ota\_1, 0x210000, 1M,

Tabuľku partícii je možné plne prispôsobiť pre potreby finálnej aplikácie. V prostredí ESP-IDF je možné tabuľku partícii importovať v .csv formáte s vlastnou štruktúrou logického rozdelenia flash pamäte. V procese kompilácie je tabuľka partícii skompilovaná do binárnej podoby. Každej partícii je možné priradiť aj značku (Flag), ktorým je možné nastaviť partícii systémovú funkcionalitu, napríklad šifrovanie jej obsahu na danom ofsete, ak sa šifrovanie flash pamäte používa.

Bootovateľné partície (typ app) sú šifrované vždy bez nutnosti nastaviť značku (Flag), ak je povolené šifrovanie flash pamäte. Na Obr. 5 je bloková schéma flash pamäte, kde je viditeľný ofset zápisu Bootloadera, tabuľky partícii a jednotlivých aplikačných a systémových partícii, ktoré tabuľka definuje vyššie v .csv formáte.



Obr. 5 Bloková schéma flash pamäte s ofsetmi a rozdelením jednotlivých partícii

* 1. eFuses

Mikrokontróler ESP32 obsahuje niekoľko interných pamäťových blokov – eFuses (elektronická poistka – jednorázovo programovateľná pamäť) [17]. 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, šifrovanie flash pamäte, Secure Boot, DEBUG).

Po zapísaní do systémových eFuse nie je možné jej obsah softvérovo prečítať, používa ich výhradné 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 takzvané systémové poistky. Blok eFuse BLK1 je úložiskom symetrického AES kľúča pre šifrovanie / dešifrovanie flash pamäte. Veľkosť tejto eFuse pamäte je 256 bitov.

Pamäťový blok eFuse BLK2 má rovnakú veľkosť 256 bitov. Je úložiskom pre šifrovací AES kľúč, ktorý sa používa pri výpočte odtlačku, ktorým je možné verifikovať softvérový Bootloader a zabezpečiť tak bootovací proces. Posledným eFuse pamätovým blokom je BLK3, ktorým je možné nastaviť špecifickú MAC adresu mikrokontroléru ESP32, prípadne môže byť použitá aj na používateľskú aplikáciu.

1. Nízkopríkonový režim mikrokontroléru ESP32

Nízka spotreba je jednou z priorít IoT aplikácii [18]. Tieto aplikácie nazývame aj ULP (Ultra-Low Power), teda aplikácie s ultra-nízkou spotrebou elektrickej energie, alebo s nízkopríkonovým režimom. 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, ktorý rozlišujeme na základe toho, ktoré periférie a hlavné časti mikrokontroléru sú vypnuté, neaktívne. Jednotlivé podporované režimy spánky mikrokontrolérom ESP32 sú opísané v Tab. 1. Pre opätovné zapnutie hlavné procesora, modemu WiFi / Bluetooth konektivity je nutné prebudiť hlavný procesor.

Tab. 1 Prevádzkové režimy mikrokontroléru ESP32 a stav periférii

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prevádzkový režim** | **Aktívny** | **Modem-sleep** | **Light-sleep** | **Deep-sleep** |
| **Hardvér** | - | - | - | - |
| Xtensa CPU | ZAP | ZAP | PAUSED | VYP |
| WiFi/Bluetooth modem | ZAP | VYP | VYP | VYP |
| RTC pamäť a periférie | ZAP | ZAP | ZAP | ZAP |
| ULP koprocesor | 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 kontróler má Timer (časovač), prostredníctvom ktorého dokáže zobudiť hlavný procesor mikrokontróleru 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 mikrokontróler reagovať čítaním analógovej hodnoty vývodu a pri prekročení rozhodovacieho prahu prebudí hlavný procesor Xtensa mikrokontroléru 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 [19] využíva 32-bitové inštrukcie s možnosťou využívania pamäte RTC SLOW, ktorá má 32- bitový adresný priestor. 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.

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. Vývojársky framework ESP-IDF

Framework pre vývoj IoT aplikácii [20] založený na jazyku C pre platformu ESP32. Obsahuje API (rozhranie pre programovanie aplikácii), zahŕňa knižnice a ukážkové zdrojové kódy - projekty pre základnú obsluhu zberníc, rozhraní a komponenty pre obsluhu konkrétnej aplikácie. Obsahuje toolchain pre kompiláciu programu s build systémom CMake a Ninja.

Cieľom frameworku je ponúknuť základné implementácie pre vývojárov IoT aplikácii s Bluetooth a WiFi konektivitou, ktoré sú bezpečné, poskytujú možnosť riadenia spotreby elektrickej energie zariadenia, ktorá je častokrát kľúčovým parametrom úspešnej (kritickej) aplikácie prevádzkovanej na batériu pre IoT aplikácie.

Súčasťou ESP-IDF aj jednoduché konzolové rozhranie, ktoré slúži pre výber cieľového projektu, jeho kompiláciu vstavanými nástrojmi, možnosť konfigurácie každého príkladu zvlášť, možnosť využitia vstavaných vývojárskych nástrojov pre prácu s flash pamäťou, pamäťou eFuses ESP32 a pre kryptografické operácie.

* 1. Vývojárske nástroje v ESP-IDF

Framework ESP-IDF má viacero vývojárskych nástrojov, ktoré sú spustiteľné Python scripty. Zároveň dokážu spracovať aj vstupné argumenty, ktoré sa zadávajú do konzolovej aplikácie ESP-IDF. Python script dokáže vstupný argument prečítať a spustiť funkciu, ktorá je týmto argumentom definovaná priamo v scripte, prípadne využíva import iných .py scriptov, ktoré majú danú funkciu definovanú a dokážu ju vykonať.

Základným nástrojom je idf.py zabezpečuje funkcie nástroja make, teda vykonáva príkazy súvisiace s kompiláciou programu zo zdrojov. Príkazom idf.py menuconfig umožňuje vyvolať konfiguračné menu – Menuconfig pre nastavenie aktuálne otvoreného projektu pre potreby vývojára. Dokáže iniciovať zápis skompilovaného programu (firmvéru) a ďalších súčastí do flash pamäte a obdobné operácie súvisiace s programom.

Nástroj esptool.py slúži na priamy zápis do flash pamäte (najčastejšie je spúšťaný nástrojom idf.py), dokáže z flash pamäte aj čítať jej obsah. Označenie ESPTOOL je zároveň aj označením pre balík vstavaných nástrojov do prostredia ESP-IDF (esptool.py, espsecure.py a espefuse.py).

Pre kryptografické operácie je do prostredia ESP-IDF vstavaný nástroj espsecure.py. Nástroj espefuse.py slúži pre prácu s jednorázovo programovateľnými pamäťami eFuses. Všetky nástroje je možné obsluhovať z príkazového riadku ESP-IDF. Nakoľko však nástroj espsecure.py a espefuse.py nemá podporu spustenia v celom systéme Windows ale iba z ich priečinka, kde sú umiestnené, je nutné balík nainštalovať s celosystémovou podporou, čo zaručí, že je možné nástroj spustiť kdekoľvek v operačnom systéme z príkazového riadku.

Do frameworku ESP-IDF sú implementované aj komponenty vďaka ktorým je možné konfigurovať špecifické požiadavky aplikácie, napríklad - využitie hardvérového kryptografického akcelerátora AES (Advanced Encryption Standard), využitie koprocesora pre aplikácie s nízkym prúdovým odberom, vyhradenie stacku pre hlavný task, umožnenie spätnej kompatibility s predchádzajúcimi verziami ESP-IDF (týka sa najmä deprecated funkcií v programových implementáciách so starším API), nastavenie ignorácie a obídenie bezpečnostných prvkov pre vývojárske účely a iné...

* 1. Vývoj frameworku ESP-IDF

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 3.3, v neskorších mesiacoch vývoja následne 4.0 a pre finálnu aplikáciu ESP-IDF vo verzii 4.2 [21], 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 pokým bude dostávať aktualizácie a budú v nej vyriešené chyby a bude dostávať bezpečnostné záplaty vývojármi Espressif Systems a komunitou vývojárov. Dĺžka podpory pre verzie 4.X je 30 mesiacov. Verzia ESP-IDF 4.2 má softvérovú podporu do Júna 2023. Vývoj aktuálnych verzií frameworku ESP-IDF a nasledujúcich sa orientuje aj na podporu nových platforiem ESP32-C3 a ESP32-C6, ktoré sú najnovším prírastkom do rodiny procesorov ESP32 a majú novšie funkcie, ktoré nájdu využitie predovšetkým v bezpečných IoT aplikáciách.

Hlavnou výhodou frameworku ESP-IDF v porovnaní s prostredím Arduino IDE je možnosť programovania bližšie k fyzickej vrstve zariadenia, čo umožňuje vyvíjať komplexnejšie programy. Využíva operačný systém reálneho času - FreeRTOS [22]. 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.

Každej 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 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 ofset.

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.

**Fragment kódu – Queue s čakaním tasku consumer na vykonanie tasku producer, výmena inkrementovanej hodnoty – dynamického počítadla so sekundovou inkrementáciou:**

QueueHandle\_t q=NULL;   
  
//TASK PRIJIMATELA (CAKA POKYM TASK ODOSIELATELA POSLE CISLO)  
void consumer\_task(void \*pvParameter) {   
unsigned long counter;   
if(q == NULL){   
printf("Queue nepripravene ");   
return;   
}   
while(1){ //prijem hodnoty z inter-task komunikacie  
//hodnota sa vypise az po prijati na UART monitor  
xQueueReceive(q,&counter,(TickType\_t )(1000/portTICK\_PERIOD\_MS));   
printf("Hodnota prijata cez queue: %lu \n",counter);   
vTaskDelay(500/portTICK\_PERIOD\_MS); //wait for 500 ms   
} }   
  
//TASK ODOSIELATELA (ODOSIELA CISLO KAZDU SEKUNDU)  
void producer\_task(void \*pvParameter){   
unsigned long counter=1;   
if(q == NULL){   
printf("Queue nepripravene \n");   
return;   
}   
while(1){ //odosielanie hodnot do intertask komunikacie  
//Hodnota sa na UART monitor vypise pri kazdom odoslani  
printf("Odoslana hodnota cez Queue: %lu \n",counter);   
xQueueSend(q,(void \*)&counter,(TickType\_t )0); // add the counter value to the queue  
counter++;   
vTaskDelay(1000/portTICK\_PERIOD\_MS); //wait for a second   
} }   
  
void app\_main() {   
q=xQueueCreate(20,sizeof(unsigned long)); //Vytvorenie Queue  
if(q != NULL){   
printf("Queue uspesne vytvorene\n");   
vTaskDelay(1000/portTICK\_PERIOD\_MS); //wait for a second  
xTaskCreate(&producer\_task,"producer\_task",2048,NULL,5,NULL);   
printf("Producer task spusteny\n");  
xTaskCreate(&consumer\_task,"consumer\_task",2048,NULL,5,NULL);   
printf("Consumer task spusteny\n");   
}  
else{ printf("Vytvorenie Queue zlyhalo");   
} }

1. Arduino Core

Implementácia podpory mikrokontrolérov ESP32 do prostredia Arduino IDE. Obsahuje nástroj esptool.py [23] 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 [24] je navrhnuté pre programovanie a vývoj blízko k aplikačnej vrstve mikrokontrolérov. Je tak vhodnou alternatívou pre používateľov a vývojárov s nízkou znalosťou programovania, umožňuje im vytvárať jednoduché programy aj pre IoT aplikácie aj bez rozsiahlejších znalostí programovania a programátorských prístupov.

Prostredie Arduino Core je obľúbené najmä pre ľahkú syntax jazyka Wiring. Pre náročnejších užívateľov je možné spúšťať aj konkrétne funkcie frameworku ESP-IDF a jeho komponenty. Neumožňuje však nastavenie špecifickej konfigurácie, vykonať voľbu oscilátora (interný / externý) pre časovače, prepínať APP a PRO CPU. Taktiež nemá podporu konzolovej aplikácie, či vstavaných vývojárskych nástrojov ako vo frameworku ESP-IDF, používateľ nedokáže obsluhovať a vytvoriť program pre ULP koprocesor v rozhraní Arduino IDE.

Funkcie Arduino Core sú značne obmedzené a ponúkajú iba možnosť využitia základných možností mikrokontroléru ESP32, ktoré sú však ďaleko rozsiahlejšie. Nedokáže tak využiť plný potenciál tejto mikrokontrolérovej platformy. 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 – úlohy vytvorené prostredníctvom plánovača nezávisle na sebe podobne ako vo frameworku ESP-IDF. Nakoľko jazyk Wiring nemá funkciu app\_main(), je nutné tasky vytvárať vo funkcii setup(), ktorá sa vykoná raz alebo v nekonečnej slučke loop(), ktorá je označená ako hlavný task – ekvivalent k spomenutej app\_main() funkcii.

Arduino Core je stále vo vývoji na Githube a ukážkové programové implementácie (projekty) sa vytvárajú dodatočne podľa už existujúcich programových implementácii vo frameworku ESP-IDF, ktorého API využíva.

1. Vzdialená aktualizácia firmvéru

OTA je termín opisujúci metódy distribúcie aktualizácii softvéru, konfigurácii a firmvéru pre zariadenia [25]. V praxi sa stretávame predovšetkým s takzvanými remote OTA aktualizáciami, ktoré sú distribuované prostredníctvom centrálneho 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, notifikáciou, prípadne dokážu aktualizáciu stiahnuť a nainštalovať bez nutnosti akcie používateľa, ktoré používateľa informujú o úspešnej aktualizácii softvéru, programu 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.

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.

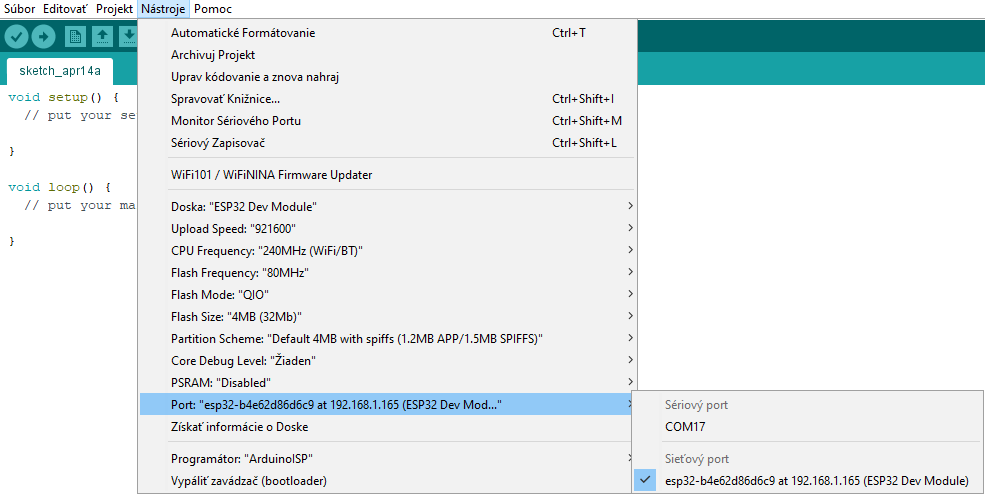
OTA aktualizácie našli uplatnenie aj pre zariadenia IoT (Internet vecí), ktoré je možné vzdialene aktualizovať cez internet, alebo z LAN siete na základe podporovanej OTA metódy bez nutnosti fyzického prístupu k zariadeniu [26]. Medzi IoT platformy podporujúce OTA aktualizácie sa radí aj mikrokontróler ESP32, ktorý som využil vo svojej diplomovej práci. Ako som spomenul v predchádzajúcej kapitole, mikrokontróler je možné programovať v rôznych vývojových prostrediach. „“

Prostredia sa líšia 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 partícii, ktorá definuje logické rozdelenie flash pamäte na viacero bootovateľných partícii.

* 1. Aktualizácia firmvéru v Arduino Core

Projekty ukážkových 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 nahrať firmvér. Port sa zobrazuje v prostredí Arduino IDE v časti portov, kde sú aj COM (Sériová linka) porty, ktoré slúžia na komunikáciu s mikrokontrolérom cez UART rozhranie a najmä pre účely UART monitoru. Pri sieťovom OTA porte nie je možné túto funkcionalitu využiť. Programová implementácia pre tento typ aktualizácie - Basic OTA [27] 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 vstavaného projektu Basic OTA.

Aby bolo možné vôbec aktualizácie vykonávať, prvý krát 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 nahrať aktualizovaný program do ESP32 cez OTA metódu, vyžaduje sa, aby mal používateľ nainštalovaný program Python minimálne vo verzii 3, ktorý riadi celý proces aktualizácie (nahrávania nového programu cez sieť) do ESP32 automatizovane. Na 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 programu 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 mikrokontróler 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.

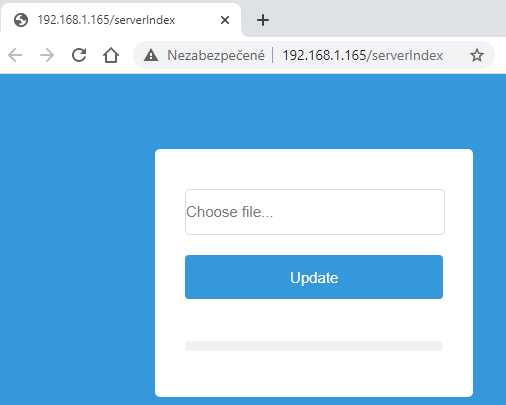
Nový program je možné po úspešnej aktualizácii nahrať priamo z vývojového prostredia Arduino IDE. Je nutné dodať, že nový program musí taktiež obsahovať OTA časť predchádzajúceho programu, ak chce používateľ do budúcna program opäť aktualizovať. V prípade nahratia nového firmvér cez OTA bez OTA implementácie sa nový firmvér nahrá, 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 príjme 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 otvorenom texte, alebo jeho hashu v MD5 [28] 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 kolízii, 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 projektu [29]. Tento typ OTA aktualizácie využíva ESP32 spustené v režime HTTP webservera, na ktorý je možné pristúpiť prostredníctvom prehliadača. Vzhľad a grafiku webovej stránky je možné štylizovať cez CSS (kaskádové) štýly. 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 domenové\_meno.local (post-fix .local je nutný pre použitie a rozpoznanie doménového mena v LAN sieti) pre jej správne načítanie v prehliadači. Pre úspešné rozpoznanie zariadenia v sieti je nutné používať na koncom zariadení DNS server v LAN sieti, nie DNS server poskytovateľa internetového pripojenia. Na Obr. 7 je vizualizované webové rozhranie OTA Web Updater, ktoré slúži pre aktualizáciu firmvéru.



Obr. 7 OTA Web Updater

Súčasťou webového rozhrania je HTML formulár, prostredníctvom ktorého je možné nahrať binárny (skompilovaný) súbor nového programu a tým aktualizovať softvér. Binárny súbor je možné vygenerovať v Arduino IDE a uložiť ho pre jeho neskoršie nahratie prostredníctvom HTML formulára. 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 príjme 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. Mikrokontróler po prevzatí firmvéru neoveruje jeho integritu. Potenciálne nebezpečnú aktualizáciu - firmvér tak spustí.

* 1. Aktualizácie firmvéru v ESP-IDF

Framework ESP-IDF obsahuje pokročilejšie implementácie OTA aktualizácii. Nájdeme tu iba vstavané projekty pre vzdialené OTA aktualizácie, ktoré sa realizujú spôsobom prebratia aktualizácie z externého zdroja - cloudu, webservera, ktorý je dosiahnuteľný cez internet. Mikrokontróler vykonáva požiadavku na preddefinované sieťové umiestnenie, kde je distribuovaný firmvér. Pripojenie je možné realizovať cez technológiu WiFi, alebo Ethernet (riadiaci čip LAN8720 [30] / IP101GRI [31]), ktorý je možné pripojiť k ESP32 cez RMII (Reduced media-independent interface) [32] rozhranie a používať ho s WiFi ovládačom ako PHY (fyzická vrstva) [33] perifériu.

Pre overenie možností aktualizácii som využil technológiu WiFi a projekty Simple OTA a Native OTA, ktoré sú v ESP-IDF dostupné. Oba projekty 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, 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. Využíva sa teda Chain of Trust – reťaz dôvernosti.

**Certifikát certifikačnej autority v .pem formáte, ktorý je vložený do programu v procese kompilácie:**

-----BEGIN CERTIFICATE-----  
MIIDTzCCAjcCFGoeGOBevdCzZX61CSkNsw1yLOgeMA0GCSqGSIb3DQEBCwUAMGQx  
CzAJBgNVBAYTAlVTMRMwEQYDVQQIDApGYWtlIFN0YXRlMRYwFAYDVQQHDA1GYWtl  
IExvY2FsaXR5MRUwEwYDVQQKDAxGYWtlIENvbXBhbnkxETAPBgNVBAMMCGVzcDMy  
LnNrMB4XDTE4MTIxNDE4MTcyNVoXDTIzMTIxMzE4MTcyNVowZDELMAkGA1UEBhMC  
VVMxEzARBgNVBAgMCkZha2UgU3RhdGUxFjAUBgNVBAcMDUZha2UgTG9jYWxpdHkx  
FTATBgNVBAoMDEZha2UgQ29tcGFueTERMA8GA1UEAwwIZXNwMzIuc2swggEiMA0G  
CSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDeimA7p/eIaGtXPM2QyD3+tDI+6ltB  
bPzHUPupBK3EYi1bCtHGg0Lq8cbCdFVhJJnvRGoIdz/knhTIkMphMvTC5UJLooPx  
RnI9KSdXWs2n/wXYvPS9Txthz6zv0Ua9Dp+OcXvz0wzG1skhd+RPAm9zUJDJ42P1  
FIjcehxH93GgrHzXz6zklFt01/8IAcBt7gl8/qBnjJNA8SS0BoYgfRnNOvn6k1yk  
ZZtyfIp68NWlfdveMd8Ig2pUIuvTh7MyUFjumcDLqAs2J+SenUYHEcfZJK4Vmdfo  
bZam4hx8f0BGrxqDhDMuTmh/Jqa8ZYANfR7ctD1DU6u7aceGYmSgCiNpAgMBAAEw  
DQYJKoZIhvcNAQELBQADggEBACA5n68AsjspvR3N33jF83FZqDXuvQbmY7lBm2nN  
Yh2265f9mRGrOO3z6lHKUfs5BJXxxahAKPQ07H9zQuYvIW0y8Jcgs4FlFhAuqgss  
KWuqJicf/noRPq/MC7mTLSGVrCo3R3x8U842j/Jdi0Ygb7AgFMp5F7DoS88QEMuK  
2UbJSA0czc24NImg9Y8Ve6VS4m+lsjsl77L15JCuDeZ7UBegmYoDmEIC/MohSD5t  
goX0+fCH4EsIkd6Zh8P6GTlRCglyGrSFgKTV/6HPSptAFvMFUofnfedCYJXrkjKc  
psVaEdVBIz/a4jcB6mkQQm06+KusHmwsc/VNYLD6+pjmIE0=  
-----END CERTIFICATE-----

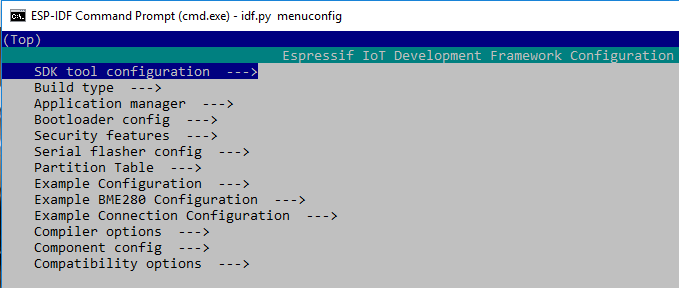
Pri realizácii mojej bakalárskej práce som túto problematiku riešil a certifikát pre webserver som generoval a podpísal vlastnou certifikačnou autoritou. Certifikačná autorita je stále platná a rovnako tak i certifikát webservera, nakoľko bol generovaný na obdobie piatich rokov - do roku 2023**.** Využil som certifikáty generované algoritmom RSA s dĺžkou kľúča 2048-bitov, nakoľko kratšie kľúče nie sú podporované na strane mikrokontroléru ESP32. Certifikáty je možné generovať aj metódou ECC (eliptických kriviek) s využitím krivky prime256v1 s dĺžkou kľúča 256 bitov.

Obe metódy so spustiteľnými .bat súbormi pre automatizované vygenerovanie certifikátov kryptografickým nástrojom OpenSSL a inštrukciami a rozšíreným popisom je súčasťou prílohy A rovnako aj s konfiguračným súborom pre webserver, kde je nutné doplniť cestu k certifikátu.

**Generovanie certifikátu certifikačnej autority, webservera nástrojom OpenSSL:**

rem CA cert   
openssl genrsa -out myCA.key 2048   
openssl req -x509 -config certificate-authority-options.conf -new -nodes -key myCA.key -sha256 -days 1825 -out myCA.pem   
openssl x509 -outform pem -in myCA.pem -out myCA.crt   
  
rem server cert   
openssl genrsa -out server.key 2048   
openssl req -config options.conf -new -key server.key -out server.csr   
openssl x509 -req -in server.csr -CA myCA.pem -CAkey myCA.key -CAcreateserial -out server.pem -days 1825 -sha256 -extfile server.ext   
openssl x509 -outform pem -in server.pem -out server.crt

Certifikát certifikačnej autority v .pem (čitateľom) formáte som do oboch projektov vložil do predpripraveného - prázdneho súboru „ca\_cert.pem“ v priečinku server\_certs. Certifikát je vložený do aplikácie v procese kompilácie. Konfigurácia oboch základných projektov s OTA implementáciou sa realizuje cez konfiguračné menu - takzvaný Menuconfig, ktorý je možné vyvolať priamo z konzolovej aplikácie ESP-IDF. Na Obr. 8 je Menuconfig otvorený v projekte Native OTA frameworku ESP-IDF. Na sú viditeľné tri používateľské konfiguračné podmenu pre projekt - „Example Configuration“, „Example BME280 Configuration“, „Example Connection Configuration“.



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 – používateľského projektu. Ak ide o projekt 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.

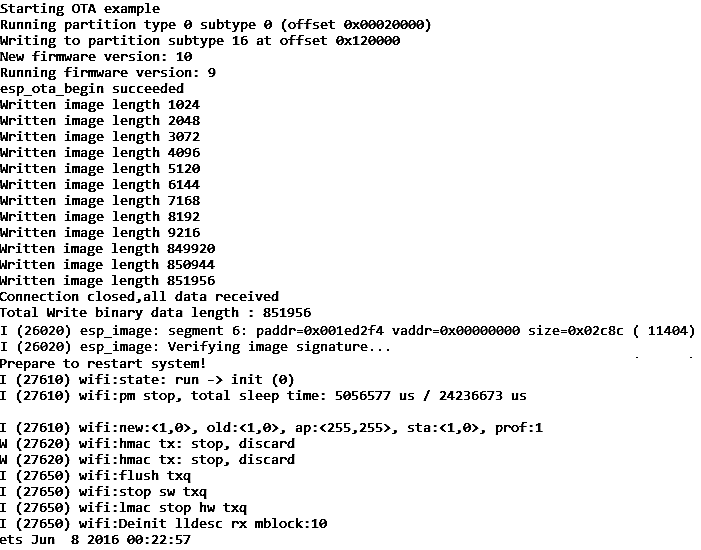
Projekty Simple aj Native OTA ponúkajú vytvorené menu vo svojich zložkách projektov s názvom „Example Connection Configuration“ a „Example Configuration“ zadať SSID a heslo WiFi siete štandardu WPA / WPA2 - PSK a taktiež aj adresu domény (prípadne IP adresu) webservera a absolútnu cestu k firmvéru odkiaľ jeho možné prevziať. Následne je možné program v konzolovej aplikácii frameworku ESP-IDF skompilovať a nahrať 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 mikrokontróler ESP32 prebrať. Danú cestu som mu nastavil aj v konfiguračnom menu projektu „Example Configuration“.

Projekty 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\_0 alebo OTA\_1 (do tej, ktorej firmvér aktuálne nie je bežiaci a používaný). Projekt Simple OTA [34] 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 projektu Native OTA [35] je obsahovo robustnejšia a obsahuje rôzne mechanizmy pracujúce s aktualizáciou, ktoré rozhodujú o jej bootovaní a reštarte mikrokontroléru. Po zápise firmvéru do partície OTA\_0 alebo OTA\_1 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. Na Obr. 9 je záznam UART monitoru pri úspešnej aktualizácii firmvéru.

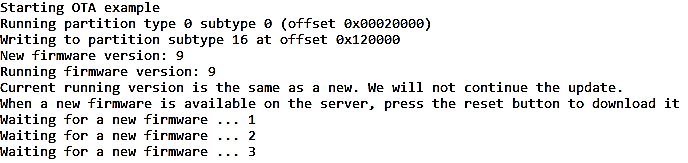
Spustený firmvér z partície FACTORY z ofsetu 0x20000 projektu Native OTA vo verzii 9 deteguje nový firmvér s verziou 10 pri prevzatí aktualizácie, ktorý je zapísaný na ofset 0x120000 – partícia OTA\_0, nastáva prepis OTA\_DATA partície pre prioritné bootovanie nového firmvéru a vykoná sa softvérový reštart pre zavedenie nového firmvéru do RAM pamäte Bootloaderom.



Obr. 9 Úspešná aktualizácia firmvéru, reštart systému

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 projektu 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 stiahnutého a aktuálne bežiaceho firmvéru identické, mikrokontróler čaká v nekonečnej slučke s inkrementovacím počítadlom a výzvou na reštart mikrokontróleru cez EN (RESET) tlačidlo každú sekundu. Tento stav je na Obr. 10 – výstup z UART monitora.



Obr. 10 Stiahnutie identického firmvéru, čakanie v slučke na reštart

Samotná aktualizácia pozostáva z viacerých krokov, ktoré sú na blokovej schéme Obr. 11. Schéma obsahuje aj popisy jednotlivých krokov aktualizácie s opisom logiky, ktorá sa pri rozhodovaní o prepnutí príznaku pre OTA\_DATA partíciu vykonáva.



Obr. 11 ESP32 - Native OTA – proces aktualizácie

Tento typ projektu je šetrnejší k flash pamäti a obsahuje základné mechanizmy pre overenie verzie firmvéru. Nakoľko však ani jeden z projektov 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 nahratý 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. Mikrokontróler ESP32 je možné zabezpečiť rôznymi spôsobmi, ktoré prostredie ESP-IDF umožňuje na úrovni hardvéru mikrokontroléru ESP32. Jednotlivé metódy sú opísané v Tab. 2 a v následujúcich kapitolách vysvetlené aj s ich implementáciou v prostredí ESP-IDF pre platformu ESP32.

Tab. 2 Metódy zabezpečenia mikrokontroléru

|  |  |
| --- | --- |
| **Metóda** | **Popis** |
| Digitálny podpis | Umožňuje overiť integritu (nepozmenenosť) podpísaného firmvéru. Využíva súkromný kľúč pre podpísanie a verejný pre overenie digitálneho podpisu. |
| Secure Boot | Umožňuje overiť obraz softvérového Bootloadera a zamedziť tak spusteniu nedôveryhodného Bootloadera, firmvéru. Implementuje aj digitálny podpis. Pre overenie obrazu Bootloadera využíva AES symetrický kľúč, ktorý sa používa iba pre operáciu šifrovania. |
| Šifrovanie flash pamäte | Umožňuje šifrovať dôležité partície a miesta vo flash pamäti mikrokontroléru ESP32 a v reálnom čase ich pri spúšťaní dešifrovať. Využíva symetrický AES kľúč pre obe operácie. |

* 1. 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 [36]. Digitálny podpis vychádza z asymetrickej kryptografie, ktorá používa dvojicu kľúčov - súkromný a verejný.

Súkromný kľúč je použitý pre podpísanie firmvéru, ktorý je distribuovaný cez OTA službu - server. Pre prenos firmvéru sa využíva bezpečný prenosový kanál – HTTPS protokol. Proces podpísania firmvéru súkromným kľúčom vydavateľa (podpisujúceho) je blokovou schémou znázornený na Obr. 12.



Obr. 12 Podpísanie firmvéru dôveryhodným vydavateľom

Verejný kľúč je použitý pre overenie digitálneho podpisu. V procese kompilácie je vložený do softvérového Bootloadera, ktorý realizuje toto overenie pri bootovaní firmvéru s možnosťou nastavenia tohto overenia aj v procese samotnej aktualizácie, kedy je firmvér zapísaný na dostupnú bootovateľnú (APP) partíciu. Proces overenia digitálneho podpisu je opísaný blokovou schémou na  Obr. 13.



Obr. 13 Overenie digitálneho podpisu verejným kľúčom podpisujúceho

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, 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. Pre samotný digitálny podpis sa využíva podpisová schéma ECDSA.

* + 1. Digitálny podpis – implementácia v prostredí ESP-IDF

Pre kryptografické operácie je možné využiť nástroj espsecure.py [37], 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). Generovanie sa vykonáva python funkciami z knižnice ECDSA [38]. Nástroj OpenSSL je možné taktiež využiť v procese generovania súkromného kľúča, ktorého dĺžka je 256 bitov.

**Fragment .py scriptu pre generovanie súkromného kľúča:**

def generate\_signing\_key(args):  
if os.path.exists(args.keyfile):   
raise esptool.FatalError("ERROR: Key file %s already exists" % args.keyfile)   
if args.version == "1":   
sk = ecdsa.SigningKey.generate(curve=ecdsa.NIST256p)   
with open(args.keyfile, "wb") as f:   
f.write(sk.to\_pem())   
print("ECDSA NIST256p private key in PEM format written to %s" % args.keyfile)

**Výstup .py scriptu 🡪 private.pem – súkromný kľúč**

-----BEGIN EC PRIVATE KEY----- MHcCAQEEIARIx9L3giT0FjCKy746uZlf1iCt4kwEoxTzyT4gJVxEoAoGCCqGSM49 AwEHoUQDQgAEYE6lFHaI0/ncxx4MXqDTIhoyVG0LAe+lN2FQhXdKnU5Y8VVOzNuS I/c1xRf3q4ZTYL4fHXBggZWVrzJQjUKKNw==   
-----END EC PRIVATE KEY-----

Eliptická krivka NIST256p je definovaná (opísaná) nad konečným (Galoisovým) poľom vzťahom:

**E(Gp): y2 = x3 + ax + b (mod p)**

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, pri akcii „ON\_UPDATE“. Táto akcia je tiež dostupná v predmetnom menu s možnosťou zapnutia.

**Fragment .py scriptu pre extrakciu verejného kľúča zo súkromného kľúča:**

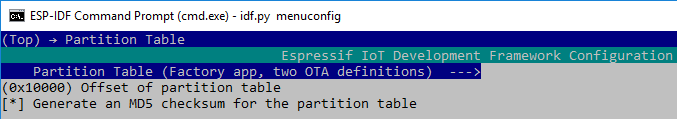
def extract\_public\_key(args):   
\_check\_output\_is\_not\_input(args.keyfile, args.public\_keyfile)   
if args.version == "1":   
""" Load an ECDSA private key and extract the embedded public key as raw binary data. """   
sk = \_load\_ecdsa\_signing\_key(args.keyfile)   
vk = sk.get\_verifying\_key()   
args.public\_keyfile.write(vk.to\_string())   
print("%s public key extracted to %s" % (args.keyfile.name, args.public\_keyfile.name))

Ďalším parametrom, ktorý je nutné nastaviť je relatívna cesta (vzhľadom na koreňový priečinok projektu) 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 ofsetu tabuľky partícii.

Preto je nevyhnutné cez Menuconfig v časti „Partition Table“ zmeniť parameter - začiatočný ofset z 0x8000 na 0x10000. Po posunutí začiatočného ofsetu tabuľky partícii sa posunú aj všetky ostatné partície. V tabuľke Tab.3 je možné vidieť posun ofsetu u všetkých aplikáčných partícii. Na Obr. 14 je viditeľná konfigurácia ofsetu tabuľky partícii z pôvodnej hodnoty 0x8000 na 0x10000.

Tab. 3 Rozdiel ofsetu aplikačných partícii po a pred posunutím

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Názov partície** | **Typ** | **Podtyp** | **Aktuálny ofset** | **Pôvodný ofset** | **Veľkosť** |
| FACTORY | APP | FACTORY | 0x20000 | 0x10000 | 1MB |
| OTA\_0 | APP | OTA\_0 | 0x120000 | 0x110000 | 1MB |
| OTA\_1 | APP | OTA\_1 | 0x220000 | 0x210000 | 1MB |



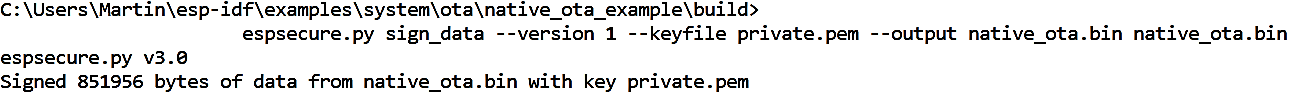
Obr. 14 Nastavenie ofsetu tabuľky partícii v Menuconfig

Podpísanie firmvéru je nutné realizovať manuálne po kompilácii firmvéru pred nahrávaním do dosky cez USB-UART rozhranie, alebo pred vložením firmvéru na OTA webserver. Pre podpísanie firmvéru [39] 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.

**Fragment .py scriptu, ktorý realizuje podpísanie firmvéru súkromným kľúčom:**

def sign\_data(args): \_  
check\_output\_is\_not\_input(args.keyfile, args.output) \_  
check\_output\_is\_not\_input(args.datafile, args.output)   
if args.version == '1': return sign\_secure\_boot\_v1(args)   
  
def sign\_secure\_boot\_v1(args):   
""" Sign a data file with a ECDSA private key, append binary signature to file contents """   
if len(args.keyfile) > 1:   
raise esptool.FatalError("Secure Boot V1 only supports one signing key")   
sk = \_load\_ecdsa\_signing\_key(args.keyfile[0])   
  
# calculate signature of binary data   
binary\_content = args.datafile.read()   
signature = sk.sign\_deterministic(binary\_content, hashlib.sha256)   
  
# back-verify signature   
vk = sk.get\_verifying\_key()   
vk.verify(signature, binary\_content, hashlib.sha256)   
  
# throws exception on failure  
if args.output is None or os.path.abspath(args.output) == os.path.abspath(args.datafile.name):   
# append signature to input file   
args.datafile.close()   
outfile = open(args.datafile.name, "ab")   
else:   
# write file & signature to new file   
outfile = open(args.output, "wb")   
outfile.write(binary\_content)   
outfile.write(struct.pack("I", 0))   
  
# Version indicator, allow for different curves/formats later   
outfile.write(signature)   
outfile.close()   
print("Signed %d bytes of data from %s with key %s" % (len(binary\_content), args.datafile.name, args.keyfile[0].name))

V procese podpisovania firmvéru sa z pôvodného súboru „native\_ota.bin“ vypočíta 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. 15 je dostupná vizualizácia podpísania firmvéru súkromným kľúčom v konzolovej aplikácii frameworku ESP-IDF s kryptografickým nástrojom espsecure.py.

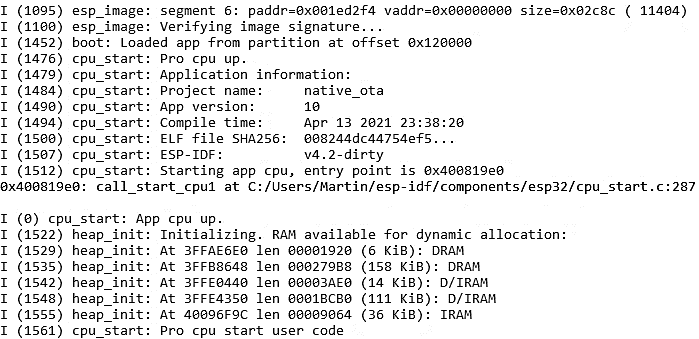


Obr. 15 Podpísanie firmvéru súkromným kľúčom v konzolovej aplikácii ESP-IDF

Tento firmvér je možné nahrať 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 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. Verejným kľúčom podpisujúceho, ktorý pozná 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.

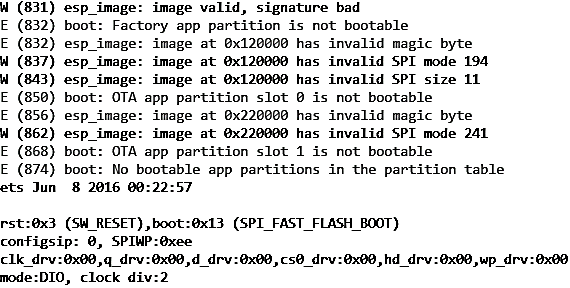
Firmvér je dôveryhodný a je ho možné spustiť bezpečne na mikrokontroléri ESP32. Úspešnosť samotného overenia spustí následne funkciu z programu Native OTA, ktorá overuje, či je verzia bežiaceho firmvéru iná, ako verzia stiahnutého. V prípade, že je digitálny podpis pri stiahnutom firmvéri overený, API 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. Úspešné overenie digitálneho podpisu firmvéru v procese bootovania je na Obr. 16 – výstup UART monitora.



Obr. 16 Úspešné overenie digitálneho podpisu firmvéru pri bootovaní, spustenie

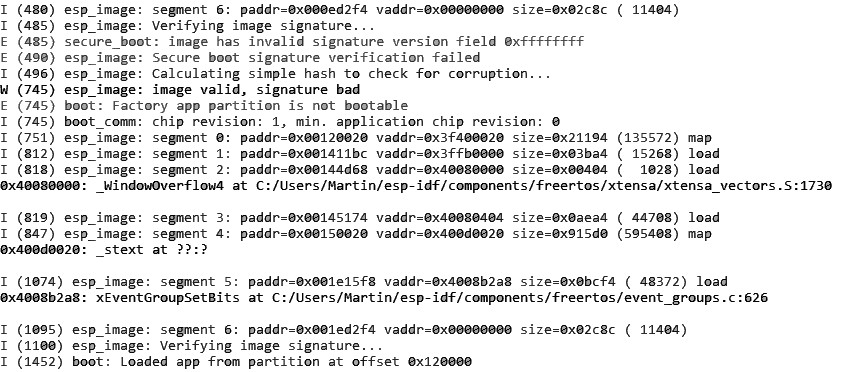
Ak firmvér neobsahuje digitálny podpis, alebo nie je platný, alebo hash firmvéru (z dôvodu jeho pozmenenia treťou osobou) nie je zhodný z dešifrovaným hashom, Native OTA sa k overeniu verzie firmvéru nedostane a aktualizácia je ignorovaná. Tento typ overenia sa realizuje pri každom bootovaní firmvéru, 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. Ak sa nepodarí spustiť firmvér ani jednej z dostupných partícii, bootovací proces sa ukončí a mikrokontróler nenabootuje firmvér, vykonáva cyklický reštart a opakuje pokus o bootovanie. Na Obr. 17 je prípad nenabootovania firmvéru ani z jednej dostupnej aplikačnej partície.



Obr. 17 Nenabootovaný firmvér ani z jednej aplikačnej partície

Prípad, kedy nie je overený digitálny podpis firmvéru z príznakovej partície – FACTORY (ofset 0x20000) je ukázaný na výpise UART monitora na Obr. 18. Bootloader následne bootuje firmvér z OTA\_0 partície (ofset 0x120000), kde digitálny podpis overí a zavedie firmvér do RAM pamäte. Existujúci príznak v OTA\_DATA partícii nie je prepísaný. Po reštarte mikrokontroléru ESP32 bude v procese bootovania prioritne bootovaný ako prvý firmvér znova z partície FACTORY, kde digitálny podpis overený nebol.



Obr. 18 Neúspešné overenie dig. podpisu jednej z partícii

* 1. Secure Boot v ESP-IDF

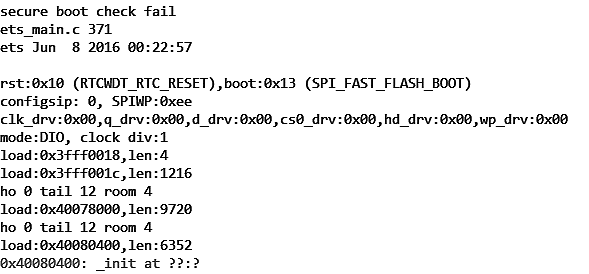
Metóda zabezpečenia Bootloadera a bootovacieho procesu [40]. Secure Boot kombinuje metódu digitálneho podpisu firmvéru a jeho overenia softvérovým Bootloaderom obsiahnutom vo flash pamäti v procese bootovania a priamo v procese aktualizácie pre akciu „ON\_UPDATE“. Zároveň je Secure Boot rozšírený o funkcionalitu overenia softvérového Bootloadera na začiatku bootovacieho procesu, ktorú obsluhuje hardvérový Bootloader (First-Stage Bootloader) uložený v ROM pamäti.

V princípe ide o porovnanie referenčného odtlačku a aktuálne vypočítaného odtlačku obrazu softvérového Bootloadera, ktorý je zapísaný vo flash pamäti na preddefinovaný ofset 0x1000 s využitím AES šifrovacieho kľúča s dĺžkou 256 bitov. Kľúč je symetrický, využíva sa však iba na proces šifrovania. AES kľúč je uložený v jednorázovo programovateľnej pamäti eFuse BLK2 s veľkosťou 256 bitov, odkiaľ ho nie je možné softvérovo prečítať / prepísať. Cieľom výpočtu odtlačku a následného overenia je kontrola, že je softvérový Bootloader vo flash pamäti dôveryhodný a zhoduje sa s odtlačkom, ktorý je ako referenčný uložený vo flash pamäti na preddefinovanom ofsete 0x0.

Zaručuje tak, že Bootloader uložený vo flash pamäti je od dôveryhodného vydavateľa, ktorý dokázal pre tento Bootloader vytvoriť aj odtlačok s využitím referenčného AES šifrovacieho kľúča (rovnaký je zapísaný v eFuse BLK2), ktorý má uschovaný. Ak by Secure Boot funkcionalita nebola implementovaná, mohlo by mať za následok spúšťanie neovereného firmvéru, ktorý by mohol podvrhnúť útočník, spoločne s upraveným Bootloaderom, ktorý by ignoroval digitálny podpis firmvéru.

Secure Boot je možné zapnúť cez konfiguračné menu - Menuconfig v podmenu „Security Features“. Neúspešné overenie Bootloadera môže byť spôsobené aj chýbajúcim odtlačkom na očakávanom ofsete 0x0 vo flash pamäti. Celý proces Secure Boot využíva Chain of Trust – reťaz dôvernosti. Pokým nie je overený softvérový Bootloader, nevykoná sa bootovanie firmvéru, ani overenie jeho digitálneho podpisu.

Neúspešné overenie softvérového Bootloadera je ukázané výstupe UART monitora na Obr. 19. Mikrokontróler ESP32 sa reštartuje v nekonečnej slučke a vypisuje cyklický výpis o neúspešnom overení Bootloadera. Nevykoná sa pokus o bootovanie firmvéru z dostupných partícii, nakoľko softvérový Bootloader nebol overený hardvérovým Bootloaderom, nie je splnený predpoklad pre bootovanie firmvéru.

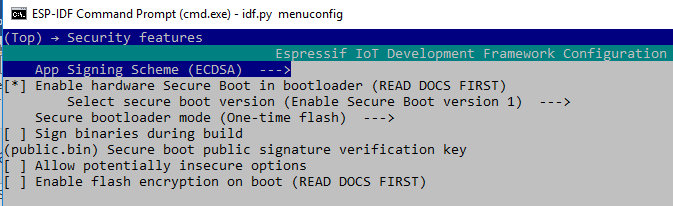


Obr. 19 Secure Boot - neoverený softvérový Bootloader, zakázaná fáza bootovania

Z pohľadu prevádzky Secure Bootu je možné zvoliť si jednu z jeho dvoch implementácii – „Re-flashable“, ktorá je vhodná pre vývoj aplikácii a umožňuje nahrávať AES šifrovací kľúč pre Secure Boot viackrát do určitej časti flash pamäte. Tento režim nie je vhodný pre produkčné - finálne aplikácie, nakoľko útočník dokáže získať kľúč po prečítaní flash pamäte, nakoľko nie je chránený v eFuse, kde môže ku kľúču pristupovať iba hardvér – Bootloader uložený v ROM pamäti.

Druhou implementáciou Secure Bootu, ktorú je možné použiť je „One-Time Flash“, ktorý je vhodný pre produkčné aplikácie s jednorazovým zápisom Secure Boot AES šifrovacieho kľúča do eFuse BLK2. Zároveň je použitie tejto funkcionality permanentné bez možnosti jej vypnutia v budúcnosti. V ďalejšej podkapitole je vysvetlená implementácia v prostredí ESP-IDF práve pre „One-Time Flash“ implementáciu Secure Boot.

Na Obr. 20 je základné nastavenie podmenu „Security Features“ v Menuconfigu pre produkčnú verziu Secure Boot s využitím „One-Time Flash“. Ako je z konfiguračného menu zrejmé, firmvér nie je v procese kompilácie podpísaný, nakoľko je možné zvoliť možnosť automatizovaného vloženia verejného kľúča do Bootloadera v procese kompilácie, alebo podpísanie firmvéru. Z toho dôvodu sa podpísanie firmvéru realizuje manuálne v konzolovej aplikácii prostredia ESP-IDF s využitím Python nástroja espsecure.py.



Obr. 20 Nastavenie Secure Bootu a verejného kľúča pre Bootloader v Menuconfigu

Zapnutím tejto funkcionality cez Menuconfig v časti „Security Features“ však ešte nie je Secure Boot plne aktívny, nakoľko sa musí zapnúť manuálne zápisom AES šifrovacieho kľúča do eFuse BLK2 a zároveň je nutné zapísať potvrdzovací bit do 1-bitovej eFuse ABS\_DONE\_0, ktorá permanentne zapne Secure Boot.

Využil som Secure Boot V1, ktorého funkčnosť je opísaná v tejto kapitole. Hardvérová funkcionalita Secure Boot existuje aj vo verzii V2, ktorá je však viazaná na modernejšie ESP32 platformy modelov z produkcie posledných rokov (napr. ESP32-S2, ESP32-C3, ESP32-C6), ktoré využívajú eFuse ABS\_DONE\_1 pre spustenie Secure Boot V2. Použitý mikrokontróler ESP32-WROOM-32 Secure Boot druhej verzie nepodporuje, aj keď má predmetnú eFuse k dispozícii.

* + 1. Secure Boot - implementácia v prostredí ESP-IDF

Šifrovací kľúč AES pre Secure boot je nutné vygenerovať. Pre generovanie kľúča je možné využiť viackrát používaný vstavaný kryptografický nástroj espsecure.py. Príkazom espsecure.py generate\_ generate flash\_encryption\_key secure-bootloader-key-256.bin som vygeneroval 256-bitový kľúč z rôznych faktorov operačného systému (python funkcia *os.random()*), ktoré zvyšujú entropiu - náhodnosť kľúča.

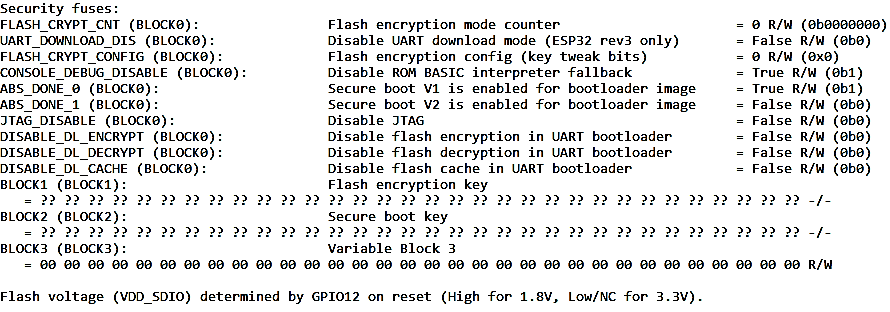
**Fragment .py scriptu pre generovanie kľúča pre Secure Boot:**

def generate\_flash\_encryption\_key(args):   
print("Writing %d random bits to key file %s" % (args.keylen, args.key\_file.name)) args.key\_file.write(os.urandom(args.keylen // 8))

Takto vygenerovaný kľúč je potrebné zapísať do jednorázovo programovateľnej pamäte eFuse BLK2 s veľkosťou 256 bitov, ktorá je pre tento kľúč určená. Pre prácu s eFuses existuje v ESP-IDF nástroj espefuse.py [41], 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 AES šifrovací kľúč pre funkcionalitu Secure Boot. Na Obr. 21 je sumár eFuses mikrokontroléru ESP32 zobrazených cez nástroj espefuse.py s využitím príkazu espefuse.py summary. Príkaz môže vyžadovať doplnenie COM portu, kde je mikrokontróler ESP32 detegovaný. Do existujúceho príkazu sa doplní napríklad -p COM20.

Na predmetnom obrázku je viditeľné, že v bloku eFuse BLK1 a BLK2 je zapísaný kľúč, ktorý nie je možné softvérovo prepísať / prečítať (nie je možnosť R/W pre dané bloky – viditeľné v pravej časti atribútov), teda sa hodnota týchto blokov nezobrazí v tomto softvérovom nástroji pre hodnoty, ktoré do eFuses už boli zapísané.



Obr. 21 Sumár eFuses zobrazených cez nástroj espefuse.py

Nástrojom espefuse.py som AES šifrovací kľúč zapísal do eFuse BLK2 s využitím príkazu espefuse.py burn\_key secure\_boot secure-bootloader-key-256.bin. Každý príkaz zápisu do eFuse je nutné potvrdiť zápisom hodnoty (textu) „BURN“ do konzolovej aplikácie prostredia ESP-IDF pre vykonanie jednorázového zápisu. Od tohto momentu je eFuse BLK2 chránená pred softvérovým zápisom a čítaním.

Pre permanentné zapnutie Secure Bootu, je nutné zapísať aj potvrdzovací bit do eFuse ABS\_DONE\_0. Zápis som realizoval príkazom espefuse.py burn\_efuse ABS\_DONE\_0.

**Fragment .py scriptu pre funkciu burn\_key, ktorá umožňuje zápis kľúča do eFuse:**

def burn\_key(esp, efuses, args):   
datafile\_list = args.keyfile[0:len([keyfile for keyfile in args.keyfile if keyfile is not None]):]  
block\_name\_list = args.block[0:len([block for block in args.block if block is not None]):]  
efuses.force\_write\_always = args.force\_write\_always   
no\_protect\_key = args.no\_protect\_key   
util.check\_duplicate\_name\_in\_list(block\_name\_list)   
if len(block\_name\_list) != len(datafile\_list):   
raise esptool.FatalError("The number of blocks (%d) and datafile (%d) should be the same." % (len(block\_name\_list), len(datafile\_list)))   
print("Burn keys to blocks:")   
for block\_name, datafile in zip(block\_name\_list, datafile\_list):   
efuse = None for block in efuses.blocks:   
if block\_name == block.name or block\_name in block.alias:   
efuse = efuses[block.name]   
  
if efuse is None:   
raise esptool.FatalError("Unknown block name - %s" % (block\_name))   
num\_bytes = efuse.bit\_len // 8   
data = datafile.read()   
revers\_msg = None   
  
if block\_name in ("flash\_encryption", "secure\_boot\_v1"):   
revers\_msg = "\tReversing the byte order"   
data = data[::-1]   
print(" - %s -> [%s]" % (efuse.name, util.hexify(data, " ")))   
  
if revers\_msg:   
print(revers\_msg)   
if len(data) != num\_bytes:   
raise esptool.FatalError("Incorrect key file size %d.   
Key file must be %d bytes (%d bits) of raw binary key data." % (len(data), num\_bytes, num\_bytes \* 8)) efuse.save(data)   
  
if block\_name in ("flash\_encryption", "secure\_boot\_v1"):   
if not no\_protect\_key:   
print("\tDisabling read to key block")   
efuse.disable\_read()   
if not no\_protect\_key:   
print("\tDisabling write to key block")   
efuse.disable\_write()   
  
if args.no\_protect\_key:   
print("Key is left unprotected as per --no-protect-key argument.")   
msg = "Burn keys in efuse blocks"   
  
if no\_protect\_key:   
msg += "The key block will left readable and writeable (due to --no-protect-key)"   
else: msg += "The key block will be read and write protected (no further changes or readback)"  
print(msg)   
efuses.burn\_all()   
print("Successful")

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 funkcionalita 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 ofsete 0x1000.

Aby bolo možné overiť softvérový Bootloader a zabezpečiť proces bootovania firmvéru, je nutné vygenerovať odtlačok (digest), ktorý je výstupom algoritmu SBDA (Secure Bootloader Digest Algorithm) a zapísať ho do flash pamäte na ofset 0x0, kde ho očakáva Secure Boot [42].

* + 1. SBDA algoritmus

Algoritmus SBDA spúšťa hardvérový Bootloader. Načíta AES šifrovací kľúč uložený v eFuse BLK2 v reverznej bitovej reprezentácii a obraz softvérového Bootloadera. Pred obraz Bootloadera dosadí 128 bajtový vygenerovaný inicializačný vektor. Algoritmus vykoná zarovnanie obrazu Bootloadera modulo 128 a doplní 0xFF do reprezentácie reťazca modulo 128. Na každých 16 bajtov otvoreného textu obrazu Bootloadera sa aplikuje bloková šifra AES256 v ECB móde s využitím AES šifrovacieho kľúča z eFuse BLK2 s dĺžkou 256 bitov.

Výsledný šifrovaný text má reverznú bitovú reprezentáciu. Algoritmus vymení bajt každého 4-bajtového slova šifrovaného textu, vypočíta SHA-512 hash výsledného 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. Princíp algoritmu SBDA je totožný, využijeme lokálne dostupný vygenerovaný AES šifrovací kľúč „secure-bootloader-key-256.bin“, ktorý sme do eFuse v predchádzajúcom kroku zapísali. 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 na preddefinovaný ofset 0x0.

**Fragment .py scriptu pre generovanie digestu Bootloadera:**

def digest\_secure\_bootloader(args):   
""" Calculate the digest of a bootloader image, in the same way the hardware secure boot engine would do so. Can be used with a pre-loaded key to update a secure bootloader. """  
\_check\_output\_is\_not\_input(args.keyfile, args.output)   
\_check\_output\_is\_not\_input(args.image, args.output)   
\_check\_output\_is\_not\_input(args.iv, args.output)   
if args.iv is not None:   
print("WARNING: --iv argument is for TESTING PURPOSES ONLY")   
iv = args.iv.read(128)   
else: iv = os.urandom(128)   
plaintext\_image = args.image.read()   
args.image.seek(0)   
  
# secure boot engine reads in 128 byte blocks (ie SHA512 block # size), but also doesn't look for any appended SHA-256 digest   
fw\_image = esptool.ESP32FirmwareImage(args.image)   
if fw\_image.append\_digest:   
if len(plaintext\_image) % 128 <= 32:   
  
# ROM bootloader will read to the end of the 128 byte block, but not   
# to the end of the SHA-256 digest at the end   
new\_len = len(plaintext\_image) - (len(plaintext\_image) % 128)   
plaintext\_image = plaintext\_image[:new\_len]   
  
# if image isn't 128 byte multiple then pad with 0xFF (ie unwritten flash)   
# as this is what the secure boot engine will see   
  
if len(plaintext\_image) % 128 != 0:   
plaintext\_image += b"\xFF" \* (128 - (len(plaintext\_image) % 128))   
  
plaintext = iv + plaintext\_image   
  
# Secure Boot digest algorithm in hardware uses AES256 ECB to   
# produce a ciphertext, then feeds output through SHA-512 to   
# produce the digest. Each block in/out of ECB is reordered   
# (due to hardware quirks not for security.)   
  
key = \_load\_hardware\_key(args.keyfile)   
backend = default\_backend()   
cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=backend)   
encryptor = cipher.encryptor()   
digest = hashlib.sha512() for block in get\_chunks(plaintext, 16): block = block[::-1]   
  
# reverse each input block   
cipher\_block = encryptor.update(block)   
  
# reverse and then byte swap each word in the output block cipher\_  
block = cipher\_block[::-1] for block in get\_chunks(cipher\_block, 4):   
  
# Python hashlib can build each SHA block internally   
digest.update(block[::-1])   
if args.output is None: args.output = os.path.splitext(args.image.name)[0] + "-digest-0x0000.bin"  
with open(args.output, "wb") as f:   
f.write(iv) digest = digest.digest()   
for word in get\_chunks(digest, 4):   
f.write(word[::-1])   
  
# swap word order in the result   
f.write(b'\xFF' \* (0x1000 - f.tell()))   
# pad to 0x1000   
f.write(plaintext\_image) print("digest+image written to %s" % args.output)

Príkaz má vstup - šifrovací AES kľúč s dĺžkou 256 bitov v binárnom formáte (rovnaký bol zapísaný do eFuse BLK2) a obrazu Bootloadera z priečinku projektu build. Pre generovanie odtlačku musí mať vývojár k dispozícii šifrovací kľúč a zodpovedá za jeho archiváciu a ochranu. Pri zápise digestu do flash pamäte je nutné zvoliť cieľový ofset na 0x0, kde ho očakáva hardvérový Bootloader.

Zápis digestu do flash pamäte som realizoval použitím príkazu: esptool.py write\_flash 0x0 bootloader-digest.bin. Po zapísaní digestu do flash pamäte a úspešnom overení odtlačku Bootloadera je možné pristúpiť k ďalšej fáze bootovacieho procesu – k bootovaniu firmvéru. Pri spustení mikrokontroléru sa vykoná hardvérové vypočítanie digestu, kedy hardvér (ROM Bootloader) pristúpi k vyčítaniu obsahu eFuse BLK2 a zo známeho ofsetu, kde je zapísaný softvérový Bootloader (0x1000) sa prevezme jeho obraz.

Vykoná sa algoritmus SBDA a výsledný digest porovná hardvérový Bootloader s digestom uloženým vo flash pamäti na ofsete 0x0. V prípade, že sú oba digesty identické, hardvérový Bootloader umožní spustiť softvérový Bootloader a ten pristúpi k bootovaniu firmvéru (s overením jeho digitálneho podpisu pri bootovaní, ale aj pri vzdialenej aktualizácii firmvéru…).

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. V procese nahrávania firmvéru cez USB-UART rozhranie sa prepíše aj softvérový Bootloader, tabuľka partícii na základe konfigurácie prostredia z ktorého sa nahrávanie firmvéru realizuje.

Vypočítaný digest sa tak nebude zhodovať s digestom zapísaným na ofsete 0x0 (za predpokladu, že nebude prepísaný) pri nahrávaní firmvéru. V prípade, ak by útočník dokázal zapísať iba firmvér na konkrétny ofset a prepísal by aj príznak v OTA\_DATA partícii, firmvér by nabootovaný nebol, nakoľko nie je digitálne podpísaný kľúčom, ktorý útočník k dispozícii nemá. Na Obr. 22 je v blokovej schéme zakreslený priebeh spúšťania systému mikrokontroléru ESP32 s využitím funkcionality Secure Boot po fázu pred bootovaním firmvéru.



Obr. 22 Bloková schéma procesu Secure Boot po bootovanie firmvéru

* 1. Šifrovanie flash pamäte

Nakoľko funkcionalita Secure Boot nerieši ochranu firmvéru a softvérového Bootloadera, ale iba bootovací proces, odporúča sa na základe dokumentácie ESP-IDF Secure Boot doplniť o šifrovanie flash pamäte [43] a využívať tieto funkcionality spolu pre vysokú úroveň bezpečnosti. Šifrovať je možné obsah celej flash pamäte, alebo jej časti.

Šifrovaná časť flash pamäte nie je potencionálnym útočníkom spustiteľná, nakoľko dokáže stiahnuť cez USB-UART rozhranie iba zašifrovaný text (ak sa využíva produkčná verzia šifrovania flash pamäte). Bloková schéma funkcionality šifrovania flash pamäte je na Obr. 23. OTA firmvér môže byť distribuovaný štandardne v otvorenom texte, alebo po úprave v programovej implementácii v šifrovanom texte, ktorý nie je pri zápise do flash pamäte už šifrovaný.



Obr. 23 Šifrovanie flash pamäte – Release režim

Táto funkcionalita dokáže zároveň obsah flash pamäte dešifrovať v reálnom čase, ak sa daná partícia používa. Využíva symetrický AES kľúč s dĺžkou 256 bitov pre šifrovanie a dešifrovanie obsahu flash pamäte. Kľúč je uložený v jednorázovo programovateľnej pamäti eFuse BLK1 s veľkosťou 256 bitov, ktorá je pre tento kľúč vyhradená.

V základnom nastavení šifrovania flash pamäte sú šifrované sektory:

* softvérový Bootloader,
* tabuľka partícii,
* všetky firmvérové partície – APP (aplikačné),
* digest Bootloadera na ofsete 0x0 – (ak sa využíva Secure Boot).

Iné typy partícii môžu byť šifrované dodatočne pridaním príznaku – flagu encrypted do tabuľky partícii.

**Tabuľka partícii je zapísaná v .csv formáte s príznakom šifrovaných aplikačných partícii:**

# 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, encrypted,  
ota\_0, app, ota\_0, 0x110000, 1M, encrypted,  
ota\_1, app, ota\_1, 0x210000, 1M, encrypted,

* + 1. Proces šifrovania flash pamäte

Po vytvorení buildu a jeho nahratí do mikrokontroléru ESP32 sú všetky dáta vo flash pamäti v otvorenom texte - nešifrované. Za predpokladu, že je spustená aj funkcionalita Secure Boot, hardvérový Bootloader overí softvérový Bootloader pre možnosť spustenia ďalších fáz bootovacieho procesu.

Po úspešnom overení softvérový Bootloader na základe príznaku zo súboru „sdkconfig“ pre príznak y (yes) pre makro prislúchajúce k zapnutiu funkcionality šifrovania flash pamäti načíta hodnotu eFuse FLASH\_CRYPT\_CNT. Ak je jej hodnota 0 (ešte nešifrovaná flash pamät), nastaví a aktivuje sa blok šifrovania obsahu flash pamäte. Bootloader nastaví 4-bitovú eFuse FLASH\_CRYPT\_CONFIG na hodnotu 0xF.

Operácie samotného šifrovania flash pamäte už vykonáva hardvérový Bootloader, nakoľko softvér nemá prístup pre zápis a čítanie do už zapísaných eFuses, ktoré sú od momentu zapísania chránené. Šifrujú sa partície a sektory vo flash pamäti označené značkou (Flagom z .csv súboru) encrypted. Pre šifrovanie aj dešifrovanie sa využíva (symetrický) AES kľúč zapísaný do eFuse BLK1 s veľkosťou 256-bitov, ktorá je na tento účel určená.

Pri väčších partíciách môže trvať šifrovanie až minútu. Softvérový Bootloader po ukončení šifrovania nastaví eFuse FLASH\_CRYPT\_CNT na hodnotu 0x01, čo znamená, že je obsah flash pamäte šifrovaný a opätovné šifrovanie sa pri reštarte a spúšťaní systému nevykoná. Ak je použitý režim Development, od tohto momentu je eFuse chránená aj proti opätovnému zápisu.

Šifrovanie flash pamäte je možné prevádzkovať v dvoch režimoch:

* Development (vývojársky) režim,
* Release (produkčný) režim.

Pre režim Development nastavuje softvérový Bootloader bity eFuse DISABLE\_DL\_DECRYPT a DISABLE\_DL\_CACHE na hodnotu 1, čím umožní Bootloaderu cez UART opätovne nahrávať už iba šifrovaný firmvér. Pri stiahnutí firmvéru – obsahu flash pamäte je obsah dešifrovaný a vývojár tak získava výstup v otvorenom texte, čo je pre produkčné aplikácia nebezpečné, nakoľko sa k firmvéru môže dostať útočník a spustiť ho na svojej mikrokontrolérovej platforme.

Zároveň však v tomto režime nie je eFuse FLASH\_CRYPT\_CNT chránená proti opätovnému zápisu. Zápis (prepis) tejto pamäte je podľa dokumentácie ESP-IDF možné vykonať maximálne tri krát, čím je možné šifrovanie flash pamäte po testovacej fáze projektu vypnúť a používať mikrokontróler ESP32 bez tejto funkcie v ďalšom vývoji / produkčnej aplikácii.

V produkčnom režime Release nastavuje Bootloader eFuse DISABLE\_DL\_ENCRYPT, DISABLE\_DL\_DECRYPT a DISABLE\_DL\_CACHE na hodnotu 1, čím zabraňuje dešifrovaniu obsahu flash pamäte pri jej čítaní cez UART. Výstupom v tomto prípade je tak zašifrovaný text, ktorý nie je spustiteľný na inej mikrokontrólerovej platforme ESP32, nakoľko kľúč pre šifrovanie / dešifrovanie je uložený v eFuse, ktorý nie je softvérovo prečítateľná. Útočník tak bez znalosti kľúča nedokáže firmvér spustiť.

Režim Release šifrovania flash pamäte taktiež zavádza ochranu proti zápisu do eFuse FLASH\_CRYPT\_CNT. Funkcionalitu šifrovania flash pamäte tak už nie je možné v budúcnosti vypnúť, ani využiť dostupné 3 prepisy ako v prípade Development režimu. Po prvotnom nastavení eFuses a zašifrovaní obsahu flash pamäte sa následne mikrokontróler ESP32 reštartuje a bootuje už partície v šifrovanom texte, ktoré v reálnom čase dešifruje s využitím AES symetrického kľúča v eFuse BLK1. Dešifrovaný firmvér sa tak zavedie do RAM pamäte a spustí.

* + 1. Algoritmus šifrovania flash pamäte

Algoritmus šifrovania flash pamäte využíva blokovú šifru AES256, ktorá pracuje na 16 B blokoch dát. Pri tomto algoritmu pracuje bloková šifra na 32 B blokoch dát, teda sa používajú dva bloky AES v sérii. AES256 používa symetrický kľúč z eFuse BLK1 s dĺžkou 256 bitov v reverznej bitovej reprezentácii.

Pre proces šifrovania a dešifrovania sa používa AES opačne. Proces šifrovania je tvorený funkciou AES decrypt a proces dešifrovania flash pamäte funkciou AES encrypt. Pri šifrovaní sa na každý 32 B blok (2 bloky v sérii) otvoreného textu aplikuje unikátny šifrovací kľúč, ktorý je odvodený od hlavného a je XOR – ovaný (exkluzívny súčet) s offsetom bloku otvoreného textu vo flash pamäti.

XOR-ovanie konkrétnych bitov kľúča závisí od 4-bitovej eFuse FLASH\_CRYPT\_CONFIG, ktorá je štandardne nastavená na hodnotu 0xF. To zaručuje, že sú XOR-ované všetky bity odvodeného AES kľúča s ofsetom bloku dát. Zmena hodnoty zapísanej v eFuse FLASH\_CRYPT\_CONFIG by mohla znížiť kryptografickú bezpečnosť šifrovania flash pamäte, nakoľko by sa operácia XOR nevykonala pre všetky bity AES kľúča.

Nastavenie bitov eFuse FLASH\_CRYPT\_CONFIG a operácia XOR pre rozsahy bitov kľúča:

* ak je nastavený 1. bit, bity 0 až 66 sú XOR-ované,
* ak je nastavený 2. bit, bity 67 až 131 sú XOR-ované,
* ak je nastavený 3. bit, bity 132 až 194 sú XOR-ované,
* ak je nastavený 4. bit, bity 195 až 256 sú XOR-ované.

**Fragment .py scriptu vykonávajúci operáciu šifrovania súboru pre nahratím do flash pamäte:**

def \_flash\_encryption\_operation(output\_file, input\_file, flash\_address, keyfile, flash\_crypt\_conf, do\_decrypt):   
key = \_load\_hardware\_key(keyfile)   
if flash\_address % 16 != 0:   
raise esptool.FatalError("Starting flash address 0x%x must be a multiple of 16" % flash\_address)   
if flash\_crypt\_conf == 0:   
print("WARNING: Setting FLASH\_CRYPT\_CONF to zero is not recommended")   
if esptool.PYTHON2:   
tweak\_range = \_flash\_encryption\_tweak\_range(flash\_crypt\_conf)   
else: tweak\_range = \_flash\_encryption\_tweak\_range\_bits(flash\_crypt\_conf)   
key = int.from\_bytes(key, byteorder='big', signed=False)   
aes = None block\_offs = flash\_address   
while True: block = input\_file.read(16)   
if len(block) == 0: break elif len(block) < 16:   
if do\_decrypt: raise esptool.FatalError("Data length is not a multiple of 16 bytes")   
pad = 16 - len(block)   
block = block + os.urandom(pad)   
print("Note: Padding with %d bytes of random data (encrypted data must be multiple of 16 bytes long)" % pad)   
if (block\_offs % 32 == 0) or aes is None:   
# each bit of the flash encryption key is XORed with tweak bits derived from the offset of 32 byte block of flash   
block\_key = \_flash\_encryption\_tweak\_key(key, block\_offs, tweak\_range)   
aes = ECB(block\_key) block = block[::-1] # reverse input block byte order   
# note AES is used inverted for flash encryption, so   
# "decrypting" flash uses AES encrypt algorithm and vice   
# versa. (This does not weaken AES.)   
if do\_decrypt: block = aes.encrypt(block)   
else: block = aes.decrypt(block)   
block = block[::-1]   
# reverse output block byte order   
output\_file.write(block)   
block\_offs += len(block)

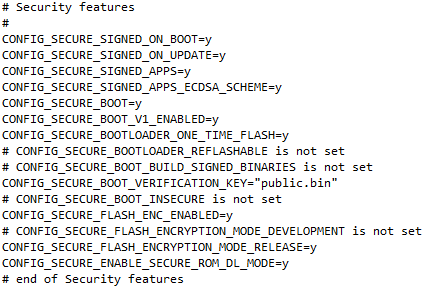
* + 1. Šifrovanie flash pamäte – implementácia v prostredí ESP-IDF

Prvým krokom pre spustenie šifrovania flash pamäte je vygenerovanie a zapísanie symetrického kľúča do príslušnej jednorázovo programovateľnej pamäte eFuse BLK1. S využitím nástroja espsecure.py a príkazom espsecure.py generate\_flash\_encryption\_key my\_flash\_encryption\_key.bin je možné kľúč vygenerovať. Funkcia pre generovanie AES kľúča je totožná ako v prípade kľúča pre Secure Boot.

Vygenerovaný kľúč je následne potrebné zapísať do vyhradenej jednorazovo programoteľnej pamäte eFuse BLK1, ktoré je pre neho určená. Príkazom espefuse.py --port PORT burn\_key flash\_encryption my\_flash\_encryption\_key.bin sa kľúč zapíše do eFuse BLK1. Aby šifrovanie flash pamäte mohlo byť spustené pre Release režim (použité pre hlavnú aplikáciu) bolo nutné vyhotoviť build na základe zvolenej konfigurácie z Menuconfig.

Makrá pre šifrovania flash pamäte sú obsiahnuté v súbore „sdkconfig“ a definujú makrá pre zvolený režim šifrovania flash pamäte. Makrá sú vložené do Bootloadera v procese vytvárania buildu. Pri prvom spustení vykoná Bootloader úpravy v eFuses, ktoré sú ovplyvnené navoleným režimom šifrovania flash pamäte.

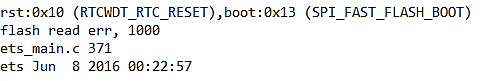
Na Obr. 24 sú definované makrá v súbore „sdkconfig“ pre funkcionality Secure Boot a šifrovanie flash pamäte (obe) pre Release režim, ktoré sú v procese kompilácie vložené do softvérového Bootloadera. Na základe makier a nastavení sa vykoná prvotné zašifrovanie flash pamäte. Nakoľko ide o produkčný (Release) režim, nie je možné neskôr funkcionalitu šifrovania flash pamäte vypnúť.



Obr. 24 Konfigurácia bezpečnostných makier pre Secure Boot a šifrovanie flash pamäte

Nastavením eFuses v produkčnom režime šifrovania flash pamäte je zároveň zabránené aj dešifrovaniu obsahu flash pamäte, ktorá môže byť stále prečítaná cez USB-UART rozhranie. To by malo za následok, že by potencionálny útočník dokázal získať pôvodný firmvér, alebo celý obsah flash pamäte v otvorenom texte a dokázal by ho spustiť na svojom hardvéri.

Po prvotnom spustení funkcionality šifrovania flash pamäte vykoná hardvérový Bootloader šifrovanie obsahu flash pamäte na preddefinovaných oddieloch a partíciách vo flash pamäti s označením encrypted. Od tohto momentu je možné zapísať cez UART iba šifrovaný firmvér. V prípade, že je firmvér nahratý do mikrokontroléru ako otvorený text, nedôjde k jeho spusteniu, nakoľko po jeho dešifrovaní nie je v spustiteľnej podobe, tento stav je na Obr. 25, ktorý je výpisom z UART monitoru po nahratí firmvéru v otvorenom texte, mikrokontróler sa reštartuje v nekonečnom cykle a indikuje v chybovej hláške nemožnosť načítania obsahu flash pamäte, ktorá nie je šifrovaná.



Obr. 25 Problém s načítaním obshu flash pamäte

Nakoľko využívam Secure Boot a šifrovanie flash pamäte súčasne, proces nahrávania firmvéru do mikrokontroléru ESP32 cez USB-UART rozhranie je zložitejší a vyžaduje viacero krokov s presnou postupnosťou. V prvom rade je nutné skompilovať program do binárnej podoby pre vytvorenie firmvéru aplikácie v otvorenom texte.

Následne je potrebné podpísať súkromným kľúčom firmvér v otvorenom texte a zašifrovať ho s použitím symetrického AES kľúča. Ďalším krokom je nahratie firmvéru do flash pamäte mikrokontroléru na konkrétny ofset. V tomto prípade je efektívne vytvorenie jednoduchého spustiteľného súboru .bat, ktorý dokáže všetky kroky vykonať automatizovane.

Šifrovanie firmvéru (obdobne pre Bootloader, digest a iné súbory zapísané vo flash pamäti, ktoré majú byť šifrované), sa realizuje príkazom: espsecure.py encrypt\_flash\_data --keyfile my\_flash\_encryption\_key.bin --address 0x20000 -o ./build/app-encrypted.bin ./build/native\_ota.bin. Príkaz má viacero parametrov – AES symetrický kľúč pre šifrovanie, adresu (začiatočný ofset), kde bude firmvér zapísaný, nakoľko algoritmus využíva operáciu XOR s ofsetom blokov dát.

Ďalšími parametrami je zašifrovaný súbor a pôvodný súbor. Do parametrov nie je možné dosadiť rovnaký názov súboru pre zašifrovaný a pôvodný. Výstupom algoritmu pri takomto vstupe súborov je prázdny súbor firmvéru s veľkosťou 0 B. Príkazom esptool.py write\_flash 0x20000 ./build/app-encrypted.bin je možné zapísať šifrovaný firmvér na konkrétne umiestnenie do flash pamäte, aby bol korektne dešifrovaný a spustiteľný v RAM pamäti.

OTA aktualizácie je však možné naďalej distribuovať v otvorenom texte a k ich zašifrovaniu dôjde v procese uloženia do flash pamäte mikrokontroléru ESP32 automaticky. Pri zmene funkcie pre zápis OTA aktualizácie (funkciou esp\_partition\_write()) do flash pamäte je možné zapísať už zašifrovanú aktualizáciu, ktorú musí vydavateľ zašifrovať ešte pred umiestnením na OTA úložisko.

Aktualizovaný firmvér v takomto prípade nebude v procese aktualizácie šifrovaný. Ak vydavateľ pri takejto zmene nahrá firmvér v otvorenom texte, mikrokontróler ho nedokáže spustiť, nakoľko ho proces dešifrovania prevedie do nespustiteľnej podoby, bootovanie zlyhá.

1. Realizácia programu pre aktualizáciu senzorového uzla

Pri návrhu a následnej realizácii senzorového uzla na mikrokontroléri ESP32 bola priorita predovšetkým implementovať čo najviac bezpečnostných mechanizmov z nástrojov prostredia ESP-IDF pre overenie integrity firmvéru a spustenie iba overeného firmvéru, ktorý mohol byť do flash pamäte mikrokontroléra nahratý cez rozhranie UART.

Ako som spomenul v predchádzajúcej kapitole, využil som ukážkový projekt Native OTA vo frameworku ESP-IDF, ktorý zaručuje prebratie aktualizácie (firmvéru) z externého webservera s overením verzie firmvéru. Okrem toho zabezpečuje pripojenie na webserver cez šifrovaný protokol HTTPS, čím zaručuje bezpečný prenosový kanál. Základný projekt som následne rozšíril o digitálny podpis pre overenie integrity firmvéru. V začiatkoch realizácie diplomovej práce a aplikácie pre senzorový uzol som využíval ESP-IDF vo verzii release 3.3, ktorá bola v roku 2019 najvyššou release verziou.

Táto verzia ESP-IDF ešte nemala integrovanú podporu pre implementáciu digitálneho podpisu. Digitálny podpis bolo možné cez Menuconfig nastaviť, avšak problém nastal v procese kompilácie, kedy sa mal verejný kľúč vložiť do obrazu Bootloadera. Následnou aktualizáciou frameworku na novšiu release verziu 4.0 sa mi podarilo digitálny podpis implementovať s plnou podporou. Firmvér som umiestnil cez FTP klienta na vlastný webserverver dostupný na doméne „https://esp32.sk“ na preddefinované umiestnenie, kde ho očakáva mikrokontróler.

V poslednej fáze testovania aplikácie som využil ESP-IDF v najnovšej verzii 4.2. Native OTA projekt sa v programovej implementácii zmenil, na čo ma upozornil kompilátor, ktorý detegoval deprecated funkcie. Z toho dôvodu som musel vykonať korekciu hlavičkových súborov a podľa aktuálnej verzie upraviť niekoľko volaní funkcií v hlavnom tasku.

Podarilo sa mi implementovať aktualizáciu firmvéru s overením integrity v plnom rozsahu. Digitálny podpis overuje mikrokontróler ESP32 pri každom bootovaní firmvéru z flash pamäte a tiež aj priamo v procese aktualizácie pre akciu „ON\_UPDATE“, kedy sa overuje verzia práve stiahnutého – uloženého firmvéru do flash pamäte.

Následnou implementáciou funkcionality Secure Boot v Release režime som zabezpečil bootovací proces a spúšťanie firmvéru od dôveryhodného vydavateľa, nakoľko šifrovací kľúč nie je softvérovo prečítateľný z eFuse a má ho k dispozícii iba vydavateľ pre vygenerovanie digestu, ktorý je zapísaný do flash pamäte na ofset 0x0. Nakoľko môže byť firmvér a obsah celej flash pamäte stiahnutý cez UART linku, bolo nutné zamedziť prečítaniu a zároveň zamedziť možnosti spustenia firmvéru na iných mikrokontorléroch.

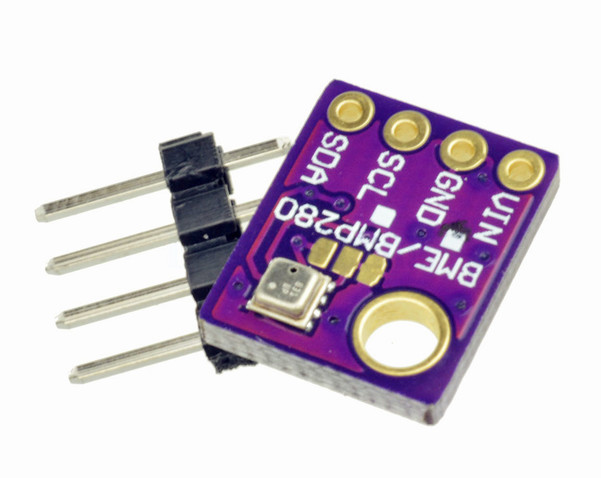
Z toho dôvodu som využil šifrovanie flash pamäte - funkcionalitu mikrokontroléru ESP32, ktorá štandardne obsah najdôležitejších oblastí – digest, Bootloader a firmvérové partície šifruje a pri zavádzaní danej oblasti do RAM pamäte dešifruje. Kľúč pre šifrovanie / dešifrovanie flash pamäte je chránený v eFuse a nie je softvérovo čitateľný ani prepisovateľný. Potencionálny útočník tak dokáže obsah flash pamäte prečítať, avšak ho nedokáže spustiť, nakoľko štandardný mikrokontróler ESP32 nedokáže spustiť šifrovaný firmvér. Využil som Release režim funkcionality šifrovania flash pamäte.

Mikrokontroléry ESP32 prechádzajú revíziami a v súčasnosti sú produkované už v revízii 3, ktorá umožňuje zakázať prostredníctvom eFuse UART\_DOWNLOAD\_DIS sťahovanie firmvéru cez UART. Nakoľko mám niekoľko-ročný mikrokontróler ESP32 ešte v revízii 0 na ktorom táto funkcionalita nie je podporovaná, nedokázal by som zamedziť stiahnutiu firmvéru bez fyzického zásahu do mikrokontroléru (odpájkovanie USB portu, odpojenie vyvedených vývodov hardvérovej UART zbernice).

* 1. Realizácia programu pre senzorový uzol

Program senzorového uzla som sa rozhodol demonštrovať na aplikácii senzorového uzla pre ESP32 navrhnutú v Bakalárskej práci v prostredí Arduino IDE. Aplikáciu som musel prepracovať do prostredia ESP-IDF, kde som sa snažil o zachovanie pôvodnej funkcionality – záznam dát z meteorologického senzora.

Pôvodne som pre senzorový uzol využíval senzor Sensirion SHT21, nakoľko som ho však nemal k dispozícii, využil som senzor Bosch BME280 [44] znázornený na Obr. 26, ktorý umožňuje záznam trojkombinácie – teplota, tlak, vlhkosť vzduchu. Využil som verziu senzora s operačnou logikou 3,3V pre plnú kompatibilitu s operačnou logikou mikrokontroléru ESP32.



Obr. 26 Bosch BME280 - senzor teploty, tlaku, vlhkosti vzduchu

Pre ovládanie a komunikáciu so senzorom Bosch BME280 som využil implementáciu ovládača (Drivera) v jazyku C od Bosch Sensortec [45]. Senzor komunikuje s mikrokontrolérom po I2C (Intel-Integrated-Circuit) zbernici [46] 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.

V mojom prípade bola odhadnutá nadmorská výška oproti skutočnej (Šuňava, okres Poprad – 854 m.n.m) nižšia o približne 20 metrov, nakoľko som senzor prevádzkoval v interiéri z dôvodu logiky webovej aplikácie pre riadenie výstupu na základe interiérovej teploty. Nevyužil som kompenzovaný – referenčný tlak vzduchu z lokálnej meteostanice pre aktuálnu nadmorskú výšku, ale využil som „strednú“ hodnotu 1013.25 hPa, čo je atmosférický tlak na hladine mora.

Fragment zdrojového kódu v jazyku C - vzorec pre prepočet absolútneho tlaku vzduchu na relatívny:  
altitude = 44330 \* (1.0 - pow(bme280\_compensate\_pressure\_double(v\_uncomp\_pressure\_s32)/100 / 1013.25, 0.1903));

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);

Vzorec pre prepočet absolútneho tlaku vzduchu na relatívny využíva aj teplotu vzduchu. Nakoľko bol senzor prevádzkovaný v interiéri, teplota bola voči vonkajšej vyššia aj o niekoľko desiatok stupňov vzhľadom na zimné a jarné mesiace, čo do prepočtu vnieslo istú nepresnosť, rádovo ± 1 hPa. V programovej implementácii 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 (nazýva sa aj FORCED).

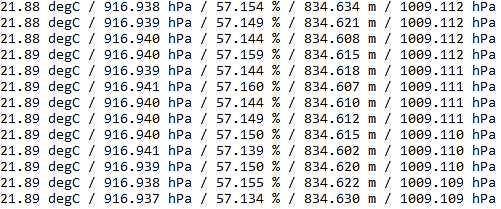
Normálny prevádzkový režim umožňuje využiť vyššie vzorkovanie nameraných údajov pre dosiahnutie presných meraní spoločne s nastavením koeficienta IIR (filter s nekonečnou impulznou odozvou) filtra, ktorý kompenzuje krátkodobé výkyvy tlaku vzduchu a teploty. Vzorkovanie je možné nastaviť v rozsahu 1 až 16 krát [47]. 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). Na Obr. 27 je záznam z UART monitora, ktorý poukazuje na konzistentnosť nameraných dát v režime NORMAL s frekvenciou záznamov 25Hz, teda 40 záznamov za sekundu. Teplota využíva 16 - násobné vzorkovanie, tlak vzduchu (absolútny) a vlhkosť vzduchu 2 – násobné vzorkovanie.

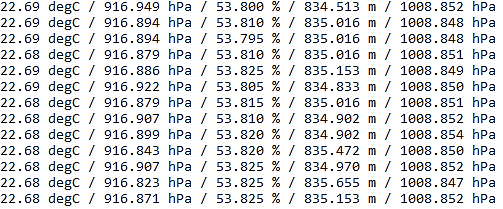
Výpis na z UART monitora obsahuje záznam veličín zľava doprava:

* teplota vzduchu,
* absolútny tlak vzduchu pre aktuálnu nadmorskú výšku,
* vlhkosť vzduchu,
* odhadnutá nadmorská výška,
* relatívny tlak vzduchu prepočítaný na hladinu mora.



Obr. 27 Konzistentnosť meraní senzora BME280 v NORMAL móde

FORCED režim nebol tak stabilný a neponúkal konzistentné merania, ako je možné vidieť na Obr. 28 z výstupu UART monitora Namerané údaje sa v čase menili skokovito pri každom meracom cykle nad i pod strednú hodnotu (známu z merania normálnym prevádzkovým módom). Ku skokovitému priebehu nameraných veličín prispel aj fakt, že v tomto režime nie sú krátkodobé výkyvy kompenzované IIR filtrom a taktiež sa nevykonáva viacero priemerovaných meraní.



Obr. 28 Konzistentnosť meraní senzora BME280 vo FORCED móde

Mikrokontróler ESP32 využíva z pôvodného konceptu senzorového uzla ovládanie digitálneho výstupu GPIO 23, ktorý riadi SSR (Solid-state relay) relé [48]. 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 [49] je dostupný v prílohe A na CD nosiči. V Tab. 4 je tabuľkové zapojenie vývodov mikrokontroléru ESP32 s perifériami, ktoré boli pre senzorový uzol použité.

Tab. 4 Pripojenie vývodov ESP32 k vývodom použitých periférii

|  |  |
| --- | --- |
| **ESP32** | **Bosch BME280** (senzor teploty, tlaku a vlhkosti vzduchu) |
| 3,3V | Vcc |
| GND | GND |
| GPIO22 (HW SCL) | SCL |
| GPIO21 (HW SDA) | SDA |
| **ESP32** | **SSR relé OMRON G3MB-202P** - externé 5V napájanie |
| 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 na Obr. 29 s konfiguráciou cieľovej (referenčnej) teploty a hysterézy [50]. Základné rozhranie spoločne s vygenerovaním a nakonfigurovaním certifikátov certifikačnej autority a webservera pre realizáciu zabezpečeného spojenia bolo vytvorené v mojej bakalárskej práci.



Obr. 29 Webové rozhranie termostatu pre ovládanie relé

Návod na inštaláciu webového rozhrania, generovanie jednotlivých certifikátov kryptografickým nástrojom OpenSSL je dostupné v prílohe A na CD nosiči v časti Návody. Návod je univerzálny pre operačný systém webservera Cent OS a pre serverovú službu HTTPD [51]. Súčasťou príloh je aj opis funkcií webového rozhrania, ktoré ponúkala základná verzia z bakalárskej práce.

Aplikácia samotného senzorového uzla využíva prenos údajov na rovnaké webové rozhranie, ktoré poskytuje OTA aktualizáciu firmvéru. Následne som použil projekt HTTPS\_request, ktorý je dostupný v ESP-IDF a skombinoval som ho s projektom Native OTA. Vykonal som niekoľko úprav v konfiguračných súboroch aj v samotnom programe, nakoľko pôvodne projekt HTTPS\_request využíva samostatný súbor pre certifikát certifikačnej autority s iným názvom ako Native OTA a v inom umiestnení (hĺbka priečinka).

Úpravou som dosiahol, že OTA aj HTTPS request využíva rovnaký súbor s certifikátom certifikačnej autority. Programová implementácia oboch projektov (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 projektu HTTPS\_request využíva GET request. Nakoľko pre prenos údajov mikrokontróler vykonáva POST request, musel som do programovej implementácie vykonať úpravu 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 [52]).

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)* [53]... V Tab. 5 je zoznam taskov vykonávajúcich HTTPS request na webserver s popisom funkcie, ktorú vykonávajú.

Tab. 5 Tasky senzorového uzla pre odosielanie a načítanie dát z webového rozhrania

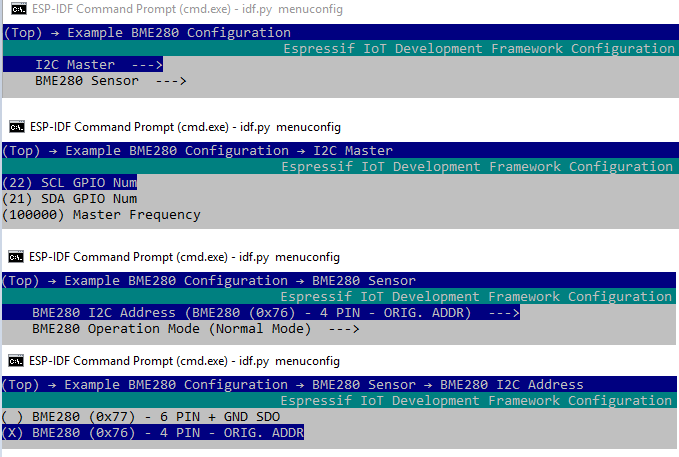
|  |  |
| --- | --- |
| **Názov tasku** | **Funkcia** |
| https\_get\_task | Vykonáva pravidelný HTTPS POST request s dátami obsiahnutými v tele správy. Pri spustení tasku sa vyskladá request z nameraných údajov, ktoré sú obsiahnuté v globálnych premenných. Spustením .php scriptu sa vykoná aj logika 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 funkcionalitu, 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 vedúcim práce sa nám podarilo implementovať pôvodnú programovú implementáciu v jazyku C do frameworku ESP-IDF pre možnosť inicializácie senzora Bosch BME280 v dvoch prevádzkových 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“ [54], ktorý je dostupný v priečinku projektu samostatné druhé menu (prvé menu je existujúca konfigurácia Native OTA projektu). Menu je rozvetvené na dve podmenu - „I2C Master“ a „BME280 Sensor“. Prvá podmenu „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. 30 opisuje 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.



Obr. 30 Vlastné konfiguračné menu pre voľbu I2C parametrov a adresy, režimu BME280

Logická úroveň vývodu SD0 mení LSB (Najmenej významný bit) komunikačnej I2C adresy. Taktiež je možné vybrať operačný mód senzora Bosch BME280 NORMAL / FORCED v ktorom bude pracovať. 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 obsahujú predponu „CONFIG\_“ a sú uložené do súboru „sdkconfig“ po úpravách v Menuconfigu, kde sú jednotlivé menu vyobrazené.

Pre prístup aplikácie napísanej v jazyku C k súboru „sdkconfig“ v projekte je nutné v programe definovať makro „COMBINED\_INIT\_CODE“. Následne už môže aplikácia používať jednotlivé makrá zo súboru „sdkconfig“. Makro podporované v „sdkconfig“ môže mať dátový typ boolean (binárny yes/no), integer (celé číslo), string (otvorený text). 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 (hexadecimálna) 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 pri inicializácii komunikácie mikrokontroléru ESP32 so senzorom. Konfiguračné menu vždy zadefinuje iba jeden režim, čím zabraňuje voľbe zvoliť oba režimy súčasne. Zaručuje tak aj štart iba jedného tasku pre komunikáciu so senzorom Bosch BME280 na základe zvoleného 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). Bloková schéma riešenia kompletného senzorového uzla komunikujúceho so serverom zabezpečeným prenosovým kanálom je na Obr. 31.



Obr. 31 Bloková schéma riešenia senzorového uzla na báze ESP32

* 1. Úpravy webového rozhrania

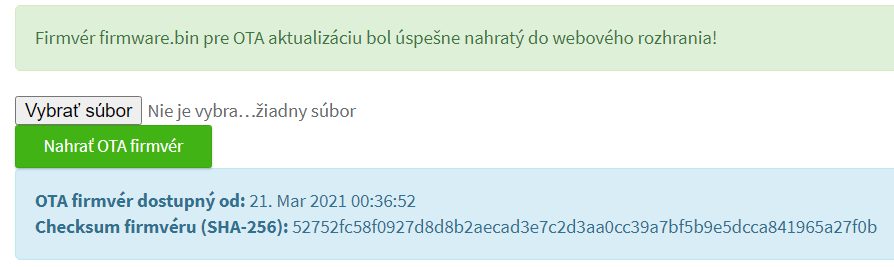
Realizácia diplomovej práce na strane webservera spočívala v úpravách existujúceho webového rozhrania po grafickej stránke, počtu a následnej vizualizácii zaznamenávaných údajov zo senzorového uzla. Zaznamenané údaje vo webovom rozhraní sú vizualizované na hlavnej stránke – ukážka zaznamenaných dát na Obr. 32.



Obr. 32 Vizualizácia nameraných údajov vo webovom rozhraní

Úpravy backendu (funkčnosti na serverovej strane) sa týkali PHP [55] súboru zapisdata.php, ktorý dokáže prevziať dáta, ktoré odosiela mikrokontróler 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 nahrať firmvér pre mikrokontróler ESP32 v binárnom formáte cez HTML formulár. Grafické rozhranie webaplikácie s možnosťou nahratia firmvéru do úložiska webservera je na Obr. 33).

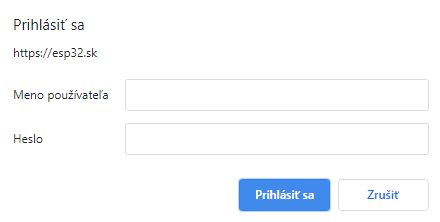


Obr. 33 Grafické rozhranie webstránky pre nahratie OTA firmvéru cez HTML formulár

Serverový jazyk PHP pri spracovaní overí formát súboru (očakáva .bin), následne ho premenuje na „firmware.bin“ (uložený firmvér na disku počítača po kompilácii projektu Native OTA je „native\_ota.bin“) a uloží ho do preddefinovaného – koreňového umiestnenia priečinku webaplikácie, kde ho očakáva mikrokontróler vykonávajúci GET požiadavku pre stiahnutie jeho obsahu a vykonanie aktualizácie firmvéru.

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. Nakoľko som chcel autorizovať používateľský prístup pre nahratie firmvéru do úložiska webservera, kde ho očakáva mikrokontróler, implementoval som v PHP autorizáciu HTTP Basic Auth [56]. Prihlasovací formulár autorizácie HTTP Basic Auth je na Obr. 34, slúži pre zadanie informácie pre používateľské meno a heslo.



Obr. 34 HTTP Basic Auth

**Fragment kódu v jazyku PHP pre implementáciu HTTP Basic Auth**

<?php  
$pouzivatelske\_meno = "Username"; //Meno používateľa   
$pouzivatelske\_heslo = "Password"; //Heslo  
  
$valid\_passwords = array ($pouzivatelske\_meno => $pouzivatelske\_heslo);   
$valid\_users = array\_keys($valid\_passwords);   
$user = $\_SERVER['PHP\_AUTH\_USER'];   
$pass = $\_SERVER['PHP\_AUTH\_PW'];   
  
//Overenie, ci bol používateľ autorizovaný  
$validated = (in\_array($user, $valid\_users)) && ($pass == $valid\_passwords[$user]);   
if (!$validated) {   
header('WWW-Authenticate: Basic realm="My Realm"');   
header('HTTP/1.0 401 Unauthorized');   
die ("Not authorized"); //ukončenie PHP scriptu, nenačíta sa stránka za autorizáciou  
}  
?>

Tento typ HTTP autorizácie umožňuje používateľovi zadať autentizačné údaje po vyzvaní serverom pri vstupe na stránku, ktorá vyžaduje autorizáciu. Autentizačné údaje sú potrebné pre úspešnú autorizáciu a následné sprístupnenie webovej stránky. V prípade neúspešnej autorizácie môže klient opakovať požiadavku a zadať znova meno a heslo. V prípade, že klient okno prihlasovacieho formulára zatvorí, webserver mu odpovie návratový kódom HTTP 401 – Unauthorized.

Výhodou tejto autoziácie je veľmi jednoduchá implementácia na strane webservera, ktorá sa môže vzťahovať na celú webovú stránku, alebo jej časť. Autorizáciu je možné umiestniť napríklad do backendu - spracovania HTML formulára, ktorým sa nový firmvér pre mikrokontróler do úložiska webservera nahráva. Klient tak dokáže prezerať celú stránku, zadať cieľový firmvér pre nahratie a po spustení spracovania HTML formulára ho webserver vyzve k zadaniu autentizačných informácii pre dokončenie PHP scriptu.

Klient je po úspešnom overení autorizovaný (má platnú reláciu) štandardne 30 minút, ak nie je v konfigurácii webservera nastavený iný interval. Medzi výhody autorizácie určite patrí aj širokospektrálna podpora na strane klientov, nakoľko ho podporujú všetky webové prehliadače. Meno a heslo sa pri odoslaní autentizačných informácii prevedie na jeden textový reťazec otvoreného textu, ktorý je tvorený menom, dvojbodkou a heslom.

Výsledný reťazec sa zakóduje do BASE64 [57] formátu a je poslaný serveru. Tento reťazec je veľmi jednoduché dekódovať a tak sa HTTP autentizácia využíva iba pri zabezpečenom prenosovom kanáli – HTTPS, ktorý vytvára šifrované spojenie priamo medzi serverom a klientom. Pri HTTP spojení cez internet sa tento typ autentizácie nevyužíva, nakoľko ktokoľvek dokáže obsah správy s autentizačnými dátami prečítať a dekódovať.

Pri HTTP spojení sa autentizácia využíva iba v LAN sieti, kde poznáme všetky sieťové zariadenia a sieť je dôveryhodná. Existuje aj množstvo variantov autorizácii, napríklad Digest, či Bearer token, ktorý je vhodný pre autorizáciu požiadaviek, využíva sa napríklad v cloudových riešeniach pre API s obmedzeným počtom požiadaviek za hodinu (Tento typ autentizácie využíva populárna služba OpenWeather, ktorá sprostredkúva informácie o počasí celosvetovo v JSON / XML formáte). Zdrojové kódy celej webaplikácie pre senzorový uzol sú dostupné spoločne s konfiguráciou webservera dostupné v prílohe A na CD médiu.

* 1. 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 napäťových úrovní pre možnosť napájania externým zdrojovom napájania v rozsahu 6V až 18V.

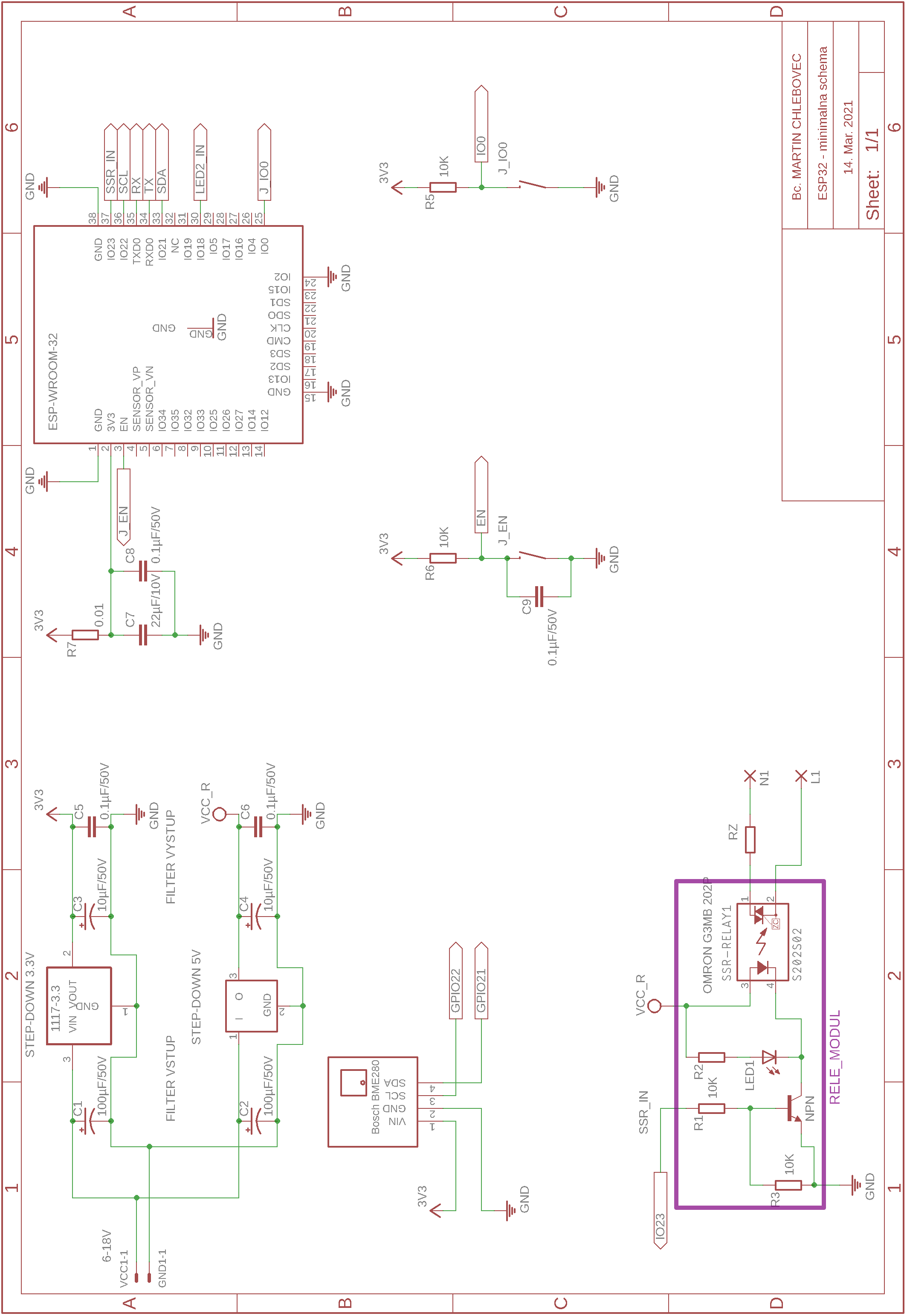
V prípade použitia samostatného čipu sú do schémy zakomponované tlačidlá pre signály EN a BOOT, ktorými je možné nastavovať špecifický režim čipu ESP32 na základe kombinácie ich stlačenia. Špecifickou sekvenciou stlačení je možné nastaviť hlavný čip ESP32 do režimu nahrávania programu (tzv. programovací 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 napäťový nábeh signálov v čase. Nakoľko sú oba signály pri spúšťaní mikrokontroléru v logickej 0, doska sa prepne do režimu sťahovania programu (DOWNLOAD\_FLASH\_CONTENTS), 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 (odpojení a pripojení napájania) nedokáže reagovať až do fyzického reštartu tlačidlom, kedy sa spustí firmvér z bootovatelnej partície. 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. 35) 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 periférii s mikrokontrolérom ESP32 je plne ekvivalentná s tabuľkovým zapojením vývodov z tabuľky (Tab. 4) v kapitole 7.1.

Na výstup relé je možné pripojiť záťaž (v minimálnej schéme zapojenia na Obr. 35 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. Schéma je exportovaná z programu pre návrh elektrotechnických schém Autodesk Eagle [58].



Obr. 35 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 funkcionalitu do prostredia ESP-IDF.

Metóda digitálneho podpisu má široké využitie nielen v mikrokontrólerovej 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 skráti 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 funkcionalitu permanentne zapol 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 prepisovateľná, alebo čitateľná, nedokáže potencionálny 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).

Podarilo sa mi demonštrovať funkciu senzorového uzla s bezpečnou aktualizáciou firmvéru cez internet s využitím bezpečného prenosového kanála a vývojárskych nástrojov v prostredí ESP-IDF, ktorými som implementoval vysokú mieru bezpečnosti pre bezpečnú prevádzku IoT platformy ESP32 v rôznych aplikáciách, kde sa vyžaduje vzdialená aktualizácia firmvéru cez internet.

Platforma je zároveň chránená aj pred fyzickým útokom podvrhnutia firmvéru cez USB-UART rozhranie útočníkom. Taktiež je chránený aj firmvér pred prečítaním a spustením útočníkom v prípade jeho stiahnutia z flash pamäte mikrokontroléru ESP32, čomu sa pri aktuálnej revízii použitého čipu ESP32 nedá zabrániť systémovou funkcionalitou. Firmvér je pri stiahnutí cez USB-UART rozhranie vo formáte šifrovaného textu, ktorý nie je spustiteľný bez dešifrovania s využitím symetrického kľúča bezpečne archivovaného v eFuse BLK1.

Súčasťou príloh je aj návod pre vygenerovanie certifikátov pre webserver, certifikačnú autoritu, kompletné nastavenie webservera. Pre mikrokontróler 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. ESP-NOW [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp\_now.html
3. USING THE BLE FUNCTIONALITY OF THE ESP32 [online]. Electronics-Lab.com [cit 2020-03-14]. Dostupné z: https://www.electronics-lab.com/project/using-the-ble-functionality-of-the-esp32/
4. 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
5. 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
6. Getting started with MicroPython on the ESP32 [online]. MicroPython [cit. 2021-04-15]. Dostupné z: https://docs.micropython.org/en/latest/esp32/tutorial/intro.html
7. Getting started with Lua Language on ESP32 [online]. IoT Design Pro [cit. 2019-01-23]. https://iotdesignpro.com/projects/getting-started-with-lua-programming-on-esp32
8. low.js | Node.js for microcontrollers [online]. Neonious [cit. 2021-04-15]. https://iotdesignpro.com/projects/getting-started-with-lua-programming-on-esp32
9. ESP8266 [online]. Wikipedia [cit 2021-03-31]. Dostupné z:   
   https://en.wikipedia.org/wiki/ESP8266
10. 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
11. 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
12. 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
13. Katalógový list CP2102 [online]. Silicon Labs [cit. 2017-01-17]. https://www.silabs.com/documents/public/data-sheets/CP2102-9.pdf
14. Katalógový list ESP32-S [online]. Shenzhen Anxinke Technology [cit. 2016-10-03]. https://www.es.co.th/Schemetic/PDF/ESP32.PDF
15. 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
16. 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
17. 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
18. 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
19. 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
20. 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
21. ESP-IDF Release v4.2 [online]. Github [cit. 2020-12-07]. Dostupné z: https://github.com/espressif/esp-idf/releases/tag/v4.2
22. 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
23. Esptool.py [online]. Github [cit. 2021-02-22]. Dostupné z: https://github.com/espressif/esptool/blob/master/esptool.py
24. Arduino Core for the ESP32[online]. Github [cit. 2021-02-24]. Dostupné z: https://github.com/espressif/arduino-esp32
25. Over-the-air programming [online]. Wikipedia [cit 2021-02-04]. Dostupné z: https://en.wikipedia.org/wiki/Over-the-air\_programming
26. 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
27. Basic OTA [online]. Github [cit. 2018-03-04]. Dostupné z: https://github.com/espressif/arduino-esp32/blob/master/libraries/ArduinoOTA/examples/BasicOTA/BasicOTA.ino
28. Message-Digest algorithm [online]. Wikipedia [cit 2013-03-07]. Dostupné z:   
    https://sk.wikipedia.org/wiki/Message-Digest\_algorithm
29. 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
30. Katalógový list LAN8720 [online]. MicroChip [cit. 2016-01-01]. <https://ww1.microchip.com/downloads/en/DeviceDoc/00002165B.pdf>
31. Katalógový list IP101GRI [online]. IC PLUS Corporation Manufacturer [cit. 2011-01-01]. https://datasheetspdf.com/pdf-file/1469951/ICPlus/IP101GRI/1
32. Reduced media-independent interface [online]. Wikipedia [cit 2021-03-14]. Dostupné z:   
    <https://en.wikipedia.org/wiki/Media-independent_interface#Reduced_media-independent_interface>
33. PHY [online]. Wikipedia [cit 2021-02-25]. Dostupné z:   
    https://en.wikipedia.org/wiki/PHY
34. 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
35. 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
36. 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
37. espsecure.py [online]. Github [cit. 2021-01-19]. Dostupné z: https://github.com/espressif/esptool/blob/master/espsecure.py
38. Python ECDSA [online]. Github [cit. 2021-01-25]. Dostupné z: https://github.com/tlsfuzzer/python-ecdsa
39. 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
40. 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
41. espefuse.py [online]. Github [cit. 2020-12-15]. Dostupné z: https://github.com/espressif/esptool/blob/master/espefuse.py
42. 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
43. Flash Encryption [online]. ESP-IDF Programming Guide [cit. 2021-02-27]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/security/flash-encryption.html
44. Katalógový list Bosch BME 280 [online]. Bosch Sensortec [cit. 2018-11-04]. https://readthedocs.org/projects/bme280/downloads/pdf/latest/
45. BME280 Driver - Bosch Sensortec [online]. Github [cit. 2020-07-06]. Dostupné z: https://github.com/BoschSensortec/BME280\_driver
46. I2C [online]. Wikipedia [cit 2021-02-07]. Dostupné z: https://en.wikipedia.org/wiki/I%C2%B2C
47. 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
48. Solid-state relay [online]. Wikipedia [cit 2021-03-02]. Dostupné z:   
    https://en.wikipedia.org/wiki/Solid-state\_relay
49. Katalógový list OMRON G3MB-202P [online]. Omron Electronics [cit. 2021-02-27]. http://www.omron-russia.com/doc/relay/ssr/g3mb.pdf
50. Termostat [online]. Wikipédia [cit 2019-10-06]. Dostupné z:   
    https://sk.wikipedia.org/wiki/Termostat#Hyster%C3%A9za
51. HTTPD – Dokumentácia v2.4 [online]. Apache server HTTP project [cit 2021-03-13]. Dostupné z: http://httpd.apache.org/docs/2.4/
52. RFC 1867 [online]. Internet Engineering Task Force [cit 1995-11-01]. Dostupné z: https://tools.ietf.org/html/rfc1867
53. 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
54. 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
55. PHP – Dokumentácia [online]. PHP Group [cit 2021-03-01]. Dostupné z: https://www.php.net/docs.php
56. HTTP Basic authentication [online]. Axway Documentation Portal [cit 2020-08-21]. https://docs.axway.com/bundle/APIGateway\_762\_PolicyDevFilterReference\_allOS\_en\_HTML5/page/Content/PolicyDevTopics/authn\_http\_basic.htm
57. Base64 [online]. Wikipédia [cit 2016-11-27]. Dostupné z:   
    https://sk.wikipedia.org/wiki/Base64
58. Autodesk Eagle – Overview [online]. Autodesk [cit 2021-04-10]. Dostupné z: https://www.autodesk.com/products/eagle/overview?plc=F360&term=1-YEAR&support=ADVANCED&quantity=1

Prílohy

Príloha A: CD médium – obsahuje:

* diplomová práca v elektronickej podobe,
* návod pre implementáciu funkcionalít Secure Boot, Flash Encryption, digitálny podpis v prostredí ESP-IDF v elektronickej podobe,
* opis dodatočného hardvéru pre demonštráciu senzorového uzla v elektronickej podobe,
* zdrojové kódy testovaných OTA metód v prostredí Arduino IDE a ESP-IDF v elektronickej podobe,
* hlavný projekt senzorového uzla, zdrojový kód hlavnej aplikácie senzorového uzla, použité kľúče, README v elektronickej podobe,
* zdrojové kódy PHP webaplikácie servera s firmvérom v koreňovom priečinku webaplikácie v elektronickej podobe,
* návod na generovanie certifikátov pre certifikačnú autoritu, webserver kryptografickým nástrojom OpenSSL s konfiguračným súborom webservera ssl.conf v elektronickej podobe,
* návod na implementáciu ULP – nízkopríkonového režimu na platforme ESP32 – KEMT prototypovacia doska so zberom informácii zo senzora Bosch BME680 v elektronickej podobe,
* opis čerpania informácii z literárnych zdrojov v elektronickej podobe.